DFS

Referências

- Undirected graphs (S&W),
- slides (S&W)
- Directed graphs (S&W)
- slides (S&W)

Vídeos

- Undirected graphs (S&W)
- Directed graphs (S&W)

Anatomia de uma DFS

Cada arco é um arco da arborescência, ou de retorno, ou descendente, ou cruzado.

```
public class DFSAnatomia {
    private boolean[] marked;
    private int[] edgeTo; // DepthFirstPaths

    private int   time; // anatomia
    private int[] d;    // anatomia
    private int[] f;    // anatomia

    private Queue<Integer> pre;  // pré-ordem
    private Queue<Integer> pos; // pós-ordem

    public DFSAnatomia(Digraph G) {
        marked = new boolean[G.V()];
        edgeTo = new int[G.V()];
        pre = new Queue<Integer>;
        pos = new Queue<Integer>;

        d = new int[G.V()]; // anatomia, momento em o vértice foi descoberto
        f = new int[G.V()]; // anatomia. momento que terminamos de examinar o vértice
        for (int v = 0; v < G.V(); v++)
            if (!marked(v)) {
                 dfs(G,v);
            }
    }

    private void dfs(Digraph G, int v) {
         marke[v] = true;
         d[v] = time++; // anatomia
         pre.enqueue(v);
         for (int w : G.adj[v])
             if (!marked(w)) {
                  edgeTo[w] = v; //  DepthFirstPaths
                  dfs(G, w);
             }
         pos.enqueue(v);
         f[v] = time++; // anatomia
    }
```

Ciclos

Um ciclo em um digrafo é qualquer sequência da forma $v_0 - v_1 - v_2 - \cdots - v_{p-1} - v_p$ tal que $v_{k-1} - v_k$ é um arco para $k = 1, 2, \ldots, p$ e $v_0 == v_p$.

Problema

Decidir se um dado digrafo possui um ciclo.

Certificados

Como é possível verificar a resposta?

Como é possível verificar que existe ciclo?

Como é possível verificar que não existe ciclo?

Digrafos acíclicos

Um digrafo é **aciclico** se não tem ciclo. Digrafos acíclicos são também conhecidos como DAGs.

Ordenação topológica

Uma permutação dos vértices de um digrafo é uma sequência em que cada vértice aparece uma e uma só vez.

Uma **ordenação topológica** de um digrafo é uma permutação

```
ts[0] , ts[1], ..., t[V-1]
```

dos seus vértices tal que todo arco tem a forma

```
ts[i]-ts[j] com i < j
```

`ts[0]` é necessariamente uma **fonte**.

`ts[1]` é necessariamente um **sorvedouro**.

DAGs versus ordenação topológica

É evidente que digrafos com ciclos não admitem ordenação topológica.

É menos evidente que todo DAG admite uma ordenação topológica.

A prova desse fato é um algoritmo que recebe qualquer digrafo e devolve:

- um ciclo ou
- uma ordenação topológica.

DirectedCycle

O consumo de tempo desse método é proporcional `V+E` se o digrafo for implementado com lista de adjacências.

O consumo de tempo desse métod é proporcional a V^2 se o digrafo for implementado como matriz de adjacências

```java
public class DirectedCycle {
    private boolean[] marked;        // marked[v] = has vertex v been marked?
    private int[] edgeTo;            // edgeTo[v] = previous vertex on path to v
    private boolean[] onStack;       // onStack[v] = is vertex on the stack?
    private Stack<Integer> cycle;    // directed cycle (or null if no such cycle)
    private int onCycle = -1;
    private Stack<Integer> ts;  // topological order

    // Determines whether the digraph G has a directed cycle and, if so,
    // finds such a cycle.
    public DirectedCycle(Digraph G) {
        marked  = new boolean[G.V()];
        onStack = new boolean[G.V()];
        edgeTo  = new int[G.V()];
        for (int v = 0; v < G.V(); v++)
            if (!marked[v] && cycle == null) dfs(G, v);
    }

    // check that algorithm computes either the topological order or finds a directed cycle
    private void dfs(Digraph G, int v) {
        onStack[v] = true;
        marked[v]  = true;
        for (int w : G.adj(v)) {
            // short circuit if directed cycle found
            if (cycle != null) return;

            // found new vertex, so recur
            else if (!marked[w]) {
                edgeTo[w] = v;
                dfs(G, w);
            }

            // trace back directed cycle
            else if (onStack[w]) { // arco de retorno
                onCycle = v;
                edgeTo[v] = w; // fecha o ciclo;
            }
        }
        onStack[v] = false;
        ts.push(v); // pós-ordem reversa
    }

    // does the digraph have a directed cycle?
    public boolean hasCycle() {
        return onCycle != -1;
    }

    // does the digraph have a directed cycle?
    public boolean isDag() {
        return onCycle == -1;
```

```
    }

    // return a directed cycle (as an iterable) if the digraph has a directed cycle,
    // and null otherwise
    public Iterable<Integer> cycle() {
        if (onCycle == -1) return null;
        if (cycle != null) return cycle;
        for (int x = edgeTo[onCycle]; x != onCycle; x = edgeTo[x])
            cycle.push(x);
        cycle.push(onCycle);
        return cycle;
    }

    // return a topological order of the vertices (as an interable) if the
    // digraph has a topological order (or equivalently, if the digraph is a DAG),
    // and null otherwise
    public Iterable<Integer> order() {
        if (onCycle != -1) return null;
        return ts;
    }
}
```