Semillero de Programación

Ordenamiento Topológico, Componentes Fuertemente Conexas y Estructuras de Datos

Ana Echavarría   Juan Francisco Cardona

Universidad EAFIT

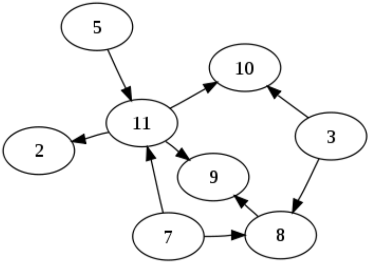
8 de marzo de 2013

Contenido

DAG

DAG

Un DAG (Directed Acyclic Graph) es un grafo dirigido que no tiene ciclos.



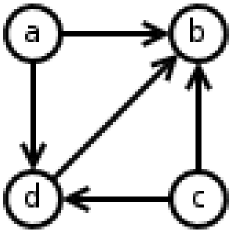
Ordenamiento Topológico

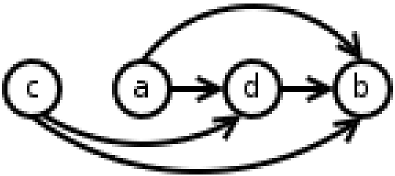
Ordenamiento Topológico

Un ordenamiento topológico o topological sort de un DAG $G = (V, E)$ es un ordenamiento lineal de sus nodos V de tal forma que si $(u, v) \in E$ entonces u aparece antes que v en el ordenamiento.

Este ordenamiento se puede ver como una forma de poner todos los nodos en una línea recta y que las aristas vayan todas de izquierda a derecha.

Ejemplo





Algoritmo

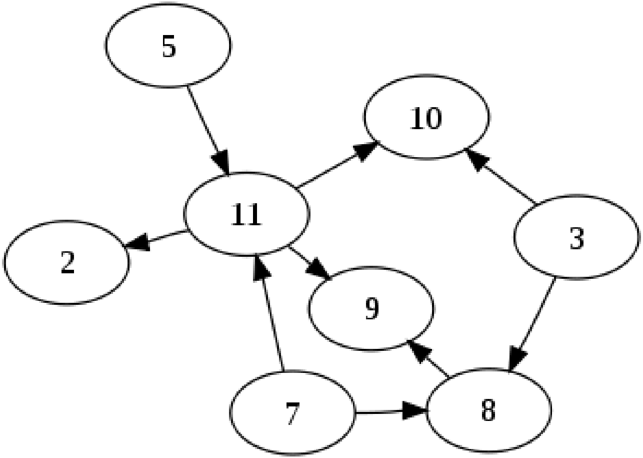
1. Hacer un DFS con el grafo
2. Cuando marco un nodo como negro, lo inserto a un vector
3. Reversar el orden de los elementos del vector
4. El vector contiene un ordenamiento topológico del grafo

Algoritmo

```
vector <int> g[MAXN];
bool seen[MAXN];
vector <int> topo_sort;

void dfs(int u){
   seen[u] = true;
   for (int i = 0; i < g[u].size(); ++i){
      int v = g[u][i];
      if (!seen[v]) dfs(v);
   }
   topo_sort.push_back(u); // Agregar el nodo al vector
}
int main(){
   // Build graph: n = verices
   topo_sort.clear();
   for (int i = 0; i < n; ++i) seen[i] = false;
   for (int i = 0; i < n; ++i) if (!seen[i]) dfs(i);
   reverse(topo_sort.begin(), topo_sort.end());
   return 0;
}
```

Ejemplo



¿Por qué funciona?

- Cuando meto un nodo a la lista, es porque ya procesé todos sus vecinos.
- Si ya procesé todos sus vecinos, ellos ya están en la lista.
- Cuando meto un nodo a la lista, todos sus vecinos ya están antes que él en la lista, entonces en el ordenamiento van a quedar después de él.
- En conclusión, en el ordenamiento que generamos, los vecinos de cada nodo van a estar después de él por lo que es un ordenamiento topológico.

Complejidad

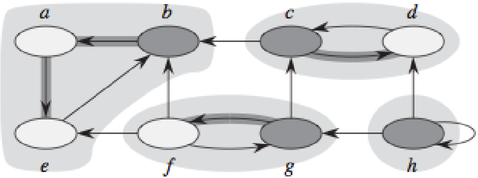
Complejidad

Hacer el ordenamiento topológico toma $O(V+E)$ para el dfs y $O(V)$ para reversar la lista. En total la complejidad es $O(V+E)$.

Componentes Fuertemente Conexas

SCC

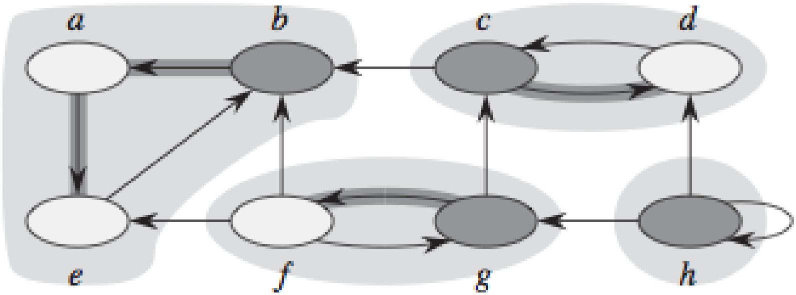
Dado un grafo dirigido $G = (V, E)$, un componente fuertemente conexa o strongly connected component (SCC) de G es un subconjunto C de nodos que cumple que para cada pareja $u, v \in C$ existe un camino de u a v y de v a u y que C es lo más grande posible.



Algoritmo

1. Crear el grafo G y el grafo G_{rev} que es el mismo que G pero con las aristas invertidas.
2. Hacer DFS en el grafo G y generar su "ordenamiento topológico" (incluir un nodo a la lista solo cuando haya visto todos los nodos alcanzables desde él.)
3. Hacer un DFS en el grafo reversado G_{rev} pero hacer las llamadas en el orden del "ordenamiento topológico".
4. Cada llamado a este último DFS halla una componente fuertemente conexa.

Ejemplo



¿Por qué funciona? I

1. Las componentes fuertemente conexas de G son las mismas que las de G_{rev}.
2. Si comprimo los nodos de una misma componente en un solo nodo, quedo con un DAG.
3. Si tengo dos componentes distintas C_1 y C_2 de manera que haya una arista de un nodo de C_1 a un nodo de C_2, entonces todos los nodos de C_1 van a quedar después que los nodos de C_2 en el "ordenamiento topológico" que se hace con el primer DFS.

¿Por qué funciona? II

4. Los nodos que quedan de primeros en el "ordenamiento topológico" son los nodos de una componente C a la cual no llega ninguna arista.
5. En el grafo G_{rev}, de la componente C no sale ninguna arista.
6. Cuando llamo el segundo DFS lo hago desde C y sólo descubro los elementos de C.
7. Cuando llamo el segundo DFS desde otro nodo este puede no tener aristas salientes o tener aristas salientes a C pero como ya descubrí todo en C sólo voy a descubrir lo que hay en la componente de ese nodo

Problema 11504 - Dominos

Problema

Hallar el mínimo número de dominós que hay que derribar a mano para que todos los dominós se derriben.

Ideas

- ¿Qué pasa con las cadenas de dominós que forman un ciclo? ¿Cuántos necesito máximo para derribarlas?

Ideas

Ideas

- ¿Qué pasa con las cadenas de dominós que forman un ciclo? ¿Cuántos necesito máximo para derribarlas?
- ¿Puedo entonces considerar los ciclos como un sólo dominó? ¿Qué algoritmo estoy utilizando?

Ideas

Ideas

- ¿Qué pasa con las cadenas de dominós que forman un ciclo? ¿Cuántos necesito máximo para derribarlas?
- ¿Puedo entonces considerar los ciclos como un sólo dominó? ¿Qué algoritmo estoy utilizando?
- ¿En el grafo que se forma cuando uno los elementos de una misma componentes cuántos dominós tengo que derribar?

Solución

1. Crear el grafo dirigido y su grafo invertido
2. Hallar la componente fuertemente conexa de cada nodo
3. Hallar cuantas aristas llegan a cada componente conexa
4. Contar cuantas componentes hay a las cuales no lleguen aristas

Variables globales

```
// El maximo numero de dominos
const int MAXN = 100005;
// El grafo
vector <int> g[MAXN];
// El grafo reversado
vector <int> grev[MAXN];
// El "ordenamiento topologico" del grafo G
vector <int> topo_sort;
// La componente fuertemente conexa a la que pertenece cada nodo
int scc[MAXN];
// El arreglo de visitado para el primer DFS
bool seen[MAXN];
 // El numero de aristas entrantes a cada componente
int in[MAXN];
```

DFS

```
// DFS donde se halla el ordenamiento topologico
void dfs1(int u){
   seen[u] = true;
   for (int i = 0; i < g[u].size(); ++i){
      int v = g[u][i];
      if (!seen[v]) dfs1(v);
   }
   topo_sort.push_back(u);
}
// DFS donde se hallan las componentes
void dfs2(int u, int comp){
   scc[u] = comp;
   for (int i = 0; i < grev[u].size(); ++i){
      int v = grev[u][i];
      if (scc[v] == -1) dfs2(v, comp);
   }
}
```

Main I

```
int main(){
   int cases; cin >> cases;
   while (cases--){
      int n, m;
      cin >> n >> m;

      // Limpiar las variables entre caso y caso
      for (int i = 0; i <= n; ++i){
         g[i].clear();
         grev[i].clear();
         topo_sort.clear();
         scc[i] = -1;
         seen[i] = false;
         in[i] = 0;
      }



```

Main II

```
    // Crear el grafo y el grafo reversado
    for (int i = 0; i < m; ++i){
        int u, v; cin >> u >> v;
        u--; v--;
        g[u].push_back(v);
        grev[v].push_back(u);
    }

    // Llamar el primer dfs
    for (int i = 0; i < n; ++i){
        if (!seen[i]) dfs1(i);
    }
    reverse(topo_sort.begin(), topo_sort.end());
    // Llamar el segundo dfs
    int comp = 0;
    for (int i = 0; i < n; ++i){
        int u = topo_sort[i];
        if (scc[u] == -1) dfs2(u, comp++);
    }
```

Main III

```

    // Ver cuantas aristas entrantes tiene cada componente
    for (int u = 0; u < n; ++u){
       for (int i = 0; i < g[u].size(); ++i){
          int v = g[u][i];
          if (scc[u] != scc[v]) in[scc[v]]++;
       }
    }

    // Sumar las componentes que tienen 0 aristas entrantes
    int count = 0;
    for (int u = 0; u < comp; ++u){
       if (in[u] == 0) count++;
    }
    cout << count << endl;
  }
  return 0;
}
```