Name	Stephen David Vaz
UID no.	2021700070
Experiment No.	8

AIM:	Branch and bound
PROBLEM STATEMENT :	To implement 0/1 Knapsack problem using Branch and Bound
ALGORITHM/ THEORY:	The 0/1 Knapsack problem is a classic optimization problem in computer science that involves selecting a subset of items from a given set of items with a specified weight and value. The objective is to maximize the value of the selected items while keeping their total weight below a given weight limit. The problem is called 0/1 Knapsack because each item can only be selected once (0/1 choice) and cannot be partially selected. The time complexity of this dynamic programming algorithm is O(nW), where n is the number of items and W is the maximum weight capacity of the knapsack. Another way to solve the 0/1 Knapsack problem is by using the branch and bound algorithm. This algorithm involves exploring the search space of possible solutions and eliminating branches that cannot lead to an optimal solution. The steps for implementing the branch and bound algorithm are: Create a priority queue to store the partial solutions to the problem. Add the empty solution to the priority queue. While the priority queue is not empty: Dequeue the solution with the highest priority (the highest expected value). If the solution is complete (all items have been selected or rejected), update the maximum value of the knapsack if the current solution is better. Otherwise, create two child solutions by adding the next item to the knapsack or rejecting it. Compute the expected value of each child solution and add it to the priority queue. Prune any branches that cannot lead to an optimal solution. The branch and bound algorithm can be more efficient than the dynamic programming algorithm in some cases because it can eliminate branches of

	the search tree that cannot lead to the optimal solution. However, the worst-case time complexity of the branch and bound algorithm is still exponential in the number of items.
PROGRAM:	

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int N;
int profit[100];
int weights[100];
int selG[100];
int cap = 0;
int max_val = 0;

void weightAndprofit (int selected[N], int *weight, int *value) {
   int i, w = 0, v = 0;
   for (i = 0; i < N; ++i) {
       if (selected[i]) {
           w += weights[i];
           v += profit[i];
       }
   }
   *weight = w;
   *value = v;
}

void printTable (int selected[N]) {
   int i;
   int val = 0;
   printf("\nSelected Items:-\n");
   printf("Item No. Weight Profit\n");
   for (i = 0; i < N; ++i) {
       if (selected[i]) {
           printf("   %d\t   %d\t  %d\n", i+1,weights[i],
profit[i]);
           val += profit[i];
       }
   }
   printf("Total Profit: \t  %d\n", val);
}
```

```
void knapsack (int selected[N], int i) {
   int curr_w, curr_val;
   weightAndprofit(selected, &curr_w, &curr_val);
   if (curr_w <= cap) {
       if (curr_val > max_val) {
           memcpy(selG, selected, sizeof(selG));
           max_val = curr_val;
       }
   }
   if (i == N || curr_w >= cap) {
       return;
   }
   int x = weights[i];
   int use[N], no_use[N];
   memcpy(use, selected, sizeof(use));
   memcpy(no_use, selected, sizeof(no_use));
   use[i] = 1;
   no_use[i] = 0;
   knapsack(use, i+1);
   knapsack(no_use, i+1);
}

int main() {
   printf("Items: ");
   scanf(" %d", &N);
   int selected[N];
   int i;
   printf("Weights: ");
   for(i = 0; i < N; i++) {
       scanf(" %d", &weights[i]);
       selected[i] = 0;
   }
   printf("Profit: ");
   for(i = 0; i < N; i++) {
       scanf(" %d", &profit[i]);
   }
   printf("Knapsack capacity: ");
   scanf(" %d", &cap);
   knapsack(selected, 0);
   printTable(selG);
```

```
    return 0;
}
```

RESULT:

```
 *  Executing task: /usr/bin/clang /Users/stephen03/Dev/repos/stepDAA/e8/bb
cs/bb2 && ../excs/bb2

Items: 4
Weights: 2 4 6 9
Profit: 10 10 12 18
Knapsack capacity: 15

Selected Items:-
Item No. Weight Profit
   1       2      10
   2       4      10
   4       9      18
Total Profit:     38
 *  Terminal will be reused by tasks, press any key to close it.
```

CONCLUSION:	Successfully implemented and understood 0/1 Knapsack in C. Also understood various methods to solve the Knapsack problem.