**LAB : 2**

**OBJECTIVE :**

1. To Implement 0/1 Knapsack and analyse complexity of algorithm.
2. To Implement solution of partitioning into two subsets so that subset sum difference is minimum (such that each element of given set is included in one of two subsets).
3. **To Implement 0/1 Knapsack and analyse complexity of algorithm.**

**Problem Statement :**

Given the weights and profits of ‘N’ items, we are asked to put these items in a knapsack that has a capacity ‘C’. The goal is to get the maximum profit from the items in the knapsack. Each item can only be selected once, as we don’t have multiple quantities of any item.

**Implementation : Bottom-Up DP**

dp[i][c] will represent the maximum knapsack profit for capacity ‘c’ calculated from the first ‘i’ items.

So, for each item at index ‘i’ (0 <= i < items.length) and capacity ‘c’ (0 <= c <= capacity), we have two options:

1. Exclude the item at index ‘i’. In this case, we will take whatever profit we get from the sub-array excluding this item => dp[i-1][c]
2. Include the item at index ‘i’ if its weight is not more than the capacity. In this case, we include its profit plus whatever profit we get from the remaining capacity and from remaining items => profits[i] + dp[i-1][c-weights[i]]

Finally, our optimal solution will be maximum of the above two values:

    dp[i][c] = max (dp[i-1][c], profits[i] + dp[i-1][c-weights[i]])

**Space and Time Complexity of solutuon is O( capacity \* Number of items)**

**Code :**

#include <iostream>

#include <vector>

using namespace std;

void INPUT()

{

#ifndef ONLINE\_JUDGE

freopen("C:/Users/arvind/Desktop/Current/input.txt", "r", stdin);

freopen("C:/Users/arvind/Desktop/Current/output.txt", "w", stdout);

#endif

}

void printSelectedElements(vector<vector<int>> &dp, const vector<int> &weights,

const vector<int> &profits, int capacity) {

//dp index 1,2 .... mapped to wt index 0,1,2

cout << "Selected weights:";

int totalProfit = dp[weights.size()][capacity];

for (int i = weights.size(); i > 0; i--) {

if (totalProfit != dp[i - 1][capacity]) {

cout << " " << weights[i - 1];

capacity -= weights[i - 1];

totalProfit -= profits[i - 1];

}

}

cout << endl;

}

int solveKnapsack(const vector<int> &profits, const vector<int> &weights, int capacity) {

// basic checks

if (capacity <= 0 || profits.empty() || weights.size() != profits.size()) {

return 0;

}

int n = profits.size();

vector<vector<int>> dp(n + 1, vector<int>(capacity + 1, 0));

//first column and row = 0 , (capacity =0 and no items resp.)

//dp index 1,2 .... mapped to wt index 0,1,2

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

for (int c = 1; c <= capacity; c++) {

int profit1 = 0, profit2 = 0;

// include the item, if it is not more than the capacity

if (weights[i - 1] <= c) {

profit1 = profits[i - 1] + dp[i - 1][c - weights[i - 1]];

}

// exclude the item

profit2 = dp[i - 1][c];

// take maximum

dp[i][c] = max(profit1, profit2);

}

}

printSelectedElements(dp, weights, profits, capacity);

// maximum profit will be at the bottom-right corner.

return dp[n][capacity];

}

int main(int argc, char \*argv[]) {

INPUT();

int n, maxCapacity;

cin >> n;

vector<int> profits(n);

vector<int> weights(n);

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

cin >> profits[i];

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

cin >> weights[i];

cin >> maxCapacity;

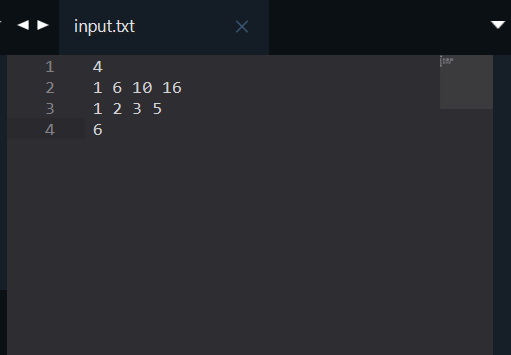
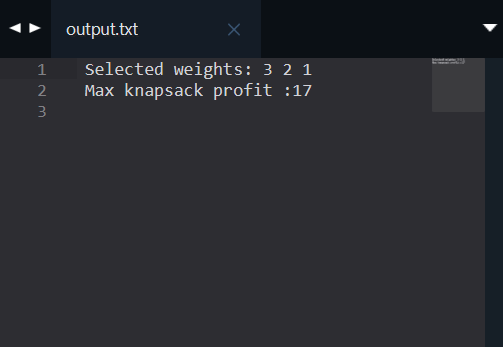
int maxProfit = solveKnapsack(profits, weights, maxCapacity);

cout << "Max knapsack profit :" << maxProfit << endl;

return 0;

}

**Output**

1. **To Implement solution of partitioning into two subsets so that subset sum difference is minimum**

**Problem Statement :**

Given a set of integers, the task is to divide it into two sets S1 and S2 such that the absolute difference between their sums is minimum.   
If there is a set S with n elements, then if we assume Subset1 has m elements, Subset2 must have n-m elements and the value of absolute value of (sum(Subset1) – sum(Subset2)) should be minimum.

**Implementation : Bottom-Up DP**

This problem is variation of 0/1 Knapsach problem.

We are needed to create two Subsets S1 and S2 such that their absolute sum difference is minimum. Suppose S1 with sum either less than or equal to subset sum S2 .

We can reframe the statement to minimize absolute sum difference of subset S1  and (TotalSum – sum of subset S1 ). We can observe that problem is similar to equal subset sum partition (where difference will be 0). To met the reqiurement of minimum sum difference, we try to get the sum of subset S1 as close as possible to TotalSum/2 (the sum of subset S1 lies in range 0 and TotalSum/2 both inclusive, as S1 is smaller sum subset). Here TotalSum is sum of all the given elements.

We can check for all the sums in range [0, TotalSum/2], that can we form a subset (subset S1 ) from given original numbers. Then we can choose subset S1 with sum as close as posiible to TotalSum/2 and all elements not in S1 wll form subset S2 .

We can apply bottom-up dynamic programming to solve problem. Here, dp[i][j] will hold a boolean value, where true represent whether we can form a subset sum ‘j’ with first ‘i’ elements. The elements of dp matrix is initialized to false and first column elements to true.

So, for each item at index ‘i’ (0 <= i <= number of elements) and capacity ‘j’ (0 <= j <= Totalum/2).

**Space and Time Complexity of solutuon is O( Number of elements\* TotalSum/2)**

**Code :**

#include <iostream>

#include <vector>

#include<climits>

using namespace std;

void INPUT()

{

#ifndef ONLINE\_JUDGE

freopen("C:/Users/arvind/Desktop/Current/input.txt", "r", stdin);

freopen("C:/Users/arvind/Desktop/Current/output.txt", "w", stdout);

#endif

}

void printSubsets(const vector<vector<bool>> &dp, const vector<int> &numbers, int firstSum) {

int n = numbers.size();

vector<bool> SubsetIndex(n, false);

//true in first subset else n second (probale greater sum subset)

int numberIndex = n;

while (firstSum and numberIndex > 0) {

if (dp[numberIndex][firstSum]) {

numberIndex--;

if (dp[numberIndex ][firstSum])

continue;

SubsetIndex[numberIndex] = true;

firstSum = firstSum - numbers[numberIndex] + 1;

}

firstSum--;

}

cout << "\nFirst Subset :";

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

{

if (SubsetIndex[i])

cout << " " << numbers[i];

}

cout << "\nSecond Subset :";

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

{

if (!SubsetIndex[i])

cout << " " << numbers[i];

}

cout << endl;

}

int findMin(const vector<int> &numbers, int n) {

int sum = 0, total = 0;

for (int i : numbers)

total += i;

sum = total / 2;

//Subset Sum Problem

vector<vector<bool>> dp(n + 1, vector<bool>(sum + 1, false));

//rows - num

//col = sum [0-tatal/2]

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

dp[i][0] = true;

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

for (int j = 1; j <= sum; ++j)

{

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

if (j - numbers[i - 1] >= 0)

dp[i][j] = dp[i][j] || dp[i - 1][j - numbers[i - 1]];

}

}

int diff = INT\_MAX;

int firstSum = total, secondSum = 0;

for (int i = sum; i >= 0 ; --i)

{

if (dp[n][i]) {

firstSum = i;

secondSum = total - firstSum;

diff = secondSum - firstSum;

break;

}

}

printSubsets(dp, numbers, firstSum);

return diff;

}

int main(){

INPUT();

int n;

cin >> n;

vector<int> input(n);

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

cin >> input[i];

int ans = findMin(input, n);

cout << "The minimum difference = " << ans;

return 0;

}

**Output**

