L12 : Dynamic Programming-2

1-Tut : LCS - Problem

Send Feedback

Given two strings S1 and S2 with lengths M and N respectively, find the length of the longest common subsequence.

A subsequence of a string S whose length is K, is a string containing characters in same relative order as they are present in S, but not necessarily contiguous. Subsequences contain all the strings of length varying from 0 to K. For example, subsequences of string "abc" are -- ""(empty string), a, b, c, ab, bc, ac, abc.

Input Format :

First line will contain T(number of test case), each test case will consist of two lines.
Line 1: String S1
Line 2: String s2

Output Format :

Length of the longest common subsequence for each test case in a newline.

Constraints :

1 <= T <= 100
1 <= M <= 100
1 <= N <= 100
Time Limit: 1 sec

Sample Input 1:

1
adebc
dcadb

Sample Output 1: 3

Explanation of Sample Input 1:

"a", "d", "b", "c", "ad", "ab", "db", "dc" and "adb" are present as a subsequence in both the strings in which "adb" has the maximum length. There are no other common subsequence of length greater than 3 and hence the answer.

Sample Input 2:

1
abcd
acbdef

Sample Output 2: 3

Explanation of Sample Input 2:

"a", "b", "c", "d", "ab", "ac", "ad", "bd", "cd", "abd" and "acd" are present as a subsequence in both the strings S1 and S2 in which "abd" and "acd" are of the maximum length. There are no other common subsequence of length greater than 3 and hence the answer.

```
#include<bits/stdc++.h>
using namespace std;
int lcs(string &str1, string &str2)
{
    int n = str1.size();
    int m = str2.size();
    int dp[n+1][m+1];
    for(int i=0;i<=n;i++)
    {
        for(int j=0;j<=m;j++)
        {
            dp[i][j] = 0;
        }
    }
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=m;j++)
        {
            if(str1[i-1] == str2[j-1])
            {
                dp[i][j] = 1+dp[i-1][j-1];
            }
            dp[i][j] = max(dp[i][j], max(dp[i-1][j], dp[i][j-1]));
        }
    }

    int ans = dp[n][m];
    return ans;
}
int main(){
    // write your code here
    int t;
    cin>>t;
    while(t--)
    {
        string str1, str2;
         cin >> str1 >> str2;
        cout << lcs(str1, str2)<<endl;
    }
    return 0;
}
```

2-Tut : Edit Distance - Problem

Send Feedback

Given two strings s and t of lengths m and n respectively, find the Edit Distance between the strings. Edit Distance of two strings is minimum number of steps required to make one string equal to other. In order to do so you can perform following three operations only :

1. Delete a character
2. Replace a character with another one
3. Insert a character

Note - Strings don't contain spaces

Input Format :

First line of input will contain T (number of test cases), each test case consists of two lines.
Line 1 : String s
Line 2 : String t

Output Format :

For each test case print the Edit Distance value in new line.

Constraints:

1 <= T <= 100
1<= m,n <= 100

Sample Input 1 :

```
1
abc
dc
```

Sample Output 1 :

```
2
```

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

int Edit_Distance(string &s1,string &s2){
    int n=s1.size();
    int m=s2.size();
    int dp[n+1][m+1];

    for(int i=0;i<=n;i++){
        for(int j=0;j<=m;j++){
            if(i==0){
                dp[i][j]=j;
            }
            else if(j==0){
                dp[i][j]=i;
            }
            else{
                if(s1[i-1]==s2[j-1]){
```

```
                dp[i][j]=dp[i-1][j-1];
            }
            else{
                int f1=1+dp[i-1][j-1];
                int f2=1+dp[i-1][j];
                int f3=1+dp[i][j-1];
                dp[i][j]=min(f1,f2);
                dp[i][j]=min(dp[i][j],f3);
            }
        }
    }
    return dp[n][m];
}

int main(){

    // write your code here
    int t;
    cin>>t;
    while(t--){
        string s1,s2;
        cin>>s1>>s2;
        cout<<Edit_Distance(s1,s2)<<endl;
    }
    return 0;
}
```

Solution :

```
/* Time complexity: O(N*M) Space complexity : O(N*M) where N and M are the size of input strings */
#include using namespace std; int helper(string &s1, string &s2, int dp[][102], int len1, int len2) { if (len1 == 0 || len2 == 0) { dp[len1][len2] = max(len1, len2); return dp[len1][len2]; } if (dp[len1][len2] != -1) { return dp[len1][len2];
```

3-Tut : Knapsnack - Problem

Send Feedback

A thief robbing a store and can carry a maximal weight of W into his knapsack. There are N items and ith item weigh wi and is of value vi. What is the maximum value V, that thief can take ?

Note: Space complexity should be O(W).

Input Format :

Line 1 : N i.e. number of items
Line 2 : N Integers i.e. weights of items separated by space
Line 3 : N Integers i.e. values of items separated by space
Line 4 : Integer W i.e. maximum weight thief can carry

Output Format :

Line 1 : Maximum value V

Constraints

1 <= N <= 10^4
1<= wi <= 100
1 <= vi <= 100
1 <= W <= 1000

Sample Input 1 :

```
4
1 2 4 5
5 4 8 6
5
```

Sample Output :

```
13
```

```cpp
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;

ll helper_(int* weights, int* values, int n, int maxWeight,ll **storage){
    if(n<0){
      return 0;
    }
    if(storage[n][maxWeight]!=-1){
      return storage[n][maxWeight];
    }
    if(weights[n]<=maxWeight){
      ll inc=values[n]+helper_(weights,values,n-1,maxWeight-weights[n],storage);
      // not including last num
      ll notinc=helper_(weights,values,n-1,maxWeight,storage);
      ll ans=max(inc,notinc);
      storage[n][maxWeight]=ans;
      return ans;
```

```
19.
20.     }else{
21.
22.       ll ans=helper_(weights,values,n-1,maxWeight,storage);
23.       storage[n][maxWeight]=ans;
24.       return ans;
25.
26.     }
27. }
28.
29. int knapsack(int* weights, int* values, int n, int maxWeight){
30.             ll **storage=new ll*[n+1];
31.             for(int i=0;i<=n;i++){
32.               storage[i]=new ll[maxWeight+1];
33.             }
34.             for(int i=0;i<=n;i++){
35.               for(int j=0;j<=maxWeight;j++){
36.                 storage[i][j]=-1;
37.               }
38.             }
39.     ll ans=helper_(weights,values,n-1,maxWeight,storage);
40.     ans=(int)ans;
41.     for(int i=0;i<=n;i++){
42.         delete storage[i];
43.     }
44.     delete storage;
45.     return ans;
46. }
47.
48. int main(){
49.     // write your code here
50.   int n;
51.   cin >> n;
52.   int* weights = new int[n];
53.   int* values = new int[n];
54.
55.   for(int i = 0; i < n; i++){
56.     cin >> weights[i];
57.   }
58.
59.   for(int i = 0; i < n; i++){
60.     cin >> values[i];
61.   }
62.
```

```
63.  int maxWeight;
64.  cin >> maxWeight;
65.
66.  cout << knapsack(weights, values, n, maxWeight);
67.    return 0;
68. }
```

4-Tut : PARTY - Problem

Send Feedback

You just received another bill which you cannot pay because you lack the money.

Unfortunately, this is not the first time to happen, and now you decide to investigate the cause of your constant monetary shortness. The reason is quite obvious: the lion's share of your money routinely disappears at the entrance of party localities.

You make up your mind to solve the problem where it arises, namely at the parties themselves. You introduce a limit for your party budget and try to have the most possible fun with regard to this limit.

You inquire beforehand about the entrance fee to each party and estimate how much fun you might have there. The list is readily compiled, but how do you actually pick the parties that give you the most fun and do not exceed your budget?

Write a program which finds this optimal set of parties that offer the most fun. Keep in mind that your budget need not necessarily be reached exactly. Achieve the highest possible fun level, and do not spend more money than is absolutely necessary.

Input Format:

First line of input will contain an integer N (number of parties).
Next line of input will contain N space-separated integers denoting the entry fee of Ith party.
Next line will contain N space-separated integers denoting the amount of fun Ith party provide.
Last line of input will contain an integer W party budget.

Output Format:

For each test case your program must output the sum of the entrance fees and the sum of all fun values of an optimal solution. Both numbers must be separated by a single space.
Note: In case of multiple cost provides the maximum fun output the minimum total cost.

Sample Input:

```
5
1 7 9 7 2
5 5 2 4 7
12
```

Sample Output:

```
10 17
```

```
1. #include<bits/stdc++.h>
2. #include<utility>
3. using namespace std;
4.
```

```
5.  pair<int, int> most_fun(int* enterance_fees, int* fun, int budget, int n, pair<int, int>**dp)
6.  {
7.          if (n == 0)
8.          {
9.                  pair<int, int> p;
10.                 p.first = 0;
11.                 p.second = 0;
12.                 return p;
13.         }
14.         if (dp[budget][n].first != -1 && dp[budget][n].second != -1)
15.         {
16.                 return dp[budget][n];
17.         }
18.         pair<int, int> ans;
19.         if (enterance_fees[0] <= budget)
20.         {
21.                 pair<int, int>option1 = most_fun(enterance_fees + 1, fun + 1, budget -
    enterance_fees[0], n - 1, dp);
22.                 option1.first += enterance_fees[0];
23.                 option1.second += fun[0];
24.                 pair<int, int>option2 = most_fun(enterance_fees + 1, fun + 1, budget, n -
    1, dp);
25.                 if (option1.second > option2.second)
26.                 {
27.                         ans= option1;
28.                 }
29.                 else if (option2.second > option1.second)
30.                 {
31.                         ans= option2;
32.                 }
33.                 else
34.                 {
35.                         if (option1.first < option2.first)
36.                         {
37.                                 ans= option1;
38.                         }
39.                         else
40.                         {
41.                                 ans= option2;
42.                         }
43.                 }
44.         }
45.         else
46.         {
```

```
                ans= most_fun(enterance_fees + 1, fun + 1, budget, n - 1, dp);
        }
        dp[budget][n] = ans;
        return ans;
}


int main(){

    // write your code here
    while (true)
        {
            int budget;
            int n;
            cin >> n;
            if (budget == 0 && n == 0)
            {
                break;
            }
            int* enterance_fees = new int[n];
            int* fun = new int[n];
            for (int i = 0; i < n; i++)
            {
                cin >> enterance_fees[i];
            }
        for(int i=0;i<n;i++){
       cin>> fun[i];
    }
            cin >> budget;

            pair<int, int>** dp = new pair<int, int> * [budget+1];
            for (int i = 0; i < budget+1; i++)
            {
                dp[i] = new pair<int, int>[n + 1];
                for (int j = 0; j < n + 1; j++)
                {
                    dp[i][j].first = -1;
                    dp[i][j].second = -1;
                }
            }


            pair<int, int> p;
            p = most_fun(enterance_fees, fun, budget, n, dp);
```

```
                cout << p.first << ' ' << p.second << endl;
        exit(0);



                for (int i = 0; i < budget + 1; i++)
                {
                        delete[]dp[i];
                }
                delete[]dp;
          }
        return 0;
    }
```

5-Tut : Subset Sum - Problem

Send Feedback

Given an array of n integers, find if a subset of sum k can be formed from the given set. Print Yes or No.

Input Format

First-line will contain T(number of test cases), each test case consists of three lines.
First-line contains a single integer N(length of input array).
Second-line contains n space-separated integers denoting the elements of array.
The last line contains a single positive integer k.

Output Format

Output Yes if there exists a subset whose sum is k, else output No for each test case in new line.

Constraints:

1 <= T <= 100
1 <= N <= 500
1 <= arr[i] <= 10^4
1 <= K <= 500
Sample Input

```
1
3
1 2 3
4
```

Sample Output

```
Yes
```

```
1. #include<bits/stdc++.h>
2. using namespace std;
3.
4. bool targetSumSubset(int n, vector<int> arr, int tar) {
5.     bool dp[n+1][tar+1]={true};
```

```
    for(int i=0;i<n+1;i++){
        for(int j=0;j<tar+1;j++){
            if(i==0 and j==0){
                dp[i][j]=true;
            }
            else if(i==0){
                dp[i][j]=false;
            }
            else if(j==0){
                dp[i][j]=true;
            }
            else{
                if(dp[i-1][j]==true){
                    dp[i][j]=true;
                }
                else{
                    int val=arr[i-1];
                    if(j>=val){
                        if(dp[i-1][j-val]==true){
                            dp[i][j]=true;
                        }
                    }
                }
            }
        }
    }
   return dp[arr.size()][tar];
}

int main(){

    // write your code here

    int t;
    cin>>t;
    while(t--)
    {
            int n;
      cin >> n;
      vector<int> arr(n);
      for (int i = 0; i < arr.size(); i++) {
        cin >> arr[i];
            }
            int tar;
```

```
        cin >> tar;
    bool a=  targetSumSubset(n, arr, tar);
    if(a==1)
    {
        cout<<"Yes"<<endl;
    }
    else{
        cout<<"No"<<endl;
                    }
    }
    return 0;
}
```

6-Tut : Maximum Sum Rectangle

Send Feedback

Given a 2D array, find the maximum sum rectangle in it. In other words find maximum sum over all rectangles in the matrix.

Input Format:

First line of input will contain T(number of test case), each test case follows as.
First line contains 2 numbers n and m denoting number of rows and number of columns. Next n lines contain m space separated integers denoting elements of matrix nxm.
Output Format:

Output a single integer, maximum sum rectangle for each test case in a newline.
Constraints

1 <= T <= 50
1<=n,m<=100
-10^5 <= mat[i][j] <= 10^5

Sample Input

```
1
4 5
1 2 -1 -4 -20
-8 -3 4 2 1
3 8 10 1 3
-4 -1 1 7 -6
```

Sample Output

```
29
```

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

int kadane(vector<int>v,int n){
    int ma=INT_MIN;
```

```
    int u=0;
    for(int i=0;i<n;i++){
        u+=v[i];
        if(u>ma){
            ma=u;
        }
        if(u<0){
            u=0;
        }
    }
    return ma;
}

int main(){

    // write your code here
    int t;
    cin>>t;
    while(t--){
        int n,m;
        cin>>n>>m;
        vector<vector<int>>mat(n,vector<int>(m));
                    for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                cin>>mat[i][j];
            }
            }


        int ma=INT_MIN;
        for(int i=0;i<n;i++){
            vector<int>ans(m);
            for(int j=i;j<n;j++){
                for(int col=0;col<m;col++) {
                    ans[col]+=mat[j][col];
                }
                ma=max(ma,kadane(ans,m));
            }
        }
        cout<<ma<<endl;

            }
    return 0;
}
```

7-Tut : Mehta and Bank Robbery - Problem

Send Feedback

One fine day, when everything was going good, Mehta was fired from his job and had to leave all the work. So, he decided to become a member of gangster squad and start his new career of robbing. Being a novice, mehta was asked to perform a robbery task in which he was given a bag having a capacity W units. So, when he reached the house to be robbed, there lay N items each having particular weight and particular profit associated with it. But, theres a twist associated, He has first 10 primes with him, which he can use atmost once, if he picks any item x, then he can multiply his profit[x] with any of the first 10 primes and then put that item into his bag. Each prime can only be used with one particular item and one item can only have atmost one prime multiplied with its profit. Its not necessary to pick all the items. If he doesnt want to use a prime with any particular item, he can simply add the profit as it is, more specifically, 1*profit[x] for xth item will get added to its total profit, and that he can do with as many items as he wants. He cannot fill his bag more than weight W units. Each item should be picked with its whole weight, i.e it cannot be broken into several other items of lesser weight. So, now to impress his squad, he wishes to maximize the total profit he can achieve by robbing this wealthy house.

Input Format:

The first line of input will contain T(number of test cases), each test will follow as.
First Line will contain two integers N and W (number of items and maximum weight respectively).
Second-line will contain N space-separated integers denoting the profit associated with the Ith item.
The third line will contain N space-separated integers denoting the weight of the Ith item.

Output Format:

Output the maximum profit obtainable for each test case in a new line.

Constraints:

1 <= T <= 20
1 <= N, W <= 500
1 <= profit[i] <= 10^4
1 <= weight[i] <= 10^4

Sample Input:

```
1
7 37
33 5 14 14 16 25 15
5 19 30 4 15 31 25
```

Sample output:

```
1591
```

```
1. #include<bits/stdc++.h>
2. using namespace std;
3.
4. int solve(vector<pair<int, int>> &arr, int n,int w)
5. {
6.     int dp[2][n+1][w+1];
```

```
7.      memset(dp, 0, sizeof(dp));
8.      sort(arr.begin(), arr.end());
9.
10.     int primes[11] = {1,2,3,5,7,11,13,17,19,23,29};
11.
12.     for(int i=1;i<=n;i++)
13.     {
14.         for(int j=1;j<=w;j++)
15.         {
16.             dp[0][i][j] = dp[0][i-1][j];
17.
18.             if(j>=arr[i-1].second)
19.             {
20.                 dp[0][i][j] = max(dp[0][i][j], dp[0][i-1][j-arr[i-1].second] + arr[i-1].first);
21.             }
22.         }
23.     }
24.
25.     for(int prime = 1;prime<=10;prime++)
26.     {
27.         int p = prime%2;
28.         for(int i=1;i<=n;i++)
29.         {
30.             for(int j=1;j<=w;j++)
31.             {
32.                 dp[p][i][j] = dp[p][i-1][j];
33.                 if(j>=arr[i-1].second)
34.                 {
35.                     int temp = max(dp[p][i-1][j-arr[i-1].second] + arr[i-1].first,
36.                                    dp[p^1][i-1][j-arr[i-1].second] + arr[i-1].first * primes[prime]);
37.
38.                     dp[p][i][j] = max(dp[p][i][j], temp);
39.                 }
40.             }
41.         }
42.     }
43.     return dp[0][n][w];
44. }
45.
46. int main(){
47.
48.     // write your code here
49.      int t;
50.     cin>>t;
```

```
    while(t--)
    {
        int n,w;
        cin>>n>>w;
        vector<pair<int, int>> arr(n);
        for(int i=0;i<n;i++)
        {
            cin>>arr[i].first;
        }
        for(int i=0;i<n;i++)
        {
            cin>>arr[i].second;
        }
        cout<<solve(arr, n, w)<<endl;
    }

    return 0;
}
```

8-Ass : Miser Man

Send Feedback

Jack is a wise and miser man. Always tries to save his money.

One day, he wants to go from city A to city B. Between A and B, there are N number of cities(including B and excluding A) and in each city there are M buses numbered from 1 to M. And the fare of each bus is different. Means for all N*M busses, fare (K) may be different or same. Now Jack has to go from city A to city B following these conditions:

1. At every city, he has to change the bus.

2. And he can switch to only those buses which have number either equal or 1 less or 1 greater to the previous.

You are to help Jack to go from A to B by spending the minimum amount of money.

Input Format:

First-line will contain T(number of the test case), each test case follows as.
First-line will contain two space-separated integers N and M.
Next, N rows will contain M space-separated integers denoting the elements of the grid.
Each row lists the fares the M busses to go form the current city to the next city.

Output Format:

For each test case print the minimum amount of fare that Jack has to give in a newline.

Constraints:

1 <= T <= 100
1 <= N, M <= 100
1 <= arr[i][j] <= 10^5

Sample Input

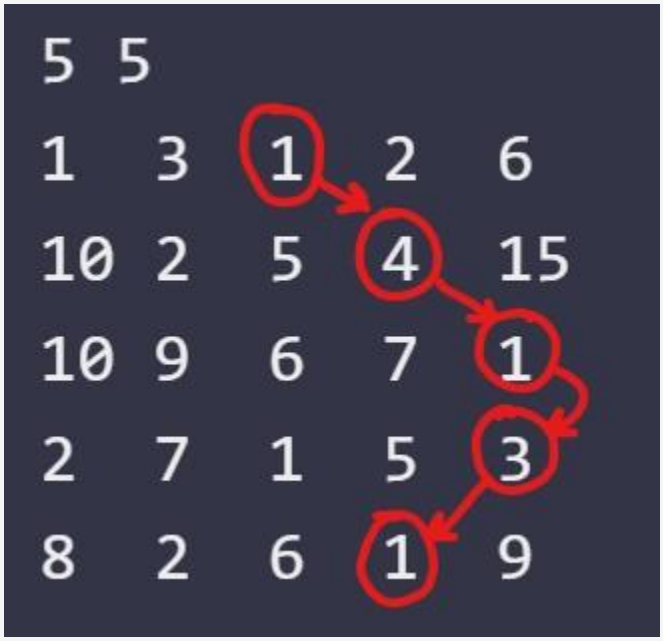
```
1
5 5
1 3 1 2 6
10 2 5 4 15
10 9 6 7 1
2 7 1 5 3
8 2 6 1 9
```

Sample Output

```
10
```

Explanation:

1 -> 4 -> 1 -> 3 -> 1: 10
This is marked and shown in the following image:




```
1.  #include<bits/stdc++.h>
2.  using namespace std;
3.
4.  int save[105][105];
5.
6.  int main(){
7.
8.      // write your code here
9.
10.     int t;
11.    cin>>t;
12.    while(t--){
```

```
13.         int n,m;
14.         cin>>n>>m;
15.         int i,j;
16.
17.         for(i=0;i<n;i++)
18.                 for(j=0;j<m;j++)
19.                         cin>>save[i][j];
20.         int res[n][m];
21.         for(i=0;i<m;i++)
22.                 res[0][i]=save[0][i];
23.         for(i=1;i<n;i++)
24.         {
25.                 for(j=0;j<m;j++)
26.                 {
27.                         int l,u,r;
28.                         l=(j>0)?(res[i-1][j-1]):INT_MAX;
29.
30.                         u=res[i-1][j];
31.                         r=(j<m-1)?res[i-1][j+1]:INT_MAX;
32.                         res[i][j]=save[i][j]+min(u,min(l,r));
33.                 }
34.         }
35.         int ans=INT_MAX;
36.         for(i=0;i<m;i++)
37.                 if(res[n-1][i]<ans)
38.                         ans=res[n-1][i];
39.         cout<<ans<<endl;
40.     }
41.
42.     return 0;
43. }
```

9-Ass : Trader Profit

Send Feedback

Mike is a stock trader and makes a profit by buying and selling stocks. He buys a stock at a lower price and sells it at a higher price to book a profit. He has come to know the stock prices of a particular stock for n upcoming days in future and wants to calculate the maximum profit by doing the right transactions (single transaction = buying + selling). Can you help him maximize his profit?

Note: A transaction starts after the previous transaction has ended. Two transactions can't overlap or run in parallel.

The stock prices are given in the form of an array A for n days.

Given the stock prices and a positive integer k, find and print the maximum profit Mike can make in at most k transactions.

Input Format:

The first line of input contains an integer T(number of test cases).
The first line of each test case contains a positive integer k, denoting the number of transactions.
The second line of each test case contains a positive integer n, denoting the length of the array A.
The third line of each test case contains n space-separated positive integers, denoting the prices of each day in the array A.

Output Format

For each test case print the maximum profit earned by Mike on a new line.

Constraints:

1 <= T <= 10^3
0 < k <= 10
2 <= n <= 10^4
0 <= elements of array A <= 10^5

Sample Input

```
3
2
6
10 22 5 75 65 80
3
4
20 580 420 900
1
5
100 90 80 50 25
```

Sample Output

```
87
1040
0
```

Explanation

Output 1: Mike earns 87 as the sum of 12 and 75 i.e. Buy at price 10, sell at 22, buy at 5 and sell at 80.
Output 2: Mike earns 1040 as the sum of 560 and 480 i.e. Buy at price 20, sell at 580, buy at 420 and sell at 900.

Output 3: Mike cannot make any profit as the selling price is decreasing day by day. Hence, it is not possible to earn anything.

```
1. #include<bits/stdc++.h>
2. using namespace std;
3.
4. int max_profit(int n,int k,int *prices,int curr_i,int ongoing,int ***dp){
5.     //if end of array reached
6.     if(curr_i==n||k==0){
7.         return 0;
8.     }
```

```
    if(dp[curr_i][k][ongoing]!=-1){
        return dp[curr_i][k][ongoing];
    }
    //ignore current num
    int opt1=max_profit(n,k,prices,curr_i+1,ongoing,dp);
    //buy or sell
    int opt2=INT_MIN;
    if(ongoing==1){
        //we can sell
                    opt2=prices[curr_i]+max_profit(n,k-1,prices,curr_i+1,0,dp);
    }else{
        //we can buy
        //buy only if k>0
        if(k>0){
            opt2=max_profit(n,k,prices,curr_i+1,1,dp)-prices[curr_i];
        }
    }
    int ans=max(opt1,opt2);
    dp[curr_i][k][ongoing]=ans;
    return ans;
}

int main(){

    // write your code here

     int q;
    cin>>q;
    while(q--){
        int k,n;
        cin>>k;
        cin>>n;
        int *prices=new int[n+1];
        for(int i=0;i<n;i++){
            cin>>prices[i];
        }
        int ***dp=new int**[n+1];
        for(int i=0;i<=n;i++){
            dp[i]=new int*[k+1];
            for(int j=0;j<=k;j++){
                dp[i][j]=new int[2];
                dp[i][j][0]=-1;
                dp[i][j][1]=-1;
            }
```

```
        }
        cout<<max_profit(n,k,prices,0,0,dp)<<endl;
        for(int i=0;i<=n;i++){
            delete dp[i];
        }
        delete dp;
        delete prices;
    }

    return 0;
}
```

10-Ass : Charlie and Pilots

Send Feedback

Charlie acquired airline transport company and to stay in business he needs to lower the expenses by any means possible. There are N pilots working for his company (N is even) and N/2 plane crews needs to be made. A plane crew consists of two pilots - a captain and his assistant. A captain must be older than his assistant. Each pilot has a contract granting him two possible salaries - one as a captain and the other as an assistant. A captain's salary is larger than assistant's for the same pilot. However, it is possible that an assistant has larger salary than his captain. Write a program that will compute the minimal amount of money Charlie needs to give for the pilots' salaries if he decides to spend some time to make the optimal (i.e. the cheapest) arrangement of pilots in crews.

Input Format:

First line will contain T(number of test case), each test case as follow.
The first line of each test case contains integer N, N is even, the number of pilots working for the Charlie's company.
The next N lines of input contain pilots' salaries. The lines are sorted by pilot's age, the salaries of the youngest pilot are given the first. Each of those N lines contains two integers separated by a space character, a salary as a captain (X) and a salary as an assistant (Y).

Constraints

1 <= T <= 20
2 ≤ N ≤ 1000
1 ≤ Y < X ≤ 100000

Output Format:

For each test case print the minimal amount of money Charlie needs to give for the pilots' salaries. in newline.

Sample Input

```
1
4
5000 3000
6000 2000
8000 1000
9000 6000
```

Sample Output

19000

Explanation

Out of various possible, optimal arrangements will be:
Plane Crew 1 will have Pilot1 as an assistant and Pilot2 as a Captain
Plane Crew2 will have Pilot3 as an assistant and Pilot4 as a Captain

Amount of money required= 3000+6000+1000+9000 =19000.

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

int dp[5001][5001];

int charlie_pilot(int cap,int ast,int n,vector<int>&cpt,vector<int>&ass)
{
    if(cap+ast>=n){
        return 0;
    }
    if(dp[cap][ast]!=-1){
        return dp[cap][ast];
    }
    int ans=INT_MAX;
    if(cap<n/2 and cap<ast){
        ans=min(ans,charlie_pilot(cap+1,ast,n,cpt,ass)+cpt[ast+cap]);
    }
    if(ast<n/2){
        ans=min(ans,charlie_pilot(cap,ast+1,n,cpt,ass)+ass[ast+cap]);
    }
    dp[cap][ast]=ans;
    return dp[cap][ast];
}

int main(){

    // write your code here

     int t;
    cin>>t;
    while(t--)
    {
        int n;
        cin>>n;
        vector<int>cpt(n);
        vector<int>ass(n);
```

```
        for(int i=0;i<n;i++){
            cin>>cpt[i]>>ass[i];
        }
        memset(dp,-1,sizeof(dp));
        cout<<charlie_pilot(0,0,n,cpt,ass)<<endl;
    }

    return 0;
}
```

11-Ass : Square Brackets

Send Feedback

You are given:

a positive integer n,
an integer k, 1<=k<=n,
an increasing sequence of k integers 0 < s1 < s2 < ... < sk <= 2n.
What is the number of proper bracket expressions of length 2n with opening brackets appearing in positions s1, s2,...,sk?

Illustration

Several proper bracket expressions:

[[]][[[]][]]
[[[][]]][][[]]

An improper bracket expression:

[[[][]]][]][[]]
There is exactly one proper expression of length 8 with opening brackets in positions 2, 5 and 7.

Task

Write a program which for each data set from a sequence of several data sets:

1. reads integers n, k and an increasing sequence of k integers from input,

2. computes the number of proper bracket expressions of length 2n with opening brackets appearing at positions s1,s2,...,sk,

3. writes the result to output.

Note: since result can be pretty large output the answer % mod (10^9 + 7).

Input Format:

The first line of the input file contains one integer T(number of test cases), each test case follows as.
The first line contains two integers n and k separated by single space.
The second line contains an increasing sequence of k integers from the interval [1;2n] separated by single spaces.

Output Format:

For each test case print the number of balanced square bracket sequence % mod (10^9 + 7), that can be formed using the above rules in a new line.

Constraints:

1 <= T <= 100
1 <= N <= 100
1 <= K <= N

Sample Input

5
1 1
1
1 1
2
2 1
1
3 1
2
4 2
5 7

Sample Output

1
0
2
3
2

Explanation

Output1: Proper bracket expressions of length 2 with opening brackets appearing in position 1- [].
Output2: Proper bracket expressions of length 2 with opening brackets appearing in position 2 - none.

Output3: Proper bracket expressions of length 4 with opening brackets appearing in position 1 - [][], [[]].

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

int mod = 1e9 + 7;
bool openings[201];
int dp[201][201];

int possibleBrackets(int open,int close,int n)
{
    if(openings[0])
    {
        return 0;
    }

    if(dp[open][close] != -1)
    {
```

```
17.         return dp[open][close];
18.     }
19.
20.     if(open > n || close > n)
21.     {
22.         return 0;
23.     }
24.
25.     if(open == n && close == n)
26.     {
27.         dp[open][close] = 1;
28.         return 1;
29.     }
30.
31.     int currIndex = (open+close+1);
32.
33.     if(open == close || openings[currIndex])
34.     {
35.         dp[open][close] = possibleBrackets(open+1,close,n);
36.     }
37.     else if(open == n)
38.     {
39.         dp[open][close] = possibleBrackets(open,close+1,n);
40.     }
41.     else
42.     {
43.         dp[open][close] = (possibleBrackets(open+1,close,n) +
    possibleBrackets(open,close+1,n))%mod;
44.     }
45.     return dp[open][close];
46. }
47.
48. int main(){
49.
50.     // write your code here
51.
52.     int d;
53.     cin >> d;
54.
55.     while(d--)
56.     {
57.         int n,k;
58.         cin >> n >> k;
59.         memset(openings,0,sizeof(openings));
```

```
        memset(dp,-1,sizeof(dp));

        for(int i = 0; i < k ; i++)
        {
            int m;
            cin >> m;
            openings[m] = true;
        }
        cout<<possibleBrackets(0,0,n)<<endl;
    }
    return 0;
}
```

12-Ass : Distinct Subsequences

Send Feedback

Given a string, count the number of distinct subsequences of it (including empty subsequence). For the uninformed, A subsequence of a string is a new string which is formed from the original string by deleting some of the characters without disturbing the relative positions of the remaining characters.

For example, "AGH" is a subsequence of "ABCDEFGH" while "AHG" is not.

Input Format:

First line of input contains an integer T which is equal to the number of test cases.
Each of next T lines contains a string s.

Output Format:

Output consists of T lines. Ith line in the output corresponds to the number of distinct subsequences of ith input string. Since, this number could be very large, you need to output ans%(10^9 + 7) where ans is the number of distinct subsequences.

Constraints:

T ≤ 100
1 <= length(S) ≤ 10^5
All input strings shall contain only uppercase letters.

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

typedef long long ll;
#define mod 1000000007
ll all_subseq(string s){
    int len=s.length();

    ll *dp=new ll[len+1];

    ll *previndice=new ll[26];
    for(int i=0;i<26;i++){
```

```
        previndice[i]=-1;
    }

    dp[0]=1;
    for(ll i=1;i<=len;i++){
        dp[i]=(dp[i-1]*2)%mod;

        if(previndice[s[i-1]-65]!=-1){

            dp[i]=(dp[i]-dp[previndice[s[i-1]-65]-1]+mod)%mod;
        }

            previndice[s[i-1]-65]=i;
    }
    ll ans=dp[len]%mod;
    delete dp;
            delete previndice;
    return ans;
}

int main(){
    // write your code here
     int t;
    cin>>t;
    while(t--){
        string s;
        cin>>s;
        map<string,int> m;
        cout<<all_subseq(s)<<endl;

    }
    return 0;
}
```

13-Ass : Smallest Super-Sequence

Send Feedback

Given two strings S and T, find and return the length of their smallest super-sequence.

A shortest super sequence of two strings is defined as the shortest possible string containing both strings as subsequences.

Note that if the two strings do not have any common characters, then return the sum of lengths of the two strings.

Input Format:

First line will contain T(number of test case), each test consists of two lines.

Line 1 : A string
Line 2: Another string

Output Format:

Length of the smallest super-sequence of given two strings for each test case in new line.

Constraints:

1 <= T <= 50
1 <= |str1|, |str2| <= 500

Sample Input:

1
ab
ac

Sample Output:

3

Sample Output Explanation:

Their smallest super-sequence can be "abc" which has length=3.

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

int smallestSuperSequenceHelper(string s1,string s2,int m,int n,int **dp){

    for(int i=m;i>=0;i--){
        for(int j=n;j>=0;j--){
            if(s1[i]==s2[j]){
                dp[i][j]=dp[i+1][j+1]+1;
            }
            else{
                dp[i][j]=min(dp[i+1][j],dp[i][j+1])+1;
            }
        }
    }

    return dp[0][0];
}

int smallestSuperSequence(string s1,string s2){
    int m=s1.length(),n=s2.length();
    int **dp=new int*[m+1];
    for(int i=0;i<=m;i++){
        dp[i]=new int[n+1];
    }
    int count=0;
    for(int i=n;i>=0;i--){
        dp[m][i]=count++;
```

```
    }
    count=0;
    for(int i=m;i>=0;i--){
        dp[i][n]=count++;
    }

    int ans=smallestSuperSequenceHelper(s1,s2,m-1,n-1,dp);
    return ans;
}

int main(){

    // write your code here
    int t;
    cin>>t;
    while(t--)
    {
        string s1,s2;
        cin>>s1>>s2;
        cout<<smallestSuperSequence(s1,s2)<<endl;
    }

    return 0;
}
```

14-Ass : Shortest Subsequence

Send Feedback

Gary has two string S and V. Now Gary wants to know the length shortest subsequence in S such that it is not a subsequence in V.

Note: input data will be such so there will always be a solution.

Input Format :

Line 1 : String S of length

Line 2 : String V of length

Output Format :

Length of shortest subsequence in S such that it is not a subsequence in V

Constraints:

1 <= |S|, |V| <= 1000

Sample Input :

babab

babba

Sample Output :

3

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

int solve(string s1,string s2)
{
	int m =s1.size();
	int n =s2.size();

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

	for (int i = 0; i < n+1; ++i)
	{
		dp[m][i] = n-i;
	}

	for (int i = 0; i < m+1; ++i)
	{
		dp[i][n] = m-i;
	}

	for (int i = m-1; i >= 0; --i)
	{
		for (int j = n-1; j >= 0; --j)
		{
			int k = j;

			while(k<s2.size())
			{
				if(s2[k]==s1[i])
				{
					break;
				}
				k++;
			}
			if (k==s2.size())
			{
				dp[i][j] = 1;
				//return 1;
			}

			int c1 = 1+dp[i+1][k+1];
			int c2 = dp[i+1][j];

			dp[i][j] = min(c1, c2);
```

```
                    }
                }

                return dp[0][0];


}

int main(){

    // write your code here
   // ios_base::sync_with_stdio(false) ;
                //cin.tie(NULL) ;

                string S,V;
                cin>>S>>V;
                cout<<solve(S,V)<<endl;

    return 0;
}
```

15-Ass : Balika Vadhu- Problem

Send Feedback

Anandi and Jagya were getting married again when they have achieved proper age. Dadi Sa invited Alok Nath to do the kanyadaan and give blessings. Alok Nath has 2 blessings. Each bessing is in the form of a string consisting of lowercase charaters(a-z) only. But he can give only one blessing of K length because some priest told him to do so. Thus he decides to generate a blessing using the other two blessings. While doing this he wants to ensure that happiness brought into their life by his blessing is maximum. The generated blessing is a common subsequence of length K of the two blessings he has. Happiness of the blessing he generates is calculated by the sum of ASCII values of characters in the blessing and he wants the happiness to be maximum. If he is not able to generate a common subsequence of length K then the happiness is 0 (zero). Alok Nath comes to you and asks you to find the maximum happiness that can be generated by the two blessings he has.

Input Format:

First line consists of number of test cases T.
Each test case consists of two strings b1 (blessing 1),b2 (blessing 2) and an integer K, each of them in separate lines.

Output Format:

Output consists of T lines each containing an integer denoting the maximum happiness value that can be generated by the two blessings.

Constraint:

1 <= T <= 50

1 <= length(b1) , length(b2) <= 100
1 <= K <= 100
Sample Input

2
asdf
asdf
3
anandi
jagya
3
Sample Output

317
0
Explanation

Output1: Maximum happiness value that can be generated by the two blessings is 317 from the sum of ASCII values of characters in the common subsequence "sdf" of length 3.

Output2: There is no way to generate a common subsequence of length 3, hence the happiness is 0.

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

int getCount(char* s1,char* s2,int m,int n,int k,int*** dp)
{
    if(m == 0 || n == 0){
        return 0;
    }

    if(k == 0){
        return 0;
    }

    if(k > m || k > n){
        return 0;
    }

    if(dp[m][n][k] > -1){
        return dp[m][n][k];
    }

    int ans = 0;
    if(s1[0] == s2[0]){
        int ascii = s1[0];
```

```
25.         int option1 = ascii + getCount(s1+1,s2+1,m-1,n-1,k-1,dp);//when included : get min
    sum for next k-1
26.         if(option1 - ascii == 0 && k > 1){
27.             option1 = 0;
28.         }
29.         int option2 = getCount(s1+1,s2,m-1,n,k,dp);//s1[0] = s2[0] but excluded to check
    wether we get min sum from remaining
30.         int option3 = getCount(s1,s2+1,m,n-1,k,dp);
31.         ans = max(option1, max(option2,option3));
32.     }else{
33.         int option1 = getCount(s1+1,s2,m-1,n,k,dp);//look for min sum in next characters
34.         int option2 = getCount(s1,s2+1,m,n-1,k,dp);
35.         ans = max(option1,option2);
36.     }
37.     dp[m][n][k] = ans;
38.     return ans;
39. }
40.
41.
42. int main(){
43.
44.     // write your code here
45.
46.      int t;
47.     cin >> t;
48.     while(t--){
49.         string s1,s2;
50.         cin >> s1;
51.         cin >> s2;
52.         int k;
53.         cin >> k;
54.         int m = s1.length();
55.         int n = s2.length();
56.         char* arr1 = new char[m];
57.         char* arr2 = new char[n];
58.         for(int i=0;i<m;i++){
59.             arr1[i] = s1[i];
60.         }
61.         for(int i=0;i<n;i++){
62.             arr2[i] = s2[i];
63.         }
64.
65.         int*** dp = new int**[m+1];
66.         for(int i=0;i<=m;i++){
```

```
            dp[i] = new int*[n+1];
            for(int j=0;j<=n;j++){
                dp[i][j] = new int[k+1];
                for(int a=0;a<=k;a++){
                    dp[i][j][a] = -1;
                }
            }
        }
        cout << getCount(arr1,arr2,m,n,k,dp) << endl;
    }

    return 0;
}
```

16-Ass : Adjacent Bit Counts

Send Feedback

For a string of n bits x1,x2,x3,...,Xn the adjacent bit count of the string (AdjBC(x)) is given by

X1*X2 + X2*X3 + X3*X4 + ... + Xn-1 * Xn

which counts the number of times a 1 bit is adjacent to another 1 bit. For example:

AdjBC(011101101) = 3

AdjBC(111101101) = 4

AdjBC(010101010) = 0

Write a program which takes as input integers n and k and returns the number of bit strings x of n bits (out of 2^n) that satisfy AdjBC(x) = k. For example, for 5 bit strings, there are 6 ways of getting AdjBC(x) = 2:

11100, 01110, 00111, 10111, 11101, 11011

Input Format:

First-line will contain T(number of the test case).
Each test case consists of a single line containing two space-separated integers N and K, a number of bits in the bit strings and desired adjacent bit count respectively.

Output Format:

For each test case print the answer in a new line.
As answer can be very large print your answer modulo 10^9+7.

Constraints:

1 <= T <= 10^5
1 <= N <= K <= 100

Sample Input

10
5 2
20 8
30 17
40 24
50 37

60 52
70 59
80 73
90 84
100 90

Sample Output

6
63426
1861225
168212501
44874764
160916
22937308
99167
15476

23076518

```
#include<bits/stdc++.h>
using namespace std;
#define mod 1000000007

int main(){

    // write your code here
    int dp[101][101][2];
    memset(dp, 0, sizeof(dp));
    dp[1][0][0] = 1;
    dp[1][0][1] = 1;
    for(int i=2;i<=100;i++)
    {
        dp[i][i-1][0] = 0;
        dp[i][i-1][1] = 1;
        dp[i][0][0] = (dp[i-1][0][0] + dp[i-1][0][1]) % mod;
        dp[i][0][1] = dp[i-1][0][0];
    }
    for(int i=2;i<=100;i++)
    {
        for(int j=1;j<i;j++)
        {
            dp[i][j][0] = (dp[i-1][j][0] + dp[i-1][j][1]) % mod;
            dp[i][j][1] = (dp[i-1][j][0] + dp[i-1][j-1][1]) % mod;
        }
    }
    int t;
    cin>>t;
```

```
    while(t--)
    {
        int n,k;
        cin>>n>>k;
        if(n==0 && k==0)
            cout<<1<<endl;
        else
        {
            int ans = (dp[n][k][0] + dp[n][k][1]) % mod;
            cout<<ans<<endl;
        }
    }

    return 0;
}
```