Soluciones entrenos OIE

Dreaming- IOI 2013

Esta historia sucedió hace mucho tiempo, cuando se creó el mundo y la IOI aún no había sido soñada. Serpiente vive en un lugar que tiene N estanques (agujeros llenos de agua), numerados 0, ..., $N-1$. Hay M senderos bidireccionales uniendo pares de estanques, por los cuales Serpiente puede viajar. Cada par de estanques están conectados (directamente o indirectamente) por a lo más una secuencia de senderos, aunque algunos pares de estanques podrían no estar conectados en absoluto (o sea $M \leq N-1$). Serpiente le toma una cierta cantidad de días viajar por cada uno de los senderos: este número puede ser distinto para cada sendero.

Canguro, el amigo de Serpiente, desea construir $N-M-1$ senderos nuevos, para que así Serpiente pueda viajar entre cualquier par de estanques. Canguro puede crear senderos entre cualquier par de estanques, y Serpiente necesitará L días para viajar a través de cada sendero creado.

Además, Canguro quiere que Serpiente viaje tan rápido como sea posible. Canguro construirá nuevos senderos de modo que la distancia entre cualquier par de estanques sea tan pequeña como sea posible. Ayuda a Canguro y Serpiente a determinar el tiempo de viaje más largo entre dos estanques después de que Canguro haya constuido los senderos necesarios.

Constraints

- $1 \leq N \leq 100000$
- $0 \leq M \leq N-1$
- $0 \leq A[i], B[i] \leq N-1$
- $1 \leq T[i] \leq 10,000$
- $1 \leq L \leq 10,000$
- 14 Puntos: $M = N-2$, y hay exactamente uno o dos senderos preexistentes saliendo de cada estanque. En otras palabras, hay dos conjuntos de estanques conectados, y en cada conjunto los senderos forman un camino sin ramificaciones.
- 10 Puntos: $M = N-2$ y $N \leq 100$
- 23 Puntos: $M = N-2$
- 18 Puntos: Como mucho hay un sendero pre-existente saliendo de cada estanque.
- 12 Puntos: $N \leq 3000$
- 23 Puntos: Constraints originales

Solución

Para conseguir los 100 puntos, basta con una solución que conecte los árboles del bosque en una forma de estrella, con la componente conexa con una mayor eccentricidad mínima de centro. Esta solución tiene una complejidad de O(n).

Código

C++

```cpp
pii calcDist(int i, vvpii & G, vi & dist1, vi & dist2, vi & dist3) {
        // calcularemos el diámetro con el método de las dos colas, que primero busca el nodo más lejano de la raíz i (maxNode)
        // después, rellenamos dist2 con la distancia entre maxNode y el resto de nodos de la componente conexa
        // así, encuentra el nodo más lejano de maxNode (maxNode2). El diámetro es la distancia entre maxNode y maxNode2.
        // finalmente, rellenamos dist3 con la distancia entre maxNode2 y el resto de nodos de la componente conexa
        // la eccentricidad de cada vértice será el máximo entre la distancia con maxNode y la distancia con maxNode2
        queue<int> Q;
        Q.push(i);
        int u, v, t, maxNode = i;
        vector<int> vertices = {}; // aquí guardaremos los vértices de la componente conexa
        dist1[i] = 0;
        while(!Q.empty()) {
                u = Q.front();
                Q.pop();
                vertices.push_back(u); // añadimos u a la lista de vértices de la componente conexa
                if (dist1[u] > dist1[maxNode]) // actualizamos maxNode si procede
                        maxNode = u;
                for (auto conexion: G[u]) { // procesamos las conexiones de u
                        v = conexion.first;
                        t = conexion.second;
                        if (dist1[v] > dist1[u]+t) { // v no visitado
                                dist1[v] = dist1[u]+t; // actualizamos la distancia de v en base a la de u
                                Q.push(v);
                        }
                }
        }
```

```
    // hacemos lo mismo para maxNode
        dist2[maxNode] = 0;
        Q.push(maxNode);
        int maxNode2 = maxNode;
        while(!Q.empty()) {
                u = Q.front();
                Q.pop();
                if (dist2[u] > dist2[maxNode2])
                        maxNode2 = u;
                for (auto conexion: G[u]) {
                        v = conexion.first;
                        t = conexion.second;
                        if (dist2[v] > dist2[u]+t) {
                                dist2[v] = dist2[u]+t;
                                Q.push(v);
                        }
                }
        }
    // hacemos lo mismo para maxNode2
        dist3[maxNode2] = 0;
        Q.push(maxNode2);
        while(!Q.empty()) {
                u = Q.front();
                Q.pop();
                for (auto conexion: G[u]) {
                        v = conexion.first;
                        t = conexion.second;
                        if (dist3[v] > dist3[u]+t) {
                                dist3[v] = dist3[u]+t;
                                Q.push(v);
                        }
                }
        }
        int ecc = 1e9;
        for (auto vert: vertices) {
                ecc = min(ecc, max(dist2[vert], dist3[vert])); // calculamos la
                 ↪  eccentricidad de vert y actualizamos el mínimo
        }
        return make_pair(ecc, dist2[maxNode2]);
}
int travelTime(int N, int M, int L, int A[], int B[], int T[]) {
        // primero, pasamos las listas de aristas a un grafo en formato "adjacency
        ↪  list"
        // guardamos cada arista en la lista de sus dos nodos, ya que el grafo es
        ↪  bidireccional
        vvpii G(N, vpii());
        for (int i = 0; i < M; i++) {
                G[A[i]].push_back(make_pair(B[i], T[i]));
                G[B[i]].push_back(make_pair(A[i], T[i]));
        }
        int CC = 0; // número de componentes conexas del grafo (un bosque)
        vi diam = {}; // diámetro de las componentes conexas
        vi ecc = {}; // guarda para cada componente conexa la eccentricidad del
        ↪  vértice con menor eccentricidad del grafo
        vi dist1; // guarda la distancia entre la raíz de la componente conexa y
        ↪  cada uno de sus nodos
```

```
        vi dist2; // guarda la distancia entre el nodo más lejano de la raíz de la componente conexa y el resto de sus nodos
        vi dist3; // guarda la distancia entre el nodo que define dist2 y el resto de nodos de la componente conexa
        dist1 = dist2 = dist3 = vi(N, 1e9); // asignamos "infinito" como valores iniciales para las distancias
        pii curCC; // aquí guardamos el valor devuelto de la componente conexa
        for (int i = 0; i < N; i++) {
                if (dist1[i] == 1e9) { // nodo i no visitado aún
                        curCC = calcDist(i, G, dist1, dist2, dist3); // {ecc, diam} de la componente conexa de i
                        ecc.push_back(curCC.first);
                        diam.push_back(curCC.second);
                        CC++;
                }
        }
        // para minimizar el camino más largo, guardaremos el grafo como una estrella, conectando los centros de cada
        // componente conexa todas al centro de la componente conexa con mayor eccentricidad
        // así, la respuesta será una de las siguientes:
        // 1. la componente conexa con el mayor diámetro
        // 2. la mayor distancia entre el centro de las dos componentes con mayor eccentricidad
        // 3. la distancia entre la segunda y la tercera mayor eccentricidad, pasando por el centro medio

        // guardamos en ecc[0] la eccentricidad mayor
        for (int i = 1; i < CC; i++) {
                if (ecc[i] > ecc[0])
                        swap(ecc[0], ecc[i]);
        }
        int ans = 0;
        if (CC > 1) {
                // guardamos en ecc[1] la sedunda eccentricidad mayor
                for (int i = 2; i < CC; i++) {
                        if (ecc[i] > ecc[1])
                                swap(ecc[i], ecc[1]);
                }
                ans = ecc[0]+ecc[1]+L; // posibilidad 2
                if (CC > 2) {
                        // guardamos en ecc[2] la tercera eccentricidad mayor
                        for (int i = 3; i < CC; i++) {
                                if (ecc[i] > ecc[2])
                                        swap(ecc[i], ecc[2]);
                        }
                        ans = max(ans, ecc[1]+ecc[2]+2*L); // posibilidad 3
                }
        }
        for (int i = 0; i < CC; i++) {
                ans = max(ans, diam[i]); // posibilidad 1
        }
        return ans;
```