Grafos

Isabel Pita.

Facultad de Informática - UCM

16 de septiembre de 2021

Grafos

- Un *grafo* es un conjunto de vértices conectados por aristas.
- Un *camino* es una secuencia de vértices conectados por aristas.
- Un *ciclo* es un camino cuyo primer y último vértice coinciden.
- Un grafo es conexo si existe un camino entre cualquier par de vértices.

Grafos

- Una componente conexa de un grafo no dirigido es un subgrafo conexo maximal.
- Un grafo es bipartito si sus vértices se pueden dividir en dos conjuntos tales que las aristan conectan vértices de un conjunto con vértices del otro conjunto.

Tipos de grafos:

- Grafos dirigidos vs grafos no dirigidos
- Grafos valorados vs grafos no valorados

Representación de los grafos

Sean V - número de vértices, E - número de aristas.

- Listas de adyacentes. Cada nodo guarda la lista de sus adyacentes.

```
// indice del adyacente y peso de la arista
using vii = vector<pair<int,int>>;
vector<vii> adjList;
```

  - Es la forma más eficiente de obtener la lista de vértices adyacentes a uno dado.

Representación de los grafos

- Matriz de adyacencia. `adjMat[V][V]`.
  - Es inviable con $V > 1000$.
  - Obtener la lista de adyacentes de un nodo tiene coste $\mathcal{O}(V)$.
  - Se utiliza en el algoritmo de Floyd (cálculo de caminos mínimos).
- Lista de aristas.
  - No debe utilizarse para obtener los vértices adyacentes.
  - Se utiliza en el algoritmo de Kruskal (cálculo del árbol de recubrimiento mínimo (MST)).

Algoritmos sobre grafos.

- Depth-First Search (DFS). Grafos dirigidos y no dirigidos.
  - Buscar componentes conexas
  - Comprobar si un grafo es bipartito
- Breath-First Search (BFS). Grafos dirigidos y no dirigidos
  - Encontrar el camino mínimo entre dos puntos en un grafo no valorado
  - Buscar componentes conexas
  - Comprobar si un grafo es bipartito
- Algoritmo de Dijkstra. Grafos dirigidos y valorados.
  - Encontrar el camino mínimo desde un vértice a todos los demás.
  - Puede aplicarse sobre grafos no dirigidos, creando aristas en los dos sentidos
  - Si se aplica sobre el grafo inverso encuentra caminos mínimos desde todos los vértices a uno dado
- Algoritmo de Floyd. Grafos dirigidos y valorados
  - Encontrar el camino mínimo entre cada par de vértices.

Recorrido en profundidad. DFS

```
using vi = vector<int>;
using vvi = vector<vi>;
vvi adjList; int V, E;
bool visited[MAX]; // Inicializar en la funcion que llam

// Obtiene el numero de nodos que se recorren
int dfs(int v) {
    int tam = 1; visited[v] = true;
    for (int w : adjList[v])
      if (!visited[w]) tam += dfs(w);
    return tam;
}
```

Complejidad $\mathcal{O}(V + E)$

Inicializar el vector `visited` en la función que llama a dfs.

Recorrido en anchura. BFS

```
vi dist;
// Calcula la distancia de los vertices al origen s
void bfs(int s) {
  queue<int> q;
  dist[s] = 0; visited[s] = true;
  q.push(s);
  while (!q.empty()) {
    int v = q.front(); q.pop();
    for (int w : adjList[v])
      if (!visited[w]) {
        dist[w] = dist[v] + 1;
        visited[w] = 1;
        q.push(w);
      }
  }
}
```

Complejidad $\mathcal{O}(V + E)$
Inicializar los vectores `dist` y `visited` en la función que llama a bfs.

Los amigos de mis amigos son mis amigos

En esta ciudad vive una serie de personas, y sabemos que algunas de ellas son amigas entre sí. De acuerdo con el refrán que dice "*Los amigos de mis amigos son mis amigos*", sabemos que si A y B son amigos y B y C son amigos, entonces también son amigos A y C.

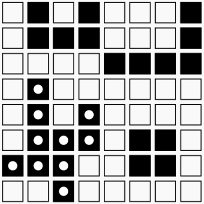
Tu misión consiste en contar las personas en el grupo de amigos más grande.

Los amigos de mis amigos

```
void resuelveCaso () {
    std::cin >> V >> E;
    adjList.assign(V,{});
    for (int i = 0; i < E; ++i) {
        int v1, v2; std::cin >> v1 >> v2;
        adjList[v1-1].push_back(v2-1); // no dirigido
        adjList[v2-1].push_back(v1-1);
    }
    memset(visited,0,V);
    int tamano = 0;
    for (int i = 0; i < V; ++i) {
        if (!visited[i]) { // Nueva componente conexa
            int tam = dfs(i);
            if (tam > tamano) tamano = tam;
        }
    }
    std::cout << tamano << '\n';
}
```

Detección de manchas negras

Dado un *bitmap* de píxeles blancos y negros, queremos saber el número de *manchas negras* que contiene y el tamaño (número de píxeles) de la mancha negra más grande. Dos píxeles negros pertenecen a la misma mancha si se puede pasar de uno a otro atravesando solamente píxeles negros y moviéndonos píxel a píxel solamente en horizontal o vertical.



En esta imagen aparecen 4 manchas y la mancha más grande (marcada con puntos blancos) tiene 10 píxeles.

Detección de manchas negras

```
const int MAX = 1000;
int df[] = {1,0,-1,0}, dc[] = {0,1,0,-1};
bool visited[MAX][MAX];
string mapa[MAX];
int F,C;

bool ok(int i, int j) {
    return 0 <= i && i < F && 0 <= j && j < C;
}

int dfs(int i, int j) {
  int tam = 1; visited[i][j] = true;
  for (int k = 0; k < 4; ++k) {
    int ni = i + df[k],  nj = j + dc[k];
    if (ok(ni,nj)&&mapa[ni][nj]=='#'&&!visited[ni][nj])
      tam += dfs(ni, nj);
  }
  return tam;
}
```

Detección de manchas negras

```
bool resuelveCaso() {
// Lectura de los datos y del mapa
...
// Inicializa matriz de  marcas
    for (int i = 0; i < F; ++i)
        for (int j = 0; j < C; ++j)
            visited[i][j] = false;
    int numero = 0, maximo = 0;
    for(int i = 0; i < F; ++i)
        for (int j = 0; j < C; ++j)
            if (!visited[i][j] && mapa[i][j] == '#') {
                maximo = max(maximo, dfs(i, j));
                ++numero;
            }
    cout << numero << ' ' << maximo << '\n';
    return true;
}
```

Algoritmo de Dijkstra

Dado un grafo dirigido y valorado (valores positivos), encontrar el camino mínimo desde un vértice a todos los demás.

```
void dijkstra(int s, vi &dist) {
  dist.assign(adjList.size(), INF);
  dist[s] = 0;
  priority_queue<ii, vii, greater<ii>> pq;
  pq.push({0, s});
  while (!pq.empty()) {
    ii front = pq.top(); pq.pop();
    int d = front.first, u = front.second;
    if (d > dist[u]) continue;
    for (auto a : adjList[u]) {
      if (dist[u] + a.first < dist[a.second]) {
        dist[a.second] = dist[u] + a.first;
        pq.push({dist[a.second], a.second});
      }}}}
```

coste: $\mathcal{O}((V+E)\log V)$.

Algoritmo de Dijkstra

Haciendo el grafo inverso podemos calcular el camino mínimo desde todos los puntos a un destino.

```
vvi inverso(V);
for (int v = 0; v < V; ++v) {
    for (int w : adjList[v]){
        inverso[w].push_back(v);
    }
}
```

Algoritmo de Floyd

- Dado un grafo dirigido y valorado (valores positivos) encontrar el camino mínimo entre cada par de vértices.
- El grafo está dado con la matriz de adyacencia.

```
int camino[MAX][MAX];

void floyd() {
  for (int k = 0; k < V; k++)
    for (int i = 0; i < V; i++)
      for (int j = 0; j < V; j++)
        if (adjMat[i][k]+adjMat[k][j]<adjMat[i][j]) {
          adjMat[i][j] = adjMat[i][k] + adjMat[k][j];
          camino[i][j] = k;
        }
}
```

Coste: $\mathcal{O}(V^3)$.

Cuestiones

- ¿Cuantas aristas tiene un grafo conexo y acíclico de V vértices?
- Calcula la complejidad de los algoritmos DFS y BFS si se utiliza como representación del grafo la matriz de adyacencia o si se utiliza como representación las listas de adyacentes.
- Dado un grafo bipartito (sin aristas duplicadas y sin autoaristas) con V vértices, ¿Cúal es el número máximo de aristas que puede tener?
- Probar que un grafo bipartito no puede tener un ciclo con un número impar de aristas.
- Dado un grafo dirigido y valorado con valores mayores que 10, ¿en cuales de los siguientes casos el camino mínimo entre dos vértices u, v nunca cambia?
    1. Se suma a todos los valores 10
    2. Se resta a todos los valores 10
    3. Se multiplican todos los valores por 10
    4. Se dividen todos los valores por 10