**CHANDIGARH UNIVERSITY**

**UNIVERSITY INSTITUTE OF ENGINEERING**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**



|  |  |
| --- | --- |
| **Submitted By:** Sahil Kaundal  **Submitted To:** Neha Dutta | |
| **Subject Name** | Design and Analysis of Algorithm Lab |
| **Subject Code** | 20CSP-312 |
| **Branch** | Computer Science Engineering |
| **Semester** | 5th |

**Experiment 7**

**Student Name:** Sahil Kaundal **UID:** 21BCS8197

**Branch:** BE CSE (Lateral Entry) **Section/Group:** 616/A

**Semester:** 5th **Date of Performance:** 31/10/2022

**Subject Name:** DAA Lab **Subject Code:** 21-CSP-312

# Aim/Overview of the practical:

Code to implement 0-1 Knapsack using Dynamic Programming

# Task to be done/ Which logistics used:

Write a program to implement 0-1 Knapsack using Dynamic Programming

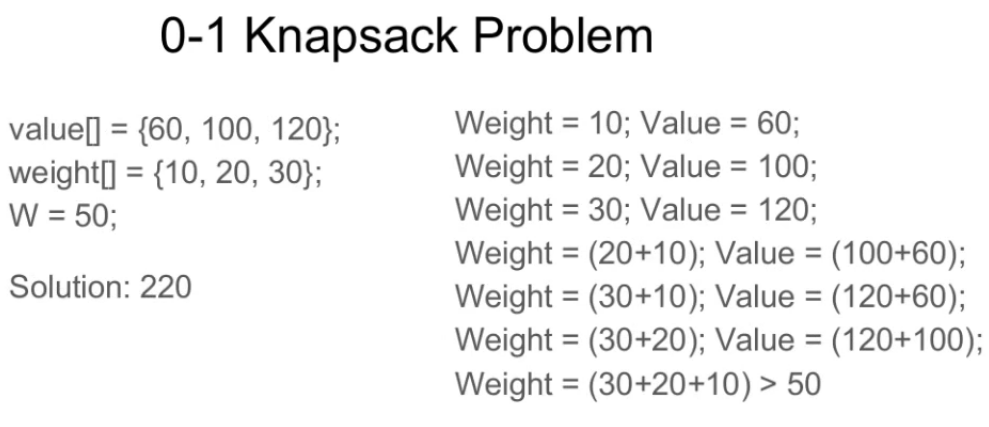
# Requirements:

Laptop or PC.

Operation system (Mac, Windows, Linux, or any)

Vs-Code with MinGw or any C++ Compiler

1. **Algorithm/Flowchart (For programming-based labs):**



* **Begin**
* **Input set of items each with a weight and a value**
* **Set knapsack capacity**
* **Create a function that returns maximum of two integers.**
* **Create a function which returns the maximum value that can be put in a knapsack of** **capacity W**

int knapSack(int W, int w[], int v[], int n)

int i, wt;

int K[n + 1][W + 1]

for i = 0 to n

for wt = 0 to W

if (i == 0 or wt == 0)

Do K[i][wt] = 0

else if (w[i - 1] <= wt)

Compute: K[i][wt] = max(v[i - 1] + K[i - 1][wt - w[i - 1]], K[i -1][wt])

else

K[i][wt] = K[i - 1][wt]

return K[n][W]

Call the function and print.

End

# Steps for experiment/practical/Code:

#include<bits/stdc++.h>

using namespace std;

int knapsack\_dp(int n, int M, int w[], int p[])

{

int i,j;

//create a matrix to memoize the values using dynamic programming

int knapsack[n+1][M+1];

//knapsack[i][j] denotes the maximum attainable value of items in knpasack with i available

//items and capacity of knapsack being j

//initializing knapsack[0][j]=0 for 0<=j<=M

//because if there is no item, no value can be attained

for(j=0;j<=M;j++)

knapsack[0][j]=0;

//initializing knapsack[i][0]=0 for 0<=i<=n,

//because in a bag of zero capacity, no item can be placed

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

knapsack[i][0]=0;

//now, filling the matrix in bottom up manner

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

{

for(j=1;j<=M;j++)

{

//check if the weight of current item i is less than or equal to the capacity of sack,

//take maximum of once including the current item and once not including

if(w[i-1]<=j)

{

knapsack[i][j]=max(knapsack[i-1][j],p[i-1]+knapsack[i-1][j-w[i-1]]);

}

//can not include the current item in this case

else

{

knapsack[i][j]=knapsack[i-1][j];

}

}

}

return knapsack[n][M];

}

int main()

{

int i,j;

int n; //number of items

int M; //capacity of knapsack

cout<<"Enter the no. of items: ";

cin>>n;

int w[n]; //weight of items

int p[n]; //value of items

cout<<"Enter the weight and price of all items: "<<endl;

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

{

cin>>w[i]>>p[i];

}

cout<<"Enter the capacity of Knapsack: ";

cin>>M;

int result=knapsack\_dp(n,M,w,p);

//the maximum value will be given by knasack[n][M], ie. using n items with capacity M

cout<<"The maximum value of items that can be put into knapsack is: "<<result;

return 0;

}

# Result/Output/Writing Summary:

The time complexity of this solution is O(n\*M).

# 

**Learning outcomes (What I have learnt):**

* To implement the concept of Knapsack Problem.
* To implement the concept of Knapsack Problem using Dynamic Programming.

**Evaluation Grid (To be created as per the SOP and Assessment guidelines by the faculty):**

|  |  |  |  |
| --- | --- | --- | --- |
| Sr. No. | Parameters | Marks Obtained | Maximum Marks |
| 1. |  |  |  |
| 2. |  |  |  |
| 3. |  |  |  |