Semillero de Programación

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

Ana Echavarría Juan Francisco Cardona

Universidad EAFIT

1 de marzo de 2013

Contenido

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

```
int main(){
    int n, m;
    while (cin >> n){
        if (n == 0) break;
        for (int i = 0; i < n; ++i){
            g[i].clear();
            color[i] = -1;
        }
        cin >> m;
        for (int i = 0; i < m; ++i){
            int u, v;
            cin >> u >> v;
            g[u].push_back(v);
            g[v].push_back(u);
        }
        if (dfs(0, 0)) puts("BICOLORABLE.");
        else puts("NOT BICOLORABLE.");
    }
     return 0;
}
```

Solución usando DFS

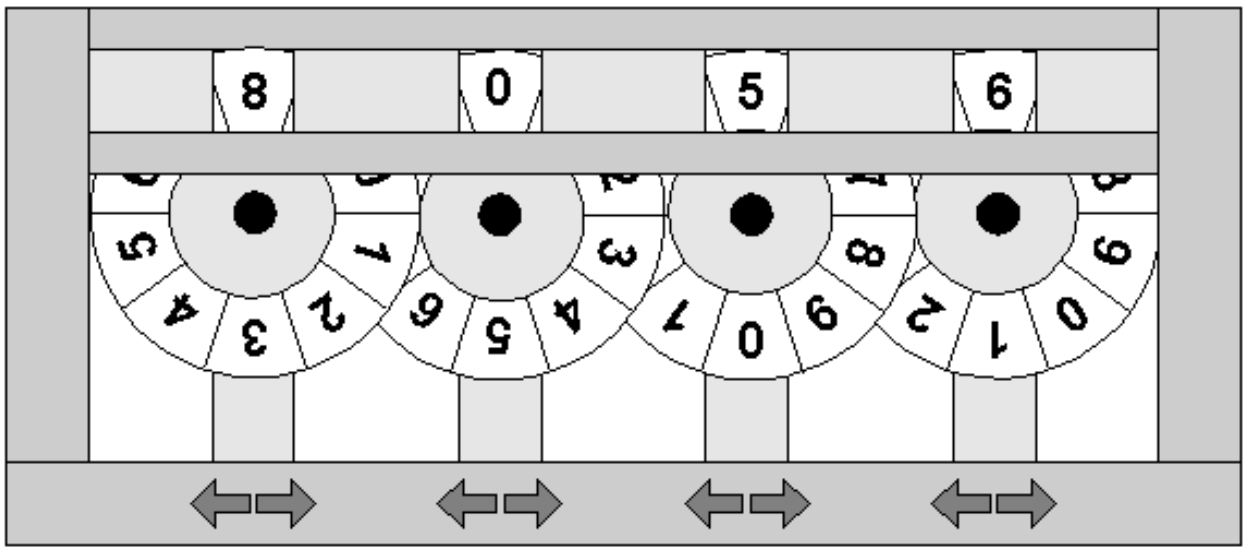
```
1  const int MAXN = 205;
2  vector <int> g[MAXN];
3  int color[MAXN];
4
5  bool dfs(int u, int paint){
6     color[u] = paint;
7     for (int i = 0; i < g[u].size(); ++i){
8        int v = g[u][i];
9        bool possible;
10       if (color[v] == -1) possible = dfs(v, 1 - paint);
11       else possible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

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

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

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

Tarea