Análisis de algoritmos

Erick Efrain Vargas Romero

Prof. Franco Martínez Edgardo Adrián

Diseño de soluciones DyV

Divide and conquer 1

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

Límites

- $1 \leq N \leq 10^5$
- $|A_i| \leq 10^9$

Solución

Para este ejercicio se ha decidido no optar por la solución de divide and conquer, sino un algoritmo más eficiente el cual es algoritmo de Kadane, el cual es de complejidad O(n), la cual es mejor que la complejidad que se obtiene con divide y venceras. El algoritmo de Kadane, en una sola iteración nos permite conocer el máximo subarreglo contiguo, pero debemos considerar que el algoritmo original asegura que habrá almenos un número positivo, cosa que para este problema no es así.

Código

```
#include <bits/stdc++.h>
/*******************************************************
 * CASOS RAROS: Que solo nos ingresen números negativos.*
 * *****************************************************/
using namespace std;

typedef long long int lli;

int main(){

    lli n, ans = 0, sum = 0, another_ans = -10e10;
    vector<lli> v;
    cin>>n;
    while(n--){
        lli a;
```

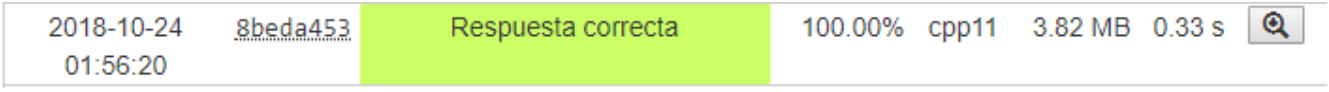
```
        cin>>a;
        v.push_back(a);
    }

    sum = 0;
    //Iteramos en todo el vector
    for(int i = 0; i < v.size(); i++){
        //El algoritmo de kadane nos dice que almenos habrá un número positivo, por tanto, la mejor
respuesta es cero
        //Además el valor de la suma se determina sumando la variable sum más el valor actual del vector,
        //si este valor es mayor que cero (positivo) entonces nuestro candidato a mejor respuesta ahora es
        //la suma entre la variable suma y el vector alctual, en caso contrario ya no hay mejor candidato
        //a suma contigua de valores más que cero, por tanto reiniciamos la variable 0.
        sum = ( (sum + v[i]) > 0 )? sum + v[i] : 0;
        //La respuesta siempre es el máximo, por tanto, buscamos el máximo entre la suma actual y la
respuesta anterior
        ans = max(sum, ans);
        another_ans = max(another_ans, v[i]);
    }
    //Kadane solo funciona si hay números positivos, si hay negativos, la respuesta será el menos negativo
    //Si el algoritmo no encontró un positivo entonces ans tendrá como resultado cero.
    if(ans == 0)
        ans = another_ans;
    cout<<ans<<endl;
}
```

Si analizamos el código que se encuentra en la parte de arriba, solo debemos concentrarnos en el ciclo for que tenemos, ya que es en si el algoritmo de forma general.

```
    for(int i = 0; i < v.size(); i++){
        sum = ( (sum + v[i]) > 0 )? sum + v[i] : 0;
        ans = max(sum, ans);
        another_ans = max(another_ans, v[i]);
    }
```

Si analizamos el siguiente trozo de código podemos ver que vamos a iterar sobre un vector v, en todos sus elementos, es decir v[0, 1, … , n] además, las operaciones que se realizan dentro del ciclo podríamos decir que son a lo mucho 7, por tanto la cota O de este ejercicio será O(n)

Captura



INVCNT – Inversion Count

Descripción

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.

Entrada

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.

Salida

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

Ejemplo

Entrada	Salida
2 3 3 1 2 5 2 3 8 6 1	2 5

Solución

En este ejercicio tenemos una implementación prácticamente idéntica a merge sort. Merge sort es un algoritmo que utiliza divide y vencerás para llegar a una óptima solución en algoritmos de ordenamiento. Lo que realiza es dividir un arreglo de números en mitades, e ir ordenando esas mitades de forma recursiva, hasta llegar al caso base, el cual es que solo se tenga un elemento a ordenar, por si mismo ese arreglo ya está ordenado, por tanto retornamos ese elemento a la llamada a la función anterior, ordenamos los elementos y volvemos a retornar ese subarreglo pero ahora ordenado.

Código

```
#include <iostream>
#include <vector>

using namespace std;

typedef long long int lli;
vector<lli> v(10000000 + 2, 0);
vector<lli> v_s(10000000 + 2, 0);

lli merge(lli begin, lli middle, lli end){
    lli count = 0;
    //We need to move inside the array so, we need to copy the information given
    lli left = begin;
    lli center = middle;
    lli index = begin;
    //We need to sort the array
    while( (left <= middle - 1) && (center <= end) ){
        //It's just a swap
        if(v[left] <= v[center])
            v_s[index++] = v[left++];
        else {
            v_s[index++] = v[center++];
            count += (middle - left);
        }
    }

    while(left <= middle - 1)
        v_s[index++] = v[left++];

    while(center <= end)
        v_s[index++] = v[center++];

    for(lli i = begin; i <= end; i++)
        v[i] = v_s[i];

    return count;
}

lli mergeSort(lli begin, lli end){
    lli middle = (begin + end) / 2;
    lli count = 0;
    if(begin < end){
        count += mergeSort(begin, middle);
        count += mergeSort(middle + 1, end);
        count += merge(begin, middle + 1, end);
    }
    return count;
}

int main(){
    lli t, n;
    cin >> t;
    while(t--){
        cin >> n;
        for(lli i = 0, a = 0; i < n; i++)
            cin >> v[i];
        cout << mergeSort(0, n - 1) << endl;
```

```
    }
    return 0;
}
```

Para resolver este ejercicio el “hack, truco” como sea que se desee llamar está al ordenar los elementos, como ya sabemos merge sort cada sub arreglo que retorna cada recursión estará ordenado por tanto podemos comparar los elementos del subarreglo izquierdo, con los del lado derecho, eso se realiza en el siguiente fragmento de código

```
while( (left <= middle - 1) && (center <= end) ){
        //It's just a swap
        if(v[left] <= v[center])
            v_s[index++] = v[left++];
        else {
            v_s[index++] = v[center++];
            count += (middle - left);
        }
    }
```

Como sabemos los elementos del subarreglo izquierdo y derecho estarán ordenados descendentemente, por tanto, sabemos que si el i-ésimo elemento del arreglo si es mayor que el j-ésimo elemento del sub arreglo derecho entonces los n elementos del sub arreglo izquierdo serán más grandes que el j-ésimo elemento con el que se está haciendo la comparación, por tanto nuestra respuesta sumará la posición inicial del subarreglo derecho, menos la posición en la que se encuentra el subarreglo izquierdo. Finalmente, retornamos la respuesta a la llamada a función que está debajo de nosotros y acumulamos la respuesta.

La complejidad de realizar esto es evidente que es $O(n \log n)$ ya que sabemos que merge tiene esa complejidad, ya que es $\log n$ el dividir el arreglo en mitades y n el ordenar esas mitades.

Captura

22660327	☐	2018-11-07 15:18:12	Inversion Count	**accepted** edit ideone it	0.14	16M	CPP14