Soluciones entrenos OIE

Connecting Supertrees

Solución

Para resolver este problema, separaremos el grafo en sus distintas componentes conexas, usando union-find. Hay que tener en cuenta que si hay cualquier pareja con 3 conexiones, es imposible encontrar tal grafo.

Por lo tanto, terminamos con una serie de componentes conexas, cuyos vértices tienen o un camino o dos entre ellos. Si una componente conexa puede construirse según las especificaciones, tiene que cumplirse los siguiente:

1. Todos los vértices de la componente conexa deben de tener por lo menos un camino entre ellos ($p[i][j] > 0$ para todo i, j en la componente conexa).
2. La componente conexa se puede partir en una serie de "serpientes". Los vértices dentro de la misma serpiente tienen un camino que los conecta y dos con los del resto de serpientes ($p[i][j] = 1$ si i y j comparten serpiente y $p[i][j] = 2$ si no es el caso.

Por lo tanto, construimos las serpientes también con union-find, asegurándonos de que no haya incongruencias respecto a estas condiciones.

Finalmente, conectamos los elementos dentro de la serpiente en línea recta. Así, solo hay un camino posible entre cualquier pareja dentro de la serpiente.

Conectamos la cabeza de todas las serpientes en un "círculo de cabezas", consiguiendo así que haya dos formas de conectarse con cualquier nodo ajeno a la serpiente.

Código

C++

```cpp
#include <iostream>
#include <vector>
#include <unordered_map>
#include "supertrees.h"
using namespace std;
typedef vector<int> vi;
typedef vector<vi> vvi;

int root(int a, vi & parent) {
	if (a == parent[a])
		return a;
	return parent[a] = root(parent[a], parent);
}

void merge(int i, int j, vi & parent, vi & rnk, int & numCCs) {
	int root_i = root(i, parent), root_j = root(j, parent);
	if (root_i == root_j) return;
	numCCs--;
	if (rnk[i] != rnk[j]) {
		if (rnk[i] > rnk[j]) parent[root_j] = parent[root_i];
		else parent[root_i] = parent[root_j];
	} else {
		parent[root_i] = parent[root_j];
		rnk[j]++;
	}
}

int construct(vvi p) {
	// encontramos las componentes conexas del grafo con el método Union-Find
	int N = (int) p.size(), numCCs = N;
	vi parent(N), rnk(N, 0);

	for (int i = 0; i < N; i++)
		parent[i] = i;
	for (int i = 0; i < N; i++) {
		for (int j = i+1; j < N; j++) {
			if (p[i][j]) {
				if (p[i][j] == 3) {
					return 0;
				}
				merge(i, j, parent, rnk, numCCs);
			} else if (root(i, parent) == root(j, parent))
				return 0;
		}
	}
```

```
        unordered_map<int, int> roots;
        int c = 0;
        for (int i = 0; i < N; i++) {
                if (parent[i] == i)
                        roots[i] = c++;
        }

        vvi CCs(numCCs, vi());
        for (int i = 0; i < N; i++)
                CCs[roots[root(i, parent)]].push_back(i);

        int CCsize, u, v, n, root_u, root_v;
        vvi result(N, vi(N, 0)), oneTails;

        for (int i = 0; i < numCCs; i++) {
                CCsize = (int) CCs[i].size();
                if (CCsize == 1) continue;

                // encontramos las "serpientes" de esta componente conexa
                n = CCsize;
                for (auto node: CCs[i]) {
                        parent[node] = node;
                        rnk[node] = 0;
                }
                for (int j = 0; j < CCsize; j++) {
                        u = CCs[i][j];
                        root_u = root(u, parent);
                        for (int k = j+1; k < CCsize; k++) {
                                v = CCs[i][k];
                                if (p[u][v] == 1)
                                        merge(u, v, parent, rnk, n);
                                else if (root_u == root(v, parent))
                                        return 0;
                        }
                }

                // nos aseguramos de que no hay incongruencias en esta componente
                ↪  conexa
                if (n == 2) return 0;

                roots.clear();
                c = 0;
                for (int j = 0; j < CCsize; j++) {
                        u = CCs[i][j];
                        root_u = root(u, parent);
                        if (root_u == u)
                                roots[u] = c++;
                        for (int k = j+1; k < CCsize; k++) {
                                v = CCs[i][k];
                                root_v = root(v, parent);
                                if (root_u != root_v) {
                                        if (p[u][v] != 2) {
                                                return 0;
                                        }
                                } else if (p[u][v] != 1) {
                                        return 0;
                                }
                        }
                }
```

```
                // conectar los elementos dentro de una serpiente
                oneTails = vvi(n, vi());
                for (int j = 0; j < CCsize; j++)
                        oneTails[roots[root(CCs[i][j],
                        ↪  parent)]].push_back(CCs[i][j]);
                for (int j = 0; j < n; j++) {
                        for (int k = 0; k < (int) oneTails[j].size() - 1; k++)
                                result[oneTails[j][k]][oneTails[j][k+1]] =
                                ↪  result[oneTails[j][k+1]][oneTails[j][k]] = 1;
                }

                // conectar el ciclo de cabezas de serpiente
                for (int j = 0; j < n-1; j++)
                        result[oneTails[j][0]][oneTails[j+1][0]] =
                        ↪  result[oneTails[j+1][0]][oneTails[j][0]] = 1;

                if (n > 1)
                        result[oneTails[0][0]][oneTails[n-1][0]] =
                        ↪  result[oneTails[n-1][0]][oneTails[0][0]] = 1;
        }
        build(result);
        return 1;
}
```