Análisis de Algoritmos

Ejercicios 10:

"Diseño de soluciones mediante programación dinámica (DP)"

INSTITUTO POLITÉCNICO NACIONAL
ESCOM

Nombre: Luis Fernando Ramírez Cotonieto

Fecha de entrega: 12 de Junio del 2021

Grupo: 3CM13

Problema 02: ELIS - Easy Longest Increasing Subsequence

ELIS - Easy Longest Increasing Subsequence

no tags

Given a list of numbers A output the length of the longest increasing subsequence. An increasing subsequence is defined as a set {i0 , i1 , i2 , i3 , ... , ik} such that 0 <= i0 < i1 < i2 < i3 < ... < ik < N and A[i0] < A[i1] < A[i2] < ... < A[ik]. A longest increasing subsequence is a subsequence with the maximum k (length).

i.e. in the list {33 , 11 , 22 , 44}

the subsequence {33 , 44} and {11} are increasing subsequences while {11 , 22 , 44} is the longest increasing subsequence.

Input

First line contain one number N (1 <= N <= 10) the length of the list A.

Second line contains N numbers (1 <= each number <= 20), the numbers in the list A separated by spaces.

Output

One line containing the lenght of the longest increasing subsequence in A.

SPOJ.com - Problem RMQSQ

...

spoj.com

Example

Input:

5

1 4 2 4 3
Output:
3

Aceptación de juez:

ID	DATE	USER	PROBLEM	RESULT	TIME	MEM	LANG
28040211	2021-06-11 07:37:42	Luis Coto	Easy Longest Increasing Subsequence	**accepted** edit ideone it	0.04	5.4M	CPP14

Análisis de complejidad:

O(1)

Explicación:

La subsecuencia creciente más larga que termina en el índice i será 1 mayor que el máximo de longitudes de todas las subsecuencias crecientes más largas que terminan en los índices antes de i, donde arr [i] <arr [j]. Por lo que, el problema satisface la propiedad de subestructura óptima ya que el problema principal se puede resolver utilizando soluciones a los subproblemas.

Código:

```
#include <stdio.h>
#include <algorithm>
#include <memory.h>

using namespace std;

int arr[11], dp[11];
int n;

int calc(int i) {
    int &res = dp[i];

    if(res != -1) return res;

    res = 0;
    for(int j = i + 1; j < n; j++)
        if(arr[i] < arr[j])
            res = max(res, calc(j) + 1);

    return res;
}

int main() {
    scanf("%i", &n);
    for(int i = 0; i < n; i++) scanf("%i", &arr[i]);

    memset(dp, -1, sizeof dp);

    int res = 0;
```

```
    for(int i = 0; i < n; i++) res = max(res, calc(i) + 1);
    printf("%i\n", res);

    return 0;
}
```

Problema 06: BYTESM2 - Philosophers Stone

BYTESM2 - Philosophers Stone

#dynamic-programming

One of the secret chambers in Hogwarts is full of philosopher's stones. The floor of the chamber is covered by h × w square tiles, where there are h rows of tiles from front (first row) to back (last row) and w columns of tiles from left to right. Each tile has 1 to 100 stones on it. Harry has to grab as many philosopher's stones as possible, subject to the following restrictions:

- He starts by choosing any tile in the first row, and collects the philosopher's stones on that tile. Then, he moves to a tile in the next row, collects the philosopher's stones on the tile, and so on until he reaches the last row.
- When he moves from one tile to a tile in the next row, he can only move to the tile just below it or diagonally to the left or right.

Given the values of h and w, and the number of philosopher's stones on each tile, write a program to compute the maximum possible number of philosopher's stones Harry can grab in one single trip from the first row to the last row.

Input

The first line consists of a single integer T, the number of test cases. In each of the test cases, the first line has two integers. The first integer h (1 <= h <= 100) is the number of rows of tiles on the floor. The second integer w (1 <= w <= 100) is the number of columns of tiles on the floor. Next, there are h lines of inputs. The i-th line of these, specifies the number of philosopher's stones in each tile of the i-th row from the front. Each line has w integers, where each integer m (0 <= m <= 100) is the number of philosopher's stones on that tile. The integers are separated by a space character.

Output

The output should consist of T lines, (1 <= T <= 100), one for each test case. Each line consists of a single integer, which is the maximum possible number of philosopher's stones Harry can grab, in one single trip from the first row to the last row for the corresponding test case.

Aceptación de juez:

ID	DATE	USER	PROBLEM	RESULT	TIME	MEM	LANG
28040500	2021-06-11 08:36:36	Luis Coto	Philosophers Stone	**accepted** edit ideone it	0.04	5.2M	CPP

Análisis de complejidad:

O(n)

Explicación:

Se utiliza el “top-down”, se considera la fila actual donde esta el algoritmo y la fila siguiente. A partir de estas dos filas, se crea una lista de listas que muestra posibles valores de suma cuando se llega a ese elemento específico. Este proceso se repite hasta que se llega a la última fila. A partir de ahí vemos todas las sumas posibles de todas las secuencias posibles. Podemos encontrar el máximo de esas sumas para responder parcialmente al problema. Después se localiza la suma maxima.

Código:

```
#include <bits/stdc++.h>
using namespace std;
typedef long long int ll;
const ll SZ = 1234567;
ll h, w;
ll mat[100+5][100+5];
ll dp[100+5][100+5];
ll solve() {
    memset(dp,0,sizeof dp);
    for(ll j=0; j<w; j++) {
        dp[0][j] = mat[0][j];
    }
    for(ll i=0; i<h-1; i++) {
        for(ll j=0; j<w; j++) {
            if(j>0) {
                dp[i+1][j-1] = max(dp[i+1][j-1], dp[i][j] + mat[i+1][j-1]);
            }
            if(j<w) {
                dp[i+1][j+1] = max(dp[i+1][j+1], dp[i][j] + mat[i+1][j+1]);
            }
            dp[i+1][j] = max(dp[i+1][j], dp[i][j] + mat[i+1][j]);
        }
    }
    ll ans = 0;
    for(ll j=0; j<w; j++) {
        ans = max(ans, dp[h-1][j]);
    }
    return ans;
```

```
}
int main() {
    ll T;
    scanf("%lld",&T);
    while(T--) {

        scanf("%lld%lld",&h,&w);

        for(ll i = 0; i<h; i++) {
            for(ll j =0 ; j<w; j++) {
                scanf("%lld",&mat[i][j]);
            }
        }
        printf("%lld\n",solve());

    }
    return 0;
}
```

Problema 07: KNAPSACK - The Knapsack Problem

KNAPSACK - The Knapsack Problem

no tags

The famous knapsack problem. You are packing for a vacation on the sea side and you are going to carry only one bag with capacity S (1 <= S <= 2000). You also have N (1<= N <= 2000) items that you might want to take with you to the sea side. Unfortunately you can not fit all of them in the knapsack so you will have to choose. For each item you are given its size and its value. You want to maximize the total value of all the items you are going to bring. What is this maximum total value?

Input

On the first line you are given S and N. N lines follow with two integers on each line describing one of your items. The first number is the size of the item and the next is the value of the item.

Output

You should output a single integer on one like - the total maximum value from the best choice of items for your trip.

SPOJ.com - Problem RMQSQ

...

spoj.com

Aceptación de juez:

ID	DATE	USER	PROBLEM	RESULT	TIME	MEM	LANG
28040898	2021-06-11 10:08:46	Luis Coto	The Knapsack Problem	**accepted** edit ideone it	0.03	19M	CPP14

Análisis de complejidad:

O(N*W)

Explicación:

Se construye un array temporal K[][] de abajo hacia arriba.

Código:

```
#include <iostream>
#include <cstdio>
#include <algorithm>
using namespace std;

int main() {
long long int a[2002][2002];
long long int weight[2002];
long long int  value[2002];
long long  int i,j,k,S,N,op;
scanf("%lld %lld",&S,&N);
for(i=1;i<=N;i++)
{
    scanf("%lld",&weight[i]);
    scanf("%lld",&value[i]);
}
for(i=0;i<N;i++)
{
```

```
    a[0][i]=0;
    a[i][0]=0;
}
for(i=1;i<=N;i++)
{
    for(j=1;j<=S;j++)
    {
     if(j<weight[i])
     a[i][j]=a[i-1][j];
        else
    {
          op=value[i]+a[i-1][j-weight[i]];
          a[i][j]=max(a[i-1][j],op);
    }
    }
}
printf("%lld\n",a[N][S]);
 return 0;
}
```