

12. Euclid Problem

CO4

From Euclid, it is known that for any positive integers A and B there exist such integers X and Y that $AX + BY = D$, where D is the greatest common divisor of A and B. The problem is to find the corresponding X, Y, and D for a given A and B.

Input The input will consist of a set of lines with the integer numbers A and B, separated with space ($A, B < 1,000,000,001$).

Output For each input line the output line should consist of three integers X, Y, and D, separated with space. If there are several such X and Y, you should output that pair for which $X \leq Y$ and $|X| + |Y|$ is minimal.

Sample Input 1

4 6

17 17

Sample Output 1

-1 1 2

0 1 17

Description:

The problem is to find the value of D and X, Y in the given equation $AX + BY = D$, where the input is given in the form, of A and B.

This can be done using the Euclid algorithm. The Euclid algorithm can be used to find the common divisor of A and B. Now the extended Euclid algorithm is used to find the coefficients X and Y.

Basic Euclidean Algorithm for GCD:

The algorithm is based on the below facts.

- If we subtract a smaller number from a larger one (we reduce a larger number), GCD does not change. So, if we keep subtracting repeatedly the larger of two, we end up with GCD.
- Now instead of subtraction, if we divide the smaller number, the algorithm stops when we find the remainder 0.

Extended Euclidean Algorithm working:

Let us assume we found the coefficients (x_1, y_1) for $(b, a \bmod b)$:

$$b \cdot x_1 + (a \bmod b) \cdot y_1 = g$$

and we want to find the pair (x, y) for (a, b) :

$$a \cdot x + b \cdot y = g$$

We can represent $a \bmod b$ as:

$$a \bmod b = a - \left\lfloor \frac{a}{b} \right\rfloor \cdot b$$

Substituting this expression in the coefficient equation of (x_1, y_1) gives:

$$g = b \cdot x_1 + (a \bmod b) \cdot y_1 = b \cdot x_1 + \left(a - \left\lfloor \frac{a}{b} \right\rfloor \cdot b \right) \cdot y_1$$

and after rearranging the terms:

$$g = a \cdot y_1 + b \cdot \left(x_1 - y_1 \cdot \left\lfloor \frac{a}{b} \right\rfloor \right)$$

We found the values of x and y :

$$\begin{cases} x = y_1 \\ y = x_1 - y_1 \cdot \left\lfloor \frac{a}{b} \right\rfloor \end{cases}$$

Algorithm:

- Take the input values of the equation A and B until EOF is terminated.
- Pass the A and B to a function extendedEuclid() to evaluate the common divisor D and coefficients X and Y
- **extendedEuclid()**
- Calculate the common divisor by performing repeated modulus operations until B=0
- Return the initial values X and Y as 1 and 0 respectively.
- Take the variables X1 and Y1 to get the updated values of X and Y
- Update X1 as y and Y1 as X-(A/B)*Y up till the recursion values retraces back

- Return the updated X, Y by modifying with obtained X1 and Y1. Also print the common divisor D

Code:

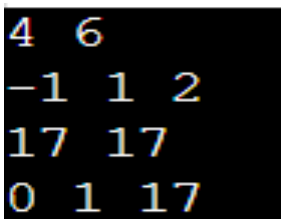
```
#include<stdio.h>

int x,y,d;

void extendedEuclid(int a, int b) {
    if (b == 0) { x = 1; y = 0; d = a; return; }
    extendedEuclid(b, a % b);
    int x1 = y;
    int y1 = x - (a / b) * y;
    x = x1;
    y = y1;
}

int main() {
    int a,b;
    while(scanf("%d %d",&a,&b) != EOF) {
        extendedEuclid(a,b);
        printf("%d %d %d\n",x,y,d);
    }
}
```

Output:



```
4 6
-1 1 2
17 17
0 1 17
```

13. Coin Problem

CO4

Given a value V . You have to make change for V cents, given that you have infinite supply of each of $C\{C_1, C_2, \dots, C_m\}$ valued coins. Find the minimum number of coins to make the change and print the coins that appear in an optimal solution.

Input:

The first line of input contains an integer T denoting the number of test cases. The first line of each test case is V and N , V is the value of cents and N is the number of coins. The second line of each test case contains N input $C[i]$, value of available coins.

Output:

Print the coins appear in an optimal solution and in a newline print the minimum number of coins to make the change and, if not possible print "-1".

Constraints: $1 \leq T \leq 100$

$$1 \leq V \leq 10^6$$

$$1 \leq N \leq 10^6$$

$$1 \leq C[i] \leq 10^6$$

Sample Input 1:

```
1
7 2
2 1
```

Sample Output 1:

```
2 2 2 1
4
```

Explanation:

Testcase 1: We can use coin with value 2 three times, and coin with value 1 one times to change a total of 7

Description:

The problem is to find the total no. of coins required to sum up to the amount specified and also the denominations of the coins used are also given out as an output. This problem can be solved using different approaches like Greedy and Dynamic

programming. The main disadvantage of the problems method is it fails to return the optimal solution in all cases. Thus we opt for Dynamic programming approach to solve the problem which returns the optimal value of the minimum no. of coins required.

Algorithm:

- Input the required data amount and the no. of coins
- Consider a list to store type of coins to be used and sort it in the descending order
- Pass the parameters amount and the list of coins into a function make_change()
- Now check if the coin value is less than the amount value
- If yes, append the coin in another list change and deduct the coin value from the amount.
- Proceed until amount = 0
- Return the list change and print the list length to get the total no. of coins.

Code: (Using greedy approach)

```
def make_change(denomination_list, amount):
```

```
    change = []
```

```
    for coin in denomination_list:
```

```
        while amount:
```

```
            if coin <= amount:
```

```
                change.append(coin)
```

```
                amount -= coin
```

```
            else:
```

```
                break
```

```
    return change
```

```
n=int(input())
```

```
while(n>0):
```

```
    l=list(map(int,input().split()))
```

```
    amount=l[0]
```

```
    coins=l[1]
```

```
n=n-1
denomination_list = [ ]
denomination_list=list(map(int,input().split()))
denomination_list.sort(reverse = True)
p=make_change(denomination_list, amount)
print(p)
print(len(p))
```

Output:

```
1
7 2
2 1
[2, 2, 2, 1]
4
```

Algorithm: (Dynamic programming)

- Input the required data amount and the no.of coins
- Consider a 1d array to store the coin types
- Pass the required parameters of amount, no.of coins and coin types into a function minCoins()
- Consider a 1d array dp and Initialize all the elements of the array by iterating through the amount as INT_MAX
- Set the base condition dp[0]=0
- Iterating in the outer loop for possible values of sum between 1 to N
- Since our final solution for sum = N might depend upon any of these values
- Inner loop denotes the index of coin array.
- For each value of sum, to obtain the optimal solution.
- After the final iteration return the last value of the dp array.

Code:

```
#include <bits/stdc++.h>
using namespace std;
```

```
int dp [1000] = {0};
int minCoins(int N, int M,int coins[])
{
    for(int i = 0;i<=N;i++)
        dp[i] = INT_MAX;
    dp[0] = 0;
    for(int i = 1;i<=N;i++)
    {
        for(int j = 0;j<M;j++)
        {
            if(coins[j] <= i)
            {
                dp[i] = min(dp[i], 1 + dp[i - coins[j]]);
            }
        }
    }
    return dp[N];
}

int main() {
    int n;
    cin>>n;
    do{
        int sum ,total_coins;
        cin>>sum>>total_coins;
        int coins[total_coins];
        for(int i=0;i<total_coins;i++)
```

```
cin>>coins[i];  
cout << minCoins(sum, total_coins,coins);  
  
n--;  
}while(n!=0);  
}
```

Output:

```
2  
7 2  
2 1  
minimum coins:4  
6 3  
4 3 1  
minimum coins:2
```

```
1  
10 4  
1 2 4 5  
minimum coins:2
```


14. Spirally traversing a matrix

CO4

Given a matrix of size $r \times c$. Traverse the matrix in spiral form.

Test case 1:

Input: $r = 4, c = 4$

matrix[][] = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}, {13, 14, 15, 16}}

Output: 1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10

Test case 2:

Input: $r = 3, c = 4$

matrix[][] = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}}

Output: 1 2 3 4 8 12 11 10 9 5 6 7

Description:

The problem is about spirally traversing a matrix clockwise

In order to print a matrix in spiral pattern (clockwise) form, we need to follow the following traversal order:

Left to right (first row)

Top to bottom (Last column)

Right to left (last row)

Bottom to top (First column)

In order to print the spiral pattern (anticlockwise), reverse the above traversal order.

Algorithm:

- Consider the rows and columns of the matrix
- Input the elements of the matrix which has to be spirally traversed
- Consider a variable t to store the total no. of elements
- Initialize $ct=0$
- Now until $ct < t$ follow the following steps
- Print the top row, i.e. Print the elements of k th row from column index l to n , and increase the count of k .
- Print the right column, i.e. Print the last column or $n-1$ th column from row index k to m and decrease the count of n .
- Print the bottom row, i.e. if $k > m$, then print the elements of $m-1$ th row from column $n-1$ to l and decrease the count of m
- Print the left column, i.e. if $l < n$, then print the elements of l th column from $m-1$ th row to k and increase the count of l .

- Decrease the value of c in each iteration.
- Thus the obtained elements are the result after traversing the matrix spirally.

Code:

```
import java.io.*;
import java.util.Scanner;
public class Main
{
    public static void main(String[] args) {
        Scanner s=new Scanner(System.in);
        int m,n;
        System.out.println("enter no.of rows and no.of columns ");
        m=s.nextInt();
        n=s.nextInt();
        int[][] a=new int[m][n];
        System.out.println("enter matrix");
        for(int i=0;i<m;i++)
        {
            for(int j=0;j<n;j++)
            {
                a[i][j]=s.nextInt();
            }
        }

        int min_r=0,min_c=0;
        int max_r=a.length-1,max_c=a[0].length-1;
        int t=m*n;
        int ct=0;
        while(ct<t)
        {
            for(int i=min_r,j=min_c;j<=max_c && ct<t;j++)
            {
                System.out.print(a[i][j]+" ");
                ct++;
            }
            min_r++;
            for(int i=min_r,j=max_c;i<=max_r && ct<t;i++)
            {
```

```
System.out.print(a[i][j]+" ");
ct++;
}
max_c--;
for(int i=max_r,j=max_c;j>=min_c && ct<t;j--)
{
    System.out.print(a[i][j]+" ");
    ct++;
}
max_r--;

for(int i=max_r, j=min_c;i>=min_r && ct<t;i--)
{
    System.out.print(a[i][j]+" ");
    ct++;
}
min_c++;
}
}
}
```

Output:

```
enter no.of rows and no.of columns
4 4
enter matrix
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10
```