# Archivo:Logo Instituto Politécnico Nacional.png - Wikipedia, la enciclopedia libreArchivo:EscudoESCOM.png - Wikipedia, la enciclopedia libreINSTITUTO POLITÉCNICO NACIONAL

# ESCUELA SUPERIOR DE CÓMPUTO

Vargas Hernández Carlo Ariel

Análisis de algoritmos

3CM15

Ejercicios 12



Índice

[**Problema 01: Minimal coverage** 1](#_Toc106314424)

[Descripción: 1](#_Toc106314425)

[Explicación: 1](#_Toc106314426)

[Complejidad: 2](#_Toc106314427)

[Código: 2](#_Toc106314428)

[Captura: 3](#_Toc106314429)

[**Problema 03: Scarecrow** 4](#_Toc106314430)

[Descripción: 4](#_Toc106314431)

[Explicación: 4](#_Toc106314432)

[Complejidad: 4](#_Toc106314433)

[Código: 5](#_Toc106314434)

[Captura: 6](#_Toc106314435)

[**Problema 09: Organizando Cajas** 6](#_Toc106314436)

[Descripción: 6](#_Toc106314437)

[Explicación: 6](#_Toc106314438)

[Complejidad: 6](#_Toc106314439)

[Código: 7](#_Toc106314440)

[Captura: 7](#_Toc106314441)

**[Problema 07: Andar chido](#_Toc106314442)** [8](#_Toc106314442)

[Descripción: 8](#_Toc106314443)

[Explicación: 9](#_Toc106314444)

[Complejidad: 9](#_Toc106314445)

[Código; 9](#_Toc106314446)

[Captura: 10](#_Toc106314447)

[**Referencias:** 10](#_Toc106314448)

# **Problema 01: Minimal coverage [1]**

## Descripción:

Dados varios segmentos en el eje X con coordenadas (Li, Ri). Escoge la mínima cantidad de segmento tal que cubran el segmento (0, M).

Entrada: La primera línea es el número de casos de prueba, seguida de una línea en blanco. Cada caso de prueba contiene un entero M (1<=M<=5000) seguida por pares “Li, Ri” (|Li|, |Ri| <=50000, i<=100000) cada uno en una línea separada. Cada caso de prueba se separa por un “0 0”.

Salida: Para cada caso de prueba imprime la mínima cantidad de segmentos que cubren (0, M) seguido de los segmentos a usar. De no poderse debe imprimir 0.

## Explicación:

La parte greedy de este problema está en el ordenar los elementos por el punto donde inician. Tomamos un pivote que será 0 y una variable “maximoDerecha” que nos indicará cual es el valor más alto que nos hemos movido hacia la derecha. Hay que tener en cuenta que no importa si los segmentos se solapan.

Iteramos sobre los segmentos ordenados si el inicio del segmento es menor o igual al pivote y su punto final es mayor al valor máximo de derecha al que he llegado, es un posible candidato a ser un segmento que se tome.

Al principio pensaba que se debía ordenar de menor a mayor por el inicio y el final, y que si había más segmentos con el mismo inicio sólo se debía tomar el de mayor final para optimizar y así la solución sería lineal, pero luego me encontré con este caso:

10

0 - 7

0 - 8

1- 9

7 - 10

8 - 9

9 - 10

Usando el método que mencione se tomaría (0 - 8), (8 - 9), (9 - 10) ya que 0-7 y 0-8 tienen el mismo inicio y este último tiene el final más grande, sin embargo la respuesta es (0 - 7), (7 - 10). Con este contra ejemplo me di cuenta que para cada posible segmento que se puede añadir hay que buscar entre los que están al frente de él si hay alguien que empieza donde termine o antes para saber si es lo mejor. Con lo que la solución se vuelve cuadrática. El algoritmo deja de buscar una vez derechaMaxima sea igual o mayor a M, en caso de no serlo quiere decir que no se logró cubrir todo el segmento de interés.

## Complejidad:

Como se mencionó antes, al ser un algoritmo greedy las respuestas que da son posibles candidatos a ser lo mejor pero no lo asegura, por lo que al tener que comparar en el peor de los casos con todos los segmentos se vuelve cuadrático: O(n2).

## Código:

#include <bits/stdc++.h>

using namespace std;

int main(){

  int t;

  int m, x, y;

  int pivote=0, maximoDerecha=0,aux;

  cin >> t;

  while(t--){

    vector<pair<int, int> > segmento, respuestas;

    cin >> m;

    while(cin >> x >> y){

      if(x == 0 && y == 0) break;

      segmento.push\_back(make\_pair(x, y));

    }

    sort(segmento.begin(), segmento.end());

    bool haycambios=true;

    int continua=0;

    pivote=0, maximoDerecha=0;

    while(haycambios){

      haycambios=false;

      maximoDerecha=pivote;

      for(int i=continua;i<segmento.size();i++){

        continua=i;

        if(segmento[i].first <= pivote){

          if(segmento[i].second > maximoDerecha){

            maximoDerecha=segmento[i].second;

            aux=i; haycambios=true;

          }

        }else{

          break;

        }

      }

      pivote=maximoDerecha;

      if(haycambios){

        respuestas.push\_back(segmento[aux]);

      }

      if(maximoDerecha>=m){

        break;

      }

    }

    if(maximoDerecha>=m){

      cout<<respuestas.size()<<endl;

      for(int i=0;i<respuestas.size();i++){

        cout<<respuestas[i].first<<" "<<respuestas[i].second<<endl;

      }

    }else{

      cout<<"0"<<endl;

    }

    if(t>0){

      cout<<endl;

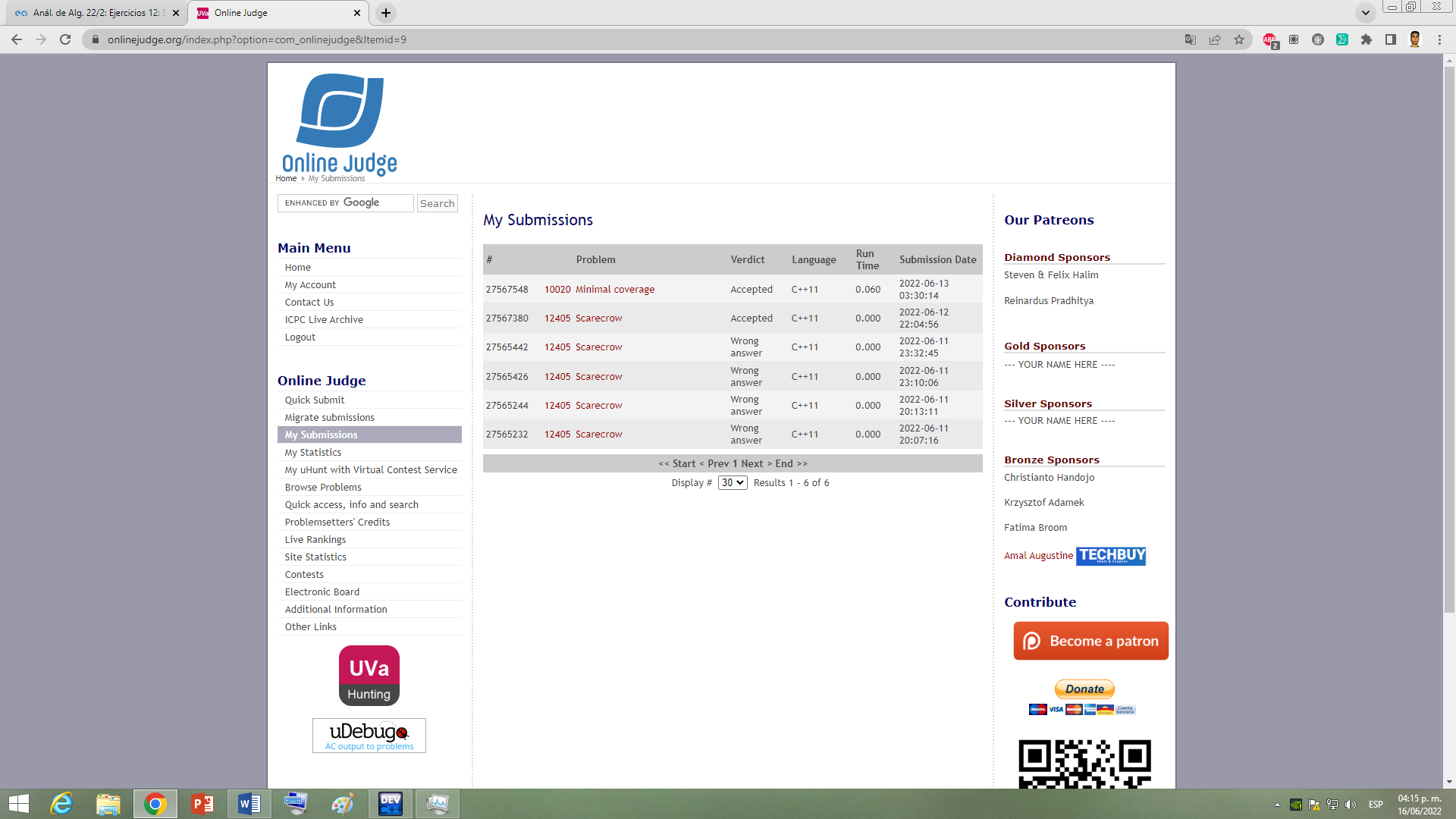
    }

  }

  return 0;

}

## Captura:



# **Problema 03: Scarecrow [2]**

## Descripción:

Dado un campo de 1xN de longitud que tiene partes fértiles e infértiles representados por “. “ y “#” respectivamente, se pide que se encuentre el número mínimo de espantapájaros necesarios para cubrir todos los puntos fértiles teniendo en cuenta que un espantapájaros puede cubrir el lugar donde está, su izquierda y derecha.

Entrada: Primero se da un numero T<=100 que es el número de casos de prueba.

Cada caso tiene un entero N (N<0<100). Después viene una línea que contiene “.” si ese punto es fértil y “#” si no lo es.

Salida: La palabra “Case i:” donde i es el número de caso, un espacio y la cantidad mínima de espantapájaros para cubrir la tierra fértil.

## Explicación:

Para la explicación de cómo resolví este greedy hay que tener en cuenta que el espantapájaros cubre donde está, su izquierda y derecha, y que no tiene sentido que un espantapájaros cubra una zona no fértil, por lo que hay que evitarlo a ser posible.

La forma de resolverlo que tome fue avanzar en el arreglo del terreno empezando desde el 1 (suponiendo que se indexa desde 0) si en la izquierda es fértil (si hay un punto) ahí debe haber un espantapájaros independientemente de si en donde se pondría el espantapájaros o enfrente de él es fértil o no. Como se cubren 3 puntos por espantapájaros nos movemos 3 puntos hacia adelante para no solapar los rangos del siguiente espantapájaros y el anterior (así aprovechamos al máximo el área cubierta), con lo que nos moveríamos a la posición 4. En caso de que haya un punto infértil (un #) nos movemos 1 hacia la derecha hasta que encontremos el primer caso mencionado.

Estas dos condiciones se repiten siempre y cuando el tamaño del arreglo que falta por explorar sea mayor a 3 ya que así se asegura que no habrá ningún problema al momento de avanzar 3 posiciones de lo contrario tenemos que ver cuantas zonas fértiles hay en ese tramo, si hay al menos 1 debe llevar un espantapájaros más.

## Complejidad:

Al solo iterar una vez por el arreglo y hacer operaciones básicas que son constantes esto se resuelve entiempo lineal O(n).

## Código:

#include<bits/stdc++.h>

using namespace std;

int main (){

  int i=0,a,b,scrw=0;

  int fertil,nofertil;

  string v;

  char c;

  vector <int> w;

  cin>>b;

  while(b--){

    v="";

    scrw=0;

    cin>>a;

    for(i=0;i<a;i++)

    {

      cin >> c;

            v += c;

    }

    for(i=1;i<=a;i++){

      if(v[i]=='.'){

        fertil++;

      }

      if(v[i-1]=='.'){

        if(i+3<=a || i+1==a || i==a || i==a-2){

          scrw++;i=i+2;

        }

      }

    }

    w.push\_back(scrw);

    scrw=0;

  }

  for(i=0;i<w.size();i++){

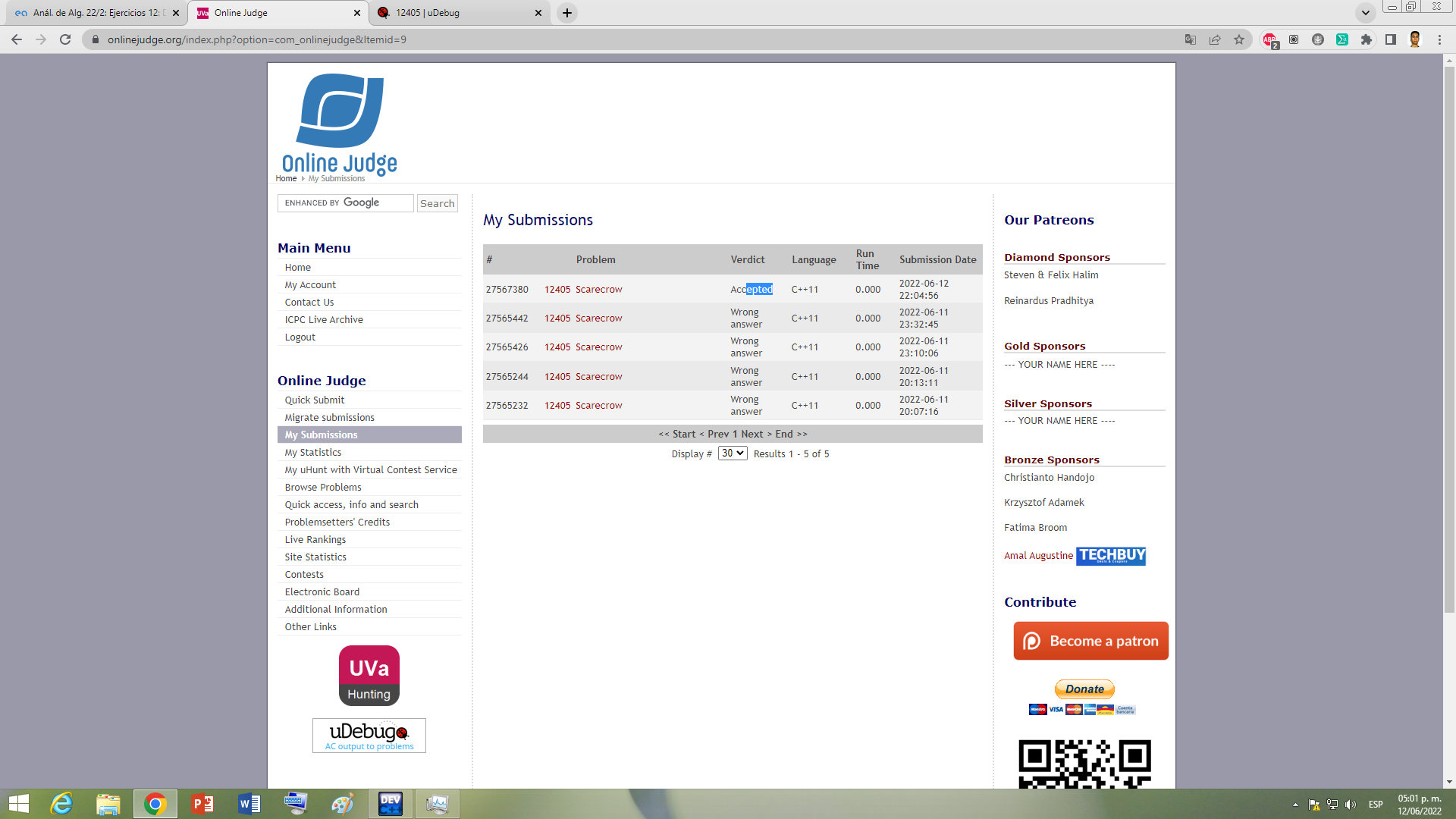
    cout<<"Case "<<(i+1)<<": "<<w[i]<<endl;

  }

  return 0;

}

## Captura:



# **Problema 09: Organizando Cajas [4]**

## Descripción:

Fito ha comenzado a trabajar en el almacén de un hotel cerca de su casa. La primera tarea que le dan es ordenar todas las cajas de una sección del almacén. Como es su primer día de trabajo, Fito quiere cumplir esta orden de la mejor forma posible. Para esto decide que la manera adecuada de ubicar las cajas es ponerlas todas las cajas formando una sola columna, logrando así mayor espacio en el almacén. De cada caja se sabe su peso y su fuerza. El riesgo de que una caja se rompa se calcula como el peso total de las cajas encima de ella menos la fuerza de la caja. Como Fito sabe que si se rompe una caja la tiene que pagar él entonces necesita ordenar las cajas de forma tal que el mayor riesgo de que se rompa alguna de ellas sea mínimo.

Entrada: La primera línea de la entrada contiene un entero N (1≤N≤50000) la cantidad de cajas que Fito debe ordenar. Después sigue N líneas cada una con un par de enteros separados por un espacio: Wi (1≤Wi≤10000) y Si (1≤Si≤109) el peso y la fuerza de la caja i respectivamente.

Salida: La salida consiste en un entero: el mayor riesgo en el orden de Fito.

## Explicación:

Se quiere minimizar la suma del peso que recibe una caja en un determinado punto, por lo que la parte greedy sería ordenar de menor a mayor peso las cajas y ver cuál es la caja cuyo riesgo sea máximo y esa sería la respuesta. No funcionaria ordenar por fuerza de la caja (al menos no como primer parámetro de ordenamiento) ya que lo que se busca minimizar es el riesgo de que la caja se rompa y eso se logra poniéndole el menor peso posible, sin embargo si hay dos cajas con el mismo peso primero irá la de menos fuerza.

## Complejidad:

Recorrer el arreglo ya ordenado es cota O(n) y sumar, restar y comparar son operaciones básicas de orden constante, sin embargo, ordenar es lo que tiene la mayor complejidad, por lo que la complejidad del algoritmo seria O(nlog(n)).

## Código:

#include <bits/stdc++.h>

using namespace std;

int main(){

  int n,w,s,pesoacumulado=0,riesgomaximo=0;

  vector <pair<int,int> >cajas;

  cin>>n;

  while(n--){

    cin>>w>>s;

    cajas.push\_back(make\_pair(w,s));

  }

  sort(cajas.begin(),cajas.end());

  for(int i=0;i<cajas.size()-1;i++){

    pesoacumulado+=cajas[i].first;

    if(riesgomaximo<pesoacumulado-cajas[i+1].second){

      riesgomaximo=pesoacumulado-cajas[i+1].second;

    }

  }

  cout<<riesgomaximo;

  return 0;

}

## Captura:

La página de MOG no me dejo iniciar sesión, así que plantearé algunos casos de prueba para mi solución.

3 3 3

10 3 1 100 100 2

2 5 1 5 100 1

3 3 2 1 6 100

Para el primer caso (el caso de ejemplo) la salida del programa es 2, que es el esperado.

Para el segundo caso, donde la caja con menor peso también tiene la mayor resistencia al ordenar queda así las cajas:

1 100 En la primer iteración se da que 1-5=-4. Como 0>-4 no se hace ningún cambio.

1 5 En la segunda iteración tenemos que 2-1=1>0 con lo que se actualiza el valor de

2 1 riesgo máximo. La última caja no se cuenta.

Si lo ordenáramos por fuerza de la caja quedaría así:

2 1 En la primer iteración se tiene 2-5=3>0 con lo que actualizamos el riesgo.

1 5 En la segunda iteración tenemos que 3-100=-99<3 con lo que no actualizamos.

1 100 Al final el valor seria 3 que es mayor que el 1 que conseguimos anteriormente.

Para el tercer caso haremos lo mismo que en el segundo para comparar, primero al ordenar por peso:

6 100 En la primer iteración tenemos que 6-1=5>0 por lo que el riesgo se actualiza.

100 1 En la segunda 106-2=104>5 con lo que se actualiza.

100 2 Al final el riesgo máximo es 104.

Ordenando por fuerza:

100 1 En la primer iteración tendríamos un riesgo de 100-2=98.

100 2 En la segunda el riesgo seria 200-100=100.

6 100 Al final el riesgo sería 100

En este caso ya no se cumple. Lo que se me ocurre es que se resolviera de las dos formas y se escogiera el riesgo menor entre ambas formas ya que no se me ocurre un caso que pueda romper eso.

# **Problema 07: Andar chido [3]**

## Descripción:

Ahora el Dr. Ethan Freud tiene N clientes en su oficina que le pagarán por llevarlos a sus casas. Para cada cliente i, le toma Ti minutos llegar desde su oficina hasta la casa del cliente. Sin embargo, sólo puede llevar a un cliente a la vez, por lo que debe dejar a los demás esperando. El problema es que como están un poco "altos en azúcar", cada cliente j que se quedó destruye Dj de sus preciados libros cada minuto. Entonces, mientras le toma 2Ti minutos dar la vuelta en llevar al cliente i, los clientes destruyeron entre todos 2TiS libros, donde S es la suma de las Dj de los clientes esperando.

Como el Dr. Ethan Freud no quiere que rompan muchos de sus libros te ha pedido que le ayudes a saber la menor cantidad de libros destruidos posibles dada la información de los clientes de esta noche.

Entrada: En la primera línea, un entero: N. En las siguientes líneas, dos enteros: Ti y Di del i-ésimo cliente.

Salida: La mínima cantidad de libros destruidos después de llevar a todos los clientes a sus casas.

## Explicación:

Según yo la parte greedy de este algoritmo es que quieres sacar tan pronto como sea posible a los más destructivos y que además te cueste el mínimo tiempo posible para regresar y que no destruyan más, con lo que se deberían ordenar las entradas de menor a mayor tiempo y de mayor a menor cantidad de libros que destruyan. Pero no funciono, debe ser que hay un caso en el que el mayor destructor de libros tiene también el mayor tiempo y entonces para cuando llega su turno ya sube demasiado la cantidad de libros destruidos.

## Complejidad:

Lineal lo que propuse O(n). La real no se cual sea pero por la cota 2<=N<=10e5 y el tiempo de 1s por caso no creo que llegue a O(n2).

## Código;

#include<bits/stdc++.h>

using namespace std;

bool sortbyCond(const pair<int, int>& a, const pair<int, int>& b){

    if (a.first != b.first)

        return (a.first < b.first);

    else

       return (a.second > b.second);

}

int main (){

  int n,t,d,s=0;

  int librosdestruidos=0;

  vector<pair<int, int> > clientes;

  cin>>n;

  while(n--){

    cin>>t>>d;

    s+=d;

    clientes.push\_back(make\_pair(t,d));

  }

  sort(clientes.begin(),clientes.end(),sortbyCond);

  for (int i = 0; i < clientes.size(); i++) {

    s-=clientes[i].second;

    librosdestruidos+=2\*clientes[i].first\*s;

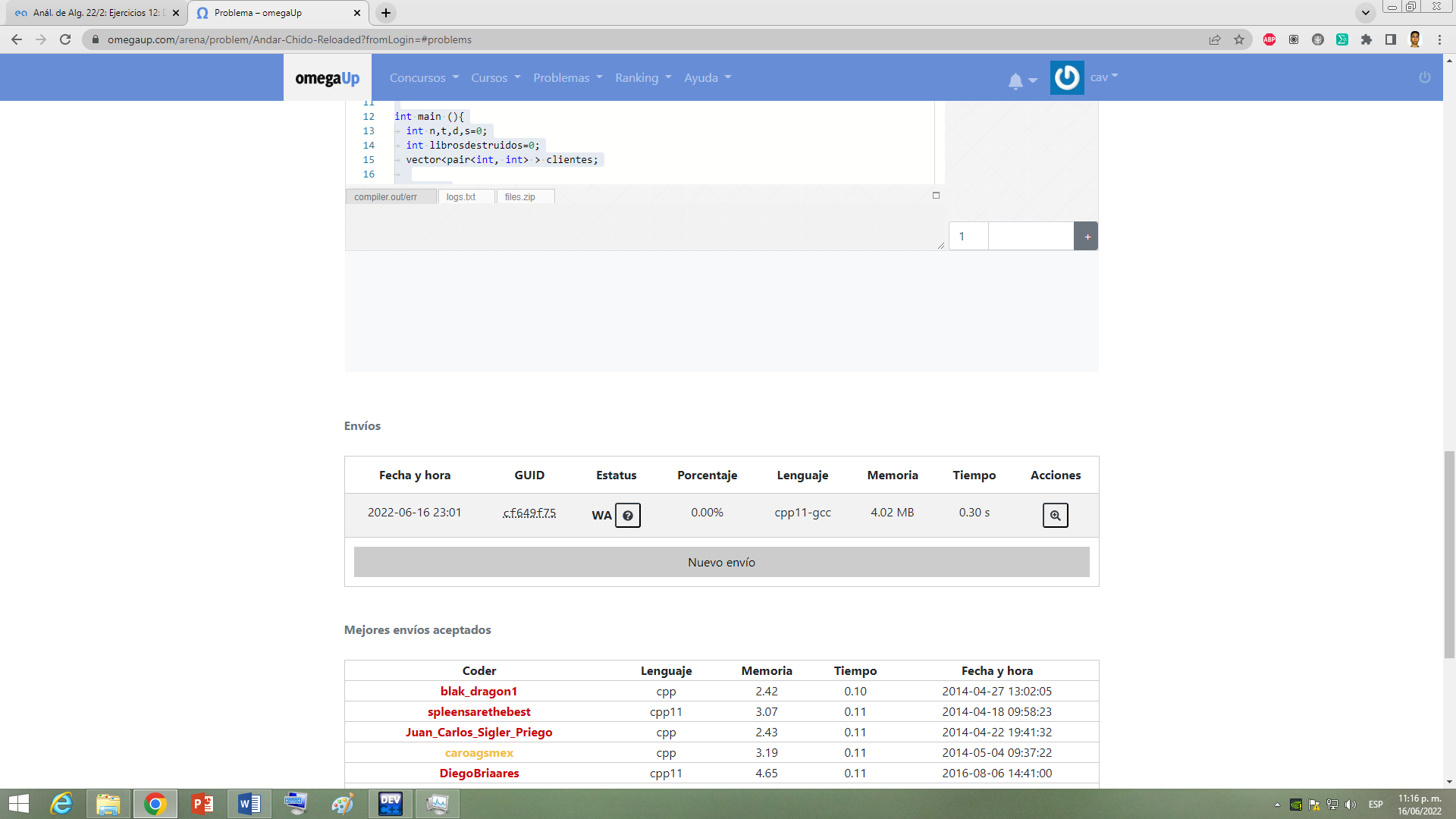
    }

    cout<<librosdestruidos;

}

## Captura:

Hay pero no de aceptado.



# **Referencias:**

[1] UVa. Minimal coverage. <https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=961>

[2] UVa. Scarecrow. <https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=657&page=show_problem&problem=3836>

[3] R. Freddy, (2014, Abr 18). Andar Chido Reloaded. <https://omegaup.com/arena/problem/Andar-Chido-Reloaded#problems>

[4] Matcom Online Grader. Organizando Cajas. <https://matcomgrader.com/problem/9236/organizando-cajas/>