Grafos

(Para el Nivel Inicial)

Julián Braier

Facultad de Ciencias Exactas y Naturales - UBA

Training Camp 2024

Gracias Sponsors!

Organizador

Diamond

Gold

Outline

Outline

Motivación

- Las Ciencias de la Computación están impregnadas de problemas de grafos.
- Problemas muy variados pueden ser resueltos representando la situación con un grafo.
- En programación competitiva: no vi competencia sin problema de grafos.

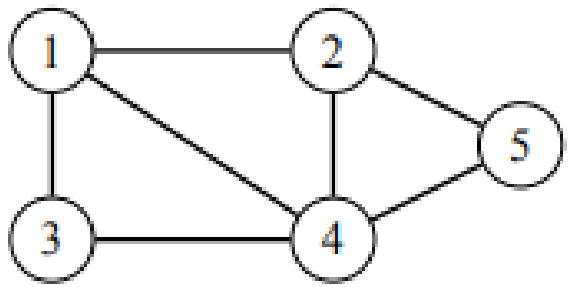
Outline

Grafo

Grafo $G = (V, E)$

- V es un conjunto de vértices, alias nodos.
- Algunos pares de nodos están conectados por aristas, alias ejes. E es el conjunto de aristas.
- **Notación:** n es la cantidad de nodos, m la cantidad de aristas.

Fig. 7.1 A graph with 5 nodes and 7 edges



Caminos y Ciclos

- Un **camino** lleva de un vértice a otro usando aristas del grafo. Por ejemplo, en la figura, $1 \to 3 \to 4 \to 5$. La longitud de un camino es la cantidad de aristas.
- Un **ciclo** es un camino que empieza y termina en el mismo vértice. Ejemplo: $1 \to 3 \to 4$.

Fig. 7.2 A path from node 1 to node 5

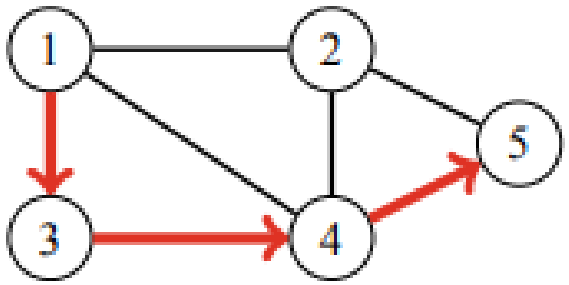
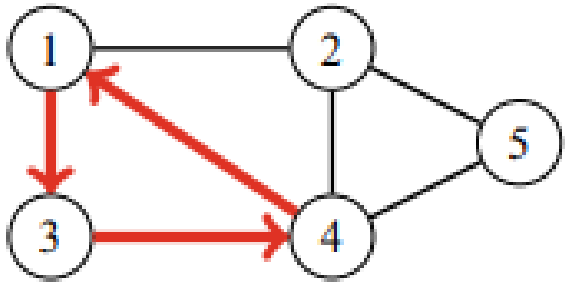


Fig. 7.3 A cycle of three nodes



Conexidad y Componentes

- Decimos que un grafo es **conexo** si existe camino entre todo par de vértices.
- A cada conjunto maximal de vértice que sea conexo le llamamos **componente**.

Fig. 7.4 The left graph is connected, the right graph is not

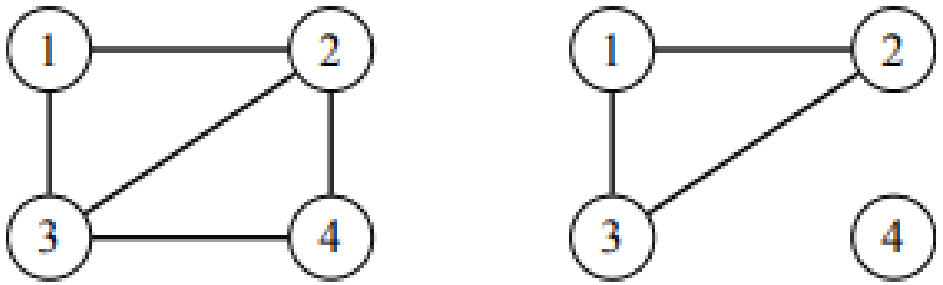
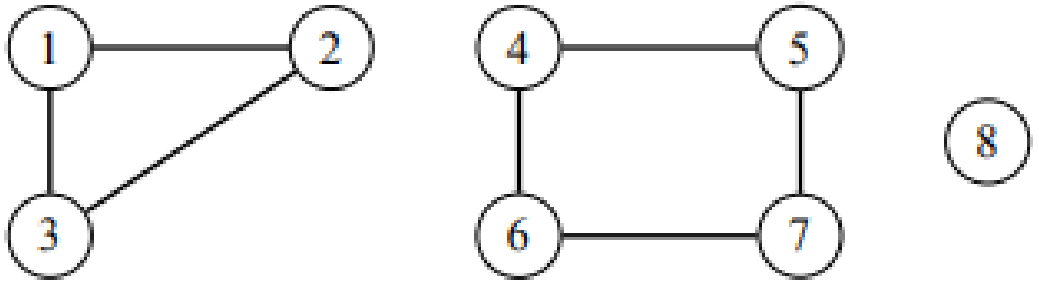


Fig. 7.5 Graph with three components

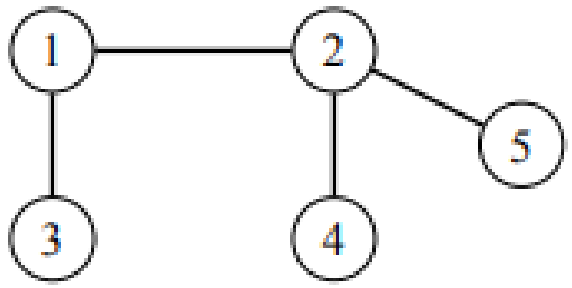


Árboles

Un árbol es un grafo:

1. Conexo
2. Sin ciclos
3. Con $m = n - 1$

Fig. 7.6 A tree



Otros Tipos de Grafos

- En un grafo **dirigido** las aristas se pueden recorrer en un único sentido.
- En un grafo **pesado**, cada arista tiene un peso asociado.

Fig. 7.7 Directed graph

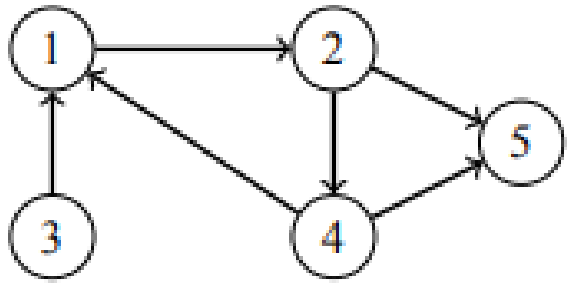
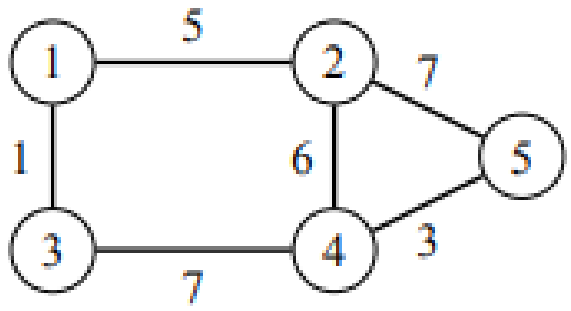


Fig. 7.8 Weighted graph



Outline

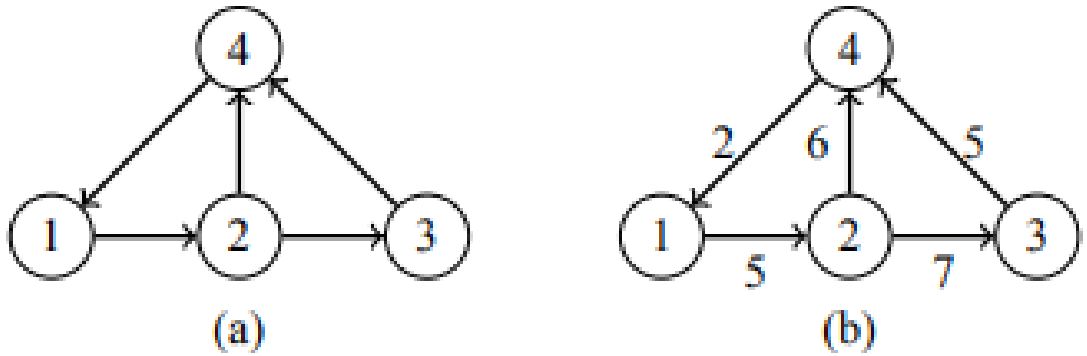
Representaciones Comunes

Para resolver con grafos, hay que representarlos en la computadora. Elegir la representación que sea más conveniente según características del grafo y qué operaciones necesite realizar el algoritmo. Vemos tres representaciones comunes:

1. Lista de adyacencias.
2. Lista de aristas.
3. Matriz de adyacencia.

Lista de Adyacencias

Fig. 7.12 Example graphs



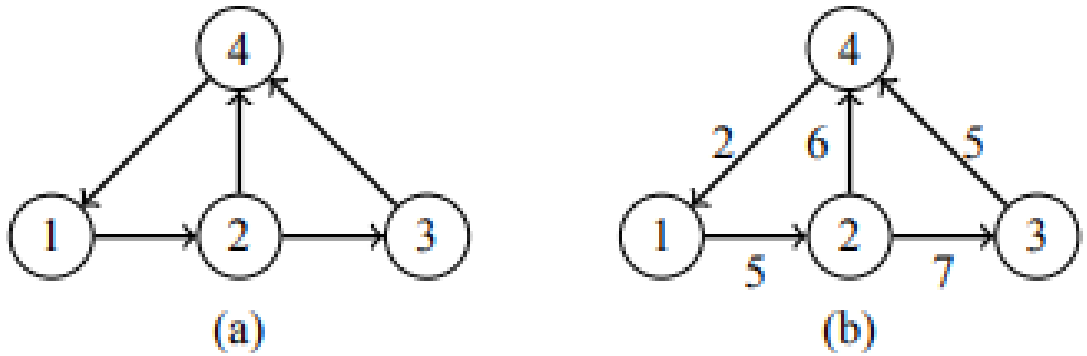

```
vector<int> adj[N];
```

The constant N is chosen so that all adjacency lists can be stored. For example, the graph in Fig. 7.12a can be stored as follows:

```
adj[1].push_back(2);
adj[2].push_back(3);
adj[2].push_back(4);
adj[3].push_back(4);
adj[4].push_back(1);
```

Lista de Adyacencias para Grafo Pesado

Fig. 7.12 Example graphs



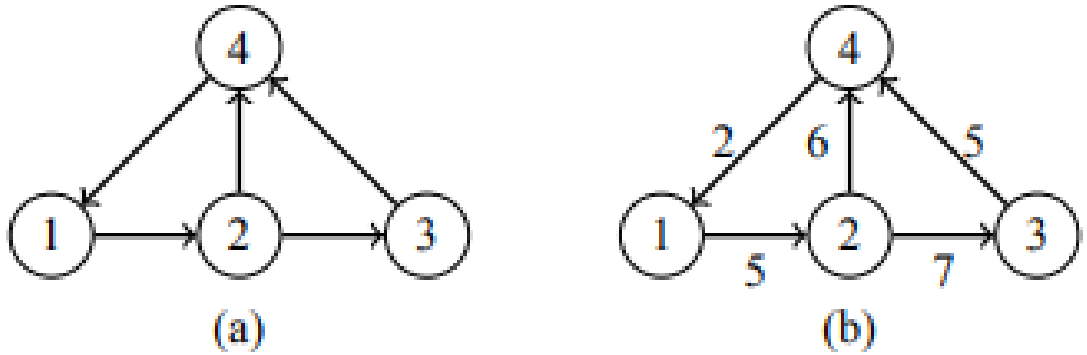

```
vector<pair<int,int>> adj[N];
```

In this case, the adjacency list of node a contains the pair (b, w) always when there is an edge from node a to node b with weight w. For example, the graph in Fig. 7.12b can be stored as follows:

```
adj[1].push_back({2,5});
adj[2].push_back({3,7});
adj[2].push_back({4,6});
adj[3].push_back({4,5});
adj[4].push_back({1,2});
```

Lista de Aristas

Fig. 7.12 Example graphs



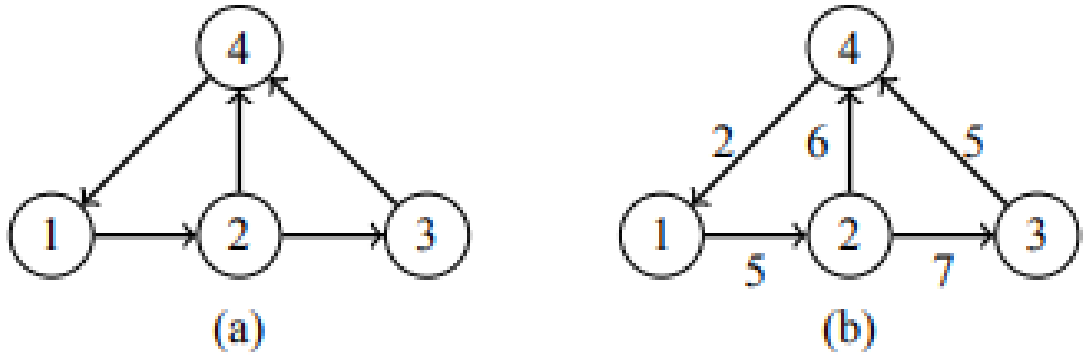

```
vector<pair<int,int>> edges;
```

where each pair (a, b) denotes that there is an edge from node a to node b. Thus, the graph in Fig. 7.12a can be represented as follows:

```
edges.push_back({1,2});
edges.push_back({2,3});
edges.push_back({2,4});
edges.push_back({3,4});
edges.push_back({4,1});
```

Lista de Aristas para Grafo Pesado

Fig. 7.12 Example graphs



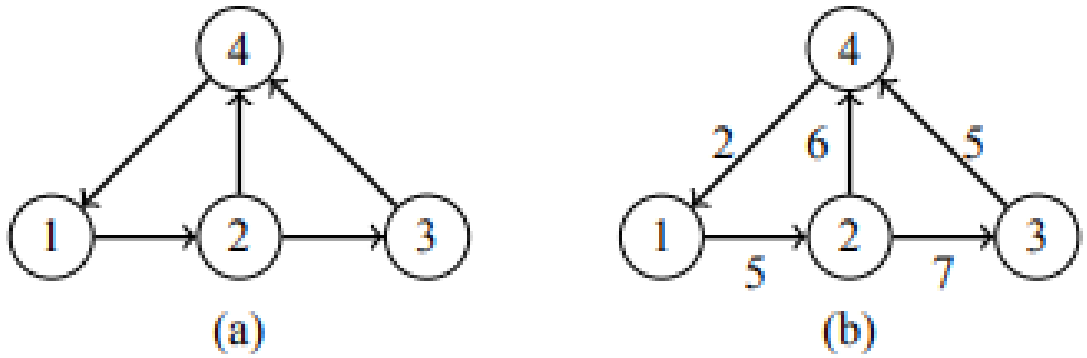

```
vector<tuple<int,int,int>> edges;
```

Each element in this list is of the form (a, b, w), which means that there is an edge from node a to node b with weight w. For example, the graph in Fig. 7.12b can be represented as follows[1]:

```
edges.push_back({1,2,5});
edges.push_back({2,3,7});
edges.push_back({2,4,6});
edges.push_back({3,4,5});
edges.push_back({4,1,2});
```

Matriz de Adyacencia

Fig. 7.12 Example graphs



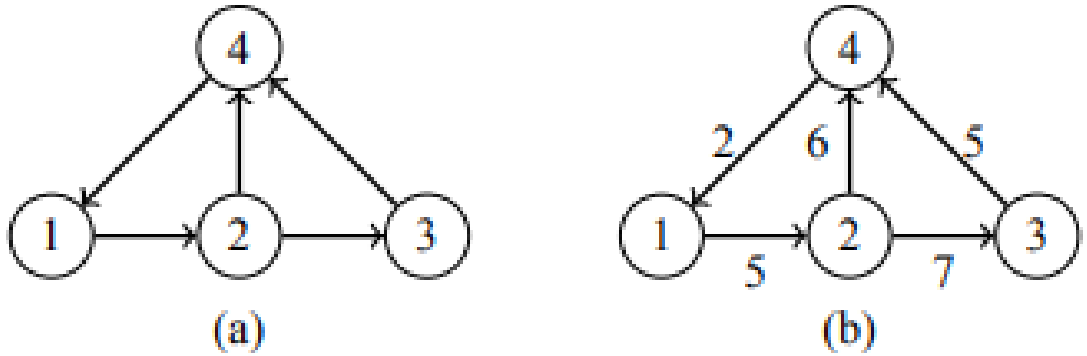

```
int adj[N][N];
```

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

$ady[u][v] = 1$ si y sólo si hay arista de vértice u a v.

Matriz de Adyacencia para Grafos Pesados

Fig. 7.12 Example graphs




```
int adj[N][N];
```

$$\begin{bmatrix} 0 & 5 & 0 & 0 \\ 0 & 0 & 7 & 6 \\ 0 & 0 & 0 & 5 \\ 2 & 0 & 0 & 0 \end{bmatrix}$$

Outline

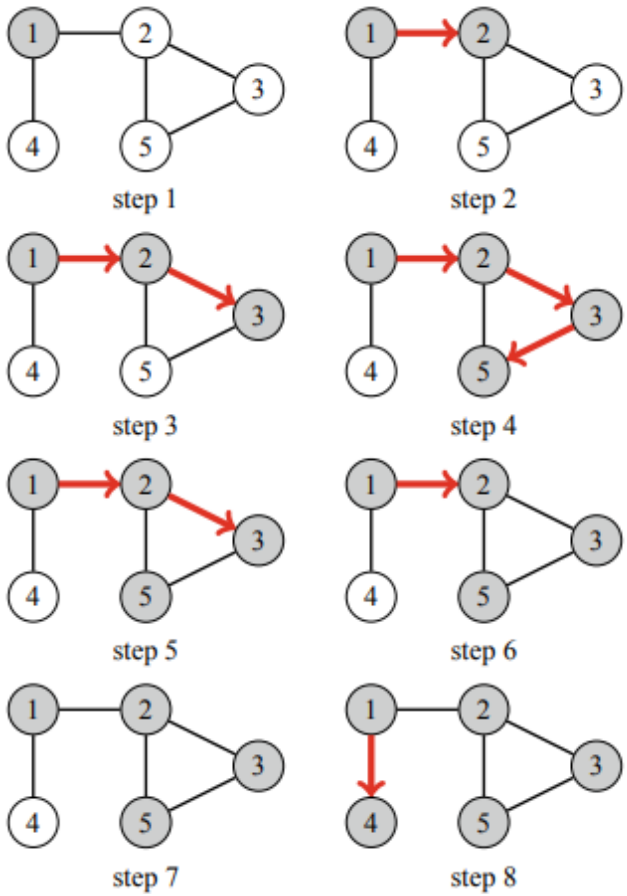
Recorridos

Vemos dos algoritmos fundamentales de grafos: depth-first search (DFS) y breadth-first search (BFS). Ambos recorren todos los nodos que pueden ser alcanzados desde un v inicial, pero en distinto orden.

DFS

- Recorrido “en profundidad”.
- Sigue por un camino mientras encuentre nodos sin explorar. Luego, vuelve a nodos previos a explorar otras partes del grafo.

Fig. 7.13 Depth-first search



Código DFS

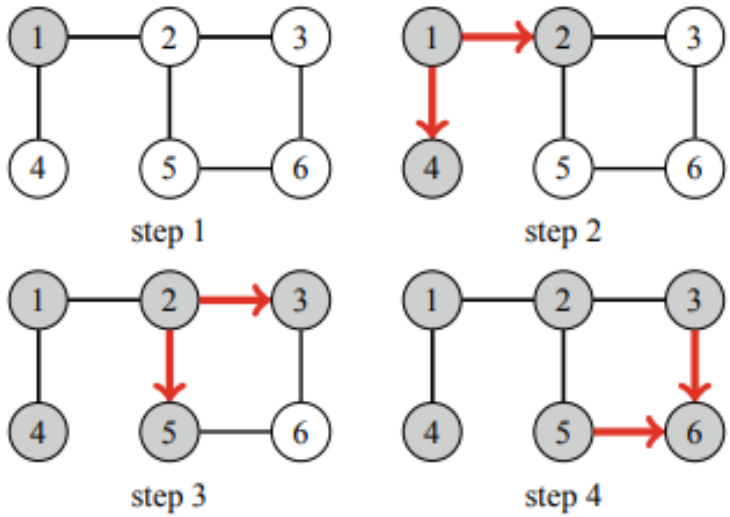
```
void dfs(int s) {
    if (visited[s]) return;
    visited[s] = true;
    // process node s
    for (auto u: adj[s]) {
        dfs(u);
    }
}
```

El grafo es representado por *adj*, una lista de adyacencias. Visited es un vector de booleanos.

BFS

- Recorrido “a lo ancho”.
- Va explorando el grafo en orden creciente de distancia desde un origen.

Fig. 7.14 Breadth-first search



Código BFS

```
queue<int> q;
bool visited[N];
int distance[N];
```

```
visited[x] = true;
distance[x] = 0;
q.push(x);
while (!q.empty()) {
    int s = q.front(); q.pop();
    // process node s
    for (auto u : adj[s]) {
        if (visited[u]) continue;
        visited[u] = true;
        distance[u] = distance[s]+1;
        q.push(u);
    }
}
```

Outline

Camino Mínimo

- Un problema recurrente es encontrar el camino de mínimo costo entre dos vértices.
- Hoy menciono cuatro algoritmos para resolver este problema:
  - BFS,
  - Bellman Ford,
  - Dijkstra y
  - Floyd-Warshall.

Bellman Ford

- En grafos pesados necesitamos otros algoritmos.
- Bellman Ford calcula camino mínimo desde un vértice hacia todos.
- Si hay ciclos negativos, el problema del camino mínimo se indefine. Bellman Ford detecta si hay un ciclo negativo.
- **Invariante:** luego de su k-ésima iteración, calcula la distancia mínima a todo v usando a lo sumo k aristas[1].
- **Complejidad:** $O(nm)$.

[1] En realidad, no exactamente...

Código Bellman Ford

```
for (int i = 1; i <= n; i++) {
    distance[i] = INF;
}
distance[x] = 0;
for (int i = 1; i <= n-1; i++) {
    for (auto e : edges) {
        int a, b, w;
        tie(a, b, w) = e;
        distance[b] = min(distance[b], distance[a]+w);
    }
}
```

Dijkstra

- Camino mínimo de uno a todos.
- Requiere que el costo de las aristas sea no negativo.
- **Invariante:** antes de la k-ésima iteración, calcula correctamente la distancia hacia los k vértices más cercanos al origen.
- En cada iteración, toma al vértice no procesado que esté a distancia mínima del origen.
- **Complejidad:** $O(min\{n^2, m\, lg\, n\})$.

Floyd-Warshall

- Computa distancia entre todo par de vértices.
- **Invariante:** luego de la k-ésima iteración, computa los caminos k-internos[2] mínimos.
- **Complejidad:** $O(n^3)$.

```
for (int k = 1; k <= n; k++) {
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            dist[i][j] = min(dist[i][j],dist[i][k]+dist[k][j]);
        }
    }
}
```

[2]Decimos que un camino es k-interno si todos los vértices intermedios (o sea, exluyendo al primero y al último) tienen un índice menor o igual a k

Tips

- Los algoritmos de 1 a todos se pueden transformar en algoritmos para encontrar caminos de todos a uno (revirtiendo el sentido de los ejes).
- Se pueden usar para computar caminos de todos a todos (haciendo n ejecuciones del algoritmo, una desde cada origen).
- Pueden usar los algoritmos para calcular distancias y también para obtener el árbol de caminos mínimos. Ojo, puede haber varios caminos mínimos, sólo alguno pertenece a este árbol.
- Si sienten que les falta información en el grafo, consideren agregar vértices para codificar esa información faltante.

Outline

Árbol Generador Mínimo

- Otro problema recurrente.
- Un árbol generador (AG) contiene a todos los nodos del grafo, a algunas $(n-1)$ de sus aristas y conecta a todos los vértices.
- El costo de un AG es la suma de los costos de las aristas que contiene.
- Comento dos algoritmos que resuelven el problema:
  - Kruskal.
  - Prim.

prim.floyd Siguiendo Enviar mensaje

36 publicaciones **347** seguidores **33** seguidos

Prim Floyd
💻 Equipo de programación competitiva
Facultad de ciencias exactas y naturales (FCEN - UBA)
EG En el Mundial de... más

https://www.instagram.com/prim.floyd/

Kruskal

- Inicializa con todos los vértices, y ninguna arista.
- Agrega golosamente a la arista de costo mínimo entre las que no generan ciclos.
- **Invariante:** luego de la i-ésima iteración, arma un bosque generador mínimo de i aristas que es subgrafo de algún AGM.
- **Otro invariante:** mantiene un bosque generador de i aristas mínimo.
- **Complejidad:** $O(min\{n^2, m\, lg\, n\})$[3].

[3] La implementación en $O(n^2)$ es poco desconocida.

Implementación

- Se ordenan las arista por costo, de menor a mayor.
- Se procesa cada arista u, v: si u y v están en distintas componentes, se agrega la arista al bosque.
- Para responder eficientemente si están en la misma componente, usamos la estructura union-find.

Prim

- Inicializa con un sólo vértice.
- Agrega golosamente a la arista de costo mínimo entre las que agregan un nuevo vértice al árbol.
- **Invariante:** luego de la i-ésima iteración, arma un bosque generador mínimo de i aristas que es subgrafo de algún AGM.
- **Complejidad:** $O(min\{n^2, m\, lg\, n\})$.

Referencias

La clase está fuertemente basada en Guide to Competitive Programming, de Antti Laaksonen.

Outline

Problemitas

- Les dejo problemas de CSES:
  - `https://cses.fi/problemset/task/1667`
  - `https://cses.fi/problemset/task/1671`
  - `https://cses.fi/problemset/task/1669`
  - `https://cses.fi/problemset/task/1672`
  - `https://cses.fi/problemset/task/1673`
  - `https://cses.fi/problemset/task/1195`
  - `https://cses.fi/problemset/task/1675`
  - `https://cses.fi/problemset/task/1676`