INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

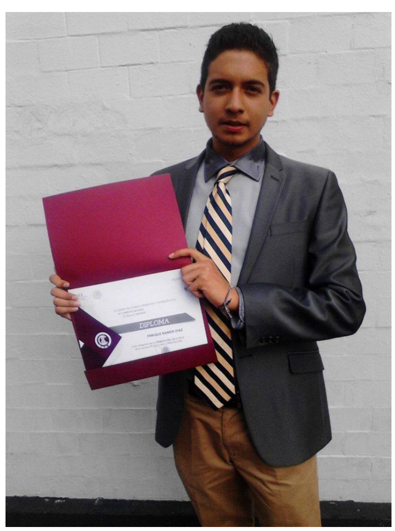
Ejercicio 06: Diseño de soluciones DyV

Unidad de aprendizaje: Análisis de Algoritmos

Grupo: 3CM3

Alumno:
Ramos Diaz Enrique

Profesor(a):
Franco Martínez Edgardo Adrián



5 de noviembre 2018

Índice

1. Divide and conquer 1

1.1. Problema

La tarea es simple, dado un arreglo **A** de números enteros se debe 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**.

1.2. Análisis y solución

El algoritmo que se implemento consiste en establecer dos variables de control: *suma*, que será la suma máxima de cualquier subarreglo contiguo, y *aux*, en donde iremos guardando ”sumas parciales”. Ambas se inicializarán con el primer elemento de arreglo.

Luego, hacer un recorrido lineal en el arreglo, en donde iremos acumulando el valor de *aux* más el valor la posición actual, y al mismo tiempo aplicamos una función **max()** entre esta suma parcial y la posición actual, para ver si realmente nos conviene realizar la suma y guardarla o ignorarla e inicializar de nuevo *aux* con el valor de la posición actual (esto depende de los valores negativos que pudiesen existir).

A su vez, necesitamos ir revisando que si estas sumas parciales en *aux* son mayor a la que tenemos en la variable *suma* (que recordemos, tiene el primer elemento del arreglo) por medio de un **max()** entre ambas, pues puede que la suma máxima sea únicamente el primer elemento.

A esta implementación se le conoce como **algoritmo voraz o greedy**, y su complejidad es lineal O(n).

1.3. Código resultante

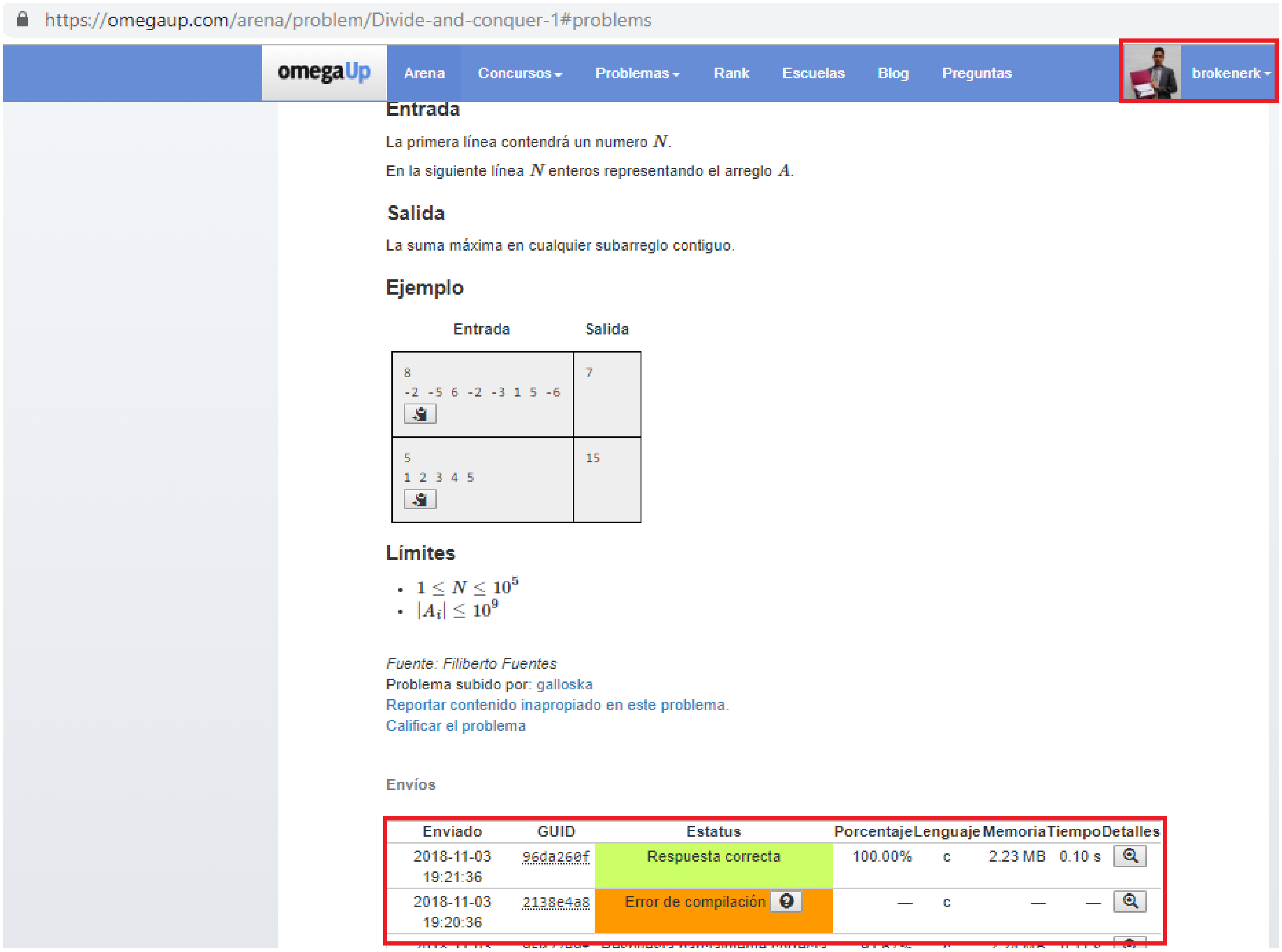
```c
#include <stdio.h>
#include <stdlib.h>

long long int max(long long int a, long long int b){
    if (a < b) return b;
    else return a;
}

int main(){
    int n;
    long long int valor;
    scanf("%d", &n);
    long long int *A = (long long int*)calloc(n,sizeof(long long int));
    for(int i = 0; i < n; i++)
        scanf("%lli", &A[i]);

    //Guardamos el primer elemento, puede que este sea la suma máxima de cualquier subarreglo
    long long int suma = A[0], aux = A[0];
    for(int i = 1; i < n; i++){
        //Vamos verificando si nos conviene realizar la suma parcial o si nos quedamos solo con la posicion actual
        aux = max(aux + A[i], A[i]);
        //Vamos comparando las sumas parciales obtenidas (con el primer elemento o entre las anteriores)
        suma = max(aux, suma);
    }
    printf("%lli", suma);
    return 0;
}
```

1.4. Validación del juez online


https://omegaup.com/arena/problem/Divide-and-conquer-1#problems

omegaUp Arena Concursos Problemas Rank Escuelas Blog Preguntas brokenerk

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

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

Fuente: Filiberto Fuentes

Problema subido por: galloska

Reportar contenido inapropiado en este problema.

Calificar el problema

Envíos

Enviado	GUID	Estatus	Porcentaje	Lenguaje	Memoria	Tiempo	Detalles
2018-11-03 19:21:36	96da260f	Respuesta correcta	100.00%	c	2.23 MB	0.10 s	
2018-11-03 19:20:36	2138e4a8	Error de compilación	—	c	—	—	

2. Amigos y regalos

2.1. Problema

Tienes dos amigos. A ambos quieres regalarles varios números enteros como obsequio. A tu primer amigo quieres regalarle **C1** enteros y a tu segundo amigo quieres regalarle **C2** enteros. No satisfecho con eso, también quieres que todos los regalos sean únicos, lo cual implica que no podrás regalar el mismo entero a ambos de tus amigos.

Además de eso, a tu primer amigo no le gustan los enteros que son divisibles por el número primo **X**. A tu segundo amigo no le gustan los enteros que son divisibles por el número primo **Y**. Por supuesto, tu no le regalaras a tus amigos números que no les gusten.

Tu objetivo es encontrar el mínimo número **V**, de tal modo que puedas dar los regalos a tus amigos utilizando únicamente enteros del conjunto 1, 2, 3, ..., **V**. Por supuesto, tú podrías decidir no regalar algunos enteros de ese conjunto.

Un número entero positivo mayor a 1 es llamado primo si no tiene divisores enteros positivos además del 1 y el mismo.

2.2. Análisis y solución

La implementación con un algoritmo de fuerza bruta consistiría en un ciclo infinito que va validando que cada elemento del conjunto n no sea divisible entre los primos X, Y (podría ser aplicando un módulo), y que posea la cantidad suficiente de elementos, o regalos, necesarios para ambos amigos; cuando encuentre el número que representa a este conjunto, romper el ciclo con un *break* e imprimir la solución, pero la complejidad de este método es lineal O(n).

Sin embargo, el proceso que se uso en el algoritmo de solución no es tan alejada del procedimiento anterior.

En primer lugar, sí necesitamos implementar una búsqueda, siendo la binaria la elegida por su complejidad de O($\log n$). La variable *inf* se inicializa el cero, la variable *sup* será el doble de la cantidad total de regalos necesarios, es decir 2*(C1 + C2).

Luego, enviamos la variable *centro*, que será la "mitad" del rango en el que se encuentra la solución [0, 2*(C1 + C2)] y representa el número de elementos del conjunto $\{1, 2, 3, ..., \text{centro}\}$, a la función **conjunto()**.

En ésta función se realizan dos verificaciones:

1. Encontrar la cantidad de números no divisibles en el conjunto recibido

   Debemos hallar el número de elementos que no son divisibles entre X, el número de elementos que no son divisibles entre Y, y el número de elementos que no son divisibles entre ambos.

   Ahora, calculamos el número de elementos que no son divisibles de forma simultánea entre ambos, realizando la resta:

$$noDivSimul = noDivX + noDivY - noDivAmbos$$

2. Verificar si las cantidades encontradas son suficiente para regalar

   Ya que validamos la condición de no divisibilidad, ahora debemos descartar regalos entre ambos amigos, pues no le podemos regalar el mismo regalo a ambos.

   Para esto, le restamos a C1 los posibles regalos que podríamos regalarle al amigo 2 (aquellos que no son divisibles entre Y), y viceversa.

   Un regalo podría aplicar para ambos amigos, y queremos evitar asignarle los mismos regalos a ambos amigos (cada amigo se lleva regalos distintos).

$$regalosA1 = max(c1 - noDivAmbos + noDivY, 0)$$
$$regalosA2 = max(c2 - noDivAmbos + noDivX, 0)$$

   Aplicamos la función **max()** con cero para evitar números negativos.

Ya sólo nos queda verificar que el total de los posibles regalos para cada amigo no supere la cantidad de regalos disponibles en el conjunto que cumplen las condiciones de no divisibilidad. Sí es el caso, devolvemos un falso e intentamos con otro tamaño de conjunto (de esto se encarga la implementación de la búsqueda binaria).

$$regalosA1 + regalosA2 <= noDivSimul$$

La complejidad de este algoritmo es de O($\log 2[C1 + C2]$).

2.3. Código resultante

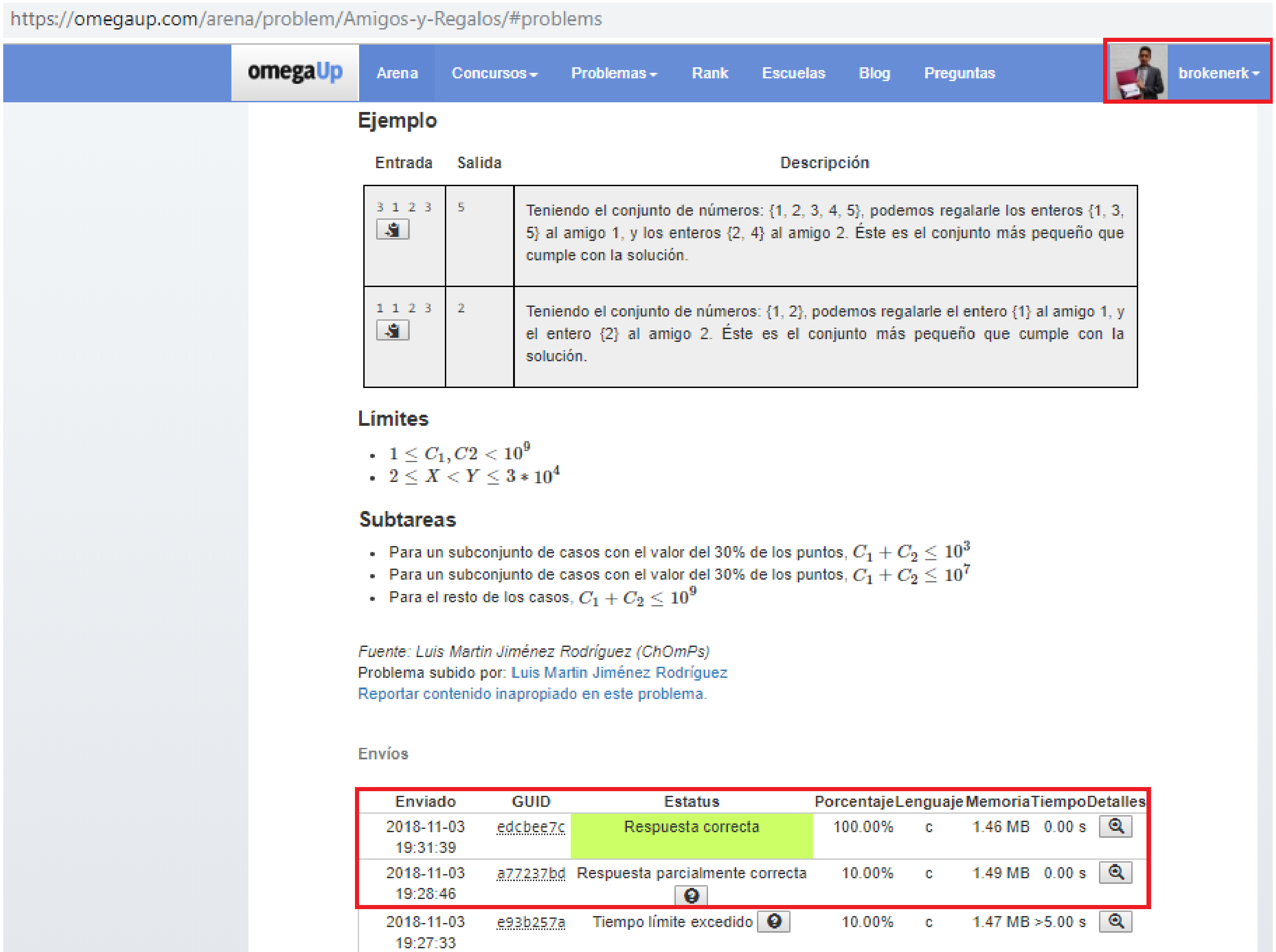
```c
#include <stdio.h>
#include <stdlib.h>

long long int c1, c2, x, y, producto;
//Funcion para descartar valores negativos
long long int max(long long int a, long long int b){
    if (a < b) return b;
    else return a;
}

//Funcion para ver si el conjunto de numeros cumple las condiciones
int conjunto(long long int n){
    //Encontrar la cantidad de numeros no divisibles
    long long int noDivX = n - (n / x);
    long long int noDivY = n - (n / y);
    long long int noDivAmbos = n - (n / producto);
    long long int noDivSimul = noDivX + noDivY - noDivAmbos;
    //Verificar si la cantidad encontrada es suficiente para regalar
    long long int regalosA1 = max(c1 - noDivAmbos + noDivY, 0);
    long long int regalosA2 = max(c2 - noDivAmbos + noDivX, 0);
    //El total de regalos no deben de superar la cantidad disponible de numeros
    if (regalosA1 + regalosA2 <= noDivSimul) return 1;
    else return 0;
}

int main(){
    scanf("%lli %lli %lli %lli", &c1, &c2, &x, &y);
    long long int inf = 0, sup = 2 * (c1 + c2), centro;
    producto = x * y;
    //Busqueda binaria para hallar el conjunto
    while (sup - inf > 1){
        centro = (inf + sup) / 2;
        if(conjunto(centro) == 1)
            sup = centro;
        else
            inf = centro;
    }
    printf("\n%lli", sup);
    return 0;
}
```

2.4. Validación del juez online



https://omegaup.com/arena/problem/Amigos-y-Regalos/#problems

omegaUp Arena Concursos Problemas Rank Escuelas Blog Preguntas brokenerk

Ejemplo

Entrada	Salida	Descripción
3 1 2 3	5	Teniendo el conjunto de números: {1, 2, 3, 4, 5}, podemos regalarle los enteros {1, 3, 5} al amigo 1, y los enteros {2, 4} al amigo 2. Éste es el conjunto más pequeño que cumple con la solución.
1 1 2 3	2	Teniendo el conjunto de números: {1, 2}, podemos regalarle el entero {1} al amigo 1, y el entero {2} al amigo 2. Éste es el conjunto más pequeño que cumple con la solución.

Límites

- $1 \leq C_1, C2 < 10^9$
- $2 \leq X < Y \leq 3 * 10^4$

Subtareas

- Para un subconjunto de casos con el valor del 30% de los puntos, $C_1 + C_2 \leq 10^3$
- Para un subconjunto de casos con el valor del 30% de los puntos, $C_1 + C_2 \leq 10^7$
- Para el resto de los casos, $C_1 + C_2 \leq 10^9$

Fuente: Luis Martin Jiménez Rodríguez (ChOmPs)
Problema subido por: Luis Martin Jiménez Rodríguez
Reportar contenido inapropiado en este problema.

Envíos

Enviado	GUID	Estatus	Porcentaje	Lenguaje	Memoria	Tiempo	Detalles
2018-11-03 19:31:39	edcbee7c	Respuesta correcta	100.00%	c	1.46 MB	0.00 s	
2018-11-03 19:28:46	a77237bd	Respuesta parcialmente correcta	10.00%	c	1.49 MB	0.00 s	
2018-11-03 19:27:33	e93b257a	Tiempo límite excedido	10.00%	c	1.47 MB	>5.00 s	

3. Cúmulo

3.1. Problema

Te encuentras con un mapa del cúmulo de estrellas R136. En el mapa, cada estrella aparece como un punto ubicada en un plano cartesiano. Te asalta de pronto una pregunta, ¿cuál será la distancia mínima entre dos estrellas en el mapa?

3.2. Análisis y solución

Una primera solución podría ser mirar todos los pares de puntos y quedarse con el más pequeño; al haber $\frac{n(n-1)}{2}$ pares de puntos, la complejidad sube hasta O(n^2).

Primero, ordenamos los puntos según la coordenada x con ayuda del algoritmo Shell, el cual tiene una complejidad de O($\log n$).

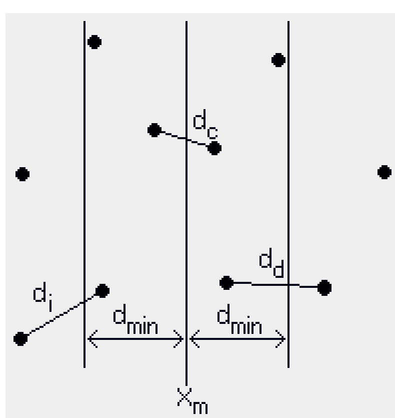
Ahora que se tiene el conjunto ordenado, se puede trazar una línea vertical, $x = x_m$, que divida al conjunto de puntos en dos: P_i (izquierda) y P_d (derecha). Ahora, tenemos 3 casos:

1. El par más cercano está en P_i
2. El par más cercano está en P_d
3. Un punto está en P_i y el otro en P_d

Si los dos estuvieran en P_i o en P_d, la distancia mínima se hallaría recursivamente, subdividiendo más el problema, por lo que ahora el problema se reduce al tercer caso: un punto en cada zona.

Llamemos d_i, d_d y d_c a las mínimas distancias en el primer caso, en el segundo, y en el tercero, respectivamente, y d_{min} al menor de d_i y d_d.

Para resolver el tercer caso, sólo hace falta mirar los puntos cuya coordenada x esté entre $x_m - d_{min}$ y $x_m + d_{min}$. Para grandes conjuntos de puntos distribuidos uniformemente, el número de puntos que caen en esa franja es la raíz de n, así que con una búsqueda exhaustiva cuya complejidad sería de O(n), ya tendríamos el problema resuelto.



La complejidad del algoritmo completo es de O(n log n).

3.3. Código resultante

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
//Estructura de una estrella como punto
typedef struct{
    double x;
    double y;
} PUNTO;
double dmin = 1e10; // Distancia minima (infinito)

//Algoritmo de ordenacion shell
void shell(PUNTO arreglo[], int n){
    int k = trunc(n/2), b = 0, i = 0;
    double tempx = 0, tempy = 0;
    while(k >= 1){
        b = 1;
        while(b != 0){
            b = 0;
            for(i = k;i <= (n-1);i++){
                if(arreglo[i-k].x > arreglo[i].x){
                    tempx = arreglo[i].x;
                    tempy = arreglo[i].y;
                    arreglo[i].x = arreglo[i-k].x;
                    arreglo[i].y = arreglo[i-k].y;
                    arreglo[i-k].x = tempx;
                    arreglo[i-k].y = tempy;
                    b = b + 1;
                }
            }
        }
        k = trunc(k/2);
    }
}

//Función que calcula la distancia entre dos puntos
double distancia(PUNTO a, PUNTO b){
    return (sqrt((a.x - b.x)*(a.x - b.x)+(a.y - b.y)*(a.y - b.y)));
}

void distDyV(PUNTO e[], int num){
    double dist;
    int inicio, fin, k, l;
    //Si no hay pares de puntos, salimos
    if(num <= 1) return;
```

```c
   //Ordenar los puntos segun la coordenada x
   shell(e, num);
   //Buscar en la izquierda recursivamente
   distDyV(e, num/2);
   //Buscar en la derecha recursivamente
   distDyV(e + num/2, (num+1)/2);

   //Hallar los límites del conjunto central
   for(inicio = num/2; inicio > 0 && e[num/2].x - e[inicio].x < dmin; inicio--);
   for(fin = num/2; fin < num-1 && e[fin].x - e[num/2].x < dmin; fin++);

   //Búsqueda exhaustiva en el conjunto central
   for(k = inicio; k <= fin; k++)
      for(l = k + 1; l <= fin; l++)
         if((dist = distancia(e[k], e[l])) < dmin)
            dmin = dist;
}

int main(){
   int n;
   scanf("%d", &n);
   PUNTO pts[n];
   for(int j = 0; j < n; j++)
      scanf("%lf %lf", &pts[j].x, &pts[j].y);
   distDyV(pts, n);
   printf("%.3lf\n", dmin);
   return 0;
}
```

3.4. Validación del juez online

4. Fuentes consultadas de apoyo

[1] (2018) Algoritmo voraz. Accessed november 2018. [Online]. Available: https://es.wikipedia.org/wiki/Algoritmo_voraz

[2] (2018) Divide y vencerás. Accessed november 2018. [Online]. Available: http://www.algoritmia.net/articles.php?id=34