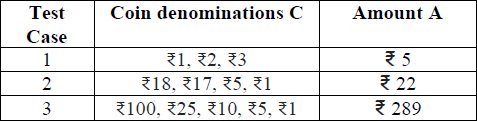
**PRACTICAL - 5**

**5.1 AIM: : A cashier at any mall needs to give change of an amount to customers many times in a day. Cashier has multiple number of coins available with different denominations which is described by a set C. Implement the PROGRAM CODE for a cashier to find the minimum number of coins required to find a change of a particular amount A. Output should be the total number of coins required of given denominations.**

**Check the PROGRAM CODE for following test cases:**



**PROGRAM CODE**:

#include <bits/stdc++.h>

using namespace std;

void minFinder(int sum, int length, int coins[])

{

int ans[sum], j = 0, i = 0;

for (i = length - 1; i >= 0; i--)

{

while (sum >= coins[i])

{

sum = sum - coins[i];

ans[j] = coins[i];

j++;

}

}

cout << "\nTotal coins required are : " << j << "\nThey are : ";

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

cout << ans[i] << " ";

cout << "\n";

}

int main()

{

int length, i, sum;

cout << "\nEnter the total types of coins : ";

cin >> length;

int coins[length];

cout << "Enter the denomination of coins : ";

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

cin >> coins[i];

int n = sizeof(coins) / sizeof(coins[0]);

sort(coins, coins + n);

cout << "Enter the final sum of coins : ";

cin >> sum;

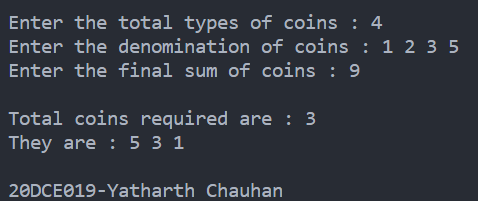
minFinder(sum, length, coins);

cout << "\n20DCE019-Yatharth Chauhan";

return 0;

}

**OUTPUT**:



**CONCLUSION:** This problem is a variation of 'Coin Change Problem'. Worst case: When the only coin present is Rs. 1 Coin. So, in that case, time complexity becomes O(A) where A is the amount to be paid.

**5.2 AIM: Implement the PROGRAM CODE 4.2 using Dynamic PROGRAM CODEming. Compare Greedy and Dynamic approach.**

**PROGRAM CODE**:

#include <iostream>

using namespace std;

int maximum(int x, int y)

{

if (x > y)

return x;

else

return y;

}

int knapsack(int bag\_capacity, int weight[], int profit[], int number)

{

int matrix[number + 1][bag\_capacity + 1];

for (int i = 0; i < number + 1; i++)

for (int j = 0; j < bag\_capacity + 1; j++)

{

if (i == 0 || j == 0)

matrix[i][j] = 0;

else if (j >= weight[i - 1])

matrix[i][j] = maximum(matrix[i - 1][j], profit[i - 1] + matrix[i - 1][j - weight[i - 1]]);

else

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

}

return matrix[number][bag\_capacity];

}

int main()

{

int number, bag\_capacity;

cout << ".....BINARY KNAPSACK PROBLEM.....";

cout << "\nEnter the size of arrays : ";

cin >> number;

int weight[number], profit[number];

cout << "\nEnter the weights :";

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

cin >> weight[i];

cout << "Enter the profits :";

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

cin >> profit[i];

cout << "Enter bag capacity : ";

cin >> bag\_capacity;

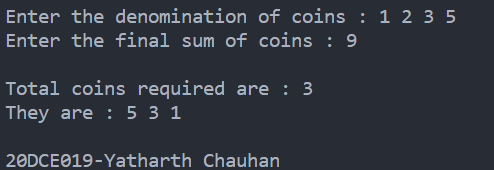
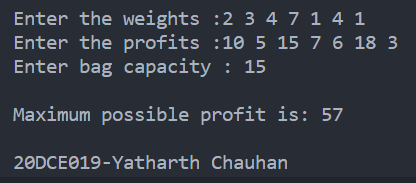
cout << "\nMaximum possible profit is: " << knapsack(bag\_capacity, weight, profit, number) << endl;

cout << "\n20DCE019-Yatharth Chauhan";

return 0;

}

**OUTPUT**:

**5.4 AIM: Given a chain < A1, A2,...,An> of n matrices, where for i=1,2,...,n matrix Ai with dimensions. Implement the PROGRAM CODE to fully parenthesize the product A1,A2,...,An in a way that minimizes the number of scalar multiplications. Also calculate the number of scalar multiplications for all possible combinations of matrices.**

|  |  |  |
| --- | --- | --- |
| **Test Case** | **n** | **Matrices with dimensions** |
| 1 | 3 | A1:3\*5,A2:5\*6,A3:6\*4 |
| 2 | 6 | A1: 30\*35, A2: 35\*15, A3: 15\*5, A4: 5\*10, A5: 10\*20, A6: 20\*25 |

**PROGRAM CODE**:

**OUTPUT**:

**CONCLUSION:**

**5.4 AIM: Program to implement all pairs shortest path.**

**PROGRAM CODE**:

**OUTPUT**:

**CONCLUSION:**