Semillero de Programación

Problemas con DFS, BFS, Componentes Fuertemente Conexas y Ordenamiento Topológico

Ana Echavarría Juan Francisco Cardona

Universidad EAFIT

1 - 8 marzo de 2013

Contenido

1. Bicoloring
2. Playing with Wheels
3. Ordenamiento Topológico
4. Componentes Fuertemente Conexas
5. Dominos

Problema 10004 - Bicoloring

Problema

Verificar si un grafo es bipartito, es decir, si se pueden usar dos colores para pintar todos los nodos de manera que dos nodos vecinos no tengan el mismo color

¿Qué técnica usar?

Pregunta

- ¿Cómo hago para verificar que el grafo sea bipartito?

¿Qué técnica usar?

Pregunta

- ¿Cómo hago para verificar que el grafo sea bipartito?
  - Pinto el primer nodo de un color y todos sus vecinos de otro color y repito el proceso con los vecinos.

¿Qué técnica usar?

Pregunta

- ¿Cómo hago para verificar que el grafo sea bipartito?
  - Pinto el primer nodo de un color y todos sus vecinos de otro color y repito el proceso con los vecinos.
- ¿Qué pasa si tengo que pintar un nodo que ya pinté antes?

¿Qué técnica usar?

Pregunta

- ¿Cómo hago para verificar que el grafo sea bipartito?
  - Pinto el primer nodo de un color y todos sus vecinos de otro color y repito el proceso con los vecinos.
- ¿Qué pasa si tengo que pintar un nodo que ya pinté antes?
  - Si el color con el que lo tengo que pintar es el mismo que tiene no pasa nada, si no es así el grafo no es bipartito.
- ¿Qué técnica puedo usar para pintar cada nodo y luego sus vecinos?

¿Qué técnica usar?

Pregunta

- ¿Cómo hago para verificar que el grafo sea bipartito?
  - Pinto el primer nodo de un color y todos sus vecinos de otro color y repito el proceso con los vecinos.
- ¿Qué pasa si tengo que pintar un nodo que ya pinté antes?
  - Si el color con el que lo tengo que pintar es el mismo que tiene no pasa nada, si no es así el grafo no es bipartito.
- ¿Qué técnica puedo usar para pintar cada nodo y luego sus vecinos?
  - Se pueden usar BFS y DFS.

Solución

```
1  int main(){
2      int n, m;
3      while (cin >> n){
4          if (n == 0) break;
5          for (int i = 0; i < n; ++i){
6              g[i].clear();
7              color[i] = -1;
8          }
9          cin >> m;
10         for (int i = 0; i < m; ++i){
11             int u, v;
12             cin >> u >> v;
13             g[u].push_back(v);
14             g[v].push_back(u);
15         }
16         if (dfs(0, 0)) puts("BICOLORABLE.");
17         else puts("NOT BICOLORABLE.");
18     }
19      return 0;
20 }
```

Solución usando DFS

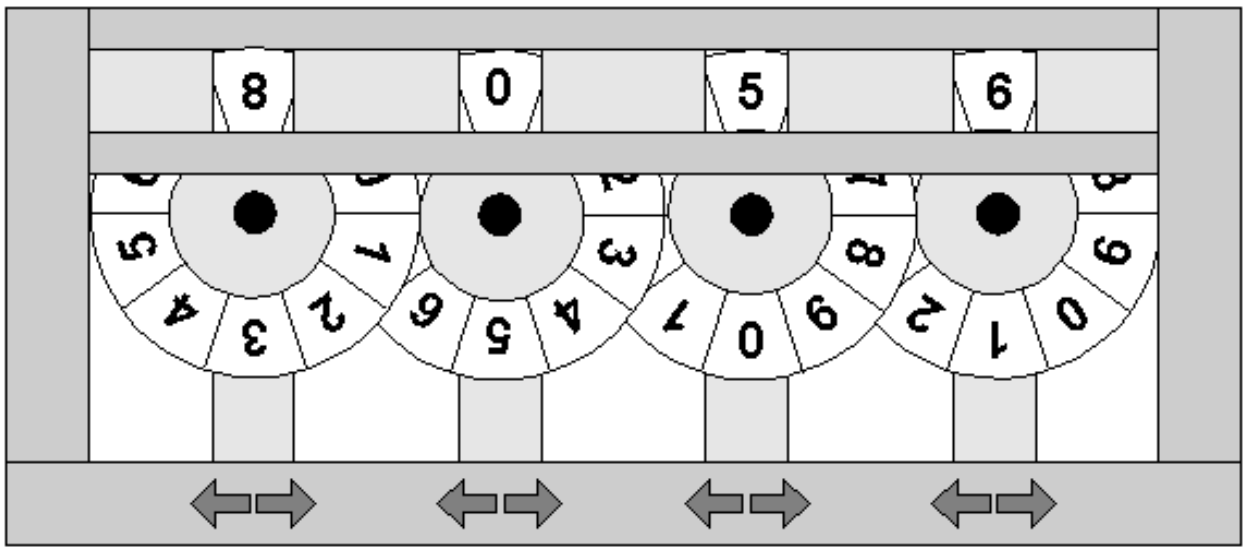
```
1  const int MAXN = 205;
2  vector <int> g[MAXN];
3  int color[MAXN];
4
5  bool dfs(int u, int paint){
6      color[u] = paint;
7      for (int i = 0; i < g[u].size(); ++i){
8          int v = g[u][i];
9          bool possible;
10         if (color[v] == -1) possible = dfs(v, 1 - paint);
11         else possible = (color[v] == (1 - paint));
12         if (!possible) return false;
13     }
14     return true;
15 }
```

Solución usando BFS

```
1  bool bfs(int s){
2     queue <int> q;
3     q.push(s);
4     color[s] = 0;
5     while (q.size() > 0){
6        int u = q.front(); q.pop();
7        for (int i = 0; i < g[u].size(); ++i){
8           int v = g[u][i];
9           if (color[v] == color[u]) return false;
10
11          if (color[v] == -1){
12             color[v] = 1 - color[u];
13             q.push(v);
14          }
15       }
16    }
17    return true;
18 }
```

Problema 10067 - Playing with Wheels

Se tiene una caja fuerte con 4 ruedas que indican cada una un número. Cada rueda tiene dos botones, uno mueve la rueda a la derecha (aumenta el número mostrado y si es 9 cambia al 0) y el otro mueve la rueda a la izquierda (disminuye el número mostrado y si es 0 cambia al 9).



Problema

- Se sabe cuál es la configuración inicial de las ruedas y cuál es la configuración final que abre la caja fuerte. Sin embargo, hay un conjunto de configuraciones prohibidas que no se pueden activar.
- El problema es hallar el **mínimo número de movimientos de las ruedas que hay que hacer para llegar de la configuración inicial a la final sin pasar por ninguna de las configuraciones prohibidas.**

¿Qué técnica usar?

Preguntas

1. ¿El problema se puede expresar como un problema de grafos?

¿Qué técnica usar?

Preguntas

1. ¿El problema se puede expresar como un problema de grafos?
2. ¿Cuáles serían los nodos?

¿Qué técnica usar?

Preguntas

1. ¿El problema se puede expresar como un problema de grafos?
2. ¿Cuáles serían los nodos?
3. ¿Cuándo se forma una arista? (¿cuándo se unen dos nodos?)

¿Qué técnica usar?

Preguntas

1. ¿El problema se puede expresar como un problema de grafos?
2. ¿Cuáles serían los nodos?
3. ¿Cuándo se forma una arista? (¿cuándo se unen dos nodos?)
4. ¿Cuántos nodos hay?

¿Qué técnica usar?

Preguntas

1. ¿El problema se puede expresar como un problema de grafos?
2. ¿Cuáles serían los nodos?
3. ¿Cuándo se forma una arista? (¿cuándo se unen dos nodos?)
4. ¿Cuántos nodos hay?
5. ¿Cuántas aristas hay?

¿Qué técnica usar?

Preguntas

1. ¿El problema se puede expresar como un problema de grafos?
2. ¿Cuáles serían los nodos?
3. ¿Cuándo se forma una arista? (¿cuándo se unen dos nodos?)
4. ¿Cuántos nodos hay?
5. ¿Cuántas aristas hay?
6. ¿El grafo cambia con cada caso de prueba o es independiente de los casos de prueba?

¿Qué técnica usar?

Preguntas

1. ¿El problema se puede expresar como un problema de grafos?
2. ¿Cuáles serían los nodos?
3. ¿Cuándo se forma una arista? (¿cuándo se unen dos nodos?)
4. ¿Cuántos nodos hay?
5. ¿Cuántas aristas hay?
6. ¿El grafo cambia con cada caso de prueba o es independiente de los casos de prueba?
7. ¿Cómo hallo el mínimo número de movimientos para llegar de un nodo al otro?

Representación del grafo

Cada nodo del grafo es un vector enteros de 4 posiciones, sin embargo en el algoritmo asumimos que los nodos son número enteros. ¿Hay alguna forma de representar estos nodos como números? ¿Si la hay, pueden dos nodos tener la misma representación?

Creación del grafo

```
1  const int MAXN = 10005;
2  // Find neighbour of node u if you move wheel d in direction dir
3  int find_neighbour(int u, int d, int dir){
4     vector <int> a(4);
5     for (int i = 0; i < 4; ++i){
6        a[i] = u % 10;
7        u /= 10;
8     }
9     a[d] = (a[d] + 10 + dir) % 10;
10
11    int ans = 0;
12    for (int i = 3; i >= 0; --i){
13       ans *= 10; ans += a[i];
14    }
15    return ans;
16 }
```

Creación del grafo y lectura números

```
1  vector <int> g[MAXN];
2  void make_graph(){ // Create out edges for all nodes
3     for (int i = 0; i <= 9999; ++i){
4        for (int d = 0; d < 4; ++d){
5           g[i].push_back(find_neighbour(i, d, -1)); // Move left
6           g[i].push_back(find_neighbour(i, d, +1)); // Move right
7        }
8     }
9  }
```

```
1  int get_num(){ // Read nodes and convert them to an integer
2     int ans = 0;
3     for (int i = 0; i < 4; ++i){
4        int d; cin >> d;
5        ans = ans * 10 + d;
6     }
7     return ans;
8  }
```

Lectura de los datos

```
1  bool forbidden[MAXN]; // The forbidden edges
2  int d[MAXN];              // The distance for start vertex
3
4  int main(){
5     make_graph();         // Create the graph
6     int cases; cin >> cases;
7     while (cases--){
8        for (int i = 0; i < MAXN; ++i) forbidden[i] = false;
9        int s = get_num(); // Read start vertex
10       int t = get_num(); // Read end vertex
11
12       int n; cin >> n;   // Read all forbidden vertices
13       while (n--) forbidden[get_num()] = true;
14
15       bfs(s);                  // Call bfs from the start vertex
16       cout << d[t] << endl; // Output distance to end vertex
17    }
18    return 0;
19 }
```

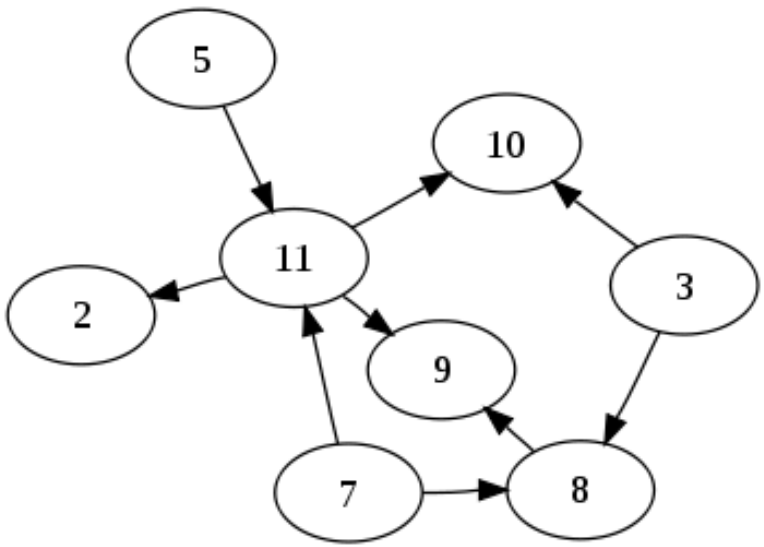
BFS

```
1  void bfs(int s){
2     for (int i = 0; i < MAXN; ++i) d[i] = -1;
3     queue <int> q;
4     q.push(s);
5     d[s] = 0;
6     while (q.size() > 0){
7        int cur = q.front(); q.pop();
8        for (int i = 0; i < g[cur].size(); ++i){
9           int next = g[cur][i];
10          // If not seen before and not forbidden add to queue
11          if (!forbidden[next] and d[next] == -1){
12             d[next] = d[cur] + 1;
13             q.push(next);
14          }
15       }
16    }
17 }
```

DAG

DAG

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



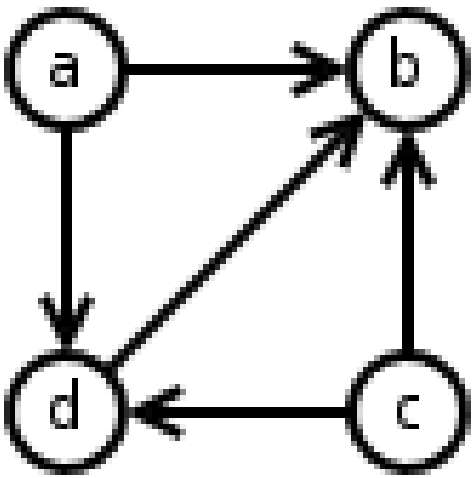
Ordenamiento Topológico

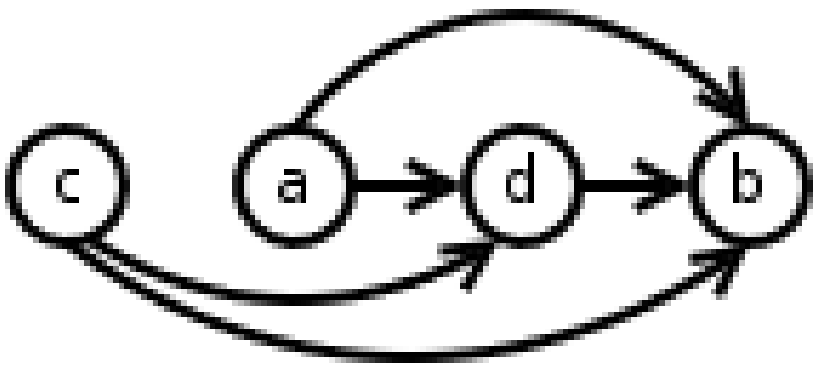
Ordenamiento Topológico

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





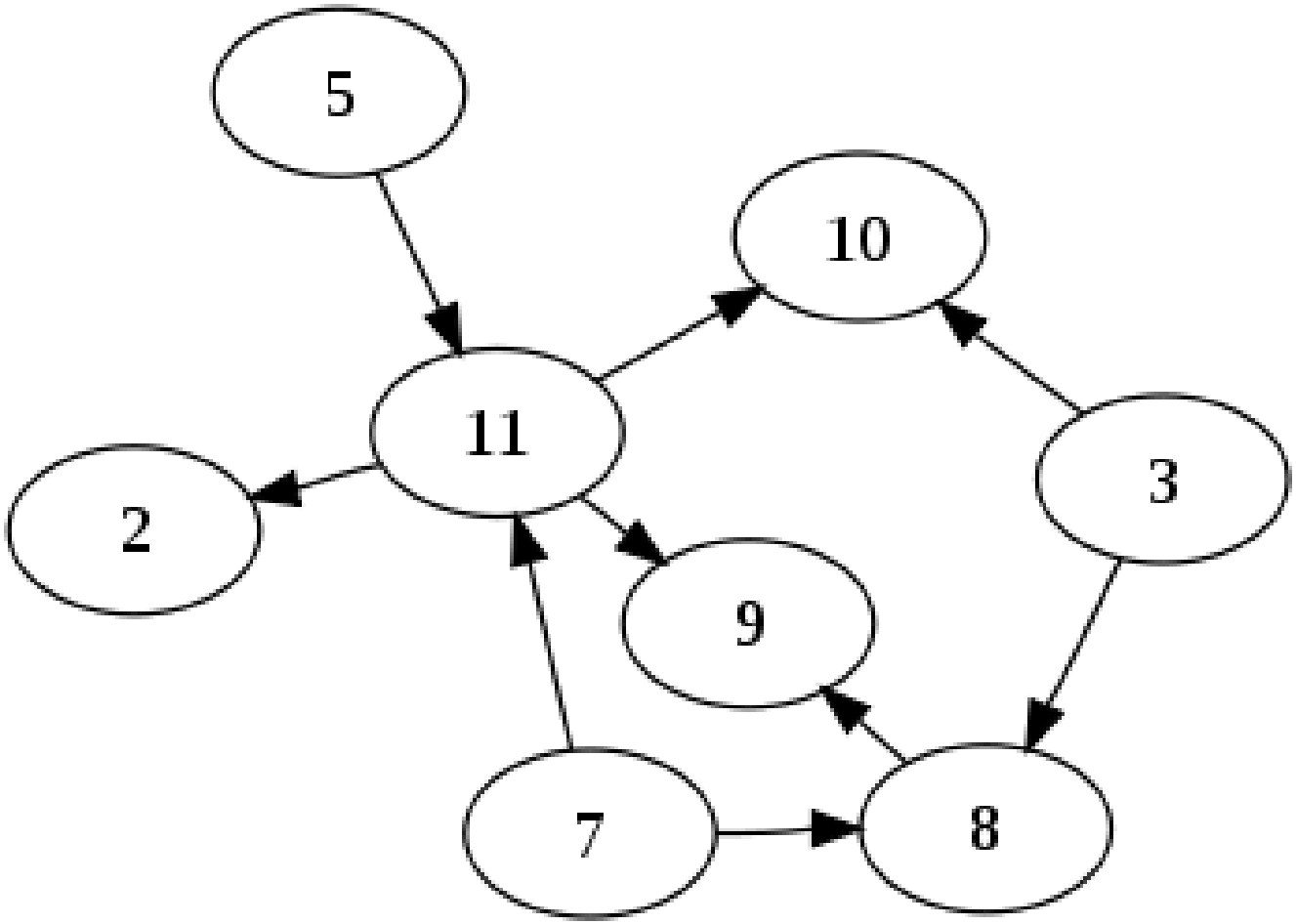
Algoritmo

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

```
1  vector <int> g[MAXN];
2  bool seen[MAXN];
3  vector <int> topo_sort;
4
5  void dfs(int u){
6     seen[u] = true;
7     for (int i = 0; i < g[u].size(); ++i){
8        int v = g[u][i];
9        if (!seen[v]) dfs(v);
10    }
11    topo_sort.push_back(u); // Agregar el nodo al vector
12 }
13 int main(){
14    // Build graph: n = verices
15    topo_sort.clear();
16    for (int i = 0; i < n; ++i) seen[i] = false;
17    for (int i = 0; i < n; ++i) if (!seen[i]) dfs(i);
18    reverse(topo_sort.begin(), topo_sort.end());
19    return 0;
20 }
```

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

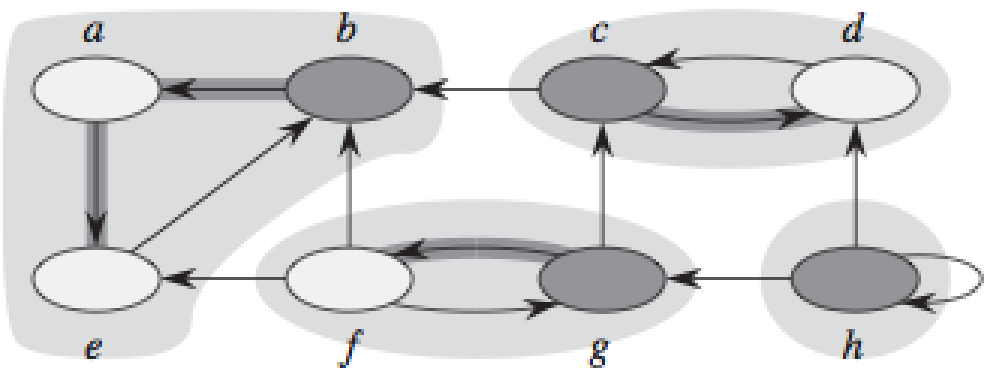
Complejidad

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

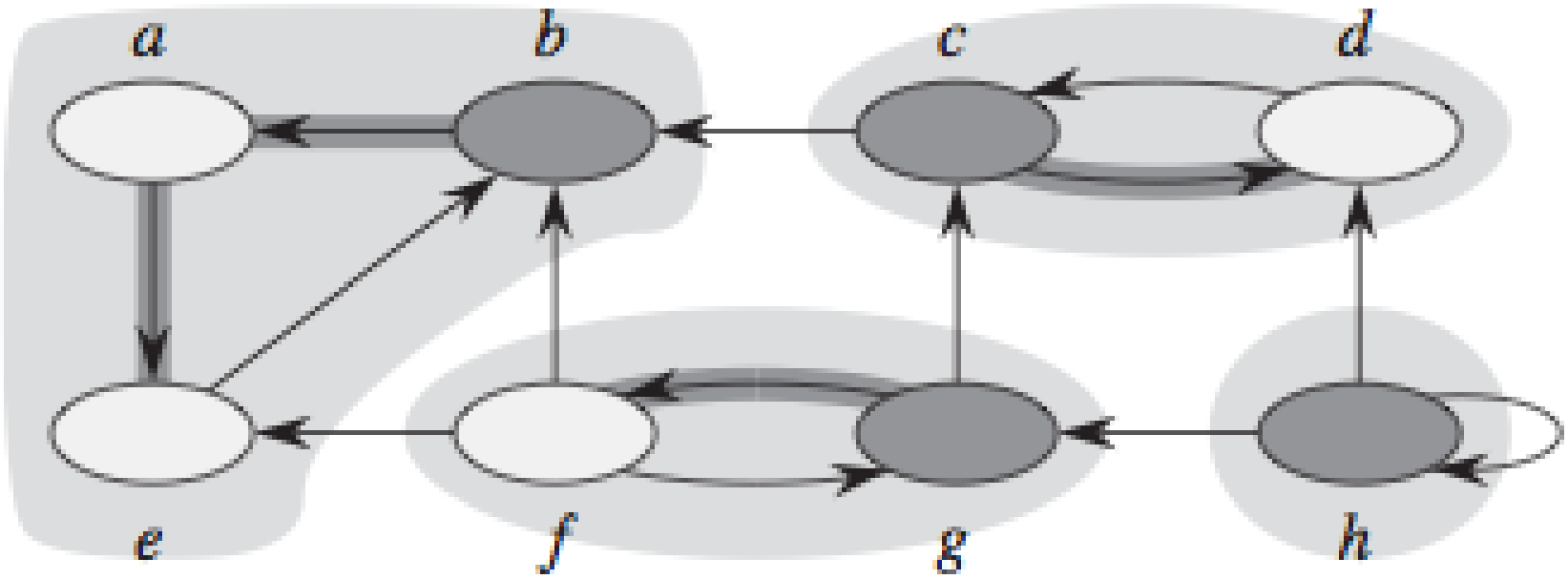
Dado un grafo dirigido $G = (V, E)$, un componente fuertemente conexa o strongly connected component (SCC) de G es un subconjunto C de nodos que cumple que para cada pareja $u, v \in C$ existe un camino de u a v y de v a u y que C es lo más grande posible.



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 dominos que hay que derribar a mano para que todos los dominos se derriben.

Ideas

Ideas

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

Ideas

Ideas

- ¿Qué pasa con las cadenas de dominos 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 dominos 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 dominos 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

```
1  // El maximo numero de dominos
2  const int MAXN = 100005;
3  // El grafo
4  vector <int> g[MAXN];
5  // El grafo reversado
6  vector <int> grev[MAXN];
7  // El "ordenamiento topologico" del grafo G
8  vector <int> topo_sort;
9  // La componente fuertemente conexa a la que pertenece cada nodo
10 int scc[MAXN];
11 // El arreglo de visitado para el primer DFS
12 bool seen[MAXN];
13  // El numero de aristas entrantes a cada componente
14 int in[MAXN];
```

DFS

```
1  // DFS donde se halla el ordenamiento
2  void dfs1(int u){
3     seen[u] = true;
4     for (int i = 0; i < g[u].size(); ++i){
5        int v = g[u][i];
6        if (!seen[v]) dfs1(v);
7     }
8     topo_sort.push_back(u);
9  }
10 // DFS donde se hallan las componentes
11 void dfs2(int u, int comp){
12    scc[u] = comp;
13    for (int i = 0; i < grev[u].size(); ++i){
14       int v = grev[u][i];
15       if (scc[v] == -1) dfs2(v, comp);
16    }
17 }
```

Main I

```
1  int main(){
2     int cases; cin >> cases;
3     while (cases--){
4        int n, m;
5        cin >> n >> m;
6
7        // Limpiar las variables entre caso y caso
8        for (int i = 0; i <= n; ++i){
9           g[i].clear();
10          grev[i].clear();
11          topo_sort.clear();
12          scc[i] = -1;
13          seen[i] = false;
14          in[i] = 0;
15       }
16
17
18
```

Main II

```
19        // Crear el grafo y el grafo reversado
20        for (int i = 0; i < m; ++i){
21            int u, v; cin >> u >> v;
22            u--; v--;
23            g[u].push_back(v);
24            grev[v].push_back(u);
25        }
26
27        // Llamar el primer dfs
28        for (int i = 0; i < n; ++i){
29            if (!seen[i]) dfs1(i);
30        }
31        reverse(topo_sort.begin(), topo_sort.end());
32        // Llamar el segundo dfs
33        int comp = 0;
34        for (int i = 0; i < n; ++i){
35            int u = topo_sort[i];
36            if (scc[u] == -1) dfs2(u, comp++);
37        }
```

Main III

```
38
39          // Ver cuantas aristas entrantes tiene cada componente
40          for (int u = 0; u < n; ++u){
41              for (int i = 0; i < g[u].size(); ++i){
42                  int v = g[u][i];
43                  if (scc[u] != scc[v]) in[scc[v]]++;
44              }
45          }
46
47          // Sumar las componentes que tienen 0 aristas entrantes
48          int count = 0;
49          for (int u = 0; u < comp; ++u){
50              if (in[u] == 0) count++;
51          }
52          cout << count << endl;
53      }
54       return 0;
55  }
```