LECTURENOTES

DESIGN & ANALYSIS OF ALGORITHM

B.Tech,4TH Semester,CSE

Preparedby:

MISS ALINA KUMARI SWAIN

Assistant Professor in Computer Science & Engineering

VikashInstitute ofTechnology, Bargarh

(ApprovedbyAICTE,NewDelhi&AffiliatedtoBPUT,Odisha)

BarahagudaCanalChowk,Bargarh,Odisha-768040

www.vitbargarh.ac.in

DISCLAIMER

- This document does not claim any originality and cannot be used as a substitute for prescribed textbooks.
- The information presented here is merely a collection by Miss ALINA KUMARI SWAIN with the inputs of students for their respective teaching assignments as an additional tool for the teaching- learning process.
- Various sources as mentioned at the reference of the document as well as freely available materials from internet were consulted for preparing this document.
- Further, this document is not intended to be used for commercial purpose and the authors are not accountable forany issues, legal or otherwise, arising out of use of this document.
- The author makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and specifically disclaim any implied warranties of merchantability or fitness for a particular purpose.

****

COURSECONTENT

DESIGN & ANALYSIS OF ALGORITHM

B.Tech,4TH Semester,CSE

- **Notation of algorithm**

{PageNo. 1}

Growth of functions, Recurences: The Master method, The Substitution method, The Iteration method, Asymptotic Notations and Basic Efficiency Classes (Use of Big O, θ, etc.) in analysis of algorithms, Mathematical Analysis of few Non-Recursive and Recursive Algorithms**.**

- **Sorting and searching techniques**

{Page No. 42}

Selection Sort, Bubble Sort, Insertion Sort, Sequential Search, Binary Search, Depth First Search and Breadth First Search, Balanced Search Trees, AVL Trees, Red-Black Trees, Heaps and Heap Sort, Disjoint Set and their Implementation, Divide and Conquer Paradigm of problem solving, Complexity analysis and understanding of Merge Sort, Quick Sort, Binary Search Trees..

- **Greedy techniques**

{PageNo. 68}

Prim's Algorithm, Kruskal's Algorithm, Dijkstra's and Bellman Ford Algorithm, Huffman Trees, Knapsack problem.

Dynamic Programming Paradigm: Floyd-Warshall Algorithm, Optimal Binary Search trees, Matrix Chain Multiplication Problem, Longest Common Subsequence Problem, 0/1 Knapsack Problem, Maximum

Network Flow Problem.

- **String matching Algorithms**

{Page No. 96}

Naive string-matching algorithm, The Rabin-Karp Algorithm, string matching with Finite Automata, Knuth Morris Pratt string matching algorithm.

Backtracking: n-Queen's problem, Hamiltonian Circuit problem, Subset-Sum problem, State Space Search Tree for these problems

- Branch and Bound **{Page No. 112}**

Travelling Salesman Problem and its State Space Search Tree.

Introduction to Computability: Polynomial-time verification, NP-Completeness and Reducibility, NP- Complete problems. Approximation Algorithms: Vertex Cover Problem.

****

REFERENCES

DESIGN & ANALYSIS OF ALGORITHM

B.Tech,4TH Semester,CSE

Books:

[1] Computer Algorithms/C++, Ellis Horowitz, SatrajSahni and Rajasekaran, 2nd Edition, 2014, Universities Press..

[2] Introduction to the Design and Analysis of Algorithms, Anany Levitin: 2nd Edition, 2009. Pearson.

[3] R. S. Salaria, Khanna, "Data Structure & Algorithms", Khanna Book Publishing Co. (P) Ltd.T.H. Cormen, C.E. Leiserson, R.L. Rivest and C. Stein, "Introduction to Algorithms", PHI Publication. 2007.

[4] A.V. Aho, J. E. Hopcroft and J.D. Ullman, "The Design and Analysis of Computer Algorithms", Pearson Education.

DigitalLearningResources:

[5] https://r.search.yahoo.com/_ylt=AwrKFZBzUuZn4gIACSu7HAx.;_ylu=Y29sbwNzZzMEcG9zAzMEdnRpZAMEc2VjA3Ny/RV=2/RE=1744357236/RO=10/RU=https%3a%2f%2fwww.vssut.ac.in%2flecture_notes%2flecture1428551222.pdf/RK=2/RS=uwRLmEQNRCYNpHJ0NMDQGMOO8ys-

UNITI:
Introduction: Algorithm, Psuedo code for expressing algorithms, Performance Analysis-Spacecomplexity,Timecomplexity,AsymptoticNotation-Bigohnotation,Omeganotation, Theta notation and Little oh notation, Probabilistic analysis, Amortized analysis.
Divideandconquer: Generalmethod,applications-Binarysearch,Quicksort,Mergesort, Strassen's matrix multiplication.

INTRODUCTIONTOALGORITHM

HistoryofAlgorithm

- Thewordalgorithmcomesfromthe name of aPersianauthor, AbuJa'farMohammedibnMusa al Khowarizmi (c. 825 A.D.), who wrote a textbook on mathematics.
- He is credited with providing the step-by-step rules for adding, subtracting, multiplying, and dividing ordinary decimal numbers.
- WhenwritteninLatin,thenamebecameAlgorismus,fromwhichalgorithm isbutasmallstep
- This word has taken on a special significance in computer science, where "algorithm" has cometorefertoamethodthatcanbeusedbyacomputerforthesolutionofaproblem
- Between400and300B.C.,thegreatGreekmathematicianEuclidinventedanalgorithm
- Findingthegreatestcommondivisor (gcd)oftwopositiveintegers.
- ThegcdofX and Yisthelargestinteger that exactlydivides both Xand Y.
- Eg.,thegcdof80and32is 16.
- The Euclidian algorithm, as it is called, is considered to be the first non-trivial algorithmever devised.

WhatisanAlgorithm?

Algorithmisasetofsteps to complete a task.

For example,

Task:tomakeacupoftea.
Algorithm:
· addwaterandmilktothekettle,
· boilit, add tea leaves,
· Addsugar, andthenserveitincup.

''a set of steps to accomplish or complete a task that is described precisely enough that a computer can run it''.

Describedprecisely:verydifficultforamachinetoknowhowmuchwater,milktobe added etc. in the above tea making algorithm.

Thesealgorithmsrunoncomputersorcomputationaldevices..Forexample,GPSinour smartphones, Google hangouts.

GPSusesshortestpathalgorithm..**Onlineshopping**usescryptographywhichusesRSAalgorithm.

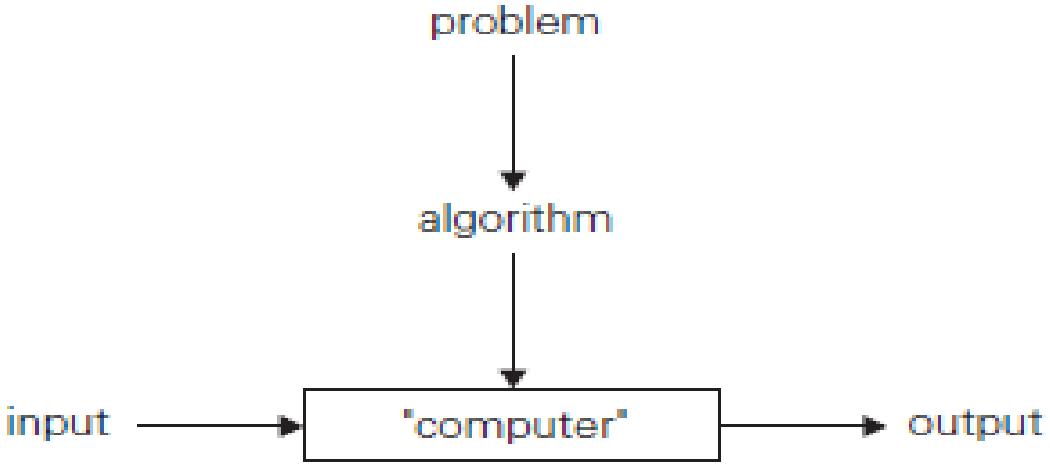
- AlgorithmDefinition1:

- Analgorithmisafinitesetofinstructionsthat,iffollowed,accomplishesaparticulartask. In addition, all algorithms must satisfy the following criteria:
  - Input.Zeroormorequantitiesareexternallysupplied.
  - Output.At least one quantityisproduced.
  - Definiteness.Eachinstructionisclearandunambiguous.
  - Finiteness.Thealgorithmterminatesafterafinitenumberofsteps.
  - Effectiveness.Everyinstructionmustbeverybasicenoughandmustbefeasible.

- AlgorithmDefinition2:

- Analgorithmisasequenceofunambiguousinstructionsfor solving aproblem,i.e.,for obtaining a required output for any legitimate input in a finite amount of time.

- Algorithmsthataredefiniteandeffectivearealsocalled*computationalprocedures*.
- Aprogramistheexpressionofanalgorithminaprogramming language



- **AlgorithmsforProblemSolving**

Themain steps forProblemSolvingare:

1. Problem definition
2. Algorithmdesign/Algorithmspecification
3. Algorithmanalysis
4. Implementation
5. Testing
6. [Maintenance]

- Step1.ProblemDefinition

What is the task to beaccomplished?
Ex:Calculatetheaverageofthegradesfora given student

- Step2.AlgorithmDesign / Specifications:

Describe:innatural language/pseudo-code/diagrams/ etc

- Step3.Algorithm analysis

Spacecomplexity-How much spaceisrequired
Timecomplexity-HowmuchtimedoesittaketorunthealgorithmComputer Algorithm
Analgorithmisaprocedure(afinitesetofwell-definedinstructions)foraccomplishing some tasks which,given an initial stateterminate in a defined end-state
The computational complexity and efficient implementation of the algorithm are importantin computing, and this depends on suitable data structures.

- Steps4,5,6:Implementation,Testing,Maintainance
  - Implementation:

DecideontheprogramminglanguagetouseC,C++,Lisp,Java,Perl,Prolog,assembly,etc. , etc.
Writeclean,welldocumentedcode

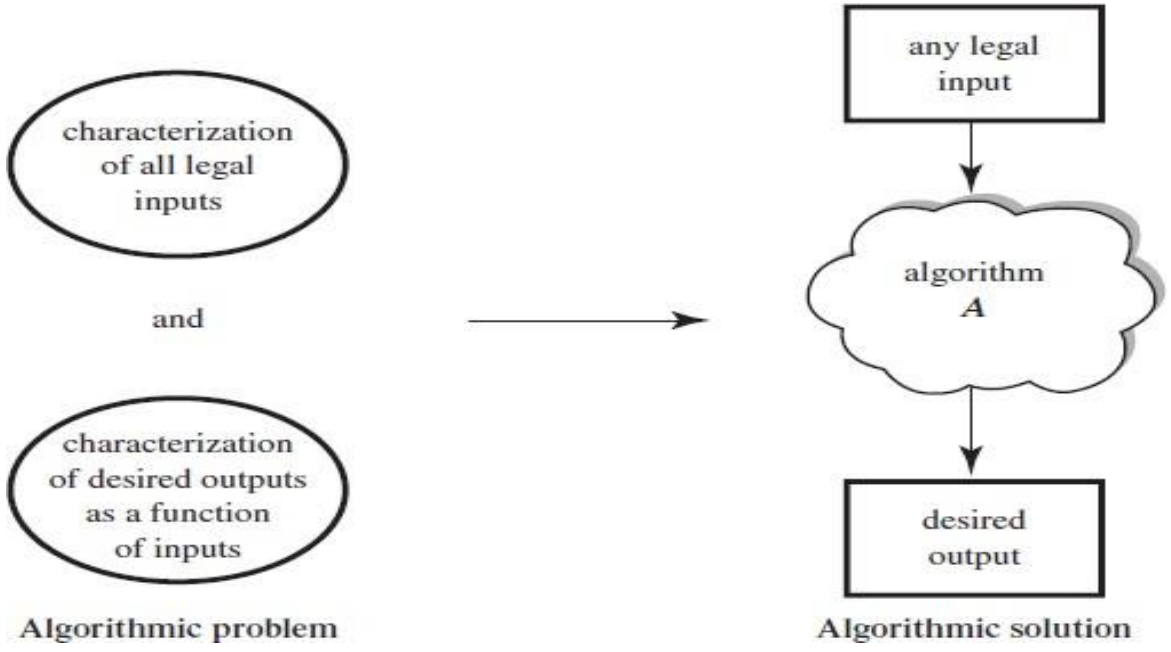
  - Test,test,test

Integratefeedbackfromusers,fixbugs,ensurecompatibilityacrossdifferentversions

  - Maintenance.

ReleaseUpdates,fix bugs

Keeping illegal inputs separate is the responsibility of the algorithmic problem, while treatingspecialclassesofunusualorundesirableinputsistheresponsibilityofthealgorithmitself.



- **4Distinct areasofstudyofalgorithms:**

- *Howtodevisealgorithms*.→Techniques–Divide&Conquer,BranchandBound, Dynamic Programming
- *Howto validate algorithms*.
- CheckforAlgorithmthatitcomputesthecorrectanswerforallpossiblelegalinputs.➔ algorithmvalidation.→FirstPhase
- Secondphase→AlgorithmtoProgram➔ProgramProvingorProgramVerification →Solutionbestatedintwoforms:
- First Form: Program which is annotated by a set of assertions about the input and output variables of the program ➔predicate calculus
- Secondform:iscalledaspecification
- 4Distinct areas of studyofalgorithms(..Contd)
- *How to analyze algorithms*.
- AnalysisofAlgorithmsorperformanceanalysisrefertothetaskofdetermininghow much computing time & storage an algorithm requires
- *Howto test a program*→2 phases →
- Debugging-Debuggingistheprocessofexecutingprogramsonsampledatasetsto determine whether faulty results occur and, if so, to correct them.
- Profiling or performance measurement is the process of executing a correct program ondata sets and measuring the time and space it takes to compute the results

PSEUDOCODE:

- AlgorithmcanberepresentedinTextmodeandGraphicmode
- GraphicalrepresentationiscalledFlowchart
- TextmodemostoftenrepresentedinclosetoanyHighlevellanguagesuchasC, Pascal→Pseudocode

- **Pseudocode:High-leveldescriptionofanalgorithm.**
- ➔MorestructuredthanplainEnglish.
- ➔Lessdetailedthanaprogram.
- ➔Preferrednotationfordescribingalgorithms.
- ➔Hidesprogramdesignissues.

- **ExampleofPseudocode**:

- Tofindthemaxelementofanarray

Algorithm *arrayMax*(*A*, *n*)
Input array *A* of *n* integers
Outputmaximumelementof*Acur*
rentMax←*A*[0]
for*i*← 1 to *n*– 1 do
if*A*[*i*]>*currentMax*then
currentMax←*A*[*i*]

return*currentMax*

- Control flow
- if … then… [else …]
- while … do …
- repeat…until…
- for… do …
- Indentationreplacesbraces
- Methoddeclaration
- Algorithm*method*(*arg*[,*arg*…])
- Input…
- Output …
- Methodcall
- *var.method*(*arg*[,*arg*…])
- Returnvalue
- return*expression*
- Expressions
- Assignment(equivalentto=)
- Equalitytesting(equivalent to==)
- n^2Superscriptsandothermathematicalformattingallowed

PERFORMANCEANALYSIS:

- WhataretheCriteriaforjudgingalgorithmsthathaveamoredirectrelationshiptoperformance?
- computingtimeandstoragerequirements.

- **Performanceevaluation**canbelooselydividedintotwomajorphases:
- aprioriestimatesand
- aposterioritesting.
- ➔referas*performanceanalysis*and*performancemeasurement*respectively

- Thespacecomplexityofanalgorithmistheamountofmemoryitneedstoruntocompletion.
- Thetimecomplexityofanalgorithmistheamountofcomputertimeitneedstorunto completion.

Space Complexity:

- Space ComplexityExample:
- Algorithmabc(a,b,c)

```
{
    returna+b++*c+(a+b-c)/(a+b) +4.0;
}
```

→ TheSpaceneededbyeachofthesealgorithmsisseentobethesumofthefollowing component.

1. Afixedpartthatisindependentofthecharacteristics(eg:number,size)oftheinputsand outputs.
The part typically includes the instruction space (ie. Space for the code), space for simple variableandfixed-sizecomponentvariables(alsocalledaggregate)spaceforconstants,and so on.

2. A variable part that consists of the space needed by component variables whose size is dependentontheparticularprobleminstancebeingsolved,thespaceneededbyreferenced variables (to the extent that is depends on instance characteristics), and the recursion stack space.

Thespacerequirements(p)ofanyalgorithmpmaythereforebewrittenas, S(P) =
c+ Sp(Instance characteristics)
Where'c'isaconstant.

Example2:
Algorithmsum(a,n)
{
s=0.0;
forI=1tondo s=
s+a[I]; return
s;
}
- Theprobleminstancesforthisalgorithmarecharacterizedbyn,thenumberof elements to be summed. The space needed d by 'n' is one word, since it is of type integer.
- Thespaceneededby'a'isthespaceneededbyvariablesoftyepearrayof floating point numbers.
- Thisisatleast'n'words,since'a'mustbelargeenoughtoholdthe'n' elements to be summed.
- So,weobtainSsum(n)>=(n+s)
- [n fora[],oneeachforn,Ia&s]

TimeComplexity:

- ThetimeT(p)takenbyaprogramPisthesumofthecompiletimeandthe run time(execution time)

- The compile time does not depend on the instance characteristics. Also we may assumethatacompiledprogramwillberunseveraltimeswithoutrecompilation.This rum time is denoted by tp(instance characteristics).

- Thenumberofstepsanyproblemstatementisassigneddependsonthekindofstateme nt.

- Forexample,commentsà0steps.
Assignment statements is 1 steps.

[Whichdoesnotinvolveanycalls toother algorithms]
Interactivestatementsuchasfor, while &repeat-untilàControlpartofthestatement.

We introduce a variable, count into the program statement to increment count with initialvalue0.Statementtoincrementcountbytheappropriateamountareintroduced into the program.

Thisisdonesothateachtimeastatementintheoriginalprogramisexecutes count is incremented by the step count of that statement.

Algorithm:

```
Algorithmsum(a,n)
{
s=0.0;
count=count+1; for
I=1 to n do
{
count=count+1;
s=s+a[I];
count=count+1;
}
count=count+1;
count=count+1;
return s;
}
```

→ Ifthecountiszerotostartwith,thenitwillbe2n+3ontermination.Soeach invocation of sum execute a total of 2n+3 steps.

2.Thesecondmethodtodeterminethestepcountofanalgorithmistobuilda table in which we list the total number ofsteps contributes byeach statement.

→Firstdeterminethenumberofstepsperexecution(s/e)ofthestatementandthe total number of times (ie., frequency) each statement is executed.

→Bycombiningthesetwoquantities,thetotalcontributionofallstatements,the step count for the entire algorithm is obtained.

Statement	*Stepsper execution*	*Frequency*	*Total*
1. AlgorithmSum(a,n) 2. { 3. S=0.0; 4. forI=1todo 5. s=s+a[I]; 6. returns; 7. }	0 0 1 1 1 1 0	- - 1 n+1 n1 -	0 0 1 n+1 n1 0
Total			2n+3

Howtoanalyse an Algorithm?

LetusformanalgorithmforInsertionsort(whichsortasequenceofnumbers).Thepseudo code for the algorithm is give below.

PseudocodeforinsertionAlgorithm:

Identifyeachlineof the pseudocodewithsymbols such asC1, C2 ..

PSeudocodeforInsertionAlgorithm	**Line Identification**
forj=2 to A length	C1
key=A[j]	C2
//InsertA[j] intosortedArrayA[1. .....j-1]	C3
i=j-1	C4
whilei>0 & A[j]>key	C5
A[i+1]=A[i]	C6
i=i-1	C7
A[i+1]=key	C8

LetCibe the cost ofithline.SincecommentlineswillnotincuranycostC3=0.

Cost	**No.OftimesExecuted**
C1	N
C2	n-1
C3=0	n-1
C4	n-1
C5	$\sum_{j=2}^{n-1} t_j$
C6	$\sum_{j=2}^{n} t_j - 1$
C7	$\sum_{j=2}^{n} t_j - 1$
C8	n-1

Runningtimeofthe algorithm is:

$T(n)=C1n+C2(n-1)+0(n-1)+C4(n-1)+C5(\sum_{j=2}^{n-1} t_j)+C6(\sum_{j=2}^{n} t_j-1)+C7(\sum_{j=2}^{n} t_j-1)+C8(n-1)$

Bestcase:

ItoccurswhenArrayissorted. All t_j

values are 1.

T(n)=C1n+C2(n-1)+0(n-1)+C4(n-1)+C5($\sum_{j=2}^{n-1}1$)+C6($\sum_{j=2}^{n} 1-1$)+C7($\sum_{j=2}^{n} 1-1$)+ C8(n-1)

=C1n+C2(n-1) +0(n-1) +C4(n-1)+C5+C8(n-1)

=(C1+C2+C4+C5+C8) n-(C2+C4+C5+C8)

·Which is oftheforman+b.

➔·Linearfunctionofn.

➔So,lineargrowth.

Worstcase:

ItoccurswhenArrayisreversesorted,andtj=j

T(n)=C1n+C2(n-1)+0(n-1)+C4(n-1)+C5($\sum_{j=2}^{n-1}$j)+C6($\sum_{j=2}^{n}$ j−1)+C7($\sum_{j=2}^{n}$ j−1)+ C8(n-1)

=C1n+C2(n-1)+C4(n-1)+C5($\frac{n(n-1)}{2}$−1)+C6($\sum_{j=2}^{n}\frac{n(n-1)}{2}$)+C7($\sum_{j=2}^{n}\frac{n(n-1)}{2}$)+C8(n-1)

whichisoftheforman^2+bn+c

Quadraticfunction.Soinworstcaseinsertionsetgrowsinn2. Why we

concentrate on worst-case running time?

· Theworst-caserunningtimegivesaguaranteedupperboundontherunningtimefor any input.

· Forsomealgorithms,theworstcaseoccursoften.Forexample,whensearching,the worstcaseoftenoccurswhentheitembeingsearchedforisnotpresent,andsearches for absent items may be frequent.

· Whynotanalyzetheaveragecase?Becauseit'softenaboutasbadastheworstcase.

Orderofgrowth:

Itisdescribedbythehighestdegreetermoftheformulaforrunningtime.(Droplower-order terms. Ignore the constant coefficient in the leading term.)

Example:Wefoundoutthatforinsertionsorttheworst-caserunningtimeisoftheform an^2 + bn + c.

Drop lower-order terms. What remains is an2.Ignore constant coefficient. It results in n^2.But wecannotsaythattheworst-caserunningtimeT(n)equalsn^2.RatherIt growsliken^2.Butit doesn't equal n^2.We saythat the runningtime is Θ (n^2) to capturethe notion that the order of growth is n^2.

Weusuallyconsideronealgorithmtobemoreefficientthananotherifitsworst-case running time has a smaller order of growth.

ComplexityofAlgorithms

The complexity of an algorithm M is the function f(n) which gives the running time and/or storage space requirement of the algorithm in terms of the size 'n' of the input data. Mostly, the storage space required by an algorithm is simply a multiple of the data size 'n'.

Complexityshall refer totherunningtimeofthealgorithm.

The function f(n), gives the running time of an algorithm, depends not only on the size 'n' of the input data but also on the particular data. The complexity function f(n) for certain cases are:

1. **BestCase**: Theminimumpossiblevalueoff(n)iscalledthebestcase.

2. **Average Case**: The expected valueoff(n).

3. **Worst Case**:Themaximum value of f(n)foranykeypossibleinput.

ASYMPTOTICNOTATION

➔Formalwaynotationtospeakaboutfunctionsandclassifythem

Thefollowingnotationsarecommonlyusenotationsinperformanceanalysisandusedto characterize the complexity of an algorithm:

1. Big–OH(O),
2. Big–OMEGA(Ω),
3. Big–THETA(Θ)and
4. Little–OH(o)

AsymptoticAnalysisof Algorithms:

Ourapproachisbasedonthe*asymptoticcomplexity*measure.Thismeansthatwedon'ttryto counttheexactnumberofstepsofaprogram,buthowthatnumbergrowswiththesizeofthe input to the program. That gives us a measure that will work for different operating systems, compilers and CPUs. The asymptotic complexity is written using big-O notation.

· Itis awaytodescribethecharacteristics of afunction in thelimit.
· Itdescribestherateofgrowthoffunctions.
· Focusonwhat'simportantbyabstractingawaylow-order termsandconstantfactors.
· Itisawaytocompare"sizes"offunctions: O≈

≤

Ω≈ ≥
Θ≈=
o ≈ <
ω≈ >

Time complexity	Name	Example
O(1)	Constant	Addinganelementtothe frontofalinkedlist
O(logn)	Logarithmic	Findinganelementina sortedarray
O(n)	Linear	Findinganelementinan unsortedarray
O(nlogn)	Linear	LogarithmicSortingnitems by'divide-and-conquer'-Mergesort
$O(n^2)$	Quadratic	Shortestpathbetweentwo nodesinagraph
$O(n^3)$	Cubic	MatrixMultiplication
$O(2^n)$	Exponential	TheTowers of Hanoi problem

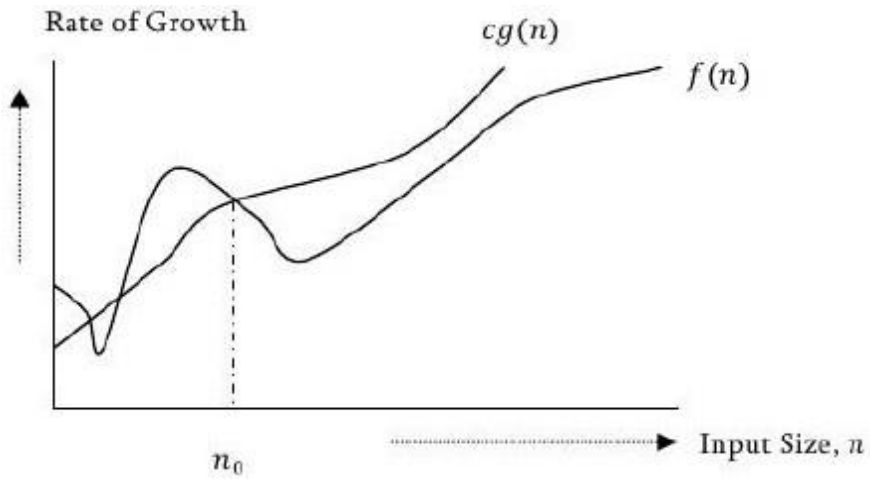
Big'oh':thefunctionf(n)=O(g(n))iffthereexistpositiveconstantscandnosuchthat f(n)<=c*g(n) for all n, n>= no.
Omega:thefunctionf(n)=(g(n))iffthereexistpositiveconstantscandnosuchthat f(n) >= c*g(n) for all n, n >= no.
Theta:thefunctionf(n)=(g(n))iffthereexistpositiveconstantsc1,c2andnosuchthatc1 g(n) <= f(n) <= c2 g(n) for all n, n >= no

Big-ONotation

This notation gives the tight upper bound of the given function. Generally we represent it as f(n) = O(g (11)). That means, at larger values of n, the upper bound off(n) is g(n). For example, if $f(n) = n^4 + 100n^2 + 10n + 50$ is the given algorithm, then n^4 is g(n). That means g(n) gives the maximum rate of growth for f(n) at larger values of n.

O—notationdefinedasO(g(n))={f(n):thereexistpositiveconstantscand$n_o$suchthat 0 <= f(n) <= cg(n) for all n >= n_o}. g(n) is an asymptotic tight upper bound for f(n). Our objective is to give some rate of growth g(n) which is greater than given algorithms rate of growth f(n).

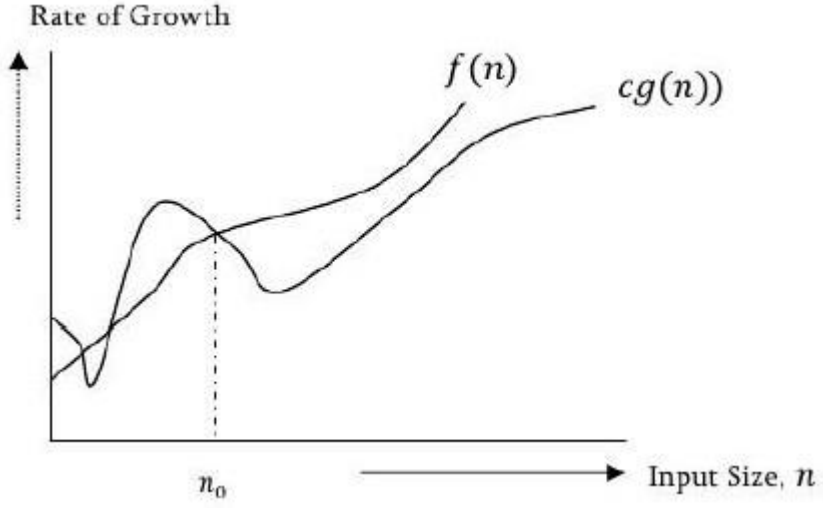
In general, we do not consider lower values of n. That means the rate of growth at lower values of n is not important. In the below figure, n_o is the point from which we consider the rate of growths for a given algorithm. Below n_o the rate of growths may be different.



NoteAnalyzethealgorithmsatlargervaluesofnonlyWhatthismeansis,belownowedo not care for rates of growth.

Omega—Ω notation

Similar to above discussion, this notation gives the tighter lower bound of the given algorithmandwerepresentitasf(n)=Ω(g(n)).Thatmeans,atlargervaluesofn,the tighter lower bound of f(n) is g
Forexample, if f(n) =100n^2+10n +50, g(n)is Ω(n^2).
The.Ω.notationasbedefinedas Ω(g(n))={f(n):thereexistpositiveconstantscand$n_o$such that 0 <= cg(n) <= f(n) forall n >= n_o}. g(n) is an asymptotic lower bound for f(n).Ω (g(n))is thesetoffunctionswith smaller orsameorderofgrowthasf(n).

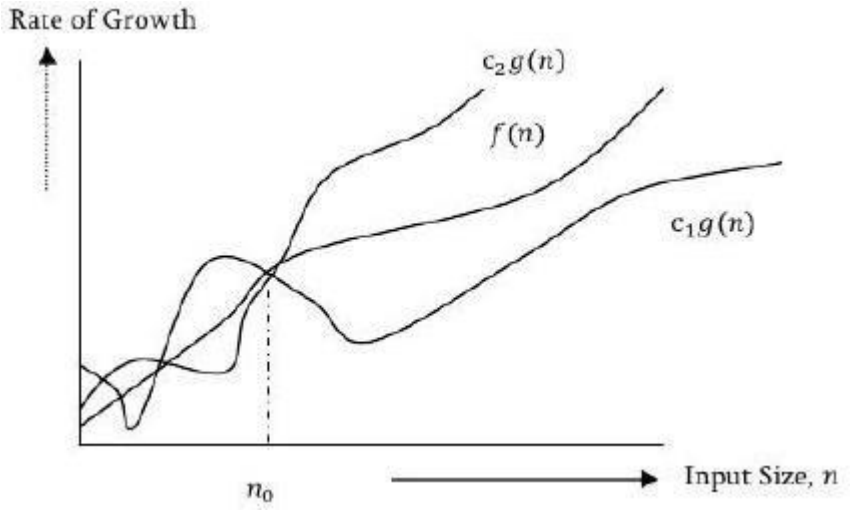


Theta-Θnotation

Thisnotationdecideswhethertheupperandlowerboundsofagivenfunctionaresameor not.Theaveragerunningtimeofalgorithmisalwaysbetweenlowerboundandupperbound.

If the upper bound (O) and lower bound (Ω) gives the same result then Θ notation will also have the same rate of growth. As an example, let us assume that f(n) = 10n + n is the expression.Then,itstightupperboundg(n)isO(n).Therateofgrowthinbestcaseisg(n)= 0(n). In this case, rate of growths in best case and worst are same. As a result, the average case will also be same.

None:Foragivenfunction(algorithm),iftherateofgrowths(bounds)forOandΩarenot same then the rate of growth Θ case may not be same.



Now consider the definition of Θ notation It is defined as Θ (g(n)) = {f(71): there exist positiveconstantsC1,C2andnosuchthatO<=5c_1g(n)<=f(n)<=c_2g(n)foralln>=n_o}. g(n) is an asymptotic tight bound for f(n). Θ (g(n)) is the set of functions with the same order of growth as g(n).

ImportantNotes

For analysis (best case, worst case and average) wetryto give upper bound (O) and lower bound(Ω)andaveragerunningtime(Θ).Fromtheaboveexamples,itshouldalsobeclear that, for a given function (algorithm) getting upper bound (O) and lower bound (Ω) and average running time (Θ) may not be possible always.
Forexample,ifwearediscussingthebestcaseofanalgorithm,thenwetrytogiveupper bound (O) and lower bound (Ω) and average running time (Θ).
In the remaining chapters we generally concentrate on upper bound (O) because knowing lowerbound(Ω)ofanalgorithmisofnopracticalimportanceandweuse9notationifupper bound (O) and lower bound (Ω) are same.

LittleOhNotation

ThelittleOhisdenotedaso.Itisdefinedas:Let,f(n}andg(n}bethenonnegative functions then

$$\lim_{n\to\propto} \frac{f(n)}{g(n)} = 0$$

suchthat f(n}=o(g{n)}i.efof nis little Oh of gofn.

f(n)=o(g(n)) ifandonlyif f'(n)=o(g(n))and f(n) !=Θ {g(n))

PROBABILISTICANALYSIS

Probabilisticanalysisistheuseofprobabilityintheanalysis of problems.

In order to perform a probabilistic analysis, we must use knowledge of, or make assumptions about, the distribution of the inputs. Then we analyze our algorithm, computing an average-case running time, where we take the average over the distribution of the possible inputs.

BasicsofProbabilityTheory

Probability theory has the goal of characterizing the outcomes of natural or conceptual "experiments." Examples of such experiments include tossing a coin ten times, rolling a die three times, playing a lottery, gambling, picking a ball from an urn containing white and red balls, and so on

Each possible outcome of an experiment is called a sample point and the set of all possible outcomes is known as the sample space S. In this text we assume that S is finite (such a sample space is called a discrete sample space). An event E is a subset of the sample space S. If the sample space consists of n sample points, then there are 2^n possible events.

Definition-Probability:Theprobabilityofaneventisdefinedtobe $\frac{|E|}{|S|}$ whereSisthe sample space.

Thenthe***indicatorrandomvariable***I{A}associatedwitheventAisdefinedas I

{A}=1 if A occurs ;
0ifAdoes not occur

The probability of event E is denoted as Prob. [E] The complement of E, denoted E, isdefined to be S - E. If E1 and E2 are two events, the probability of E1 or E2 or both happening is denoted as Prob.[E1 U E2]. The probability of both E1 and E2 occurring at the same time is denoted as Prob.[E1 0 E2]. The corresponding event is E1 0 E2.

Theorem1.5

1. Prob.[E]=1 -Prob.[E].
2. Prob.[E1UE2] =Prob.[E1]+Prob.[E2]-Prob.[E1∩E2]
<=Prob.[E1]+ Prob.[E2]

Expected valueofarandomvariable

The simplest and most useful summary of the distribution of a random variable is the average"ofthevaluesittakeson.The***expectedvalue***(or,synonymously, ***expectation***or***mean***) of a discrete random variable X is

E[X] =$\sum_x x.P_r\{X=x\}$

whichiswelldefinedifthesumisfiniteorconvergesabsolutely.

Consider a game in which you flip two fair coins. You earn $3 for each head but lose $2 for each tail. The expected value of the random variable X representing

yourearningsis

E[X]=6.Pr{2H's}+1.Pr{1H,1T}–4Pr{2T's}

=6(1/4)+1(1/2)-4(1/4)
=1

Anyoneofthesefirsticandidatesisequallylikelytobethebest-qualified sofar.Candidatei has a probability of 1/i of being better qualified than candidates 1 through i -1 and thus a probability of 1/i of being hired.

E[Xi]=1/i

So,
E[X]= E[$\sum_{i=1}^{n} X_i$]

=$\sum_{i=1}^{n} E[X]_i$
=$\sum_{i=1}^{n} 1/i$

AMORTIZEDANALYSIS

Inan***amortizedanalysis***,weaveragethetimerequiredtoperformasequenceof datastructure operations over all the operations performed. With amortized analysis, we can show that the average cost of an operation is small, if we average over a sequence of operations, even though asingle operation within the sequencemightbeexpensive. Amortized analysis differs from average-case analysis in that probability is not involved; an amortized analysis guarantees the *average performance of each operation in the worst case*.

Threemostcommontechniquesusedinamortizedanalysis:

1. **AggregateAnalysis**-inwhichwedetermineanupperboundT(n)onthetotalcost ofasequenceofnoperations.Theaveragecostperoperationisthen T(n)/n.Wetake the average cost as the amortized cost of each operation

2. **Accounting method** – When there is more than one type of operation, each type of operation mayhave a different amortized cost. The accountingmethod overcharges someoperationsearlyinthesequence,storingtheoverchargeas "prepaidcredit"on specific objects in the data structure. Later in the sequence, the credit pays for operations that are charged less than they actually cost.

3. **Potential method** - The potential method maintains the credit as the "potential energy" of the data structure as a whole instead of associating the credit with individualobjectswithinthedatastructure.Thepotentialmethod,whichislikethe accounting methodin that we determine the amortized cost of each operation and may overcharge operations early on to compensate for undercharges later

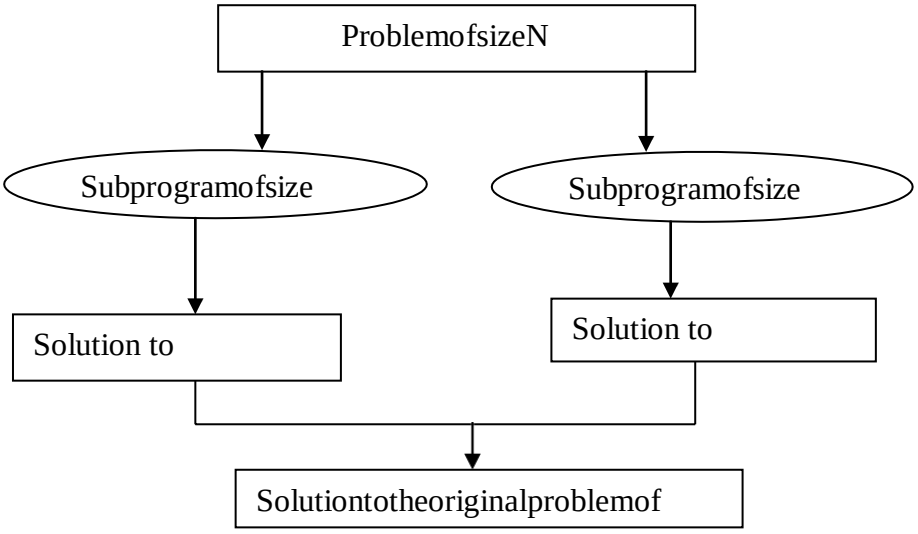
DIVIDEANDCONQUER

GeneralMethod

Individeandconquermethod,agivenproblemis,
i) Dividedinto smaller subproblems.
ii) Thesesubproblemsaresolvedindependently.
iii) CombiningallthesolutionsofsubproblemsintoasolutionofthewhoIe.

If the subproblems are large enough then divide and conquer is reapplied.
Thegeneratedsubproblemsareusuallyofsometypeastheoriginalproblem.

Hencerecurssivealgorithmsareusedindivideandconquerstrategy.



PseudocodeRepresentationofDivideandconquerruleforproblem "P"

```
AlgorithmDAndC(P)
{
ifsmall(P)thenreturnS(P) else{
dividePintosmallerinstances P1,P2,P3…Pk;
applyDAndCtoeachofthesesubprograms;//meansDAndC(P1),DAndC(P2)…..
DAndC(Pk)
returncombine(DAndC(P1),DAndC(P2)…..DAndC(Pk));
}
}
```

//P→Problem
//Heresmall(P)→Booleanvaluefunction.Ifitistrue,thenthefunctionSis
//invoked

TimeComplexityofDAndCalgorithm:

```
T(n)=T(1)if n=1
        aT(n/b)+f(n)ifn>1
```

a,b→contants.
Thisiscalledthe**generaldivideand-conquer recurrence.**

ExampleforGENERALMETHOD:
As an example, let us considerthe problem ofcomputingthe sum of n numbersa_0, ... a_{n-1}.
Ifn>1,wecandividetheproblemintotwoinstancesofthesameproblem. Theyaresumof the first | n/2|numbers
Computethesumofthe1st[n/2]numbers,andthencomputethesumofanothern/2numbers. Combine the answers of two n/2 numbers sum.
i.e.,
$a_0+. .. +a_{n-1}=(a_0+....+a_{n/2})+(a_{n/2}+.....+a_{n-1})$
Assumingthatsizenisapowerofb,tosimplifyouranalysis,wegetthefollowing recurrence for the running time T(n).
T(n)=aT(n/b)+f(n)

Thisiscalledthegeneral**divideand-conquer recurrence**.
f(n)→isafunctionthataccountsforthetimespentondividingtheproblemintosmallerones and on combining their solutions. (For the summation example, a = b = 2 and f (n) = 1.

AdvantagesofDAndC:
ThetimespentonexecutingtheproblemusingDAndCissmallerthanothermethod. This technique is ideally suited for parallel computation.
Thisapproachprovidesanefficientalgorithmincomputer science.

<u>MasterTheoremforDivideandConquer</u>
In all efficient divide and conquer algorithms we will divide the problem into subproblems, each ofwhich is some partoftheoriginal problem, and thenperformsomeadditional work to compute the final answer. As an example, if we consider merge sort [for details, refer Sorting chapter], it operates on two problems, each of which is half the size of the original, and then uses O(n) additional work for merging. This gives the running time equation:

$T(n)=2T(\frac{n}{2})+O(n)$

The following theorem can be used to determine the running time of divide and conquer algorithms.Foragivenprogramoralgorithm,firstwetrytofindtherecurrencerelationfor the problem. If the recurrence is of below form then we directly give the answer without fully solving it.

IftherecurrenceisoftheformT(n) = $aT(\frac{n}{b})+\Theta(nk\log^p n)$,where a>= 1,b> 1, k>=O andp is a realnumber, then wecandirectlygivetheanseweras:

1) Ifa>b^k,then T(n)=Θ ($n^{\log_b a}$)

2) Ifa=b^k
a. Ifp>-1,thenT(n)=Θ($n^{\log_b a}\log^{p+1} n$)
b. Ifp=-1,thenT(n)=Θ($n^{\log_b a}\log\log n$)
c. Ifp<-1,thenT(n) = Θ($n^{\log_b a}$s)

3) Ifa <b^k
a. Ifp>=0, then T(n)=Θ($n^k\log^n p$)
b. Ifp<0, thenT(n)=0(n^k)

ApplicationsofDivideandconquerruleor algorithm:

- Binarysearch,
- Quicksort,
- Mergesort,
- Strassen'smatrixmultiplication.

BinarysearchorHalf-intervalsearchalgorithm:

1. Thisalgorithmfindsthepositionofaspecifiedinputvalue(thesearch"key")within an array sorted by key value.
2. In eachstep,thealgorithmcomparesthesearchkeyvaluewiththekeyvalueofthe middle element of the array.
3. Ifthekeysmatch,thenamatchingelementhasbeenfoundanditsindex,orposition, is returned.
4. Otherwise,ifthesearchkeyislessthanthemiddleelement'skey,thenthealgorithm repeats its action on the sub-array to the **left** of the middle element or, if the search key is greater, then the algorithm repeats on sub array to the **right** of the middle element.
5. Ifthesearchelementislessthantheminimumpositionelementorgreaterthanthe maximum position element then this algorithm returns not found.

Binarysearchalgorithmbyusingrecursivemethodology:

Programforbinarysearch(recursive)	Algorithmforbinarysearch(recursive)
intbinary_search(intA[],intkey,intimin,intimax)	**Algorithmbinary_search(A,key,imin,imax)**

```
{
if(imax<imin)
   returnarrayisempty; if(key<imin
|| K>imax)
   returnelementnotinarraylistelse
   {
    intimid=(imin+imax)/2; if
    (A[imid] > key)
    returnbinary_search(A,key,imin,imid-1); else
    if (A[imid] < key)
    returnbinary_search(A,key,imid+1,imax); else
    returnimid;
   }
}
```

```
{
 if (imax<imin) then
return“arrayisempty”;
 if(key<imin || K>imax) then
return“elementnotinarraylist” else
   {
    imid=(imin+imax)/2;
     if(A[imid]>key)then
returnbinary_search(A,key,imin,imid-1); else
     if (A[imid] < key) then
returnbinary_search(A,key,imid+1,imax); else
     returnimid;
   }
}
```

TimeComplexity:
Datastructure:-Array

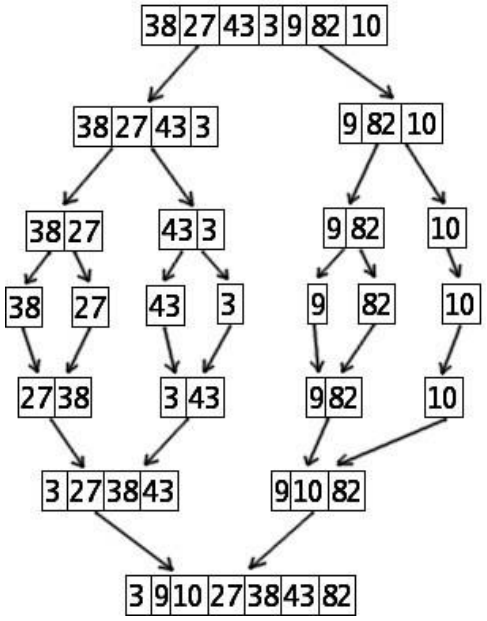
Forsuccessfulsearch	**Unsuccessfulsearch**
Worst case→ O(logn)orθ(logn) Averagecase→O(logn) orθ(logn) Bestcase→ O(1) orθ(1)	θ(logn):-forallcases.

Binarysearchalgorithmbyusingiterativemethodology:

Binarysearchprogrambyusingiterativemethodology:	**Binarysearchalgorithmbyusingiterativemethodology:**
intbinary_search(intA[],intkey,intimin,intimax) **{** **while(imax>=imin)** **{** **intimid=midpoint(imin,imax);** **if(A[imid] == key)** **returnimid;** **elseif(A[imid]<key) imin** **= imid + 1;** **else** **imax=imid-1;** **}** **}**	**Algorithmbinary_search(A,key,imin,imax)** **{** **While<(imax>=imin)>do** **{** **intimid=midpoint(imin,imax); if(A[imid]** **== key)** **returnimid;** **elseif(A[imid]<key) imin** **= imid + 1;** **else** **imax=imid-1;** **}** **}**

MergeSort:

The merge sort splits the list to be sorted into two equal halves, and places them in separate arrays. This sorting method is an example of the DIVIDE-AND-CONQUER paradigm i.e. it breaks the data into two halves and then sorts the two half data sets recursively, and finally merges them to obtain the complete sorted list. The merge sort is a comparison sort and has an algorithmiccomplexityof*O*(*n*log*n*).Elementaryimplementationsofthemergesortmakeuseof twoarrays-oneforeachhalfofthedataset.Thefollowingimagedepictsthecompleteprocedure of merge sort.



AdvantagesofMergeSort:

1. Marginallyfasterthantheheapsortforlargersets
2. MergeSortalwaysdoeslessernumberofcomparisonsthanQuickSort.Worstcasefor merge sort does about 39% less comparisons against quick sort's average case.
3. Mergesortisoftenthebestchoiceforsortingalinkedlistbecausetheslowrandom-accessperformanceofalinkedlistmakessomeotheralgorithms(suchasquicksort) perform poorly, and others (such as heap sort) completely impossible.

ProgramforMerge sort:

```
#include<stdio.h>
#include<conio.h>
int n;
void main(){
inti,low,high,z,y;
int a[10];
voidmergesort(inta[10],intlow,inthigh);
void display(int a[10]);
clrscr();
printf("\n\t\tmergesort\n");
printf("\nenterthelengthofthelist:");
scanf("%d",&n);
printf("\nenterthelistelements");
for(i=0;i<n;i++)
scanf("%d",&a[i]);
low=0;
high=n-1;
mergesort(a,low,high);
display(a);
getch();
}
voidmergesort(int a[10],intlow, int high)
```

```
{
int mid;
voidcombine(inta[10],intlow,intmid,inthigh);
if(low<high)
{
mid=(low+high)/2;
mergesort(a,low,mid);
mergesort(a,mid+1,high);
combine(a,low,mid,high);
}
}
voidcombine(inta[10],intlow,intmid,inthigh){ int
i,j,k;
inttemp[10];
k=low;
i=low;
j=mid+1;
while(i<=mid&&j<=high){
if(a[i]<=a[j])
{
temp[k]=a[i];
i++;
k++;
}
else
{
temp[k]=a[j];
j++;
k++;
}
}
while(i<=mid){
temp[k]=a[i];
i++;
k++;
}

while(j<=high){
temp[k]=a[j];
j++;
k++;
}
for(k=low;k<=high;k++)
a[k]=temp[k];
}
voiddisplay(inta[10]){ int
i;
printf("\n\nthesortedarrayis\n"); for(i=0;i<n;i++)
printf("%d\t",a[i]);}
```

AlgorithmforMergesort:

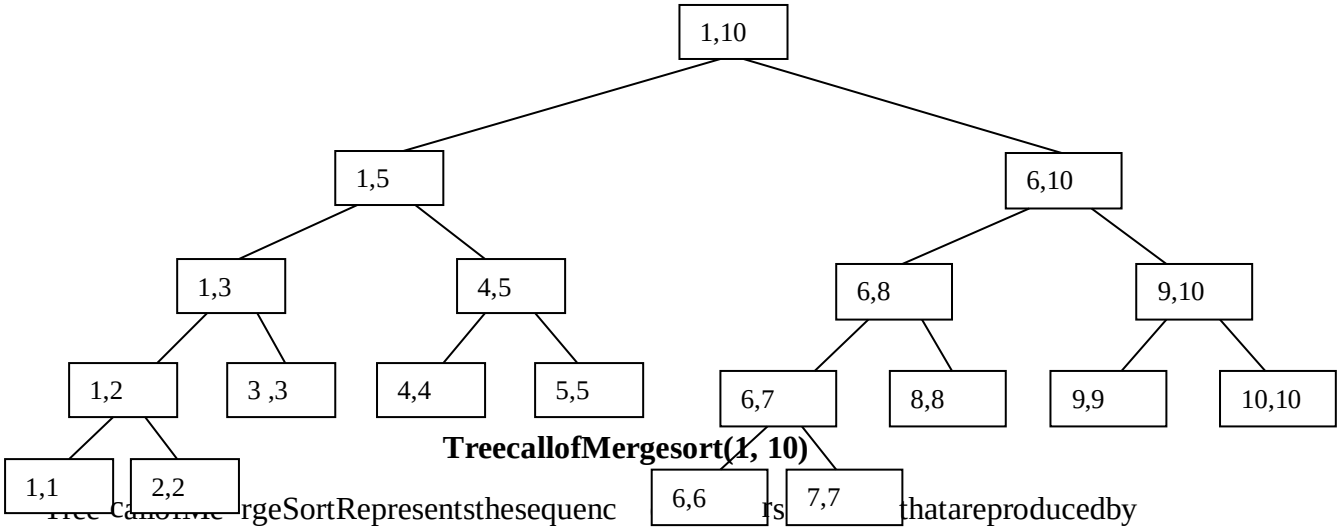
```
Algorithmmergesort(low,high)
{
if(low<high) then
{
mid=(low+high)/2;
mergesort(low,mid);
mergesort(mid+1,high);//Solvethesub-problems
Merge(low,mid,high);     // Combine the solution
}
}
voidMerge(low,mid,high){
k=low;
i=low;
j=mid+1;
while(i<=mid&&j<=high)do{
if(a[i]<=a[j]) then
{
temp[k]=a[i];
i++;
k++;
}
else
{
temp[k]=a[j];
j++;
k++;
}
}
while(i<=mid)do{
temp[k]=a[i];
i++;
k++;
}

while(j<=high)do{
temp[k]=a[j];
j++;
k++;
}
Fork=lowtohighdoa[k]
=temp[k];
}
Fork:=lowtohighdoa[k]=temp[k];
}
```

//DividingProbleminto Sub-problems and this“mid”isforfindingwheretosplittheset.

TreecallofMergesort

Consider a example: (From text book)

A[1:10]={310,285,179,652,351,423,861,254,450,520}



TreecallofMergesort(1, 10)

Tree CallofMergeSortRepresentsthesequenc rs thatareproducedby mergesort.

"Onceobservetheexplainednotesinclass room"

ComputingTimeforMergesort:

Thetimeforthemergingoperationinproportionalton,thencomputingtimeformergesort is described by using recurrence relation.

T(n)=a	**ifn=1;**
2T(n/2)+ cn	**ifn>1**

Herec,a→Constants.

Ifnispowerof2, $n=2^k$

Formrecurrencerelation

T(n)=2T(n/2) + cn

2[2T(n/4)+cn/2]+cn

4T(n/4)+2cn

2^2T(n/4)+2cn

2^3T(n/8)+3cn

2^4T(n/16)+4cn

2^k

T(1)+kcnan+c

n(log n)

ByrepresentingitbyintheformofAsymptoticnotationOis

T(n)=O(nlog n)

QuickSort

Quick Sort isanalgorithmbased on the DIVIDE-AND-CONQUER paradigmthat selects apivot element and reorders the given list in such a way that all elements smaller to it are on one side andthosebiggerthanitareontheother.Thenthesublistsarerecursivelysorteduntilthelistgets completely sorted. The time complexity of this algorithm is O (n log n).

- Auxiliaryspaceusedintheaveragecaseforimplementingrecursivefunctioncallsis O(logn)andhenceprovestobeabitspacecostly,especiallywhenitcomestolarge data sets.
- Itsworstcasehas a timecomplexityof O $(n)^2$ which can proveveryfatalforlarge datasets.Competitivesortingalgorithms

Quicksortprogram

```
#include<stdio.h>
#include<conio.h>
int n,j,i;
voidmain(){
inti,low,high,z,y;
int a[10],kk;
voidquick(inta[10],intlow,inthigh); int
n;
clrscr();
printf("\n\t\tmergesort\n");
printf("\nenterthelengthofthelist:");
scanf("%d",&n);
printf("\nenterthelistelements");
for(i=0;i<n;i++)
scanf("%d",&a[i]);
low=0;
high=n-1;
quick(a,low,high);
printf("\nsortedarrayis:");
for(i=0;i<n;i++)
printf("%d",a[i]);
getch();
}

intpartition(inta[10],intlow,inthigh){ int
i=low,j=high;
int temp;
intmid=(low+high)/2;
int pivot=a[mid];
while(i<=j)
{
while(a[i]<=pivot)
i++;
```

```
while(a[j]>pivot)
j--;
if(i<=j){
        temp=a[i];
        a[i]=a[j];
        a[j]=temp;
        i++;
        j--;
}}
returnj;
}
voidquick(inta[10],intlow,int high)
{
intm=partition(a,low,high);
if(low<m)
quick(a,low,m);
if(m+1<high)
quick(a,m+1,high);
}
```

AlgorithmforQuicksort

```
AlgorithmquickSort(a,low,high){
If(high>low) then{
m=partition(a,low,high);
if(low<m) then quick(a,low,m);
if(m+1<high)thenquick(a,m+1,high);
}}

Algorithmpartition(a,low,high){
i=low,j=high;
 mid=(low+high)/2;
 pivot=a[mid];
while(i<=j)do{ while(a[i]<=pivot)
                  i++;
               while(a[j]>pivot)
                 j--;
                   if(i<=j){temp=a[i];
                   a[i]=a[j];
                   a[j]=temp;
                 i++;
        j--;
}}
returnj;
}
```

Name	TimeComplexity			Space Complexity
	Bestcase	Average Case	Worst Case	
Bubble	O(n)	-	$O(n^2)$	O(n)
Insertion	O(n)	$O(n^2)$	$O(n^2)$	O(n)
Selection	$O(n^2)$	$O(n^2)$	$O(n^2)$	O(n)

Quick	O(logn)	O(nlogn)	$O(n^2)$	O(n+logn)
Merge	O(nlogn)	O(nlogn)	O(nlogn)	O(2n)
Heap	O(nlogn)	O(nlogn)	O(nlogn)	O(n)

ComparisonbetweenMergeandQuickSort:

- BothfollowsDivideandConquerrule.
- Statisticallybothmergesortandquicksorthavethesameaveragecasetimei.e.,O(n log n).
- MergeSortRequiresadditionalmemory.Theprosofmergesortare:itisastablesort, and there is no worst case (means average case and worst case time complexity is same).
- Quicksortisoftenimplementedinplacethussavingtheperformanceandmemoryby not creating extra storage space.
- ButinQuicksort,theperformancefallsonalreadysorted/almostsortedlistifthe pivot is not randomized. Thus why the worst case time is $O(n^2)$.

RandomizedSortingAlgorithm:(Randomquicksort)

- Whilesortingthearraya[p:q]insteadofpickinga[m],pickarandomelement(from among a[p], a[p+1], a[p+2]---a[q]) as the partition elements.
- TheresultantrandomizedalgorithmworksonanyinputandrunsinanexpectedO(n log n) times.

Algorithmfor Random Quicksort

AlgorithmRquickSort(a,p,q){ If(high>low)
then{
If((q-p)>5)then
Interchange(a,Random()mod(q-p+1)+p,p);
m=partition(a,p, q+1);
quick(a,p,m-1);
quick(a,m+1,q);
}}

Strassen'sMatrixMultiplication:

LetA and Bbetwon×nMatrices.Theproductmatrix C=ABisalsoan×n matrix whosei,j^{th} element is formed bytaking elements in the i^{th} row of A and j^{th}column of B and multiplying them to get

C(i,j)=$\sum_{1\le k\le n} A(i,k)B(k,j)$

Here1≤i&j ≤nmeansiand jareinbetween1and n.

TocomputeC(i,j) usingthisformula, weneednmultiplications.

Thedivideandconquerstrategysuggestsanotherwaytocomputetheproductoftwo n×nmatrices. ForSimplicityassumenisapowerof2thatisn=2^k Here

k→ any nonnegative integer.

Ifnisnotpoweroftwothenenoughrowsandcolumnsofzeroscanbeaddedtobothand B, so that resulting dimensions are a power of two.

LetAandBbetwon×nMatrices.ImaginethatA&Bareeachpartitionedintofoursquare sub matrices. Each sub matrix having dimensions n/2×n/2.
TheproductofABcanbecomputedbyusingpreviousformula. If AB isproduct of 2×2matrices then

$$\begin{pmatrix} A11 & A12 \\ A21 & A22 \end{pmatrix}\begin{pmatrix} B11 & B12 \\ B21 & B22 \end{pmatrix}=\begin{pmatrix} C11 & C12 \\ C21 & C22 \end{pmatrix}$$

$C_{11}=A_{11}B_{11}+A_{12}B_{21}$
$C_{12}=A_{11}B_{12}+A_{12}B_{22}$
$C_{21}=A_{21}B_{11}+A_{22}B_{21}$
$C_{22}=A_{21}B_{12}+A_{22}B_{22}$

Here8multiplicationsand 4additionsareperformed.
NotethatMatrixMultiplicationaremoreExpensivethanmatrixadditionand subtraction.

T(n)=b ifn≤2;
8T(n/2)+ cn^2 ifn>2

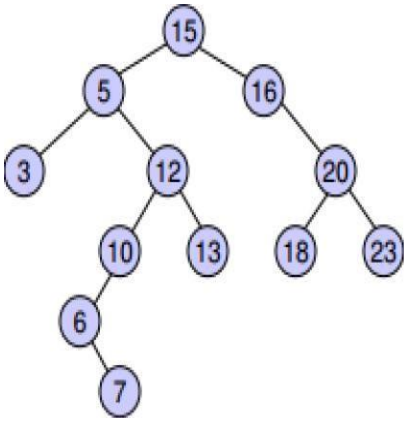
VolkerstrassenhasdiscoveredawaytocomputetheC$_{i,j}$ofaboveusing7 multiplications and 18 additions or subtractions.
Forthisfirstcompute7n/2×n/2matricesP,Q,R,S,T,U&V
P=(A11+A22)(B11+B22)
Q=(A21+A22)B11
R=A11(B12-B22)
S=A22(B21-B11)
T=(A11+A12)B22
U=(A21-A11)(B11+B12)
V=(A12-A22)(B21+B22)

C11=P+S-T+V
C12=R+T
C21=Q+S
C22=P+R-Q+U

T(n)=b ifn≤2;
7T(n/2)+ cn^2 ifn>2

UNITII:

Searching and Traversal Techniques: Efficient non - recursive binary tree traversal algorithm, Disjoint set operations, union and find algorithms, Spanning trees, Graph traversals - Breadth first search and Depth first search, AND / OR graphs, game trees, ConnectedComponents,Bi-connectedcomponents.DisjointSets-disjointsetoperations, unionandfindalgorithms,spanningtrees,connected components and biconnected components.



Efficientnonrecursivetreetraversalalgorithms

in-order:(left,root,right)
3,5,6,7,10,12,13
15,16,18,20,23

pre-order:(root,left,right) 15,
5, 3, 12, 10, 6, 7,
13,16,20,18,23

post-order:(left,right,root)
3, 7, 6, 10, 13, 12, 5,
18,23,20,16,65

NonrecursiveInordertraversal algorithm

1. Startfiomtheroot.let'sitiscurrent.
2. IfcurrentisnotNULL. pushthenodeontostack.
3. Movetoleftchildofcurrentandgotostep2.
4. IfcurrentisNULL,andstackisnotempty,popnodefromthestack.
5. Printthenodevalueandchangecurrenttorightchildofcurrent.
6. Goto step 2.

So we go on traversing all left node. as we visit the node. we will put that node into stack.rememberneedto visit parent afterthechildandasWewillencounterparentfirstwhen start from root.it'scasefor LIFO:)and hencethe stack). Oncewereach NULLnode. wewill take the node at the top of the stack. last node which we visited. Print it.
Check if there is right child to that node. If yes. move right child to stack and again start traversing left child node and put them on to stack. Once we have traversed all node. ourstack will be empty.

Nonrecursivepostordertraversalalgorithm

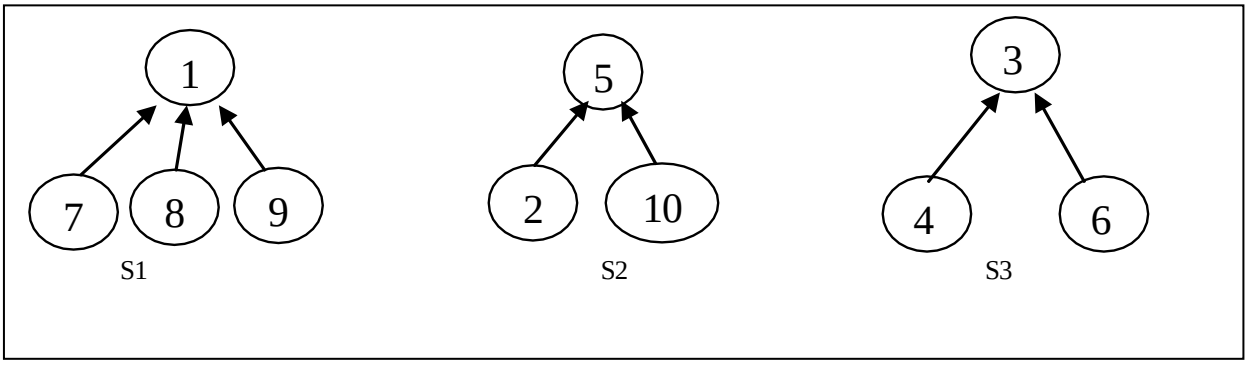
Leftnode.rightnode and lastparentnode.
Createanemptystack
DoFollowingwhilerootis not NULL
a) Pushroot'srightchildandthenroottostack.

b) Setrootasroot'sleftchild.
Popanitemfromstackandsetitasroot.
a)Ifthepoppeditemhasarightchildandtheright child isattopofstack,thenremovetherightchildfromstack, push the root back and set root as root's right child.
Ia)Elseprintroot'sdataandsetrootasNULL.
Repeatsteps2.1and2.2 whilestackisnotempty.

DisjointSets: If $S_i$and S_j,$i \neq j$aretwosets,thenthereisnoelementthatisinboth$S_i$andS_j.
Forexample:n=10elementscan bepartitionedintothreedisjointsets,

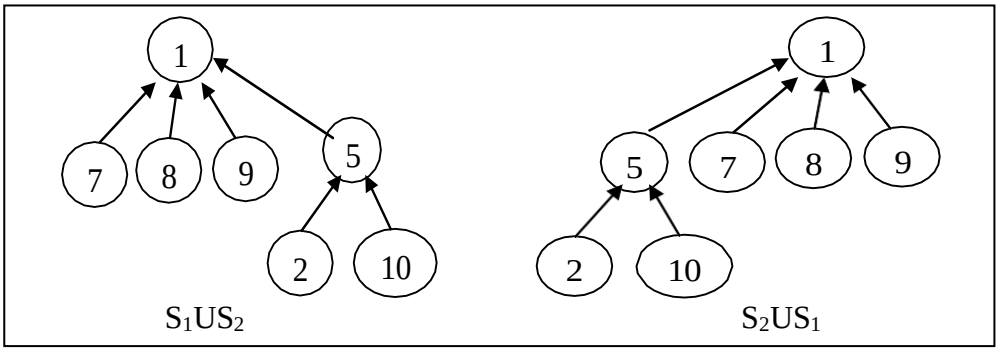
S_1={1,7,8,9}
S_2={2,5,10}
S_3={3,4,6}

Treerepresentationofsets:



DisjointsetOperations:
- Disjoint set Union
- Find(i)

Disjoint set Union: MeansCombinationoftwodisjointsetselements.Formabove example S_1 U S_2 ={1,7,8,9,5,2,10 }
ForS_1US_2treerepresentation, simplymakeoneofthetreeisasubtree of the other.

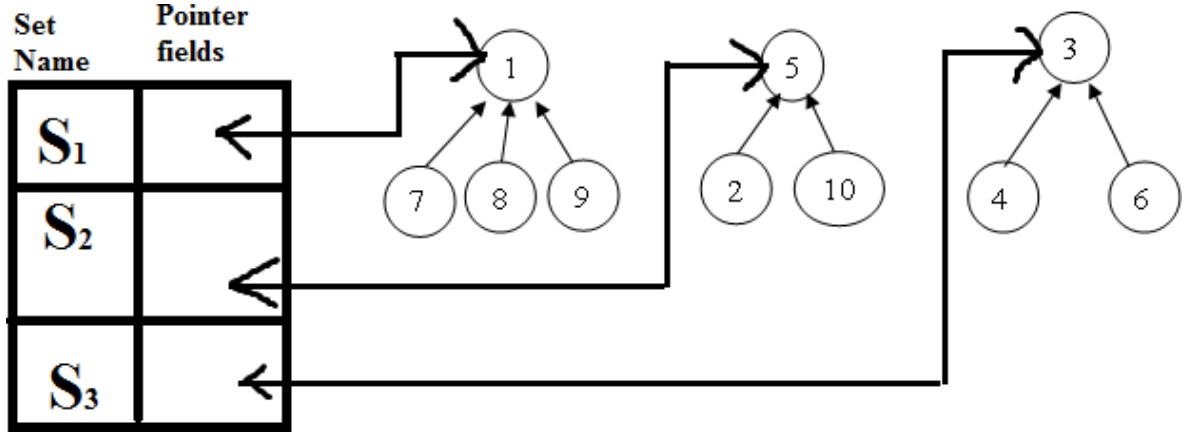


Find: Givenelementi,findthesetcontainingi.
Formabove example:
Find(4)→S_3

Find(1)→S_1
Find(10)→S_2

Datarepresentationofsets:

Tress can be accomplished easily if, with each set name, we keep a pointer to the root of the tree representing that set.



For presenting the union and find algorithms, we ignore the set names and identify sets justby the roots of the trees representing them.
For example: if we determine that element 'i' is in a tree with root 'j' has a pointer to entry 'k' in the set name table, then the set name is just **name[k]**

Forunite(**addingorcombine**)to aparticularsetweuseFindPointerfunction.
Example:IfyouwishtounitetoS$_i$andS$_j$thenwewishtounitethetreewithrootsFindPointer (S$_i$) and FindPointer (S$_j$)
FindPointer→isafunctionthattakesasetnameanddeterminestherootofthetreethat represents it.
Fordeterminingoperations:
Find(i)→ 1Stdeterminetherootofthetreeandfinditspointertoentryinsetnametable. Union(i, j)→Means union of two trees whose roots are i and j.

Ifsetcontainsnumbers1throughn,werepresentstreenode
P[1:n].
n→Maximumnumberofelements. Each

node represent in array

i	1	2	3	4	5	6	7	8	9	10
P	-1	5	-1	3	-1	3	1	1	1	5

Find(i)byfollowingthe indices, starting at i until wereachanodewithparentvalue -1.
Example: Find(6) start at 6 and then moves to 6's parent. Since P[3] is negative, we reached the root.

AlgorithmforfindingUnion(i, j):	Algorithmforfind(i)

AlgorithmSimpleunion(i,j)
{
P[i]:=j;//Accomplishestheunion
}
	AlgorithmSimpleFind(i)
	{
	While(P[i]≥0)doi:=P[i]; return
	i;
	}

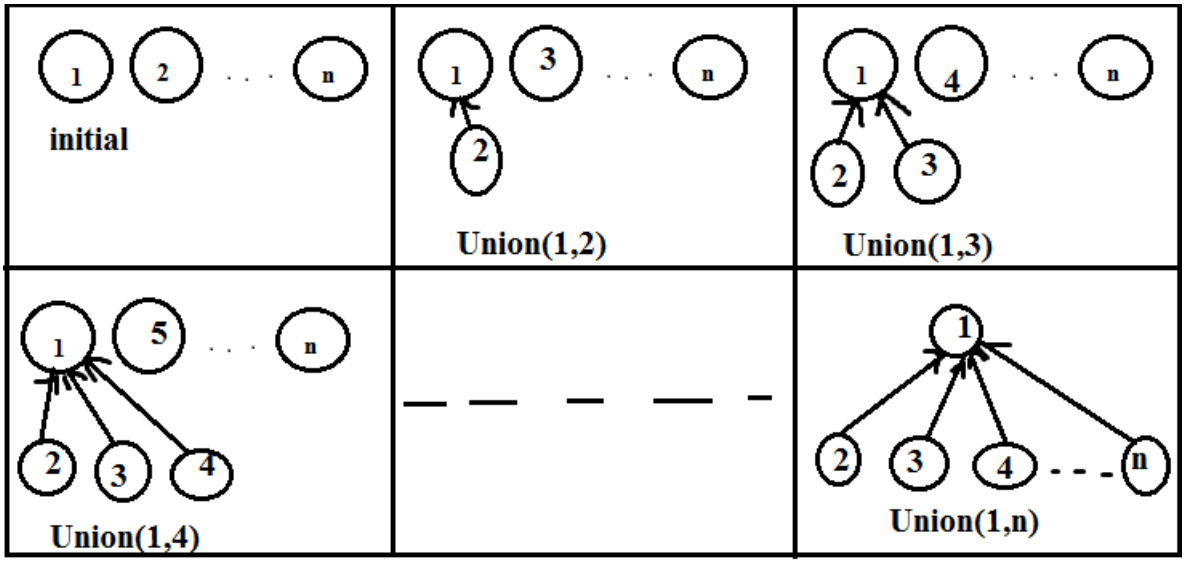
Ifnnumbersofrootsaretherethentheabovealgorithmsarenotusefulforunionandfind. For union of n trees→ Union(1,2), Union(2,3), Union(3,4),…..Union(n-1,n).
For Findiinntrees→ Find(1),Find(2),….Find(n).

Timetakenfortheunion(simpleunion) is → O(1)(constant).
Forthen-1unions→ O(n).

Timetakenforthe find foranelementatleveliofatreeis→ O(i).
Fornfinds → $O(n^2)$.

Toimprovetheperformanceofourunionandfindalgorithmsbyavoidingthecreationof degenerate trees. For this we use a weighting rule for union(i, j)

WeightingruleforUnion(i, j):
If the number of nodes in the tree with root 'i' is less than the tree with root 'j', then make 'j' the parent of 'i'; otherwise make 'i' the parent of 'j'.



Tree obtained using the weighting rule

AlgorithmforweightedUnion(i, j)

```
AlgorithmWeightedUnion(i,j)
//Unionsetswithrootsiandj,i≠j
//Theweightingrule, p[i]= -count[i] andp[j]= -count[j].
{
temp := p[i]+p[j];
if(p[i]>p[j]) then
{//ihasfewernodes. P[i]:=j;
P[j]:=temp;
}
else
{//jhasfeweorequalnodes. P[j] :=
i;
P[i] := temp;
}
}
```

For implementing the weighting rule, we need to know how many nodes there arein every tree.
Forthiswemaintainacountfieldintherootofeverytree. i→ root node
count[i]→numberofnodesinthetree.
Time required for this above algorithm isO(1) + time for remaining unchanged is determined by using **Lemma.**

Lemma:- Let T be a tree with **m** nodes created as a result of a sequence of unions each performed usingWeightedUnion. The height of T is no greater than $\lfloor \log_2 m \rfloor + 1$.

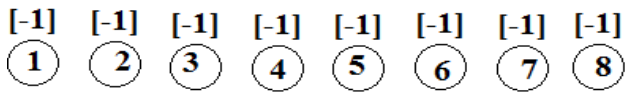
Collapsing rule: If 'j' is a node on the path from 'i' to its root and p[i]≠root[i], then set p[j] to root[i].

AlgorithmforCollapsingfind.

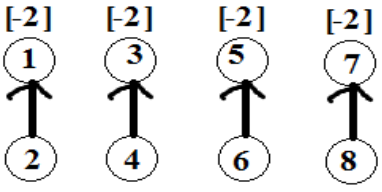
AlgorithmCollapsingFind(i)
//Findtherootofthetreecontainingelementi.
//collapsingruletocollapseallnodesformitotheroot.
{
r;=i;
while(p[r]>0) do r := p[r]; //Find the root.
While(i≠r)do//Collapsenodesfromitorootr.
{
s:=p[i];
p[i]:=r;
i:=s;
}
returnr;
}

Collapsingfindalgorithmisusedtoperformfindoperationonthetreecreatedby WeightedUnion.

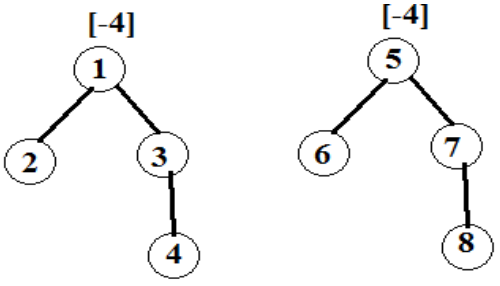
Forexample:TreecreatedbyusingWeightedUnion



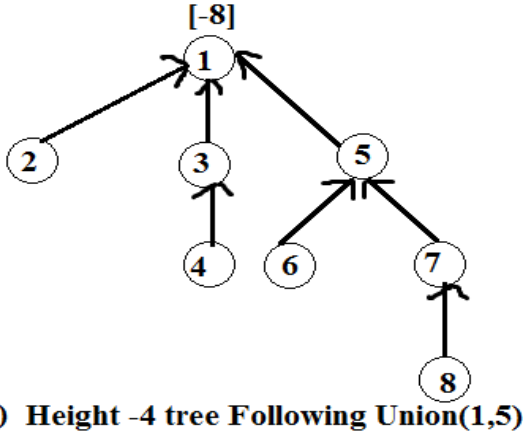
(a)initial height -1 tree



(b)Height -2 trees follwoing Union(1,2),(3,4),(5,6),(7,8)



(c) Height -3 trees following Union (1,3) and (5,7)



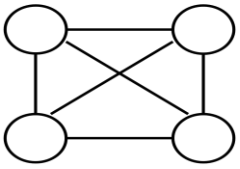
(d) Height -4 tree Following Union(1,5)

Nowprocessthefollowingeightfinds:Find(8), Find(8),…..... Find(8)
If SimpleFind is used, each Find(8) requires going up three parent link fields for a total of 24 moves to process all eight finds.

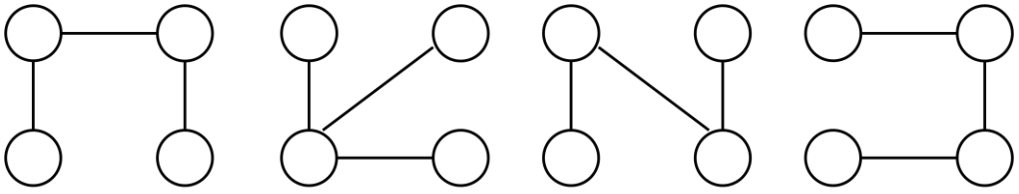
WhenCollapsingFindisuisedthefirstFind(8)requiresgoingupthreelinksandthen resetting two links. Total 13 movies requies for process all eight finds.

Spanning Tree:-

Let G=(V<E) be an undirected connected graph. A sub graph t=(V,E^1)of G is aspanningtree of G iff t is a tree.



A connected, undirected graph



Four of the spanning trees of the graph

SpanningTreeshavemanyapplications.
Example:-

Itcanbeusedtoobtainanindependentsetof circuit equationsforanelectricnetwork.
Any connected graph with n vertices must have at least n-1 edges and all connected graphs with n-1 edges are trees. If nodes of G represent cities and the edges represent possible communication links connecting two cities, then the minimum number of links needed to connect the n cities is n-1.
There are two basic algorithms for finding minimum-cost spanning trees, and both are greedy algorithms

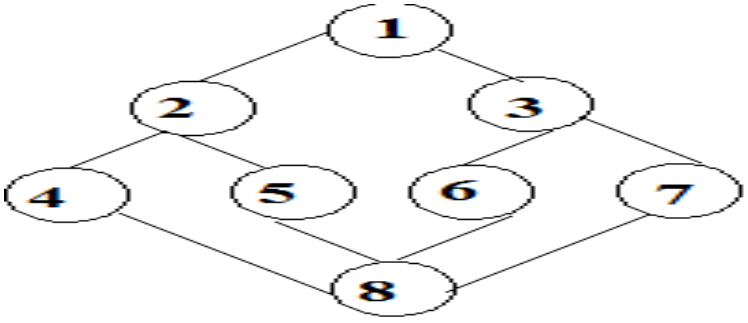
→Prim'sAlgorithm
→Kruskal'sAlgorithm

Prim'sAlgorithm:Startwithany*onenode*inthespanningtree,andrepeatedlyaddthe cheapest edge, and the node it leads to, for which the node is not alreadyin the spanning tree.

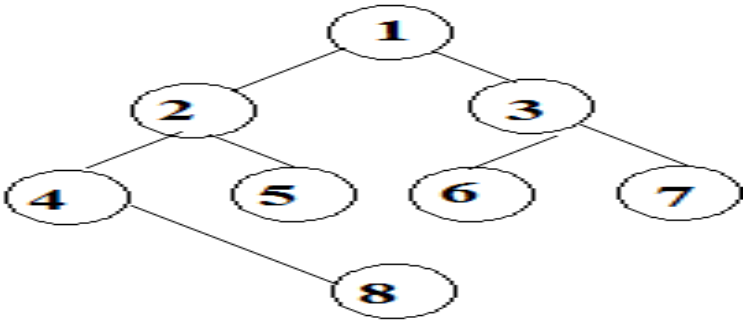
Kruskal's Algorithm: Start with *no* nodes or edges in the spanning tree, and repeatedlyadd the cheapest edge that does not create a cycle.

ConnectedComponent:
ConnectedcomponentofagraphcanbeobtainedbyusingBFST(Breadthfirstsearchand traversal) and DFST (Dept first search and traversal). It is also called the spanning tree.
BFST(Breadthfirstsearchandtraversal):

- In BFSwestartatavertexVmarkitasreached(visited).
- ThevertexVisatthistimesaidtobeunexplored(notyetdiscovered).
- Avertexissaidtobeenexplored(discovered)byvisitingallverticesadjacentfromit.
- AllunvisitedverticesadjacentfromVarevisitednext.
- Thefirstvertexon this listis the nexttobeexplored.
- Explorationcontinuesuntilnounexploredvertexisleft.
- Theseoperationscanbeperformedbyusing Queue.



Undirected Graph G



BFS Spanning tree

Thisisalsocalledconnectedgraphorspanningtree.
SpanningtreesobtainedusingBFSthenitcalledbreadthfirstspanningtrees.

AlgorithmforBFStoconvertundirectedgraphGtoConnectedcomponentorspanning tree.

AlgorithmBFS(v)
//abfsofG is beginatvertexv
//foranynode I, visited[i]=1 ifIhasalreadybeenvisited.
//thegraphG,andarrayvisited[]are global
{

```
U:=v;//qisaqueueofunexploredvertices.
Visited[v]:=1;
Repeat{
ForallverticeswadjacentfromUdo If
(visited[w]=0) then
{
Addwtoq;//wisunexploredVisited[w]:=1;
}
Ifqisemptythenreturn;//Nounexploredvertex. Delete U
from q; //Get 1st unexplored vertex.
} Until(false)
}
```

MaximumTimecomplexityandspacecomplexityofG(n,e),nodesareinadjacencylist. $T(n, e)=\theta(n+e)$

$S(n, e)=\theta(n)$

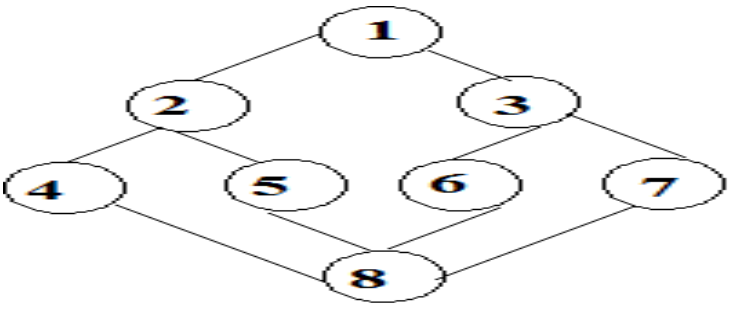
Ifnodesareinadjacencymatrixthen

$T(n, e)=\theta(n^2)$

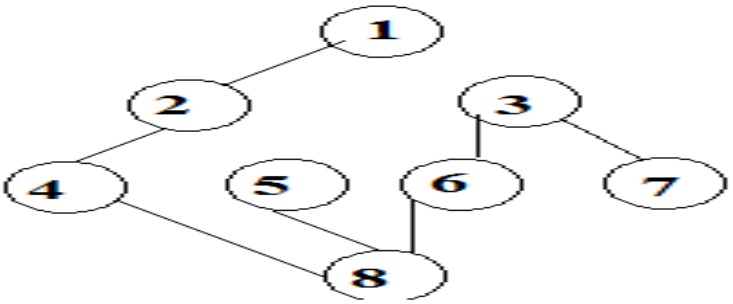
$S(n, e)=\theta(n)$

DFST(Deptfirstsearchandtraversal).:

- Dfsdifferentfrombfs
- Theexplorationofavertexvissuspended(stopped)assoonasanewvertexisreached.
- In this the exploration of the new vertex (example v) begins; this new vertex has been explored, the exploration of v continues.
- Note: exploration start at the new vertex which is not visited in other vertex exploring and choose nearest path for exploring next or adjacent vertex.



Undirected Graph G



DFS(1) Spanning tree

AlgorithmforDFStoconvertundirectedgraphGtoConnectedcomponentorspanning tree.

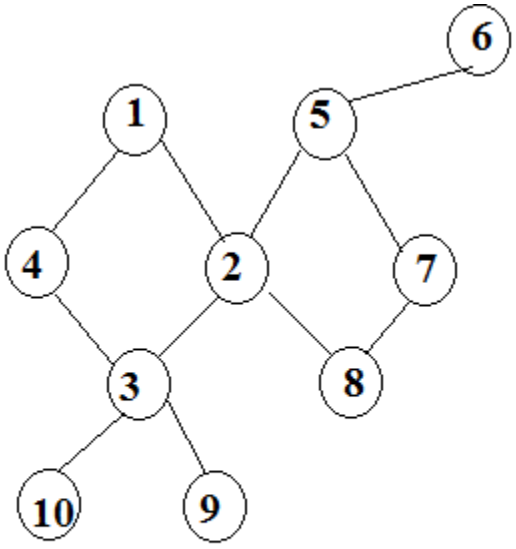
```
AlgorithmdFS(v)
//aDfsofGisbeginatvertexv
//initiallyanarrayvisited[] is settozero.
//thisalgorithmvisitsallverticesreachablefromv.
//thegraphG,andarrayvisited[] are global
{
Visited[v]:=1;
Foreachvertexwadjacentfromvdo
{
If(visited[w]=0)then DFS(w);
{
Addwtoq;//wisunexploredVisited[w]:=1;
}
}
```

MaximumTimecomplexityandspacecomplexityofG(n,e),nodesareinadjacencylist. T(n, e)=θ(n+e)
S(n, e)=θ(n)

Ifnodesareinadjacencymatrixthen
T(n, e)=θ(n^2)
S(n, e)=θ(n)

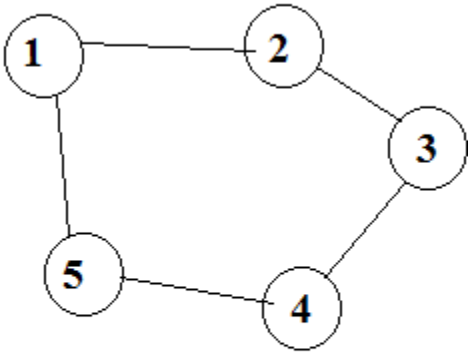
Bi-connectedComponents:

A graph G is biconnected, iff (if and only if) it contains no articulation point (joint or junction).
A vertex v in a connected graph G is an articulation point, if and only if (iff) the deletion of vertex v together with all edges incident to v disconnects the graph into two or more none empty components.



Graph G1

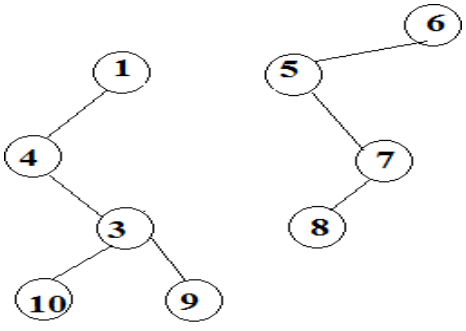
Not a biconnected graph



Graph Gb
Biconnected Graph

Thepresenceofarticulationpointsinaconnectedgraphcanbeanundesirable(unwanted) feature in many cases.
Forexample
ifG1→Communicationnetworkwith
Vertex → communication stations.
Edges→ Communication lines.

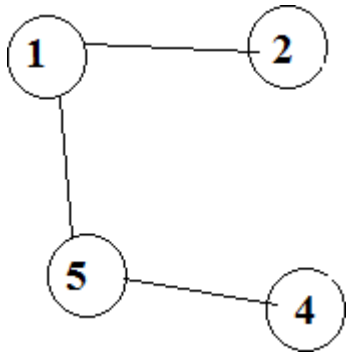
ThenthefailureofacommunicationstationIthatisanarticulationpoint,thenwelossthe communication in betweenother stations. F
FormgraphG1



After deleting vertex (2) (Here2isarticulationpoint)

Ifthegraphisbi-connectedgraph(meansnoarticulationpoint)thenifanystationi fails, we can still communicate between every two stations not including station i.

FromGraphGb



Thereisanefficientalgorithmtotestwhetheraconnectedgraphisbiconnected. Ifthecaseof graphs that are not biconnected, this algorithm will identify all the articulation points.
Once it has been determined that a connected graph G is not biconnected, it maybe desirable (suitable) to determine a set of edges whose inclusion makes the graph biconnected.

UNITIII:

Greedy method: General method, applications - Job sequencing with deadlines, 0/1 knapsack problem, Minimum cost spanning trees, Single source shortest path problem.

DynamicProgramming: Generalmethod,applications-Matrixchainmultiplication,Optimal binary search trees, 0/1 knapsack problem, All pairs shortest path problem, Travelling sales person problem, Reliability design.

GreedyMethod:

Thegreedymethodisperhaps(maybeorpossible)themoststraightforwarddesign technique, used to determine a feasible solution that may or may not be optimal.

Feasiblesolution:-Mostproblemshaveninputsanditssolutioncontainsasubsetofinputs that satisfies a given constraint(condition). Any subset that satisfies the constraint is called feasible solution.

Optimalsolution:Tofindafeasiblesolutionthateithermaximizesorminimizesagiven objective function. A feasible solution that does this is called optimal solution.

The greedy method suggests that an algorithm works in stages, considering one input at a time.Ateachstage,adecisionismaderegardingwhetheraparticularinputisinanoptimalsolution.

Greedy algorithms neither postpone nor revise the decisions (ie., no back tracking).

Example:Kruskal'sminimalspanningtree.Selectanedgefromasortedlist,check,decide, and never visit it again.

ApplicationofGreedyMethod:

- Jobsequencingwithdeadline
- 0/1knapsackproblem
- Minimum cost spanningtrees
- Singlesourceshortestpathproblem.

AlgorithmforGreedymethod

AlgorithmGreedy(a,n)
//a[1:n] containstheninputs.
{
Solution :=0;
Fori=1tondo
{
X:=select(a);
If Feasible(solution, x) then
Solution:=Union(solution,x);
}
Returnsolution;
}

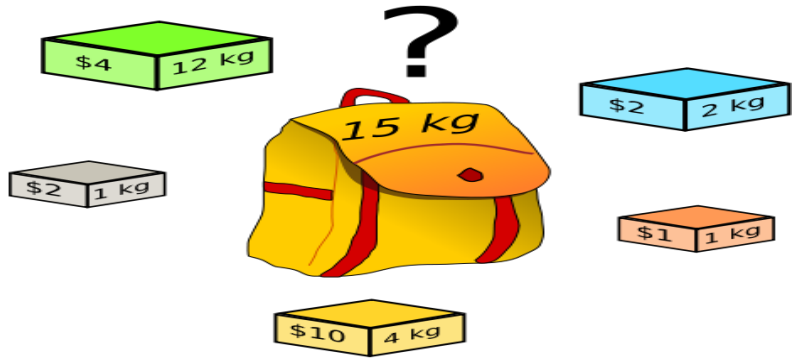
Selection→Function,thatselectsaninputfroma[]andremovesit.Theselectedinput's value is assigned to x.

Feasible→Boolean-valuedfunctionthatdetermineswhetherxcanbeincludedintothe solution vector.

Union→functionthatcombinesxwithsolutionandupdatestheobjectivefunction.

Knapsack problem

The**knapsackproblem**or**rucksack (bag) problem**is a problem in combinatorial optimization: Given a set of items, each with a mass and a value, determine the number of each item to include in a collection so that thetotal weight is less than or equal to a given limit and the total value is as large as possible



Therearetwoversionsoftheproblems

1. 0/1knapsackproblem
2. FractionalKnapsack problem
    a. BoundedKnapsack problem.
    b. UnboundedKnapsack problem.

Solutionstoknapsack problems

- **Brute-forceapproach**:-Solvetheproblemwithastraightfarwardalgorithm
- **GreedyAlgorithm**:-Keeptakingmostvaluableitemsuntilmaximumweightis reached or taking the largest value of eac item by calculating v_i=value$_i$/Size$_i$
- **Dynamic Programming**:-Solve each sub problem once and store their solutions in an array.

0/1knapsackproblem:

Lettherebe$n$items,$z_1$toz_n where$z_i$hasavalue$v_i$andweightw_i.Themaximumweight that we can carry in the bag is *W*. It is common to assume that all values and weights are nonnegative. To simplify the representation, we also assume that the items are listed in increasing order of weight.

Maximize$\sum_{i=1}^{n} v_i x_i$ subject to$\sum_{i=1}^{n} w_i x_i \leqslant W, \qquad x_i \in \{0,1\}$

Maximizethesumofthevaluesoftheitemsintheknapsacksothatthesumoftheweights mustbeless than the knapsack's capacity.

Greedyalgorithmforknapsack

AlgorithmGreedyKnapsack(m,n)
//p[i:n] and[1:n] containtheprofitsandweightsrespectively
//ifthen-objectsorderedsuchthatp[i]/w[i]>=p[i+1]/w[i+1],m→sizeofknapsackand x[1:n]→ the solution vector
{
Fori:=1tondox[i]:=0.0 U:=m;
Fori:=1ton do
{
if(w[i]>U)thenbreak;
x[i]:=1.0;
U:=U-w[i];
}
If(i<=n)then x[i]:=U/w[i];
}

Ex: - Consider 3 objects whose profits and weights are defined
as(P_1, P_2, P_3) = (25, 24, 15)
$W_1, W_2, W_3)$= (18,15,10)
n=3→number of objects
m=20→Bag capacity

Consideraknapsackofcapacity20.Determinetheoptimumstrategyforplacingtheobjects intotheknapsack.Theproblemcanbesolvedbythegreedyapproachwhereintheinputs arearrangedaccordingtoselectionprocess(greedystrategy)andsolvetheproblemin stages.Thevariousgreedystrategiesfortheproblemcouldbeasfollows.

(x_1,x_2,x_3)	$\sum x_i w_i$	$\sum x_i p_i$
(1,2/15,0)	$18x1+\frac{2}{15}x15=20$	$25x1+\frac{2}{15}x24=28.2$
(0,2/3,1)	$\frac{2}{3}x15+10x1=20$	$\frac{2}{3}x24+15x1=31$

(0,1,½)	1x15+ 1/2 x10=20	1x24+ 1/2 x15=31.5
(½,⅓,¼)	½x18+⅓x15+¼x10=16.5	½x25+⅓x24+¼x15=12.5+8+3.75 = 24.25

Analysis:-Ifwedonotconsiderthetimeconsideredforsortingtheinputsthenallofthe three greedy strategies complexity will be O(n).

JobSequencewithDeadline:

There is set of n-jobs. For anyjobi, is a integer deadling di≥0 and profit Pi>0, the profit Pi is earned iff the job completed by its deadline.

Tocompleteajobonehadtoprocessthejobonamachineforoneunitoftime.Onlyone machine is available for processing jobs.

A feasible solution for this problem is a subset J of jobs such that each job in this subset can be completed by its deadline.

Thevalue of afeasiblesolutionJisthesumoftheprofitsofthejobsin J, i.e., $\sum_{i \in j} P_i$

Anoptimalsolutionisafeasiblesolutionwithmaximumvalue.

Theprobleminvolvesidentificationofasubsetofjobswhichcanbecompletedbyits deadline.Therefore the problem suites the subsetmethodologyandcanbesolvedbythe greedy method.

Ex:-Obtaintheoptimalsequenceforthefollowingjobs.

$j_1 j_2 \quad j_3 j_4$

$(P_1, P_2, P_3, P_4) = (100,10, 15, 27)$

$(d_1, d_2, d_3, d_4) = (2,1,2, 1)$

n =4

Feasible solution	Processing sequence	Value
$j_1 j_2$ (1,2)	(2,1)	100+10=110
(1,3)	(1,3)or (3,1)	100+15=115
(1,4)	(4,1)	100+27=127
(2,3)	(2,3)	10+15=25
(3,4)	(4,3)	15+27=42
(1)	(1)	100
(2)	(2)	10
(3)	(3)	15
(4)	(4)	27

In the example solution '3' is the optimal. In this solution only jobs 1&4 are processed and the value is 127. These jobs must be processed in the order j4 followed by j1. the process of job 4 begins at time 0 and ends at time 1. And the processing of job 1 begins at time 1 and ends at time2. Therefore both the jobs are completed within their deadlines. The optimization measure for determining the next job to be selected in to the solution is according to theprofit. The next job to include is that which increases ∑pi the most, subject to the constraint that the resulting "j" is the feasible solution. Therefore the greedy strategy is to consider the jobs in decreasing order of profits.

Thegreedyalgorithm is used to obtain an optimal solution.

Wemustformulateanoptimizationmeasuretodeterminehowthenextjob is chosen.

```
algorithmjs(d,j,n)
//d→deadline,j→subsetofjobs ,n→totalnumberofjobs
//d[i]≥1 1 ≤i≤ narethedeadlines,
//thejobsareorderedsuchthatp[1]≥p[2]≥---≥p[n]
//j[i]istheith job intheoptimal solution 1 ≤i≤ k, k→ subset range
{
d[0]=j[0]=0;
j[1]=1;
k=1;
fori=2tondo{
r=k;
while((d[j[r]]>d[i])and[d[j[r]]≠r))do
r=r-1;
if((d[j[r]]≤d[i])and(d[i]>r))then
{
forq:=kto(r+1)setp-1doj[q+1]=j[q];
j[r+1]=i;
k=k+1;
}
}
returnk;
}
```

Note:Thesizeofsubsetjmustbelessthanequaltomaximumdeadlineingivenlist.

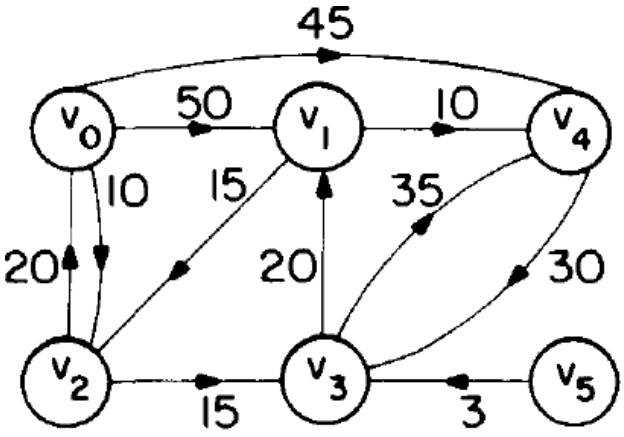
SingleSourceShortestPaths:

- Graphs can be used to represent the highway structure of a state or country with vertices representing cities and edges representing sections of highway.
- The edges have assigned weights which may be either the distance between the 2cities connected by the edge or the average time to drive along that section ofhighway.
- For example if A motorist wishing to drive from city A to B then we must answer the following questions
    - Isthereapathfrom A toB
    - If thereismorethanonepathfrom Ato Bwhichistheshortestpath
- Thelengthofapathisdefined to bethesumoftheweightsoftheedgeson that path.

Given a directed graph G(V,E) with weight edge w(u,v). e have to find a shortest path

fromsource vertex S∈v to every other vertex v1∈ V−S.

- TofindSSSPfordirectedgraphsG(V,E)thereareetwodifferentalgorithms.
  - Bellman-FordAlgorithm
  - Dijkstra'salgorithm
- Bellman-FordAlgorithm:-allow–veweightedgesininputgraph.Thisalgorithm eitherfindsashortestpathformsourcevertex**S**∈**V**toothervertex**v**∈**V**ordetecta–veweightcyclesinG,hencenosolution.Ifthereisnonegativeweightcyclesare reachableformsourcevertex**S**∈**V**toeveryothervertex**v**∈**V**
- Dijkstra'salgorithm:-allowsonly+veweightedgesintheinputgraphandfindsa shortest path from source vertex **S**∈**V**toeveryother vertex **v**∈**V**.



	Path	Length
1)	$v_0\ v_2$	10
2)	$v_0\ v_2\ v_3$	25
3)	$v_0\ v_2\ v_3\ v_1$	45
4)	$v_0\ v_4$	45

Graph and shortest paths from v_0 to all destinations

- Consider the above directed graph, if node 1 is the source vertex, then shortest pathfrom 1 to 2 is 1,4,5,2. The length is 10+15+20=45.
- Toformulateagreedybasedalgorithmtogeneratetheshortestpaths,wemustconceive of a multistage solution to the problem and also of an optimization measure.
- This ispossiblebybuildingtheshortestpathsoneby one.
- Asanoptimizationmeasurewecanusethesumofthelengthsofallpathssofar generated.
- Ifwehavealreadyconstructed'i'shortestpaths,thenusingthisoptimizationmeasure, the next path to be constructed should be the next shortest minimum length path.
- The greedy way to generate the shortest paths from Vo to the remaining vertices is to generate these paths in non-decreasing order of path length.
- For this 1st, a shortest path of the nearest vertex is generated. Then a shortest path tothe 2nd nearest vertex is generated and so on.

AlgorithmforfindingShortestPath

```
AlgorithmShortestPath(v,cost,dist,n)
//dist[j],1≤j≤n,issettothelengthoftheshortestpathfromvertexvtovertexjingraphg with n-vertices.
//dist[v]is zero
{
for i=1 to n do{
s[i]=false;
dist[i]=cost[v,i];
}
s[v]=true;
dist[v]:=0.0;//putvins for
num=2 to n do{
//determinen-1pathsfrom v
chooseuformamongthoseverticesnotinssuchthatdist[u]isminimum. s[u]=true; // put
u in s
for(eachwadjacenttouwiths[w]=false)do if(dist[w]>(dist[u]+cost[u,
w])) then
dist[w]=dist[u]+cost[u,w];
}
}
```

MinimumCostSpanningTree:

SPANNINGTREE: -A Subgraph'n' ofograph'G'iscalledasaspanningtreeif

(i) Itincludesalltheverticesof'G'

(ii) Itisatree

Minimum cost spanning tree: For a given graph 'G' there can be more than one spanning tree. If weights are assigned to the edges of 'G' then the spanning tree which has the minimum cost of edges is called as minimal spanning tree.

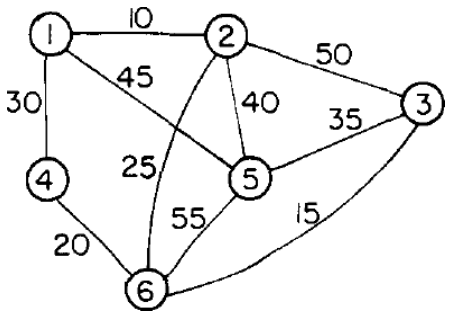
Thegreedymethodsuggeststhataminimumcostspanningtreecanbeobtainedbycontacting the tree edge by edge. The next edge to be included in the tree is the edge that results in a minimum increase in the some of the costs of the edges included so far.

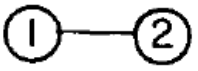
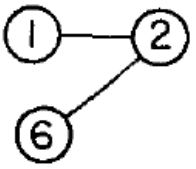
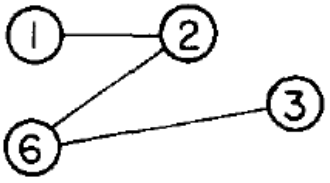
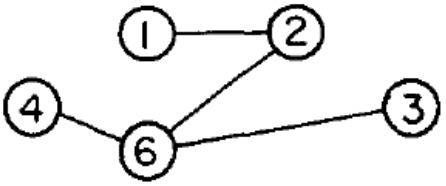
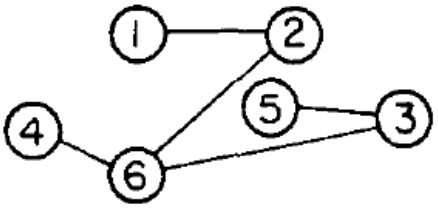
There are two basic algorithms for finding minimum-cost spanning trees, and both are greedy algorithms

→Prim'sAlgorithm

→Kruskal'sAlgorithm

Prim'sAlgorithm:Startwithany*onenode*inthespanningtree,andrepeatedlyaddthe cheapest edge, and the node it leads to, for which the node is not alreadyin the spanning tree.



Edge	Cost	Spanning tree
(1,2)	10	
(2,6)	25	
(3,6)	15	
(6,4)	20	
(1,4)	reject	
(3,5)	35	

Stages in Prim's Algorithm

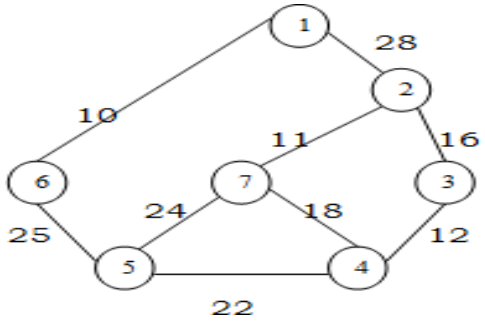
PRIM'SALGORITHM:-

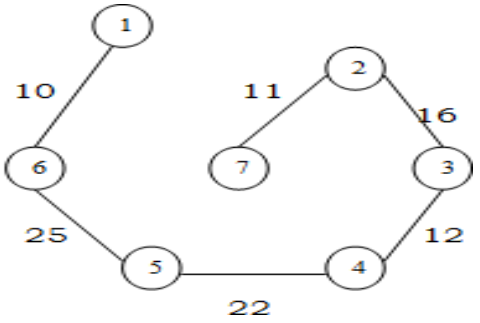
i) Selectanedgewithminimumcostandincludeintothespanningtree.

ii) Amongalltheedgeswhichareadjacentwiththeselectededge,selecttheone with minimum cost.

iii) Repeatstep2until'n'verticesand(n-1)edgesarebeenincluded.Andthesub graphobtained does notcontain any cycles.

Notes: - At every state a decision is made about an edge of minimum cost to be included into the spanning tree. From the edges which are adjacent to the last edge included in the spanning tree i.e. at every stage the sub-graph obtained is a tree.





Prim'sminimumspanningtreealgorithm

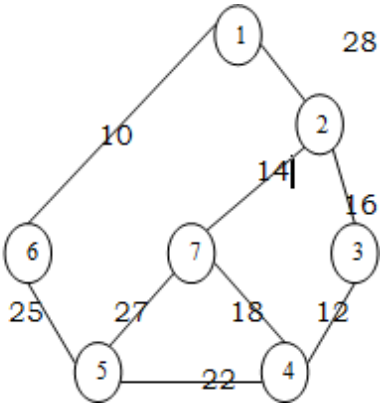
```
AlgorithmPrim(E,cost,n,t)
//EisthesetofedgesinG.Cost(1:n,1:n)is the
//Costadjacencymatrixofannvertexgraphsuchthat
//Cost(i,j)iseitherapositiverealno.or∞ifnoedge(i,j)exists.
//Aminimumspanningtreeiscomputedand
//Storedinthe array T(1:n-1, 2).
//(t(i,1),+t(i,2))isanedgeintheminimumcostspanningtree.Thefinalcostisreturned
        {
        Let (k, l) be an edge with min cost in E
        Min cost: = Cost (x,l);
        T(1,1):=k;+(1,2):=l;
fori:=1tondo//initializenear
        if (cost (i,l)<cost (i,k) then n east (i):l;else
        near (i): = k;
        near (k): = near (l): = 0;
        for i: = 2 to n-1 do
{//findn-2additionaledgesfort
letjbeanindexsuchthatnear(i)≠0&cost(j,near(i))isminimum; t (i,1): = j +
(i,2): = near (j);
min cost: = Min cost + cost (j, near (j));
near (j): = 0;
fork:=1tondo//updatenear()
if ((near (k) ≠0) and (cost {k, near (k)) > cost (k,j)))
then near Z(k): = ji
}
returnmincost;
}
```

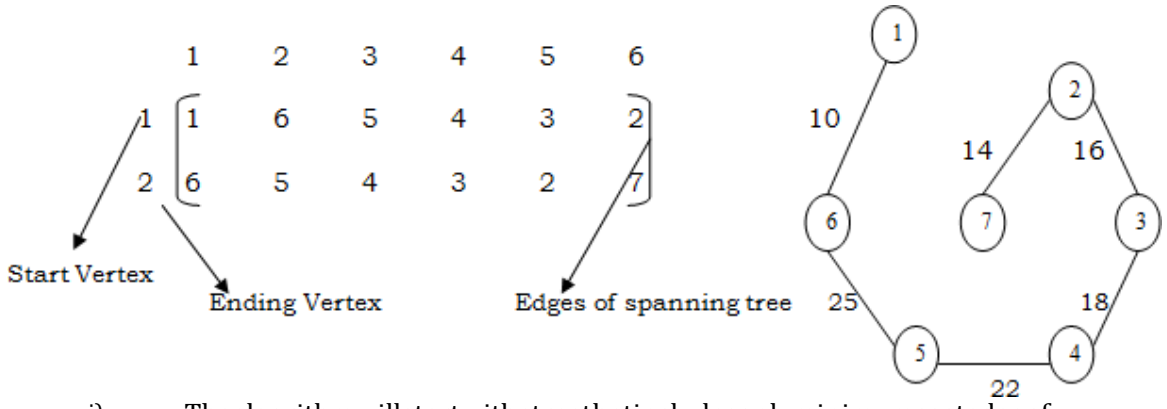
ThealgorithmtakesfourargumentsE:setofedges,costisnxnadjacencymatrixcostof (i,j)= +ve integer, if an edge exists between i&j otherwise infinity. 'n' is no/:of vertices. 't' is a(n-1):2matrix which consists oftheedgesofspanningtree.

E={(1,2),(1,6),(2,3),(3,4),(4,5),(4,7),(5,6),(5,7),(2,7)}

n={1,2,3,4,5,6,7)

Cost	1	2	3	4	5	6	7
1	α	28	α	α	α	10	α
2	28	α	16	α	α	α	14
3	α	10	α	12	α	α	α
4	α	α	12	α	22	α	18
5	α	α	α	22	α	25	24
6	10	α	α	α	25	α	α
7	α	14	α	18	24	α	α





i) Thealgorithmwillstartwithatreethatincludesonlyminimumcostedgeof G.Thenedgesareaddedtothistreeonebyone.

ii) The next edge (i,j) to be added is such that i is a vertex which is already included in the treed and j is a vertex not yet included in the tree and cost of i,j is minimum among all edges adjacent to 'i'.

iii) Witheachvertex'j'nextyetincludedinthetree,weassignavaluenear'j'. The value near 'j' represents a vertex in the tree such that cost (j, near (j)) is minimum among all choices for near (j)

iv) Wedefinenear(j):=0forallthevertices'j'thatarealreadyinthetree.

v) The next edge to include is defined by the vertex 'j' such that (near (j))≠0 and cost of (j, near (j)) is minimum.

Analysis:-

Thetimerequiredbytheprincealgorithmisdirectlyproportionaltotheno/:ofvertices.Ifa graph'G'has'n'verticesthenthetimerequiredbyprim'salgorithmis**$O(n^2)$**

Kruskal's Algorithm: Start with *no* nodes or edges in the spanning tree, and repeatedly add the cheapest edge that does not create a cycle.

InKruskalsalgorithmfordeterminingthespanningtreewearrangetheedgesinthe increasing order of cost.

i) All the edges are considered one by one in that order and deleted from the graphandareincludedintothespanningtree.

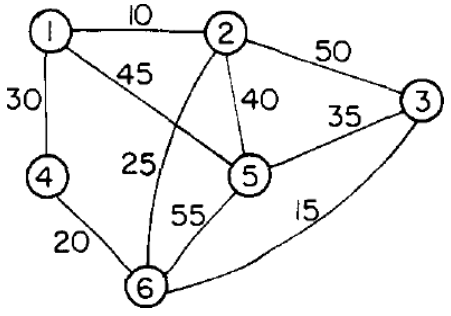
ii) At every stage an edge is included; the sub-graph at a stage need not be a tree. Infect it is a forest.

iii) At the end if we include 'n' vertices and n-1 edges without forming cycles then wegetasingleconnectedcomponentwithoutanycyclesi.e.atreewithminimum cost.

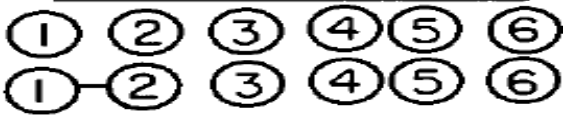
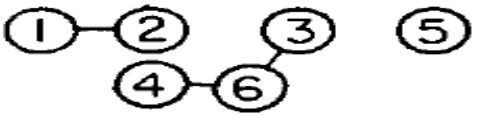
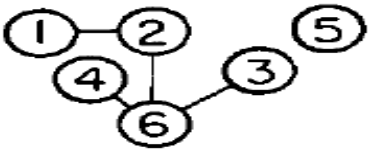
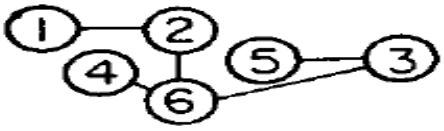
At every stage, as we include an edge in to the spanning tree, we get disconnected trees representedbyvarioussets.Whileincludinganedgeintothespanningtreeweneedto checkitdoesnotformcycle.Inclusionofanedge(i,j)willformacycleifi,jbothareinsame set.Otherwisetheedgecanbeincludedintothespanningtree.

Kruskalminimumspanningtree algorithm

```
AlgorithmKruskal(E,cost,n,t)
//EisthesetofedgesinG.'G'has'n'vertices
//Cost{u,v}isthecostofedge(u,v)tistheset
//ofedgesintheminimumcostspanningtree
//Thefinalcostisreturned
{constructaheapoutoftheedgecostsusingheapify;
        fori:=1tondoparent(i):=-1//placeindifferent sets
//each vertex is in different set          {1}{1}{3}
        i: = 0; min cost: = 0.0;
        While(i<n-1)and(heapnotempty))do
{
Deleteaminimumcostedge(u,v)fromtheheaps;andreheapifyusingadjust; j:= find
(u); k:=find (v);
if(j≠k)then
{i:=1+1;
   +(i,1)=u;+(i,2)=v;
    mincost:=mincost+cost(u,v);
    Union (j,k);
    }
}
if(i≠n-1)thenwrite("Nospanningtree"); else
   return mincost;
}
```



Consider the above graph of , Using Kruskal's method the edges of this graph are considered for inclusionintheminimumcostspanningtreeintheorder(1,2),(3,6),(4,6),(2,6),(1,4), (3, 5), (2, 5), (1, 5), (2, 3), and (5, 6). This corresponds to the cost sequence10, 15, 20, 25,30, 35, 40, 45, 50, 55. The first four edges are included in T. The next edge to be consideredis (I, 4). This edge connects two vertices already connected in T and so it is rejected. Next,the edge (3, 5) is selected and that completes the spanning tree.

Edge	Cast	Spanning Forest
(1,2)	10	
(3,6)	15	
(4,6)	20	
(2,6)	25	
(1,4)	30	(reject)
(3,5)	35	

Stages in Kruskal's algorithm

Analysis:-Iftheno/:ofedgesinthegraphisgivenby/E/thenthetimeforKruskals algorithm is given by 0 (|E| log |E|).

Dynamic Programming

Dynamic programming is a name, coined by Richard Bellman in 1955. Dynamic programming, as greedy method, is a powerful algorithm design technique that can be used when the solution to the problem may be viewed as the result of a sequence of decisions. In the greedy method we make irrevocable decisions one at a time, using a greedy criterion. However, in dynamic programming we examine the decision sequence to see whether an optimal decision sequence contains optimal decision subsequence.

When optimal decision sequences contain optimal decision subsequences, we can establish recurrence equations, called *dynamic-programming recurrence equations,* that enable us to solve the problem in an efficient way.

Dynamicprogrammingisbasedontheprincipleofoptimality(alsocoinedbyBellman). The principle of optimality states that no matter whatever the initial state and initial decision are, the remaining decision sequence must constitute an optimal decision sequence with regard to the state resulting from the first decision. The principle implies that an optimal decision sequence is comprised of optimal decision subsequences. Since the principle of optimality may not hold for some formulationsof some problems, it is necessary to verify that it does hold for the problem being solved. Dynamic programming cannot be appliedwhen this principle does not hold.

Thestepsinadynamicprogrammingsolutionare:

- Verifythattheprincipleofoptimalityholds
- Setupthedynamic-programmingrecurrence equations
- Solve the dynamic-programming recurrence equations forthe value of the optimal solution.
- Performatracebackstepinwhichthesolutionitselfisconstructed.

5.1MULTISTAGEGRAPHS

A multistage graph G = (V, E) is a directed graph in which the vertices arepartitioned intok≥2disjointsetsVi,1≤i≤k.Inaddition,if<u,v>isanedgeinE,thenu∈Viandv ∈Vi+1 for some i, 1 ≤i< k.

Let the vertex ‘s’ is the source, and ‘t’ the sink. Let c (i, j) be the cost of edge <i, j>. Thecost of a path from ‘s’ to ‘t’ is the sum of the costs of the edges on the path. The multistage graph problem is to find a minimum cost path from ‘s’ to ‘t’. Each set Vi defines a stage in the graph.BecauseoftheconstraintsonE,everypathfrom‘s’to ‘t’startsinstage1,goestostage 2,thentostage3,thentostage4,andsoon,andeventuallyterminates in stage k.

A dynamic programming formulation for a k-stage graph problem is obtained byfirst noticing that every s to t path is the result of a sequence of k– 2 decisions. The ith

decision involvesdeterminingwhichvertexin$_{vi+1,}$1≤i≤k-2,istobe onthepath.Letc (i, j) be the cost of the path from source to destination. Then using the forward approach, we obtain:

cost(i, j) =min{c(j, l) +cost (i + 1, l)}
lcVi+ 1
<j,l>cE

ALGORITHM:

AlgorithmFgraph(G,k,n,p)
// The input is a k-stage graph G = (V, E) with n vertices //
indexedinorderorstages.Eisasetofedgesandc[i,j]//isthe cost of (i,
j). p [1 : k] is a minimum cost path.
{
cost[n]:=0.0;
forj:=n -1 to 1 step – 1do
{ //computecost[j]
let r be a vertex such that (j, r) is an edge of G
and c [j, r] + cost [r] is minimum; cost[j]:= c
[j, r] + cost [r];
d[j]:= r:
}
p [1] := 1; p [k] := n; //Findaminimumcostpath.
for j := 2 to k - 1 do p [j] := d [p [j - 1]];}

The multistage graph problem can also be solved using the backward approach. Letbp(i, j)beaminimumcostpathfromvertexstojvertexinVi.LetBcost(i,j)bethecostofbp(i, j).Fromthebackwardapproachweobtain:

Bcost(i,j)=min{Bcost(i–1,l)+c(l,j)} l e Vi
- 1
<l,j>eE

AlgorithmBgraph(G,k,n,p)
//SamefunctionasFgraph{
Bcost [1] := 0.0; for j := 2 to n do { // Compute Bcost
[j].
Letrbe such that(r,j)is anedge of
GandBcost[r]+c[r,j]isminimum;
Bcost[j]:=Bcost[r]+c[r,j];
D[j]:=r;
} //findaminimumcostpath
p[1]:=1;p[k]:=n;
forj:=k-1to2dop[j]:=d[p[j+1]];
}

ComplexityAnalysis:

Thecomplexityanalysisofthealgorithmisfairlystraightforward.Here,ifGhas~E~edges, thenthetimeforthefirstforloopisCJ(V~+~E).

EXAMPLE1:

Findtheminimumcostpathfromstotinthemultistagegraphoffivestagesshownbelow. Do thisfirstusingforwardapproachandthenusingbackwardapproach.

FORWARDAPPROACH:

Weusethefollowingequationtofindtheminimumcostpathfromstot:cost(i, j)=min{c(j,l)+cost(i+1,l)}
lcVi+1
<j,l>cE
cost(1,1)=min{c(1,2)+cost(2,2),c(1,3)+cost(2,3),c(1,4)+cost(2,4),c(1,5)+ cost(2, 5)}
=min{9+cost(2,2),7+cost(2,3),3+cost(2,4),2+cost(2,5)}

Nowfirststartingwith,

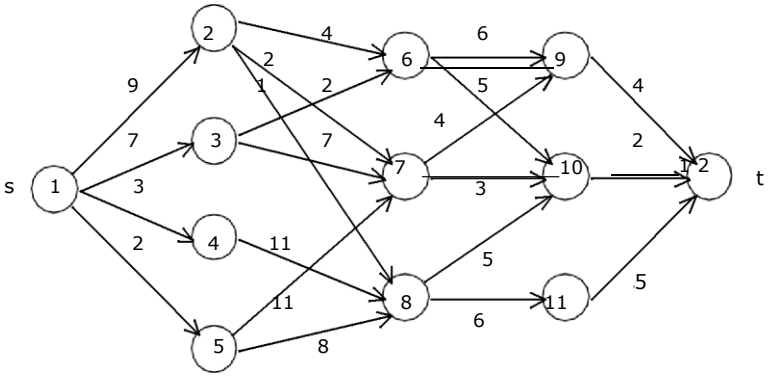
cost(2,2)=min{c(2,6)+cost(3,6),c(2,7)+cost(3,7),c(2,8)+cost(3,8)}=min{4 + cost(3,6),2+cost(3,7),1+cost(3,8)}

cost(3,6)=min{c(6,9)+cost(4,9),c(6,10)+cost(4,10)}
=min{6+cost(4,9),5+cost(4,10)}

cost(4,9)=min{c(9,12)+cost(5,12)}=min{4+0)=4cost(4,

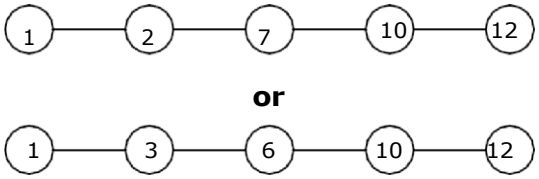
10)=min{c(10,12)+cost(5,12)}=2

Therefore,cost(3,6)=min{6+4,5+2}=7



cost(3,7)=min{c(7,9)+cost(4,9),c(7,10)+cost(4,10)}
=min{4+cost(4,9),3+cost(4,10)}

cost(4,9)=min{c (9,12)+cost(5,12)}=min{4+0}=4Cost(4,

10)= Thepathis 

min

{c

(10,

2)+cost(5,12)}=min{2+0}=2Therefore,cost(3,7)=min{4+4,3

+2}=min{8,5}=5

_____

cost(3,8)=min{c(8,10)+cost(4,10),c(8,11)+cost(4,11)}
=min{5+cost(4,10),6+cost(4+11)}

cost(4,11)=min{c(11,12)+cost(5,12)}=5

Therefore,cost(3,8)=min{5+2,6+5}=min{7,11}=7

_____

Therefore,cost(2,2)=min{4+7,2+5,1+7}=min{11,7,8}=7

_____

Therefore,cost(2,3)=min{c(3,6)+cost(3,6),c(3,7)+cost(3,7)}
=min{2+cost(3,6),7+cost(3,7)}
=min{2+7,7+5}=min{9,12}=9

cost(2,4)=min{c(4,8)+cost(3,8)}=min{11+7}=18cost(2,5)=
min{c(5,7)+cost(3,7),c(5,8)+cost(3,8)}=min{11+5,8+
7}=min{16,15}=15

_____

Therefore,cost(1,1)=min{9+7,7+9,3+18,2+15}=min
{16,16,21,17}= 16

Theminimumcostpathis16.

BACKWARD APPROACH:

Weusethefollowingequationtofindtheminimumcostpathfromttos:Bcost(i,J)=min

{Bcost(i–1,l)+c(l,J)}

lcvi–1
<l,j>cE

Bcost(5,12)=min{Bcost(4,9)+c(9,12),Bcost(4,10)+c(10,12),
Bcost(4,11)+c(11,12)}
=min{Bcost(4,9)+4,Bcost(4,10)+2,Bcost(4,11)+5}

Bcost(4,9)=min{Bcost(3,6)+c(6,9),Bcost(3,7)+c(7,9)}
=min{Bcost(3,6)+6,Bcost(3,7)+4}

Bcost(3,6)=min{Bcost(2,2)+c(2,6),Bcost(2,3)+c(3,6)}
=min{Bcost(2,2)+4,Bcost(2,3)+2}

Bcost(2,2)=min{Bcost(1,1)+c(1,2)}=min{0+9}=9Bcost(2,3)=min

{Bcost(1,1)+c(1,3)}=min{0+7}=7Bcost(3, 6)= min{9+ 4, 7+ 2} =

min{13,9}=9

Bcost(3,7)=min{Bcost(2,2)+c(2,7),Bcost(2,3)+c(3,7),Bcost(2,5)+c(5,
7)}

Bcost(2,5)=min{Bcost(1,1)+c(1,5)}=2

Bcost(3,7)=min{9+2,7+7,2+11}=min{11,14,13}=11Bcost(4,9)=min {9

+6,11+4}=min{15,15}=15

Bcost(4,10)=min{Bcost(3,6)+c(6,10),Bcost(3,7)+c(7,10),
Bcost(3,8)+c(8,10)}

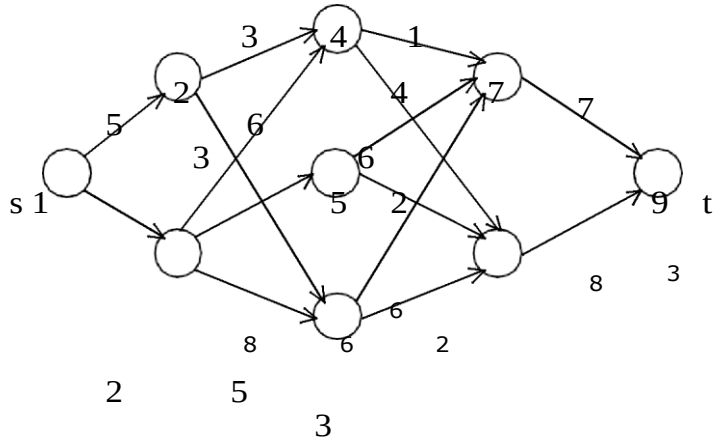
Bcost(3,8)=min{Bcost(2,2)+c(2,8),Bcost(2,4)+c(4,8),
Bcost(2,5)+c(5,8)}

Bcost(2,4)=min{Bcost(1,1)+c(1,4)}=3
Bcost(3,8)=min{9+1,3+11,2+8}=min{10,14,10}=10Bcost(4,10)=min{9
+5,11+3,10+5}=min{14,14,15)=14
Bcost(4,11)=min{Bcost(3,8)+c(8,11)}=min{Bcost(3,8)+6}=min{10+6}=
16

Bcost(5,12)=min{15+4,14+2,16+5}=min{19,16,21}=16.**EXAMPLE**

2:

Findtheminimumcostpathfromstotinthemultistagegraphoffivestagesshownbelow.Do thisfirstusingforwardapproachandthenusingbackwardapproach.



SOLUTION:FORWARD

APPROACH:

cost(i,J)=min{c (j,l)+cost (i+1,l)}
lc Vi + 1
<J, l>EE

cost(1,1)=min{c(1,2)+cost(2,2),c(1,3)+cost(2,3)}
=min{5+cost(2,2),2+cost(2,3)}

cost(2,2)=min{c(2,4)+cost(3,4),c(2,6)+cost(3,6)}
=min{3+cost(3,4),3+cost(3,6)}

cost(3,4)=min{c(4,7)+cost(4,7),c(4,8)+cost(4,8)}
=min{(1+cost(4,7),4+cost(4,8)}

cost(4,7)=min{c(7,9)+cost(5,9)}=min{7+0)=7cost(4,8)

=min{c(8,9)+cost(5,9)}=3

Therefore,cost(3,4)=min{8,7}=7

cost(3,6)=min{c(6,7)+cost(4,7),c(6,8)+cost(4,8)}
=min{6+cost(4,7),2+cost(4,8)}=min{6+7,2+3}=5

Therefore,cost(2,2)=min{10,8}= 8

cost(2,3)=min{c(3,4)+cost(3,4),c(3,5)+cost(3,5),c(3,6)+cost(3,6)}

cost(3,5)=min{c(5,7)+cost(4,7),c(5,8)+cost(4,8)}=min{6+7,2+3}=5

Therefore,cost(2,3)=min{13,10,13}=10

cost(1,1)=min{5+8,2+10}=min{13,12}=12

BACKWARDAPPROACH:

Bcost(i,J)=min{Bcost(i–1,l)=c(l,J)}
l∈vi–1
<l,j>∈E

Bcost(5,9)=min{Bcost(4,7)+c(7,9),Bcost(4,8)+c(8,9)}
=min{Bcost(4,7)+7,Bcost(4,8)+3}

Bcost(4,7)=min{Bcost(3,4)+c(4,7),Bcost(3,5)+c(5,7),
Bcost(3,6)+c(6,7)}
=min{Bcost(3,4)+1,Bcost(3,5)+6,Bcost(3,6)+6}

Bcost(3,4)=min{Bcost(2,2)+c(2,4),Bcost(2,3)+c(3,4)}
=min{Bcost(2,2)+3,Bcost(2,3)+6}

Bcost(2,2)=min{Bcost(1,1) +c(1, 2)}=min{0+5} = 5

Bcost(2,3)=min(Bcost(1,1) +c(1, 3)}=min{0+2} = 2

Therefore,Bcost(3,4)=min{5+3,2 +6}=min{8,8} = 8

Bcost(3,5)=min{Bcost(2,3)+c(3,5)}=min{2+5}=7

Bcost(3,6)=min{Bcost(2,2)+c(2,6),Bcost(2,3)+c(3,6)}=min
{5+5,2+8}=10

Therefore,Bcost(4,7)=min{8+1,7+6,10+6}=9

Bcost(4,8)=min{Bcost(3,4)+c(4,8),Bcost(3,5)+c(5,8),Bcost
(3,6)+c(6,8)}
=min{8+4,7+2,10+2}=9

Therefore,Bcost(5,9)=min{9+7,9+3}=12**All**

pairsshortest paths

Intheallpairsshortestpathproblem,wearetofindashortestpathbetweeneverypairof verticesinadirectedgraphG.Thatis,foreverypairofvertices(i,j),weare tofindashortestpathfromitojaswellasonefromjtoi.Thesetwopathsarethesamewhen Gis undirected.

When noedge has a negative length, the all-pairs shortest pathproblemmaybe solved by using Dijkstra's greedy single source algorithm n times, once with each of the n verticesasthesourcevertex.

TheallpairsshortestpathproblemistodetermineamatrixAsuchthatA(i,j)isthelength ofashortestpathfromitoj.ThematrixAcanbeobtainedbysolvingnsingle-source

problemsusingthealgorithmshortestPaths.Sinceeachapplicationofthisprocedure requiresO(n^2)time,thematrixAcanbeobtainedinO(n^3)time.

Thedynamicprogrammingsolution,calledFloyd'salgorithm,runsinO(n^3)time.Floyd's algorithm works even when the graph has negative length edges (provided there are no negative length cycles).

The shortest i to j path in G, i ≠ j originates at vertex i and goes through some intermediatevertices(possiblynone)andterminatesatvertexj.Ifkisan intermediate vertex on this shortest path, then the subpaths from i to k and from k to j mustbeshortestpathsfromitokandktoj,respectively.Otherwise,theitojpathisnot of minimum length. So, the principle of optimality holds. Let A^k (i, j) represent the lengthofashortestpathfromitojgoingthroughnovertexofindexgreaterthank,weobtain:

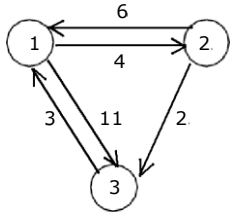
$$Ak(i,j)=\{\min\{\min_{1\le k<n}\{A^{k-1}(i,k)+A^{k-1}(k,j)\},c(i,j)\}$$

AlgorithmAllPaths(Cost,A,n)
//cost[1:n,1:n]isthecostadjacencymatrixofagraphwhich
//nvertices;A[I,j]isthecostofashortestpathfromvertex
//itovertexj.cost[i,i]=0.0,for1≤i≤n.
{
fori:=1tondo
forj:=1tondo
A [i, j] := cost [i, j]; //copycostintoA.
for k := 1 to n do
fori:=1tondo
forj:=1tondo
A[i,j]:=min(A[i,j],A[i,k]+A[k,j]);
}

Complexity Analysis:A Dynamic programming algorithm based on this recurrence involves in calculating n+1 matrices, each of size n x n. Therefore, the algorithm has a complexityofO(n^3).

Example1:

GivenaweighteddigraphG=(V,E)withweight.Determinethelengthoftheshortest pathbetweenallpairsofverticesinG.Hereweassumethattherearenocycleswithzero or negative cost.



Costadjacencymatrix(A^0)=

0	4	11
6	0	2
3	∞	0

Generalformula:min{A^{k-1}(i,k)+A^{k-1}(k,j)},c(i,j)}
1<k<n

Solvetheproblemfordifferentvaluesofk=1,2

and3**Step1**:Solvingtheequationfor,k=1;

A1(1,1)=min{(A^o(1,1)+A^o(1,1)),c(1,1)}=min{0+0,0}=0A1(1,
2)=min{(A^o(1,1)+A^o(1,2)),c(1,2)}=min{(0+4),4}=4

A1(1,3)=min{(A^o(1,1)+A^o(1,3)),c(1,3)}=min{(0+11),11}=11A1(2,
1)=min{(A^o(2,1)+A^o(1,1)),c(2,1)}=min{(6+0),6}=6

A1(2,2)=min{(A^o(2,1)+A^o(1,2)),c(2,2)}=min{(6+4),0)}=0A1(2,
3)=min{(A^o(2,1)+A^o(1,3)),c(2,3)}=min{(6+11),2}=2A1(3,1)=
min{(A^o(3,1)+A^o(1,1)),c(3,1)}=min{(3+0),3}=3A1(3,2)=min
{(A^o(3,1)+A^o(1,2)),c(3,2)}=min{(3+4),oc}=7A1(3,3)=min{(A^o
(3,1)+A^o(1,3)),c(3,3)}=min{(3+11),0}= 0

$A_{(1)}$= ~0 4 11□
~ ~
~6 0 2~
~L3 7 0~U

Step2:Solvingtheequationfor,K=2;

A_2(1, 1)=min{(A^1(1,2) +A^1(2,1),c(1,1)}=min{(4+6),0}+A^1 =0
A_2(1, 2)=min{(A^1(1,2) (2,2),c(1,2)}=min{(4+0),4}+A^1(2, =4
A_2(1, 3)=min{(A^1(1,2) 3),c(1,3)}=min{(4+2),11} =6
A_2(2, 1)=min{(A(2,2) +A(2, 1),c(2, 1)}=min{(0+6), 6} = 6
A_2(2, 2)=min{(A(2,2) +A(2, 2),c(2, 2)}=min{(0+0), 0} = 0
A_2(2, 3)=min{(A(2,2) +A(2, 3),c(2, 3)}=min{(0+2), 2} = 2
A_2(3, 1)=min{(A(3,2) +A(2, 1),c(3, 1)}=min{(7+6), 3} = 3
A_2(3, 2)=min{(A(3,2) +A(2, 2),c(3, 2)}=min{(7+0), 7} = 7
3)=min{(A(3,2) +A(2, 3),c(3, 3)}=min{(7+2), 0} = 0

$A_{(2)}$= ~0 4 61
~ 2~
~6 0 ~
~L3 7 0~~

Step3:Solvingtheequationfor,k=3;

A3(1, 1)=min{A^2(1,3) +A^2(3, 1),c(1, 1)}=min{(6+3), 0} = 0
A3(1, 2)=min{A^2(1,3) +A^2(3, 2),c(1, 2)}=min{(6+7), 4} = 4
A3(1, 3)=min{A^2(1,3) +A^2(3, 3),c(1, 3)}=min{(6+0), 6} = 6
A3(2, 1)=min{A^2(2,3) +A^2(3, 1),c(2, 1)}=min{(2+3), 6} = 5
A3(2, 2)=min{A^2(2,3) +A^2(3, 2),c(2, 2)}=min{(2+7), 0} = 0
A3(2, 3)=min{A^2(2,3) +A^2(3, 3),c(2, 3)}=min{(2+0), 2} = 2
A3(3, 1)=min{A^2(3,3) +A^2(3, 1),c(3, 1)}=min{(0+3), 3} = 3
A3(3, 2)=min{A^2(3,3) +A^2(3, 2),c(3, 2)}=min{(0+7), 7} = 7

A3(3,3)=min{A^2(3,3)+A^2(3,3),c(3,3)}=min{(0+0),0}=0

$A_{(3)}$= ~0 4 6~
0 ~
2~
~5~3 7 0~]

TRAVELLINGSALESPERSON PROBLEM

Let G=(V,E)beadirectedgraphwithedgecostsCij.Thevariablecijisdefinedsuch that cij>0 for allIand$_j$andcij=a if<i, j>oE.Let|V| = nandassumen>1.A tour ofG is a directed simple cycle that includes everyvertex in V. The cost of a tour is the sum of the cost of the edges on the tour. The traveling sales person problem is to find a tour of minimum cost. The tour is to be a simple path that starts and ends atvertex1.

Let g (i, S) be the length of shortest path starting at vertex i, going through allverticesinS, and terminating at vertex 1. The function g (1, V – {1}) is the length of an optimal salesperson tour. From the principal of optimality it follows that:

$$g(1,V-\{1\})=2\sim k\sim n\sim c1k\sim g\sim k,V\sim\sim 1,k\sim\sim \quad -- \quad 1$$

$$\min$$

Generalizingequation1,weobtain(forioS) -- 2

$$g(i,S)=\min\{ci\,j \quad jES$$

TheEquationcanbesolvedforg(1,V–1})ifweknowg(k,V–{1,k})forall choicesofk.

ComplexityAnalysis:

Foreachvalueof|S|there $+gi,S-j_{\text{»}}$aren–1choicesfori.Thenumberofdistinct sets S of sizeknotincluding1andiisI$k\sim\sim n-2\sim$.

Hence,thetotalnumberofg(i,S)'stobecomputedbeforecomputingg(1,V–{1})is:

$$\sim n-2\sim \quad \sim\sim n\sim 1\sim\sim \quad \sim k \quad \sim \quad k\sim 0 \quad \sim \quad \sim$$

Tocalculatethissum,weusethebinominaltheorem:

$$n-1 \quad [((n-2)((n-2)((n-2) \quad (n-1)111 \quad 11+\mathbf{i} \quad il+ii \quad il+\text{----}\sim\sim\sim \quad \sim \quad \sim\sim 0 \quad)\sim 1)\sim 2) \quad ((n-2)1 \quad \sim(n\sim^2)\sim\sim$$

Accordingtothebinominaltheorem:

$$[((n-2)((n-2)((n-2 \quad il \quad 11+\mathbf{i} \quad il+\mathbf{i} \quad \sim\sim\sim\sim\sim\sim\sim\sim \quad ((n-2)1 \quad \sim\sim\sim=2n-2 \quad \sim\sim 0 \quad \sim\sim 1\sim\sim 2\sim \quad \sim(n-2))]$$

Therefore,

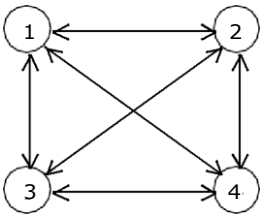
$$n-1 \quad \sim(n\_1\sim\sim_{\sim k} \quad \sim n\_2' \quad =(n-1)_2 n\sim 2 \quad \sim$$

This is Φ (n 2^{n-2}), so there are exponential number of calculate. Calculating one g (i, S) requirefindingtheminimumofatmostnquantities.Therefore,theentirealgorithmisΦ($n^2 2^{n-2}$).Thisisbetterthanenumeratingalln!differenttourstofindthebestone. So,wehavetradedonexponentialgrowthforamuchsmallerexponentialgrowth.

Themostseriousdrawbackofthisdynamicprogrammingsolution is the space needed, which is O (n 2^n). This is too large even for modestvalues of n.

Example1:

Forthefollowinggraphfindminimumcosttourforthetravelingsalesperson problem:



Thecostadjacencymatrix= $\begin{bmatrix} 0 & 10 & 15 & 20 \\ 5 & 0 & 9 & 10 \\ 6 & 13 & 0 & 12 \\ 8 & 9 & 0 & \infty \end{bmatrix}$

Letusstartthetourfromvertex1:

$$g(1,V-\{1\})=\min_{2<k<n}\{c_{1k}+g(k,V-\{1,K\})\} \quad - \quad (1)$$

Moregenerallywriting:

$$g(i, s) = \min \{cij + g(J, s - \{J\})\} \quad - \quad (2)$$

Clearly, $g(i, T) = ci1$, $1 \le i \le n$. So,

g(2, T)=C21 =5

g(3, T)=C31 =6

g(4, ~)=C41=8

Usingequation–(2)weobtain:

g(1,{2,3,4})=min{c12+g(2,{3, 4},c13+g(3,{2,4}),c14+g(4,{2,3})}

g(2, {3,4})=min{c23+g(3,{4}), c24+g(4,{3})}
=min{9+g(3,{4}), 10+g(4, {3})}

g(3, {4})=min{c34+g(4,T)}=12+8=20

g(4, {3})=min{c43+g(3,~)}=9 +6=15

Therefore,g(2,{3,4})=min{9+20,10+15}=min{29,25}=25

g(3,{2,4})=min{(c32+g(2,{4}),(c34+g(4,{2})}

g(2,{4})=min{c24+g(4,T)}=10+8=18

g(4,{2})=min{c42+g(2,~)}=8+5=13

Therefore,g(3,{2,4})=min{13+18,12+13}=min{41,25}=25

g(4,{2,3})=min{c42+g(2,{3}),c43+g(3,{2})}

g(2, {3})=min{c23+g(3,~}= 9 +6 =15

g(3, {2})=min{c32+g(2,T}= 13 +5 =18

Therefore,g(4,{2,3})=min{8+15,9+18}=min{23,27}=23

g(1,{2,3,4})=min{c12+g(2,{3,4}),c13+g(3,{2,4}),c14+g(4,{2,3})}=min
{10+25,15+25,20+23}=min{35,40,43}=35

Theoptimaltourforthegraphhaslength=35The

optimaltouris: 1, 2,4,3, 1.

OPTIMALBINARYSEARCHTREE

Letusassumethatthegivensetofidentifiersis{a1,...,an}with a1< a2<<an.
Letp(i)betheprobabilitywithwhichwesearchforai.Letq(i)betheprobabilitythatthe identifierxbeingsearchedforissuchthatai<x<ai+1,0≤i≤n(assumea0=-~and an+1 = +oc). We have to arrange the identifiers in a binary search tree in a way that minimizestheexpectedtotalaccesstime.
Inabinarysearchtree,thenumberofcomparisonsneededtoaccessanelementatdepth'd'isd+1,s oif'ai'isplacedatdepth'di',thenwewanttominimize:

$$\sum_{i=1}^{n} Pi(1+di).$$

Let P (i) be the probability with which we shall be searching for'ai'. Let Q (i) be the probability of an un-successful search. Every internal node represents a point where a successful search may terminate. Every external node represents a point where an unsuccessfulsearchmayterminate.

Theexpectedcostcontributionfortheinternalnodefor'ai'is:

$$P(i)*level(ai).$$

UnsuccessfulsearchterminatewithI =0(i.eatanexternalnode).Hencethecostcontributionforthisnodeis:

Q(i)*level((Ei)-1)

Theexpectedcostofbinarysearchtreeis:

$$\sum_{i=1}^{n} P(i)*level(ai) + \sum_{i=0}^{n} Q(i)*level((Ei)-1)$$

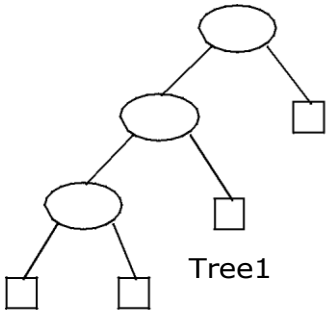
Givenafixedsetofidentifiers,wewishtocreateabinarysearchtreeorganization.We mayexpectdifferentbinarysearchtreesforthesameidentifiersettohavedifferent performancecharacteristics.

The computation of each of these c(i, j)'s requires us to find the minimum of m quantities.Hence,eachsuchc(i,j)canbecomputedintimeO(m).Thetotaltimeforall c(i,j)'swithj–i=mistherefore$O(nm–m^2)$.

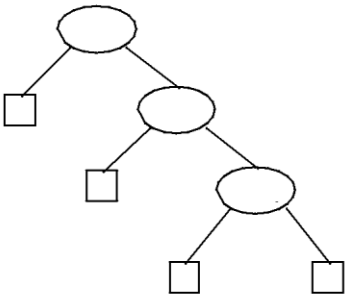
Thetotaltimetoevaluateallthec(i,j)'sandr(i,j)'sistherefore:

$$\sum(nm-m^2)=O(n^3)\quad 1<m<n$$

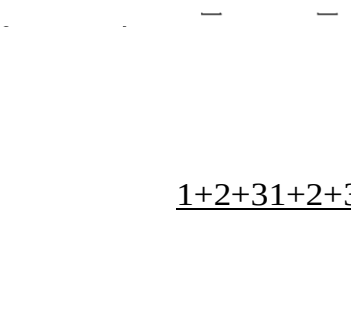
Example1:Thepossiblebinarysearchtreesfortheidentifierset(a1,a2,a3)=(do,if, stop) are as follows. Given the equal probabilitiesp(i)=Q(i)=1/7foralli, wehave:



Tree1



Tree2



Tree3

Cost(tree#1)= $(1x1+1x2+1x3)/7 + (1x1)/7$

1+2+31+2+3+36+915

Cost(tree#3)=~1x1+1x2+1x3~~+(~1x1+1x2+1x3+1x3~~

~7 7 7) ~7 7 7 7)

= 1+2+3 + 1+2+3+3 ~ 6+9 15
7 7 (1x1+ 1

Cost(tree#4)=~1 x1+1x2~1x3~~~ ~ x2 +1 x3+1x3~~

~7 7 7) ~7 7 7 7)

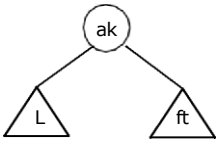
=1+2+3 1+2+3+3 6+9 15
7

Cost(tree#2)= (1x1+1 1x2~ (~1x2+ 1x2+1 x2+1x2~
~7 7 x2+7) +~7 7 7 7)

=1+2+2 + 2+2+2+2 ~ 5+8 13
7 7 7~ 7

Huffman coding tree solved bya greedy algorithm has a limitation of having the data only attheleavesanditmustnotpreservethepropertythatallnodestotheleftoftheroot have keys, which are less etc. Construction of an optimal binary search tree is harder, because the data is not constrained to appear only at the leaves, and also because the tree mustsatisfythebinarysearchtreepropertyanditmustpreservethe property that all nodes to the left of the root have keys, which are less.

A dynamic programming solution to the problem of obtaining an optimal binarysearch tree can be viewed by constructing a tree as a result of sequence of decisionsby holding theprincipleofoptimality.Apossibleapproachtothisistomakeadecisionaswhich of the ai's be arraigned to the root node at 'T'. If we choose 'ak' then is clear that theinternalnodesfora1,a2,.....ak-1aswellastheexternalnodesfortheclassesEo,E1, .. Ek-1willlieintheleftsubtree,L,oftheroot.Theremainingnodeswillbein therightsubtree,ft.Thestructureofanoptimalbinarysearchtreeis:



Cost(L)= $\sum_{i=1}^{K} P(i)*level(a_i)+\sum_{i=0}^{K} Q(i)*level(E_i)-1)$

Cost(ft)= $\sum_{i=K}^{n} P(i)*level(a_i)+\sum_{i=K}^{n} Q(i)*level\ (E_i)-1)$

TheC (i, J) can becomputedas:

C(i, J) =min {C(i, k-1)+C (k, J)+P (K)+w(i,K-1)+w(K, J)}
i<k<J

=min {C (i, K-1)+C (K, J)} +w (i, J) -- (1)
i<k<J

WhereW(i, J) =P (J) +Q (J) +w(i, J-1) -- (2)

InitiallyC(i,i)=0andw(i,i)=Q(i)for0≤i≤n.

Equation (1) maybe solved for C (0, n) byfirst computing all C (i, J) such that J -i = 1 Next, we can compute all C (i, J) such that J - i = 2, Then all C (i, J) with J - i = 3 andsoon.
C(i,J)isthecostoftheoptimalbinarysearchtree'Tij'duringcomputationwerecord the root R (i, J) of each tree'Tij'. Then an optimal binary search tree may be constructedfromtheseR(i,J).R(i,J)isthevalueof'K'thatminimizesequation(1).

WesolvetheproblembyknowingW(i,i+1),C(i,i+1)andR(i,i+1),0 ≤ i<4;
KnowingW(i,i+2),C(i,i+2)andR(i,i+2),0≤i<3andrepeatinguntilW(0,n), C(0,n)andR(0,n)areobtained.

Theresultsaretabulatedtorecovertheactualtree.

Example1:

Letn=4,and(a1,a2,a3,a4)=(do,if,need,while)LetP(1:4)=(3,3,1,1)andQ(0: 4)=(2,3,1,1,1)

Solution:

TableforrecordingW(i,j),C(i,j)andR(i,j):

Column Row	**0**	**1**	**2**	**3**	**4**
0	2,0,0	3,0,0	1, 0, 0	1, 0, 0,	1, 0, 0
1	8,8,1	7,7,2	3, 3, 3	3, 3, 4	
2	12,19, 1	9,12,2	5, 8, 3		
3	14,25, 2	11,19, 2			
4	16,32, 2				

Thiscomputationiscarriedoutrow-wisefromrow0torow4.Initially,W(i,i)=Q (i)andC(i,i)=0andR(i,i)=0,0≤i<4. Solving for

C (0, n):

First,computingallC(i,j)suchthatj-i=1;j=i+1andas0≤i<4;i=0,1,2and 3; i< k ≤ J. Start with i = 0; so j = 1; as i< k ≤j, so the possible value fork = 1

W(0,1)=P(1)+Q(1)+W(0,0)=3+3+2=8
C(0,1)=W(0,1)+min{C(0,0)+C(1,1)}=8
R(0,1)=1(valueof'K'thatisminimumintheaboveequation).

Nextwithi=1;soj=2;asi<k≤j,sothepossiblevaluefork=2

W(1, 2) =P(2)+Q(2)+W(1, 1) =3+1 +3=7
C(1, 2) =W(1,2)+min{C(1, 1) +C(2, 2)}=7
R(1, 2) =2

Nextwithi=2;soj=3;asi<k≤j,sothepossiblevaluefork=3

W(2,3) =P(3)+Q(3)+W(2, 2) =1+1 +1 = 3

C(2,3) =W(2,3)+min{C(2, 2) +C(3, 3)}=3 +[(0 +0)] =3
ft(2,3) =3

Nextwithi=3;soj=4;asi<k≤j,sothepossiblevaluefork=4

W(3,4) =P(4)+Q(4)+W(3,3) =1+1 +1 =3
C(3,4) =W(3,4)+min{[C(3,3) +C(4, 4)]} =3 +[(0 +0)] =3
ft(3,4) =4

Second,ComputingallC(i,j)suchthatj-i=2;j=i+2andas0≤i<3;i=0,1,2;i<k≤J.Startwithi=0;soj=2;asi<k≤J,sothepossiblevaluesfor k=1 and 2.

W(0,2)=P(2)+Q(2)+W(0,1)=3+1+8=12
C(0,2)=W(0,2)+min{(C(0,0)+C(1,2)),(C(0,1)+C(2,2))}=12
+min{(0+7,8+0)}=19
ft(0,2)=1
Next,withi=1;soj=3;asi<k≤j,sothepossiblevaluefork=2and3.

W(1, 3) =P(3) +Q(3)+W(1,2)=1+1+7=9
C(1, 3) =W(1, 3)+min{[C(1,1)+C(2,3)],[C(1,3) 2) +C(3, 3)]}
=W(1, +min{(0+3),(7+0)}=9+3= 12
ft(1,3)=2

Next,withi=2;soj=4;asi<k≤j,sothepossiblevaluefork=3and4.

W(2,4)=P(4)+Q(4)+W(2,3)=1+1+3=5
C(2,4)=W(2,4)+min{[C(2,2)+C(3,4)],[C(2,3)+C(4,4)]
=5+min{(0+3),(3+0)}=5+3=8
ft(2, 4)=3

Third,ComputingallC(i,j)suchthatJ-i=3;j=i+3andas0≤i<2;i=0,1;i< k ≤J.Startwithi=0;soj=3;asi<k≤j,sothepossiblevaluesfork=1,2and3.

W(0,3)=P(3)+Q(3)+W(0, 2) = 1+ 1=+12=14
C(0,3) W(0,3)+min{[C(0,0)+C(1, 3)],[C(0, 1)+C(2,3)], [C(0,2)+C(3,=3)]} +11=25
+0)}=14
ft(0,3) 14+min{(0+12),(8+3),(19=2

Startwithi=1;soj=4;asi<k≤j,sothepossiblevaluesfork=2,3and4.

W(1,4) =P(4)+Q(4)+W(1,3)=1+1+9=11=W
C(1,4) (1,4)+min{[C(1,1)+C(2,4)],[C(1, 2) +C(3, 4)],
[C(1,3)+C(4,4)]} +8 =19
ft(1,4) =11+min{(0+8),(7+3),(12+0)}=11=2

Fourth,ComputingallC(i,j)suchthatj-i=4;j=i+4andas0≤i<1;i=0;i<k≤ J.

Startwithi=0;soj=4;asi<k≤j,sothepossiblevaluesfork=1,2,3and4.

W(0, 4) =P(4) +Q(4)+W(0, 3) =1+1 +14=16

C(0, 4) =W(0, 4)+min{[C(0, 0) +C(1, 4)],[C (0, 1) +C(2, 4)],
[C (0, 2) +C(3, 4)],[C (0, 3) +C(4, 4)]}
=16+min[0+19,8+8,19+3,25+0]=16+16=32ft(0,
4)=2

FromthetableweseethatC(0,4)=32istheminimumcostofabinarysearchtreefor (a1,a2,a3,a4).Therootofthetree'T04'is'a2'.

Hencetheleftsubtreeis'T01'andrightsubtreeisT24.Therootof'T01'is'a1'andthe root of 'T24' is a3.

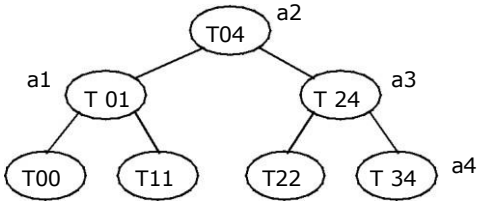
Theleftandrightsubtreesfor'T01'are'T00'and'T11'respectively.TherootofT01is 'a1'

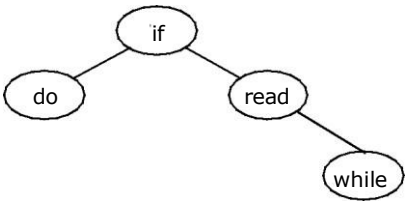
TheleftandrightsubtreesforT24areT22andT34respectively. The root

of T24 is 'a3'.

TherootofT22isnull

TherootofT34isa4.





Example2:

Consider four elements a1, a2, a3 and a4 with Q0 = 1/8, Q1 = 3/16, Q2 = Q3 = Q4 = 1/16andp1=1/4,p2=1/8,p3=p4=1/16.Constructanoptimalbinarysearchtree.Solving for C (0, n):

First,computingallC(i,j)suchthatj-i=1;j=i+1andas0≤i<4;i=0,1,2and3;i < k ≤ J. Start withi = 0; so j = 1;as i< k ≤ j,so the possible value for k =1

W(0,1)=P(1)+Q(1)+W(0,0)=4+3+2=9

C(0,1)=W(0,1)+min{C(0,0)+C(1,1)}=9+[(0+0)]=9ft(0,
1)=1(valueof'K'thatisminimumintheaboveequation).

Nextwithi=1;soj=2;asi<k≤j,sothepossiblevaluefork=2

W(1,2)=P(2)+Q(2)+W(1,1)=2+1+3=6
C(1,2)=W(1,2)+min{C(1,1)+C(2,2)}=6+[(0+0)]=6ft(1,
2)=2

Nextwithi=2;soj=3;asi<k≤j,sothepossiblevaluefork=3

W(2, 3) =P(3)+Q(3)+W(2, 2) =1+1 +1= 3
C(2, 3) =W(2,3)+min{C(2, 2) +C(3, 3)}=3+[(0+0)]=3

ft(2,3)=3

Nextwithi=3;soj=4;asi<k≤j,sothepossiblevaluefork=4

W(3,4) =P(4)+Q(4)+W(3,3) =1+1 +1 =3

C(3,4) =W(3,4)+min{[C(3,3) +C(4, 4)]} =3 +[(0 +0)] =3

ft(3,4) =4

Second,ComputingallC(i,j)suchthatj-i=2;j=i+2andas0≤i<3;i=0,1,2;i<k ≤ J

Startwithi=0;soj=2;asi<k≤j,sothepossiblevaluesfork=1and2.

W(0,2)=P(2)+Q(2)+W(0,1)=2+1+9=12
C(0,2)=W(0,2)+min{(C(0,0)+C(1,2)),(C(0,1)+C(2,2))}=12 +
min{(0+6,9+0)}=12+6=18
ft(0,2)=1
Next,withi=1;soj=3;asi<k≤j,sothepossiblevaluefork=2and3.
W(1, 3) =P(3) +Q(3)+W(1,2)=1+1+6=8
C(1, 3) =W(1, 3)+min{[C(1,1)+C(2,3)],[C(1, 2) +C(3, 3)]}
=W(1, 3)+min{(0+3),(6+0)}=8+3= 11
ft(1,3)=2

Next,withi=2;soj=4;asi<k≤j,sothepossiblevaluefork=3and4.

W(2,4)=P(4)+Q(4)+W(2,3)=1+1+3=5
C(2,4)=W(2,4)+min{[C(2,2)+C(3,4)],[C(2,3)+C(4,4)]
=5+min{(0+3),(3+0)}=5+3=8
ft(2, 4)=3

Third,ComputingallC(i,j)suchthatJ-i=3;j=i+3andas0≤i<2;i=0,1;i<k≤
J.Startwithi=0;soj=3;asi<k≤j,sothepossiblevaluesfork=1,2and3.

W(0,3)=P(3)+Q(3)+W(0,2)=1+1+12=14
C(0,3)=W(0,3)+min{[C(0,0)+C(1,3)],[C(0,1)+C(2,3)],[C(0,
2)+C(3,3)]}
=14+min{(0+11),(9+3),(18+0)}=14+11=25ft(0,
3)=1

Startwithi=1;soj=4;asi<k≤j,sothepossiblevaluesfork=2,3and4.

W(1,4) =P(4)+Q(4)+W(1,3)=1+1+8=10=W
C(1,4) (1,4)+min{[C(1,1)+C(2,4)],[C(1, 2) +C(3, 4)],
[C(1,3)+C(4,4)]} +8 =18
=10+min{(0+8),(6+3),(11+0)}=10=2
ft(1,4)

Fourth,ComputingallC(i,j)suchthatJ-i=4;j=i+4andas0≤i<1;i=0;
i<k≤J.Startwithi=0;soj=4;asi<k≤j,sothepossiblevaluesfork=1,2,3and 4.

W(0, 4) =P(4) +Q(4)+W(0, 3) =1+1 +14=16

C(0, 4) =W(0, 4)+min{[C(0, 0) +C(1, 4)],[C (0, 1) +C(2, 4)],
[C (0, 2) +C(3, 4)],[C (0, 3) +C(4, 4)]}

=16+min[0+18,9+8,18+3,25+0]=16+17=33R(0,4)
=2

TableforrecordingW(i,j),C(i,j)andR(i,j)

Column Row	**0**	**1**	**2**	**3**	**4**
0	2,0,0	1,0,0	1, 0, 0	1, 0, 0,	1, 0, 0
1	9,9,1	6,6,2	3, 3, 3	3, 3, 4	
2	12,18, 1	8,11,2	5, 8, 3		
3	14,25, 2	11,18, 2			
4	16,33, 2				

FromthetableweseethatC(0,4)=33istheminimumcostofabinarysearchtreefor (a1, a2, a3, a4)

Therootofthetree'T04'is'a2'.

Hencetheleftsubtreeis'T01'andrightsubtreeisT24.Therootof'T01'is'a1'andthe root of 'T24' is a3.

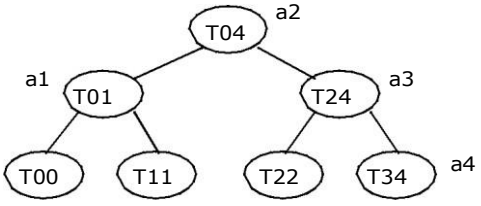
Theleftandrightsubtreesfor'T01'are'T00'and'T11'respectively.TherootofT01is 'a1'

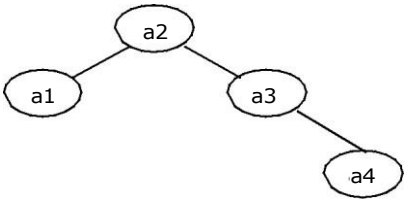
TheleftandrightsubtreesforT24areT22andT34respectively. The root

of T24 is 'a3'.

TherootofT22isnull.

The root of T34 is a4.





0/1–KNAPSACK

Weare givennobjectsanda knapsack. Eachobjectihasapositiveweightwianda positive valueVi.TheknapsackcancarryaweightnotexceedingW.Filltheknapsacksothatthe valueofobjectsintheknapsackisoptimized.

A solution to the knapsack problem can be obtained by making a sequence ofdecisions on the variables x1, x2, ... , xn. A decision on variable xi involves determining which of the values 0 or 1 is to be assigned to it. Let us assume that

decisions onthexiaremadeintheorder xn,xn-1, ... x1.Followinga decisiononxn, we may be in one of two possible states: thecapacityremaininginm –wnanda profit ofpnhasaccrued.Itisclearthattheremainingdecisionsxn-1, ... , x1mustbeoptimal withrespecttotheproblemstateresultingfromthedecisiononxn.Otherwise,xn,.. ..,x1willnotbeoptimal.Hence,theprincipalofoptimalityholds.

Fn(m)=max{fn-1(m),fn-1(m-wn)+pn} -- 1

Forarbitraryfi(y),i>0,thisequationgeneralizesto:

Fi(y)=max{fi-1(y),fi-1(y-wi)+pi} – 2

Equation-2canbesolvedforfn(m)bybeginningwiththeknowledgefo(y)=0forally and fi (y) = - ~, y< 0. Then f1, f2, . . . fn can be successively computed using equation–2.

Whenthewi'sareinteger,weneedtocomputefi(y)forintegery,0≤y≤m.Sincefi(y) = - ~ for y < 0, these function values need not be computed explicitly. Sinceeach fi can be computed from fi- 1 in **Θ** (m) time, it takes **Θ** (m n) time tocomputefn.When thewi'sarerealnumbers,fi(y)isneededforrealnumbersysuchthat0<y≤m.So,fi cannotbeexplicitlycomputedforallyinthisrange.Evenwhenthewi'sareinteger,the explicit **Θ** (m n) computation of fn may not be the most efficient computation. So, we explore**analternativemethodforbothcases.**

Thefi(y)isanascendingstepfunction;i.e.,thereareafinitenumberofy's,0=y1<y2 <....<yk,suchthatfi(y1)<fi(y2)<.....<fi(yk);fi(y)=-~,y<y1;fi(y)=f (yk),y≥yk;andfi(y)=fi(yj),yj≤y≤yj+1.So,weneedtocomputeonlyfi(yj),1≤j ≤k.WeusetheorderedsetS^{i}={(f(yj),yj)| 1≤j≤k}torepresentfi(y).Eachnumber of S^{i} is a pair (P, W), where P = fi (yj) and W = yj. Notice that S^{0} = {(0, 0)}. We can computeS^{i+1}fromSibyfirstcomputing:

Si1={(P,W)|(P–pi,W–wi)eS^{i}}

Now,S^{i+1}canbecomputedbymergingthepairsinS^{i}andSi1together.NotethatifSi+1 contains twopairs (Pj,Wj) and(Pk,Wk) withthe property that Pj≤PkandWj>Wk, then the pair (Pj, Wj) can be discarded because of equation-2. Discarding or purging rulessuchasthisonearealsoknownasdominancerules.Dominatedtuplesgetpurged.In theabove,(Pk,Wk)dominates(Pj,Wj).

Reliability Design

The problem is to design a system that is composed of several devices connected in series. LetribethereliabilityofdeviceDi(thatisriistheprobabilitythatdeviceiwill function properly) then the reliability of the entire system is fTri. Even if the individual devices are very reliable (the ri's are very close to one), the reliability ofthe system may not be very good. For example, if n = 10 and ri = 0.99, i≤i≤ 10, then fTri= .904. Hence, it is desirable to duplicate devices. Multiply copies of the same device type are connected in parallel.

If stage i contains mi copies of device Di. Then the probability that all mi have a malfunctionis (1 -ri)mi. Hencethereliabilityofstagei becomes1 – (1 -r)^{mi}.

i

Thereliabilityofstage'i' isgivenbyafunction ~i(mi).

Our problem is to use device duplication. This maximization is to be carried out under a costconstraint.Letcibethecostofeachunitofdeviceiandletcbethemaximumallowable cost of the system being designed.

Wewishtosolve:

Maximize~*qi*(*mi*~
1<*i*<*n*

Subjectto~*Cimi*<*C*
1≤*i*≤*n*

mi≥1andinterger,1≤i≤n

AssumeeachCi>0,eachmimustbeintherange1 ≤mi≤ui,where

$$ui \sim\sim\sim C + Ci \sum_{1 \le k \le n} C \Big/ Ci$$

Theupperbounduifollowsfromtheobservationthatmj≥1 An

optimal solution m1, m2 ..... mn is the resultofasequenceofdecisions,onedecis ion for each mi.

Letfi(x)representthemaximumvalueofSu $\sum_{1 \le i < j} q\$m_J$

bject to the constrains:

$$\sum_{1 \le i < j} C_J m_J \sim x \quad \text{and} \quad 1 \le m_j \le u_J, 1 \le j \le i$$

Thelastdecisionmaderequiresonetochoosemnfrom {1,2, 3, ... un}

Onceavalueofmnhasbeenchosen,theremainingdecisionsmustbesuchastousethe remaining funds C – Cn mn in an optimal way.

Theprincipleofoptimalityholds on

$$f_n(C) = \max\{O_n(m_n) f_{n-1}(C - C_n m_n)\} \quad 1 \le m_n \le u_n$$

for anyfi(xi),i>1,thisequationgeneralizesto

$$f_n(x) = \max\{c_i(m_i) f_i(x - C_i m_i)\} \quad 1 \le m_i \le u_i$$

clearly,f0(x)=1forallx,0≤x≤Candf(x)=-ooforallx<0.Let

S^iconsistoftuplesoftheform(f,x),wheref=fi(x).

Thereisatmostonetupleforeachdifferent‘x’,thatresultfromasequenceof decisionsonm1, m2,..... mn.Thedominancerule(f1,x1)dominate(f2,x2)iff1≥f2 andx1≤x2.Hence,dominatedtuplescanbediscardedfromS^i.

Example1:

DesignathreestagesystemwithdevicetypesD1,D2andD3.Thecostsare$30,$15and $20respectively.TheCostofthesystemistobenomorethan$105.Thereliabilityof eachdeviceis0.9,0.8and0.5respectively.

Solution:

Weassumethatififstage Ihasmidevicesoftypeiinparallel,then$\emptyset_i(m_i) = 1 - (1 - r_i)^{m_i}$

Since,wecanassumeeachci>0,eachmimustbeintherange1≤mi≤ui.Where:

$$u_i = \left\lfloor C + C_i - \sum_{j=1}^{n} C_j \Big/ C_i \right\rfloor$$

Usingtheaboveequationcomputeu1,u2andu3.

$$u1 = \frac{105+30-(30+15+20)}{30} = \frac{70}{30} = 2$$

$$u2 = \frac{105+15-(30+15+20)}{15} = \frac{55}{15} = 3$$

$$u3 = \frac{105+20-(30+15+20)}{20} = \frac{60}{20} = 3$$

WeuseS-*i*:*stagenumberandJ*:*no.ofdevicesinstagei*=miS°

$=\{f_o(x),x\}$ *initially* $f_o(x)=1$ *and* $x=0$, *so*, $S^o=\{1,0\}$

ComputeS^1,S^2andS^3asfollows:

S1=dependsonu1value,asu1=2,so

$S1=\{S1,S^1\}$
1 2

S2=dependsonu2value,asu2=3,so

$_S2=\{S^2,S^2,S^2\}$
1 2 3

S3=dependsonu3value,asu3=3,so

$S3=\{S^3,S^3,S^3\}$
1 2 3

Nowfind ,1 S (*x*), *x*

$f1(x)=\{01(1)f_o\sim\sim,01(2)f0()\}$Withdevicesm1=1andm2=2Compute$\emptyset$1(1)

and$\emptyset$1(2)usingtheformula:$\emptyset i(mi))=1-(1-ri)mi$

~~1~~1~~1~r~m1 $=1-(1-0.9)^1=0.9$

1 1~(2)=1-(1-0.9)2 $=0.99$

S ~~f1~x~,x~~
1 ~ ~0.9,30

$S2$

1=10.99,30+30}=(0.99,60

Therefore,$S^1=\{(0.9,30),(0.99,60)\}$

Nextfind2~~~2$f(x),x$~~
S
1

$f2(x)=\{02(1)*f1(),02(2)*f1(),02(3)*f1()\}$

~2~1~~1~~1~*rI*~=*mi*1–(1–0.8)=1–0.2=0.8

~2~2~~1~~1~0.8~2=0.96

*0*2(3)=1-(1-0.8)3=0.992

={(0.8(0.9),30+15),(0.8(0.99),60+15)}={(0.72,45),(0.792,75)}=
{(0.96(0.9),30+15+15),(0.96(0.99),60+15+15)}
={(0.864,60),(0.9504,90)}

={(0.992(0.9),30+15+15+15) ,(0.992(0.99),60+15+15+15)}
={(0.8928,75),(0.98208,105)}
*S*2={S^2,S^2,S^2}
1 2 3

ByapplyingDominanceruletoS^2:

Therefore,S2={(0.72,45),(0.864,60),(0.8928,75)}DominanceRule:

If S^i contains two pairs (f1, x1) and (f2, x2) with the property that f1 ≥ f2 and x1 ≤ x2, then (f1, x1) dominates (f2, x2), hence by dominance rule (f2, x2) can be discarded. Discarding or pruning rules such as the one above is known as dominance rule. Dominatingtuples will be present inS^{i}andDominated tuples has to be discarded from Si.

Case1:iff1≤f2andx1>x2thendiscard(f1,x1)

Case2:iff1≥f2andx1<x2thediscard(f2,x2)

Case3:otherwisesimplywrite(f1,x1)

S2={(0.72,45),(0.864,60),(0.8928,75)}

*Ø*3(1)=1~~1_*rI*~*mi*=1–(1–0.5)^1=1–0.5=0.5

*Ø*2~2~~1~~1=0.75
~*S* 0.5~2
3 =0.875
ØS^2 ~3~~1~~1
~ 0.5~3
3*S*3 ~
2

*S*13 ={(0.5(0.72),45+20),(0.5(0.864),60+20),(0.5(0.8928),75+20)}

*S*13={(0.36,65),(0.437,80),(0.4464,95)}

S_2^3 ={(0.75 (0.72), 45 + 20 + 20), (0.75 (0.864), 60 + 20 +20),
(0.75(0.8928),75+20+20)}

={(0.54,85),(0.648,100),(0.6696,115)}

S_3^3 = { □0.875(0.72),45+20+20+20), □0.875(0.864),60+ 20+20+20, □0.875(0.8928),75+20+20+20 □ }

S_3^3={(0.63,105),(1.756,120),(0.7812,135)}

Ifcostexceeds105,removethattuples

S3={(0.36,65),(0.437,80),(0.54,85),(0.648,100)}

Thebestdesignhasareliabilityof0.648andacostof100.Tracingbackforthesolution throughS^i'swecandeterminethatm3=2,m2=2andm1=1.

OtherSolution:

Accordingtotheprincipleofoptimality:

fn(C) =max{~n(mn). fn-1(C -Cnmn)withfo(x) =1and 0≤x ≤C;1~ *mn*<*un*

Since,wecanassumeeachci>0,eachmimustbeintherange1≤mi≤ui.Where:

S2={(0.75(0.72),45 +20+20),(0.75(0.864),60+ ~ (n ~ ~ 20+20), *u*$_i$

=~i*C*+*Ci*_~*CJ*r/*Ci*I~

~ *i* ~ ~

Usingtheaboveequationcomputeu1,u2andu3.

$u1 = \frac{105 - 30}{30} \sim \frac{70}{30} = 2$

$u2 = \frac{105 - 15 - 30}{15} \sim \frac{55}{15} \sim 3$

$u3 = \frac{105 - 15 - 30 + 20}{20} = \frac{60}{20} = 3$

f3(105)=max{~3(m3).f2(105-20m3)}1<*m*3!*u*3

=max{3(1)f2(105-20),63(2)f2(105-20x2),~3(3)f2(105-20x3)}=max{0.5 f2(85),0.75f2(65),0.875f2(45)}

=max{0.5x0.8928,0.75x0.864,0.875x0.72}=0.648.

=max{2(m2).f1(85 -15m2)}
1!*m*2!*u*2

f2 (85) =max{2(1).f1(85-15),~2(2).f1(85-15x2),~2(3).f1(85-15x3)} = max{0.8f1(70),0.96f1(55),0.992f1(40)}

=max{0.8x0.99,0.96x0.9,0.99x0.9}=0.8928

f1 (70) =max{~1(m1).f0(70- 30m1)}

1!*m*1!*u*1

=max{~1(1)f0(70-30),t1(2)f0(70-30x2)}

=max{~1(1)x1,$_{t1(2)}$x1}=max{0.9,0.99}=0.99

f1 (55) =max{t1(m1).f0(55-30m1)}

1!*m*1!*u*1

=max{~1(1)f0(50-30),t1(2)f0(50-30x2)}

=max{~1(1)x1,$_{t1(2)}$x-oo}=max{0.9,-oo}=0.9

f1 (40) =max{~1(m1).f0(40-30m1)}

1!*m*1!*u*1

=max{~1(1)f0(40-30),t1(2)f0(40-30x2)}

=max{~1(1)x1,$_{t1(2)}$x-oo}=max{0.9,-oo}=0.9

f2 (65)=max{2(m2).f1(65-15m2)} 1 !

*m*2 ! *u*2

=max{2(1)f1(65-15),62(2)f1(65-15x2),~2(3)f1(65-15x3)}=max{0.8f1(50),

0.96f1(35),0.992f1(20)}

=max{0.8x0.9,0.96x0.9,-oo}=0.864

f1(50) = max{~1(m1).f0(50 - 30m1)} 1

! *m*1 ! *u*1

=max{~1(1)f0(50-30),t1(2)f0(50-30x2)}

=max{~1(1)x1,$_{t1(2)}$x-oo}=max{0.9,-oo}=0.9f1(35)=max

~1(m1).f0(35-30m1)}

1!*m*1!*u*1

=max{~1(1).f0(35-30),~1(2).f0(35-30x2)}

=max{~1(1)x1,$_{t1(2)}$x-oo}=max{0.9,-oo}=0.9

f1(20)=max{~1(m1).f0(20-30m1)} 1 !

*m*1 ! *u*1

=max{~1(1)f0(20-30),t1(2)f0(20-30x2)}

=max{~1(1)x-,~1(2)x-oo}=max{-oo,-oo}=-oof2

(45) = max {2(m2). f1(45 -15m2)}

1!*m*2!*u*2

=max{2(1)f1(45-15),~2(2)f1(45-15x2),~2(3)f1(45-15x3)}=max{0.8f1(30),

0.96f1(15),0.992f1(0)}

=max{0.8x0.9,0.96x-,0.99x-oo}=0.72

f1(30)=max{~1(m1). f0(30-30m1)} 1<*m*1~*u*1

=max{~1(1)f0(30-30),t1(2)f0(30-30x2)}

=max{~1(1)x1,$_{t1(2)}$x-oo}=max{0.9,-oo}=0.9Similarly,f1(15)=-, f1 (0) =

-.

Thebestdesignhasareliability=0.648and

Cost=30x1+15x2+20x2=100.

TracingbackforthesolutionthroughS^{i}‘swecandeterminethat:m3 = 2,m2 =2 and m1 =

1.

UNITIV:

Backtracking: Generalmethod,applications-n-queenproblem,sumofsubsetsproblem,graph coloring, Hamiltonian cycles.

BranchandBound: Generalmethod,applications-Travellingsalespersonproblem,0/1 knapsack problem- LC Branch and Bound solution, FIFO Branch and Bound solution.

Backtracking(Generalmethod)

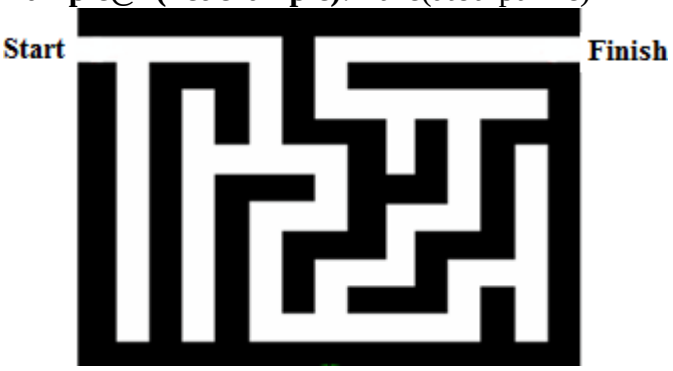
Manyproblemsaredifficulttosolvealgorithmically.Backtrackingmakes itpossibletosolveat least some large instances of difficult combinatorial problems.

Supposeyouhavetomakeaseriesofdecisionsamongvariouschoices, where

- Youdon'thaveenoughinformationtoknowwhattochoose
- Eachdecisionleadstoanewsetofchoices.
- Somesequenceof choices(morethan one choices)maybeasolutiontoyourproblem.

Backtrackingisamethodical(Logical)wayoftryingoutvarioussequencesofdecisions,until you find one that "works"

Example@1(net example):Maze(atourpuzzle)



Givenamaze, findapathfromstarttofinish.

- Inmaze,ateachintersection, youhavetodecidebetween3or fewerchoices:
  - ✓ Go straight
  - ✓ Go left
  - ✓ Go right
- Youdon'thaveenoughinformationtochoosecorrectly
- Eachchoiceleadstoanothersetofchoices.
- Oneormoresequencesof choices mayor maynotlead to asolution.
- Manytypes of mazeproblem can besolved with backtracking.

Example@2(text book):

Sortingthearrayofintegersina[1:n]isaproblemwhosesolutionisexpressiblebyann-tuple x_i→ is the index in 'a' of the i^{th}smallest element.

Thecriterionfunction'P'istheinequalitya[x_i]≤a[x_{i+1}]for1≤i≤n S_i→ is finite and includes the integers 1 through n.

m_i→sizeofsetS_i

$m=m_1m_2m_3---m_n$ntuples that possiblecandidatesforsatisfyingthe function P.

Withbruteforceapproachwouldbetoformallthesen-tuples,evaluate(judge)eachonewithP and save those which yield the optimum.

By using backtrack algorithm; yield the same answer with far fewer than 'm' trails.Manyoftheproblemswesolveusingbacktrackingrequiresthatallthesolutionssatisfya complex set of constraints.

Foranyproblemtheseconstraints can bedivided into two categories:

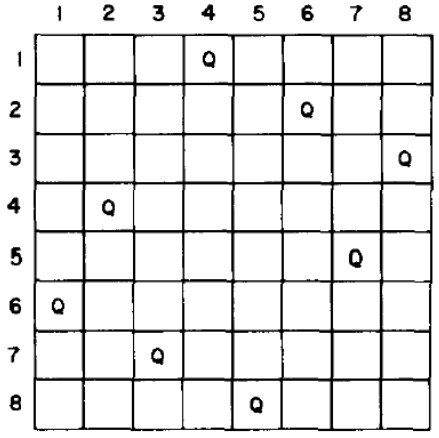
- Explicitconstraints.
- Implicitconstraints.

Explicitconstraints: Explicitconstraintsarerulesthatrestricteach$x_i$totakeonvaluesonly from a given set.
Example:$x_i \geq 0$orsi={allnonnegativerealnumbers}
X_i=0or1orS_i={0,1}
$l_i \leq x_i \leq u_i$ors_i={a:$l_i \leq a \leq u_i$}
Theexplicitconstraintdependsontheparticularinstance I oftheproblembeingsolved.All tuples that satisfy the explicit constraints define a possible solution space for I.
ImplicitConstraints:
Theimplicitconstraintsarerulesthatdeterminewhichofthetuplesinthesolutionspaceof I satisfythecriterionfunction.Thusimplicitconstraintsdescribethewayinwhichthe $X_i$must relate to each other.
ApplicationsofBacktracking:

- NQueensProblem
- Sumof subsets problem
- Graphcoloring
- Hamiltonian cycles.

N-QueensProblem:

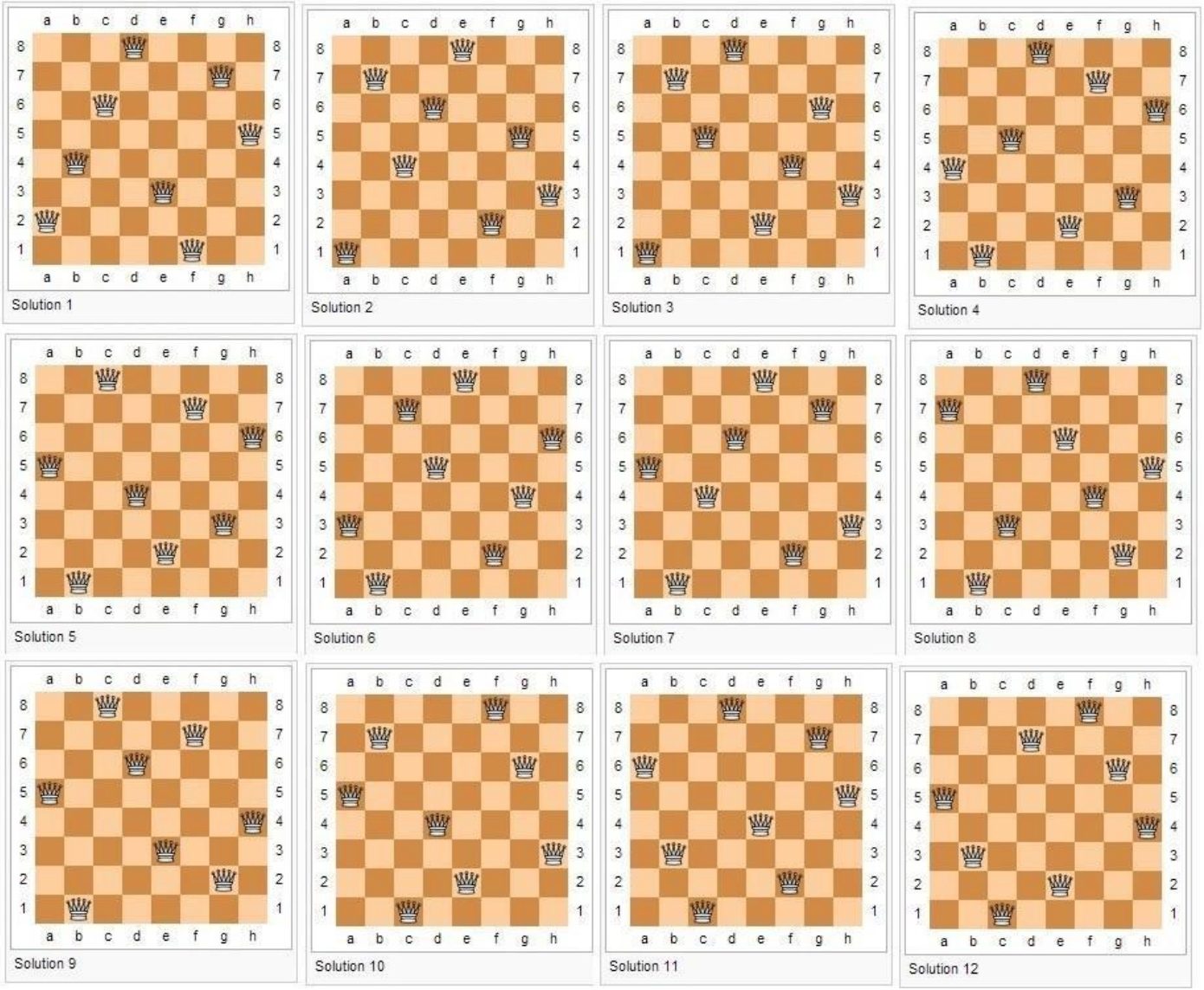
Itisaclassiccombinatorialproblem.Theeightqueen'spuzzleistheproblemofplacingeight queens puzzle is the problem of placing eight queens on an 8×8 chessboard so that no two queens attack each other. That is so that no two of them are on the same row, column, or diagonal.
The8-queenspuzzleisanexampleofthemoregeneraln-queensproblemofplacingnqueenson an n×n chessboard.



One solution to the 8-queens problem

Herequeenscanalsobenumbered1through8
Each queen must be on a different row
Assume queen 'i' is to be placed on row 'i'
Allsolutionstothe8-queensproblemcanthereforeberepresentedass-tuples(x_1,x_2,x_3—x_8) xi→ the column on which queen 'i' is placed
si→{1, 2, 3, 4, 5, 6, 7, 8}, $1 \leq i \leq 8$
Thereforethesolution space consistsof8^8 s-tuples.
Theimplicitconstraintsforthisproblemarethatnotwox_i'scanbethesamecolumnandnotwo queens can be on the same diagonal.
Bythesetwoconstraintsthesizeofsolutionpacereducesfrom88tuplesto8!Tuples. Form example s_i(4,6,8,2,7,1,3,5)

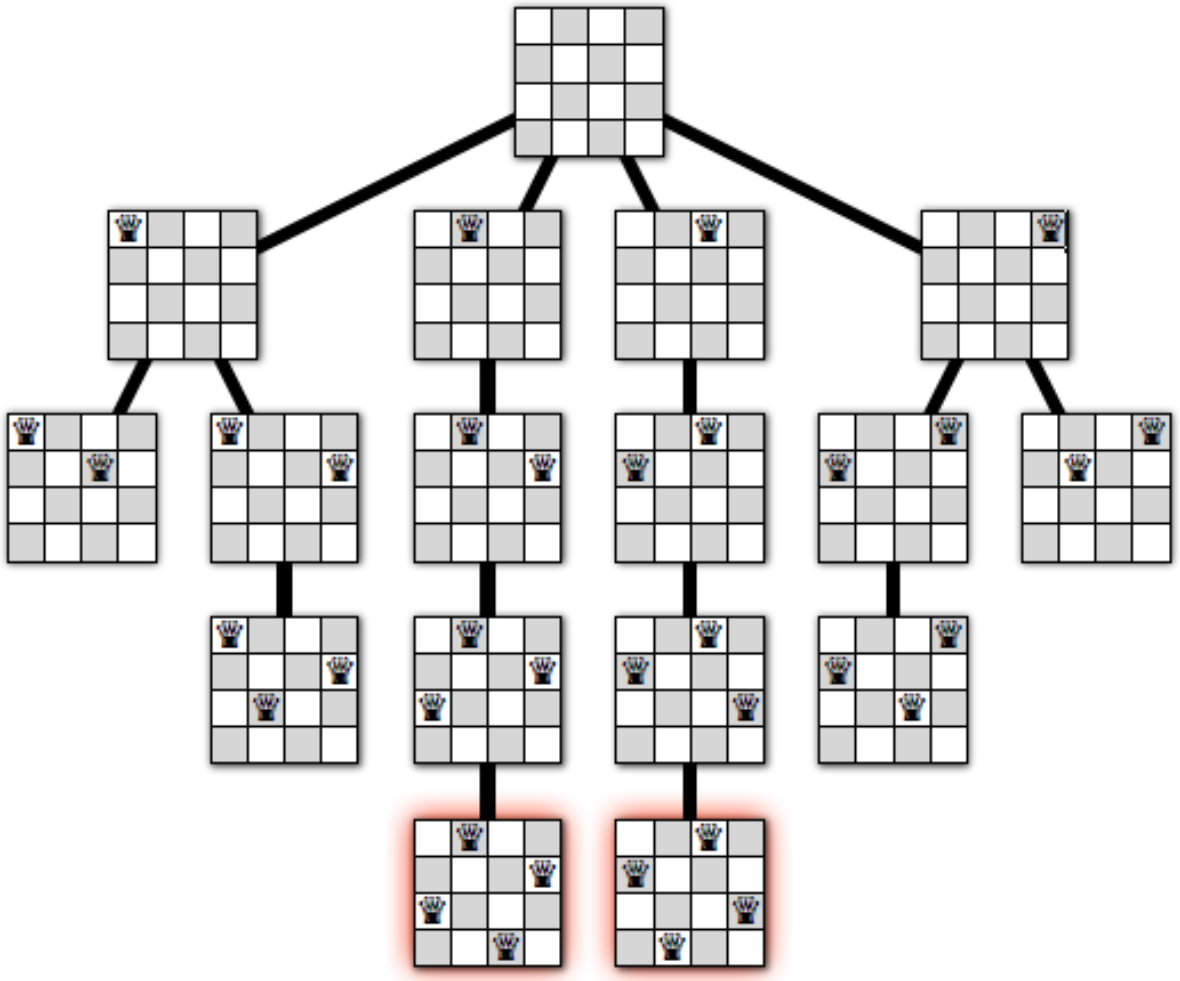
Inthesamewayforn-queensaretobeplacedonann×nchessboard,thesolutionspaceconsists of all n! Permutations of n-tuples (1,2, ----------n).



Somesolutiontothe8-Queens problem

Algorithmfornewqueenbeplaced	**Allsolutionstothen·queensproblem**

AlgorithmPlace(k,i)
//Returntrueifaqueencanbeplacedinkth row &ith column
//Otherwisereturnfalse
{
forj:=1 tok-1 do
if(x[j]=iorAbs(x[j]-i)=Abs(j-k))) then
return false
returntrue
}
	AlgorithmNQueens(k,n)
	//itsprintsallpossibleplacementsofn- queens on an n×n chessboard.
	{
	fori:=1tondo{
	ifPlace(k,i)then
	{
	X[k]:=I;
	if(k==n)thenwrite(x[1:n]);
	else NQueens(k+1, n);
	}
	}}



The complete recursion tree for our algorithm for the 4 queens problem.

SumofSubsetsProblem:

Givenpositivenumbersw_i1≤i≤n,&m,heresumofsubsetsproblemisfindingallsubsetsof w_i whose sums are m.

Definition:Givenndistinct+venumbers(usuallycalledweights),desire(want)tofindall combinations of these numbers whose sums are m. this is called sum of subsets problem.
To formulate this problem by using either fixed sized tuples or variable sized tuples.
Backtrackingsolutionusesthefixedsizetuplestrategy.

For example:
If n=4(w_1,w_2, w_3,w_4)=(11,13,24,7)and m=31.
Thendesiredsubsets are (11, 13,7)&(24, 7).
Thetwo solutions aredescribedbythe vectors (1,2, 4)and (3, 4).

In generalallsolutionarek-tuples(x_1,x_2,x_3---x_k)1≤k≤n,differentsolutionsmayhave different sized tuples.

- Explicitconstraintsrequires$x_i \in$ {j/j isan integer1≤ j≤ n}
- Implicitconstraintsrequires:
  No two bethesame&thatthesum of thecorrespondingw$_i$'sbem
  i.e.,(1, 2, 4)&(1, 4, 2) represents thesame. Another constraint is $x_i<x_{i+1}$1≤ i ≤ k

W_i→weightofitemi

M→Capacityof bag (subset)
X_i→the element of thesolution vector iseitheroneorzero.
$X_i$valuedependingonwhethertheweightwiisincludedornot. If X_i=1
then wi is chosen.
IfX_i=0thenwiisnotchosen.

$$\underbrace{\sum_{i=1}^{k} W(i)X(i)}_{\text{Total sum till now}} + \underbrace{\sum_{i=k+1}^{n} W(i)}_{\text{Still there}} \geq M$$

Theaboveequationspecifythatx_1,x_2,x_3,---$x_k$cannotleadtoananswernodeifthiscondition is not satisfied.

$$\sum_{i=1}^{k} W(i)X(i) + W(k+1) > M$$

Theequationcannotleadtosolution.

$$B_k(X(1), \ldots, X(k)) = true \text{ iff } \left(\sum_{i=1}^{k} W(i)X(i) + \sum_{i=k+1}^{n} W(i) \geq M \text{ and } \sum_{i=1}^{k} W(i)X(i) + W(k+1) \leq M \right)$$

$$s = \sum_{j=1}^{k-1} W(j)X(j), \quad \text{and} \quad r = \sum_{j=k}^{n} W(j)$$

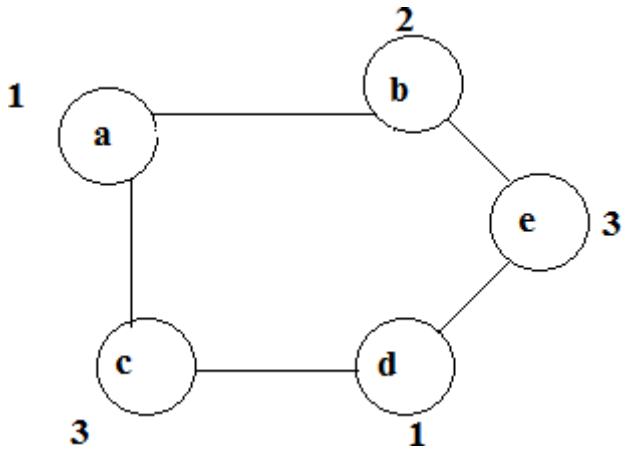
Recursivebacktrackingalgorithmforsumofsubsetsproblem

AlgorithmSumOfSub(s,k,r)
{
//$s = \sum_{j=1}^{k-1} W(j)X(j)$, and $r = \sum_{j=k}^{n} W(j)$
X[k]=1
If(S+w[k]=m)thenwrite(x[1:]);//subsetfound.
Else if (S+w[k] + w{k+1] ≤ M)
ThenSumOfSub(S+w[k],k+1,r-w[k]);
if((S+r-w{k]≥ M)and(S+w[k+1]≤M))then
{
X[k]=0;
SumOfSub(S,k+1,r-w[k]);
}
}

GraphColoring:

Let G be a undirected graph and 'm' be a given +ve integer. The graph coloring problem is assigningcolorstotheverticesofanundirectedgraphwiththerestrictionthatnotwoadjacent vertices are assigned the same color yet only 'm' colors are used.
The optimization version calls for coloring a graph using the minimum number of coloring.
Thedecisionversion,knownasK-coloringaskswhetheragraphiscolourableusingatmostk- colors.
Notethat,if'd'isthedegreeofthegivengraphthenitcan be coloredwith'd+1'colors.
Them-colorabilityoptimizationproblemasksforthesmallestinteger'm'forwhichthegraphG can be colored. This integer is referred as "**Chromatic number**" of the graph.

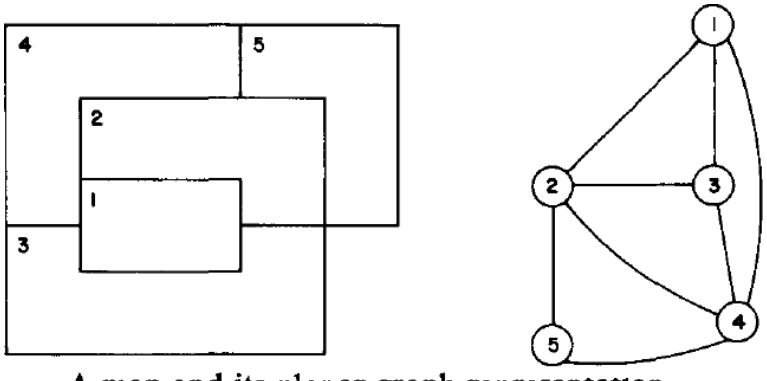
Example



- Abovegraphcanbecolored with3 colors1, 2, &3.
- Thecolor of each nodeisindicatednext to it.
- 3-colorsareneededtocolorthisgraphandhencethisgraph'ChromaticNumber is 3.

- Agraphissaidtobeplanariffitcanbedrawninaplane(flat)insuchawaythatnotwo edges cross each other.
- **M-Colorabilitydecisionproblem**isthe4-colorproblemforplanargraphs.
- Givenanymap,cantheregionsbecoloredinsuchawaythatnotwoadjacentregions have the same color yet only 4-colors are needed?
- Tosolvethisproblem,graphsareveryuseful,becauseamapcaneasilybetransformed into a graph.
- Eachregionofthemapbecomesanode,andiftworegionsareadjacent,thenthe corresponding nodes are joined by an edge.

  - **Example:**



  - **A map and its planar graph representation**

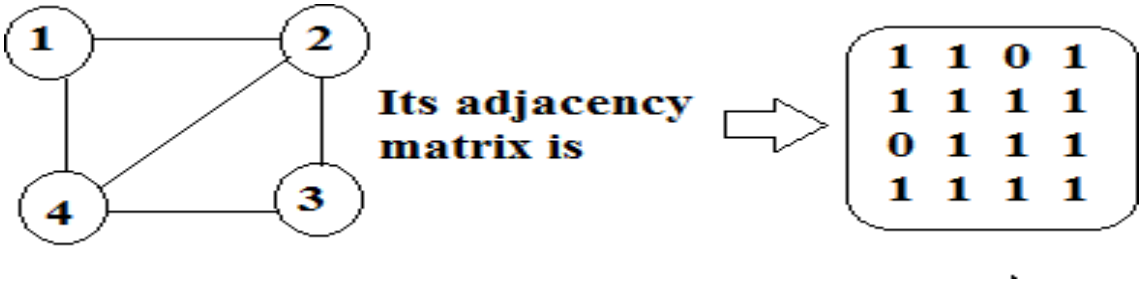
Theabovemaprequires4 colors.

- Manyyears, it was knownthat 5-colors wererequiredtocolorthismap.

- Afterseveralhundredyears,thisproblemwassolvedbyagroupofmathematicianswith the help of a computer. They show that 4-colors are sufficient.

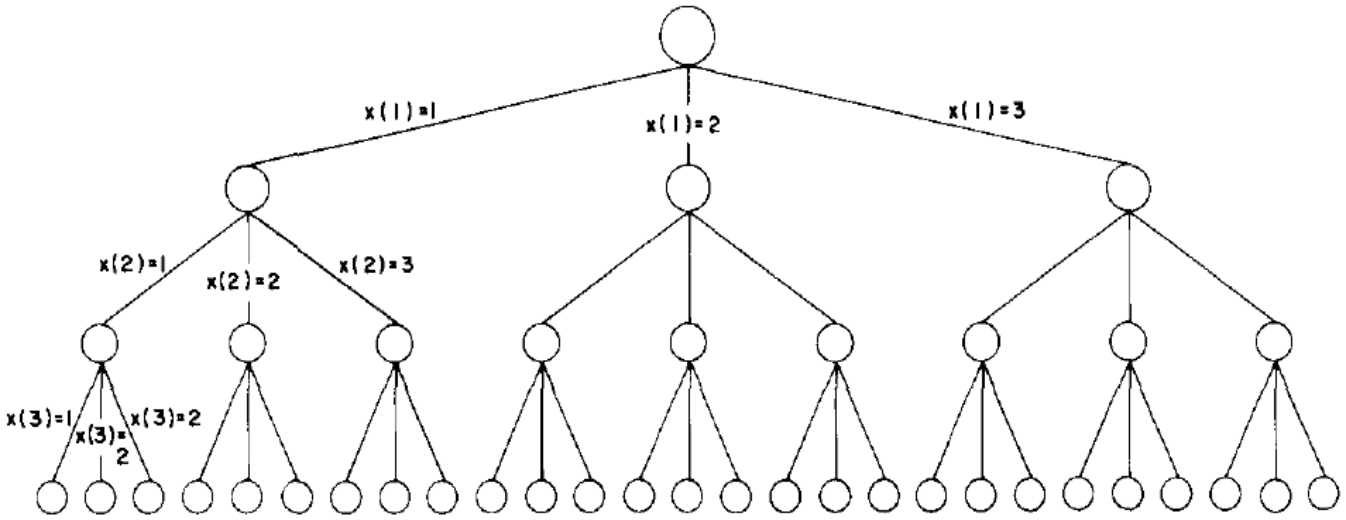
Supposewerepresentagraphbyitsadjacencymatrix G[1:n, 1:n]

Ex:



HereG[i,j]=1if (i,j)isanedgeofG,andG[i,j]=0 otherwise.
Colorsarerepresentedbytheintegers1,2,---mandthesolutionsaregivenbythen-tuple(x1, x2,---xn)
xi→Colorofnodei.

StateSpaceTreefor
n=3→ nodes
m=3→colors



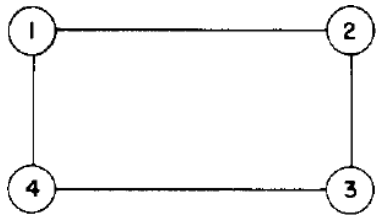
State space tree for MCOLORING when $n = 3$ and $m = 3$

1st node coloured in 3-ways
2ndnodecolouredin3-ways
3rdnodecoloured in3-ways
Sowecancolour in thegraph in 27possibilities of colouring.

Findingallm-coloringofagraph	Gettingnext color

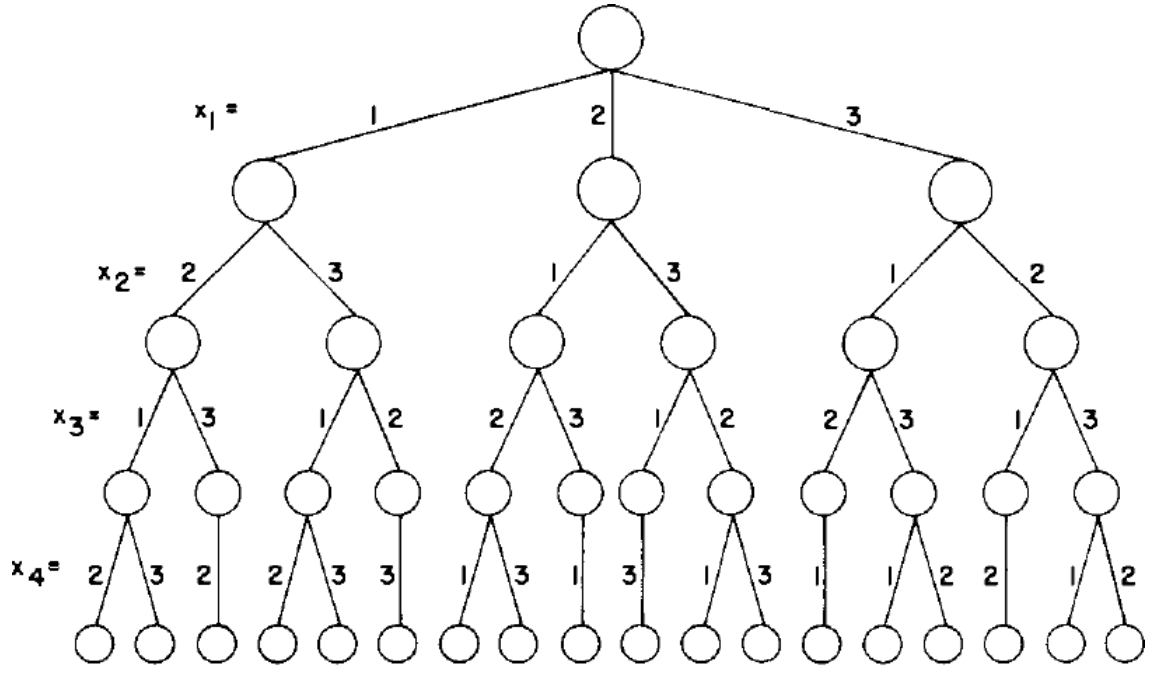
AlgorithmmColoring(k){
//g(1:n,1:n)→booleanadjacencymatrix.
//k→index(node)ofthenextvertextocolor.
repeat{
nextvalue(k);//assigntox[k]alegalcolor.
if(x[k]=0) then return; // no new color possible
if(k=n)thenwrite(x[1:n];
else mcoloring(k+1);
}
until(false)
}
	AlgorithmNextValue(k){
	//x[1],x[2],---x[k-1]havebeenassigned integer values in the range [1, m]
	repeat{
	x[k]=(x[k]+1)mod(m+1);//nexthighestcolor
	if(x[k]=0)thenreturn;//allcolorshave been used.
	forj=1 to ndo
	{
	if((g[k,j]≠0)and(x[k]=x[j])) then
	break;
	}
	if(j=n+1)then return;//newcolorfound
	} until(false)
	}

Previouspaper example:



Adjacencymatrixis

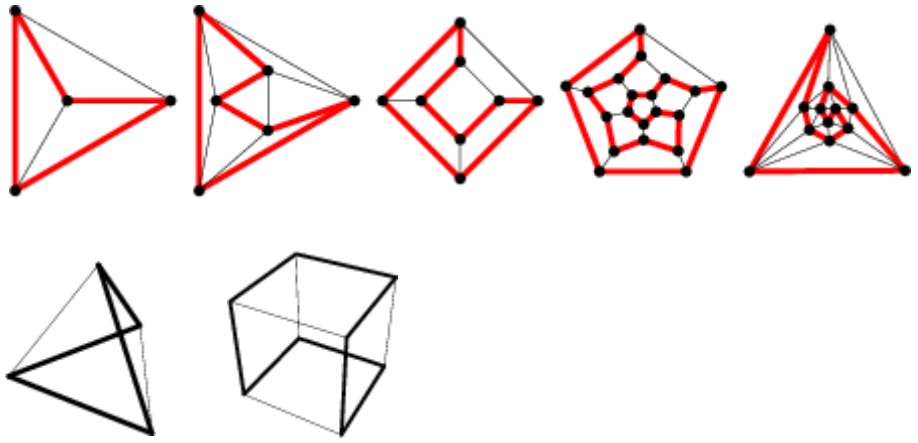
$$\begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$



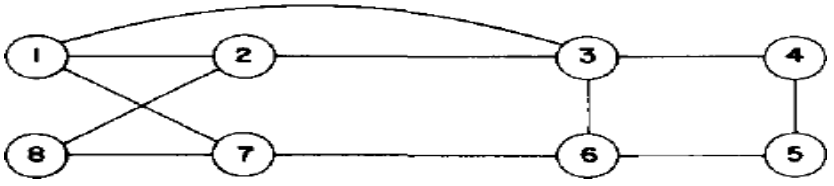
A 4 node graph and all possible 3 colorings

HamiltonianCycles:

- **Def:** Let G=(V, E) be a connected graph with n vertices. A Hamiltonian cycle is a round trip path along n-edges of G that visits every vertex once & returns to its startingposition.
- ItisalsocalledtheHamiltoniancircuit.
- Hamiltoniancircuitisagraphcycle(i.e.,closedloop)throughagraphthatvisitseach node exactly once.
- AgraphpossessingaHamiltoniancycleissaidtobeHamiltoniangraph.
  Example:

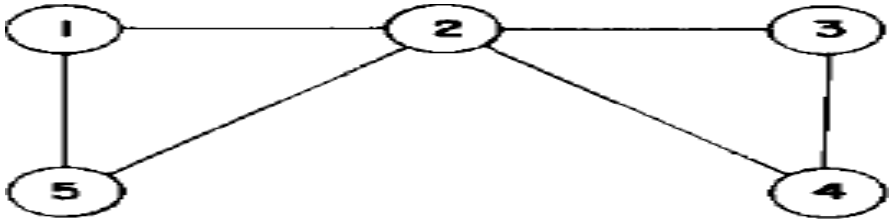


- In graph G, Hamiltonian cycle begins at some vertiex v1 $\in$ G and the vertices ofGarevisitedintheorder$v_1,v_2,---v_{n+1}$,thentheedges(v_i,v_{i+1})areinE, $1 \leq i \leq n$.



→g1

Theabovegraphcontains Hamiltoniancycle:1,2,8,7,6,5,4,3,1



TheabovegraphcontainsnoHamiltoniancycles.

- Thereisnoknowneasywaytodeterminewhetheragivengraphcontainsa Hamiltonian cycle.
- Byusingbacktrackingmethod,it can be possible
  - Backtrackingalgorithm,thatfindsalltheHamiltoniancyclesinagraph.
  - Thegraphmaybedirected or undirected.Onlydistinct cycles areoutput.
  - From graphg1backtrackingsolution vector={1,2,8,7, 6,5,4,3, 1}
  - Thebacktrackingsolutionvector$(x_1,x_2,---x_n)$
    x_i→i^{th} visited vertex of proposed cycle.

- Byusingbacktrackingweneedtodeterminehowtocomputethesetofpossible vertices for x_k if x_1,x_2,x_3---x_{k-1}have already been chosen.
  If k=1 then x1 canbeanyof then-vertices.

Byusing"NextValue"algorithmtherecursivebacktrackingschemetofindallHamiltoman cycles.

Thisalgorithmisstarted by1st initializingtheadjacencymatrix G[1:n,1:n]thensettingx[2:n] to zero & x[1] to 1, and then executing Hamiltonian (2)

GeneratingNextVertex	**FindingallHamiltonian Cycles**

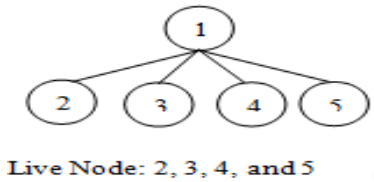
AlgorithmNextValue(k)
{
//x[1:k-1]→ispathof k-1distinct vertices.
//ifx[k]=0,thennovertexhasyetbeen assigned to x[k]
Repeat{
X[k]=(x[k]+1)mod(n+1);//Nextvertex
If(x[k]=0) then return;
If(G[x[k-1],x[k]]≠0)then
{
Forj:=1tok-1 doif(x[j]=x[k])then break;
//Checkfor distinctness
If(j=k)then//iftrue,thenvertexisdistinct
If((k<n) or (k=n) and G[x[n], x[1]]≠0))
Then return ;
}
}
Until (false);
}
	AlgorithmHamiltonian(k)
	{
	Repeat{
	NextValue(k);//assignalegalnextvaluetox[k]
	If(x[k]=0) then return;
	If(k=n)thenwrite(x[1:n]);
	Else Hamiltonian(k+1);
	} until(false)
	}

Branch&Bound

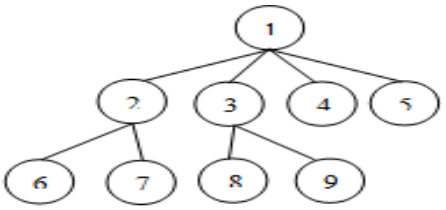
Branch & Bound (B & B) is general algorithm (or Systematic method) for finding optimal solution of various optimization problems, especially in discrete and combinatorial optimization.

- TheB&Bstrategyisverysimilartobacktrackinginthatastatespacetreeisusedtosolve a problem.
- Thedifferencesarethatthe B&B method
  - ✓ Doesnot limit ustoanyparticularwayoftraversingthetree.
  - ✓ Itis used onlyforoptimizationproblem
  - ✓ Itis applicabletoawidevarietyofdiscretecombinatorialproblem.
- B&B israthergeneraloptimizationtechniquethatappliieswherethegreedy method& dynamic programming fail.
- It is much slower, indeed (truly), it often (rapidly) leads to exponential time complexities in the worst case.
- The term B&B refers to all state space search methods in which all children of the"E-node" are generated before any other "live node" can become the "E-node"
  - ✓ **Livenode→**isanodethathasbeengeneratedbutwhosechildrenhavenotyetbeen generated.
  - ✓ **E-node→**isalivenodewhosechildrenarecurrentlybeingexplored.

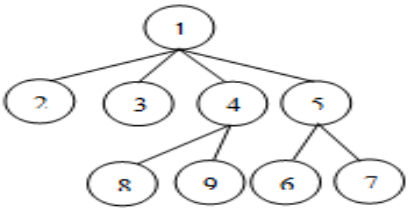
✓ **Dead node→** is a generated node that is not to be expanded or explored any further. All children of a dead node have already been expanded.



Live Node: 2, 3, 4, and 5



FIFO Branch & Bound (BFS) Children of E-node are inserted in a queue.



LIFO Branch & Bound (D-Search) Children of E-node are inserted in a stack.

➢ Two graph search strategies, BFS & D-search (DFS) in which the exploration of a newnode cannot begin until the node currently being explored is fully explored.

➢ BothBFS&D-search(DFS)generalizedtoB&Bstrategies.

✓ **BFS→**like state space search will be called FIFO (First In First Out) search as the list of live nodes is "First-in-first-out" list (or queue).

✓ **D-search (DFS)→** Like state space search will be called LIFO (Last In First Out) search as the list of live nodes is a "last-in-first-out" list (or stack).

➢ In backtracking, bounding function areused to help avoid the generation of sub-trees that do not contain an answer node.

➢ Wewilluse3-typesofsearchstrategiesinbranchandbound

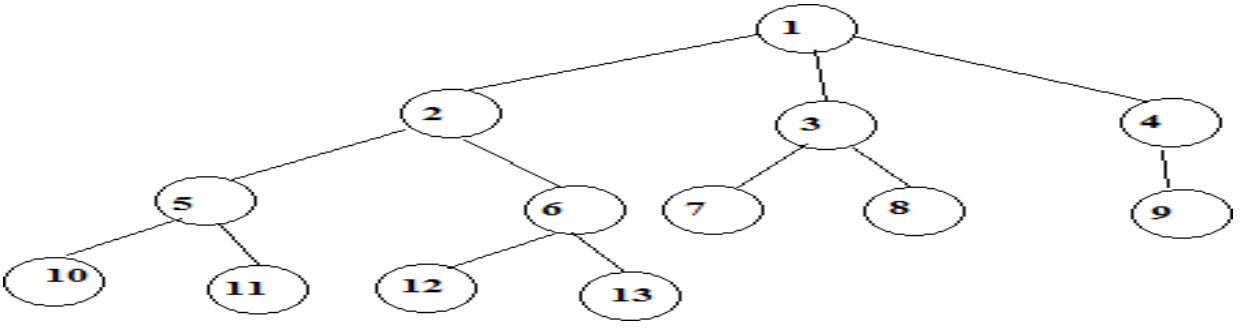
1) FIFO(FirstInFirstOut)search
2) LIFO(LastInFirstOut)search
3) LC(LeastCount)search

FIFOB&B:

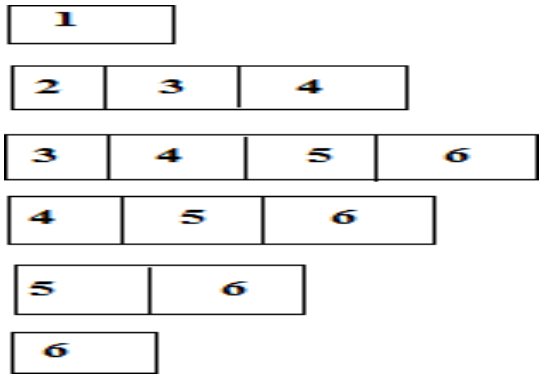
FIFOBranch&BoundisaBFS.
Inthis,childrenofE-Node(orLivenodes)areinsertedinaqueue. Implementation of list of live nodes as a queue

✓ Least()→Removes theheadoftheQueue

✓ Add()→Adds the nodetotheendoftheQueue



Assumethatnode '12'isananswer nodeinFIFOsearch,1st wetakeE-nodehas'1'



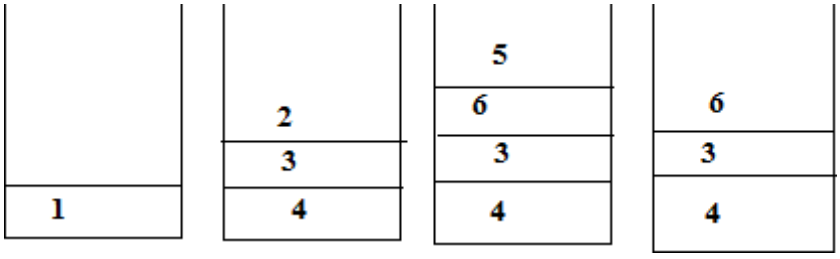
LIFOB&B:

LIFOBrach&BoundisaD-search(orDFS).

InthischildrenofE-node(livenodes)areinsertedinastack

Implementation of List of live nodes as a stack

- ✓ Least()→Removes thetopofthestack
- ✓ ADD()→Addsthenodeto the top ofthestack.



LeastCost(LC)Search:

The selection rule for the next E-node in FIFO or LIFO branch and bound is sometimes "blind". i.e., the selection rule does not give any preference to a node that has a very good chance of getting the search to an answer node quickly.

The search for an answer node can often be speeded by using an "intelligent" ranking function. It is also called an approximate cost function "**Ĉ**".

Expendednode(E-node)isthelivenodewiththebest**Ĉ**value.

Branching: A set of solutions, which is represented by a node, can be partitioned intomutually (jointly or commonly) exclusive (special) sets. Each subset in the partition is represented by a child of the original node.

Lower bounding: An algorithm is available for calculating a lower bound on the cost of any solution in a given subset.

EachnodeXinthesearchtreeisassociatedwithacost:**Ĉ(X)**

C=cost of reaching the current node, X(E-node) form the root + The cost of reaching an answer node form X.

Ĉ=g(X)+H(X).

Example:

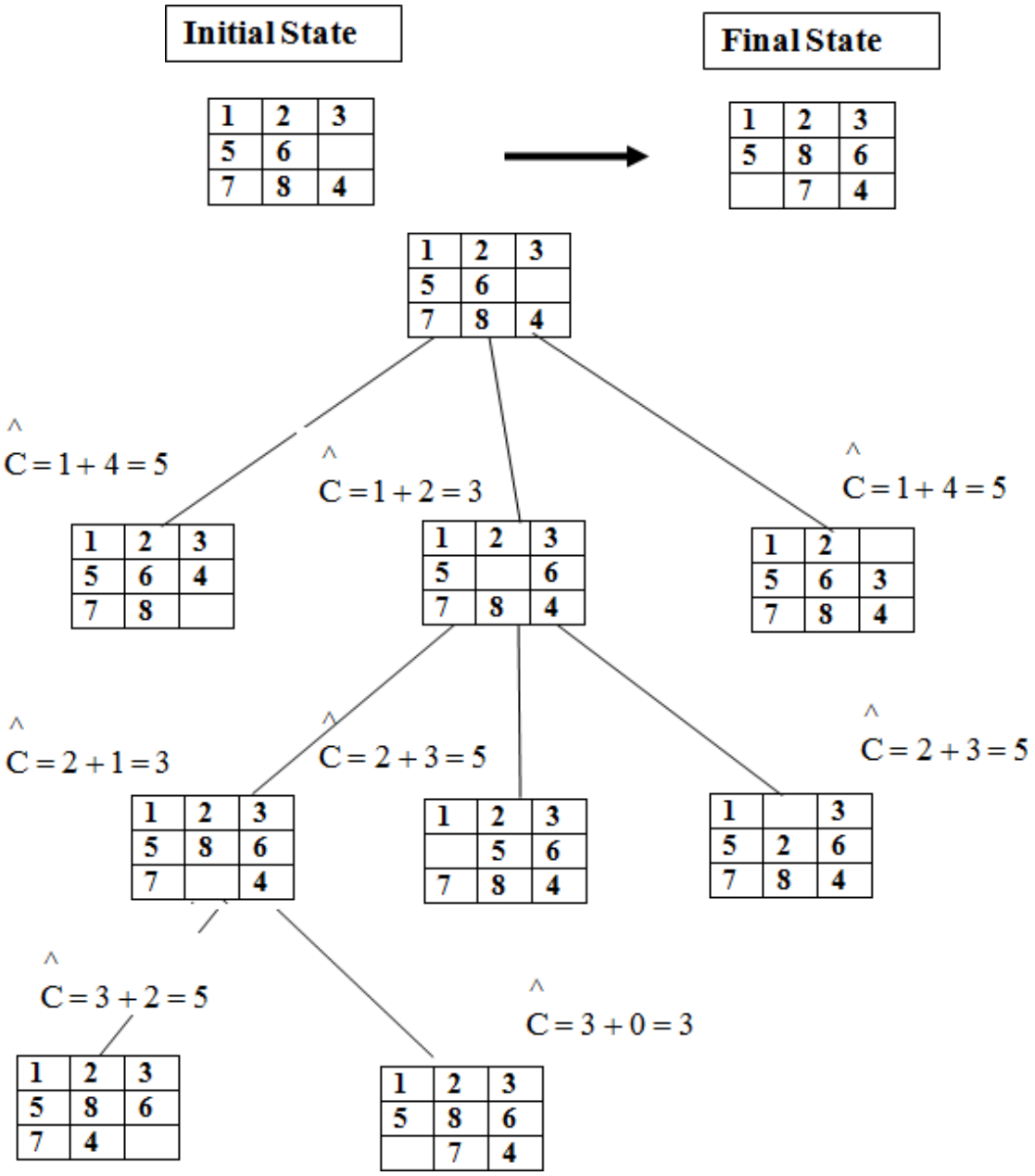
8-puzzle

Costfunction: **Ĉ**=g(x) +h(x)

where h(x)=thenumberofmisplacedtiles

andg(x)=thenumberofmovessofar

Assumption: move one tile in any direction cost 1.



Note: Incaseoftie,choosetheleftmostnode.

TravellingSalesman Problem:

Def:- FindatourofminimumcoststartingfromanodeSgoingthroughothernodes only once and returning to the starting point S.

TimeConmlexityofTSPforDynamicProgrammingalgorithmis$O(n^2 2^n)$

B&B algorithms for this problem, the worest case complexity will not be any better than $O(n^2 2^n)$ but good bunding functions will enables these B&B algorithms to solve some problem instances in much less time than required by the dynamic programming alogrithm.

Let$G=(V,E)$beadirectedgraphdefininganinstancesofTSP. Let

C_{ij}→ cost of edge <i, j>

$C_{ij} = \infty$if$<i,j> \notin E$

$|V|=n$→totalnumberofvertices.

Assumethateverytour starts& ends at vertex1.

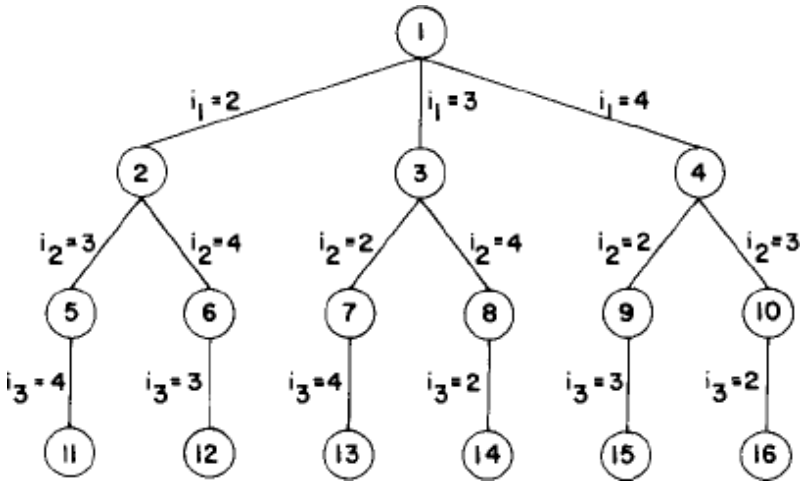
SolutionSpaceS={1, Π, 1/$\Pi$is a permutationof(2,3. 4 ----------n)} then $|S|=(n-1)!$

ThesizeofSreducedbyrestrictingS

Sothat$(1,i_1,i2,\text{-----}i_{n-1},1\} \in Siff<i_j,i_{j+1}> \in E.O \leq j \leq n-1, i_0-i_n=1$

Scanbeorganizedinto"Statespacetree".

ConsiderthefollowingExample



Statespacetreeforthetravellingsalespersonproblemwithn=4and$i_0=i_4=1$ The

above diagram shows tree organization of a complete graph with |V|=4.

Each leafnode 'L'is a solution nodeand represents thetourdefinedbythepath from the root to L.

Node12representsthetour.

$i_0=1,i_1=2,i_2=4,i_3=3,i_4=1$

Node14representsthetour.

$i_0=1,i_1=3,i_2=4,i_3=2,i_4=1.$

TSPissolvedbyusingLCBranch&Bound:

TouseLCBBtosearchthetravellingsalesperson"Statespacetree"firstdefineacostfunction C(.) and other 2 functions $\hat{C}$(.) & u(.)

Suchthat$\hat{C}(r) \leq C(r) \leq u(r)$→ forallnodesr.

CostC(.)→isthesolutionnode1withleastC(.)correspondstoashortesttourin G.

C(A)={Length of tour defined by the path from root to A if A is leaf
Costofaminimum-costleafinthesub-treeA,ifAisnotleaf}

From (1) $\hat{C}(r) \leq C(r)$ then $\hat{C}(r) \rightarrow$ is thelengthofthepathdefinedatnodeA.

Fromprevious example the path defined at node 6 is $i_0, i_1, i_2 = 1,2,4$ & it consists edge of <1,2>&<2,4>

A better $\hat{C}(r)$ can be obtained by using the reduced cost matrix corresponding to G.

- A row (column) is said to be reduced iff it contains at least one zero & remaining entries are non negative.
- A matrix is reduced iff every row & column is reduced.

$$\begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix}$$

(a) Cost Matrix

$$\begin{bmatrix} \infty & 10 & 17 & 0 & 1 \\ 12 & \infty & 11 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & 12 & \infty \end{bmatrix}$$

(b) Reduced Cost Matrix
L = 25

Given the following cost matrix:

$$\begin{bmatrix} inf & 20 & 30 & 10 & 11 \\ 15 & inf & 16 & 4 & 2 \\ 3 & 5 & inf & 2 & 4 \\ 19 & 6 & 18 & inf & 3 \\ 16 & 4 & 7 & 16 & inf \end{bmatrix}$$

- The TSP starts from node 1: **Node 1**
- Reduced Matrix: To get the lower bound of the path starting at node 1

Row # 1: reduce by 10	Row #2: reduce 2	Row #3: reduce by 2
$\begin{bmatrix} inf & 10 & 20 & \\ 15 & inf & 16 & \\ 3 & 5 & inf & 2 \\ 19 & 6 & 18 & inf \\ 16 & 4 & 7 & 16 \end{bmatrix}$	$\begin{bmatrix} inf & 10 & 20 & \\ 13 & inf & 14 & \\ 3 & 5 & inf & 2 \\ 19 & 6 & 18 & inf \\ 16 & 4 & 7 & 16 \end{bmatrix}$	$\begin{bmatrix} inf & 10 & 20 & \\ 13 & inf & 14 & \\ 1 & 3 & inf & 0 \\ 19 & 6 & 18 & inf \\ 16 & 4 & 7 & 16 \end{bmatrix}$
Row # 4: Reduce by 3:	Row #5: Reduce by 4	Column 1: Reduce by 1

$\begin{bmatrix} inf & 10 & 20 & \\ 13 & inf & 14 & \\ 1 & 3 & inf & \\ 16 & 3 & 15 & in \\ 16 & 4 & 7 & 16 \end{bmatrix}$	$\begin{bmatrix} inf & 10 & 20 & \\ 13 & inf & 14 & \\ 1 & 3 & inf & \\ 16 & 3 & 15 & in \\ 12 & 0 & 3 & 12 \end{bmatrix}$	$\begin{bmatrix} inf & 10 & 20 & \\ 12 & inf & 14 & \\ 0 & 3 & inf & \\ 15 & 3 & 15 & in \\ 11 & 0 & 3 & 12 \end{bmatrix}$
Column2:Itisreduced.	Column 3: Reduceby3 $\begin{bmatrix} inf & 10 & 17 & 0 & 1 \\ 12 & inf & 11 & 2 & 0 \\ 0 & 3 & inf & 0 & 2 \\ 15 & 3 & 12 & inf & 0 \\ 11 & 0 & 0 & 12 & inf \end{bmatrix}$	Column4:Itisreduced. Column5:Itisreduced.

Thereduced cost is: RCL=25
Sothecostofnode1is:Cost(1)=25 The reduced matrix is:

Cost (1) = 25

$$\begin{bmatrix} inf & 10 & 17 & 0 & 1 \\ 12 & inf & 11 & 2 & 0 \\ 0 & 3 & inf & 0 & 2 \\ 15 & 3 & 12 & inf & 0 \\ 11 & 0 & 0 & 12 & inf \end{bmatrix}$$

- **Choosetogotovertex 2:Node2**

  - Costof edge<1,2>is: A(1,2) =10
  - Setrow#1 =infsincewearechoosingedge<1,2>
  - Setcolumn # 2=infsincewearechoosingedge<1,2>
  - SetA(2,1)=inf
  - Theresultingcostmatrixis:

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ inf & inf & 11 & 2 & 0 \\ 0 & inf & inf & 0 & 2 \\ 15 & inf & 12 & inf & 0 \\ 11 & inf & 0 & 12 & inf \end{bmatrix}$$

  - Thematrixisreduced:
  - RCL=0
  - Thecost ofnode2(Consideringvertex2 fromvertex1) is:

  Cost(2)= cost(1)+ A(1,2)= 25 + 10 = 35

- **Choosetogotovertex 3:Node3**

- Costof edge<1,3>is:A(1,3) =17 (Inthereducedmatrix
- Set row#1 =inf since we arestartingfrom node1
- Setcolumn # 3 = inf sincewearechoosingedge<1,3>
- SetA(3,1)=inf
- Theresultingcostmatrixis:

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ 12 & inf & inf & 2 & 0 \\ inf & 3 & inf & 0 & 2 \\ 15 & 3 & inf & inf & 0 \\ 11 & 0 & inf & 12 & inf \end{bmatrix}$$

Reducethematrix: Rowsarereduced

Thecolumnsarereducedexceptforcolumn#1:

Reduce column 1 by 11:

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ 1 & inf & inf & 2 & 0 \\ inf & 3 & inf & 0 & 2 \\ 4 & 3 & inf & inf & 0 \\ 0 & 0 & inf & 12 & inf \end{bmatrix}$$

Thelower bound is:RCL=11

Thecostofgoingthrough node3 is:

cost(3)=cost(1)+RCL+A(1,3)=25 +11 +17=53

- Choosetogo tovertex4: **Node 4**

Rememberthatthecostmatrixistheonethatwasreducedatthestartingvertex1 Cost of edge <1,4> is: A(1,4) = 0

Set row#1 =inf since we arestartingfrom node1

Setcolumn#4=infsincewearechoosingedge<1,4> Set

A(4,1) = inf

Theresultingcostmatrixis:

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ 12 & inf & 11 & inf & 0 \\ 0 & 3 & inf & inf & 2 \\ inf & 3 & 12 & inf & 0 \\ 11 & 0 & 0 & inf & inf \end{bmatrix}$$

Reducethematrix: Rowsare reduced

Columnsarereduced The
lower bound is: RCL = 0
Thecostofgoingthrough node4 is:
cost(4)=cost(1)+RCL+A(1,4) =25 +0 +0=25

- **Choosetogoto vertex5: Node5**

  - Rememberthatthecostmatrix istheonethatwasreducedatstartingvertex1
  - Costof edge<1,5>is: A(1,5) =1
  - Set row#1 =inf since we arestartingfrom node1
  - Setcolumn # 5 = infsincewearechoosingedge<1,5>
  - SetA(5,1)=inf
  - Theresultingcostmatrixis:

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ 12 & inf & 11 & 2 & inf \\ 0 & 3 & inf & 0 & inf \\ 15 & 3 & 12 & inf & inf \\ inf & 0 & 0 & 12 & inf \end{bmatrix}$$

Reducethematrix:
Reducerows:
Reducerow #2: Reduceby2

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ 10 & inf & 9 & 0 & inf \\ 0 & 3 & inf & 0 & inf \\ 15 & 3 & 12 & inf & inf \\ inf & 0 & 0 & 12 & inf \end{bmatrix}$$

Reducerow #4: Reduceby3

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ 10 & inf & 9 & 0 & inf \\ 0 & 3 & inf & 0 & inf \\ 12 & 0 & 9 & inf & inf \\ inf & 0 & 0 & 12 & inf \end{bmatrix}$$

Columnsare reduced
Thelowerboundis:RCL=2+3=5 The cost
of going through node 5 is:
cost(5)=cost(1)+RCL+A(1,5)=25 +5 +1=31

Insummary:

Sothelivenodeswehavesofarare:

- ✓ 2: cost(2)=35, path:1->2
- ✓ 3: cost(3)=53, path:1->3
- ✓ 4: cost(4)=25, path: 1->4
- ✓ 5: cost(5)=31, path:1->5

Explorethenodewiththelowestcost:Node4hasacostof25 Vertices to be explored from node 4: 2, 3, and 5

Nowwearestartingfromthecostmatrixat node 4is:

Cost (4) = 25

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ 12 & inf & 11 & inf & 0 \\ 0 & 3 & inf & inf & 2 \\ inf & 3 & 12 & inf & 0 \\ 11 & 0 & 0 & inf & inf \end{bmatrix}$$

- **Choosetogotovertex 2:Node6 (pathis 1->4->2)**

Costof edge<4,2>is: A(4,2) =3

Setrow#4 =inf since weareconsideringedge<4,2>

Setcolumn#2=infsinceweareconsideringedge<4,2> Set A(2,1) = inf

Theresultingcostmatrixis:

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ inf & inf & 11 & inf & 0 \\ 0 & inf & inf & inf & 2 \\ inf & inf & inf & inf & inf \\ 11 & inf & 0 & inf & inf \end{bmatrix}$$

Reducethematrix: Rowsare reduced

Columnsarereduced The lower bound is: RCL = 0

Thecostofgoingthrough node2 is:

cost(6)=cost(4)+RCL+A(4,2)=25 +0 +3=28

- **Chooseto go tovertex 3: Node7 (pathis 1->4->3)**

Costof edge<4,3>is: A(4,3) =12
Setrow#4 =inf since weareconsideringedge<4,3>
Setcolumn#3=infsincewareconsideringedge<4,3> Set
A(3,1) = inf
Theresultingcostmatrixis:

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ 12 & inf & inf & inf & 0 \\ inf & 3 & inf & inf & 2 \\ inf & inf & inf & inf & inf \\ 11 & 0 & inf & inf & inf \end{bmatrix}$$

Reducethematrix:

Reducerow #3: by2:

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ 12 & inf & inf & inf & 0 \\ inf & 1 & inf & inf & 0 \\ inf & inf & inf & inf & inf \\ 11 & 0 & inf & inf & inf \end{bmatrix}$$

Reducecolumn #1: by 11

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ 1 & inf & inf & inf & 0 \\ inf & 1 & inf & inf & 0 \\ inf & inf & inf & inf & inf \\ 0 & 0 & inf & inf & inf \end{bmatrix}$$

Thelower bound is: RCL=13
SotheRCLofnode7(Consideringvertex3fromvertex4)is: Cost(7) = cost(4) + RCL + A(4,3) = 25 + 13 + 12 = 50

- Choosetogoto vertex 5:**Node 8**(pathis 1->4->5)

Costofedge<4,5>is: A(4,5) =0
Setrow#4 =inf since weareconsideringedge<4,5>
Setcolumn#5=infsinceweareconsideringedge<4,5> Set
A(5,1) = inf
Theresultingcostmatrixis:

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ 12 & inf & 11 & inf & inf \\ 0 & 3 & inf & inf & inf \\ inf & inf & inf & inf & inf \\ inf & 0 & 0 & inf & inf \end{bmatrix}$$

Reduce the matrix:
Reducedrow2:by11

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ 1 & inf & 0 & inf & inf \\ 0 & 3 & inf & inf & inf \\ inf & inf & inf & inf & inf \\ inf & 0 & 0 & inf & inf \end{bmatrix}$$

Columnsare reduced
Thelower bound is: RCL=11
Sothecostofnode8(Consideringvertex5fromvertex4)is:
Cost(8) = cost(4) + RCL + A(4,5) = 25 + 11 + 0 = 36

In summary: Sothe live nodes wehavesofarare:

- ✓ 2: cost(2)=35, path:1->2
- ✓ 3: cost(3)=53, path: 1->3
- ✓ 5: cost(5)=31, path:1->5
- ✓ 6: cost(6)=28, path:1->4->2
- ✓ 7: cost(7)=50, path:1->4->3
- ✓ 8: cost(8)=36, path:1->4->5

- Explorethenodewiththelowest cost:Node6 hasacost of 28
- Verticestobeexploredfromnode 6:3and 5
- Nowwearestartingfromthecostmatrixat node 6is:

Cost (6) = 28

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ inf & inf & 11 & inf & 0 \\ 0 & inf & inf & inf & 2 \\ inf & inf & inf & inf & inf \\ 11 & inf & 0 & inf & inf \end{bmatrix}$$

- **Choosetogoto vertex3: Node9(pathis 1->4->2->3)**

Costof edge<2,3>is: A(2,3) =11
Setrow#2 =inf since weareconsideringedge<2,3>
Setcolumn#3=infsincewearecons ideringedge<2,3> Set
A(3,1) = inf
Theresultingcostmatrixis:

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ inf & inf & inf & inf & inf \\ inf & inf & inf & inf & 2 \\ inf & inf & inf & inf & inf \\ 11 & inf & inf & inf & inf \end{bmatrix}$$

Reducethematrix:Reducerow #3: by2

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ inf & inf & inf & inf & inf \\ inf & inf & inf & inf & 0 \\ inf & inf & inf & inf & inf \\ 11 & inf & inf & inf & inf \end{bmatrix}$$

Reducecolumn #1: by11

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ inf & inf & inf & inf & inf \\ inf & inf & inf & inf & 0 \\ inf & inf & inf & inf & inf \\ 0 & inf & inf & inf & inf \end{bmatrix}$$

Thelower bound is: RCL=2 +11 =13
Sothecostofnode9(Consideringvertex3fromvertex2)is: Cost(9) =
cost(6) + RCL + A(2,3) = 28 + 13 + 11 = 52

- **Choosetogotovertex 5:Node10 (path is1->4->2->5)**

Costof edge<2,5>is: A(2,5) =0
Setrow#2 =inf since weareconsideringedge<2,3>
Setcolumn#3=infsincewearecons ideringedge<2,3> Set
A(5,1) = inf
Theresultingcostmatrixis:

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ inf & inf & inf & inf & inf \\ 0 & inf & inf & inf & inf \\ inf & inf & inf & inf & inf \\ inf & inf & 0 & inf & inf \end{bmatrix}$$

Reducethematrix: Rows reduced
Columnsreduced
The lower bound is: RCL = 0
Sothecostofnode10(Consideringvertex5fromvertex2)is:
Cost(10) = cost(6) + RCL + A(2,3) = 28 + 0 + 0 = 28

Insummary: **Sothelivenodeswehavesofarare:**

- ✓ 2: cost(2)=35, path:1->2
- ✓ 3: cost(3)=53, path:1->3
- ✓ 5: cost(5)=31, path:1->5
- ✓ 7: cost(7)=50, path:1->4->3
- ✓ 8: cost(8)=36, path:1->4->5
- ✓ 9:cost(9)=52, path: 1->4->2->3
- ✓ 10:cost(2)=28, path: 1->4->2->5
- Explorethenodewiththelowest cost: Node10 has a cost of 28
- Verticestobe explored fromnode10: 3
- Nowwearestartingfromthecostmatrixat node 10is:

Cost (10)=28

$$\begin{bmatrix} inf & inf & inf & inf & inf \\ inf & inf & inf & inf & inf \\ 0 & inf & inf & inf & inf \\ inf & inf & inf & inf & inf \\ inf & inf & 0 & inf & inf \end{bmatrix}$$

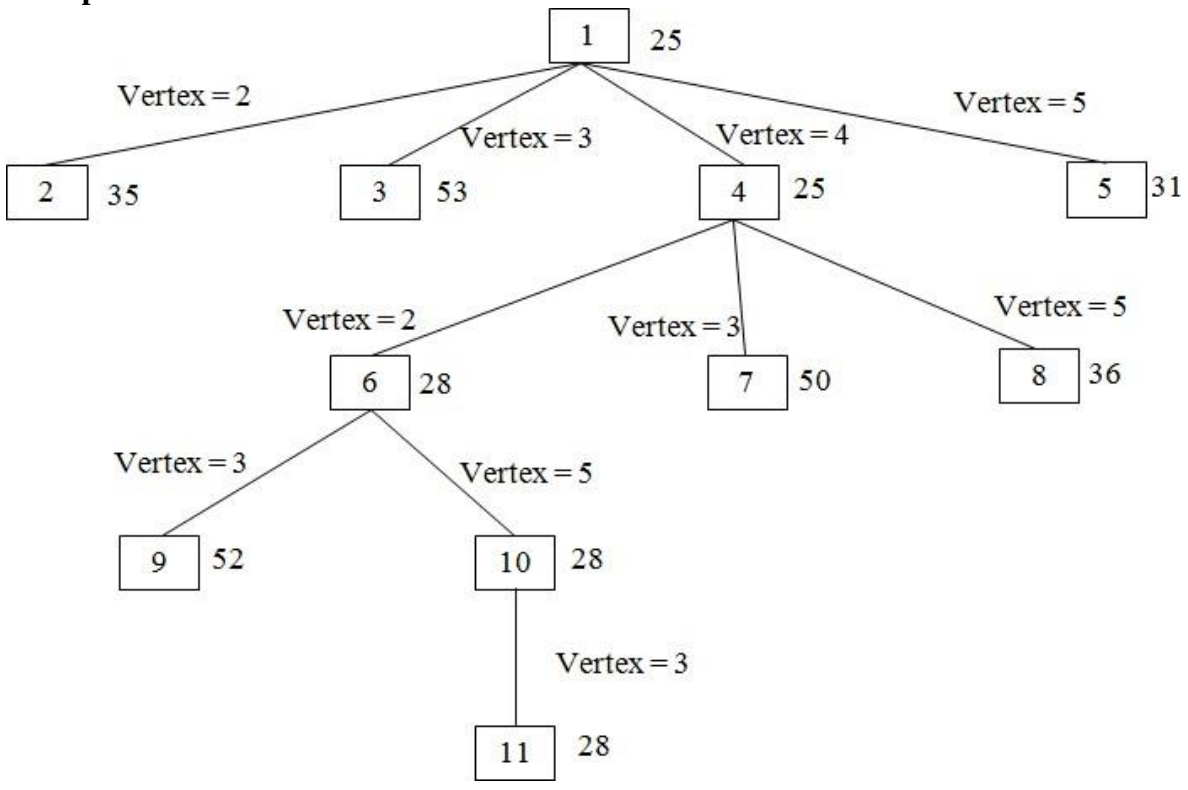
- **Choosetogo tovertex3: Node11(pathis1->4->2->5->3)**

Costof edge<5,3>is: A(5,3) =0
Setrow#5 =inf since weareconsideringedge<5,3>
Setcolumn#3=infsincewearecons ideringedge<5,3> Set
A(3,1) = inf
Theresultingcostmatrixis:

$$\begin{vmatrix} inf & inf & inf & inf & inf \\ inf & inf & inf & inf & inf \\ inf & inf & inf & inf & inf \\ inf & inf & inf & inf & inf \\ inf & inf & inf & inf & inf \end{vmatrix}$$

Reducethe matrix: Rows reduced
Columnsreduced
The lower bound is: RCL = 0
Sothecostofnode11(Consideringvertex5fromvertex3)is:
Cost(11) = cost(10) + RCL + A(5,3) = 28 + 0 + 0 = 28

StateSpaceTree:



O/1KnapsackProblem

WhatisKnapsackProblem: Knapsackproblemisaproblemincombinatorialoptimization, Given a set of items, each with a mass & a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit &the total value is as large as possible.

O-1KnapsackProblemcanformulateas. Lettherebenitems,Z_1**to**$Z_n$where$Z_i$hasvalue P_i&weightw_i.Themaximumweightthatcancarryinthebagism. All values and weights are non negative.

Maximizethesumofthevaluesoftheitemsintheknapsack,sothatsumoftheweightsmust be less than the knapsack's capacity m.

Theformulacanbestatedas

$$\text{maximize} \sum_{1 \le i \le n} p_i x_i$$

$$\text{subject to} \sum_{1 \le i \le n} w_i x_i \le M$$

X_i**=0or1** $1 \le i \le n$

Tosolveo/1knapsack problemusingB&B:

- Knapsackisamaximizationproblem
- Replace the objective function $\sum p_i x_i$ by the function $-\sum p_i x_i$ to make it into a minimizationproblem
- Themodifiedknapsackproblemisstatedas

$$\text{Minimize} -\sum_{i=1}^{n} p_i x_i$$

$$\text{subject to} \sum_{i=1}^{n} w_i x_i < m,$$

$$x_i \in \{0,1\}, 1 \le i \le n$$

- Fixedtuplesizesolutionspace:
  - Every leaf node in state space tree represents an answer for which $\sum_{1 \le i \le n} w_i x_i \leqslant m$ isananswernode;otherleafnodesareinfeasible
  - Foroptimalsolution,define

$$c(x) = -\sum_{1 \le i \le n} p_i x_i$$

foreveryanswernodex

- For infeasible leaf nodes, $c(x) = \infty$
- Fornonleafnodes

c(x)=min{c(lchild(x)),c(rchild(x))}

- Definetwofunctionsc^(x)andu(x)suchthatforevery node x,

c^(x)≤c(x)≤u(x)

- Computingc^(·)andu(·)

Let x be a node at level j, $1 \leq j \leq n+1$
Cost of assignment: $-\sum_{1 \leq i < j} p_i x_i$
$c(x) \leq -\sum_{1 \leq i < j} p_i x_i$
We can use $u(x) = -\sum_{1 \leq i < j} p_i x_i$

Using $q = -\sum_{1 \leq i < j} p_i x_i$, an improved upper bound function $u(x)$ is

$$u(x) = \text{ubound}(q, \sum_{1 \leq i < j} w_i x_i, j-1, m)$$

```
Algorithmubound(cp,cw,k,m)
{
//Input:    cp:Currentprofittotal
//Input:    cw:Currentweighttotal
//Input:    k:  Indexoflastremoveditem
//Input:m:      Knapsackcapacity
b=cp;c=cw;
for i:=k+1 to n do{
     if(c+w[i]≤m)then{
          c:=c+w[i];b=b-p[i];
     }
}
returnb;
}
```

UNITV:

NP-HardandNP-Completeproblems: Basicconcepts,nondeterministicalgorithms,NP- Hard and NPComplete classes, Cook's theorem.

Basicconcepts:

NP,NondeterministicPolynomialtime

Theproblemshasbestalgorithmsfortheirsolutionshave"Computingtimes",thatcluster into two groups

Group1	Group2
> Problems withsolution time bound by a polynomial of a small degree. > Italsocalled"Tractable Algorithms" > MostSearching&Sortingalgorithmsare polynomial time algorithms > **Ex:** OrderedSearch**(O(logn)),** Polynomialevaluation**O(n)** Sorting **O(n.log n)**	> Problemswithsolutiontimesnotb ound by polynomial(simply non polynomial) > Thesearehardorintractablep roblems > Noneoftheproblemsinthisgrouphasbe ensolvedbyanypolynomial timealgorithm > **Ex:** TravelingSalesPerson**O($n^2 2^n$)** Knapsack**O($2^{n/2}$)**

Noone has beenabletodevelopapolynomialtimealgorithmforanyprobleminthe 2ndgroup (i.e., group 2)

So,itiscompulsoryandfindingalgorithmswhosecomputingtimesaregreaterthan polynomialveryquicklybecausesuchvastamountsoftimetoexecutethatevenmoderate size problems cannot be solved.

TheoryofNP-Completeness:

Showthatmayoftheproblemswithnopolynomialtimealgorithmsarecomputationaltime algorithmsarecomputationallyrelated.

Therearetwoclassesofnon-polynomialtimeproblems

1. NP-Hard
2. NP-Complete

NPCompleteProblem: AproblemthatisNP-Completecansolvedinpolynomialtimeif andonlyif(iff)allotherNP-Completeproblemscanalsobesolvedinpolynomialtime.

NP-Hard: Problemcanbesolvedinpolynomialtimethenall NP-Completeproblemscanbe solved in polynomial time.

AllNP-CompleteproblemsareNP-HardbutsomeNP-Hardproblemsarenotknowtobe NP-Complete.

NondeterministicAlgorithms:

Algorithmswiththepropertythattheresultofeveryoperationisuniquelydefinedaretermed asdeterministicalgorithms.Suchalgorithmsagreewiththewayprogramsareexecutedona computer.

Algorithmswhichcontainoperationswhoseoutcomesarenotuniquelydefinedbutare limitedtospecifiedsetofpossibilities.Suchalgorithmsarecallednondeterministic algorithms.

Themachineexecutingsuchoperationsisallowedtochooseanyoneoftheseoutcomes subjecttoaterminationconditiontobedefinedlater.

Tospecifynondeterministicalgorithms,thereare3newfunctions.

Choice(S), arbitrarily chooses one of the elements of sets S

Failure (), Signals an Unsuccessful completion

Success(), Signalsasuccessfulcompletion.

ExampleforNonDeterministicalgorithms:

Algorithm Search(x){
//Problemistosearchanelementx
//outputJ,suchthatA[J]=x;orJ=0ifxisnotinA
J:=Choice(1,n);
if(A[J]:=x)then{
Write(J);
Success();
}
else{
write(0);
failure();
	Whenever there is a set of choices thatleadstoasuccessfulcompletionthe n one such set of choices is always made and the algorithm terminates.
	A Nondeterministicalgorithm terminates unsuccessfully if and only if (iff) there exists no set of choices leading to a successful signal.

}	

NondeterministicKnapsackalgorithm	

AlgorithmDKP(p,w,n,m,r,x){
W:=0;
P:=0;
fori:=1tondo{
x[i]:=choice(0,1);
W:=W+x[i]*w[i];
P:=P+x[i]*p[i];
}
if((W>m)or(P<r))thenFailure();
elseSuccess();
}
	p,givenProfits
	w,givenWeights
	n,Numberofelements(numberof porw)
	m,Weightofbaglimit
	P,FinalProfit
	W,Finalweight

TheClassesNP-Hard&NP-Complete:

Formeasuringthecomplexityofanalgorithm,weusetheinputlengthastheparameter.For example,AnalgorithmAisofpolynomialcomplexityp()suchthatthecomputingtimeofA isO(p(n))foreveryinputofsize n.

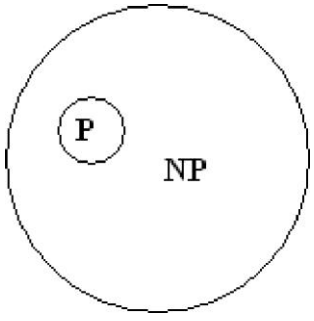
Decisionproblem/Decisionalgorithm: Anyproblemforwhichtheanswerisitherzeroor one is decision problem. Any algorithm for a decision problem is termed a decision algorithm.

Optimizationproblem/Optimizationalgorithm: Anyproblemthatinvolvesthe identificationofanoptimal(eitherminimumormaximum)valueofagivencostfunctionis knownasanoptimizationproblem.Anoptimizationalgorithmisusedtosolvean optimization problem.

P,isthesetofalldecisionproblemssolvablebydeterministicalgorithmsinpolynomial time.

NP,isthesetofalldecisionproblemssolvablebynondeterministicalgorithmsin polynomial time.

Sincedeterministicalgorithmsarejustaspecialcaseofnondeterministic,bythiswe concluded that **P ⊆ NP**



CommonlybelievedrelationshipbetweenP&NP

ThemostfamousunsolvableproblemsinComputerScienceisWhetherP=NPorP≠NP Inconsideringthisproblem,s.cookformulatedthefollowingquestion.

IfthereanysingleprobleminNP,suchthatifweshowedittobein'P'thenthatwould imply that P=NP.

Cookansweredthisquestionwith

Theorem:SatisfiabilityisinPifandonlyif(iff)P=NP

-)Notationof Reducibility

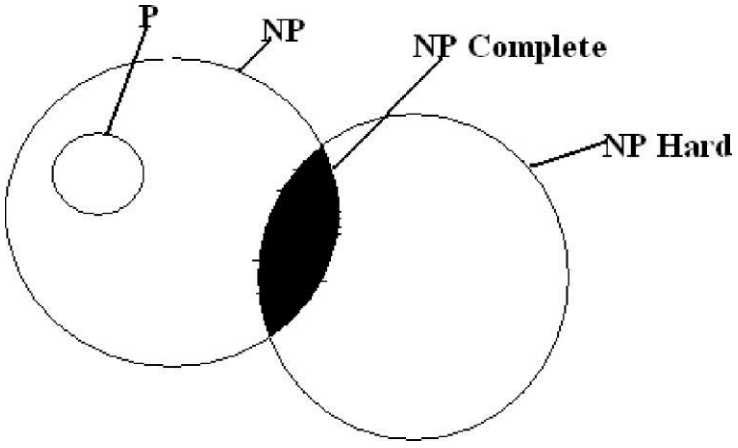
LetL_1andL_2beproblems,ProblemL_1reducestoL2(written$\mathbf{L_1 \alpha L_2}$)ifthereisawayto solve L_1 bya deterministic polynomialtimealgorithmusinga deterministic algorithmthatsolves L_2in polynomial time

Thisimpliesthat,ifwehaveapolynomialtimealgorithmforL_2,ThenwecansolveL1in polynomial time.

Hereα**-)**isatransitiverelationi.e.,$\mathbf{L_1 \alpha L_2}$and$\mathbf{L_2 \alpha L_3}$then$\mathbf{L_1 \alpha L_3}$

AproblemLisNP-Hardifandonlyif(iff)satisfiabilityreducestoLie.,**StatisfiabilityαL**

AproblemLisNP-Completeifandonlyif(iff)LisNP-Hardand**LЄNP**



Commonly believed relationship among P, NP, NP-Complete and NP-Hard

MostnaturalproblemsinNPareeitherinPorNP-complete.

ExamplesofNP-completeproblems:

- Packingproblems:SET-PACKING,INDEPENDENT-SET.
- Coveringproblems:SET-COVER,VERTEX-COVER.
- Sequencingproblems:HAMILTONIAN-CYCLE,TSP.
- Partitioningproblems:3-COLOR,CLIQUE.
- Constraintsatisfactionproblems:SAT,3-SAT.
- Numericalproblems:SUBSET-SUM,PARTITION,KNAPSACK.

Cook's Theorem: StatesthatsatisfiabilityisinP if andonlyifP=NP If P=NP then satisfiability is in P
IfsatisfiabilityisinP,thenP=NP To do this

> A-) Anypolynomialtimenondeterministicdecisionalgorithm.

I-) Inputofthatalgorithm

Then formula Q(A,I), Such that Qis satisfiableiff'**A'** has a successful termination with Input **I**.

> Ifthelengthof'**I**'is'**n'**andthetimecomplexityof**A**is**p(n)**forsomepolynomial **p()**thenlengthofQis**$O(p^3(n)logn)=O(p^4(n))$**

ThetimeneededtoconstructQisalso**$O(p^3(n)logn)$.**

> Adeterministicalgorithm'**Z'**todeterminetheoutcomeof'**A**'onanyinput'**I**'

AlgorithmZcomputes'**Q**'andthenusesadeterministicalgorithmforthesatisfiabilityproblemtodeterminewhether'**Q**'issatisfiable.

> If**O(q(m))**isthetimeneededtodeterminewhetheraformulaoflength'**m**'issatisfiablethenthecomplexityof'**Z**'is**$O(p^3(n)logn+q(p^3(n)logn))$.**

> Ifsatisfiabilityis'**p**',then'**q(m)**'isapolynomialfunctionof'**m**'andthe complexity of '**Z**' becomes '**O(r(n))'** for some polynomial '**r()'.**

> Hence, if satisfiability is in **p**, then for every nondeterministic algorithm **A** in **NP**, we can obtain a deterministic **Z** in **p.**

Bythisweshowsthatsatisfiabilityisin**p**then**P=NP**