Hall Ticket Number:

14CS IT404

II/IV B.Tech (Regular/Supplementary) DEGREE EXAMINATION

April, 2017

Fourth Semester

Time: Three Hours

Common for CSE & IT

Design And Analysis of Algorithms

Maximum: 60 Marks

Answer Question No.1 compulsorily. (1X12 = 12 Marks)

Answer ONE question from each unit. (4X12=48 Marks)

1 Answer all questions (1X12=12 Marks)

a) What is a pseudo code?
b) What is the time complexity of Quick Sort?
c) Define optimality principle.
d) What is spanning tree?
e) What is the difference between Greedy method and Dynamic Programming?
f) List various representations of graph.
g) What is e-node?
h) What is a stack?
i) What is articulation point?
j) What is the difference between backtracking and Branch and Bound?
k) What is feasible solution?
l) What is P and NP?

UNIT I

2. What is time complexity and discuss various methods of evaluating time complexity of algorithm in detail by illustrating with examples. 12M

(OR)

3. a) Write short notes on Recursive algorithms. 4M
   b) Explain Merge Sort algorithm to sort the list of elements using Divide and Conquer technique. 8M

UNIT II

4. Write short notes on Minimum Cost Spanning Tree problem. Explain with example Kruskal's algorithm for finding minimum-cost spanning tree. 12M

(OR)

5. a) Write and explain the general method of Greedy method. 6M
   b) What is 0/1 Knapsack problem? Define merging and purging rules of O/1 Knapsack problem. 6M

UNIT III

6. a) Explain briefly about Breadth First Search and write the pseudocode for Breadth First Search . 8M
   b) Write short notes on ~~Biconnected~~ Strongly connected components. 4M

(OR)

7. a) Write the control abstraction of backtracking and write backtracking algorithm for n-queen problem. 8M
   b) Define the following i)Problem state ii)Answer state iii)State space tree 4M

UNIT IV

8. a) Explain the following
   i)Control Abstractions for LC – search.
   ii)FIFO branch and Bound
   iii)LC Branch and Bound. 12M

(OR)

9. a) Explain the method of reduction to solve TSP problem using Branch and Bound. 8M
   b) Write short notes on Complexity measures. 4M

a) What is a pseudo code? → 1M

→ pseudo code notation is one of the way to describe an algorithm pseudo code notations contain part english and some part of mathematical operations.

b) What is the time complexity of Quick sort? → 1M

→ The time complexity of Quick sort in best and average case is $O(n\log n)$.

The time complexity of Quicksort in worst case is $O(n^2)$.

c) Define optimality Principle → 1M

→ Solutions which maximize or minimizes objective function are called optimal solution.

Maximizes objective function means getting maximum resultant value as a solution.

Minimizes objective function means getting minimum resultant value as a solution.

This Principle is known as optimality Principle.

d) What is spanning tree? → 1M

→ A spanning tree of a graph is just a subgraph that contains all the vertices and is a tree. A graph may have many spanning trees.

e) What is the difference between Greedy method and dynamic Programming? → 1M

Greedy	Dynamic.
1. Only one decision takes place at a time	1. More than one decision takes place at a time.
2. It is also called "top down approach"	2. It is also called "bottom up approach"
Applications i) Fractional knapsack problem ii) job sequence with deadline	Applications i) 0-1 knapsack problem ii) Longest common subsequence.

f) List various representations of graph? → 1M

Adjacency list, Adjacency matrix

g) What is e-node? → 1M

→ An e-node is nothing but a live node, it will generate children.

h) What is a stack? → 1M

Stack is a order of collection of homogeneous data item in which insertion and deletion takes place at same end.

Stack is also known as LIFO (Last in First out)

i.e Last inserted element is deleted first.

What is articulation point? → 1M

If the deletion of vertex 'v' from given graph, results the graph disconnects into two, Then that type of vertex 'v' is called articulation point.

What is difference between backtracking and Branch and Bound → 1M

Backtracking	Branch and Bound
1. It is applicable for decision making problems	1. It is applicable for optimising Problems
2. If you get one solution, Then we are going to stop that problem.	2. The Problem is continued until you get optimal solution.
3. It is applicable for both minimising & maximising Problems.	3. It is applicable only for minimising problems.

What is feasible solution? → 1M

The solution which satisfy implicit constraints of a problem is called feasible solution.

What is P and NP? → 1M

P

The set of problems which can be solved with in the polynomial time are called P. (Polynomial time Problems)

NP:

The set of problems which can be verified with in the polynomial time are called NP

2. What is time complexity and discuss various methods of evaluating time complexity of algorithm in detail by illustrating with examples. 10 M

→ Time Complexity: → 2M

The amount of time required to compile a program is called Time Complexity.

There are two ways to find time complexity.

1. Master's method using Decrease and conquer
2. Master's method using Divide and conquer.

1. Masters Method using Decrease and conquer. → 5M

If $T(n) = \begin{cases} aT(n-b)+f(n) & n \geq d \\ \text{constant} & n<d \end{cases}$. then

$T(n) = O(n^k)$ where $a<1$

$T(n) = O(n^{k+1})$ where $a=1$

$T(n) = O(n^k a^{n/b})$ where $a>1$

and where a is +ve , $b \geq 1$, $f(n) = n^k$.

Example:

$$T(n) = \begin{cases} T(n-1)+n & n \geq 1 \\ c & n<1 \end{cases}$$

Compare this recurrence relation with Decrease and conquer recurrence relation.

we get $a=1$, $b=1$, $f(n)=n$

$n^k = n \Rightarrow \boxed{k=1}$.

If $a=1$, $T(n) = O(n^{k+1})$

$= O(n^{1+1})$

Time Complexity, $T(n) = O(n^2)$

. Master's method using Divide and Conquer. →5M

If $T(n) = \begin{cases} aT(n/b) + f(n) & n > d \\ \text{constant} & n \le d \end{cases}$

where a is +ve $b > 1$ then

1) $T(n) = \theta(n^{\log_b a})$ if $f(n) = O(n^{\log_b a - \epsilon})$ for $\epsilon > 0$

2) $T(n) = \theta[f(n)]$ if $f(n) = \Omega(n^{\log_b a + \epsilon})$ for $\epsilon > 0$

3) (i) $T(n) = \theta[n^{\log_b a} (\log n)^{k+1}]$ if $k \ge 0$

(ii) $T(n) = \theta(n^{\log_b a} \log(\log n))$ if $k = -1$

(iii) $T(n) = \theta(n^{\log_b a})$ if $k < -1$

if $f(n) = \theta(n^{\log_b a} (\log n)^k)$.

Example:

$T(n) = 8T(n/2) + n^2$.

(i) $T(n) = 8T(n/2) + n^2$.

Here $a = 8$, $b = 2$ $f(n) = n^2$.

i) $f(n) = O(n^{\log_b a - \epsilon})$

$= O(n^{\log_2 8 - \epsilon})$

$= O(n^{3\log_2 2 - \epsilon})$

$= O(n^{3-\epsilon})$

if $\epsilon = 1$, $f(n) = n^{3-1} = n^2$ then

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

$= \theta(n^{\log_2 8})$

$= \theta(n^3)$.

3. a) write a short note on Recursive algorithms. 4M. Again

The algorithm in which function calls itself is known as Recursive algorithm. (or) direct recursive.

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

factorial of a given number is one of example of recursive algorithm:

Algorithm of factorial:

```
Algorithm fact (n)
{
    if n<=1 then
        write (1);
    else
    {
        w = n * fact (n-1);
        write (" factorial is " , w);
    }
}
```

b) explain merge sort algorithm to sort the list of elements using divide and conquer technique. 8 M

Merge sort:

1. while doing merge sort we follow "Divide and Conquer" approach
2. first we divide the given list (A) into sublists by finding the mid element.
3. we find the mid element by taking the average of low and high indexes of list.

4) Again divide the left sublist of list until one element is reached.

5) After reaching the single element in the left sublist, we compare the elements in the left element in the left sublist and right element in the left sublist and merge the elements into new list (1)

6) And then go the right sublist of given list(A) and follow the same approach and we get new list (B).

7) later we merge both new list (1) and new list (2), and elements are sorted in list (B).

8) If there is any one sublist we copy the elements into list(B).

9) And we copy elements of list (B) into list (A).

10) Therefore we get list of elements in sorting order.

2+2 M

Algorithm for "Merge sort";-

```
1. while (i<=m && j<=n)
   {
        if (a[i]<a[j]){
            b[k]=a[i];
            i++;
            k++;
        }else if (a[i]>a[j]){
            b[k]=a[j];
            j++;
            k++;
        }
        else
        {   b[k]=a[i];
            i++;
            k++;
        }
}
```

4M

```
2. while (i<=m){
        b[k]= a[i];
        i++;
        k++;
   }
3. while (j<=h){
        b[k] = a[j];
        j++;
        k++;
   }
```

4. Copying elements of "b" list into the 'a' list.

```
for (i=1; i<=h; i++)
        a[i] = b[i];
```

UNIT - II

4. Write a short notes on minimum cost Spanning Tree problem. Explain with example Kruskal's algorithm for finding minimum - cost spanning tree.

→ **Minimum cost spanning tree:**

A spanning tree of a graph is just a subgraph that contains all the vertices and is a tree. A graph may have many spanning trees.

The minimum spanning tree for a given graph is the spanning tree of minimum cost for that graph.

3m

Applications

Computer networks

Telephone networks

Kruskal'
1. Thi
co

Kruskal's Algorithm:

1. This algorithm creates a forest of trees initially the forest consists of 'n' single node trees (& no edges). At each step we add '1' edge. So that it joins two trees together.

Steps:

1. The forest is constructed with each node as seperate tree.
2. The edges are placed in priority queue.
3. until we have added n-1 edges
4. extract deepest edge from queue.
5. If it forms a cycle reject it else add it to the forest which will join two together.
6. In every step will have joined '2' trees in the forest together, so that at the end there will be only one tree in 'T'.

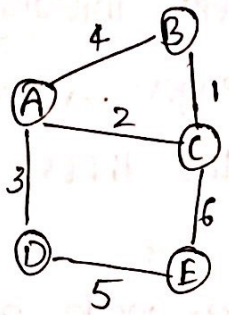
3M

Algorithm

```
Algorithm kruskal()
{
    T = ∅
    for each v ∈ V
        makeset(v)
    Sort(E) by increasing edge weight w
    for each (u,v) ∈ E (in sorted order)
    if (findset(u) != findset(v))
        T = T ∪ {u,v}
        Union(findset(u), findset(v))
}
```

3M

Example:



Step 1: Draw edges in sorted order.

B —1— C

A —2— C

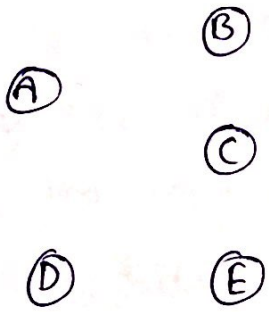
A —3— D

A —4— B

D —5— E

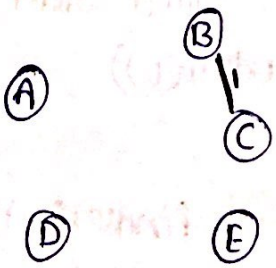
C —6— E

Step 2: Draw the vertices.



Step 3:

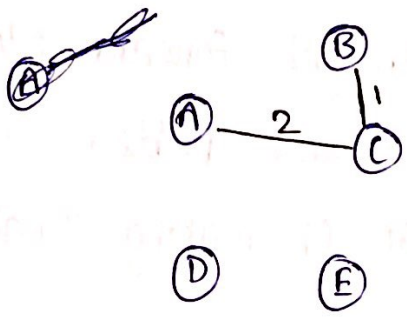
B —1— C added to the step 2.



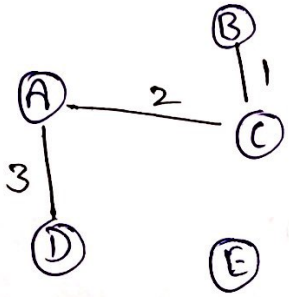
step 4:

Step4:

Now add next edge (A) —2— (C) to above step.



Step5: Add (A) —3— (D) to the above step

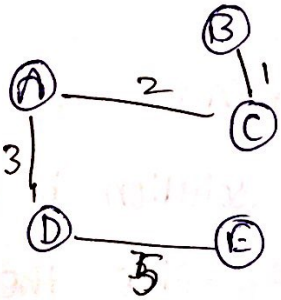


Step6: Add (A) —4— (B) to the above step.

If we Add (A) —4— (B) it forms cycle.

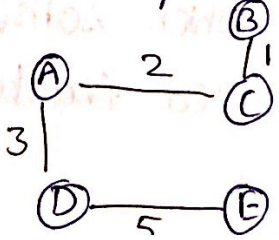
So don't add (A) —4— (B) X

Step7: Add next edge (D) —5— (E) to step 5.



Step8: Add (C) —6— (E) to step 7. But if you add (C) —6— (E) it forms cycle.

So don't add (C) —6— (E) edge.



3M

5.
a) write and explain the general method of Greedy method. What rules of

Greedy method:

Greedy method is a straight forward algorithm designing technique which solves some problems whose solutions are viewed as a result of making single decision at a time. 2M

General method of Greedy method:

```
Algorithm Greedy (A, n)
{
        Soli = 0;
        for i ← 1 to n
        {   xi = select(A)
            if feasible (x) then
                Soli = add (x, sol);
        }
        return sol;
}
```

1) we select one solution (A). 4M

2) Check whether selected solution is feasible or not.

3) If selected solution is feasible then it is added to solution set.

feasible solution:

The solution which satisfies implicit constraints of a problem is called feasible solution.

6) What is 0/1 knapsack problem? Define merging & purging rules of 0/1 knapsack problem.

0/1 knapsack problem:

Suppose there was a store which contains different types of items. Let those items be $n_1, n_2 - n_n$ types. The profits and weights of the items are $P_1, P_2 - P_n$ and $w_1, w_2 - w_n$ respectively. 3M

Now, a thief want to rob the store for that, he wants an empty box of size 'M'. Now this problem is, in what way he has to place items in the box, such that he should get maximum profit. This is called as "knapsack problem".

Merging and purging rules of 0/1 knapsack

Let us consider an item with weight w_i and capacity of box be 'M'.

1) if $w_i < M$ then we are going to merge that item into the box. 3M
2) if $w_i > M$ (weight of item greater than knapsack capacity (M)) then we are going to purge that item from box.

UNIT - III.

6. a) Explain briefly about Breadth first search and write the pseudocode for BFS.

1) Breadth first search is one of the Graph tree traversal.

2) The purpose of Graph tree traversal is to visit each and every node from the given Graph.

3) In BFs, we start traversing from source vertex 'v' and make it as 'T' in visited table.

4) Now find all adjacent vertices of 'v' and check whether they are unvisited or not, if they are unvisited then they are placed in queue while removing source 'v' & make all ~~neighbouring~~ adjacent vertices as 'T' in visited table.

5) Repeat this procedure until all vertices are visited.

6) Finally the queue is empty.

Pseudocode for BFs:

Algorithm BFS (s)

Input: 's' is source vertex

Output: Mark all vertices that can be visited from 's'.

```
1. for each vertex V
2. do flag[v] = false
3.  Q = empty queue;
4. flag[s] = true;
5. enqueu (Q,s);
```

e 6. while Q is not empty

7. do v := dequeue (Q)

8. for each w adjacent to v.

9. do if flag[w] : false

10. then flag [w]: true

11. enqueue (Q, w);

4M

b) Write a short notes on strongly connected components. 4M.

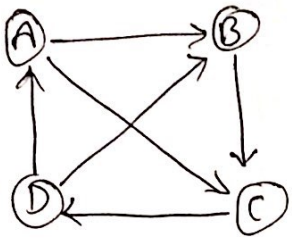
→ Strongly connected components:

Strongly connected components of a directed graph is a maximum set of vertices in which there is a path from one vertex to another vertex. 1M

Step1 : Test whether the given graph is strongly connected or not.

1. Perform a DFS on a directed graph G, starting of any arbitary vertex s
2. If there is any vertex of G that is not verified by DFS then the graph is not strongly connected.
3. If the first DFS (G) visits each vertex of (G) then reverse all edges of 'G' & Perform another DFS starting at 's' is the reverse graph.
4. If every vertex (G) visited by the second DFS then the graph is strongly connected.

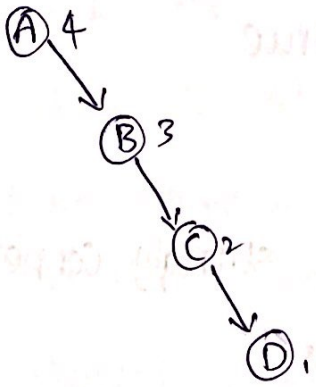
Eg:-



Source vertex : A.

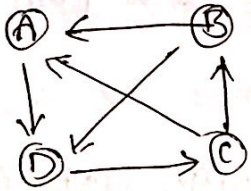
Sol:-

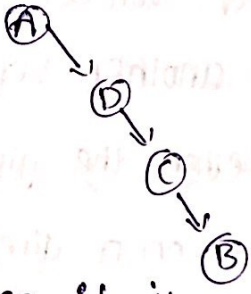
DFS (G):



Recursive of G:

DFS (R(G))





Satisfied hence it is strongly connected.

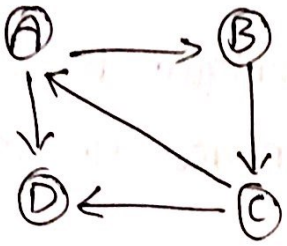
Step2: Determine Strongly connected components

1. Perform the DFS and no the vertices in order to completion of reverse graph.
2. Construct a reverse graph.
3. Perform the DFS and the reverse graph starting with the highest number of vertex according to no. assigned to step(1)
4. If DFS of reverse graph doesn't reach the all vertex starting the all DFS from the highest number

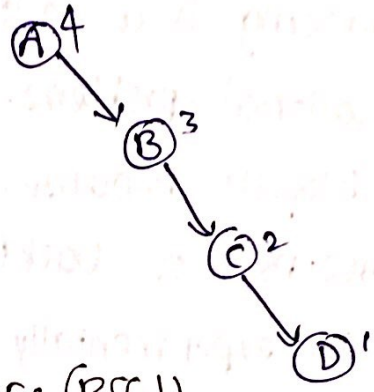
remaining vertices.

5. Each tree resulting forest is strongly connected component of G.

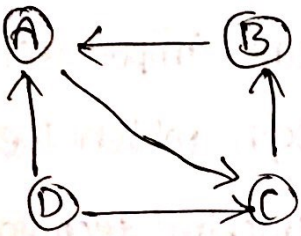
Eg:



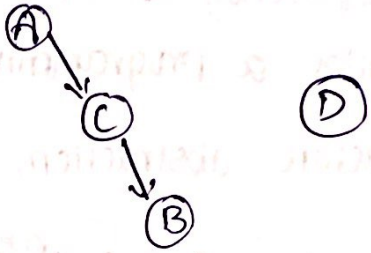
DFS of G



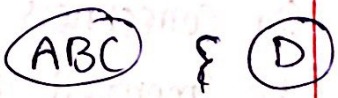
R(G)



DFS (R(G))



∴ The no. of connected components are



3m

Steps:

Condensation Graph

Strongly connected component of a directed graph that can be collapsed into a single vertex building of a new graph such that there is no cycle for above graph.

7. a) Write the control abstraction of backtracking and write Algorithm backtracking for n-queen problem.

→ Control abstraction of backtracking

1. Backtracking is a well-known technique for solving combinational problems. It is of interest to programming methodologists because.

(1) correctness of backtracking programs may be difficult to ascertain experimentally and

(2) efficiency is often of paramount importance. This Paper applies a programming methodology, which we call control structure abstraction, to the backtracking technique.

(3) In the context of efficiency, it provides sufficient conditions for correctness of an initial program which may subsequently be improved for efficiency while preserving correctness.

Algorithm for n-queens:

```
Algorithm place (k, I)
{ // return tree if a queen can be placed in kth row &
  Ith column. otherwise it returns false
    for j=1 to k-1 do
        if ((x[j] = I))
            or (abs (x[j] - I) = abs (j - k))
        then return false;
    return true;
}
```

(or)

```
Algorithm Nqueen (k, n)
{
    for i=1 to n do
    {
        if place (k, i) then
        {
            x[k] = i;
            if (k=n) then write (x[1:n]);
            else nqueen (k+1, n);
        }
    }
}
```

4M

b) Define the following i) Problem state ii) Answer state iii) State space tree.

→ (i) Problem State:

Each node in a depth First Search (DFS) Tree (OR)

(OR)

Each node in a state space tree

1M

(ii) Answer State:

Answer states are those solution states s for which the Path from root node to s defines a tuple that is a member of the set of solutions. These states satisfy implicit constraints. State space tree is the tree organization of the solution space.

2M

iii) State space tree:

A possible tree organization for the case n=4, A tree is called Permutation tree or State space tree. 4m

UNIT - IV

8 a) Explain the following

i) Control Abstraction for LC-Search

ii) FIFO branch and Bound

iii) LC Branch and Bound

i) Control Abstraction for LC-search:

```
Algorithm LCsearch (t)
{
    if *t is an answer node then output *t and return;
    E := t; // E-node
    Initialize the list of live nodes to be empty;
    repeat
    {
        for each child x of E do
        {
            if x is an answer node then output
                the path from x to t and return;
            Add (x); // x is a new live node
            (x → parent) := E; // pointer for path to root.
        }
        if there are no more live nodes then
        {
            write ("NO answer node"); return;
        }
        E := Least();
    }until(false);
}
```

4m

1) FIFO branch and Bound :

FIFO branch and Bound algorithm for the 0-1 knapsack problem can begin with upper = upperbound value of root.

If lowerbound value > upper then the node becomes kill node and we have to update upper.

updated upper = min (previous upper, next upperbound value, nextnext upperbound value)

lower bound $\hat{c}(x) = P_1x_1 + P_2x_2 + P_3x_3 + P_4x_4$ (follow Greedy)

upper bound $\hat{u}(x) = P_1x_1 + P_2x_2 + P_3x_3 + P_4x_4$ (follow dynamic)

Ex:- No. of items = 4

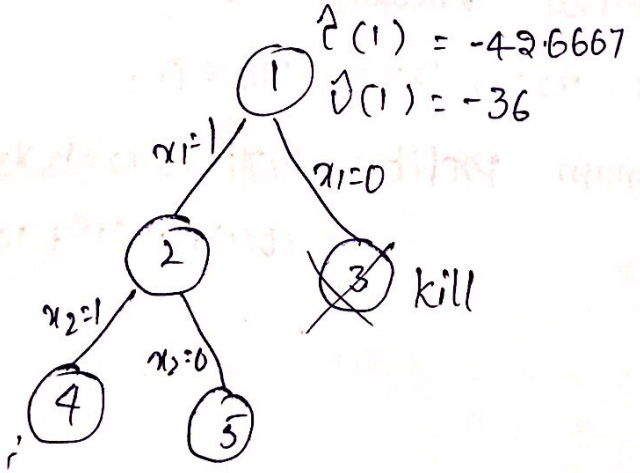
$(P_1, P_2, P_3, P_4) = (10, 12, 14, 10)$

$(w_1, w_2, w_3, w_4) = (2, 4, 8, 9)$, $M = 20$.

Convert maximising problem into minimising Problem

$(P_1, P_2, P_3, P_4) = (-10, -12, -14, -10)$

$(w_1, w_2, w_3, w_4) = (2, 4, 8, 9)$, $M = 20$.



4 Marks

$\hat{c}(1) = P_1x_1 + P_2x_2 + P_3x_3 + P_4x_4$

$x_1 = 1 \quad M = 18$

$x_2 = 1 \quad M = 14$

$x_3 = 1 \quad M = 6$

$x_4 = 6/9 \quad M = 0$

$\hat{c}(1) = -10 - 12 - 14 - 10\left(\frac{6}{9}\right)$

$= -42.6667.$

$\hat{u}(1) = P_1x_1 + P_2x_2 + P_3x_3 + P_4x_4$

$x_1 = 1$

$x_2 = 1$

$x_3 = 1$

$x_4 = 0$

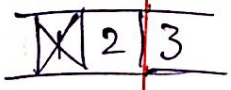
$\hat{u}(1) = -36.$

upper $= -36.$

lowerbound > upper

$-42.666 > -36$ (false)

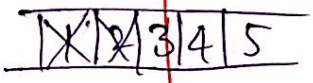
∴ live node becomes E-node.



Similarly

$\hat{c}(2) = -42.667 \qquad \hat{u}(2) = -36.$

$\hat{c}(2) >$ upper (false)



$\hat{c}(3) = -34.889 \qquad \hat{u}(3) = -26$

$\hat{c}(3) >$ upper

$-34.88 > -36$ ✓ TRUE.

Kill node 3.

follow Same procedure,

∴ Final Answer is

$x_1 = 1, \; x_2 = 1, \; x_3 = 1, \; x_4 = 0.$

Maximum profit $= P_1x_1 + P_2x_2 + P_3x_3 + P_4x_4$

$= 10(1) + 12(1) + 14(1) + 10(0)$

$= \underline{36}.$

(iii) LC Branch and bound :

LC Branch and bound for 0-1 knapsack problem.

It is necessary to conceive state space tree.

Given profits, weights & no.of items and capacity. for Minimising problems we use Branch and bound.

So convert maximising problem into minimising problem

Ex:- $n=4$, $(P_1, P_2, P_3, P_4) = (10, 10, 12, 14)$

$(w_1, w_2, w_3, w_4) = (2, 4, 6, 9)$, $M = 15$.

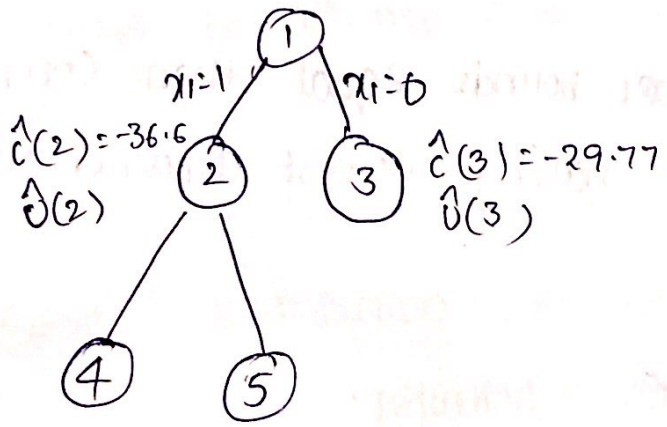
Maximising converted to minimising problem

$(P_1, P_2, P_3, P_4) = (-10, -10, -12, -14)$

$(w_1, w_2, w_3, w_4) = (2, 4, 6, 9)$.

$M = 15$.

4M



you have to find both $\hat{c}$ & $\hat{u}$ for each node.

if $\hat{c}(2) < \hat{c}(3)$ then expand '2'. follow same until you get all x_i values.

$\hat{c}(2)$ =	$\hat{u}(2)$
$x_1 = 1$	$x_1 = 1$
$x_2 = 1$	$x_2 = 1$
$x_3 = 1$	$x_3 = 1$
$x_4 = 3/9$	$x_4 = 0$

$\hat{c}(2) = P_1x_1 + P_2x_2 + P_3x_3 + P_4x_4 = -36.6$. $\hat{u}(2) = -32$.

Similarly find $\hat{c}(3) = -29.77$ $\hat{u}(3) = -22$.

$\hat{c}(2) < \hat{c}(3)$

∴ expand node 2.

if $\hat{c}(2) = \hat{c}(3)$ & $\hat{u}(2) = \hat{u}(3)$ then expand both nodes.

if lower bounds equal then compare upperbound expand tree having lowest upper bound value.

) Explain the method of reduction to solve TSP problem using Branch and Bound

Travelling Salesman problem:

Input: Complete directed weighted Graph

Output: Start traversing from one vertex, and visit all vertices exactly once and reach starting vertex with minimum cost.

1) It is solved by using state space tree.

2) Apply row reduction and column reduction to given matrix.

$\hat{c}$ (root) = sum of row reduction + Sum of column reduction

3) $\hat{c}$ (child) = $\hat{c}$ (root) + q (edge$_{ij}$, edge$_{ij}$) + P.

To find P:

i) place infinities in ith row and jth column of Parent matrix.

2) place q[j,s] = ∞ in parent matrix.

3) Then apply row reduction and column reduction

P = Sum of row reduction + sum of column reduction

4) Then expand child with minimum $\hat{c}(x)$ value.

5) Find out the path from root to leaf node.

6) Finally that is the path of travelling with minimum cost.

8m

b) Write a short notes on Complexity measures.

→ Complexity of an algorithm is a measure of amount of time / space required by an algorithm for an input of given size 'n'.

2. What do we measure?

In Analysing an algorithm, rather than a piece of code, we will try to predict the no. of times "the principle activity" Sorting algorithm we might count the number of comparisions Performed, and if it is an algorithm to find some optimal solution. If it is a graph coloring algorithm we might count the no. of times we check that a colored node is compatible with its neighbours.

4M