# Bangalore Institute of Technology

**Department of Computer Science and Engineering**

**DESIGN AND ANALYSIS OF ALGORITHMS LAB (BCSL404)**

**Program 6**

Design and implement C Program to solve 0/1 Knapsack problem using Dynamic Programming method.

**Aim:** To implement 0/1 Knapsack problem using Dynamic programming

**Definition:** using **Dynamic programming**

It gives us a way to design custom algorithms which systematically search all possibilities (thus guaranteeing correctness) while storing results to avoid recomputing (thus providing efficiency).

We are given a set of *n* items from which we are to select some number of items to be carried in a knapsack(BAG). Each item has both a *weight* and a *profit*. The objective is to choose the set of items that fits in the knapsack and maximizes the profit.

Given a knapsack with maximum capacity *W*, and a set *S* consisting of *n* items , Each item *i* has some weight *wi* and benefit value *bi* (all *wi , bi* and *W* are integer values)

Problem: How to pack the knapsack to achieve maximum total value of packed items?

**ALGORITHM**

//(n items, W weight of sack) Input: n, wi,,, vi and W – all integers

//Output: V(n,W)

// Initialization of first column and first row elements

Repeat for i = 0 to n

set V(i,0) = 0

Repeat for j = 0 to W

Set V(0,j) = 0

//complete remaining entries row by row

Repeat for i = 1 to n

repeat for j = 1 to W

if ( wi <= j ) V(i,j)) = max{ V(i-1,j), V(i-1,j-wi) + vi }

if ( wi > j ) V(i,j) = V(i-1,j)

Print V(n,W)

**PROGRAM:**

**#include<stdio.h>**

**int w[10],p[10],n;**

**int max(int a,int b)**

**{**

**return a>b?a:b;**

**}**

**int knap(int i,int m)**

**{**

**if(i==n) return w[i]>m?0:p[i];**

**if(w[i]>m) return knap(i+1,m);**

**return max(knap(i+1,m),knap(i+1,m-w[i])+p[i]);**

**}**

**int main()**

**{**

**int m,i,max\_profit;**

**printf("\nEnter the no. of objects:");**

**scanf("%d",&n);**

**printf("\nEnter the knapsack capacity:");**

**scanf("%d",&m);**

**printf("\nEnter profit followed by weight:\n");**

**for(i=1;i<=n;i++)**

**scanf("%d %d",&p[i],&w[i]);**

**max\_profit=knap(1,m);**

**printf("\nMax profit=%d",max\_profit);**

**return 0;**

**}**

**Input/Output:**

Enter the no. of objects:4

Enter the knapsack capacity:6

Enter profit followed by weight:

78 2

45 3

92 4

71 5

Max profit=170

**Program 7** :

Design and implement C Program to solve discrete Knapsack and continuous Knapsack problems using greedy approximation method.

This program first calculates the profit-to-weight ratio for each item, then sorts the items based on this ratio in non-increasing order. It then fills the knapsack greedily by selecting items with the highest ratio until the knapsack is full. If there's space left in the knapsack after selecting whole items, it adds fractional parts of the next item. Finally, it prints the optimal solution and the solution vector.

Here's a simplified version of the C program to solve discrete Knapsack and continuous Knapsack problems using the greedy approximation method:

**#include<stdio.h>**

**void knapsack(int n, float weight[], float profit[], float capacity)**

**{**

**float x[20], tp = 0;**

**int i, j, u;**

**u = capacity;**

**for (i = 0; i < n; i++)**

**x[i] = 0.0;**

**for (i = 0; i < n; i++) {**

**if (weight[i] > u)**

**break;**

**else {**

**x[i] = 1.0;**

**tp = tp + profit[i];**

**u = u - weight[i];**

**}**

**}**

**if (i < n)**

**x[i] = u / weight[i];**

**tp = tp + (x[i] \* profit[i]);**

**printf("\nThe result vector is:- ");**

**for (i = 0; i < n; i++)**

**printf("%f\t", x[i]);**

**printf("\nMaximum profit is:- %f", tp);**

**}**

**int main()**

**{**

**float weight[20], profit[20], capacity;**

**int num, i, j;**

**float ratio[20], temp;**

**printf("\nEnter the no. of objects:- ");**

**scanf("%d", &num);**

**printf("\nEnter the wts and profits of each object:- ");**

**for (i = 0; i < num; i++) {**

**scanf("%f %f", &weight[i], &profit[i]);**

**}**

**printf("\nEnter the capacity of knapsack:- ");**

**scanf("%f", &capacity);**

**for (i = 0; i < num; i++) {**

**ratio[i] = profit[i] / weight[i];**

**}**

**for (i = 0; i < num; i++) {**

**for (j = i + 1; j < num; j++) {**

**if (ratio[i] < ratio[j]) {**

**temp = ratio[j];**

**ratio[j] = ratio[i];**

**ratio[i] = temp;**

**temp = weight[j];**

**weight[j] = weight[i];**

**weight[i] = temp;**

**temp = profit[j];**

**profit[j] = profit[i];**

**profit[i] = temp;**

**}**

**}**

**}**

**knapsack(num, weight, profit, capacity);**

**return(0);**

**}**

**Input/Output:**

**1) Enter the no. of objects:- 4**

**Enter the wts and profits of each object:- 56 23 78 45 98 76 78 78**

**Enter the capacity of knapsack:- 100**

**The result vector is:-: 1.00000 0.000000 0.000000 0.000000**

**Maximum profit is:- 78.0**

**2) Enter the no. of objects:- 3**

**Enter the wts and profits of each object:- 20 30 25 40 10 35**

**Enter the capacity of knapsack:- 40**

**The result vector is:-: 1.00000 1.000000 0.25**

**Maximum profit is:- 82.5**