Soluciones entrenos OIE
Traffic - IOI 2010

Aunque Canadá es un país bastante grande, muchas zonas están deshabitadas y la mayoría de la población vive cerca de la frontera sur. La Autopista Trans-Canadiense, terminada en 1962, conecta a la gente que vive en esta franja de tierra, desde St. John's en el Este, hasta Victoria en el Oeste, una distancia de 7821 km.

Como ya sabrás, a los canadienses les gusta mucho el hockey. Después de cada partido de hockey, miles de fans vuelven a su casa en coche, causando una fuerte congestión de tráfico en las carreteras. Un rico empresario quiere comprar un equipo de hockey y construir un nuevo estadio. Tu tarea es ayudarle a seleccionar el lugar donde debe estar situado el estadio para minimizar la congestión de tráfico después de los partidos de hockey.

El país está organizado en ciudades conectadas por una red de carreteras. Todas las carreteras son bidireccionales (se puede circular en las dos direcciones), y hay exactamente una ruta que conecta cualquier par de ciudades. El nuevo estadio debe ser construido en una de las ciudades, a la que llamaremos la ciudad estadio. Después de un partido de hockey, todos los aficionados viajan de la ciudad arena a la ciudad donde viven, excepto los que ya viven en la ciudad arena, que no se han de desplazar. La cantidad de congestión en cada carretera es proporcional al número de aficionados que viajan por la carretera. Debes ubicar la ciudad estadio de tal manera que la cantidad de congestión en la carretera más congestionada sea lo más pequeña posible. Si hay varios lugares igualmente buenos para colocar el estadio, puedes elegir cualquiera de ellos.

Detalles de la implementación

Debes implementar la siguiente función:

int LocateCentre(int n**, int** $p[]$**, int** $s[]$**, int** $d[]$**)**.

- n representa el número de ciudades. Las ciudades están numeradas del 0 al $n-1$. En todos los casos habrá como mucho 2000000000 (2 mil millones) de aficionados.
- $p[]$ es una lista con n elementos. El elemento $p[i]$ representa el número de fans del hockey que viven en la ciudad i.
- $s[]$ y $d[]$ son dos listas con $n-1$ elementos cada una. Los elementos $s[i]$ y $d[i]$ indican que la i-ésima carretera va desde la ciudad $s[i]$ hasta la ciudad $d[i]$.
- Todas las carreteras son bidireccionales y hay exactamente una ruta para viajar entre cada par de ciudades. En otras palabras, hay exactamente un camino para llegar desde la ciudad i hasta la ciudad j para cualquier $0 \leq i, j \leq n-1$ con $i \neq j$.
- La función ha de devolver la ciudad en la que se debería colocar el estadio.

Subtareas

Subtask	Puntos	n	Otras constraints
1	25	$n \leq 1000$	$s[i] = i$ y $d[i] = i + 1$ para $0 \leq i < n - 1$
2	25	$n \leq 1000000$	Las mismas constraints que en la primera subtarea.
3	25	$n \leq 1000$	Nada
4	25	$n \leq 1000000$	Nada

Solución

Es importante recordar que hay exactamente un camino entre cada par de ciudades. Esto significa que el grafo que constituye las diferentes ciudades y carreteras es un árbol. Si no sabes lo que es un grafo o lo que es un árbol, te recomiendo leer lo que pone en esta página web: https://aprende.olimpiada-informatica.org/algoritmia-grafos.

Lo primera observación a la que hay que llegar es que si ponemos el estadio en una cierta ciudad, la carretera con más congestión será una carretera que salga de la ciudad donde está el estadio. Imaginemos por un momento que el estadio está en la ciudad i pero la carretera con más congestión va desde j hasta k, siendo j y k diferentes de i. Esto nos lleva a una contradicción, ya que todos los aficionados tendrán que pasar por una carretera conectada a i para poder llegar a esta carretera. Esto implica que la carretera conectada a i tendrá como mínimo la misma congestión que la carretera entre j y k. Esto contradice el hecho de que la carretera entre j y k sea la más congestionada. Por lo tanto si pongo el estadio en i, la carretera con más congestión tiene que ser una que salga de i.

Para cada ciudad i podemos fijar por lo tanto el estadio en esa ciudad (ponerlo ahí temporalmente) y calcular cual será la máxima congestión. Lo que haremos será iterar sobre todas las carreteras que salgan de la ciudad i y comprobar cuantos aficionados pasarán por ahí después del partido. Hemos mencionado antes que el grafo es un árbol. Lo que podemos hacer es una búsqueda por profundidad (un DFS) para contar el número de aficionados que habrá en el subárbol de cada nodo (Si no entiendes lo que es un DFS o un subárbol te recomiendo leerte los manuales que tenemos de la OIE [1] u otros recursos en Internet). El número de aficionados que pasarán por una carretera que vaya de i a un hijo suyo en el árbol j, es el número de aficionados en el subárbol de j. Por otro lado, el número de aficionados que pasarán por la carretera que vaya de i a su padre en el árbol es el número total de aficionados menos el número de aficionados en el subárbol de i. La carretera por la que pase más gente de todas las que salgan de i será la que tenga la máxima congestión de tráfico. Nuestro objetivo es encontrar el nodo para el cual la carretera con más congestión tenga la mínima congestión posible.

[1] https://aprende.olimpiada-informatica.org/algoritmia-graph-traversal

Código

C++

```cpp
#include "traffic.h"
#include <bits/stdc++.h>
using namespace std;
#define MAX_N 1e6 + 7
vector<vector<int> > adj(MAX_N);
vector<int> aficionados(MAX_N), subarbol(MAX_N);
int total_aficionados = 0, minimo = 2e9 + 7, mejor_nodo = -1;

void dfs(int a, int p) {
    subarbol[a] = aficionados[a];
    int maxima_congestion = 0;

    for(int b : adj[a]) if(b != p) {
        dfs(b, a);
        subarbol[a] += subarbol[b];
        maxima_congestion = max(maxima_congestion, subarbol[b]);
    }

    maxima_congestion = max(maxima_congestion,
                            total_aficionados - subarbol[a]);

    if(maxima_congestion < minimo) {
        mejor_nodo = a;
        minimo = maxima_congestion;
    }
}

int LocateCentre(int n, int p[], int d[], int s[]) {
    for(int i = 0; i < n; i++) {
        aficionados[i] = p[i];
        total_aficionados += p[i];
    }

    for(int i = 0; i < n - 1; i++) {
        adj[d[i]].push_back(s[i]);
        adj[s[i]].push_back(d[i]);
    }

    dfs(0, 0);

    return mejor_nodo;
}
```