Experiment 10

Aim:Program to implement 0/1 Knapsack Problem using Branch & Bound Method.

1.1 CO Attained: CO2, CO3 and CO5

1.2 Objective:

We are a given a set of n objects which have each have a value vi and a weight wi. The objective of the 0/1 Knapsack problem is to find a subset of objects such that the total value is maximized, and the sum of weights of the objects does not exceed a given threshold W.

1.3 Resources: Turbo c/Dev C++

1.4 Program Logic:

The branch and bound algorithm is similar to backtracking but is used for optimization problems. It performs a graph transversal on the space-state tree, but general searches BFS instead of DFS. During the search bounds for the objective function on the partial solution are determined. At each level the best bound is explored first, the technique is called best bound first. If a complete solution is found then that value of the objective function can be used to prune partial solutions that exceed the bounds.

1.Sort all items in decreasing order of V/W so that upper bound can be computed using Greedy Approach.(The nodes taken in the image are accordingly.)
2.Initialize profit, max = 0
3. Create an empty queue, Q.
4.Create a dummy node of decision tree and enqueue it to Q. Profit and weight of dummy node are 0.
5.Do while (Q is not empty).
6.Extract an item from Q. Let the item be x.
7.Compute profit of next level node. If the profit is more than max, then update max. (Profit from root to this node (include this node)).
8.Compute bound of next level node. If bound is more than max, then add next level node to Q.(Upper Bound of the maximum Profit in subtree of this node)
9.Consider the case when next level node is not considered as part of solution and add a node to queue with level as next, but weight and profit without considering next level nodes.

10.5 Procedure:

1. Create: Open Dev C++/C and write a program after that save the program with the .c extension.

2. Compile: Alt + F9
3. Execute: Ctrl + F10

10.6Program Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef enum{ NO, YES } BOOL;
int N;
int vals[100];
int wts[100];
int cap = 0;
int mval = 0;
void getWeightAndValue (BOOL incl[N], int *weight, int *value) {
int i, w = 0, v = 0;
for (i = 0; i< N; ++i) {
if (incl[i]) {
w += wts[i];
v += vals[i];
}
}
*weight = w;
*value = v;
}
void printSubset (BOOL incl[N]) {
int i;
int val = 0;
printf("Included = { ");
for (i = 0; i< N; ++i) {
if (incl[i]) {
printf("%d ", wts[i]);
```

```
val += vals[i];
}
}
printf("}; Total value = %d\n", val);
}
void findKnapsack (BOOL incl[N], int i) {
int cwt, cval;
getWeightAndValue(incl, &cwt, &cval);
if (cwt <= cap) {
if (cval>mval) {
printSubset(incl);
mval = cval;
}
}
if (i == N || cwt >= cap) {
return;
}
int x = wts[i];
BOOL use[N], nouse[N];
memcpy(use, incl, sizeof(use));
memcpy(nouse, incl, sizeof(nouse));
use[i] = YES;
nouse[i] = NO;
findKnapsack(use, i+1);
findKnapsack(nouse, i+1);
}
int main(int argc, char const * argv[]) {
printf("Enter the number of elements: ");
scanf(" %d", &N);
BOOL incl[N];
```

```
int i;
for (i = 0; i < N; ++i) {
printf("Enter weight and value for element %d: ", i+1);
scanf(" %d %d", &wts[i], &vals[i]);
incl[i] = NO;
}
printf("Enter knapsack capacity: ");
scanf(" %d", &cap);
findKnapsack(incl, 0);
return 0;
}
```

10.7Conclusion:

```
Enter the number of elements: 5
Enter weight and value for element 1: 23 6
Enter weight and value for element 2: 21 8
Enter weight and value for element 3: 45 9
Enter weight and value for element 4: 22 18
Enter weight and value for element 5: 67 12
Enter knapsack capacity: 50
Included = { 23 }; Total value = 6
Included = { 23 21 }; Total value = 14
Included = { 23 22 }; Total value = 24
Included = { 21 22 }; Total value = 26
```

10.8Analysis:

Its worst case time complexity is still given by $O(2^n)$, in cases where the entire tree has to be explored. However, in its best case, only one path through the tree will have to explored, and hence its best case time complexity is given by O(n).

10.9Lab Viva Questions:

1. What is the most efficient approach to solve this problem?
2. Define branch and bound.
3. Define 0/1 knapsack problem.
4. Differentiate 0/1 Knapsack and Fractional Knapsack.