CS8451- DESIGN AND ANALYSIS OF ALGORITHMS

UNIT-I

Notion of an Algorithm – Fundamentals of Algorithmic Problem Solving – Important Problem Types – Fundamentals of the Analysis of Algorithm Efficiency – Analysis Framework – Asymptotic Notations and its properties – Mathematical analysis for Recursive and Non-recursive algorithms.

PART –A

1.Differentiate Time Complexity from Space Complexity.(Apr 2010/Nov 2012)

- Time efficiency, also called **time complexity** of an algorithm is the amount of computer time it needs to run to completion.
- Space efficiency, also called **space complexity** of an algorithm is the amount of memory it needs to run to completion.

2.What is a Recurrence Equation? (Apr 2010)

The recurrence equation is an equation that defines a sequence recursively.

It is normally in following form-

T(n)=T(n-1)+n for n>0 .....(1)

T(0)=0 .....(2)

Here equation 1 is called recurrence relation and equation 2 is called initial condition.

The recurrence equation can have infinite number of sequences.

Recurrence equation can be solved using

- Substitution method
- Master's method.

3.Define Big 'Oh' notation. (Apr/May 2012) (Apr/May 2013)

A function t (n) is said to be in O(g(n)), denoted t (n) ∈ O(g(n)), if t (n) is bounded above by some constant multiple of g(n) for all large n, i.e., if there exist some positive constant c and some nonnegative integer n_0 such that

$t(n) \leq cg(n)$ for all $n \geq n_0$.

4.What do you mean by algorithm (Apr/May 2013)

An algorithm is a sequence of unambiguous instructions for solving a problem in a finite amount of time. In addition, all algorithms must satisfy the following criteria:

1. Input
2. Output
3. Definiteness
4. Finiteness
5. Effectiveness.

5. Define theta notation (Nov/Dec 2014)

A function t (n) is said to be in _(g(n)), denoted t (n) _(g(n)), if t (n) is bounded both above and below by some positive constant multiples of g(n) for all large n, i.e., if there exist some positive constants c1 and c2 and some nonnegative integer n0 such that c2g(n) ≤ t (n) ≤ c1g(n) for all n ≥ n0.

6.What is meant by substitution method (Nov/Dec 2014)

One of the methods for solving recurrence relation is called the substitution method. There are two types of substitution

- Forward substitution
- Backward substitutions

7.Write an algorithm to find the number of binary digits in the binary representation of a positive decimal integer? (Apr/May 2015)

The following algorithm finds the number of binary digits in the binary representation of a positive decimal integer.

ALGORITHM Binary(n)

//Input: A positive decimal integer n

//Output: The number of binary digits in n's binary representation

count ←1

while n > 1 do

count ←count + 1

n←_n/2_

return count

8.Write down the properties of asymptotic notations? (Apr/May 2015)

If $t_1(n) \in O(g1(n))$ and $t_2(n) \in O(g_2(n))$, then

$t_1(n) + t_2(n) \in O(\max\{g_1(n), g_2(n)\})$.

This property implies that the algorithm's overall efficiency is determined by the part with a higher order of growth, i.e., its least efficient part:

$t_1(n) \in O(g_1(n))$

$t_1(n) + t_2(n) \in O(\max\{g_1(n), g_2(n)\})$.

$t_2(n) \in O(g_2(n))$

9.Write the recursive Fibonacci algorithm and its recurrence relation (Nov/Dec 2015)

ALGORITHM F(n)

- //Computes the nth Fibonacci number recursively by using its definition
- //Input: A nonnegative integer n
- //Output: The nth Fibonacci number

if n ≤ 1 return n

else return F(n − 1) + F(n − 2)

Recurrence Relation

A(n) = A(n − 1) + A(n − 2) + 1 for n > 1,

A(0) = 0, A(1) = 0.

The recurrence A(n) − A(n − 1) − A(n − 2) = 1 is quite similar to recurrence, but its right-hand side is not equal to zero. Such recurrences are called **inhomogeneous recurrences**.

10. Define Algorithm.

An algorithm is a sequence of unambiguous instructions for solving a problem in a finite amount of time.

11. Write about Notion of Algorithm.

Problem

Algorithm

Input Computer Output

12. Write a short note on Algorithm Design and Analysis of Process.

- Understand the problem
- Decide on
- Computational Device
- Exact Vs Approximate Algorithms
- Data Structures
- Algorithm Design Techniques
- Design an algorithms
- Prove Correctness
- Analyze the Algorithm
- Code the Algorithm

13. What are the 2 kinds of Algorithm Efficiency?

Time Efficiency-How fast your algorithm runs?

Space Efficiency-How much extra memory your algorithm needs?

14. How can you specify Algorithms?

Algorithms can be specified in a natural language or pseudo code.

15. What is Pseudo Code?

Pseudo Code is a mixture of Natural Language and Programming Language Constructs such as functions, loops, decision making statements..etc

16. What are the Important Problem Types?

- Sorting
- Searching
- String Processing
- Graph Problem
- Cominatorial Problem
- Geometric Problem
- Numerical Problem

17. How can you Classify Algorithms

Among several ways to classify algorithms, the 2 principal alternatives are

- To group algorithms according to types of problem they solve
- To group algorithms according to underlying design techniques they are based upon

18. Write the Euclid Algorithm

Algorithm Euclid(m,n)

Step 1: While n not equal do Step 2: r = m mod r

Step 3: m=n

Step 4: n=r Step 5: return n

19. What is Sorting Problem?

Sorting algorithm is rearrange the items of a given list in descending/ascending order. Sorting algorithms classified into

- Stable Sorting Algorithm
- Non-Stable Algorithm

20. What is Searching Problem?

Finding a given value, called search key in a given set. Searching Algorithms needs more memory space and sorted array.

21. What is Graph Problem?

Graph is a collection of edges and vertices. G=(V,E). For eg. Traversal Algorithms, Shortest Path Algorithm, Graph Coloring Problem

22. What is Combinatorial Problem?

This problem that ask to find a combinatorial object such as permutations, combinations or a subset. Combinatorial problems are most difficult to solve. For eg Travelling sales man problem

23. Differentiate Time Efficiency and Space Efficiency?

Time Efficiency measured by counting the number of times the algorithms basic operation is executed. Space Efficiency is measured by counting the number of extra memory units consumed by the algorithm.

24. What are the features of efficient algorithm?

Free of ambiguity
Efficient in execution time
Concise and compact
Completeness
Definiteness
Finiteness

25. Define Order of Algorithm

The order of algorithm is a standard notation of an algorithm that has been developed to represent function that bound the computing time for algorithms.

The order of an algorithm is a way of defining its efficiency. It is usually referred as O-notation

26. Define Big Omega Notation.

Omega notation provides a lower bound for the function t..

A function t(n) is said to in Omega (g(n)), if there exist some positive constant C and some non negative integer N0, such that t(n)>=Cg(n) for all n>=n_0

27. What are the different types of time complexity?

The time complexity can be classified into 3 types, they are

- Worst case analysis
- Average case analysis
- Best case analysis

28. How the algorithm's time efficiency is measured.

Time efficiency indicates how fast an algorithm runs. Time taken by a program to complete its task depends on the number of steps in an algorithm.

The time taken by an algorithm is the sum of compile time and execution time. The compile time does not depend on the instance characteristics

29. Define *O*-Notation.

A function t(n) is said to be in ***O*** (g(n)), denoted t(n) Є ***O*** (g(n)), if t(n) is bounded above by some constant multiple of g(n) for all large n, ie ., if there exist some positive constant c and some nonnegative integer n0 such that t(n) <= cg(n) for all n >= n0

30. What is Fibonacci Numbers?

The Fibonacci numbers are an important sequence of integers in which every element is equal to the sum of its two immediate predecessors. There are several algorithms for computing the Fibonacci numbers with drastically different efficiency.

31. What is recursive call?

An algorithm is said to be recursive if the same algorithm invoked in the body.

There are 2 types of algorithm. They are

1 Direct Recursive

2 Indirect Recursive

32. What is meant by Direct Recursive call?

An algorithm that calls itself is direct recursive call.

```
Eg.     int fun(int x)
{
if(x<=0)
return x; return (fun(x-1));
}
```

33. Define indirect recursive call?

Algorithm A is said to be indirect recursive if it calls another algorithm which in turn call A

Example:

```
Void main()
{
int fun(int x) if(x<=0)
return x; return (fun1(x-1));
}
int fun1(int y)
{
return fun(y-1)
}
```

34. **Give the Euclid's algorithm for computing gcd(m, n) (MAY / JUNE 2016)**

ALGORITHM Euclid_gcd(m, n)

//Computes gcd(m, n) by Euclid's algorithm

//Input: Two nonnegative, not-both-zero integers m and n

//Output: Greatest common divisor of m and n

while n ≠ 0 do

r ←m mod n

m←n

n←r

return m

Example: gcd(60, 24) = gcd(24, 12) = gcd(12, 0) = 12.

35.Compare the order of growth n(n-1)/2 and n2. (MAY / JUNE 2016)

n	$n(n-1)/2$	n^2
Polynomial	**Quadratic**	**Quadratic**
1	0	1
2	1	4
4	6	16
8	28	64
10	45	10^2
10^2	4950	10^4
Complexity	Low	High
Growth	Low	high

$n(n-1)/2$ is lesser than the half of n^2

PART-B

1.Write the Insertion sort algorithm and estimate its running time. 8 marks (Nov 2015)

Algorithm:

Input: sequence ⟨a1, a2, …, an⟩ of numbers.

Output: permutation ⟨a'1, a'2, …, a'n⟩ such that a'1 ≤ a'2 ≤ … ≤ a'n .

INSERTION-SORT (A, n) ▷ A[1 . . n]

for j ← 2 to n
do key ← A[j]
i ← j – 1
while i > 0 and A[i] > key
do A[i+1] ← A[i]
i ← i – 1
A[i+1] = key

INSERTION-SORT (A, n) ▷ A[1 . . n]

for j ← 2 to n

do key ← A[j]

i ← j – 1

while i > 0 and A[i] > key

do A[i+1] ← A[i]

i ← i – 1

A[i+1] = key

Running time:

• The running time depends on the input: an already sorted sequence is easier to sort.

• Parameterize the running time by the size of the input, since short sequences are easier to sort than long ones.

• Generally, we seek upper bounds on the running time, because everybody likes a guarantee.

Analysis:

Worst-case: (usually) • T(n) = maximum time of algorithm on any input of size n. Average-case: (sometimes) • T(n) = expected time of algorithm over all inputs of size n. • Need assumption of statistical distribution of inputs. Best-case: (bogus) • Cheat with a slow algorithm that works fast on some input.

2. Find the closest asymptotic tight bound by solving the recurrence equation T(n)=8T(n/2)+n2 with (T(1)=1) using Recursion tree method. [Assume that T(1)∈Θ(1)]. 8 marks (Nov/Dec 2015)

Solve: T(n) = 8T(n/2) + n 2 with(T(1) = 1)

T(n) = n 2 + 8T(n/2)

= n 2 + 8(8T(n 2 2) + (n 2) 2)

= n 2 + 82T(n 2 2) + 8(n 2 4))

= n 2 + 2n 2 + 82T(n 2 2)

= n 2 + 2n 2 + 82 (8T(n 2 3) + (n 2 2) 2)

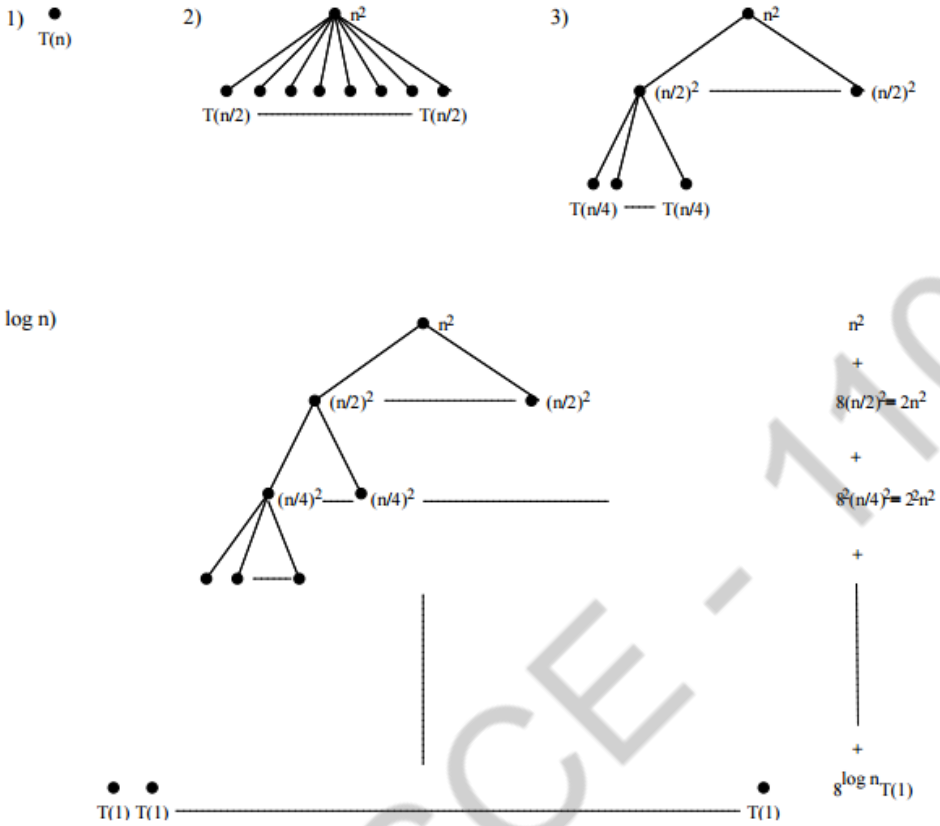
= n 2 + 2n 2 + 83T(n 2 3) + 82 (n 2 4 2))

= n 2 + 2n 2 + 22n 2 + 83T(n 2 3)

= . . .

= n 2 + 2n 2 + 22n 2 + 23n 2 + 24n 2 + . . .

Recursion tree method:



3. Show how to implement a stack using two queues. Analyze the running time of the stack operations. 10 marks (Nov/Dec 2015)

The pseudocode is as follows:

```
Public class Twoqueuestack
{
Queue q1,q2;
int flag=0;
//0:the stack is empty;
//1: all data are stored in queue q1;
//2: all data are stored in queue q2;
//Always push into the empty queue,then move all data from the other queue to the current queue
//Always pop element from the queue containing data.
//Suppose q1 and q2 are implemented by linked list.The queues never get full.
public void push(Object e)
{
switch(flag)
{
case 0: q1.enQueue(e);
          Flag = 1;
          break;
case 1: q2.enqueue(e);
        While (!q1.isEmpty())
           q2.enQueue(q1.deQueue());
        Flag= 2;
        break;
case 2: q1.enQueue(e);
         While (!q2.isEmpty())
           q1.enQueue(q2.deQueue());
        Flag= 1;
        break;
default: error 'illegal state';
```

```
}
}
Public Object pop()
{
switch(flag)
{
case 0:error 'underflow—stack is empty,can't pop';
case 1: retElemnent =q1.deQueue();
        if (q1.isEmpty())
           flag= 0;
       return retElement;
       break;
case 2: retElemnent =q2.deQueue();
        if (q2.isEmpty())
           flag= 0;
       return retElement;
       break;
default: error 'illegal state';
}
}
}
```

Running time:

The running time for push operation is O(n).

The running time for pop operation is θ(1)

4. Derive the worst case analysis of merge sort using suitable illustrations 8 marks (Apr/ May 2015)

Let T(n) to be denote the worst case of running time of merge sort the time taken by this algorithm to sort an array of n elements dividing A into sub array B and C.

$T(n)=T(n/2)+T(n/2)+C_n$

$T(n)=2T(n/2)+Cn$

T(n) : time required for sort ,

T(n/2) : time required for sorting left sub list and right sub list ,

Cn : time required for combining

Recurrence for Merge sort

Input size: n- size of array

Basic operation: diving and Merging

Recurence equation:

$\mathbf{T}(n) = 2T(n/2)+Cn$ --------→1

$T(1)=0$

$n=n/2$

$\mathbf{T}(n) = 2T(n/2*2)+C(n/2)$

$T(n)= 2T(n/4)+ C(n/2)$----------→2

Sub 2 and 1

$T(n) = 2 [2T(n/4)+C(n/2)] + Cn$

$T(n) = 4T(n/4)+2C(n/2)] + Cn$ [stick 2/2C(n) =Cn]

$T(n) = 4T(n/4)+Cn + Cn$

$T(n) = 4T(n/4)+2Cn$ ----------→3

$n = n/4$

$\mathbf{T}(n) = 2T(n/4*2)+C(n/4)$

$= 2T(n/8)+C(n/4)$----------→4

Sub 4 in 3

$T(n) = 4[2T(n/8)+C(n/4)] +2Cn$

$T(n) = 8T(n/8)+4C(n/4) +2Cn$ [stick 4/4Cn =Cn]

$T(n) = 8T(n/8)+Cn +2Cn$

$T(n) = 8T(n/8)+3Cn$

$T(n) = 2^3T(n/2^3)+3Cn$

Assign k=3

$T(n) = 2^k T(n/2^k)+kCn$

Assign n= 2^k

$T(n) = nT(n/n)+kCn$ [stick $T(n/n)=T(1)$]

$T(n) = nT(1)+k\,Cn$ [ignore $nT(1)$ this term because it is small]

$T(n) = k\,Cn$

Take log on both side of

$n= 2^k$

$\log n = \log_2 k$

$\log n = k \log_2$ [$\log_2=1$]

$\log n = \mathbf{k}$

$T(n) = \log n.\, Cn$

$T(n) = \log n\, .n$

$T(n) = n \log n$

T(n) = O(n logn)

Master method:

$C(n) = 2C(n/2)+Cn$ a=2 , b=2 , d=n [d=cn]

$T(n) = n^{\log_b a}$

$= n^{\log_2 2}$

= n Vs n [n =n]

$C(n) = \Theta(n^d \log n)$ [d=1]

$C(n) = \Theta(n \log n)$

All cases have same efficiency: $\Theta(n \log n)$

Number of comparisons in the worst case is close to theoretical minimum for comparison-based sorting.

5. Give the definition and Graphical Representation of O-Notation. (8) *(*MAY 2016)

Asymptotic notation is a notation, which is used to take meaningful statement about the

efficiency of a program.

The efficiency analysis framework concentrates on the order of growth of an algorithm's basic operation count as the principal indicator of the algorithm's efficiency.

To compare and rank such orders of growth, computer scientists use three notations, they

are:

- O - Big oh notation
- Ω - Big omega notation
- Θ - Big theta notation

Let *t(n)* and *g(n*) can be any nonnegative functions defined on the set of natural numbers.

The algorithm's running time *t(n)* usually indicated by its basic operation count *C(n),* and *g(n),*

some simple function to compare with the count.

O - Big oh notation

A function t*(n)* is said to be in *O(g(n))*, denoted ( ) ∈ (( )), if *t (n)* is bounded above

by some constant multiple of *g(n)* for all large *n,* i.e., if there exist some positive constant *c* and

some nonnegative integer *n*0 such that

( ) ≤ ( ) ≥ 0.

Where *t(n)* and *g(n*) are nonnegative functions defined on the set of natural numbers.

O = Asymptotic upper bound = Useful for worst case analysis = Loose bound

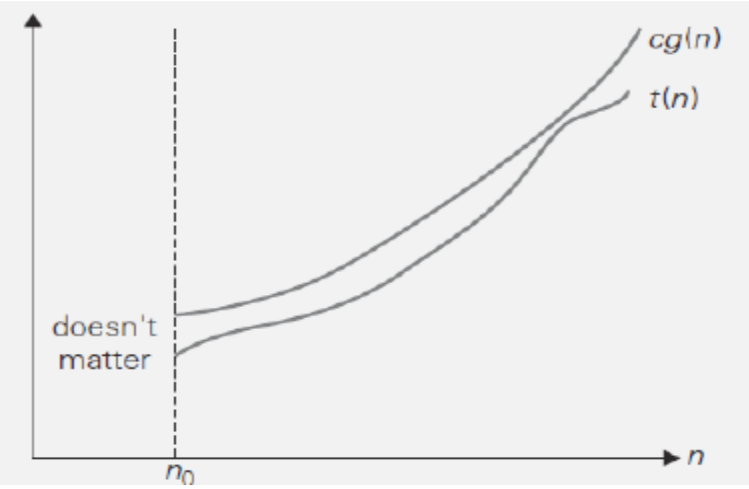


FIGURE Big-oh notation: ( ) ∈ (( )).

Example 1: Prove the assertions 100 + 5 ∈ (2).

Proof: $100n + 5 \leq 100n + n$ (for all $n \geq 5$)

$= 101n$

$\leq$ 101n2 (∵ ≤ 2)

Since, the definition gives us a lot of freedom in choosing specific values for constants **c** and **n0**. We have c=101 and n0=5

Example 2: Prove the assertions 100 + 5 ∈ ( ).

Proof: $100n + 5 \leq 100n + 5n$ (for all $n \geq 1$)

$= 105n$

i.e., $100n + 5 \leq 105n$

i.e., t(n) ≤ cg(n)

∴100 + 5 ∈ with c=105 and n0=1

6. Give an algorithm to check whether all the Elements in a given array of n element sare distinct. Find the worst case complexity of the same. (8) (MAY 2016)

Element uniqueness problem: check whether all the Elements in a given array of n elements are distinct.

ALGORITHM UniqueElements(A[0..n − 1])

//Determines whether all the elements in a given array are distinct

//Input: An array A[0..n − 1]

//Output: Returns "true" if all the elements in A are distinct and "false" otherwise

for i ←0 **to** n − 2 **do**

for j ←i + 1 **to** n − 1 **do**

if A[i]= A[j] **return false**

return true

Worst case Algorithm analysis

- The natural measure of the input's size here is again n (the number of elements in the array).

- Since the innermost loop contains a single operation (the comparison of two elements), we should consider it as the algorithm's basic operation.
- The number of element comparisons depends not only on n but also on whether there are equal elements in the array and, if there are, which array positions they occupy.
- We will limit our investigation to the worst case only**.**
- One comparison is made for each repetition of the innermost loop, i.e., for each value of the loop variable j between its limits i + 1 and n − 1; this is repeated for each value of the outer loop, i.e., for each value of the loop variable i between its limits 0 and n − 2.

$$C_{worst}(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} [(n-1) - (i+1) + 1] = \sum_{i=0}^{n-2} (n-1-i)$$
$$= \sum_{i=0}^{n-2} (n-1) - \sum_{i=0}^{n-2} i = (n-1) \sum_{i=0}^{n-2} 1 - \frac{(n-2)(n-1)}{2}$$
$$= (n-1)^2 - \frac{(n-2)(n-1)}{2} = \frac{(n-1)n}{2} \approx \frac{1}{2} n^2 \in \Theta(n^2).$$

7. Give the recursive algorithm which finds the number of binary digits in the binary representation of a positive decimal integer. Find the recurrence relation and complexity. (16) (MAY 2016)

An investigation of a recursive version of the algorithm which finds the number of binary

digits in the **binary representation** of a positive decimal integer.

ALGORITHM :

BinRec(n)

//Input: A positive decimal integer n

//Output: The number of binary digits in n's binary representation

if $n = 1$ **return** 1

else return *BinRec*(n/2)+ 1

Algorithm analysis:

The number of additions made in computing *BinRec*(n/2) is A(n/2), plus one more addition is made by the algorithm to increase the returned value by 1.

This leads to the recurrence

$A(n) = A($ n/2 $) + 1$ for $n > 1$.

Since the recursive calls end when n is equal to 1 and there are no additions made then, the initial condition is $A(1) = 0$.

The standard approach to solving such a recurrence is to solve it only for $n = 2k$

A(2k) = A(2k−1) + 1 for k > 0,

A(20) = 0.

backward substitutions

A(2k) = A(2k−1) + 1 substitute A(2k−1) = A(2k−2) + 1

= [A(2k−2) + 1]+ 1= A(2k−2) + 2 substitute A(2k−2) = A(2k−3) + 1

= [A(2k−3) + 1]+ 2 = A(2k−3) + 3 . . . . .

= A(2k−i) + i

= A(2k−k) + k.

Thus, we end up with A(2k) = A(1) + k = k, or, after returning to the original variable n = 2k and hence k = log2 n,

A(n) = log2 n ϵ Θ (log2 n).

UNIT II

BRUTE FORCE AND DIVIDE-AND-CONQUER 9

Brute Force - Closest-Pair and Convex-Hull Problems-Exhaustive Search - Traveling Salesman Problem - Knapsack Problem - Assignment problem. Divide and conquer methodology – Merge sort – Quick sort – Binary search – Multiplication of Large Integers – Strassen's Matrix Multiplication-Closest-Pair and Convex-Hull Problems.

1. Define Convex-Hull Problem.

A set of points(finite or infinite) on the plane is called convex if for any two points P and Q in the set, the entire line segment with the end points at P and Q belongs to the set**.**

2. Define closest-pair problem?

The closest-pair problem calls for finding the two closest points in a set of *n* points.

It is the simplest of a variety of problems in computational geometry that deals with proximity of points in the plane or higher-dimensional spaces.

One of the important applications of the closest-pair problem is cluster analysis in statistics.

3. What is an extreme point?

An ***extreme point*** of a convex set is a point of this set that is not a middle point of any line segment with endpoints in the set.

For example, the extreme points of a triangle are its three vertices, the extreme points of a circle are all the points of its circumference.

4. Define binary search Tree?

A binary search tree t is a binary tree, either it is empty or each node in the tree contains an identifier and

1) all identifiers in the left subtree of t are less than the identifier in the root node t.

2) all identifier in the right subtree of t are greater than the identifier in the root node t.

3) the left and right subtrees of t are also binary search trees.

5. Explain the principle used in quick sort?

It is a partition method using the particular key the given table is partitioned into 2 subtables so that first, the original key will be its proper position in the sorted sequence and secondly, all keys to the left of this key will be less in value and all keys to the right of it will be greater in values

6. What is binary search?

The binary search algorithm is one of the most efficient searching techniques which requires the list to be sorted in ascending order.

To search for an element in the list, the binary search algorithms split the list and locate the middle element of the list.

First compare middle key K1, with given key K

If K1=K then the element is found.

7. Explain principle of Optimality?

The principle of optimality says that an optimal solution to any instance of an optimization problem is composed of optimal solution to its subinstances.

8. What is Divide and Conquer Algorithm

It is a general algorithm design techniques that solved a problem's instance by dividing it into several smaller instance, solving each of them recursively, and then combining their solutions to the original instance of the problem.

9. What is Brute Force?

Brute Force is a straightforward approach to solving problem, usually directly based on the problem's statement and definitions of the concepts involved..

10. What are the different criteria used to improve the effectiveness of

algorithm?

Input- Zero or more quantities are externally supplied

Output-At least one quantity is produced

Definiteness-Each instruction is clear and unambiguous

Finiteness-If we trace out the instructions of an algorithm, then for all case the algorithm terminates after a finite number of steps

Effectiveness-Every instruction must be very clear.

11. What are the objectives of sorting algorithm?

To rearrange the items of a given list

To search an element in the list

12. Why is bubble sort called by the name?

The sorting problem is to compare adjacent elements of the list and exchange them if they are out of order. By doing it repeatedly, we end up bubbling up the largest element to the last position on the list. The next pass bubbles up the second largest element and so on until, after n-1 pass the list is sorted.

13. What are the 3 variations in transform and conquer?

The principle variations of transform and conquer techniques are

Instance Simplification

Representation Change

Problem Reduction

14. Explain principle of Optimality?

The principle of optimality says that an optimal solution to any instance of an optimization problem is composed of optimal solution to its subinstances.

15. What is the Quick sort?

In quicksort, the division into subarrays is made so that the sorted subarrays do not need to be merged later.

16. Write the Anlysis for the Quick sot.

In analyzing QUICKSORT, we can only make the number of element comparisions c(n). It is easy to see that the frequency count of other operations is of the same order as C(n).

17. Is insertion sort better than the merge sort?

Insertion sort works exceedingly fast on arrays of less then 16 elements, though for large „n" its computing time is $O(n^2)$.

18. Write a algorithm for straightforward maximum and minimum

```
Algorithm straight MaxMin(a,n,max,min)
//set max to the maximum and min to the minimum of a[1:n]
{
max := min: = a[i]; for i = 2 to n do
{
if(a[i] >max) then max: = a[i]; if(a[i] >min) then min: = a[i];
}
```

19 Give the recurrence relation of divide-and-conquer?

The recurrence relation is

$$T(n) = \begin{cases} g(n) \\ T(n_1) + T(n_2) + \text{----+ } T(n_k) + f(n) \end{cases}$$

20 Explain the principle used in quick sort?

It is a partition method using the particular key the given table is partitioned into 2 subtables so that first, the original key will be its proper position in the sorted sequence

and secondly, all keys to the left of this key will be less in value and all keys to the right of it will be greater in values.

21. Write the algorithm for Iterative binary search?

```
Algorithm BinSearch(a,n,x)
//Given an array a[1:n] of elements in nondecreasing // order, n>0, determine whether x is present
{
low : = 1; high : = n;
while (low <  high) do
{
mid : = [(low+high)/2];
if(x            a[mid]) then high:= mid-1;

else if (x   a[mid]) then low:=mid
+ 1; else return mid;
}
return 0;
}
```

22. What are internal nodes?

- The circular node is called the internal nodes.
- "Internal node in a Binary Tree or any Tree is the node which has at least one child.
- Basically the node which is not a leaf node is called internal node."

23. List the advantages of binary search?

Less time is consumed

The processing speed is fast

The number of iterations is less. It takes n/2 iterations.

Binary search, which is used in Fibonacci Series, involves addition and subtraction rather than division. It is priori analysis, since it can be analyzed before execution.

24. Define binary search Tree?

A binary search tree t is a binary tree, it is empty or each node in the tree contains an identifier and all identifiers in the left subtree of t are less than the identifier in the root node t. all identifier in the right subtree of t are greater than the identifier in the root node t.the left and right subtrees of t are also binary search trees.

25. Give the General strategy divide and conquer method. (MAY / JUNE 2016)

A **divide and conquer algorithm** works by recursively breaking down a problem into two

or more sub-problems of the same (or related) type (**divide**), until these become simple enough to be solved directly (**conquer**).

Divide-and-conquer algorithms work according to the following general plan:

1. A problem is divided into several subproblems of the same type, ideally of about equal

size.

2. The subproblems are solved (typically recursively, though sometimes a different algorithm is employed, especially when subproblems become small enough).

3. If necessary, the solutions to the subproblems are combined to get a solution to the original problem.

Example: Merge sort, Quick sort, Binary search, Multiplication of Large Integers and Strassen's Matrix Multiplication.

26.What is Closest-Pair Problem? (MAY / JUNE 2016)

The closest-pair problem finds the two closest points in a set of n points. It is the simplest of a variety of problems in computational geometry that deals with proximity of

points in the plane or higher-dimensional spaces.

27.Design a brute-force algorithm for computing the value of a polynomial $P(x)=a_n x^n+a_{n-1}x^{n-1}+\ldots\ldots+a_1 x\ldots..a_0$ at a given point x_0 and determine its worst-case efficiency class (Apr/May 2015)

The number of time the basic operation is executed c(n)=

$$c(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1$$

$\sum_{j=i+1}^{n-1} 1$ = (n-1)-(i+1)+1 [Where (n-1) upperbound, (i+1) lowebound)]

= n-1-i-1+1

= n-i-1

$c(n) = \sum_{i=0}^{n-2} n-i-1$ [n-i+1, i=0,n-0-1=n-1]

i=1, n-1-1= n-2

i=2 ,n-2-1=n-3

i=n-2, n-(n-2)-1=n-n+2-1=1] = (n-1)+(n-2)+(n-3)+.....+1

= ((n-1).n)/2

$$c(n) = \sum_{i=0}^{n-2} i = ((n-1).n)/2$$

$=n^2/2 - n/2$ [eliminate constant 2]

$C(n)=n^2-n$

$C(n) = O(n^2)$

selection sort is a O(n2 algorithm on all inputs.

28.Derive the complexity of Binary Search algorithm (Apr/May 2015)

Best case: 1 comparision

$\Omega(1)$

Worst case: keep reducing the search the space n , n/2,n/4… till 1

$C_{worst}(n)=C_{worst}(n/2)+1$

Worst case: O(logn)

Average case: ∅(logn)

29.Give the mathematical notation to determine if a convex direction is towards left or right and write the algorithm (Nov/Dec 2015)

Algorithm: Quickhull(P1,p2)

P1 & p2 are extreme points (admit vertical supporting lines)

S_1 ← points to the right of p1p2

S_2 ← points to the right of p2p1

FindHull (S_1,A,B)

FindHull(S_2, B,A)

Algorithm: FindHull(P,A,B):

- If P is empty, then return.
- Find (extreme) point $C \in P$ orthogonally farthest from AB.
- CH update: replace AB with AC followed by CB. (inflate baloon)
- Partition P-{C} into Q_0, Q_1, Q_2 as shown.
- Discard Q_0 inside triangle ABC (by Caratheodory).
- Recursively call FindHull(Q_1,A,C) and FindHull(Q_2,C,B).

30.Prove that any comparison sort algorithm requires Ω(n log n) comparisons in the worst case (Nov/Dec 2015)

C(n) = 2C(n/2)+Cn a=2 , b=2 , d=n [d=cn]

T(n) = $n^{\log_b a}$

= $n^{\log_2 2}$

= n Vs n [n =n]

C(n) = Θ(n^d log n) [d=1]

C(n) = Θ(n log n)

- All cases have same efficiency: Θ(n log n)
- Number of comparisons in the worst case is close to theoretical minimum for comparison-based sorting.

PART-B

1. Explain the convex hull problem and the solution involved behind it 6 marks (Apr/May 2015)

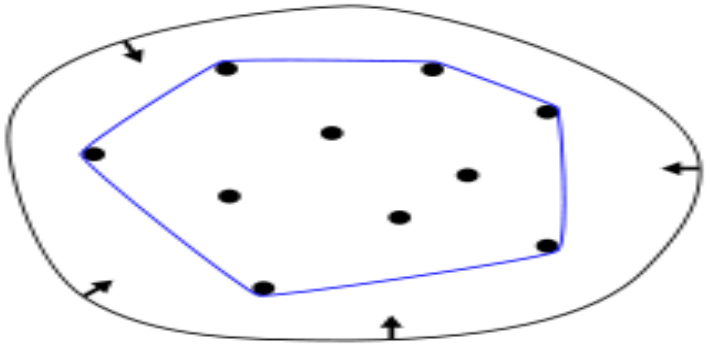
DEFINITION: Convex Hull

- A set of points in the plane is called convex.
- **Convex set:** for any two points p and q in the set, the entire line segment with the endpoints at p and q belongs to the set.

convex **Not convex**

The convex hull: of a set S of points is the smallest convex set containing S.

- The "smallest" requirement means that the convex hull of S must be a subset of any convex set containing S.



input = set of points: p1, p2, p3, p4, p5, p6, p7, p8, p9,p10

output = representation of the convex hull: p4, p5, p8, p2, p9,p10

convex polygon: sequence of line segments that ends where it begins

THEOREM:

- The convex hull of any set **S of n > 2** points not all on the same line is a convex polygon with the vertices at some of the points of S.
- Vertices of the polygon are called extreme points.
- We need to know which pairs of points need to be connected.

Convex hull problem:

The problem is to construct the boundary of convex hull in two dimension given a finite set of n points.

Algorithm:

//**INPUT:** set P of 2D points

//Four types of output can be obtained for above problem:

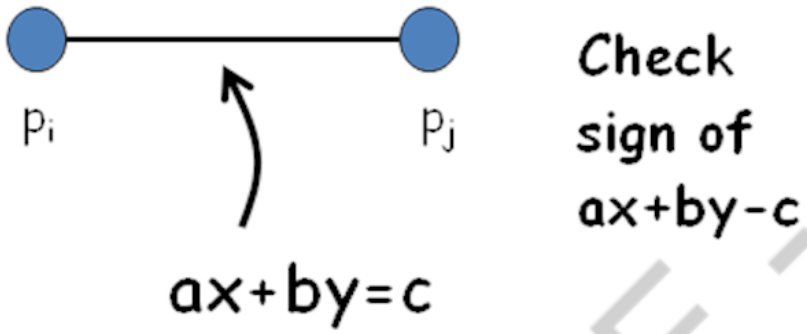
1. All the points on the hull, in arbitrary order
2. The extreme points, in arbitrary order
3. All the points on the hull, in boundary traversal order

4. The extreme points, in boundary traversal order

- For each (distinct) pair of points in the set, compute a, b, and c to define the line ax + by = c.
- For each other point, plug its x and y coordinates into the expression ax + by – c.
- If they all have the same sign (all positive or all negative), then this pair of points is part of the convex hull.

How can we solve the convex hull problem?

P_i and p_j are on the boundary if and only if all other points lie at the same side of the line segment between p_i and p_j



- line passing through (x_i,y_i) and (x_j,y_j) is:

  ax + by = c `

The equation of the line l through pi and pj is given by

$$\frac{x-x1}{x2-x1} = \frac{y-y1}{y2-y1}$$

from which we derive

a = (y2 −y1)

b = −(x2 −x1)

c = y1(x2 −x1)−x1(y2 −y1).

The time efficiency of this algorithm $O(n^3)$

The above mention line divides the plane into two half planes:

- For all the points in one of them, ax +by>c
- For all the points in the other ax + by <c

Algorithm:

for each of n(n-1)/2 pairs of distinct points

for each of the other n – 2 points

find the sign of ax + by – c

This is the simplest brute force approach to solve this problem.

2. Write the algorithm to perform Binary Search and compute its run time complexity 8 marks (Nov/Dec 2015)

- Binary search is a remarkably efficient algorithm for searching in a sorted array.
- Given a sorted array of integers that the elements in the array are arranged either increasing or decreasing orders.
- To find the given element x is present or not.
- Suppose x present, to determine the position of x.
- Suppose X is not present, just set zero

Steps:

Select the middle element from the array list. X element to be searched in the list.

Three cases:

Case 1: **x== A[mid]** [When x is equal to the middle element, discarded remaining elements

Case 2: **x < A[mid]** [When x is less than the middle element, discarded remaining elements

Case 3:**x >A[mid]** [When x is greater than the middle element, discarded remaining elements]

Finding a middle element:

Mid= (start+end)/2

Algorithm: Binarysearch(A[0…n-1],x,n)

```
//Input: An array A[0…n-1]sorted in ascending order and search x
//Output: An index of the array element is equal to x or -1 no such element
start← 0
end ← n-1
while start ≤ end
mid ← [ (start + end)/2 ]
i if A[mid] = x
return mid
elseif  x< A[mid]
end ← mid -1
else
start ← mid + 1
return -1
```

Explanation:

- The algorithm takes 3 values as input 1. Array A 2. Size of an array n 3. Element to search x
- Initialize start and end has zero
- Using formula to find middle element.

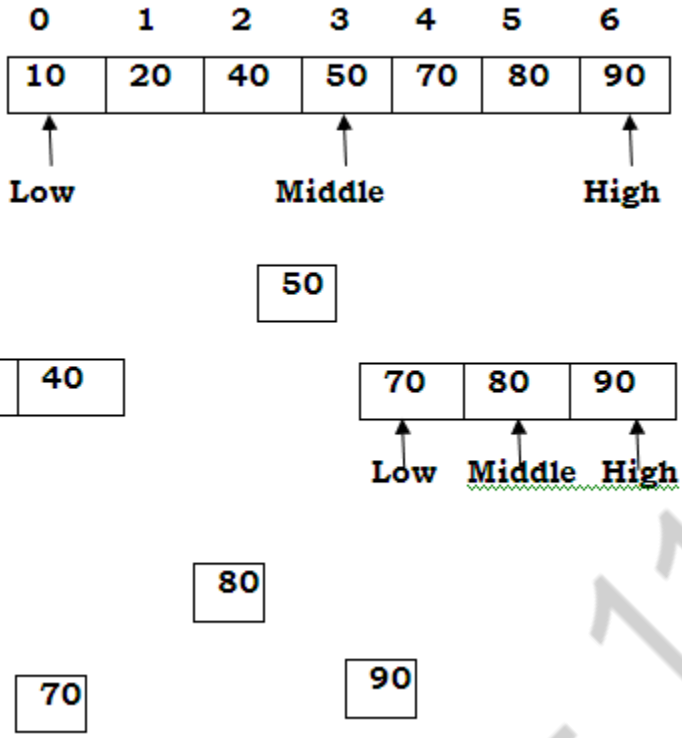
To get the result in 3 possibilities

1.If both equal position of the element is returned

2.If the value is less than the middle. The value of high is decreased to mid -1

3.If the value is greater than the middle. The value of high is decreased to mid+ 1

Example: Key (x) = 80



Return element 80 in the position of 5

Analysis of binarysearch:

Best case: 1 comparision

Ω (1)

Worst case: keep reducing the search the space n , n/2,n/4… till 1

$C_{worst}(n)=C_{worst}(n/2)+1$

Where:

$C_{worst}(n)$: Time required for i/p size n

$C_{worst}(n/2)$: time required to search for the key in left sublist or right sublist

1 : time required for comparing the key with the middle element

$C_{worst}(n) = C_{worst}(n/2)+1$ ----------→1

n= n/2 $C_{worst}(n) = C_{worst}(n/2/2)+1$

$C_{worst}(n) = C_{worst}(n/4)+1$----------→2

Sub 2 in 1

$C_{worst}(n) = C_{worst}(n/4)+1+1$

$C_{worst}(n) = C_{worst}(n/4)+2$----------→3

n= n/4 $C_{worst}(n) = C_{worst}(n/2)+1$

$C_{worst}(n) = C_{worst}(n/4/2)+1$

$C_{worst}(n) = C_{worst}(n/8)+1$ ----------→4

Sub 4 in 3

$C_{worst}(n) = C_{worst}(n/8)+1+2$

$C_{worst}(n) = C_{worst}(n/8)+3$

$C_{worst}(n) = C_{worst}(n/2^3)+3$

Where k =3

$C_{worst}(n) = C_{worst}(n/2^k)+k$

Assume $n= 2^k$

$C_{worst}(n) = C_{worst}(2^k/2^k)+k$

$C_{worst}(n) = C_{worst}(1)+k$ [$C_{worst}(1)=1$]

$C_{worst}(n) = 1+k$

Take log on both sides

$n= 2^k$

$\log n = \log_2 k$

$\log n = k \log_2$ [$\log_2 = 1$]

$\log n = \mathbf{k}$

$C_{worst}(n) = 1 + \log n$

$C_{worst}(n) = \mathbf{O(\log n)}$

Worst case: O(logn)

Average case: Ø(logn)

3. Compute the multiplication of given two matrices using Strasen's matrix multiplication method:

$$A = \begin{bmatrix} 1 & 0 & 2 & 1 \\ 4 & 1 & 1 & 0 \\ 0 & 1 & 3 & 0 \\ 5 & 0 & 2 & 1 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 2 & 1 & 0 & 4 \\ 2 & 0 & 1 & 1 \\ 1 & 3 & 5 & 0 \end{bmatrix}$$

8 marks (Nov/Dec 2015)

The standard method of matrix multiplication of two n x n matrices takes $T(n) = O(n^3)$ in brute force approach . Stassen's algorithm is a Divide-and-Conquer algorithm that beat the bound. The usual multiplication of two n x n matrices takes

If C= AB. then we: have the following

$c_{11} = a_{11}b_{11} + a_{12}b_{21}$

$c_{12} = a_{11}b_{12} + a_{12}b_{22}$

$c_{21} = a_{21}b_{11} + a_{22}b_{21}$

$c_{22} = a_{21}b_{12} + a_{22}b_{22}$

8 n/2 * n/2 matrix multiples plus 4 n/2 * n/2 matrix additions [Brute force]

Strassen's matrix multiplication:

Strassen showed how two matrices can be multiplied using only 7 multiplications and 18 additions:

$c_{11} = M1 + M4 – M5 + M7$

$c_{12} = M3 + M5$

$c_{21} = M2 + M4$

$c_{22} = M1 + M3 – M2 + M6$

Consider calculating the following 7 products:

$M1 = (a_{11} + a_{22}) * (b_{11} + b_{22})$

$M2 = (a_{21} + a_{22}) * b_{11}$

$M3 = a_{11}*(b_{12} – b_{22})$

$M4 = a_{22} * (b_{21} – b_{11})$

$M5 = (a_{11} + a_{12}) * b_{22}$

$M6 = (a_{21} – a_{11}) * (b_{11} + b_{12})$

$M7 = (a_{12} – a_{22}) * (b_{21} + b_{22})$

Effective analysis of strassen's matrix multiplication:

If we let T(n) be the running time of Strassen's algorithm, then it satisfies the following recurrence relation:

Let A and B be two n*n matrices where n is power of 2.

T(n) = 7T(n/2)----------→1 (apply back word substitution method n= n/2,n/4….. k)

Assume n= 2^k

log n = $\log_2^k$

log n =k $\log_2$ [$\log_2$=1]

log n = **k**

T(n) = 7logn

T(n)=$n^{2.807}$

Recurrence relation for 7 products and 18 additions:

A(n)= 7A(n/2)+18$(n/2)^2$

A(n)=$n^{2.807}$

4. **Write down the algorithm to construct a convex hull based on divide and conquer strategy. 8 marks (Nov/Dec 2015)**

In divide and conquer convex hull is called Quick hull, because of resemblance to quick sort.

Steps:

1. Find the points with minimum and maximum x coordinates; those are bound to be part of the convex hull.
2. Use the line formed by the two points to divide the set in two subsets of points, s1 is the set of points to left of line and s2 is the set of points to right of line.
3. Identify *extreme points* P_1 and P_2 (leftmost and rightmost).
4. Compute *upper hull(upperbound)* recursively:

a. find point P_{max} that is farthest away from line P_1P_2

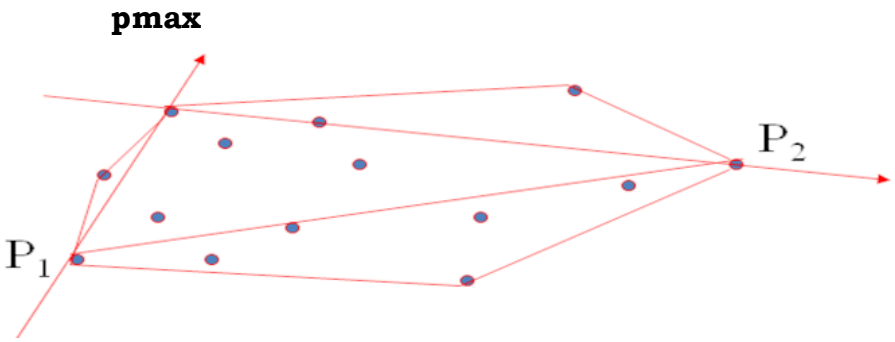
b. compute the upper hull of the points to the left of line P_1P_{max}

c. compute the upper hull of the points to the left of line $P_{max}P_2$

5. Compute *lower hull* in a similar manner

6. The points lying inside of that triangle cannot be part of the convex hull and can therefore be ignored in the next steps.

7. Repeat the previous two steps on the two lines formed by the triangle.



The analytical geometry IF p1(x1,x2), p2(x2,y2) and p3(x3,y3) are the three arbitrary points in the Cartesian plane.

The area of triangle p1,p2,p3 is equal to one half of the magnitude.

$$\frac{1}{2}\begin{vmatrix} x1 & y1 & 1 \\ x2 & y2 & 1 \\ x3 & y3 & 1 \end{vmatrix} = x1y2 + x3y1 + x2y3 - x3y2 - x2y1 - x1y3$$

if signed area >0, then a→ b →c is anticlockwise

if signed area<0, then a→ b →c is clockwise

if signed area =0, then a→ b →c is collinear

Algorithm:

Quickhull(P1,p2)

P1 & p2 are extreme points (admit vertical supporting lines)

S_1 ← points to the right of p1p2

S_2 ← points to the right of p2p1

FindHull (S_1,A,B)

FindHull(S_2, B,A)

Algorithm:

FindHull(P,A,B):

- If P is empty, then return.
- Find (extreme) point $C \in P$ orthogonally farthest from AB.
- CH update: replace AB with AC followed by CB. (inflate baloon)
- Partition P-{C} into Q_0, Q_1, Q_2 as shown.
- Discard Q_0 inside triangle ABC (by Caratheodory).
- Recursively call FindHull(Q_1,A,C) and FindHull(Q_2,C,B).

Effective analysis of convex hull:

T(n) = T(x) + T(y) + T(z) + T(v) + O(n),

where x + y + z +v <= n.

worst case: $\Theta(n^2)$

T(n) = T(n-1) + O(n) solve this we get $\Theta(n^2)$

Average case: $\Theta(n)$ (under reasonable assumptions about distribution of points given)

If points are not initially sorted by *x*-coordinate value, this can be accomplished in $O(n \log n)$ time

Several $O(n \log n)$ algorithms for convex hull .

5. Find the optimal solution to the fractional knapsack problem with given data:

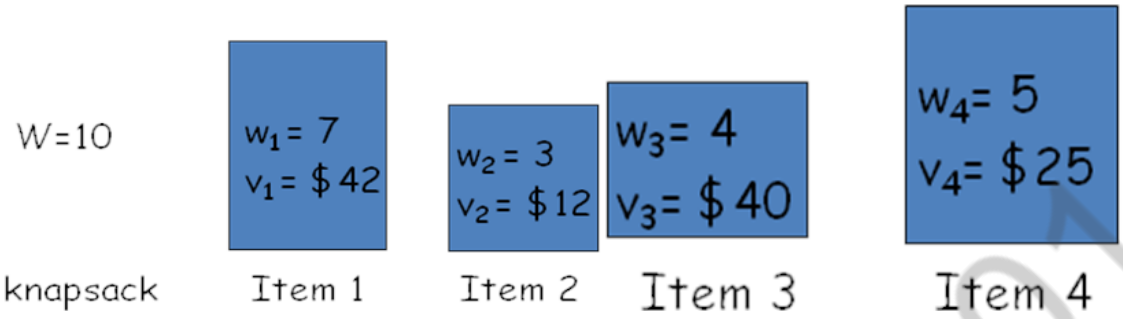
Item	Weight	Benefit
A	**2**	**60**
B	**3**	**75**
C	**4**	**90**

8 marks (Nov/Dec 2015)

optimal solution to the fractional knapsack

- There are n different items
- Each item has a weight and value.

- A knapsack of capacity W, find the most valuable subset of the items that fit into the knapsack
- A transport plane has to deliver the most valuable set of items to a remote location without exceeding its capacity



How about taking items in decreasing order of value/weight?

- The exhaustive search approach to this problem generates all the subsets of the set of n items given, the total weight of each subset in order to identify feasible subsets
- This algorithm only finds the max possible value that can be carried in the knapsack.
- Travelling salesman and knapsack problem is also knows as **NP-Hard problems**

6. State and Explain Mergesort algorithm and give the recurrence relation and efficiency (16) (MAY / JUNE 2016)

Mergesort:

- Mergesort is based on divide-and-conquer technique
- Sort a sequence of *n* elements into non-decreasing order.
- *Divide*: Divide the *n*-element sequence to be sorted into two subsequences of *n/2* elements each
- *Conquer:* Sort the two subsequences recursively using merge sort.
- *Combine*: Merge the two sorted subsequences to produce the sorted answer.

ALGORITHM *Mergesort(A*[0..*n* − 1]*)*

//Sorts array $A[0..n-1]$ by recursive mergesort

//Input: An array $A[0..n-1]$ of orderable elements

//Output: Array $A[0..n-1]$ sorted in nondecreasing order

if $n > 1$

copy $A[0.. \lfloor n/2 \rfloor - 1]$ to $B[0.. \lfloor n/2 \rfloor - 1]$

copy $A[\lfloor n/2 \rfloor .. n-1]$ to $C[0.. \lfloor n/2 \rfloor - 1]$

Mergesort($B[0.. \lfloor n/2 \rfloor - 1]$*)*

Mergesort($C[0.. \lfloor n/2 \rfloor - 1]$*)*

Merge(B, C, A) //see below

ALGORITHM *Merge(*$B[0..p-1]$, $C[0..q-1]$, $A[0..p+q-1]$*)*

//Merges two sorted arrays into one sorted array

//Input: Arrays $B[0..p-1]$ and $C[0..q-1]$ both sorted

//Output: Sorted array $A[0..p+q-1]$ of the elements of B and C

$i \leftarrow 0$; $j \leftarrow 0$; $k \leftarrow 0$

while $i < p$ **and** $j < q$ **do**

if $B[i] \leq C[j]$

$A[k] \leftarrow B[i]$; $i \leftarrow i + 1$

else $A[k] \leftarrow C[j]$; $j \leftarrow j + 1$

$k \leftarrow k + 1$

if $i = p$

copy $C[j..q-1]$ to $A[k..p+q-1]$

else copy $B[i..p-1]$ to $A[k..p+q-1]$

Explanation:

Mergesort is a recursive algorithm that splits the array into various sub problems. These subproblem are sorted independently by calling the merge algorithm. Once these subprobles are solved they are combined together to get a single sorted array list.

The *merging* of two sorted arrays can be done as follows.

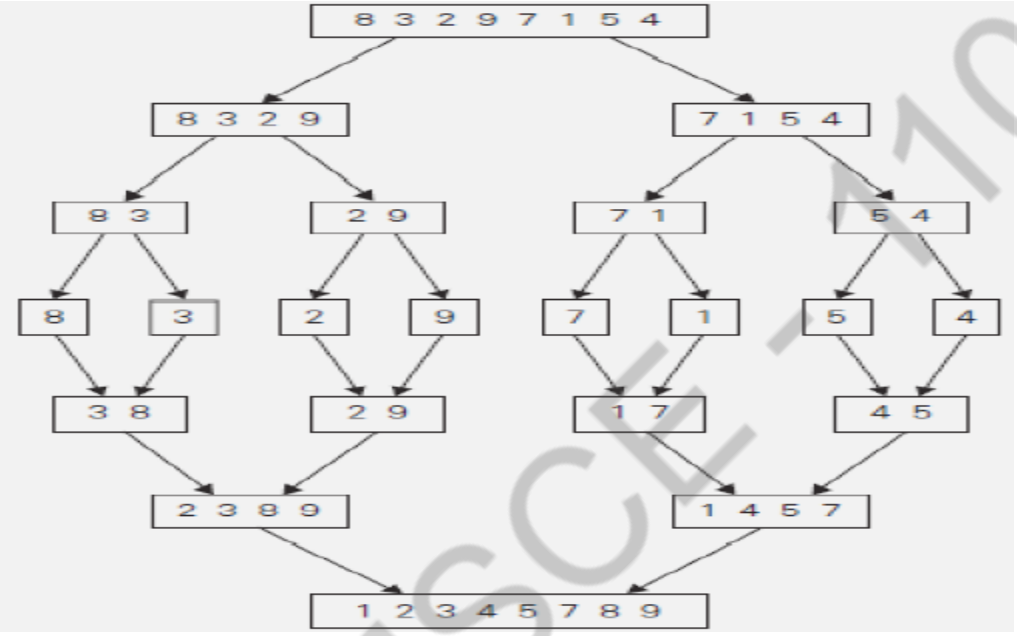
- Two pointers (array indices) are initialized to point to the first elements of the

arrays being merged.

- The elements pointed to are compared, and the smaller of them is added to a new array being constructed; after that, the index of the smaller element is incremented to point to its immediate successor in the array it was copied from.
- This operation is repeated until one of the two given arrays is exhausted, and then the remaining elements of the other array are copied to the end of the new array.

Example:

The operation of the algorithm on the list 8, 3, 2, 9, 7, 1, 5, 4 is illustrated in Figure .



Analysis of merge sort:

Let T(n) be the time taken by this algorithm to sort an array of n elements dividing A into sub array B and C.

$T(n)=T(n/2)+T(n/2)+C_n$

$T(n)=2T(n/2)+Cn$

T(n) : time required for sort ,

T(n/2) : time required for sorting left sub list and right sub list ,

Cn : time required for combining

Recurrence for Merge sort

Input size: n- size of array

Basic operation: diving and Merging

Recurence equation:

T(n) = 2T(n/2)+Cn --------→1

T(1)=0

n=n/2

T(n) = 2T(n/2*2)+C(n/2)

T(n)= 2T(n/4)+ C(n/2)----------→2

Sub 2 and 1

T(n) = 2 [2T(n/4)+C(n/2)] + Cn

T(n) = 4T(n/4)+2C(n/2)] + Cn [stick 2/2C(n) =Cn]

T(n) = 4T(n/4)+Cn + Cn

T(n) = 4T(n/4)+2Cn ----------→3

n = n/4

T(n) = 2T(n/4*2)+C(n/4)

= 2T(n/8)+C(n/4)----------→4

Sub 4 in 3

T(n) = 4[2T(n/8)+C(n/4)] +2Cn

T(n) = 8T(n/8)+4C(n/4) +2Cn [stick 4/4Cn =Cn]

T(n) = 8T(n/8)+Cn +2Cn

T(n) = 8T(n/8)+3Cn

$T(n) = 2^3T(n/2^3)+3Cn$

Assign k=3

$T(n) = 2^k T(n/2^k)+kCn$

Assign n= 2^k

$T(n) = nT(n/n)+kCn$ [stick $T(n/n)=T(1)$]

$T(n) = nT(1)+k\,Cn$ [ignore $nT(1)$ this term because it is small]

$T(n) = k\,Cn$

Take log on both side of

$n= 2^k$

$\log n = \log_2 k$

$\log n = k \log_2$ [$\log_2=1$]

$\log n =$ **k**

$T(n) = \log n . Cn$

$T(n) = \log n \,.n$

$T(n) = n \log n$

T(n) = O(n logn)

Master method:

$C(n) = 2C(n/2)+Cn$ a=2 , b=2 , d=n [d=cn]

$T(n) = n^{\log_b a}$

$= n^{\log_2 2}$

= n Vs n [n =n]

$C(n) = \Theta(n^d \log n)$ [d=1]

$C(n) = \Theta(n \log n)$

- **All cases have same efficiency: $\Theta(n \log n)$**
- **Number of comparisons in the worst case is close to theoretical minimum for comparison-based sorting:**

7. Explain the method used for performing Multiplication of two large integers. Explain how divide and conquer method can be used to solve the same. (16) (MAY

/ JUNE 2016)

Some applications like modern cryptography require manipulation of integers that are over

100 decimal digits long. Since such integers are too long to fit in a single word of a modern

computer, they require special treatment.

In the conventional pen-and-pencil algorithm for multiplying two n-digit integers, each of

the n digits of the first number is multiplied by each of the n digits of the second number for the total of n2 digit multiplications.

The divide-and-conquer method does the above multiplication in less than n2 digit multiplications.

Example: $23 * 14 = (2 \cdot 101 + 3 \cdot 100) * (1 \cdot 101 + 4 \cdot 100)$

$= (2 * 1)102 + (2 * 4 + 3 * 1)101 + (3 * 4)100$

$= 2 \cdot 102 + 11 \cdot 101 + 12 \cdot 100$

$= 3 \cdot 102 + 2 \cdot 101 + 2 \cdot 100$

$= 322$

The term $(2 * 1 + 3 * 4)$ computed as $2 * 4 + 3 * 1 = (2 + 3) * (1 + 4) - (2 * 1) - (3 * 4)$.

Here

$(2 * 1)$ and $(3 * 4)$ are already computed used. So only one multiplication only we have

to do.

For any pair of two-digit numbers $a = a1a0$ and $b = b1b0$, their product c can be

computed by the formula $c = a * b = c2102 + c1101 + c0$,

where

c2 = a1 ∗ b1 is the product of their first digits,

c0 = a0 ∗ b0 is the product of their second digits,

c1 = (a1 + a0) ∗ (b1 + b0) − (c2 + c0) is the product of the sum of the

a's digits and the sum of the b's digits minus the sum of c2 and c0.

Now we apply this trick to multiplying two n-digit integers a and b where n is a positive even number. Let us divide both numbers in the middle to take advantage of the divide-and conquer technique. We denote the first half of the a's digits by $a1$ and the second half by $a0$; for b, the notations are $b1$ and $b0$, respectively. In these notations, $a = a1a0$ implies that $a = a110n/2 + a0$ and $b = b1b0$ implies that $b = b110n/2 + b0$. Therefore, taking advantage of the same trick we used for two-digit numbers, we get

$C = a * b = (a110n/2 + a0) * (b110n/2 + b0)$

$= (a1 * b1)10n + (a1 * b0 + a0 * b1)10n/2 + (a0 * b0)$

$= c210n + c110n/2 + c0,$

where

$c2 = a1 * b1$ is the product of their first halves,

$c0 = a0 * b0$ is the product of their second halves,

$c1 = (a1 + a0) * (b1 + b0) - (c2 + c0)$

If $n/2$ is even, we can apply the same method for computing the products $c2$, $c0$, and $c1$.

Thus, if n is a power of 2, we have a recursive algorithm for computing the product of two n-digit integers. In its pure form, the recursion is stopped when n becomes 1. It can also be stopped when we deem n small enough to multiply the numbers of that size directly.

The multiplication of n-digit numbers requires three multiplications of $n/2$-digit numbers,

the recurrence for the number of multiplications $M(n)$ is $M(n) = 3M(n/2)$ for $n > 1$, $M(1) = 1$.

Solving it by backward substitutions for $n = 2k$ yields

$M(2k) = 3M(2k-1)$

$= 3[3M(2k-2)]$

$= 32M(2k-2)$

$= \ldots$

$= 3iM(2k-i)$

$= \ldots$

$= 3kM(2k-k)$

$= 3k.$

(Since $k = \log_2 n$)

$\boldsymbol{M(n) = 3\log_2}$

$\boldsymbol{n = n\log_2}$

$\boldsymbol{3 \approx n1.585.}$

(On the last step, we took advantage of the following property of logarithms: $a\log_b c = c\log_b a$.)

Let $A(n)$ be the number of digit additions and subtractions executed by the above algorithm

in multiplying two n-digit decimal integers. Besides $3A(n/2)$ of these operations needed to compute

the three products of $n/2$-digit numbers, the above formulas require five additions and one

subtraction. Hence, we have the recurrence

$A(n) = 3 \cdot A(n/2) + cn$ for $n > 1$, $A(1) = 1$.

By using Master Theorem, we obtain $\boldsymbol{A(n) \in \Theta(n\log_2 3),}$

which means that the total number of additions and subtractions have the same asymptotic order of growth as the number of multiplications.

Example: For instance: a = 2345, b = 6137, i.e., n=4.

Then $C = a * b = (23*102+45)*(61*102+37)$

$C = a * b = (a1 10n/2 + a0) * (b1 10n/2 + b0)$

$= (a1 * b1)10n + (a1 * b0 + a0 * b1)10n/2 + (a0 * b0)$

$= (23 * 61)104 + (23 * 37 + 45 * 61)102 + (45 * 37)$

$= 1403 \bullet 104 + 3596 \bullet 102 + 1665$

$= 14391265$

UNIT-III

DYNAMIC PROGRAMMING AND GREEDY TECHNIQUE

Computing a Binomial Coefficient – Warshall's and Floyd' algorithm – Optimal Binary Search Trees – Knapsack Problem and Memory functions. Greedy Technique– Prim's algorithm- Kruskal's Algorithm-Dijkstra's Algorithm-Huffman Trees.

PART-A

1. What is articulation point?

A vertex of a connected graph G is said to be in articulation point, if its removal with all edges incident to it breaks the graph into disjoint pieces.

2. Explain principle of Optimality?

The principle of optimality says that an optimal solution to any instance of an optimization problem is composed of optimal solution to its subinstances.

3. What is spanning tree?

Let G={V,E} be an undirected connected graph. A sub graph t={V,E} of G is a spanning tree of G, if t is a tree.

4. What is Dynamic programming?

Dynamic programming is an algorithm design technique for solving problem with overlapping subprograms. The smaller subprograms are solved only once and recording the results in a table from which the solution to the original problem is obtained.

5. What is greedy method?

The greedy method is the most straight forward design, which is applied for change making problem. The greedy technique suggests constructing a solution to an optimization problem through a sequence of steps, each expanding a partially

constructed solution obtained so far, until a complete solution to the problem is reached. On each step, the choice made must be feasible, locally optimal and irrevocable.

6. List the advantage of greedy algorithm

Greedy algorithm produces a feasible solution

Greedy method is very simple to solve a problem

Greedy method provides an optimal solution directly.

7. Define the term control abstraction?

Control abstraction is a procedure whose flow of control is clear but whose primary operations are specified by other procedures whose precise meanings are left undefined.

8. Define warshall's algorithm?

Warshall's algorithm is an application of dynamic programming technique, which is used to find the transitive closure of a directed graph.

9. Define Floyd's algorithm?

Floyd's algorithm is an application, which is used to find all the pairs shortest paths problem. Floyd's algorithm is applicable to both directed and undirected weighted graph, but they do not contain a cycle of a negative length.

10. Define prim's algorithm.

Prim's algorithm is a greedy and efficient algorithm, which is used to find the minimum spanning tree of a weighted connected graph.

11. How efficient is prim's algorithm?

The efficiency of the prim's algorithm depends on data structure chosen for the graph.

12. Define Kruskal's algorithm?

Kruskal's algorithm is another greedy algorithm for the minimum spanning tree problem. Kruskal's algorithm constructs a minimum spanning tree by selecting edges in increasing order of their weights provided that the inclusion does not create a cycle. Kruskals algorithm provides a optimal solution.

13. What is path compression?

The better efficiency can be obtained by combining either variation of quick union with path compression. Path compressions make every node encountered during the execution of a find operation point to the tree's node.

14. Define Dijkstra's Algorithm?

Dijkstra's algorithm solves the single source shortest path problem of finding shortest paths from a given vertex (the source), to all the other vertices of a weighted graph or digraph. Dijkstra's algorithm provides a correct solution for a graph with non negative weights.

15. Define Huffman trees?

A Huffman tree is binary tree that minimizes the weighted path length from the root to the leaves containing a set of predefined weights. The most important application of Huffman trees are Huffman code.

16. What do you mean by Huffman code?

A Huffman code is a optimal prefix tree variable length encoding scheme that assigns bit strings to characters based on their frequencies in a given text.

17. What is meant by compression ratio?

Huffman's code achieves the compression ratio, which is a standard measure of compression algorithms effectiveness of (3-2.25)/3*100 = 0.75/3*100 = 0.25 *100 = 25%.

18. List the advantage of Huffman's encoding?

a. Huffman's encoding is one of the most important file compression methods.

b. It is simple

c. It is versatility

d. It provides optimal and minimum length encoding

19. What is dynamic Huffman encoding?

In dynamic Huffman encoding, the coding tree is updated each time a new character is read from the source text. Dynamic Huffman encoding is used to overcome the drawback of simplest version.

20. What do you mean by optimal solution?

Given a problem with n inputs, we obtain a subset that satisfies come constraints. Any subset that satisfies these constraints is called a feasible solution. A feasible solution, which either maximizes or minimizes a given objective function is called optimal solution.

21. What is Knapsack problem?

A bag or sack is given capacity n and n objects are given. Each object has weight wi and profit pi .Fraction of object is considered as xi (i.e) 0<=xi<=1 .If fraction is 1 then entire object is put into sack. When we place this fraction into the sack we get wixi and pixi.

22. Define weighted tree.

A directed binary tree for which each edge is labeled with a real number (weight) is called as weighted tree.

23. What is the use of TVSP?

In places where the loss exceeds the tolerance level boosters have to the placed. Given a network and loss tolerance level the tree vertex splitting problems is to determine an optimal placement of boosters.

24.**What is the Greedy choice property?**

- The first component is greedy choice property (i.e.) a globally optimal solution can arrive at by making a locally optimal choice.
- The choice made by greedy algorithm depends on choices made so far but it cannot depend on any future choices or on solution to the sub problem.
- It progresses in top down fashion.

25. Write the specification of TVSP

Let T= (V, E, W) be a weighted directed binary tree where V vertex set

E edge set

W weight function for the edge.

W is more commonly denoted as w (i,j) which is the weight of the edge <i,j> ε E.

26. Write the difference between the Greedy method and Dynamic programming.

Greedy method	**Dynamic programming**
1 Only one sequence of decision is Generated.	1 Many number of decisions are Generated.
2 It does not guarantee to give an Optimal solution always.	2 It definitely gives an optimal Solution always.

27. Define dynamic programming.

Dynamic programming is an algorithm design method that can be used when a

solution to the problem is viewed as the result of sequence of decisions.

28. What are the features of dynamic programming?

Optimal solutions to sub problems are retained so as to avoid recomputing their values. Decision sequences containing subsequences that are sub optimal are not considered. It definitely gives the optimal solution always.

29. What are the drawbacks of dynamic programming?

Time and space requirements are high, since storage is needed for all level. Optimality should be checked at all levels.

30. Write the general procedure of dynamic programming.

The development of dynamic programming algorithm can be broken into a sequence of 4 steps.

1. Characterize the structure of an optimal solution.
2. Recursively define the value of the optimal solution.
3. Compute the value of an optimal solution in the bottom-up fashion.
4. Construct an optimal solution from the computed information.

31. Define the single source shortest path problem.

Dijkstra's algorithm solves the single source shortest path problem of finding shortest paths from a given vertex(the source), to all the other vertices of a weighted graph or digraph. Dijkstra's algorithm provides a correct solution for a graph with non negative weights.

32. State Assignment problem. *(*MAY / JUNE 2016)

There are n people who need to be assigned to execute n jobs, one person per job. (That is,

each person is assigned to exactly one job and each job is assigned to exactly one

person.) The cost that would accrue if the ith person is assigned to the jth job is a known quantity [,] for each pair , = 1, 2, . . . , . The problem is to find an assignment with the minimum total cost.

33.Write down the optimization technique used for Warshall's algorithm. State the rules and assumptions which are implied behind that. (Apr/May 2015)

In Warshall's algorithm we construct a sequence of Boolean matrices (n *n):

A = W [0] ,W[1] ,W[2] , ... ,W[n] =T,

Where A is adjacency matrix and T transitive closure.

This can be done from digraph D as follows.

1. **$W^{(1)}$ [i , j]=1** if and only if there is a walk from vi to vj with elements of a subset of {v1} as interior vertices.
2. **$W^{(2)}$ [i,j]=1** if and only if there is a walk from vi to vj with elements of a subset of {v1 , v2 } as interior vertices.
3. **Continuing this process,**
4. **$W^{(k)}$ [i,j] = 1** if and only if there is a walk from vi to vj with elements of a subset of { v1 , v2 , …, v k } as interior vertices

Note: In constructing W[k] from W[k −1] we shall either keep zeros or change some zeros to ones. No one's ever get changed to zeros

Recurrence relation:

$$W^{(k)}[i,j] = W^{(k-1)}[i,j] \text{ or } (W^{(k-1)}[i,k] \text{ and } W^{(k-1)}[k,j])$$

34.**List out the memory functions used under Dynamic programming (Apr/May 2015)**

Top down approach

Bottom up approach

Knapsack memory functions

35.State how Binomial Coefficient is computed (Nov/Dec 2015)

- Computing binomial coefficients is non optimal problem but can be solved using dynamic programming.
- Binomial coefficients are coefficients of the binomial formula :

$$(a + b)^n = C(n,0)a^n b^0 + C(n,1)a^{n-1} b^1 \ldots + C(n,k)a^{n-k}b^k + \ldots + C(n,n)a^0 b^n$$

Recurrence relation is defined by:

$C(n,k) = C(n-1,k) + C(n-1,k-1)$ for $n > k > 0$

$C(n,0) = 1, \quad C(n,n) = 1$ for $n \geq 0$

Value of *C*(*n*,*k*) can be computed by filling a table:

	0	**1**	**2**	**. . .**	***k*-1**	***k***
0	**1**					
1	**1**	**1**				
2	**1**	**2**	**1**			
3	**1**	**3**	**3**			
.						
***n*-1**	**1**				***C*(*n*-1,*k*-1)**	***C*(*n*-1,*k*)**
n	**1**					***C*(*n*,*k*)**

ALGORITHM *Binomial*(n, k)

```
//Computes C(n, k) by the dynamic programming algorithm
//Input: A pair of nonnegative integers n ≥ k ≥ 0
//Output: The value of C(n, k)
for i ← 0 to n do
    for j ← 0 to min(i, k) do
        if j = 0 or j = i
            C[i, j] ← 1
        else C[i, j] ← C[i − 1, j − 1] + C[i − 1, j]
return C[n, k]
```

Example: input n=5 k=4

	0	1	2	3	4
0	1				
1	1	1			
2	1	2	1		
3	1	3	3	1	
4	1	4	6	4	1
5	1	5	10	10	5

C(5,4)=5

Efficiency analysis:

A(n,k) = sum for upper triangle + sum for lower rectangle

$A(n,k) = \sum_{i=1}^{k} \sum_{j=1}^{i-1} 1 + \sum_{i=1}^{n} \sum_{j=1}^{k} 1$

$$A(n,k) = \sum_{i=1}^{k} (i-1) + \sum_{i=1}^{n} K$$

$A(n,k) = n^3$

A: total number of additions.

36. What is the best algorithm suited to identify the topography for a graph. Mention its efficiency factors. (Nov/Dec 2015)

Breadth-First search

Dijkstra's method

PART-B

1. How do you construct a minimum spanning tree using Kruskal's algorithm? Explain? 4 marks (Apr/May 2015)

- Kruskal's algorithm is a greedy algorithm in graph theory.
- Kruskal's Algorithm is finds a minimum spanning tree for a weighted , connected graph.
- Kruskal's Algorithm is work with edges, rather than vertex
- From V vertices to find v-1 edges.

Steps for finding MST using Kruskal's algorithm:

1.Sort the edges in increasing order of their weight.

2.Select the smallest edge that do not generate a cycle.

3.Repeat step#2 until there are (V-1) edges in the spanning tree.

Algorithm: Kruskal(G)

// Input: A weighted connected graph G = (V, E)

// Output: Set of edges comprising a MST sort the edges E by their weights

$E_T \leftarrow \emptyset$ // initialize the set of tree edges

while $|E_T| + 1 < |V|$ do

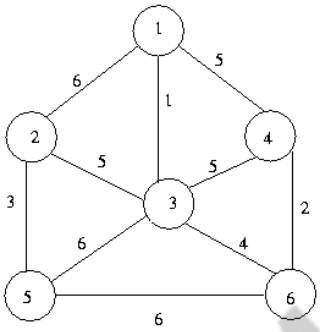
e ← next edge in E

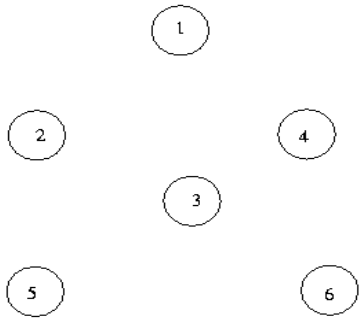
if $E_T \cup \{e\}$ //does not have a cycle then

$E_T \leftarrow E_T \cup \{e\}$

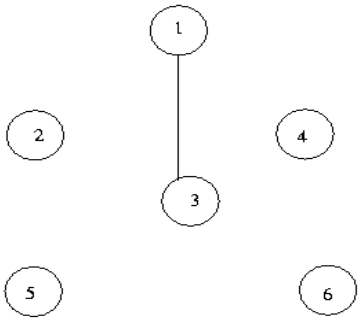
return E_T

Example:

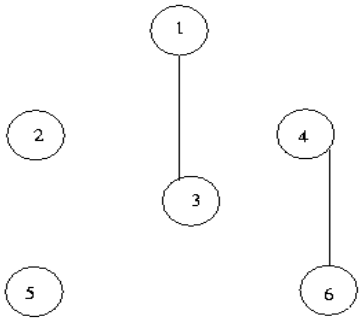




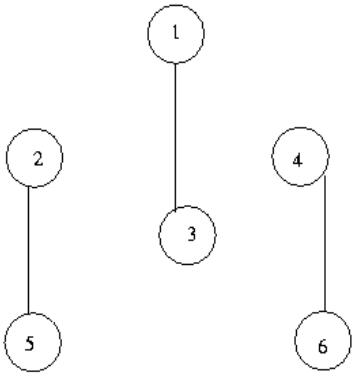
Initial Configuration



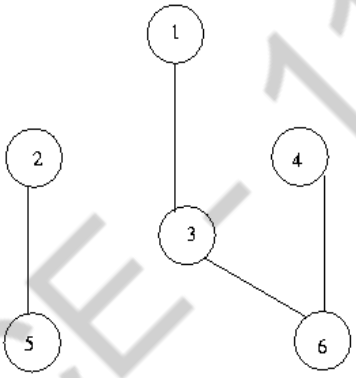
step1. choose (1,3)



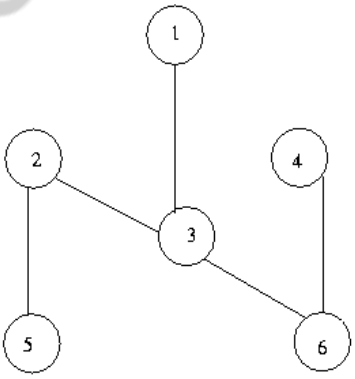
step2. choose (4,6)



step3. choose (2,5)



step4. choose (3,6)



step5. choose (2,3)

Effective analysis:

This algorithm repeatedly chooses the smallest-weight edge that does not form a cycle.

Kruskal's Algorithm is O(E log V) using efficient cycle detection.

2. Write an algorithm to construct the optimal binary search tree given the roots r(i,j), 0≤i≤j≤n. Also prove that this could be performed in time O(n). 8marks

(Apr/May 2015)

Definition OBST:

- OBST is a binary search tree which provides the smallest possible search time for a given sequence or access probabilities.
- OBST has minimum depth and should be balanced.
- In OBST the average number of comparisons in a search will be small.

Steps:

1. Let a1 to an is the set of identifiers such that a1<a2<...<an
2. Let pi be the probability with which we can search for ai
3. The problem is to construct a binary search tree with optimal cost such a tree is called OBST

OBST need 2 tables:

1. Cost table: build a cost table of size i*j where i from 1 to n+1 , j from 0 to n
2. Root table: build a root table of size i*j where i from 1 to n , j form 1 to n

Notation:

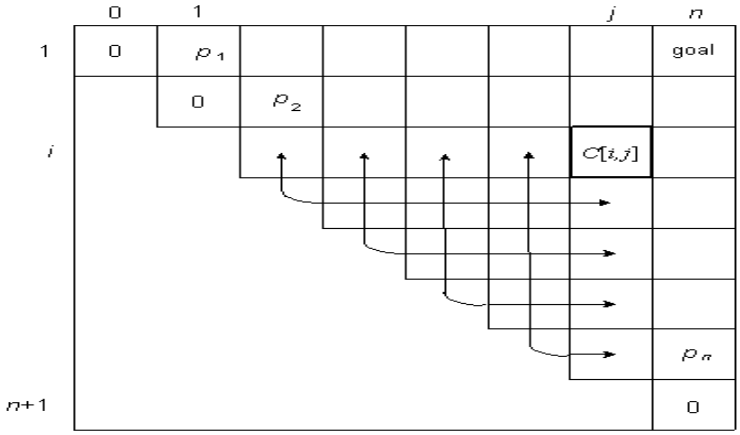
$p_{ij}=p_{i+1}+p_{i+2}+....p_j = \sum_{s=i}^{j} ps$, $p^{0,n} = 1$ [p1,p2,p3,p4,....p2= $p^{1,5}$]

$C_{i,j}$ = optimal weighted path length for a subtree containing the elements i+1....j

$C_{i,i} = 0$; $C_{i+j} = p_i$ [$C_{i,i}$ no element (0) , C_{i+j}, 1 element probably p_i]

$C_{i,j} = p_{i,j} + \min_{i< k \leq j} \{ C_{i,k-1} + C_{k+1,j} \}$ // cost table [min weight from left and right path]

$R_{i,j} = k$ that minimizes the last expression**//Root table** [choose the root node R_{ij}]



ALGORITHM *OptimalBST*$(P[1..n])$

```
//Finds an optimal binary search tree by dynamic programming
//Input: An array P[1..n] of search probabilities for a sorted list of n keys
//Output: Average number of comparisons in successful searches in the
//        optimal BST and table R of subtrees' roots in the optimal BST
for i ← 1 to n do
    C[i, i − 1] ← 0
    C[i, i] ← P[i]
    R[i, i] ← i
C[n + 1, n] ← 0
for d ← 1 to n − 1 do //diagonal count
    for i ← 1 to n − d do
        j ← i + d
        minval ← ∞
        for k ← i to j do
            if C[i, k − 1] + C[k + 1, j] < minval
                minval ← C[i, k − 1] + C[k + 1, j]; kmin ← k
        R[i, j] ← kmin
        sum ← P[i]; for s ← i + 1 to j do sum ← sum + P[s]
        C[i, j] ← minval + sum
return C[1, n], R
```

Effective analysis:

Input size: n- size of array

Basic operation: comparison

Booth operations are executed same number of times.

C(n) is dependent on the input size of matrix.

The number of multiplications made for every pair of specific values of variables i, j and k is

$c(n) = \sum_{i=1}^{n} \quad \sum_{j=1}^{n} \quad \sum_{k=1}^{n} 1$ = n3

C(n)= O(n^3)

3.The binary string below is the title of a song encoded using Huffman codes.

0011000101111101100111011101100000100111010010101. Given the letter frequencies listed in the table below, build the Huffman codes and use them to decode the title. In cases where there are multiple "greedy" choices, the codes are assembled by combining the first letters (or groups of letters) from left to right, in the order given in the table. Also, the codes are assigned by labelling the left and right branches of the prefix/code tree with '0' and '1', respectively.

Letter	**a**	**h**	**v**	**w**	**''**	**e**	**t**	**l**	**o**
Frequency	**1**	**1**	**1**	**2**	**2**	**2**	**3**	**3**	

10 marks (Nov/Dec 2015)

Write the procedure to compute Huffman code 6 marks (Nov/Dec 2015)

Humming code:

- A Huffman trees is a method for the construction of Minimum Redundancy codes.
- Huffman trees are constructed for encoding a given text of n characters.
- Encoding a given text, each character is associated with some bit sequence.

Steps:

1. Create a terminal node for each ai ∈ Σo , with probability p(ai) and let S = the set of terminal nodes.
2. Select nodes x and y in S with the two smallest probabilities.
3. Replace x and y in S by a node with probability p(x) + p(y). Also, create a node in the tree which is the parent of x and y.
4. Repeat (2)-(3) untill |S|=1.

Algorithm:

```
Huffman(a)
n = |a|
Q = |a|
for i ← 1 to n-1
        S = new node // s is a terminal node
        left[S] = Extract-Min(Q) //Extract the minimum probability node in left side
        right[S] = Extract-Min(Q)  //Extract the minimum probability node in rigth side
        p[S] = p[left[S]] + p[right[S]] //add the probability
        insert(Q,S)
return Extract-Min(Q) // root of the tree
```

3. Discuss about the algorithm and pseudocode to find the Minimum Spanning Tree using Prim's Algorithm. Find the Minimum Spanning Tree for the graph shown below. And Discuss about the efficiency of the algorithm Write and analyze the Prim's Algorithm 6 marks (Nov 2015/Jun 2016)

Prim's algorithm is an algorithm used to find a minimum cost spanning tree for connected weighted undirected graph.

Spanning Tree:

A spanning tree of a graph G is connects all vertices without loops.

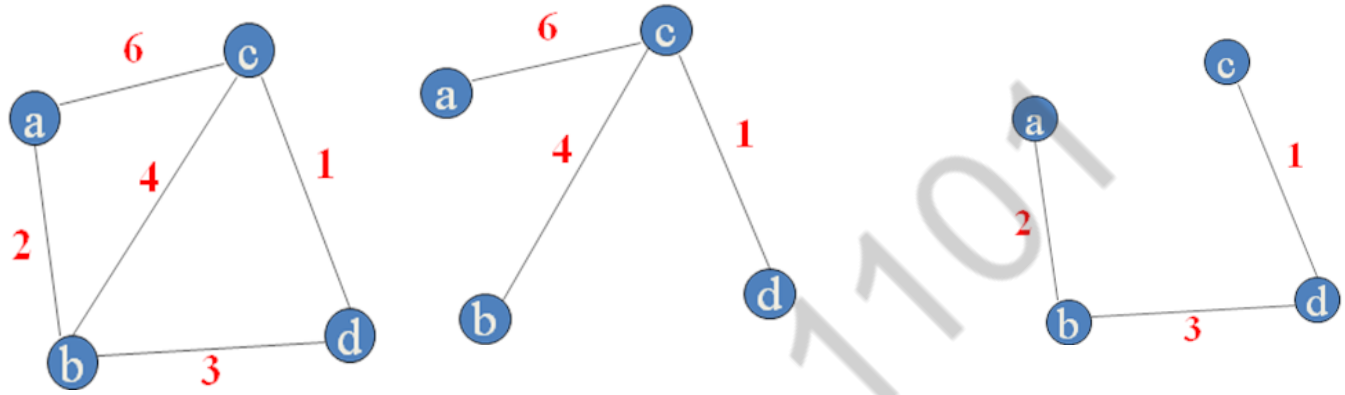
Minimum spanning tree: (MST)

A minimum spanning tree of a weighted connected graph G is a spanning tree with minimum or smallest weight.

Weight of the tree:

A weight of the tree is defined as the sum of weights of all its edges.

Example:



Graph W(T1)=11 W(T2)=6

Here W(T2) is a minimum spanning tree.

Steps in prim's algorithm:

1. Start from any arbitrary vertex
2. Find the edge that has minimum weight from all known vertices
3. Stop when the tree covers all vertices.

Algorithm: Prim (G)

//Input: A weighted connected graph G={V,E}

//output: E_T, the set of edges composing a minimum spanning tree of G

$V_T \leftarrow \{v0\}$ // Set of vertices visited already

$E_T \leftarrow \emptyset$ // Set of edges visted already [$\emptyset$ =null]

for i← 1 to V-1 do //find minimum edges e between Vertices v and u.

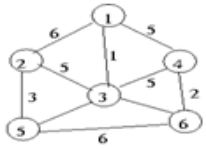
//Such that v (visited) is in V_T & u(unvisited) is in V-V_T

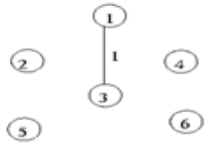
$V_T \leftarrow V_T$ U {u*} //Add u to V_T

$E_T \leftarrow E_T$ U { e*} //Add the edges to the spanning tree

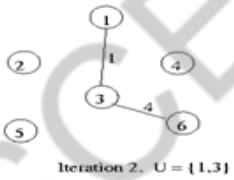
return E_T

Example:

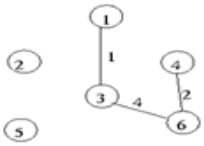




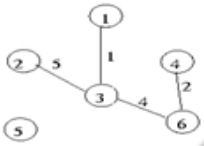
Iteration 1. U = {1}



Iteration 2. U = {1,3}



Iteration 3. U = {1,3,6}



Iteration 4. U = {1,3,6,4}

Iteration 5. U = {1,3,6,4,2}

Effective Analysis:

If the graph is represent the adjacent matrix then the rime complexity of Prim's algorithm is

$T(n)= O(n^2)$

Prim's algorithm using a binary heap to implement a priority queue is

$O(E \log E)=O(E \log V)$

Proof of Correctness of Prim's Algorithm:

Theorem: Prim's algorithm finds a minimum spanning tree.

Proof:

Let G = (V,E) be a weighted, connected graph.

Let T be the edge set that is grown in Prim's algorithm.

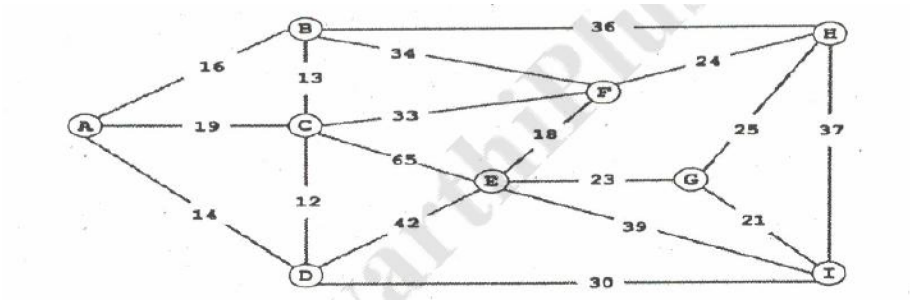
The proof is by mathematical induction on the number of edges in T and using the MST Lemma.

Basis: The empty set ∅ is promising since a connected, weighted graph always has at least one MST.

Induction Step: Assume that T is promising just before the algorithm adds a new edge

e = (u,v). Let U be the set of nodes grown in Prim's algorithm. Then all three conditions in the MST Lemma are satisfied and therefore T U e is also promising.

4. Consider the following weighted graph.



Give the list of edges in the MST in the order that prim's algorithm inserts them. Start Prim's algorithm from vertex A. 10 marks (Nov/Dec 2015)

Algorithm: Prim (G)

//Input: A weighted connected graph G={V,E}

//output: E_T, the set of edges composing a minimum spanning tree of G

$V_T \leftarrow \{v0\}$ // Set of vertices visited already

$E_T \leftarrow \emptyset$ // Set of edges visted already [$\emptyset$ =null]

for i← 1 to V-1 do //find minimum edges e between Vertices v and u.Such that v (visited) is

in V_T & u(unvisited) is in $V-V_T$

$V_T \leftarrow V_T \cup \{u^*\}$ //Add u to V_T

$E_T \leftarrow E_T \cup \{e^*\}$ //Add the edges to the spanning tree

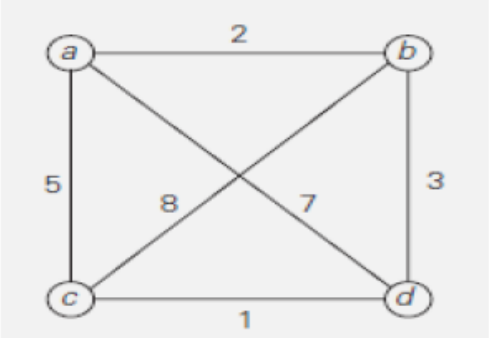
return E_T

6. Find all the solution to the traveling salesman problem (cities and distance shown
below) by exhaustive search. Give the optimal solutions. (16) (MAY / JUNE 2016)

The ***traveling salesman problem (TSP)*** is one of the combinatorial problems. The problem asks to find the shortest tour through a given set of *n* cities that visits each city exactly once before returning to the city where it started.

The problem can be conveniently modeled by a weighted graph, with the graph's vertices
representing the cities and the edge weights specifying the distances. Then the problem can be
stated as the problem of finding the shortest ***Hamiltonian circuit*** of the graph. (A Hamiltonian
circuit is defined as a cycle that passes through all the vertices of the graph exactly once).

A Hamiltonian circuit can also be defined as a sequence of $n + 1$ adjacent vertices
$v_{i0}, v_{i1}, \ldots, v_{in-1}, v_{i0}$, where the first vertex of the sequence is the same as the last one and all the
other $n - 1$ vertices are distinct. All circuits start and end at one particular vertex. Figure 2.4 presents a small instance of the problem and its solution by this method.



Tour	Length
a ---> b ---> c ---> d ---> a	I = 2 + 8 + 1 + 7 = 18
a ---> b ---> d ---> c ---> a	**I = 2 + 3 + 1 + 5 = 11 optimal**
a ---> c ---> b ---> d ---> a	I = 5 + 8 + 3 + 7 = 23
a ---> c ---> d ---> b ---> a	**I = 5 + 1 + 3 + 2 = 11 optimal**
a ---> d ---> b ---> c ---> a	I = 7 + 3 + 8 + 5 = 23
a ---> d ---> c ---> b ---> a	I = 7 + 1 + 8 + 2 = 18

FIGURE Solution to a small instance of the traveling salesman problem by exhaustive search.

UNIT-IV

ITERATIVE IMPROVEMENT

The Simplex MeFthod-The Maximum-Flow Problem – Maximm Matching in Bipartite Graphs- The Stable marriage Problem.

PART-A

1. What is maximum flow?

Maximum flow is a maximum feasible flow between source and sink.

In maximum flow problem can computed the greatest rate at which material can be moved from source to sink without violating any capacity constraints.

2. Define flow?

Flow is the rate at which the material moves through the underlying object.

3. Define flow conservation network?

The total amount of material entering an intermediate vertex must be equal to the total amount of material leaving the vertex. This condition is called flow-conservation network.

4 What is stable marriage problem?

The goal of this problem is to find stable matching between two sets (men and women) with various preferences to each other.

The stable marriage problem is an important algorithmic version of bipartite matching problem.

5.What is linear programming problem?

Linear programming problem is a problem in which we have to find the maximum (or minimum value) of a linear objective function.

6. **What is an optimal value and optimal solution?**

The desired largest (or smallest) value of the objective function is called the optimal value and the collection of values of x,y,z,.. that gives the optimal value constitutes an optimal solution. The variables x,y,z,.. are called the decision variables.

7. What is state space tree? *(*MAY / JUNE 2016)

Backtracking and branch bound are based on the construction of a state space tree, whose nodes reflect specific choices made for a solution's component . Its root represents an initial state before the search for a solution begins. The nodes of the first level the tree represent the made for the first component of solution, the nodes of the second evel represent the Choices for the second components & so on

8. State Extreme point theorem. ***(*MAY / JUNE 2016)**

Any linear programming problem with a nonempty bounded feasible region has an optimal

solution; moreover, an optimal solution can always be found at an extreme point of the problem's feasible region. Extreme point theorem states that if S is convex and compact in a locally convex space, then S is the closed convex hull of its extreme points: In particular, such a set has extreme points. Convex set has its extreme points at the boundary. Extreme points should be the end points of the line connecting any two points of convex set.

8. What do you mean by 'perfect matching' in bipartite graphs? (Apr/May 2015)

- A bipartite graph is an undirected graph G=(V,E) in which V can be partitioned into two sets V_1 and V_2 such that (u,v) ∈ E implies either u ∈ V_1 and v ∈ V_{12} or vice versa.
- That is, all edges go between the two sets V_1 and V_2 and not within V_1 and V_2.

- A matching M is a subset of edges such that each node in V appears in at most one edge in M.

- Maximal Matching: A maximal matching is a matching to which no more edges can be added without increasing the degree of one of the nodes to two; it is a local maximum.

- Maximum Matching: A maximum matching is a matching with the largest possible number of edges; it is globally optimal.

Given an undirected graph G=(V,E)
A bipartite graph G=(V,E) with vertex partition V=L∪R.(a)A matching with cardinality 2.(b) A maximum matching with cardinality 3.

9.Define flow 'cut' (Apr/May 2015)

Cut is a collection of arcs such that if they are removed there is no path from s to t (Source to Sink).
A cut is said to be minimum in a network whose capacity is minimum over all cuts of the networks

10 Define Network Flow and Cut. (Nov/Dec 2015)

Flow network: Flow network is a directed graph G=(V,E) such that each edge has a non-negative capacity c(u,v)≥0.

PART-B

1.Write the procedure to initialize Simplex which determines if a linear program is feasible or not? 4 marks (Nov2015 /May 2016)

- The simplex method to solve linear programming problems.

- Linear programming problem is used to find minimum or maximum value of a linear objective function.

- The general problem of optimizing a linear function of several variables subject to a set of linear constraints is linear programming.

Character:

- It should not be evaluate any infeasible solution.
- It should be capable of giving better & better basic feasible solution.
- It should be able to identify optimality and terminate.

Steps

- **Initiaization :** present a given linear programming problem in standard form.
- **Find initial basic feasible solution**
- **initial simplex table**
- **Optimality test:** If all the entries in the objective row are non negative- stop.Else go to step 5
- **5.Finding the entering variable :** corresponds to the smallest (the most negative) entry in the bottom row of the tableau.

o **Finding the departing variable:** corresponds to the smallest nonnegative ratio of , in the column determined by the entering variable.

o **finding pivoting :** The entry in the simplex table in the entering variable's column and the departing variable's row is called the pivot.

o **Applying Gauss Jordan elimination method**

o **If all element in the bottom of row is non negative stop if otherwise repeat the step from 4.**

2. Explain Max-Flow Problem 4 marks (Nov 2015 / May 2016)

Maximum flow is a maximum feasible flow between source and sink.

In maximum flow problem can computed the greatest rate at which material can be moved from source to sink without violating any capacity constraints.

Basic terminologies:

Flow network: Flow network is a directed graph G=(V,E) such that each edge has a non-negative capacity c(u,v)≥0.

Two distinguished vertices exist in G namely:

- **Source (denoted by s):** In-degree of this vertex is 0.
- **Sink (denoted by t) :** Out-degree of this vertex is 0
- Flow network is an integer-valued function f defined on the edges of G satisfying **0 ≤ f(u,v) ≤ c(u,v),** for every Edge (u,v) in E.

Properties of flow network:

- **Capacity Constraint:** (flow is not exceed the capicity): (for all)∀ u,v ∈V, (flowof edges) **f(u,v) ≤c(u,v)**
- **Skew Symmetry:** ∀ u,v∈V, **f(u,v)= -f(v,u)**
- **Flow conservation:** (whatever is coming the same thing is going out):∀ u ∈V – {s,t}

$$\sum_{u \in V} f(v,u) = \sum_{u \in V} f(u,v)$$

3. Write down the optimality condition and algorithmic implementation for finding M-augmenting paths in bipartite graphs. 8 marks (Apr/May 2015)

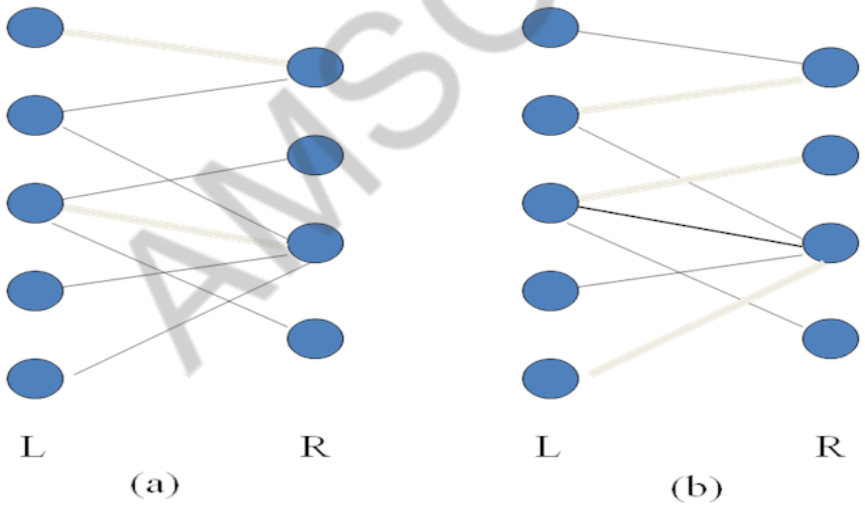
- A bipartite graph is an undirected graph G=(V,E) in which V can be partitioned into two sets V_1 and V_2 such that (u,v) ∈ E implies either u ∈ V_1 and v ∈ V_{12} or vice versa.
- That is, all edges go between the two sets V_1 and V_2 and not within V_1 and V_2.
- A matching M is a subset of edges such that each node in V appears in at most one edge in M.

- Maximal Matching: A maximal matching is a matching to which no more edges can be added without increasing the degree of one of the nodes to two; it is a local maximum.

- Maximum Matching: A maximum matching is a matching with the largest possible number of edges; it is globally optimal.

Algorithm:

1. Given an undirected graph G=(V,E)
2. A matching is a subset of edges M⊆E such that for all vertices v∈V,at most one edge of M is incident on v.
3. A vertex v ∈V is matched by matching M if some edge in M is incident on v;otherwise, v is unmatched.
4. A maximum matching is a matching of maximum cardinality,that is, a matching M such that for any matching M',
5. we have .



A bipartite graph G=(V,E) with vertex partition V=L∪R.(a)A matching with cardinality 2.(b) A maximum matching with cardinality 3.

4. Briefly describe on the Stable marriage problem 6 marks (Apr/May 2015)

- Stable marriage problem is an interesting version of bipartite matching.
- It is the problem of finding a stable matching between two sets of elements
- A marriage matching M is called stable if there is no blocking pair for it. Otherwise it is called unstable.
- The stable marriage problem is to find a stable marriage matching according to men's & women's given preferences.
- A matching is a mapping from the elements of one set to the element of other set.

Algorithm: Gale -Shapley algorithm

//Input: List of men, women, and their preference list

//Output: A stable marriage matching.

Step 1: initially all the men & women being free.

Step 2: While there are free men, arbitrarily select one of them and do the following.

Step 3: Return the set n matched pair

assign each person to be free;

while (some man ***m*** is free)

w := first women on ***m***'s list;

if (***w*** is currently assigned to some man ***p***)

assign ***p*** to be free;

end if

assign ***m*** to ***w***;

foreach ($m' \in successors_w(m)$)

delete (***m'***, ***w***);

end loop

end loop

Theorem: Termination and number of steps

Proof:

- Once a woman is matched, she stays matched (her partner can change).
- When the partner of a woman changes, this is to a more preferable partner for her: at most $n - 1$ times.

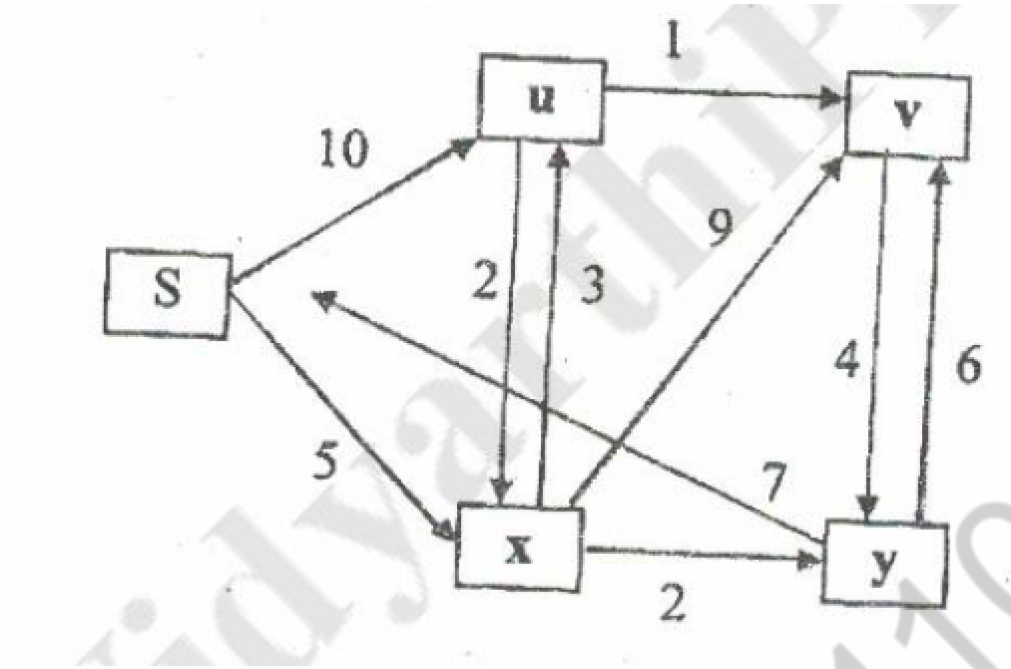
Every step, either an unmatched woman becomes matched, or a matched woman changes partner: at most n^2 steps.

Effective Analysis:

Time complexities $O(n^3)$ and $O(n^2)$.

average-case analysis: we assume that the men's lists are chosen independently and uniformly at random; the women's lists can be arbitrary but must be fixed in advance.

5. How do you compute maximum flow for the following graph using Ford-Fulkerson method? 10 marks (Apr/May 2015)



The Ford – Fulkerson Method:

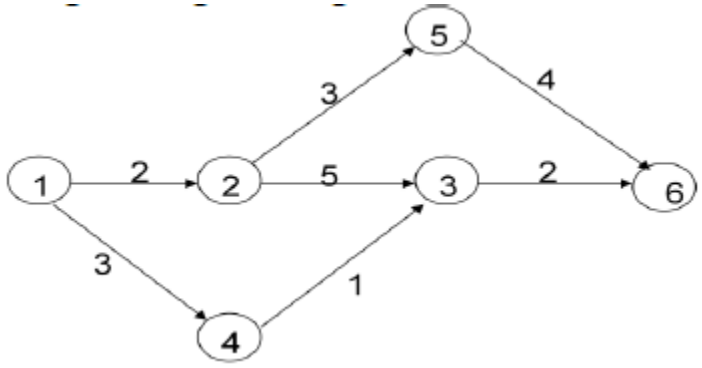
- The Ford-Fulkerson method for solving the maximum-flow problem.
- We call it's a "method" rather than an "algorithm" because it encompasses several implementations with different running times.

Algorithm: Ford –Fulkerson Method(G,s,t)

```
FORD-FULKERSON-METHOD(G, s, t)
1  initialize flow f to 0
2  while there exists an augmenting path p
3       do augment flow f along p
4  return f
```

Problem:

14 (b)Apply the shortest Augmenting Path algorithm to the network shown below. (16) (MAY / JUNE 2016)



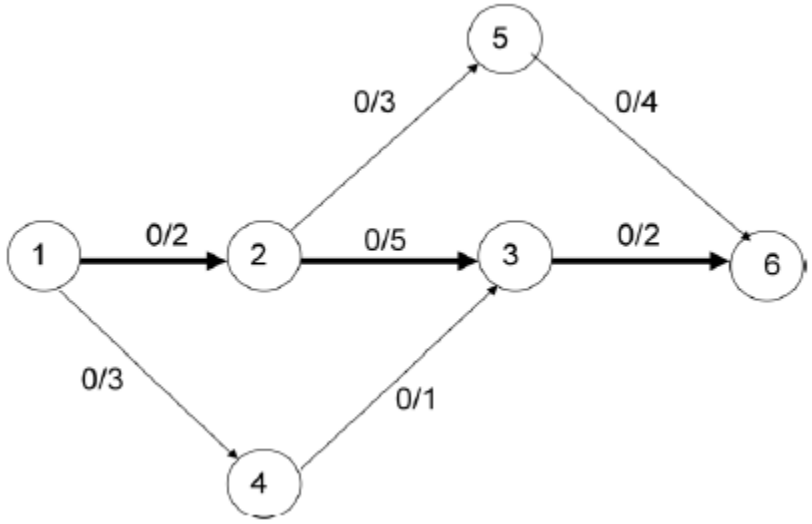
Augmenting Path (Ford-Fulkerson) Method

· Start with the zero flow (*xij* = 0 for every edge).

· On each iteration, try to find a *flow-augmenting path* from source to sink, which a path along which some additional flow can be sent.

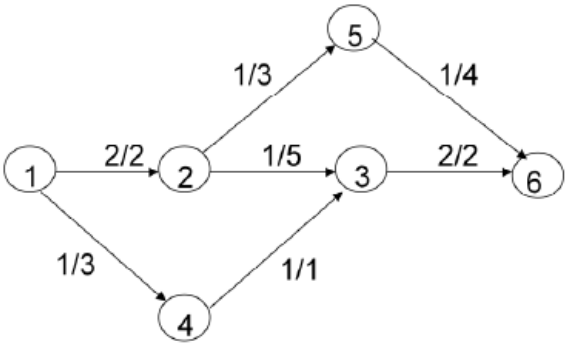
· If a flow-augmenting path is found, adjust the flow along the edges of this path to get a flow of increased value and try again.

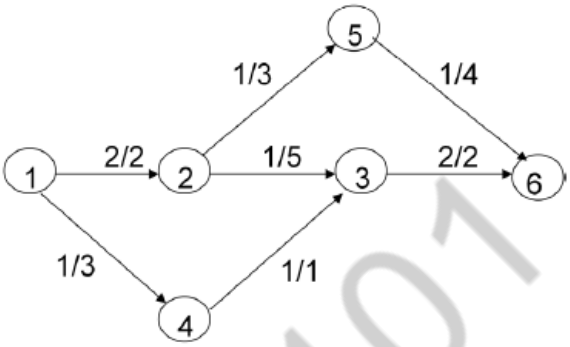
· If no flow-augmenting path is found, the current flow is maximum.



Augmenting path: 1→2 →3 →6

xij/uij





UNIT-V

COPING WITH THE LIMITATIONS OF ALGORITHM POWER

Limitations of Algorithm Power-Lower-Bound Arguments-Decision Trees-P, NP and NP-Complete Problems--Coping with the Limitations - Backtracking – n-Queens problem – Hamiltonian Circuit Problem – Subset Sum Problem-Branch and Bound – Assignment problem – Knapsack Problem – Traveling Salesman Problem- Approximation Algorithms for NP – Hard Problems – Traveling Salesman problem – Knapsack problem.

PART-A

1. Define backtracking?

Depth first node generation with bounding function is called backtracking. The backtracking algorithm has its virtue the ability to yield the answer with far fewer than m trials.

2. What is Hamiltonian cycle in an undirected graph?

A Hamiltonian cycle is round trip along n edges of G that visits every vertex once and returns to its starting position.

3. What is Feasible solution?

It is obtained from given n inputs Subsets that satisfies some constraints are called feasible solution. It is obtained based on some constraints

4. What is optimal solution?

It is obtained from feasible solution.

Feasible solution that maximizes or minimizes a given objective function It is obtained based on objective function.

5. List the application of backtracking technique?

The application of backtracking technique is 8-Queens problem

6. Given an application for knapsack problem?

The knapsack problem is problem in combinatorial optimization. It derives its name from the maximum problem of choosing possible essential that can fit into one bag to be carried on a trip. A similar problem very often appears in business, combinatory, complexity theory, cryptography and applied mathematics.

7.Define subset sum problem?

Subset sum problem is a problem, which is used to find a subset of a given set S={S1,S2,S3,.....Sn} of n positive integers whose sum is equal to given positive integer d.

8.What is heuristic?

A heuristic is a common sense rule drawn from experience rather than from a mathematically proved assertion. For example, going to the nearest un visited city in the travelling salesman problem is good example for heuristic.

9.State the concept of branch and bound method?

The branch and bound method 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.

10.Give the upper bound and lower bound of matrix multiplication algorithm?

Upper bound: The given algorithm does n*n*n multiplication hence at most n*n*n multiplication are necessary.

Lower bound: It has been proved in the literature that at least n*n multiplication are necessary.

11.What is state space tree?

Backtracking and branch bound are based on the construction of a state space tree, whose nodes reflect specific choices made for a solution's component. Its root represents an initial state before the search for a solution begins. The nodes of the first level in the tree represent the choices made for the first component of a solution, the nodes of the second level represent the choices for the second components & so on.

12.What is promising and non promising node?

A node in a state space tree is said to be promising, if it corresponds to a partially constructed solution that may still lead to a complete solution. Otherwise, a node is called non- promising.

13.What are the additional items are required for branch and bound compare to backtracking technique?

Compared to backtracking, branch and bound requires 2 additional items.

A way to provide, for every node of a node of a state space tree, a bound on the best value of the objective function on any solution that can be obtained by adding further components to the partial solution represented by the node.

The value of the best solution seen so far.

14.Differentiate backtracking and branch bound techniques.

- Backtracking is applicable only to non optimization problems.
- Backtracking generates state space tree in depth first manner.
- Branch and bound is applicable only to optimization problem.
- Branch and bound generated a node of state space tree using best first rule.

15.State 8 – Queens problem.

The problem is to place eight queens on a 8 x 8 chessboard so that no two queen

"attack" that is, so that no two of them are on the same row, column or on the diagonal.

16.State Sum of Subsets problem.

Given n distinct positive numbers usually called as weights , the problem calls for finding all the combinations of these numbers whose sums are m.

17.State m – colorability decision problem.

Let G be a graph and m be a given positive integer. We want to discover whether the nodes of G can be colored in such a way that no two adjacent nodes have the same color yet only m colors are used.

18.Define chromatic number of the graph.

The m – colorability optimization problem asks for the smallest integer m for which the graph G can be colored. This integer is referred to as the chromatic number of the graph.

19.Define a planar graph.

A graph is said to be planar iff it can be drawn in such a way that no two edges cross each other.

20.What are NP- hard and Np-complete problems?

The problems whose solutions have computing times are bounded by polynomials of small degree.

21.What is a decision problem?

Any problem for which the answer is either zero or one is called decision problem.

22.What is maxclique problem?

A maxclique problem is the optimization problrm that has to determine the size of a largest clique in Grapg G where clique is the maximal subgraph of a graph.

23.What is approximate solution?

A feasible solution with value close to the value of an optimal solution is called approximate solution.

24. Give the purpose of lower bound. (MAY / JUNE 2016)

- The elements are compared using operator < to make selection.
- Branch and bound is an algorithm design technique that uses lower bound comparisons.
- Main purpose is to select the best lower bound.
- Example: Assignment problem and transportation problem.

25. What is The Euclidean minimum spanning tree problem? (MAY / JUNE 2016)

The Euclidean minimum spanning tree or EMST is a minimum spanning tree of a set of n

points in the plane where the weight of the edge between each pair of points is the Euclidean distance between those two points. In simpler terms, an EMST connects a set of dots using lines such that the total length of all the lines is minimized and any dot can be reached from any other by following the lines.

26.How NP-Hard problems are different from NP-Complete? (Apr/May 2015)

Approximate algorithms is a different approach to solve NP-complete optimization problems.The use of fast (polynomially bounded) algorithms that are not guaranteed to give the best solution but will give one that is close to the optimal.

One of the way to describe approximation algorithm is the ratio between the value of the algorithm's output and the value of an optimal solution.

Let A be an approximation algorithm, we denote A(I) the feasible solution A chooses for

input I.we define:

$$\mathbf{rA(I) = val\frac{I, A(I)}{opt(I)}\ for\ minimization\ problem}$$

$$\mathbf{rA(I) = val\frac{opt\ (I)}{val(I, A(I))}\ for\ maximization\ problem}$$

Where rA(I)≥ 1. From the above equations worst case ratio is defined using the functions.

$R_A(m)$= max { rA(I)| I such that opt(I)=m }

$S_A(n)$ = max { rA(I) | I of size n }

$R_A(m)$ – infinite sum for some m(i.e) when the maximum ratio is not well defined the set of inputs being considered as infinite.

27.Define Hamiltonian Circuit problem (Apr/May 2015)

Given an undirected and connected graph and only two nodes x & y. we have to find a path from x to y by visiting all the nodes only ones.

A Hamiltonian circuit exists, it starts at vertex a. This problem can be solved using backtracking approach.

Make vertex **a** root of the state space tree. The state space tree is generated in order to find all the Hamiltonian cycles in the graph.

For instance a-b-c-d-e-f (dead end), a-d-c-e-d (dead end) and a-b-c-e –f (dead end) because this path is not reach the root vertex **a**

For instance a-b-f-e-c-d-a (Hamiltonian circuit) because this path reach the root vertex **a**.

28.Draw the decision tree for comparison of three values (Nov/Dec 2015)

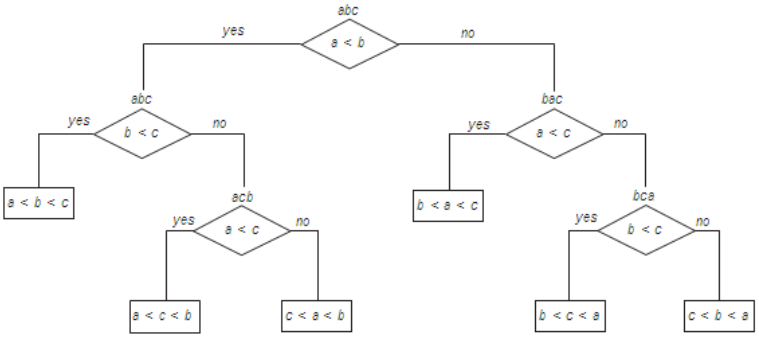


FIGURE 11.3 Decision tree for the three-element insertion sort.

PART-B

1. Suggest an approximation algorithm for travelling salesperson problem. Assume that the cost function satisfies the triangle inequality. 8 marks (Apr/May 2015)

Approximation algorithm:

Polynomial-time approximation algorithm is said to be a ***c-approximation algorithm***, where c = 1, if the accuracy ratio of the approximation it produces does not exceed c for any instance of the problem in question:

$$r(S_a) \leq C. \quad [C= \text{constant}]$$

Approximation algorithm for travelling salesperson problem:

- A polynomial-time approximation algorithm with a finite performance ratio on all instances of the traveling salesman problem.
- As the following theorem shows, the answer turns out to be no, unless P = NP.

THEOREM 1

If P ≠ *NP*, there exists no c-approximation algorithm for the traveling salesman problem, i.e., there exists no polynomial-time approximation algorithm for this problem so that for all instances

$$f(S_a) \leq C\ f(S^*)$$

PROOF :

An approximation algorithm A and a constant c exist

We will show that this algorithm could then be used for solving the Hamiltonian circuit problem in polynomial time.

Let G be an arbitrary graph with n vertices.

We map G to a complete weighted graph G' by assigning weight 1 to each edge in G and adding an edge of weight cn + 1 between each pair of vertices not adjacent in G.

If G has a Hamiltonian circuit, its length in G' is n; hence, it is the exact solution **S *** to the traveling salesman problem for G'.

If **S** is an approximate solution obtained for G' by algorithm A, then f(s) ≤ cn by the assumption.

If G does not have a Hamiltonian circuit in G, the shortest tour in G' will contain at least one edge of weight cn + 1, and hence **$f(S_a) \geq f(S^*) > cn$.**

Satisfies the triangle inequality:

Taking into account the two derived inequalities, we could solve the Hamiltonian circuit problem for graph G in polynomial time by mapping G to G' , applying algorithm A to get tour s_a in G' , and comparing its length with cn.

Since the Hamiltonian circuit problem is *NP*-complete, we have a contradiction unless

P =*NP*.

2. Explain how job assignment problem could be solved, given n tasks and n agents where each agent has a cost to complete each task, using Branch and Bound technique. 8 marks (Apr/May 2015)

Assignment Problem:

- We want to assign *n* people to *n* jobs so that the total cost of the assignment is as small as possible (lower bound)

Select one element in each row of the cost matrix *C* so that:

- no two selected elements are in the same column; and

- the sum is minimized

For example:

	Job 1	*Job* 2	*Job* 3	*Job* 4
Person *a*	9	2	7	8
Person *b*	6	4	3	7
Person *c*	5	8	1	8
Person *d*	7	6	9	4

Branch and Bound technique:

Lower bound: Any solution to this problem will have total cost of at least: sum of the smallest element in each row = 2 + 3 + 1 + 4 =10

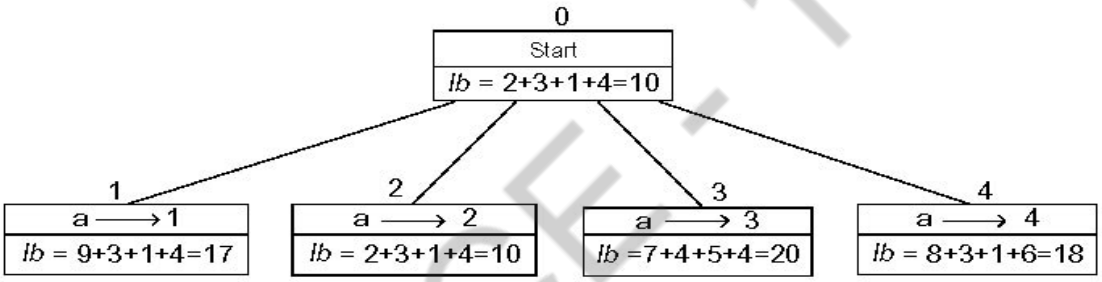


Figure 11.5 Levels 0 and 1 of the state-space tree for the instance of the assignment problem being solved with the best-first branch-and-bound algorithm. The number above a node shows the order in which the node was generated. A node's fields indicate the job number assigned to person *a* and the lower bound value, *lb*, for this node.

State-space levels 0, 1, 2

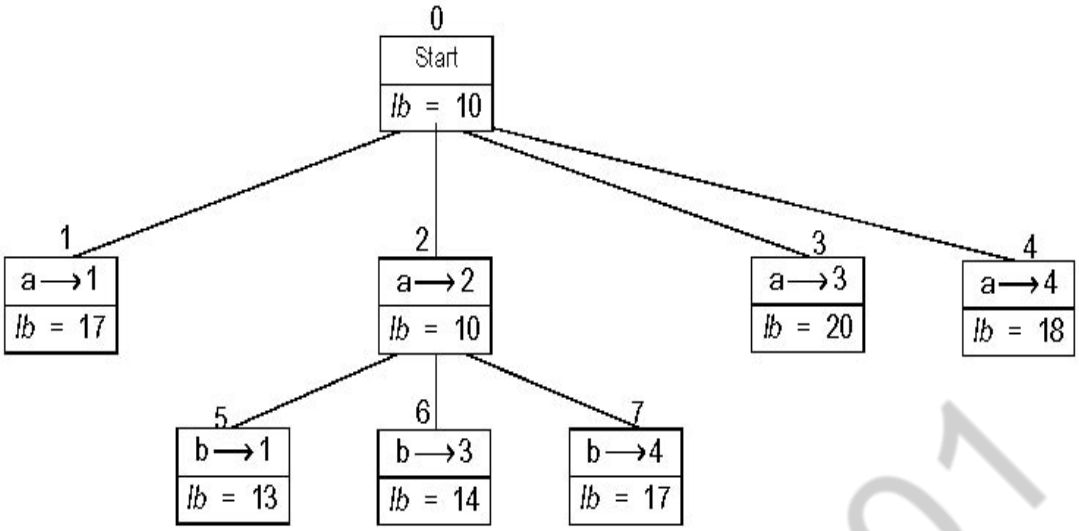


Figure 11.6 Levels 0, 1, and 2 of the state-space tree for the instance of the assignment problem being solved with the best-first branch-and-bound algorithm

Complete state-space:

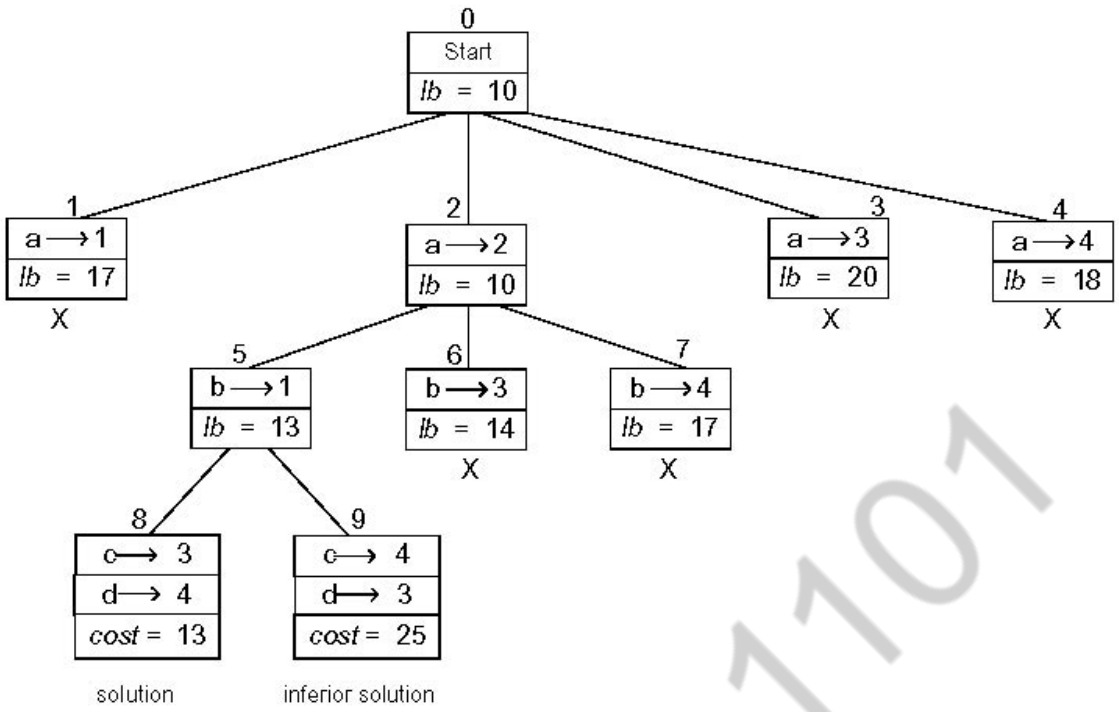


Figure 11.7 Complete state-space tree for the instance of the assignment problem solved with the best-first branch-and-bound algorithm

Node 9, correspondents to the feasible solution { a → 2, b → 1, c → 4, d → 3} with the total cost of 25.

Cost is larger than the cost of the solution represented by leaf 8 , node 9 is simply terminated.

3. The knight is placed on the first block of an empty board and, moving according to the rules of chess, must visit each square exactly once. Solve the

above problem using backtracking procedure. 10 marks (Apr/May 2015)

Backtracking is an algorithm paradigm that tries different solutions until find a solution that "works".

Barcktracking works in incremental way and is an optimization over the native solution.

The knight is placed on the first block of an empty board and, moving according to the rules of chess, must visit each square exactly once.

Naïve algorithm for knight's tour

The naïve algorithm is to generate all tours one by one and check if the generated tour satisfies the constraints.

```
While there are untried tours
{
  Generate the next tour
  If this tour covers all squares
 {
    Print this path;
}
}
```

Backtracking algorithm for knight's tour

Following is the backtracking algorithm for knight's tour problem.

```
if all squares are visited
        Print the solution
else
```

a. Add one of the next moves to solution vector and recursively ckecj if this move leads to a solution.
b. If the move chosen in the above step doesn't lead to a solution then remove this move from the solution vector and try other alternative moves
c. If none of the alternatives work then return false

4. Implement an algorithm for knapsack problem using NP-Hard approach. 6 marks (Apr/May 2015)

The knapsack problem, another well-known *NP*-hard problem, given n items of known weights w and values v1,...,vn) = f(s),...,w and a knapsack of weight capacity W, find the most valuable subset of the items that fits into the knapsack. We saw how this problem can be solved by exhaustive search, dynamic programming,and branch-and-bound. Now we will solve this problem by approximation algorithms.

EXAMPLE 5 Let us consider the instance of the knapsack problem with the knapsack capacity 10 and the item information as follows:

item	weight	value
1	7	$42
2	3	$12
3	4	$40
4	5	$25

Computing the value-to-weight ratios and sorting the items in nonincreasing order of these efficiency ratios yields

item	weight	value	value/weight
1	4	$40	10
2	7	$42	6
3	5	$25	5
4	3	$12	4

- To learn about branch-and-bound, first we look at breadth-first search using the knapsack problem

- Then we will improve it by using best-first search.
- Remember the default strategy for the 0-1 knapsack problem was to use a depth-first strategy, not expanding nodes that were not an improvement on the previously found solution.
- 0-1 Knapsack using the branch and bound.
- Now look at all promising, unexpanded nodes and expand beyond the one with the best bound.
- We often arrive at an optimal solution more quickly than if we simply proceeded blindly in a predetermined order.

Computing Upper Bound:

(1) The worst pay off per weight unit.

$v_1/w_1 \geq v_2/w_2 \geq \ldots \geq v_n/w_n$

(2) upper bound function can be computed using following formula:

$$Ub = v+(W-w)(v_{i+1}/w_{i+1})$$

Example:

- Capacity W is 10

Item	Weight	Value	Value / weight
1	4	$40	10
2	7	$42	6
3	5	$25	5
4	3	$12	4

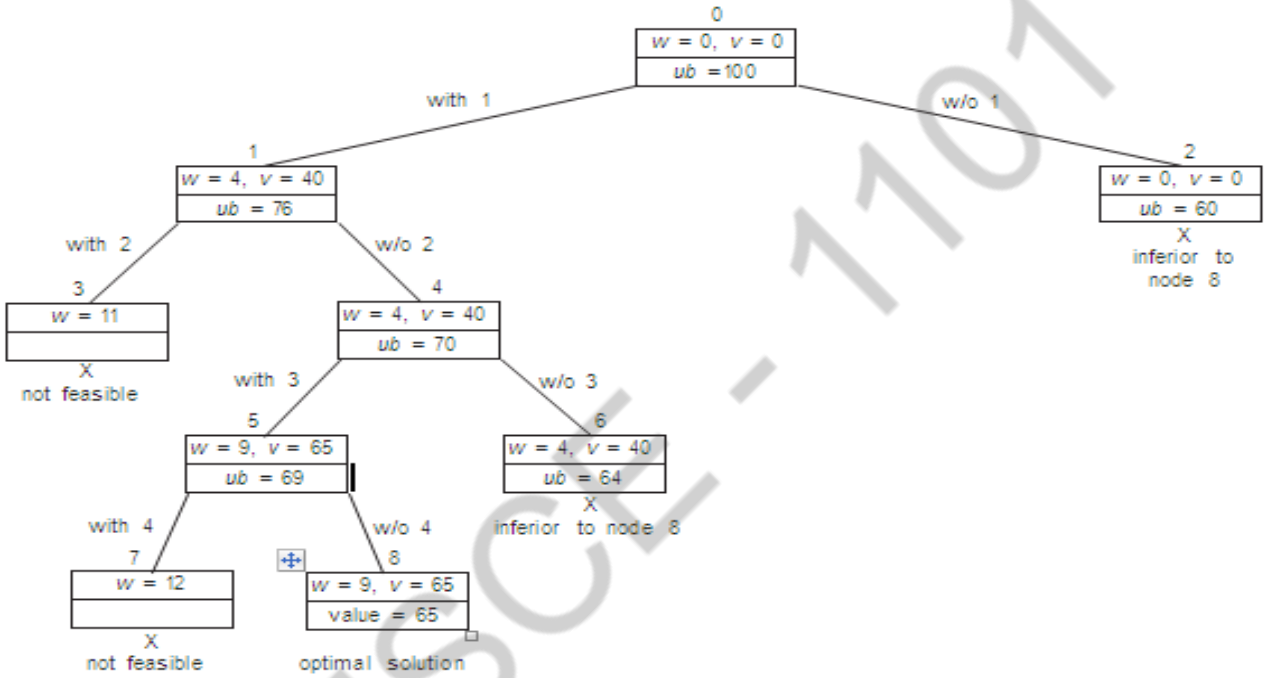


FIGURE 12.8 State-space tree of the best-first branch-and-bound algorithm for the instance of the knapsack problem.

- To compute the upper bound, use

o ub = v + (W – w)(v_{i+1}/w_{i+1})

- So the maximum upper bound is

o pick no items, take maximum profit item

o ub = 0+(10 – 0)*($10) = $100

- After we pick item 1, we calculate the upper bound as v=40 and w= 4

o all of item 1 (4, $40) + partial of item 2 (7, $42)

o $40 + (10-4)*6 = $76

- If we don't pick item 1: now w=0 and v=0

o ub = (10 – 0) * ($6) = $60

- So the maximum upper bound is pick no items, take maximum profit item

– $40 + (10-4)*6 = $76

- After we pick item 2, we calculate the upper bound as w=(w1+w2) =(4+7)=11 v=(v1+v2)=(40+42) =82

o This exceed the capacity of knapsack. Hence we will not consider this node

- If we don't pick item 2: w = 4 and v = 40

o all of item 2 (7, $42) + partial of item 3 (5, $25)

o ub= $42 + (10-7)*5 = $70

- So the maximum upper bound is pick no items, take maximum profit item

$42 + (10-7)*5 = $70

- After we pick item 3, we calculate the upper bound as w=(w1+w3)5+4= 9 and v= (v1+v3)=40+25=65

o all of item 3 (9, $65) + partial of item 4 (3, $12)

o ub= $65 + (10-9)*4 = $69

- If we don't pick item 3: we calculate the upper bound as w= 4 and v= 40

o all of item 3 (4, $40) + partial of item 4 (3, $12)

o ub= $40 + (10-4)*4 = $60

- So the maximum upper bound is pick no items, take maximum profit item

o ub= $65 + (10-9)*4 = $69

- After we pick item 4, we calculate the upper bound as w=(w3+w4)= 9+3 =12 and v= (v3+v4)=65+12=76

- This exceed the capacity of knapsack. Hence we will not consider this node

- If we don't pick item 4: we calculate the upper bound as w= 9 v=65

o all of item 4 (9, $65) + partial of item 5 (0, $0)

o ub= $65 + (10-9)*0 = $65

o Thus solution is pick up { item1 , item 3} and gain maximum profit $65

5. Using an example prove that, satisfiability of Boolean formula in 3-conjunctive Normal Form is NP-complete 12 marks (Nov/Dec 2015)

Approximate algorithm for NP-Hard problems:

Approximate algorithms is a different approach to solve NP-complete optimization problems.The use of fast (polynomially bounded) algorithms that are not guaranteed to give the best solution but will give one that is close to the optimal.

One of the way to describe approximation algorithm is the ratio between the value of the

algorithm's output and the value of an optimal solution.

Let A be an approximation algorithm, we denote A(I) the feasible solution A chooses for input I.we define:

$$\mathbf{rA(I) = val\frac{I, A(I)}{opt(I)}\ for\ minimization\ problem}$$

$$\mathbf{rA(I) = val\frac{opt\ (I)}{val(I, A(I))}\ for\ maximization\ problem}$$

Where rA(I)≥ 1. From the above equations worst case ratio is defined using the functions.

$R_A(m)$= max { rA(I)| I such that opt(I)=m }

$S_A(n)$ = max { rA(I) | I of size n }

$R_A(m)$ – infinite sum for some m(i.e) when the maximum ratio is not well defined the set of inputs being considered as infinite.

Accuracy measures:

Two types of accuracy measures mathematically defined as:

1. Accuaracy ratio of an approximate solution sa

r(sa) = f(sa) /f(s*) for minimization problems

r(sa) = f(s*) / f(sa) for maximization problems

where f(sa) and f(s*) are values of the objective function f for the approximate solution sa and actual optimal solution s*

2. Performance ratio of the algorithm A the lowest upper bound of r(sa) on all instances

6. Show that the Hamiltonian Path problem reduces to the Hamiltonian Circuit Problem and vice versa. 10 marks (Nov/Dec 2015)

Hamiltonian Circuit problem:

Given an undirected and connected graph and only two nodes x & y. we have to find a path from x to y by visiting all the nodes only ones.

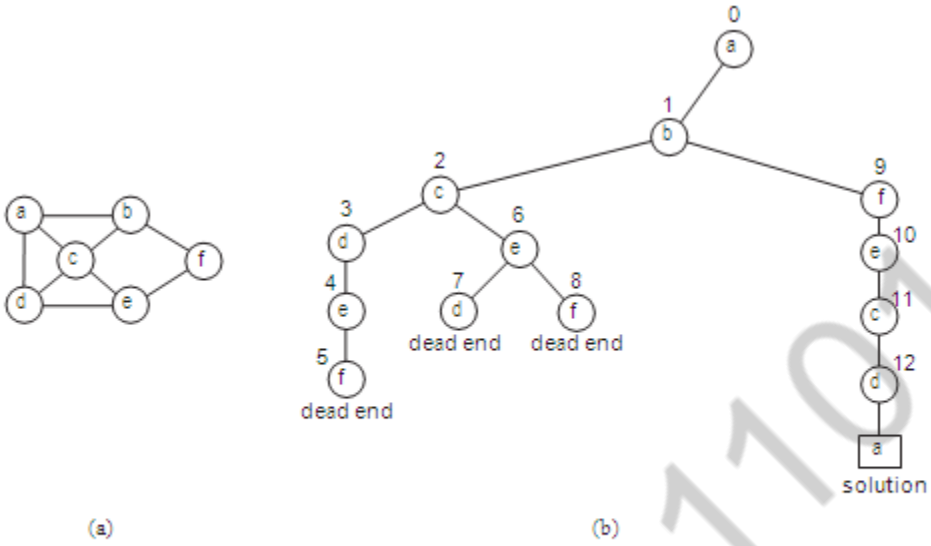


FIGURE 12.3 (a) Graph. (b) State-space tree for finding a Hamiltonian circuit. The numbers above the nodes of the tree indicate the order in which the nodes are generated.

A Hamiltonian circuit exists, it starts at vertex a. This problem can be solved using backtracking approach.

Make vertex **a** root of the state space tree. The state space tree is generated in order to find all the Hamiltonian cycles in the graph.

For instance a-b-c-d-e-f (dead end), a-d-c-e-d (dead end) and a-b-c-e –f (dead end) because this path is not reach the root vertex **a**

For instance a-b-f-e-c-d-a (Hamiltonian circuit) because this path reach the root vertex **a**.

Subset- Sum Problem:

Let s = {s1, s2... sn} be a set of n positive integers then we have to find a subset whose sum is equal to given positive integer d.

Algorithm:

Step 1: sort the set element in ascending order

Step 2: start with an empty set

Step 3: Add to the subset next element from the list

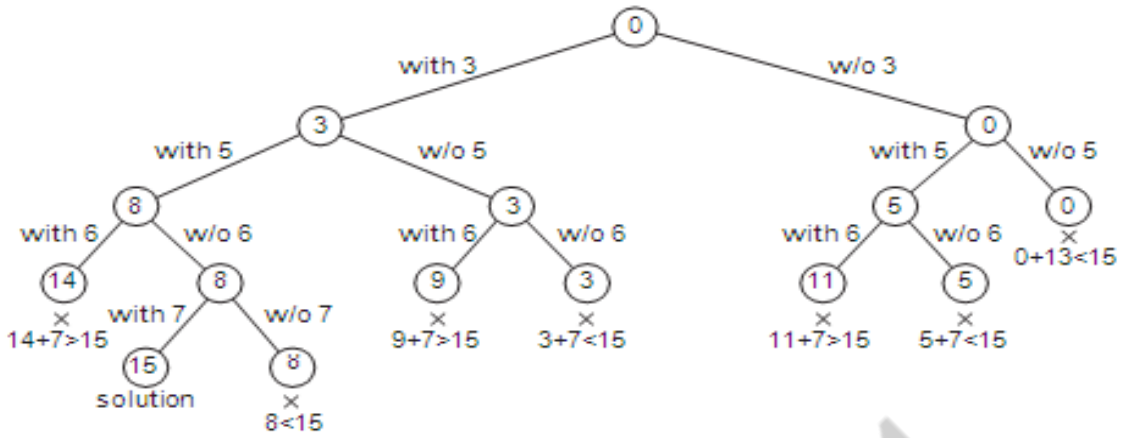
Step 4: If the subset is having sum d then stop with that subset as the solution

Step 5: It the subset is not feasible or if we have reached is not feasible or if we have reached at the end of the set then backtrack through the subset until we find the suitable solution.

Step 6: If the subset is feasible then repeat 2.

Step 7: If we have visited all the elements without finding a suitable subset and if no backtracking is possible then stop without solution.

Example: Complete state-space tree of the backtracking algorithm applied to the instance $A = \{3, 5, 6, 7\}$ and $d = 15$ of the subset-sum problem. The number inside a node is the sum of the elements already included in the subsets represented by the node. The inequality below a leaf indicates the reason for its termination.



First solution { 3,5,7}

We can terminate the node as nonpromising if either of the following two inequalities holds:

$s' + s_{i+1} > d$ (the sum's is too large)

$s' + \sum_{j=i+1k=0}^{n} sj < d$ (the sum s' is too small)

7. What is an approximation algorithm? Give example 6 marks (Nov/Dec 2015)

Approximate algorithm for NP-Hard problems:

Approximate algorithms is a different approach to solve NP-complete optimization problems.The use of fast (polynomially bounded) algorithms that are not guaranteed to give the best solution but will give one that is close to the optimal.

One of the way to describe approximation algorithm is the ratio between the value of the algorithm's output and the value of an optimal solution.

Let A be an approximation algorithm, we denote A(I) the feasible solution A chooses for input I.we define:

$$\mathbf{rA(I) = val\frac{I, A(I)}{opt(I)}\ for\ minimization\ problem}$$

$$\mathbf{rA(I) = val\frac{opt\ (I)}{val(I, A(I))}\ for\ maximization\ problem}$$

Where rA(I)≥ 1. From the above equations worst case ratio is defined using the functions.

$R_A(m)$= max { rA(I)| I such that opt(I)=m }

$S_A(n)$ = max { rA(I) | I of size n }

$R_A(m)$ – infinite sum for some m(i.e) when the maximum ratio is not well defined the set of inputs being considered as infinite.

Accuracy measures:

Two types of accuracy measures mathematically defined as:

3. Accuaracy ratio of an approximate solution sa

r(sa) = f(sa) /f(s*) for minimization problems

r(sa) = f(s*) / f(sa) for maximization problems

where f(sa) and f(s*) are values of the objective function f for the approximate solution sa and actual optimal solution s*

4. Performance ratio of the algorithm A the lowest upper bound of r(sa) on all instances

Approximate Algorithm for the Traveling slaesman problem:

The approximation algorithm for TSP is done with different heuristics.They are

- Nearest neighbour
- Multifragment

- Christofides
- 2-opt
- 3-opt
- Lin- Kernighan

Nearest neighbour algorithm:

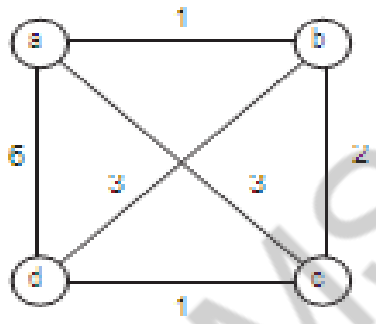
The nearest neighbour is a greedy method for approximating a solution to the travelling salesman problem. The performance ratio is unbounded above, even for the important subset of Euclidean graphs.

Step 1: choose an arbitrary city as the start.

Step 2: Repeat the following operation until all the cities have been visited go to the unvisited city nearest the one visited last.

Step3: Return to the starting city.

Example:



The optimal tour route by nearest neighbour problem is a-b-c-d-a and the length is 10.

By exhaustive search the tour a-b-d-c-a yields a length of 8.

The accuracy ratio r(sa) is:

r(sa) = f(sa)/f(s) = 10/8 = 1.25 [the tour sa is 25% longer than the optimal tour s*)

Multifragment- heuristic algorithm:

TSP problem considers as the problem of finding a minimum-weight collection of edges in a given complete weighted graph so that all the vertices have degree 2.

Step1: sort the edges in non decresing order of weights.initialize the set of tour edges to be constructed to empty set

Step2: Add next edge on the sorted list to the tour, skipping those whose addition would created a vertex of degree 3 or a cycle of length less than n.

Step3 : repeat this step until a tour of length n is obtained

Example:

Appling the algorithm, to the graph in the above figure yield{(a,b)(c,d)(b,c((a,d)}

This set of edges forms the same tour as the one produced by the nearest neighbour algorithm.

The multi-fragment heuristic algorithm tends to produce significantly better tour than the nearest neighbour algorithm.

Twice around the tree algorithm:

This is an algorithm considers Hamiltonian and spanning trees of the same graph. Since removing a vertices is equivalent to the structure of spanning tree algorithm.

The sequence of steps are:

Step1: Construct a minimum spanning tree of the graph (e.g) by prims or kruskal algorithm).

Step2: Starting at an arbitrary vertex create a path that goes twice around the tree and returns to the same vertex.

Step3: create a tour from the circuit constructed in step2 by making shorcuts to avoid visiting intermediate vertices more than once.

Example:

Let us apply this algorithm to the graph in figure.The minimum spanning tree of this graph is made up of edges (a,b) (b,c) (b,d) and (d,e). A twice round the tree walk that start and ends art a is (a,b,c,b,d,e,d,b,a.

Eliminating the second b(a shorycut from c to d), the second d, and the third b (a shorcut e to a) yeilds the hamiltonian circut a,b,c,d,e,a of length 39

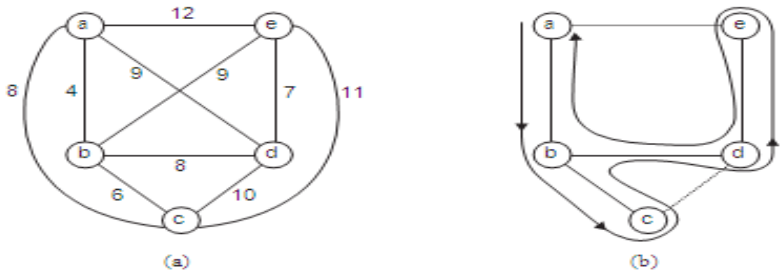


FIGURE 12.11 Illustration of the twice-around-the-tree algorithm. (a) Graph. (b) Walk around the minimum spanning tree with the shortcuts.

Tour: a-b-c-d-e-a

Christofides Algorithm:

Step1: construct a minimum spanning tree of the graph

Step 2: Add edges of a minimum weight matching of all the odd vertices in the minimum

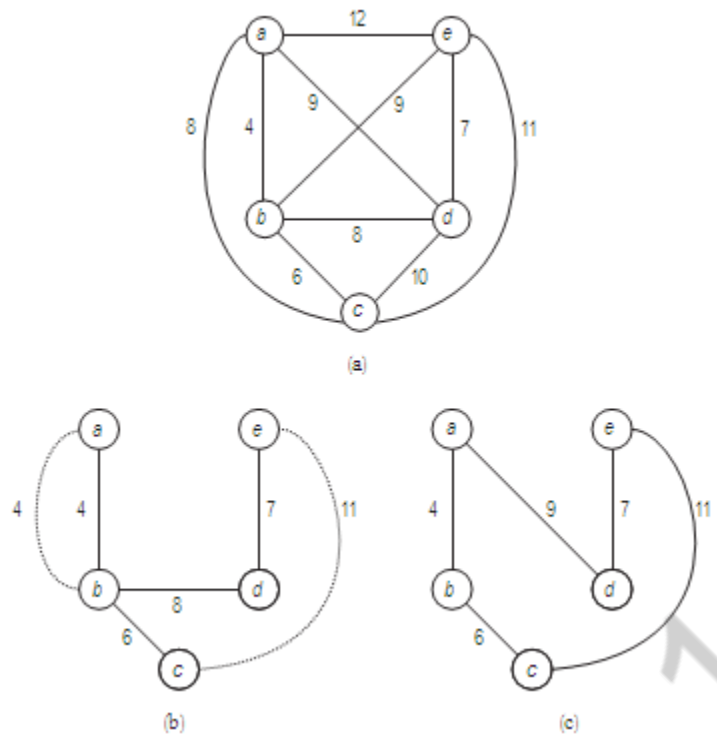
spanning tree

Step3: Find an eulerian circuit of the multigraph obtained in step2

Step4: Create a tour from the path constructed in step2 by making shortcuts to avoid visiting

intermediate vertices more than once.

Example:



Euclidean Instances:

Theorem: if p≠ NP, ther exists no approximation algorithm for TSP with a finite performance ratio.

Definition : An instance of TSP is called Euclidean, if its distances satifiey two conditions:

1. **Symmetry:** d[i,j] = d[j,i] for any pair of cities i and j

2. **Trianfle inequality** d[i,j] ≤ d[i,k] + d[k,j] for any cities i,j,k

Local Search heuritics for TSP:

- Start with some inital tour (e.g. nearest neighbor).

- On each iteration explore the current tour's neighborhood by exchanging a few edges in it.
- If the new tour is shorter, make it the current tour.
- Otherwise consider another edge change. If no change yeilds a shorter tour, the current tour is returned as the output.

Example:

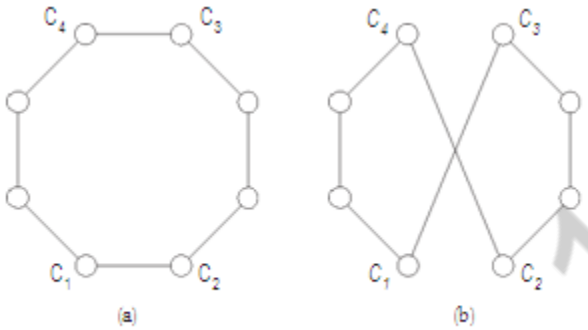


FIGURE 12.13 2-change: (a) Original tour. (b) New tour.

Approximation Algorithm for the knapsack problem:

- There are N items and each item has a weight(w) and value(v)
- There is a bag can hold a maximum weight W.
- The item should be chosen into bag to maximum the total value while still keeping the whole weight less than W.

Greedy Algorithms for Knapsack problem:

The first approach is to select the items in decreasing order of their value to weight ratio. Then compute the value to weight ratio vi/wi, i=1,2,..n and select the item in decreasing order of these ratios.

Greedy algorithm for the discrete knapsack problem:

Step1: compute the value to weight ratios ri = vi/wi i=1,2,..n for the items given

Step2: sort the items in non increasing order of the ratio computed in step1.

Step3 : Repeat the following operation until no item in left in the sorted list.

If the current item on the list fit into knapsack, place it otherwise proceed to the next

item.

Example: W =10 (capacity)

Item	Weight	Value
1	7	$42
2	3	$12
3	4	$40
4	5	$25

Arrange the given items in non increasing order vi/wi.

The greedy algorithm will select the item with weight 4 ,skip item weight 7(W=10)

Item	Weight	Value	Value / weight
1	4	$40	10

2	7	$42	6
3	5	$25	5
4	3	$12	4

- The item sets in the knapsack will be { item 1 and item 3}
- The solution seems to be optimal to this particular instance of the problem.
- But this method does not fix a finite upper bound on the accuracy

Greedy Algorithm for the continuous knapsack problem:

Step1: compute the value to weight ration vi/wi, i=1,2..n for the item given

Step2: sort the items in non increasing order of the ratios computed in step1.

Step3 : Repeat the following operation until no item in left in the sorted list.

If the current item on the list fit into knapsack, place it otherwise Take its largest

Fraction to fill the knapsack to its full capacity and stop

Example: same problem

In discrete algorithm either the item will be take or not take.

But in continuous knapsack problem, the algorithm will take the first item of weight 4 and then 6/7 of the next item on the sorted list to fill the knapsack to its capacity.

This algorithm always yield an optimal solution to the continuous knapsack problem.

Approximation Schemes :

The approximation algorithm for the discrete knapsack problem has an accuracy level:

$$\frac{f(s*)}{f(s)} \leq 1 + \frac{1}{k}$$

k= is an integer parameter in the range 0≤ k ≤ n and n is the size.

Knapsack Algorithm : that generates all the subset of k items or less and for each one that fits into knapsack it adds the remaining in non increasing order of their value to weight ratios.

The subset of the highest value obtained in this fashion is returned as the algorithm's output.

Let us find the solution for the previously mention knapsack problem with capacity =10 using Knapsack algorithm

Item	Weight	Value	Value / weight
1	4	$40	10
2	7	$42	6
3	5	$25	5
4	3	$12	4

Subset	**Added item**	**Value**
Ø	1,3,4	$69
1	3,4	$69
2	4	$46
3	1,4	$69
4	1,3	$69
[1,2]	Not feasible	
[1,3]	4	$69
[1,4]	3	$69
[2,3]	Not feasible	
[2,4]		$46
[3,4]	1	$69

The algorithm yield {1,3,4} which is the optimal solution for this instance.

The total number of subsets the algorithm generates before adding extra elements is

$$\sum_{j=0}^{K} \binom{n}{j} = \sum_{j=0}^{K} n(n-1)\ldots\frac{n-j+1}{j!} \leq \sum_{j=0}^{K} \binom{n}{j} \leq \sum_{j=0}^{K} n3 = (k+1)n^k$$

This takes O(n) time to find all the possible subsets.

The algorithm's effieciency is in $O(kn^{k+1})$

8. Give any five undecidable problems and explain the famous halting problem. (16) (MAY / JUNE 2016) (Out of Syllabus)

· Halting Problem.

· Post correspondence problem.

· Hilbert's tenth problem: the problem of deciding whether a Diophantine equation (multivariable polynomial equation) has a solution in integers.

· Determining if a context-free grammar generates all possible strings, or if it is ambiguous.

· The word problem in algebra and computer science.

· The word problem for certain formal languages.

· Determining whether a λ-calculus formula has a normal form.

In computability theory and computational complexity theory, an undecidable problem is a decision

problem for which it is known to be impossible to construct a single algorithm that always leads to

a correct yes-or-no answer.

Some decision problems cannot be solved at all by any algorithm. Such problems are

called

undecidable, as opposed to decidable problems that can be solved by an algorithm. A famous

example of an undecidable problem was given by Alan Turing in 1936.1 The problem in question

is called the halting problem: given a computer program and an input to it, determine whether the

program will halt on that input or continue working indefinitely on it.

Here is a surprisingly short proof of this remarkable fact. By way of contradiction, assume that A is

an algorithm that solves the halting problem. That is, for any program P and input I,

$$A(P, I) = \begin{cases} 1, & \text{if program } P \text{ halts on input } I; \\ 0, & \text{if program } P \text{ does not halt on input } I. \end{cases}$$

We can consider program P as an input to itself and use the output of algorithm A for pair (P, P) to construct a program Q as follows:

$$Q(P) = \begin{cases} \text{halts}, & \text{if } A(P, P) = 0, \text{ i.e., if program } P \text{ does not halt on input } P; \\ \text{does not halt}, & \text{if } A(P, P) = 1, \text{ i.e., if program } P \text{ halts on input } P. \end{cases}$$

Then on substituting Q for P, we obtain

$$Q(Q) = \begin{cases} \text{halts}, & \text{if } A(Q, Q) = 0, \text{ i.e., if program } Q \text{ does not halt on input } Q; \\ \text{does not halt}, & \text{if } A(Q, Q) = 1, \text{ i.e., if program } Q \text{ halts on input } Q. \end{cases}$$

This is a contradiction because neither of the two outcomes for program Q is possible, which

completes the proof.

9. State the subset-sum problem and Complete state-space tree of the backtracking

algorithm applied to the instance A={3, 5, 6, 7} and d=15 of the subset-sum problem. (MAY / JUNE 2016)

The ***subset-sum problem*** finds a subset of a given set $A = \{a1, \ldots, an\}$ of n positive integers whose sum is equal to a given positive integer d. For example, for $A = \{1, 2, 5, 6, 8\}$ and

$d = 9$, there are two solutions: {1, 2, 6} and {1, 8}. Of course, some instances of this problem may

have no solutions.

It is convenient to sort the set's elements in increasing order. So, we will assume that

$a1 < a2 < \ldots < an$.

A = {3, 5, 6, 7} and d = 15 of the subset-sum problem. The number inside a node is the sum

of the elements already included in the subsets represented by the node. The inequality below a leaf

indicates the reason for its termination.

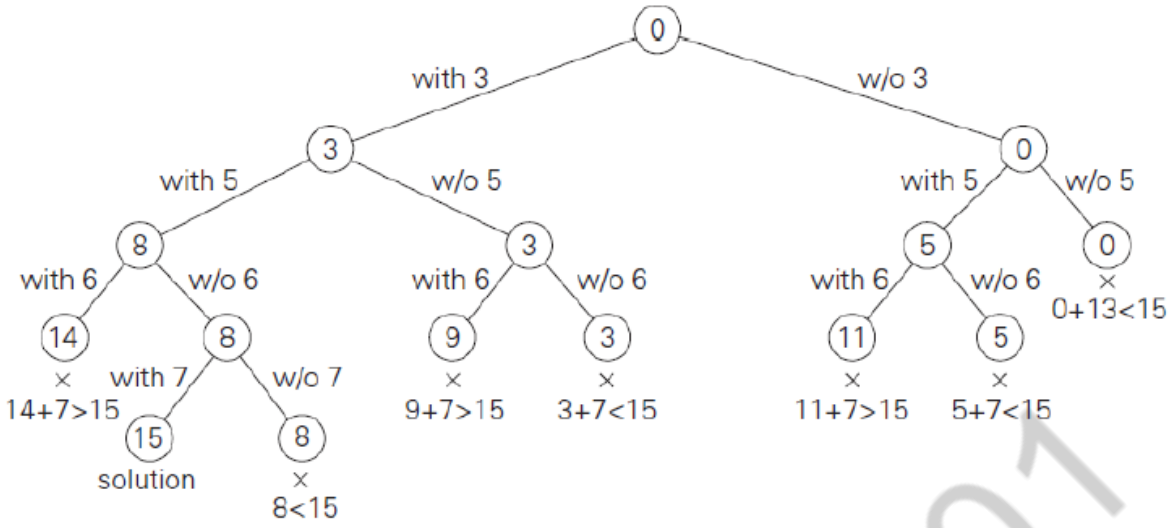


FIGURE Complete state-space tree of the backtracking algorithm applied to the instance

Example:

· The state-space tree can be constructed as a binary tree like that in Figure 5.15 for the instance $A = \{3, 5, 6, 7\}$ and $d = 15$.

· The root of the tree represents the starting point, with no decisions about the given elements made as yet.

· Its left and right children represent, respectively, inclusion and exclusion of $a1$ in a set being sought. Similarly, going to the left from a node of the first level corresponds to inclusion of $a2$ while going to the right corresponds to its exclusion, and so on.

· Thus, a path from the root to a node on the ith level of the tree indicates which of the first I numbers have been included in the subsets represented by that node.

· We record the value of s, the sum of these numbers, in the node.

· If s is equal to d, we have a solution to the problem. We can either report this result

and stop or, if all the solutions need to be found, continue by backtracking to the node's parent.

· If *s* is not equal to *d,* we can terminate the node as non-promising if either of the following two inequalities holds:

$$s + a_{i+1} > d \quad \text{(the sum } s \text{ is too large)},$$

$$s + \sum_{j=i+1}^{n} a_j < d \quad \text{(the sum } s \text{ is too small)}.$$