Rajan Gautam

19BCP101

Div. II, CE 19

SOT, PDEU

Pandit Deendayal Energy University

School of Technology

Design & Analysis of Algorithm (20CP209P)

B. Tech - Computer Science & Engineering (Sem-IV)

Rajan Gautam

Table of Contents

Rajan Gautam

Lab 7 Assignment: Solving Optimization Problems through Branch and Bound technique and draw state-space diagram.

AIM: To Solve Graph & Optimization Problems through Branch & Bound technique and draw state-space diagram. Analyze the complexity for both the problems.

Problem 1: For the given two arrays for set of items, Weight matrix W= {12,2,1,1,4} and value matrix V= {4,2,1,2,10} and knapsack capacity M=15, print the maximum value of Knapsack and corresponding weight in capacity. Print all the items that thief will take away. Use LC-Branch and Bound.

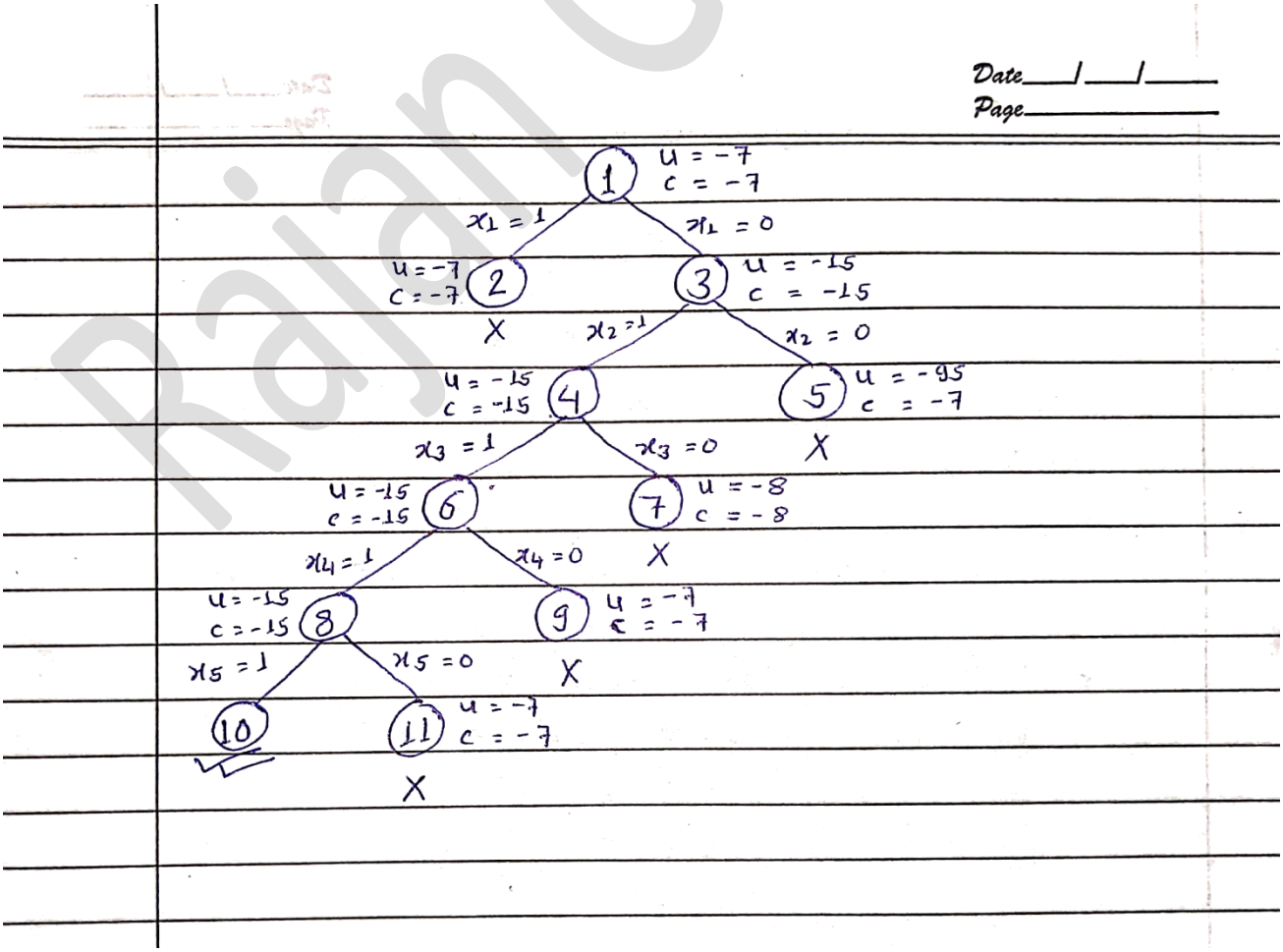
SOLUTION:

We have given

V = {4,2,1,2,10}

W = {12,2,1,1,4}

M = 15



Rajan Gautam

Hence, Required Answer → {0, 1, 1, 1, 1}

Maximum Profit → 15

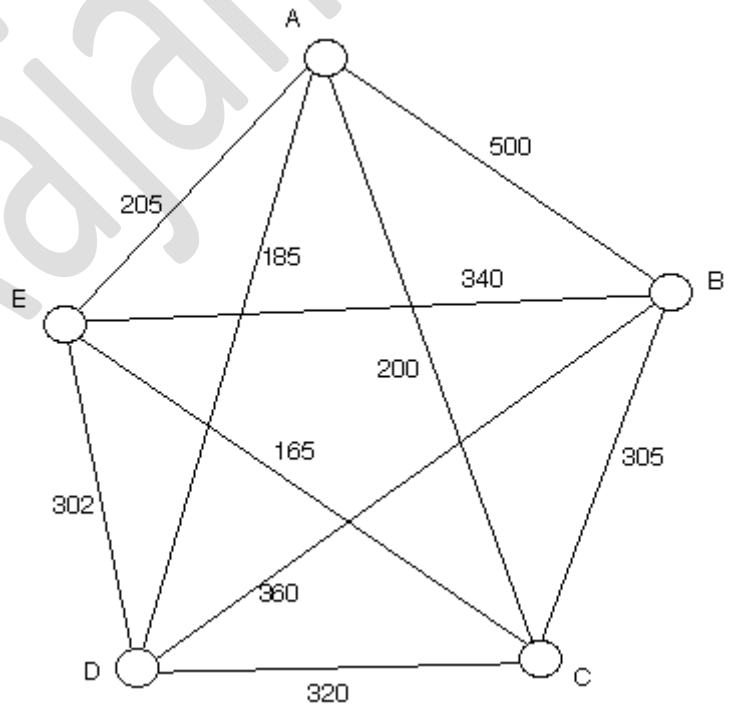
Weight of Items taken → 2, 1, 1, 4

Maximum Weight → 8

TIME COMPLEXITY:

The time complexity for solving Knapsack problem using branch and bound is O(2n) because while creating the combinational tree there will be n possibility to either take it or not take it. So, by permutation and combination we can say that in worse case the number of nodes that the algorithm will explore is 2n.

Problem-2: Write a C/C++ program to solve Travelling Salesman problem for given graph using Branch and Bound technique using FIFO method. What is the shortest possible route that the salesman must follow to complete his tour?



Rajan Gautam

CODE:

```
#include <stdio.h>
#include <stdbool.h>
#include <limits.h>
#include <string.h>
#define N 5

int final_path[N + 1];
bool visited[N];
int final_res = INT_MAX;

void copyToFinal(int curr_path[])
{
    for (int i = 0; i < N; i++)
    {
        final_path[i] = curr_path[i];
    }
    final_path[N] = curr_path[0];
}

int firstMin(int adj[N][N], int i)
{
    int min = INT_MAX;
    for (int k = 0; k < N; k++)
    {
        if (adj[i][k] < min && i != k)
        {
            min = adj[i][k];
        }
    }
    return min;
}
int secondMin(int adj[N][N], int i)
{
    int first = INT_MAX, second = INT_MAX;
    for (int j = 0; j < N; j++)
    {
        if (i == j)
        {
            continue;
        }

        if (adj[i][j] <= first)
```

Rajan Gautam

```
43.         {
44.             second = first;
45.             first = adj[i][j];
46.         }
47.         else if (adj[i][j] <= second && adj[i][j] != first)
48.         {
49.
50.             second = adj[i][j];
51.         }
52.     }
53.     return second;
54. }
55. void TSPRec(int adj[N][N], int curr_bound, int curr_weight, int level, int curr_path[])
56. {
57.     if (level == N)
58.     {
59.         if (adj[curr_path[level - 1]][curr_path[0]] != 0)
60.         {
61.             int curr_res = curr_weight + adj[curr_path[level - 1]][curr_path[0]];
62.             if (curr_res < final_res)
63.             {
64.                 copyToFinal(curr_path);
65.                 final_res = curr_res;
66.             }
67.         }
68.         return;
69.     }
70.
71.     for (int i = 0; i < N; i++)
72.     {
73.         if (adj[curr_path[level - 1]][i] != 0 && visited[i] == false)
74.         {
75.             int temp = curr_bound;
76.             curr_weight += adj[curr_path[level - 1]][i];
77.             if (level == 1)
78.             {
79.                 curr_bound -
    = ((firstMin(adj, curr_path[level - 1]) + firstMin(adj, i)) / 2);
80.             }
81.             else
82.             {
83.                 curr_bound -
    = ((secondMin(adj, curr_path[level - 1]) + firstMin(adj, i)) / 2);
84.             }
```

Rajan Gautam

```
            if (curr_bound + curr_weight < final_res)
            {
                curr_path[level] = i;
                visited[i] = true;
                TSPRec(adj, curr_bound, curr_weight, level + 1, curr_path);
            }
            curr_weight -= adj[curr_path[level - 1]][i];
            curr_bound = temp;
            memset(visited, false, sizeof(visited));
            for (int j = 0; j <= level - 1; j++)
            {
                visited[curr_path[j]] = true;
            }
        }
    }
}

void TSP(int adj[N][N])
{
    int curr_path[N + 1];
    int curr_bound = 0;
    memset(curr_path, -1, sizeof(curr_path));
    memset(visited, 0, sizeof(curr_path));
    for (int i = 0; i < N; i++)
    {
        curr_bound += (firstMin(adj, i) + secondMin(adj, i));
    }
    curr_bound = (curr_bound & 1) ? curr_bound / 2 + 1 : curr_bound / 2;
    visited[0] = true;
    curr_path[0] = 0;
    TSPRec(adj, curr_bound, 0, 1, curr_path);
}
char ans[5] = {'A', 'B', 'C', 'D', 'E'};
int main()
{
    int adj[N][N] = {{0, 500, 200, 185, 205},
                     {500, 0, 305, 360, 340},
                     {200, 305, 0, 320, 165},
                     {185, 360, 320, 0, 302},
                     {205, 340, 165, 302, 0}};

    TSP(adj);
    printf("Minimum Cost: %d\n", final_res);
    printf("Path Taken: \n");
```

Rajan Gautam

```
129.      for (int i = 0; i <= N; i++)
130.      {
131.          printf("%c \t", ans[final_path[i]]);
132.      }
133.
134.      return 0;
135.  }
136.
```

OUTPUT:

```
PROBLEMS   OUTPUT   TERMINAL   ···                1: Code

PS F:\Programs\C Programming\Projects\Design and Analysis of Algorithm> cd "f:\Programs\
C Programming\Projects\Design and Analysis of Algorithm\Branch and Bound\" ; if ($?) { g
cc travellingsalesman.c -o travellingsalesman } ; if ($?) { .\travellingsalesman }
Minimum Cost: 1220
Path Taken:
A       D       B       C       E       A
PS F:\Programs\C Programming\Projects\Design and Analysis of Algorithm\Branch and Bound>
```

Minimum Cost → **1220**

Path Taken → **A D B C E A**

TIME COMPLEXITY:

The worst-case time complexity will be O(n!) as we will have to see all the permutation of all the n cities and so all the nodes in the tree will be considered.

Link: https://github.com/rgautam320/Design-and-Analysis-of-Algorithm-Lab/tree/master/Lab_7_Branch-Bound

Rajan Gautam