Name	Harsh Chandra
UID no.	2021700013
Experiment No.	8

AIM:	**Branch and bound (To implement 0/1 Knapsack problem using Branch and Bound.)**
Program 1	
ALGORITHM/ THEORY:	1. Sort all items in decreasing order of ratio of value per unit weight so that an upper bound can be computed using Greedy Approach. 2. Initialize maximum profit, maxProfit = 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 following while Q is not empty. • Extract an item from Q. Let the extracted item be u. • Compute profit of next level node. If the profit is more than maxProfit, then update maxProfit. • Compute bound of next level node. If bound is more than maxProfit, then add next level node to Q. • 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.

PROGRAM:

```
#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("};\nTotal 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() {
        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;
}
```

RESULT:

```
Enter the number of elements: 7
Enter weight and value for element 1: 2 10
Enter weight and value for element 2: 3 5
Enter weight and value for element 3: 5 15
Enter weight and value for element 4: 7 7
Enter weight and value for element 5: 1 6
Enter weight and value for element 6: 4 18
Enter weight and value for element 7: 1 3
Enter knapsack capacity: 15
Included = { 2 };
Total value = 10
Included = { 2 3 };
Total value = 15
Included = { 2 3 5 };
Total value = 30
Included = { 2 3 5 1 };
Total value = 36
Included = { 2 3 5 1 4 };
Total value = 54


...Program finished with exit code 0
Press ENTER to exit console.
```

CONCLUSION:	From this experiment, I understood difference between knapsack and 0/1 knapsack problem and how to apply branch and bound algorithm for solving o/1 knapsack problem.