ANÁLISIS DE ALGORITMOS

Alumno: Martín Eduardo Barriga Vargas

Grupo 3CM3

Ejercicio 05: Análisis de algoritmos recursivos

3 de mayo de 2019

Divide And Conquer 1

Puntos	14.07	Límite de memoria	32 MiB
Límite de tiempo (caso)	1s	Límite de tiempo (total)	1m0s
Límite de entrada (bytes)	10 KiB		

Descripción

Edgardo se puso un poco intenso este semestre y puso a trabajar a sus alumnos con problemas de mayor dificultad.

La tarea es simple, dado un arreglo A de números enteros debes imprimir cual es la suma máxima en cualquier subarreglo contiguo.

Por ejemplo si el arreglo dado es {-2, -5, **6, -2, -3, 1, 5**, -6}, entonces la suma máxima en un subarreglo contiguo es **7**.

Entrada

La primera línea contendrá un numero N

En la siguiente línea N enteros representando el arreglo A

Salida

La suma máxima en cualquier subarreglo contiguo.

Ejemplo

Entrada	Salida
8 −2 −5 6 −2 −3 1 5 −6	7
5 1 2 3 4 5	15

2019-04-25 06:02:00	1fdb9c99	Respuesta correcta	100.00%	cpp	6.89 MB	0.32 s	
2019-04-25 06:00:51	ef754174	Respuesta parcialmente correcta	33.33%	cpp	3.82 MB	0.34 s	
2019-04-25 05:58:29	14c0bed9	Respuesta parcialmente correcta	33.33%	cpp	3.84 MB	0.32 s	
2019-04-25 05:56:07	4bc2526b	Respuesta parcialmente correcta	16.67%	cpp	3.42 MB	0.32 s	
2019-04-25 05:54:39	aef190a8	Respuesta parcialmente correcta	33.33%	cpp	3.41 MB	0.32 s	
2019-04-25 05:52:41	a1220033	Límite de memoria excedido	50.00%	cpp	>32.00 MB	0.38 s	

```
1.  #include <iostream>
2.  #include <vector>
3.  #include <math.h>
4.  using namespace std;
5.
6.  long long int maximo,valorMaximo=-INFINITY;
7.  int n;
8.   long long int  max(long int *v, int pos){
9.      if(pos < (n - 1)){
10.         maximo = max(v,pos+1);
11.         if(v[pos]+maximo < v[pos])
12.         {
13.             if(valorMaximo < v[pos]) valorMaximo = v[pos];
```

```
14.             return v[pos];
15.         }
16.         else{
17.
      if(valorMaximo < v[pos] + maximo) valorMaximo =v[pos]
    +maximo;
18.             return (v[pos] + maximo);
19.         }
20.     }
21.     else{
22.         return v[pos];
23.     }
24. }
25.
26. int main(){
27.     cin >> n;
28.     long int v[n], num;
29.     for(int i = 0; i < n; i++){
30.         cin >> num;
31.         v[i] = num;
32.     }
```

```
33.     max(v,0);
34.     cout << valorMaximo<< endl;
35.     return 0;
36. }
```

Explicación de la solución:

El problema se puede reducir a decir que para que un número forme parte del subarreglo es conveniente preguntar si el valor de la suma de los dígitos de ese subarreglo(que tiene a la derecha) es positivo o negativo, en caso de ser positivo, entonces a ese número le conviene formar parte del subarreglo que tenía en la derecha, sino entonces él solito formaría su propio subarreglo. Con esto dicho sólo tenemos que preguntar por la suma del subarreglo que comienza en la posición siguiente a la mío, es decir n+1, hacer el diagnóstico si nos conviene sumarnos a él, en ese caso regresamos la suma del número en la posición n más el valor del subarreglo de n+1, en caso contrario sólo regresamos el valor del número en la posición n.

Análisis de complejidad:

$T(n) = T(n-1) + 1$
$T(0) = 1$
Reordenando términos
$T(n) - T(n-1) = 1$
Es no homogénea
Haciendo el cambio $x^{k=1} = T(n), b = 1, d = 0$
Obtenemos su ecuación característica $(x-1)(x-1) = 0$
Por lo cual $r1 = 1, r2 = 1$
$T(n) = c1(1)^n + nc2(1)^n$ Si c1 o c2 son diferentes de 0, la complejidad será **O(n)**, lo cual se puede analizar al ver cómo está compuesta la función, pues va recorriendo el arreglo de tamaño n de forma recursiva hasta llegar al límite, además de eso sólo hace operaciones lineales.

Con pocos ceros consecutivos

Puntos	24.47	Límite de memoria	32 MiB
Límite de tiempo (caso)	10s	Límite de tiempo (total)	1m0s
Límite de entrada (bytes)	10 KiB		

Descripción

Escribe un programa que calcule el número cadenas binarias distintas de longitud N que no tienen más de dos ceros consecutivos.

Entrada

Un entero N.

Puedes suponer que $0 \leq N \leq 33$

Salida

Un entero que sea el número cadenas binarias distintas de longitud N que no tienen más de dos ceros consecutivos.

Ejemplo

Entrada	Salida
2	4
3	7

2019-04-25 03:34:33	dbf1b446	Respuesta correcta	100.00%	cpp	3.07 MB	5.63 s
2019-04-25 03:08:55	cb7ec887	Respuesta parcialmente correcta	20.61%	cpp	3.27 MB	0.05 s
2019-04-25 02:47:52	e862d4cf	Respuesta parcialmente correcta	17.68%	cpp	3.09 MB	0.07 s
2019-04-25 02:45:58	e5508146	Respuesta parcialmente correcta	14.75%	cpp11	3.08 MB	0.04 s

```
1.  #include <iostream>
2.  #include <math.h>
3.  using namespace std;
4.
5.  int ceros(int n){
6.      if(n == 0) return 1;
7.      else if(n<4) return 2*ceros(n-1) - (n-1)/2;
8.      else{
9.          return ceros(n-1) + ceros(n-2)+ ceros(n-3) ;
10.     }
11. }
12. int main(){
13.     int res, n;
14.     cin >> n;
15.     res = ceros(n);
16.     cout << res << endl;
17. }
```

Explicación de la solución:

Se observa en nuestras cadenas de longitud n en donde almacenamos nuestros 0's y 1's que representan un número en binario, para la longitud n, tendrá exactamente las mismas cadenas que la longitud n-1 pero con un 1 antes o un 0.
Ejemplo:

Longitud n = 3

0 0 0
0 0 1
0 1 0
0 1 1

1 0 0
1 0 1
1 1 0
1 1 1

Longitud n = 2

0 0
0 1
1 0
1 1

Longitud n = 4

0 0 0 0
0 0 0 1
0 0 1 0
0 0 1 1
0 1 0 0
0 1 0 1
0 1 1 0
0 1 1 1

1 0 0 0
1 0 0 1
1 0 1 0
1 0 1 1
1 1 0 0
1 1 0 1
1 1 1 0
1 1 1 1

Entonces, enfocándonos en la resolución del problema recordamos que nos piden la cantidad de cadenas que no tienen más de 2 ceros consecutivos para una longitud n, debido a que sabemos que la mitad de la cantidad de cadenas que tenemos llevan un 1 al inicio y es la copia exacta de las cadenas de longitud n-1, sabemos que parte de nuestra solución será el resultado del problema para la longitud n-1, que se puede entender como ceros(n-1). Ahora sólo hace falta saber cuantas de nuestras cadenas a las que le agregamos un 0 seguirán siendo válidas. Entonces recordamos que a esas antes de agregarles el 0 al inicio, debido a la longitud 4, antes también tenían la mitad de 0's y la mitad de 1's (Respuesta de la longitud 3) Por lo cual aplicamos el mismo razonamiento y decimos que el resultado de los que tienen 1's al inicio será el

resultado de la longitud n = 2, es decir, ceros(n-2). Ahora sólo nos falta saber cuáles son los números que tampoco formaban parte de esa mitad, la cual es el resultado de la longitud n = 3. Debido a esto, se regresa nuestra solución, planteada como: return ceros(n-1) + ceros(n-2) + ceros(n-3).
Para los números que sean menores a 4, esto no se va a cumplir, así que simplemente les asignaremos el valor que ya conocemos, el cuál es 7 para n=3, 4 para n=2 y 1 para n=1. Para que se viera todavía más el uso de divide y vencerás calculé los valores de estos números usando 2*ceros(n-1) - (n-1)/2, y en el momento que n fuera igual a 0, regresaba un 1. Esta fórmula formaba parte de mi solución inicial, pero sólo funcionaba para esos 3 primeros números. Es por eso que decidí dejarla para la solución de esos casos y no reemplazarla por las condiciones que mencioné anteriormente.

Análisis de complejidad:

$T(n) = T(n-1) + T(n-2) + T(n-3) + 1, n > 4$
$T(n) = T(n-1) + 1, 4 > n > 0$
$T(0) = 1$
Reordenando
$T(n) - T(n-1) - T(n-2) - T(n-3) = 1$
Es no homogénea
Haciendo el cambio $x^{k=3} = T(n), b = 1, d = 0$
Obtenemos la ecuación característica
$(x^3 - x^2 - x - 1)(x - 1) = 0$
$r1 = 1, r2 = 1.8$
$T(n) = c1(1)^n + c2(1.8)^n$, por lo tanto la complejidad es $O(1.8^n)$

INVCNT - Inversion Count

#graph-theory #number-theory #shortest-path #sorting #bitmasks

Let A[0...n - 1] be an array of n distinct positive integers. If i < j and A[i] > A[j] then the pair (i, j) is called an inversion of A. Given n and an array A your task is to find the number of inversions of A.

Input

The first line contains t, the number of testcases followed by a blank space. Each of the t tests start with a number n (n <= 200000). Then n + 1 lines follow. In the ith line a number A[i - 1] is given (A[i - 1] <= 10^7). The (n + 1)th line is a blank space.

Output

For every test output one line giving the number of inversions of A.

Example

Input:
2

3
3
1
2

5
2
3
8
6
1

Output:
2
5

ID		DATE	PROBLEM	RESULT	TIME	MEM	LANG
23687741	☐	2019-04-26 10:44:13	Inversion Count	**accepted** edit ideone it	0.16	18M	CPP
23687733	☐	2019-04-26 10:43:12	Inversion Count	wrong answer edit ideone it	0.16	16M	CPP
23687717	☐	2019-04-26 10:40:49	Inversion Count	wrong answer edit ideone it	0.15	16M	CPP
23687684	☐	2019-04-26 10:34:01	Inversion Count	wrong answer edit ideone it	0.16	16M	CPP
23687669	☐	2019-04-26 10:30:23	Inversion Count	wrong answer edit ideone it	0.15	16M	CPP14

```
1.  #include<iostream>
2.  #include<vector>
3.  using namespace std;
4.  long long int merge(vector <int> &v, int ini, int fin){
5.      if(ini!= fin){
6.          int mid = (ini + fin)/2;
7.          long long int acum;
8.          acum = merge(v,ini,mid);
9.          acum += merge(v, mid+1, fin);
10.         int izq = ini;
11.         int der = mid+1;
12.         int aux;
```

```
13.         vector <int> v2;
14.         while(izq <= mid && der<=fin){
15.             if(v[izq] > v[der]){
16.                 v2.push_back(v[der]);
17.                 der++;
18.                 acum+=mid-izq+1;
19.             }
20.             else{
21.                 v2.push_back(v[izq]);
22.                 izq++;
23.             }
24.         }
25.         while(izq <= mid){
26.             v2.push_back(v[izq]);
27.             izq++;
28.         }
29.         while(der <= fin){
30.             v2.push_back(v[der]);
31.             der++;
32.         }
```

```
33.         while(!v2.empty()){
34.             v[fin] = v2.back();
35.             v2.pop_back();
36.             fin--;
37.         }
38.         return acum;
39.     }
40.     return 0;
41. }
42. int main(){
43.     int p,n,i,j,num;
44.     long long int acum;
45.     cin >> p;
46.     for(i = 0; i < p; i++){
47.         cin >> n;
48.         vector <int> v;
49.         for(j = 0; j < n; j++){
50.             cin >> num;
51.             v.push_back(num);
52.         }
```

```
53.         acum = merge(v, 0, v.size()-1);
54.         /*for(j=0; j <n; j++){
55.             cout << v[j] <<" ";
56.         }*/
57.         cout << acum << endl;
58.     }
59.     return 0;
60. }
```

Explicación de la solución:

Entre las primeras ideas que se pueden llegar a tener es el hacer una comparación de todos los números con todos y con un contador ir llevando el total de números que cumplen con la condición, sin embargo esto es $O(n^2)$
Por lo que comenzamos a pensar en otra manera de lograrlo. Básicamente la idea anterior se podría implementar con un ordenamiento burbuja, por lo cual esto nos hace pensar que también se podría intentar algo con ordenamiento merge, el cual tiene una complejidad menor: $O(nlog(n))$.
Entonces comenzamos a partir nuestro arreglo tal como lo haríamos con merge, llegaríamos al caso base y luego comenzaríamos a regresar en la pila de recursión. Cuando nos toca que hacer la combinación de los elementos ordenados del lado izquierdo y los elementos del lado derecho comparamos uno con el otro para saber el orden que tienen que poner en el nuevo recipiente que los contendrá (esto lo hago utilizando un nuevo vector, una vez que esté completo vaciaremos su contenido en el vector original otra vez). Sin embargo el núcleo de nuestro algoritmo se encuentra al hacer al comparación, pues nos damos cuenta de que si el elemento de la derecha es menor que el de la izquierda le tenemos que sumar a nuestro contador de inversiones el indice límite del conjunto izquierdo menos la posición del número que estamos cambiando, pues con esto estaremos contando la cantidad de números que eran menores que el número del rango derecho que estamos cambiando y una

vez acabado este proceso, regresamos el acumulado. Unimos nuestros resultados de todas nuestras llamadas a función y lo regresamos.

Análisis de complejidad:

$T(n) = 2T(n/2) + n, n > 0$
$T(0) = 1$
Utilizando teorema maestro
$f(n) = n, a = 2, b = 2, c = 1, k = 1$
$2 = 2^1$, por lo tanto $f(n) = \theta(n^{log(2)})$
$T(n) = \theta(n^{log(2)} log(n))$

La complejidad resultante tiene mucho sentido, pues ya sabíamos que el merge tienen complejidad O(nlogn), así que como sólo estamos metiendo operaciones lineales extras al algoritmo orignal, no debería cambiar mucho.

STRLCP - Longest Common Prefix

no tags

The LCP (Longest Common Prefix) of two strings A[1..la] and B[1..lb] is defined as follows:

LCP(A[1..la],B[1..lb]) = max{L | L<=la && L<=lb && A[1..L] == B[1..L]}

Given an original string and several operations, you should write a program to process all the operations.

Input

The first line will be number of test cases T.
The first line of each test case is a string S with length L (1 <= L <= 100000).
The second line contains an integer Q(1 <= Q <= 150000), representing the number of operations.

Each of the following Q lines represents an operation:
Q i j: print LCP(S[i..L], S[j..L])
R i char: replace the i-th character of S with char
I i char: insert character char after the i-th character of S

Output

For each "Q i j" operation, print the answer.

Example

Input:
1
madamimadam
7
Q 1 7
Q 4 8

Q 10 11
R 3 a
Q 1 7
I 10 a
Q 2 11

Output:
5
1
0
2
1

ID		DATE	PROBLEM	RESULT	TIME	MEM	LANG
23710653	☐	2019-05-02 18:55:27	Longest Common Prefix	**accepted** edit ideone it	0.08	16M	CPP14
23710641	☐	2019-05-02 18:49:55	Longest Common Prefix	**accepted** edit ideone it	0.08	15M	CPP14
23710613	☐	2019-05-02 18:36:40	Longest Common Prefix	**accepted** edit ideone it	0.08	16M	CPP14

```
1. #include<iostream>
2. #include <string>
3. using namespace std;
4.
5. int lcp (string s, int a, int b){
6.     if(s[a] == s[b] && a < s.size() && b < s.size()){
7.         return 1 + lcp (s, a+1, b+1);
8.     }
```

```
9.      else{
10.         return 0;
11.     }
12. }
13. void remplazar(string &s, int a, char c){
14.     s[a] = c;
15. }
16.
17. void insertar(string &s, int a, int pos, char c){
18.     if(pos == a){
19.         s.push_back(c);
20.     }
21.     else{
22.         char letra = s.back();
23.         s.pop_back();
24.         insertar(s, a, pos-1,c);
25.         s.push_back(letra);
26.     }
27.
28. }
```

```
29.
30. int main(){
31.     int t,q,a,b,res;
32.     char c,mov;
33.     string s;
34.     cin >> t;
35.     for(int i = 0; i < t; i++){
36.         cin >>s >> q;
37.         for( int j = 0; j < q; j++){
38.             cin >> mov >> a;
39.             if(mov == 'R' || mov == 'I') cin >> c;
40.             else cin >> b;
41.             if(mov == 'R') remplazar(s, a-1, c);
42.             else if(mov =='I') insertar(s, a, s.size(),c);
43.             else{
44.                 res = lcp(s, a-1, b-1);
45.                 cout << res << endl;
46.             }
47.
48.         }
```

```
49.     }
50.     return 0;
51. }
```

Explicación de la solución:

Una vez que se lee el string y la cantidad de operaciones que se van a realizar, procedemos a leer las instrucciones una por una, si la entrada es igual a R o a I, leeremos un caracter que representará la letra que agregaremos al string, una vez que se hizo esto, si el caracter era un R entonces se reemplaza el caracter en la posición indicada por el nuevo caracter, si es una I, entonces vacíamos nuestro string (como si fuera una pila) en un string2, agregamos a nuestro string el nuevo caracter y luego vaciamos el string2(como si fuera pila) a nuestro string. En caso de que el caracter haya sido una Q, se llama a una función con los parámetros “i” y “j” y un contador de coincidencias comenzando en 0, se llamará a sí misma siempre y cuando el caracter del string en la posición i sea igual al caracter del string en la posición j y siempre y cuando “i” y “j” sean menores a la longitud del string, se retorna el 1 más el valor que nos devuelva la función , la cual se llama aumentando tanto la posición “i” como “j” y se aumenta en uno nuestro contador de coincidencias. En caso de que no se haya cumplido la posición regresamos un 0.

Análisis de complejidad:

$T(n) = T(n-1) + 1$
$T(0) = 1$
Reordenando términos
$T(n) - T(n-1) = 1$
Es no homogénea
Haciendo el cambio $x^{k=1} = T(n), b = 1, d = 0$
Obtenemos su ecuación característica $(x-1)(x-1) = 0$
Por lo cual $r1 = 1, r2 = 1$
$T(n) = c1(1)^n + nc2(1)^n$
La complejidad sería **O(n)** para cada query.

SKYLINE - Skyline

no tags

The director of a new movie needs to create a scaled set for the movie. In the set there will be N skyscrapers, with distinct integer heights from 1 to N meters. The skyline will be determined by the sequence of the heights of the skyscrapers from left to right. It will be a permutation of the integers from 1 to N.

The director is extremely meticulous, so she wants to avoid a certain sloping pattern. She doesn't want for there to be ANY three buildings in positions i, j and k, i < j < k, where the height of building i is smaller than that of building j, and building j's height is smaller than building k's height.

Your task is to tell the director, for a given number of buildings, how many distinct orderings for the skyline avoid the sloping pattern she doesn't like.

Input

There will be several test cases in the input. Each test case will consist of a single line containing a single integer N ($3 \leq N \leq 1{,}000$), which represents the number of skyscrapers. The heights of the skyscrapers are assumed to be 1, 2, 3, ..., N. The input will end with a line with a single 0.

Output

For each test case, output a single integer, representing the number of good skylines - those avoid the sloping pattern that the director dislikes - modulo 1,000,000. Print each integer on its own line with no spaces. Do not print any blank lines between answers.

Example

Input:

```
3
4
0
```

Output:
5
14

ID		DATE	PROBLEM	RESULT	TIME	MEM	LANG
23713175	☐	2019-05-03 12:08:07	Skyline	**accepted** edit ideone it	0.00	6.5M	C++ 4.3.2
23712856	☐	2019-05-03 10:28:59	Skyline	wrong answer edit ideone it	0.00	15M	CPP14
23712855	☐	2019-05-03 10:28:42	Skyline	wrong answer edit ideone it	0.00	15M	CPP14
23712820	☐	2019-05-03 10:16:09	Skyline	wrong answer edit ideone it	0.00	15M	CPP14
23712735	☐	2019-05-03 09:57:26	Skyline	wrong answer edit ideone it	0.00	15M	CPP14
23712723	☐	2019-05-03 09:55:46	Skyline	wrong answer edit ideone it	0.00	16M	CPP14
23688104	☐	2019-04-26 12:20:19	Skyline	wrong answer edit ideone it	0.00	16M	CPP
23688101	☐	2019-04-26 12:19:41	Skyline	wrong answer edit ideone it	0.00	16M	CPP
23688099	☐	2019-04-26 12:18:58	Skyline	wrong answer edit ideone it	0.00	16M	CPP

```
1.  #include <iostream>
2.  using namespace std;
3.
4.  int matriz[1000][1000];
5.  void hankel(int n){
6.      if(n==3){
7.          matriz[0][0]=1;
8.          matriz[0][1]=3;
9.          matriz[0][2]=5;
10.         matriz[0][3]=5;
11.     }
```

```
12.     else{
13.         hankel(n-1);
14.         matriz[n-3][0] = 1;
15.         for(int i = 1; i < n; i++){
16.             matriz[n-3][i]= (matriz[n-3][i-1] + matriz[n-4]
    [i])%1000000;
17.         }
18.         matriz[n-3][n] = matriz[n-3][n-1];
19.     }
20. }
21. int main(){
22.     int n,resultado;
23.     hankel(1000);
24.     do{
25.         cin >> n;
26.         if(n !=0){
27.             resultado = matriz[n-3][n];
28.             cout << resultado << endl;
29.         }
30.     }while(n !=0);
```

```
31.
32.     return 0;
33. }
```

Explicación de la solución:

Al inicio este problema nos hace ver que tenemos que encontrar de alguna forma alguna relación entre los resultados de las permutaciones anteriores (resultado para n-1) que fueron válidas con las que nos están pidiendo para nuestra n. Sin embargo, después de hacer pruebas de escritorio y encontrar los resultados que deben de dar para las primeras n “a mano” se ve que esta sucesión pertenece a los números de catalán, los cuáles pueden ser calculados con el uso de una matriz, de nombre hekel. Por lo cuál optamos a simplemente realizar la matriz que contenga la sucesión desde el 3 hasta el 1000, pues es el rango posible de la entrada. Programamos el teorema, el cual nos indica que al inicio la matriz está llena de 0. Después dice que la posición i, j en una matriz(matriz[i][j]) será igual a la suma de matriz [i][j-1] + [i-1][j], excepto cuando j = 0, en este caso valdrá 1. Además la cantidad de columnas que se llenarán con respecto a la fila será de i-1. Para consultar el resultado para alguna n, se obtendrá del valor en matriz[n-3][n]. Así que sólo mandamos llamar a la función con la n hasta la que queremos llegar, nuestro caso base es n=3, cuando cae ahí se meten en la matriz valores que ya conocemos de la sucesión para esa n. En caso contrario, se manda a llamar a la funcion con n-1 como parámetro, y una vez que regresa la pila de recursión avanzando a la siguiente línea, se sabe que la fila anterior ya está llenada, así que calculamos los valores para fila actual. Una vez que se termina el proceso de la función, se imprime el valor en la posición matriz[n-3][n], el n-3 es debido a que comenzamos a llenar a partir del valor n=3.

Análisis de complejidad:

$T(n) = T(n-1) + n + 1, n > 3$
$T(3) = 1$
Reordenando términos
$T(n) - T(n-1) = n + 1$
Es no homogénea
Haciendo el cambio $x^{k=1} = T(n), b = 1, d = 1$
Se obtiene la ecuación característica $(x-1)(x-1)(x-1)^2 = 0$
$r1 = 1, r2 = 1, r3 = 1, r4 = 1$
$T(n) = c1(1)^n + c2n(1)^n + c3n^2(1)^n + c4n^2(1)^n$
Por lo tanto la complejidad es $O(n^2)$

Bibliografía: http://www.eafranco.com/docencia/analisisdealgoritmos/files/ejercicios/06.pdf