Algoritmos

Pedro Hokama

Fontes

- [*clrs*] Algoritmos: Teoria e Prática (Terceira Edição) Thomas H. Cormen, Charles Eric Leiserson, Ronald Rivest, Ronald L. Rivest e Clifford Stein.
- [*timr*] Algorithms Illuminated Series, Tim Roughgarden

Apresentação Baseada:

- Stanford Algorithms
  `https://www.youtube.com/playlist?list=PLXFMmlk03Dt7Q0xr1PIAriY5623cKiH7V`
  `https://www.youtube.com/playlist?list=PLXFMmlk03Dt5EMI2s2WQBsLsZl7A5HEK6`
- Conjunto de Slides dos Professores Cid C. de Souza, Cândida N. da Silva, Orlando Lee, Pedro J. de Rezende
- Conjunto de Slides do Professores Cid C. de Souza para a disciplina MO420

Qualquer erro é de minha responsabilidade.

Componentes Conexas (em grafos não direcionados)

- Seja $G = (V, E)$ um grafo **não direcionado**.
- As componentes conexas de um grafo são as partes do grafo que são isoladas.

Definição

Uma **Componente Conexa** é uma classe de equivalência da relação $u \sim v \iff$ existe um $u - v$ caminho em G. ($\sim$ é uma relação de equivalência)

- Desejamos encontrar todas as componente conexas de um grafo:
  - Verificar se a rede está desconectada
  - Visualização de Grafo (e como subrotina para outros algoritmos)
  - Clusterização (de páginas web, imagens, pessoas)

Usando a BFS para encontrar Componentes Conexas

Algoritmo 1: Busca em Largura (BFS)

Entrada: Um Grafo G, e um vértice fonte s

```
1 Marcar s como visitado;
2 Seja F uma Fila inicializada com s;
3 enquanto F ≠ ∅ faça
4     u = F.pop();
5     para cada aresta {u, v} faça
6         se v está não-visitado então
7             Marcar v como visitado;
8             F.push(v).
```

Algoritmo 2: ComponentesConexas

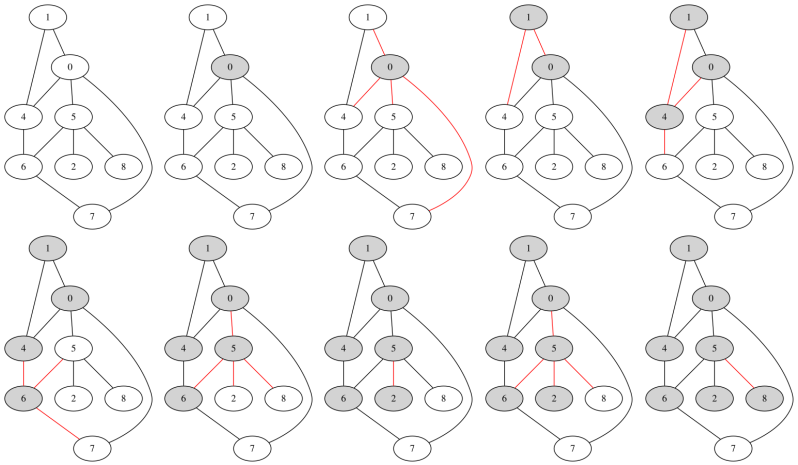
Entrada: Um Grafo G

```
1 Marcar v ∈ V como não-visitado;
2 para cada vértice v de |V| faça
3     se v está não-visitado então
4         BFS(G, i);
```

- Vértices marcados como visitado na mesma chamada de BFS estão na mesma componente conexa.
- Tempo de Execução $O(m + n)$

Busca em Profundidade (Depth-First Search - DFS)

- Explora mais agressivamente, tentando ir o mais longe possível. Retrocede (backtrack) somente quando necessário. Exemplo:



DFS

- A ideia é que para decidir qual o próximo vértice a ser visitado utilizamos uma PILHA (Last In First Out - LIFO).
- Podemos fazer explicitamente, ou podemos fazer usando recursão.

Algoritmo 3: DFS

Entrada: Um Grafo G, e um vértice fonte s

```
1 Marcar s como visitado ;
2 para toda aresta {s, v} faça
3     se v é não visitado então
4         DFS(G, v);
```


Propriedades da DFS

Teorema

Ao final do DFS, v é visitado $\iff$ existem um caminho de s até v em G

- O DFS é só um caso especial da busca genérica, e a prova desse Teorema é a mesma.
- O Tempo de execução é $O(n_s + m_s)$ onde n_s é o número de vértices alcançáveis por s e m_s o número de arestas alcançáveis por s. Basta notar que no máximo é feito uma chamada recursiva de DFS para cada vértice alcançável, e cada aresta é analisada no máximo duas vezes.

Ordenação Topológica

Definição

Uma **Ordenação Topológica** de um grafo direcionado $G = (V, A)$ é uma ordenação f dos vértices de V, tal que:

1. f rotula os vértices com valores $\{1, \ldots, n\}$
2. $(u, v) \in A \implies f(u) < f(v)$

- Pode ser usado para resolver problemas de precedência. Suponha um Grafo onde os vértices são disciplinas, e as arestas indicam pré-requisitos. Ou a produção de um produto que tem várias etapas.
- É útil como pre-processamento em vários algoritmos.
- Se o Grafo tiver um ciclo direcionado então não existe uma ordenação topológica.
- Ordenação topológica só funciona em **Grafo Direcionado Acíclico** (Directed Acyclic Graph - DAG) !!!!

Ordenação Topológica - intuição

- Se eu tenho um DAG eu posso encontrar uma Ordenação topológica
- Todo DAG tem um Sorvedouro (*Sink*), um vértice sem nenhuma aresta saindo. Por que?
- Encontrar o Sorvedouro v, atribuir $f(v) = n$, fazer a recursão em $V - \{v\}$.

Algoritmo 4: DFS

Entrada: Um Grafo G, e um vértice fonte s

```
1 Marcar s como visitado ;
2 para toda aresta {s, v} faça
3     se v é não visitado então
4         DFS(G, v);
5 f(s) = rotulo_atual;
6 rotulo_atual − −;
```

Algoritmo 5: Top-Sort

Entrada: Um Grafo G

```
1 Marcar v ∈ V como não-visitado;
2 rotulo_atual = |V|;
3 para cada aresta v ∈ V faça
4     se v está não-visitado então
5         DFS(G, v);
```


Propriedades da Ordenação Topológica

- Tempo de Execução Linear $O(|V| + |E|)$, já que cada vértice é visitado uma vez.
- O Algoritmo funciona corretamente, ou seja, se $(u, v) \in A$ então $f(u) < f(v)$.
  1. Caso 1: u é visitado pela DFS antes de v. Então a chamada recursiva de v vai terminar antes, e portanto v será rotulado antes e $f(v)$ será maior.
  2. Caso 2: v é visitado pela DFS antes de u. Como não existe um caminho direcionado de v para u (já que é um DAG) então u não é alcançado e logo a chamada recursiva de v vai terminar antes, e portanto v será rotulado antes e $f(v)$ será maior.