INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

ANALISIS DE ALGORITMOS

PROFESOR TITULAR: FRANCO MARTINEZ EDGARDO ADRIAN

EJERCICIOS #12:
DISEÑO DE SOLUCIONES MEDIANTE PROGRAMACIÓN VORAZ (GREEDY)

LEMUS RUIZ MARIANA ELIZABETH
2020630211

GRUPO: 3CM12

LEMUS RUIZ MARIANA ELIZABETH
3CM12

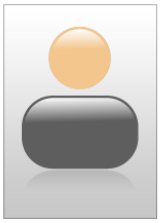
EJERCICIOS 12: DISEÑO DE SOLUCIONES MEDIANTE PROGRAMACION VORAZ (GREEDY)

INSTRUCCIONES:
De los siguientes 10 problemas que se plantean resolver al menos 4 problemas para completar el ejercicio.

EJERCICIO 01: 10020 MINIMAL COVERAGE

Usuario:

Página de perfil de lemusmarianar



Captura de aceptación:

27059376	10020 Cobertura mínima		Aceptado	C++11	0.110	2021-12-15 22:12:41

<< Inicio < 1 anterior >> final del próximo >

Monitor # 30 Resultados 1 - 8 de 8

Explicación del problema

Descripción

Dados varios segmentos de línea (en el eje X) con coordenadas [Li, Ri]. Debes elegir el mínimo cantidad de ellos, de modo que cubrirían completamente 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 en la entrada debe contener un número entero M (1 ≤ M ≤ 5000), seguido de pares "Li Ri" (| Li|, | Ri| ≤ 50000, i ≤ 100000), cada uno en una línea separada. Cada caso de prueba de entrada se termina por par"0 0".

Cada caso de prueba estará separado por una sola línea.

Salida

Para cada caso de prueba, en la primera línea de salida, su programa debe imprimir el número mínimo de líneas segmentos que pueden cubrir el segmento [0, M]. En las siguientes líneas, las coordenadas de los segmentos, ordenados por su

extremo izquierdo (Li), deben imprimirse en el mismo formato que en la entrada. El par "0 0" no debe ser impreso. Si [0, M] no se puede cubrir con segmentos de línea dados, su programa debe imprimir "0" (sincitas).

Imprima una línea en blanco entre las salidas para dos casos de prueba consecutivos.

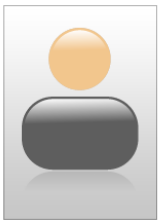
Código:

```
#include <iostream>
#include <vector>
#include <utility>
#include <algorithm>
using namespace std;

int main(void)
{
    int cases;
    cin >> cases;
    while(cases--)
    {
        int M, izq, der, auxI, auxD;
        cin >> M;
        vector <pair<int, int> > izqDers;
        bool encontrado = false, neg = false;
        while(cin >> izq >> der && (izq != 0 || der != 0))
            izqDers.push_back(make_pair(izq,der));
        encontrado = false;
        vector <pair<int, int> > pedazos;
        sort(izqDers.begin(), izqDers.end());
        /*
        Se declara un pivote que servirá para saber cual es la posicion
última que se
        quiere cubrir. (pivoteIzq)
        Se declara un índice que hace referencia a un segmento del
arreglo que cumple
        con la condición y que posteriormente se va a agreagar al
arreglo de resultados (agregarPos)
        Se declara un indice que corresponde a la posicion derecha del
segmento (limDer)
        */
        int pivoteIzq = 0, agregarPos, limDer;
        bool agregarValores = true;
        int i = 0;
        /*
        Código que se ejecuta mientras haya un segmento que se pueda
agregar
        al arreglo de las soluciones.
        */
        while(agregarValores)
        {
            limDer = pivoteIzq;
            agregarValores = false;
            //Se busca el último segmento que sea menor o igual que el
pivote izquierda
            while(i < izqDers.size() && izqDers[i].first <= pivoteIzq)
```

```
            {
                /*
                Se pregunta si el valor derecho del segmento actual es
mayor que
                el limite derecho actual lo cual significa que se va
extendiendo la cobertura total,
                acecándose a cubrir de 0 a M
                */
                if(izqDers[i].second > limDer)
                {
                    agregarPos = i;
                    limDer = izqDers[i].second;
                    agregarValores = true;
                }
                i++;
            }
            /*
            Se actualiza el pivote izquierdo, que se convierte en el
limite derecho
            esto para ir recorriendo los limites e ir buscando los
segmentos que cumplan
            */
            pivoteIzq = limDer;
            if(agregarValores)
                pedazos.push_back(izqDers[agregarPos]);
            /*
            Si el limite derecho es mayor que la M significa que ya se
encontraron los
            segementos que cubren de 0 a M
            */
            if(limDer >= M)
            {
                encontrado = true;
                agregarValores = false;
                break;
            }
        }
        if(!encontrado)
            cout << "0\n";
        else
        {
            //Se imprimen los valores en el arreglo de resultados
            cout << pedazos.size() << "\n";
            sort(pedazos.begin(), pedazos.end());
            for(auto a : pedazos)
                cout << a.first << " " << a.second << "\n";
        }
        if(cases > 0)
            cout << "\n";
    }
}
```

EJERCICIO 02: 10382 Watering Grass
Usuario:

Página de perfil de lemusmarianar



Captura de aceptación:

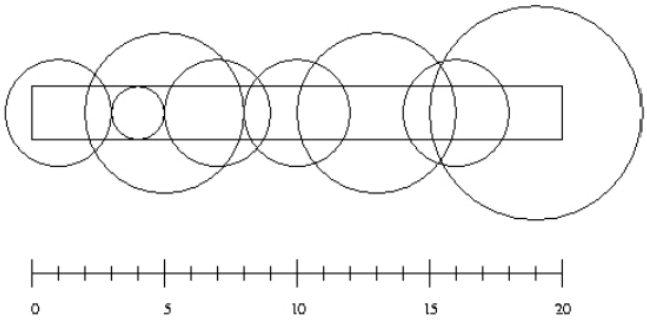
#	Problema	Veredicto	Idioma	Tiempo de ejecución	Fecha de envío
27059429	10382 Riego de hierba	Aceptado	C++11	0.000	2021-12-15 22:41:16

Explicación del problema

Descripción

Los n aspersores se instalan en una franja horizontal de césped de 1 metro de largo y w de ancho. Cada aspersor se instala en la línea central horizontal de la tira. Para cada aspersor se nos da su posición como distancia desde el extremo izquierdo de la línea central y su radio de operación.

¿Cuál es el número mínimo de aspersores que se deben encender para regar toda la franja de césped?



Entrada

La entrada consta de varios casos. La primera línea para cada caso contiene números enteros n, l y w con n ≤ 10000. Las siguientes n líneas contienen dos números enteros que dan la posición de un aspersor y su radio de operación. (La imagen de arriba ilustra el primer caso de la entrada de muestra).

Salida

Para cada salida de caso de prueba, el número mínimo de aspersores necesarios para regar toda la franja de césped.

Si es imposible regar toda la tira, la salida es "-1".

Código

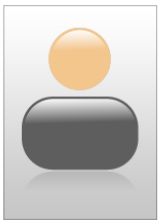
```
#include<iostream>
#include<vector>
#include<cmath>
using namespace std;

int main()
{
	long int num_sprinklers, distance_to_cover, width;
	int flag, flag_modify, complete;

	while(cin >> num_sprinklers >> distance_to_cover >> width)
	{
		vector<pair<double, double> > ranges;
		int minimum_sprinklers = 0;

		pair<int, int> sprinkler;
		pair<double, double> generate_range, aux2;
		double distance_range, actual_mark = 0.0, best_range_taken
= 0.0;

		for(long int i = 0; i < num_sprinklers; i++)
		{
			cin >> sprinkler.first >> sprinkler.second;

			if((sprinkler.second * 2) > width)
			{
				distance_range = (sqrt((4 *
pow(sprinkler.second, 2)) - (pow(width, 2)))) / (2);
				generate_range.first = sprinkler.first -
distance_range;
				generate_range.second = sprinkler.first +
distance_range;

				ranges.push_back(generate_range);
			}

		}

		//-------------------------------------------------------

		while(actual_mark < distance_to_cover)
		{

			flag = 0;
			complete = 1;
			flag_modify = 0;

			for(int j = 0; j < ranges.size(); j++)
			{

				if(ranges[j].first <= actual_mark)
				{
					if(ranges[j].second > actual_mark)
```

```
                                    {
                                          flag = 1;
                                          if((ranges[j].second - actual_mark)
> best_range_taken)
                                          {

                                                aux2.first = ranges[j].first;
                                                aux2.second =
ranges[j].second;

                                                best_range_taken =
ranges[j].second - actual_mark;
                                                flag_modify = 1;
                                          }
                                    }

                              }
                        }

                        if(flag == 0)
                        {
                              cout << "-1" << endl;
                              complete = 0;
                              break;
                        }

                        if(flag_modify == 1)
                        {
                              minimum_sprinklers++;
                              actual_mark = actual_mark + best_range_taken;
                              best_range_taken = 0.0;
                        }

                  }

                  if(complete == 1)
                        cout << minimum_sprinklers << endl;
            }


            return 0;
}
```

EJERCICIO 03: 12405 SCARECROW

Usuario:

Página de perfil de lemusmarianar



Captura de aceptación:

#	Problema	Veredicto	Idioma	Tiempo de ejecución	Fecha de envío
27059429	10382 Riego de hierba	Aceptado	C++11	0.000	2021-12-15 22:41:16
27059422	12405 Espantapájaros	Aceptado	C++11	0.000	2021-12-15 22:37:04

Explicación del problema

Descripción

Taso posee un campo muy extenso. Él planea cultivar diferentes tipos de cultivos en la próxima temporada de crecimiento. El área, sin embargo, está llena de cuervos y Taso temen que puedan alimentarse de la mayoría de los cultivos. Por este motivo, ha decidido colocar unos espantapájaros en diferentes ubicaciones del campo.

El campo se puede modelar como una cuadrícula 1 × N. Algunas partes del campo son infértiles y eso significa que no puede cultivar ningún cultivo en ellos. A espantapájaros, cuando se coloca en un lugar, cubre la celda hasta su izquierda y derecha junto con la celda en la que se encuentra.

Dada la descripción del campo, ¿cuál es el número mínimo de espantapájaros que hay que colocar para que toda la sección útil del campo está cubierta? La sección útil se refiere a las celdas donde los cultivos pueden estar creciendo.

Entrada

La entrada comienza con un número entero T (≤ 100), que denota el número de casos de prueba. Cada caso comienza con una línea que contiene un número entero N (0 <N <100). La siguiente línea contiene N personajes que describen el campo. Un punto (.) Indica un lugar de cultivo y un hash (#) indica una región infértil.

Salida

Para cada caso, envíe el número de caso primero seguido por el número de espantapájaros que deben ser introducido.

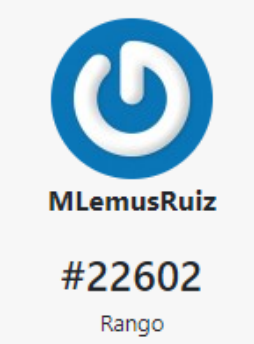
Código

```
#include <iostream>
#include <string>
using namespace std;

int main(void)
{
    int cases, casosCont = 0;
    cin >> cases;
    while(cases--)
    {
        int tamanoC, puntosConsec = 0, hashConsec = 0, puntosTotal = 0,
espantaPaj = 0, i;
        char posicion;
        string crop;
        cin >> tamanoC;
        cin >> crop;
        i = 0;
        while(i < crop.length())
        {
            if(crop[i] == '.')
            {
                espantaPaj++;
                i += 3;
            }
            else if(crop[i] == '#')
                i++;
        }
        cout << "Case " << ++casosCont << ": " << espantaPaj << "\n";
    }
}
```

LEMUS RUIZ MARIANA ELIZABETH
3CM12

EJERCICIO 04: FUNCION HORRIBLE

Usuario:



Captura de aceptación:

Enviado	GUID	Estatus	Porcentaje	Lenguaje	Memoria	Tiempo	Acciones
2021-12-15 16:45:16	49d2eee6	AC	100.00%	cpp17-gcc	7.68 MB	1.70 s	

Explicación del problema

Descripción

Dada la descripción de la función g(x) y el conjunto de x que puedes escoger, determina cuales tienes que tomar tal que se maximice el valor Z.

Entrada

En la primera línea 3 enteros N, M, K indicando el número de elementos de la función, el número de X que tienes y el número de x que debes tomar.

En las siguientes N líneas se describe la función g. Para el i-ésimo elemento de la función, se da un entero para expresar de que tipo es (1 o 2) y los valores A y B del elemento $f_i(x)$.

En la siguiente fila M enteros que x_1, x_2, ..., x_M expresan el conjunto de x con el que cuentas.

Salida

Una línea con K enteros ordenados p_1, p_2, …, p_k indicando que tomaste x_{p1}, x_{p2}, ..., x_{pk}. Si existe más de una solución, cualquiera se tomará como válida.

Código

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
bool comparacion(pair <long long int, long long int> par1, pair <long
long int, long long int> par2)
```

```
{
  return par1.first > par2.first;
}

bool comparacionFinal(pair <long long int, long long int> par1, pair
<long long int, long long int> par2)
{
  return par1.second < par2.second;
}
int main(void)
{
  ios::sync_with_stdio(false);
  cin.tie(0);
  cout.tie(0);
  long long int N, M, K, tipoAux, auxA, auxB, auxI;
  cin >> N >> M >> K;
  for(int i = 0 ; i < N ; i++)
    cin >> tipoAux >> auxA >> auxB;
  vector <pair<long long int, long long int> > numeros;
  vector <pair<long long int, long long int> > resultado;
  for(int i = 0 ; i < M ; i++)
  {
    cin >> auxI;
    numeros.push_back(make_pair(auxI, i + 1));
  }
  int contador = 0;
  sort(numeros.begin(), numeros.end(), comparacion);
  for(auto a : numeros)
  {
    resultado.push_back(a);
    contador++;
    if(contador == K)
      break;
  }
  sort(resultado.begin(), resultado.end(), comparacionFinal);
  for(auto a : resultado)
    cout << a.second << " ";
  cout << "\n";
  return 0;
}
```