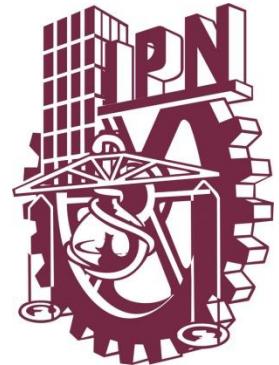


ESCOM

Instituto Politecnico Nacional



ESCOM “ESCUELA SUPERIOR DE CÓMPUTO”

ANÁLISIS DE ALGORITMOS

DISEÑO DE SOLICIONES CON PROGRAMACIÓN VORAZ

PROFE: Edgardo Adrián Franco Martínez

ALUMMNO: Rojas Alvarado Luis Enrique



GRUPO: 3CM4

I. MINIMAL COVERAGE

Mis envíos

#	Problema	Veredicto	Idioma	Tiempo de ejecución	Día de entrega
23408531	10020 Cobertura mínima	Aceptado	C ++ 11	0.060	2019-05-28 13:21:01

Given several segments of line (int the X axis) with coordinates $[L_i, R_i]$. You are to choose the minimal amount of them, such they would completely cover the segment $[0, M]$.

Input

The first line is the number of test cases, followed by a blank line.

Each test case in the input should contains an integer M ($1 \leq M \leq 5000$), followed by pairs “ $L_i R_i$ ” ($|L_i|, |R_i| \leq 50000$, $i \leq 100000$), each on a separate line. Each test case of input is terminated by pair ‘0 0’.

Each test case will be separated by a single line.

Output

For each test case, in the first line of output your programm should print the minimal number of line segments which can cover segment $[0, M]$. In the following lines, the coordinates of segments, sorted by their left end (L_i), should be printed in the same format as in the input. Pair ‘0 0’ should not be printed. If $[0, M]$ can not be covered by given line segments, your programm should print ‘0’ (without quotes).

Print a blank line between the outputs for two consecutive test cases.

Para solucionar éste problema se hizo la prueba de que un conjunto de intervalos que cubren en los cuales uno de ellos es parte de la solución más óptima. Entonces si lo remplazamos con un intervalo de máximos puntos del conjunto cuyo punto llega más a la derecha, entonces el intervalo restante no cubierto será un subconjunto del intervalo restante de la solución óptima. Por lo que simplemente se selecciona iterativamente el intervalo que se encuentra más a la derecha (y que cubra el final del intervalo anterior), repita hasta que toque el punto más a la derecha. Por lo que se almacenaron en una estructura para después poderla recorrer todo el conjunto de intervalos para poder imprimir el inicio y el final.

El orden del complejidad que pude sacar respecto a este algoritmo es de $O(n^3)$. Puesto que todo está contenido en un while que va en retroceso, esto lo hace de orden n y dentro hay 3 ciclos en los cuales el primero es de orden n y los otros 2 se conjuntan para hacer de n^2 .

```

1 #include <iostream>
2 #include <vector>
3 #include <algorithm>
4
5 using namespace std;
6
7 struct par
8 {
9     int empezar, terminar;
10
11    bool operator<(const par & otro) const
12    {
13        return empezar < otro.empezar;
14    }
15};
16
17 par todaspares[100005];
18 par solution[100005];
19
20 int main()
21 {
22     string sep = "";
23     int T, M;
24
25     cin >> T;
26
27     while (T--)
28     {
29         cout << sep;
30         sep = "\n";
31
32         cin >> M;
33
34         int numpars = 0;
35         while (cin >> todaspares[numpars].empezar >> todaspares[numpars].terminar,
36                todaspares[numpars].empezar || todaspares[numpars].terminar)
37         {
38             if (todaspares[numpars].terminar > 0 && todaspares[numpars].empezar < M)
39                 ++numpars;
40
41         sort(todaspares, todaspares + numpars);
42
43         todaspares[numpars].empezar = M + 1;
44
45         int cont = 0;
46         int izq = 0;
47         int der = 0;
48
49         while (izq < M && der < numpars)
50         {
51             solution[cont].terminar = 0;
52             for (; todaspares[der].empezar <= izq; ++der)
53                 if (todaspares[der].terminar > solution[cont].terminar)
54                     solution[cont] = todaspares[der];
55
56             if (solution[cont].terminar == izq)
57             {
58                 break;
59             }
60             izq = solution[cont].terminar;
61
62         cout << solution[cont].empezar;
63
64         for (int i = 0; i < solution[cont].terminar - solution[cont].empezar; ++i)
65             cout << " ";
66
67         cout << endl;
68     }
69
70     cout << sep;
71 }

```

```

61      ++cont;
62    }
63
64    if (izq >= M)
65    {
66      cout << cont << '\n';
67
68      for (int i = 0; i < cont; ++i)
69      {
70        cout << solution[i].empezar << ' ' << solution[i].terminar << '\n';
71      }
72    }
73    else
74      cout << "0\n";
75  }
76

```

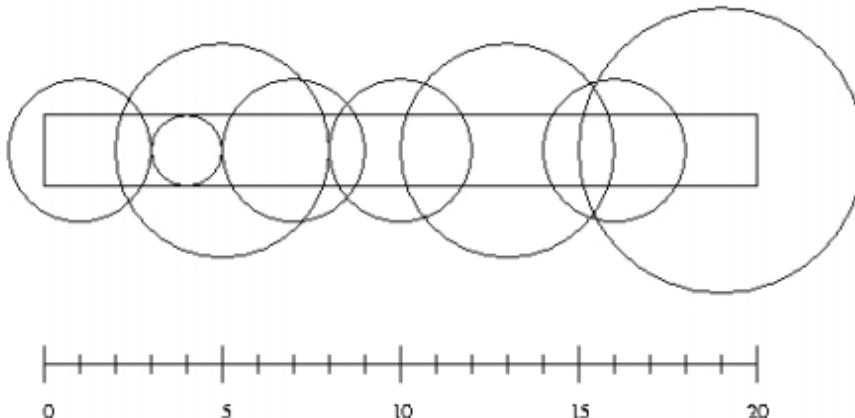
II. WATERING GRASS

Mis envíos

#	Problema	Veredicto	Idioma	Tiempo de ejecución	Día de entrega
23408631	10382 Hierba de riego	Aceptado	C ++ 11	0.000	2019-05-28 13:36:49

n sprinklers are installed in a horizontal strip of grass l meters long and w meters wide. Each sprinkler is installed at the horizontal center line of the strip. For each sprinkler we are given its position as the distance from the left end of the center line and its radius of operation.

What is the minimum number of sprinklers to turn on in order to water the entire strip of grass?



Lo que hice para este problema es, primero convertir las áreas en intervalos. Si el diámetro de un rociador es menor que el ancho del césped, el rociador no puede cubrir el césped. Así que lo eliminamos. Luego, convertimos los rociadores restantes en intervalos. Y como el problema anterior es de cobertura de intervalos en el que podemos almacenar el mínimo izquierdo y el máximo derecho cubierto por este círculo.

La complejidad de éste algoritmo de orden cuadrático $O(n^2)$ Puesto que hay un ciclo for dentro de un ciclo while que se repiten n veces.

```
1 #include <iostream>
2 #include <vector>
3 #include <algorithm>
4
5 using namespace std;
6
7
8 int main()
9 {
10     int T, largo, num, ancho;
11
12     cin >> T;
13
14     while (T--)
15     {
16         int pos, rad;
17         cin >> num >> largo >> ancho;
18         vector< pair<int,int> > v(num);
19         for(int i = 0; i < num; i++)
20         {
21             cin >> pos >> rad;
22             if(rad>=ancho)
23                 v[i]=make_pair(pos,rad);
24         }
25     }
26 }
27
28 }
```

III. Dark road

Economic times these days are tough, even in Byteland. To reduce the operating costs, the government of Byteland has decided to optimize the road lighting. Till now every road was illuminated all night long, which costs 1 Bytelandian Dollar per meter and day. To save money, they decided to no longer illuminate every road, but to switch off the road lighting of some streets. To make sure that the inhabitants of Byteland still feel safe, they want to optimize the lighting in such a way, that after darkening some streets at night, there will still be at least one illuminated path from every junction in Byteland to every other junction.

What is the maximum daily amount of money the government of Byteland can save, without making their inhabitants feel unsafe?

Input

The input file contains several test cases. Each test case starts with two numbers m and n , the number of junctions in Byteland and the number of roads in Byteland, respectively. Input is terminated by $m = n = 0$. Otherwise, $1 \leq m \leq 200000$ and $m - 1 \leq n \leq 200000$. Then follow n integer triples x, y, z specifying that there will be a bidirectional road between x and y with length z meters ($0 \leq x, y < m$ and $x \neq y$). The graph specified by each test case is connected. The total length of all roads in each test case is less than 2^{31} .

Output

For each test case print one line containing the maximum daily amount the government can save.

My Submissions

#	Problem	Verdict	Language	Run Time	Submission Date
23424889	11631 Dark roads	Accepted	C++11	0.170	2019-05-31 20:43:02

Este algoritmo es prácticamente un árbol recubridor mínimo en donde está compuesto por todos los vértices y cuya suma de sus aristas es la de menor peso. Y para iniciar el componente se usó programación orientada a objetos en c++ para poder llevar a cuenta los visitados y los que no están visitados o no están unidos, y también en sus diferentes métodos si está en el mismo nodo y poder pasar al siguiente nodo y así hacer una unión con el mínimo costo posible.

Se escanean los 2 valores de entrada y dentro de un ciclo que siempre se va a repetir hasta que saque el mínimo camino. Comparando el peso y el camino más óptimo.

El orden de complejidad de éste algoritmo es de $O(n^2)$ puesto que es un árbol recubridor mínimo.

```
1 #include <cstdio>
2 #include <vector>
3 #include <utility>
4 #include <algorithm>
5
6 using namespace std;
7
8 class EmpezarUnion {
9 private:
10     vector<int> _rango, _pad;
11     int _emNum;
12 public:
13     EmpezarUnion();
14     EmpezarUnion(int emNum);
15     void reset(int nuevoTam);
16
17     int getlider(int u);
18     bool mismo(int u, int v);
19     void EmpezarUnion(int u, int v);
20     int getemNum();
21 };
22
23 EmpezarUnion::EmpezarUnion()
24 {
25     reset(0);
26 }
27
28 void EmpezarUnion::reset(int nuevoTam)
29 {
30     _rango.assign(nuevoTam, 0);
31     _pad.assign(nuevoTam, 0);
```

```
31     _pad.asign(nuevoTam, 0);
32     _emNum = nuevoTam;
33
34     for(int i = 0; i < nuevoTam; i++)
35         _pad[i] = i;
36 }
37
38 EmpezarUnion::EmpezarUnion(int emNum)
39 {
40     reset(emNum);
41 }
42
43 int EmpezarUnion::getlider(int u)
44 {
45     if(_pad[u] == u)
46         return u;
47
48     int lider = getlider(_pad[u]);
49     _pad[u] = lider;
50     return lider;
51 }
52
53 int EmpezarUnion::getemNum()
54 {
55     return _emNum;
56 }
57
58 bool EmpezarUnion::mismo(int u, int v)
59 {
60     return getlider(u) == getlider(v);
```

```
61 L }
62
63 void EmpezarUnion::EmpezarUnion(int u, int v)
64 {
65     int liderU = getlider(u), liderV = getlider(v);
66
67     if(liderU == liderV)
68         return;
69
70     _emNum--;
71     if(_rango[liderU] > _rango[liderV])
72     {
73         _pad[liderV] = liderU;
74     }
75     else
76     {
77         _pad[liderU] = liderV;
78         if(_rango[liderU] == _rango[liderV])
79             _rango[liderV]++;
80     }
81 }
82
83 int TamArbolMin(vector<p<int, p<int, int>>& caminos, int numV)
84 {
85     sort(caminos.begin(), caminos.end());
86
87     int minW = 0, numE = (int) caminos.size(), wE;
88     EmpezarUnion spanForest(numV);
89     p<int, int> pV;
90     for(int i = 0; i < numE; i++)
```

```

91 {
92     if(spanForest.getemNum() == 1)
93         break;
94
95     wE = caminos[i].first;
96     pV = caminos[i].second;
97     if(!spanForest.mismo(pV.first, pV.second))
98     {
99         minW += wE;
100        spanForest.EmpezarUnion(pV.first, pV.second);
101    }
102
103
104    return minW;
105}
106
107 int main(void)
108 {
109     vector<p<int, p<int, int> > > caminos;
110     int numV, numE, sumE, u, v, pesoE, MinCamino;
111
112     while(1)
113     {
114         scanf("%d %d", &numV, &numE);
115         if(numV == 0 && numE == 0)
116             break;
117
118         caminos.clear();
119         sumE = 0;
120
121         for(int i = 0; i < numE; i++)
122         {
123             scanf("%d %d %d", &u, &v, &pesoE);
124             sumE += pesoE;
125             caminos.push_back(make_p(pesoE, make_p(u, v)));
126         }
127
128         MinCamino = TamArbolMin(caminos, numV);
129         printf("%d\n", sumE - MinCamino);
130     }
131
132     return 0;
133 }
```

a

IV. ESPANTAPÁJAROS

Taso owns a very long field. He plans to grow different types of crops in the upcoming growing season. The area, however, is full of crows and Taso fears that they might feed on most of the crops. For this reason, he has decided to place some scarecrows at different locations of the field.

The field can be modeled as a $1 \times N$ grid. Some parts of the field are infertile and that means you cannot grow any crops on them. A scarecrow, when placed on a spot, covers the cell to its immediate left and right along with the cell it is on.

Given the description of the field, what is the minimum number of scarecrows that needs to be placed so that all the useful section of the field is covered? Useful section refers to cells where crops can be grown.



Input

Input starts with an integer T (≤ 100), denoting the number of test cases.

Each case starts with a line containing an integer N ($0 < N < 100$). The next line contains N characters that describe the field. A dot (.) indicates a crop-growing spot and a hash (#) indicates an infertile region.

Output

For each case, output the case number first followed by the number of scarecrows that need to be placed.

#	Problema	Veredicto	Idioma	Tiempo de ejecución	Día de entrega
23424774	12405 Espantapájaros	Aceptado	C ++ 11	0.000	2019-05-31 19:49:49
23408631	10382 Hierba de riego	Aceptado	C ++ 11	0.000	2019-05-28 13:36:49
23408618	10382 Hierba de riego	Límite de tiempo excedido	C ++ 11	3.000	2019-05-28 13:34:46
23408580	10382 Hierba de riego	Límite de tiempo excedido	C ++ 11	3.000	2019-05-28 13:29:46
23408550	10382 Hierba de riego	Respuesta incorrecta	C ++ 11	0.000	2019-05-28 13:24:30
23408531	10020 Cobertura mínima	Aceptado	C ++ 11	0.060	2019-05-28 13:21:01
23336109	1213 Suma de diferentes Primes	Respuesta Incorrecta	ANSI C	0.780	2019-05-15 01:44:53

Para éste algoritmo la solución es un ciclo en el cual encuentra los cuadritos y mientras sea menor a la línea y si encuentra un . que es las que tiene que proteger le agrega 3 para que el espantapájaros pueda cubrir perfectamente y no pasen los cuervos. Y si no es el caso simplemente se mueve 1 espacio cada iteración hasta volver a la condición en la que se aumentan 3 espacios para proteger.

El orden de complejidad para este algoritmo es de $O(n^2)$ puesto que el ciclo while que valida toda la zona a cubrir está dentro de un ciclo for que se repetirá n veces para imprimir en dónde está parado.

```

1 #include <iostream>
2
3 #include <string>
4 using namespace std;
5 int N;
6 string esp;
7 int main(){
8     int TC,tc=1,i=0,k=0;
9     scanf("%d",&TC);
10    for(tc=1;tc<=TC;++tc){
11        printf("Caso %d: ",tc);
12        scanf("%d",&N);
13        cin >> esp;
14        while(i < N){
15            if(esp[i] == '.'){
16                ++k;
17                i += 3;
18            } else {
19                ++i;
20            }
21        }
22        printf("%d\n",k);
23    }
24    return 0;
25 }
```

V. Heavy cargo

My Submissions

#	Problem	Verdict	Language	Run Time	Submission Date
23424825	544 Heavy Cargo	Accepted	C++11	0.000	2019-05-31 20:14:51

Para éste algoritmo, la forma de resolverlo fue en un ciclo while que siempre se va a repetir y que dentro de ese, un ciclo for en el cual se va a escanear el nombre y el peso para pasar al siguiente, después de que el par máximo se haya cargado, escanea de nuevo el nombre y el id lo asigna y regresa de la función para comparar si el valor uno es menor que el numero 1 y si el valor 2 es menor al numero 1 entonces imprime el cargo máximo.

El orden de complejidad de éste algoritmo es de $O(n^3)$ porque para sacar el máximo y poder cargarlo se tiene que recorrer todas las variables hasta que sean menores al numero 1 y así comparar y cargar.

```
1 #include <map>
2 #include <string>
3
4
5
6 using namespace std;
7
8 #define MAX_NUM_V 200
9 #define MAX_NAME 30
10
11 int Id(char name[], map<string, int> & mnt, int * sigid);
12 void MaxPar(int CargoMax[][MAX_NUM_V], int numV);
13 int Min(int a, int b);
14
15 int main(void)
16 {
17     int CargoMax[MAX_NUM_V][MAX_NUM_V], numV, numE, cid, sigid,
18         v1, v2, peso;
19     map<string, int> mnt;
20     char name[MAX_NAME + 1];
21
22     cid = 1;
23     while(1)
24     {
25         scanf("%d %d", &numV, &numE);
26         if(numV == 0 && numE == 0)
27             break;
28
29         sigid = 0;
30         memset(CargoMax, 0, MAX_NUM_V * MAX_NUM_V * sizeof(int));
31     }
}
```

```
31
32     for(int e = 0; e < numE; e++)
33     {
34         scanf("%s", name);
35         v1 = Id(name, mnt, &sigid);
36         scanf("%s", name);
37         v2 = Id(name, mnt, &sigid);
38         scanf("%d", &peso);
39         CargoMax[v1][v2] = peso;
40         CargoMax[v2][v1] = peso;
41     }
42     MaxPar(CargoMax, numV);
43     scanf("%s", name);
44     v1 = Id(name, mnt, &sigid);
45     scanf("%s", name);
46     v2 = Id(name, mnt, &sigid);
47     if(v1 < numV && v2 < numV)
48     {
49         printf("Escenario #%d\n%d toneladas\n\n", cid, CargoMax[v1][v2]);
50     }
51     cid++;
52 }
53 return 0;
54 int Id(char name[], map<string, int> & mnt, int * sigid)
55 {
56     map<string, int>::iterator cont = mnt.find(name);
57     int regresaId;
58     if(cont == mnt.end())
59     {
```

```
60     regresaId = *sigid;
61     mnt[name] = *sigid;
62     (*sigid)++;
63 }
64 else
65     regresaId = cont->second;
66
67 return returnId;
68 }
69 void MaxPar(int CargoMax[][MAX_NUM_V], int numV)
70 {
71     int u, v, k, load;
72     for(k = 0; k < numV; k++)
73         for(u = 0; u < numV; u++)
74             for(v = 0; v < numV; v++)
75             {
76                 if(CargoMax[u][k] && CargoMax[k][v])
77                 {
78                     load = Min(CargoMax[u][k], CargoMax[k][v]);
79                     if(load > CargoMax[u][v])
80                         CargoMax[u][v] = load;
81                 }
82             }
83 }
84 int Min(int a, int b)
85 {
86     if(a > b)
87         return b;
88     return a;
89 }
```