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

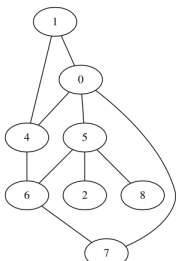
  `https://www.youtube.com/playlist?list=PLXFMmlk03Dt7Q0xr1PIAriY5623cKiH7V`

  `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

Busca em Profundidade (Depth-First Search - DFS)

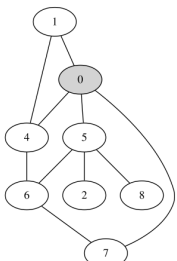
Busca em Profundidade (Depth-First Search - DFS)

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



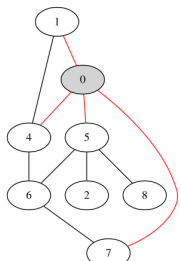
Busca em Profundidade (Depth-First Search - DFS)

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



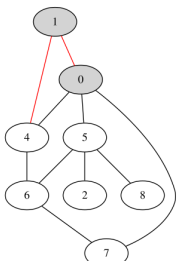
Busca em Profundidade (Depth-First Search - DFS)

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



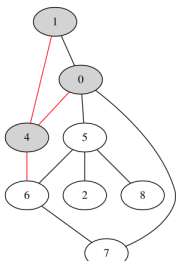
Busca em Profundidade (Depth-First Search - DFS)

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



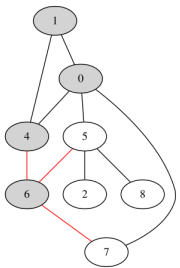
Busca em Profundidade (Depth-First Search - DFS)

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



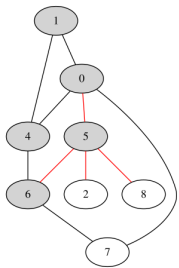
Busca em Profundidade (Depth-First Search - DFS)

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



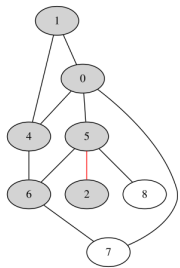
Busca em Profundidade (Depth-First Search - DFS)

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



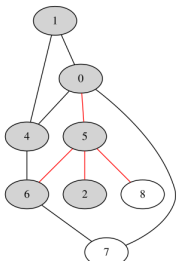
Busca em Profundidade (Depth-First Search - DFS)

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



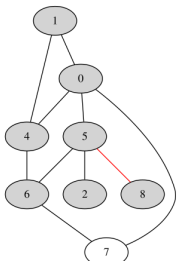
Busca em Profundidade (Depth-First Search - DFS)

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



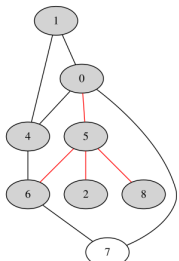
Busca em Profundidade (Depth-First Search - DFS)

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



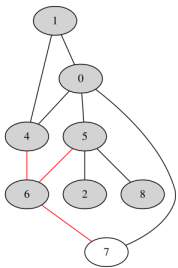
Busca em Profundidade (Depth-First Search - DFS)

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



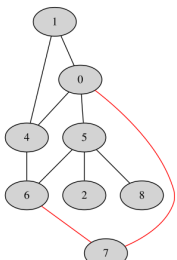
Busca em Profundidade (Depth-First Search - DFS)

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



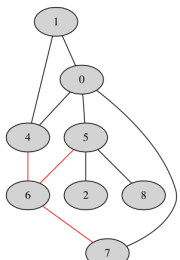
Busca em Profundidade (Depth-First Search - DFS)

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



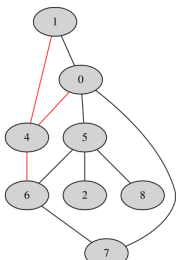
Busca em Profundidade (Depth-First Search - DFS)

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



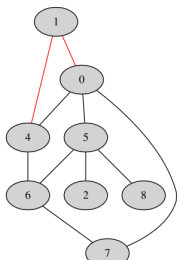
Busca em Profundidade (Depth-First Search - DFS)

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



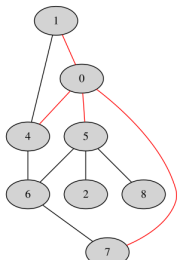
Busca em Profundidade (Depth-First Search - DFS)

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



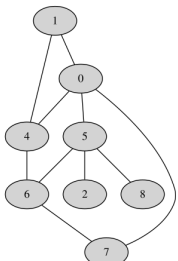
Busca em Profundidade (Depth-First Search - DFS)

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



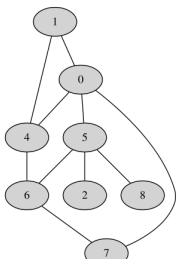
Busca em Profundidade (Depth-First Search - DFS)

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



Busca em Profundidade (Depth-First Search - DFS)

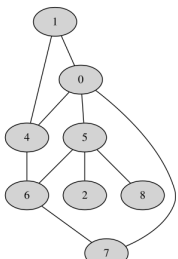
- Explora mais agressivamente, tentando ir o mais longe possível. Retrocede (backtrack) somente quando necessário. Exemplo:



- Por que ver outro algoritmo de Busca em Grafo, se a BFS já é linear?

Busca em Profundidade (Depth-First Search - DFS)

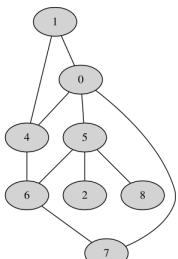
- Explora mais agressivamente, tentando ir o mais longe possível. Retrocede (backtrack) somente quando necessário. Exemplo:



- Por que ver outro algoritmo de Busca em Grafo, se a BFS já é linear?
- Outras aplicações: Ordenação Topológica de Grafos Direcionados Acíclicos, Componentes Fortemente Conexas em grafos direcionados.

Busca em Profundidade (Depth-First Search - DFS)

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



- Por que ver outro algoritmo de Busca em Grafo, se a BFS já é linear?
- Outras aplicações: Ordenação Topológica de Grafos Direcionados Acíclicos, Componentes Fortemente Conexas em grafos direcionados.
- Tempo de execução: $O(|V| + |E|)$

DFS

DFS

- A ideia é que para decidir qual o próximo vértice a ser visitado utilizamos uma PILHA (Last In First Out - LIFO).

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

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.

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.

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:

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 $\{1, \ldots, n\}$

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. $(u, v) \in A \implies f(u) < f(v)$

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.

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.

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

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

Ordenação Topológica - intuição

- Se eu tenho um DAG eu posso encontrar uma Ordenação topológica

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?

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);
```

Algoritmo 6: 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);
```

Algoritmo 7: Top-Sort

Entrada: Um Grafo G

```
1 Marcar v ∈ V como não-visitado;

3 para cada aresta v ∈ V faça
4     se v está não-visitado então
5         DFS(G, v);
```

Algoritmo 8: 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 9: Top-Sort

Entrada: Um Grafo G

1 Marcar $v \in V$ como não-visitado;
2 $rotulo\_atual = |V|$;
3 **para** *cada aresta* $v \in V$ **faça**
4 **se** v *está não-visitado* **então**
5 $DFS(G, v)$;

Propriedades da Ordenação Topológica

Propriedades da Ordenação Topológica

- Tempo de Execução Linear $O(|V| + |E|)$, já que cada vértice é visitado uma vez.

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 $f(u) < f(v)$.

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.

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.

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.

Propriedades da Ordenação Topológica

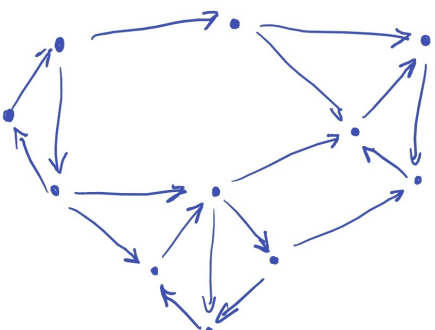
- Tempo de Execução Linear $O(|V| + |E|)$, já que cada vértice é visitado uma vez.
- O Algoritmo funciona corretamente, ou seja, se $(u, v) \in A$ então $f(u) < f(v)$.
  1. Caso 1: u é visitado pela DFS antes de v. Então a chamada recursiva de v vai terminar antes, e portanto v será rotulado antes e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

Componentes fortemente conexas - Strongly Connected Components

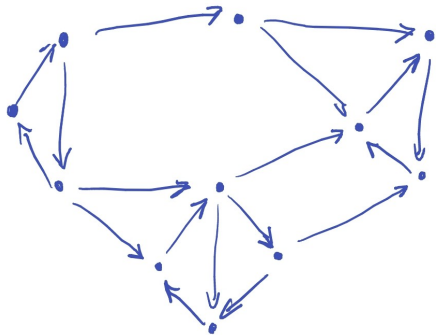
Componentes fortemente conexas - Strongly Connected Components

Definição

Componentes Fortemente Conexas de um Grafo Direcionado são classes de equivalência da relação $u \sim v \iff$ existe um $u - v$ caminho, e um $v - u$ caminho em G. ($\sim$ é uma relação de equivalência)



DFS para encontrar as SCCs?



Algoritmo de Kosaraju

O algoritmo de Kosaraju faz duas passadas no DFS (com pequenas modificações), tem complexidade $O(|V| + |E|)$.
Ideia do algoritmo:

Algoritmo de Kosaraju

O algoritmo de Kosaraju faz duas passadas no DFS (com pequenas modificações), tem complexidade $O(|V| + |E|)$.
Ideia do algoritmo:

- Reverter todos os arcos de G e chamar de G^{rev}

Algoritmo de Kosaraju

O algoritmo de Kosaraju faz duas passadas no DFS (com pequenas modificações), tem complexidade $O(|V| + |E|)$.
Ideia do algoritmo:

- Reverter todos os arcos de G e chamar de G^{rev}
- DFS-Loop em G^{rev} (salvando os tempos de termino)

Algoritmo de Kosaraju

O algoritmo de Kosaraju faz duas passadas no DFS (com pequenas modificações), tem complexidade $O(|V| + |E|)$.
Ideia do algoritmo:

- Reverter todos os arcos de G e chamar de G^{rev}
- DFS-Loop em G^{rev} (salvando os tempos de termino)
- DFS-Loop em G (na ordem de tempos de termino da primeira passada)

Algoritmo de Kosaraju

O algoritmo de Kosaraju faz duas passadas no DFS (com pequenas modificações), tem complexidade $O(|V| + |E|)$.
Ideia do algoritmo:

- Reverter todos os arcos de G e chamar de G^{rev}
- DFS-Loop em G^{rev} (salvando os tempos de termino)
- DFS-Loop em G (na ordem de tempos de termino da primeira passada)

Observações:

Algoritmo de Kosaraju

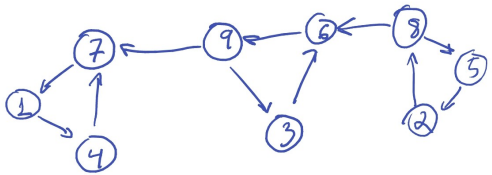
O algoritmo de Kosaraju faz duas passadas no DFS (com pequenas modificações), tem complexidade $O(|V| + |E|)$.
Ideia do algoritmo:

- Reverter todos os arcos de G e chamar de G^{rev}
- DFS-Loop em G^{rev} (salvando os tempos de termino)
- DFS-Loop em G (na ordem de tempos de termino da primeira passada)

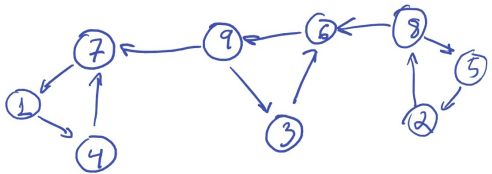
Observações:

- Computaremos lideres para cada vértice, vértices com o mesmo líder estarão na mesma SCC.

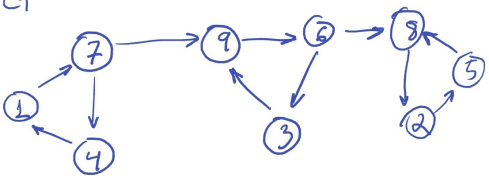
C1:



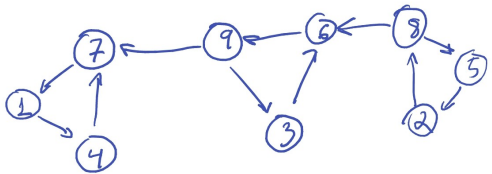
G_1:



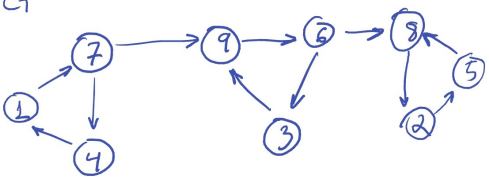
G_1^{rev}

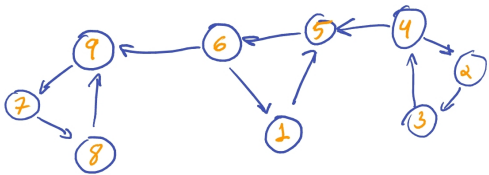


G:



G^{rev}





Algoritmo 10: DFS-Loop

Entrada: Um Grafo G

1 Marcar $v \in V$ como não-visitado;

4 Assuma que os vértices estão rotulados de 1 até n;

5 **para** $v = n$ *até* 1 **faça**

6 **se** v *está não-visitado* **então**

8 $DFS(G, v)$;

Algoritmo 12: DFS-Loop

Entrada: Um Grafo G

1 Marcar $v \in V$ como não-visitado;

4 Assuma que os vértices estão rotulados de 1 até n;
5 **para** $v = n$ *até* 1 **faça**
6 **se** v *está não-visitado* **então**

8 $DFS(G, v)$;

Algoritmo 13: DFS

Entrada: Um Grafo G, e um vértice fonte s

1 Marcar s como visitado ;

3 **para** *todo arco* (s, v) **faça**
4 **se** v *é não visitado* **então**
5 $DFS(G, v)$;

Algoritmo 14: DFS-Loop

Entrada: Um Grafo G

```
1  Marcar v ∈ V como não-visitado;
2  T = 0;
3  L = NULL;
4  Assuma que os vértices estão rotulados de 1 até n;
5  para v = n até 1 faça
6      se v está não-visitado então
7          L = v;
8          DFS(G, v);
```

Algoritmo 15: DFS

Entrada: Um Grafo G, e um vértice fonte s

```
1  Marcar s como visitado ;
2  lider(s) = L;
3  para todo arco (s, v) faça
4      se v é não visitado então
5          DFS(G, v);
6  T++;
7  f(s) = T;
```

Corretude do Kosaraju

Corretude do Kosaraju

- As SCCs de um grafo direcionado G correspondem a um "Meta-Grafo"acíclico em que:

Corretude do Kosaraju

- As SCCs de um grafo direcionado G correspondem a um "Meta-Grafo"acíclico em que:
- Os meta-vértices são as SCCs $C_1, \ldots, C_k$ de G.

Corretude do Kosaraju

- As SCCs de um grafo direcionado G correspondem a um "Meta-Grafo"acíclico em que:
- Os meta-vértices são as SCCs $C_1, \ldots, C_k$ de G.
- Existe um arco de $(C, C') \iff$ Existe $(i, j) \in A$ com $i \in C$ e $j \in C'$.

Corretude do Kosaraju

- As SCCs de um grafo direcionado G correspondem a um "Meta-Grafo"acíclico em que:
- Os meta-vértices são as SCCs $C_1, \ldots, C_k$ de G.
- Existe um arco de $(C, C') \iff$ Existe $(i, j) \in A$ com $i \in C$ e $j \in C'$.
- O grafo é acíclico, senão cada C não seria uma SCC.

Lema 1

Considere duas SCC "adjacentes"em G, sendo que existe uma aresta de C_1 para C_2. Seja f(v) = tempo de termino do DFS-Loop no G^{rev}. Então:

$$\max_{v \in C_1} f(v) < \max_{v \in C_2} f(v)$$

Prova: Seja v o primeiro vértice visitado de $C_1 \cup C_2$, na primeira passagem do algoritmo.
Note que em G^{rev} a aresta considerada está invertida.

Lema 1

Considere duas SCC "adjacentes"em G, sendo que existe uma aresta de C_1 para C_2. Seja f(v) = tempo de termino do DFS-Loop no G^{rev}. Então:

$$\max_{v \in C_1} f(v) < \max_{v \in C_2} f(v)$$

Prova: Seja v o primeiro vértice visitado de $C_1 \cup C_2$, na primeira passagem do algoritmo.
Note que em G^{rev} a aresta considerada está invertida.

1. Caso 1: $v \in C_1$, então todos os vértices de C_1 serão visitados antes de qualquer vértice de C_2

Lema 1

Considere duas SCC "adjacentes"em G, sendo que existe uma aresta de C_1 para C_2. Seja f(v) = tempo de termino do DFS-Loop no G^{rev}. Então:

$$\max_{v \in C_1} f(v) < \max_{v \in C_2} f(v)$$

Prova: Seja v o primeiro vértice visitado de $C_1 \cup C_2$, na primeira passagem do algoritmo.
Note que em G^{rev} a aresta considerada está invertida.

1. Caso 1: $v \in C_1$, então todos os vértices de C_1 serão visitados antes de qualquer vértice de C_2
2. Caso 2: $v \in C_2$, eventualmente algum vértice de C_1 é alcançado e todo C_1 é explorado antes de C_2 terminar.

Lema 1

Considere duas SCC "adjacentes"em G, sendo que existe uma aresta de C_1 para C_2. Seja f(v) = tempo de termino do DFS-Loop no G^{rev}. Então:

$$\max_{v \in C_1} f(v) < \max_{v \in C_2} f(v)$$

Prova: Seja v o primeiro vértice visitado de $C_1 \cup C_2$, na primeira passagem do algoritmo.
Note que em G^{rev} a aresta considerada está invertida.

1. Caso 1: $v \in C_1$, então todos os vértices de C_1 serão visitados antes de qualquer vértice de C_2
2. Caso 2: $v \in C_2$, eventualmente algum vértice de C_1 é alcançado e todo C_1 é explorado antes de C_2 terminar.

Corolário 1

O maior valor de f deve ficar em um SCC sorvedouro.

Corretude do Kosaraju

Corolário 1

O maior valor de f deve ficar em um SCC sorvedouro.

Corretude do Kosaraju

- Como o algoritmo começa a executar em uma SCC sorvedouro, ele encontra toda a componente sem sair dela.

Corolário 1

O maior valor de f deve ficar em um SCC sorvedouro.

Corretude do Kosaraju

- Como o algoritmo começa a executar em uma SCC sorvedouro, ele encontra toda a componente sem sair dela.
- A partir dai os vértices dessa SCC não é mais considerado e a próxima SCC sorvedouro é executada.

Corolário 1

O maior valor de f deve ficar em um SCC sorvedouro.

Corretude do Kosaraju

- Como o algoritmo começa a executar em uma SCC sorvedouro, ele encontra toda a componente sem sair dela.
- A partir dai os vértices dessa SCC não é mais considerado e a próxima SCC sorvedouro é executada.
- Dessa forma cada SCC é encontrada separadamente □.