DFS: tipos de aristas

Programación Competitiva

Marco Antonio Gómez Martín

Facultad de Informática - UCM

Recorrido en profundidad. DFS

```
using vi = vector<int>;
using vvi = vector<vi>;
vvi adjList; int V, E;
bool visited[MAX];

// Obtiene el numero de nodos que se recorren
int dfs(int u) {
    int tam = 1; visited[v] = true;
    for (int v : adjList[u]) {
      // Procesamos arista u->v
      if (!visited[v]) {
        // Nos lleva a un vértice nuevo, recorremos
        tam += dfs(v);
      }
    }
    return tam;
}
```

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

Tipos de aristas de un recorrido DFS

- *Tree edges*: aristas que forman el *árbol* del DFS.
- *Back edges*: aristas a vértices desde el vértice hacia la raíz.
- *Forward edges*: aristas a vértices alcanzables desde algún hijo ya explorado.
- *Cross edges* (solo en grafos dirigidos): aristas a otros vértices ya explorados con anterioridad.

Búsqueda de ciclos en grafos dirigidos

Programación Competitiva

Marco Antonio Gómez Martín

Facultad de Informática - UCM

Búsqueda de ciclos

En un grafo **no** dirigido:

- En cuanto haya una arista, hay un ciclo trivial (u - v - u).
- Es fácil saber si hay ciclos no triviales:
  - Contando vértices y aristas de las componentes conexas
  - Comprobando si en un recorrido llegamos a un vértice ya visitado antes (que no sea nuestro padre).

Búsqueda de ciclos

En un grafo **dirigido**:

- Hay un ciclo si en el DFS hay una *back edge*.

Búsqueda de ciclos en grafo dirigido

```
// true si hay algún ciclo.
// Termina en cuanto encuentra uno
bool dfs(int u) {

  estado[u] = TOCADO;

  for (int i = 0; i < adj[u].size(); ++i) {
   int v = adj[u][i];
   if (estado[v] == TOCADO)
    return true;
   else if ((estado[v] == NO_VISTO) && dfs(v))
    return true;
  }
  estado[u] = HUNDIDO;
  return false;
}
```

Búsqueda de puentes

Programación Competitiva

Marco Antonio Gómez Martín

Facultad de Informática - UCM

Puentes en un grafo no dirigido

Aristas que, si se eliminan, hacen crecer el número de componentes conexas del grafo. Algoritmo:

- Recorrer el grafo usando DFS
- Una arista $u \to v$ es un puente si las *back edges* del subárbol de v no salen de él.

Puentes

```
int hora; int horaVertice[MAX_V]; int alcanzable[MAX_V];

void dfs(int u, int uParent) {
  horaVertice[u] = alcanzable[u] = hora; hora++;









}
```

Puentes

```
int hora; int horaVertice[MAX_V]; int alcanzable[MAX_V];

void dfs(int u, int uParent) {
  horaVertice[u] = alcanzable[u] = hora; hora++;
  for (int i = 0; i < adj[u].size(); ++i) {
   int v = adj[u][i];









  }
}
```

Puentes

```
int hora; int horaVertice[MAX_V]; int alcanzable[MAX_V];

void dfs(int u, int uParent) {
  horaVertice[u] = alcanzable[u] = hora; hora++;
  for (int i = 0; i < adj[u].size(); ++i) {
   int v = adj[u][i];
   if (v == uParent) continue; // B.E. con el padre, que ignoramos









  }
}
```

Puentes

```
int hora; int horaVertice[MAX_V]; int alcanzable[MAX_V];

void dfs(int u, int uParent) {
  horaVertice[u] = alcanzable[u] = hora; hora++;
  for (int i = 0; i < adj[u].size(); ++i) {
   int v = adj[u][i];
   if (v == uParent) continue; // B.E. con el padre, que ignoramos
   if (horaVertice[v] == 0) { // No visitado









   } else
    alcanzable[u] = min(alcanzable[u], horaVertice[v]); // B.E.
  }
}
```

Puentes

```
int hora; int horaVertice[MAX_V]; int alcanzable[MAX_V];

void dfs(int u, int uParent) {
  horaVertice[u] = alcanzable[u] = hora; hora++;
  for (int i = 0; i < adj[u].size(); ++i) {
   int v = adj[u][i];
   if (v == uParent) continue; // B.E. con el padre, que ignoramos
   if (horaVertice[v] == 0) { // No visitado
    dfs(v, u);



    alcanzable[u] = min(alcanzable[u], alcanzable[v]);
   } else
    alcanzable[u] = min(alcanzable[u], horaVertice[v]); // B.E.
  }
}
```

Puentes

```
int hora; int horaVertice[MAX_V]; int alcanzable[MAX_V];

void dfs(int u, int uParent) {
  horaVertice[u] = alcanzable[u] = hora; hora++;
  for (int i = 0; i < adj[u].size(); ++i) {
   int v = adj[u][i];
   if (v == uParent) continue; // B.E. con el padre, que ignoramos
   if (horaVertice[v] == 0) { // No visitado
    dfs(v, u);
    if (alcanzable[v] > horaVertice[u]) {
      // La arista u-v es un puente
    }
    alcanzable[u] = min(alcanzable[u], alcanzable[v]);
   } else
    alcanzable[u] = min(alcanzable[u], horaVertice[v]); // B.E.
  }
}
```

Puentes

```
int hora; int horaVertice[MAX_V]; int alcanzable[MAX_V];

void dfs(int u, int uParent) {
  horaVertice[u] = alcanzable[u] = hora; hora++;
  for (int i = 0; i < adj[u].size(); ++i) {
   int v = adj[u][i];
   if (v == uParent) continue; // B.E. con el padre, que ignoramos
   if (horaVertice[v] == 0) { // No visitado
    dfs(v, u);
    if (alcanzable[v] > horaVertice[u]) {
      // La arista u-v es un puente
    }
    alcanzable[u] = min(alcanzable[u], alcanzable[v]);
   } else
    alcanzable[u] = min(alcanzable[u], horaVertice[v]); // B.E.
  }
}
```

Puentes

```
int hora; int horaVertice[MAX_V]; int alcanzable[MAX_V];

void dfs(int u, int uParent) {
  horaVertice[u] = alcanzable[u] = hora; hora++;
  for (int i = 0; i < adj[u].size(); ++i) {
   int v = adj[u][i];
   if (v == uParent) continue; // B.E. con el padre, que ignoramos
   if (horaVertice[v] == 0) { // No visitado
    dfs(v, u);
    if (alcanzable[v] > horaVertice[u]) {
      // La arista u-v es un puente
    }
    alcanzable[u] = min(alcanzable[u], alcanzable[v]);
   } else
    alcanzable[u] = min(alcanzable[u], horaVertice[v]); // B.E.
  }
}
```

Puentes

En la llamada:

```
hora = 1;
memset(horaVertice, 0, n*sizeof(horaVertice[0]));
//memset(alcanzable, -1, n*sizeof(alcanzable[0]));

for (int i = 1; i <= n; ++i) {
  if (!horaVertice[i]) {
   dfs(i, 0);
}
```

Ejercicio propuesto

- 796 - Critical links

Puntos de articulación

Programación Competitiva

Marco Antonio Gómez Martín

Facultad de Informática - UCM

Puntos de articulación

Un punto de articulación de un grafo es un *vértice* que, si se elimina, hace crecer el número de componentes conexas.
Algoritmo:

- Recorrer el grafo usando DFS
- Un vértice u es punto de articulación si las *back edges* de los subárboles DFS de alguno de los hijos no salen del subárbol de u.
- La raíz del recorrido DFS es un caso especial que debe considerarse aparte.

Puentes

```
int hora; int horaVertice[MAX_V]; int alcanzable[MAX_V];

void dfs(int u, int uParent) {
  horaVertice[u] = alcanzable[u] = hora; hora++;
  for (int i = 0; i < adj[u].size(); ++i) {
   int v = adj[u][i];
   if (v == uParent) continue; // B.E. con el padre, que ignoramos
   if (horaVertice[v] == 0) { // No visitado
    dfs(v, u);
    if (alcanzable[v] > horaVertice[u]) {
      // La arista u-v es un puente
    }
    alcanzable[u] = min(alcanzable[u], alcanzable[v]);
   } else
    alcanzable[u] = min(alcanzable[u], horaVertice[v]); // B.E.
  }
}
```

Puntos de articulación

```
int hora; int horaVertice[MAX_V]; int alcanzable[MAX_V];
int hijosRaiz;

void dfs(int u, int uParent) {
  horaVertice[u] = alcanzable[u] = hora; hora++;
  for (int i = 0; i < adj[u].size(); ++i) {
   int v = adj[u][i];
   if (v == uParent) continue;
   if (horaVertice[v] == 0) {
    if (uParent == 0)  hijosRaiz++;
    dfs(v, u);
    if (alcanzable[v] >= horaVertice[u])
      if (uParent != 0) {
       // u es punto de articulación.
      }
    alcanzable[u] = min(alcanzable[u], alcanzable[v]);
   } else
    alcanzable[u] = min(alcanzable[u], horaVertice[v]);
  }
```

Puntos de articulación

En la llamada:

```
hora = 1;
memset(horaVertice, 0, n*sizeof(horaVertice[0]));
memset(alcanzable, -1, n*sizeof(alcanzable[0]));

for (int i = 1; i <= n; ++i) {
  if (!horaVertice[i]) {
   hijosRaiz = 0;
   dfs(i, 0);
   if (hijosRaiz > 1)
       // La raiz es punto de articulación
  }
}
```