Algoritmos e Estruturas de Dados II – SCC-203

Grafos: Aplicações de Busca em Profundidade

Gustavo Batista

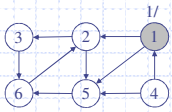
Teste para Verificar se Grafo é Acíclico

- A busca em profundidade pode ser usada para verificar se um grafo é acíclico ou contém um ou mais ciclos.
- Se uma aresta de retorno é encontrada durante a busca em profundidade em *G*, então o grafo tem ciclo.

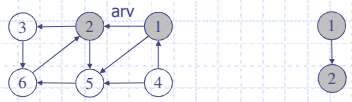
Teste para Verificar se Grafo é Acíclico

- Um grafo direcionado ou não *G* é acíclico se e somente se a busca em profundidade em *G* não apresentar arestas de retorno.
- O algoritmo de busca em profundidade pode ser modificado para detectar ciclos em grafos orientados simplesmente verificando se um vértice *w* adjacente a *v* possui cor cinza na primeira vez que a aresta *(v, w)* é percorrida.

Teste para Verificar se Grafo é Acíclico

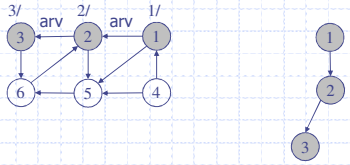


Teste para Verificar se Grafo é Acíclico



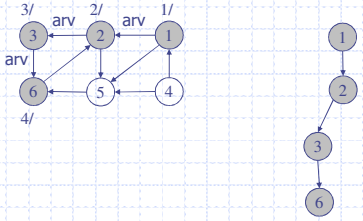
5

Teste para Verificar se Grafo é Acíclico



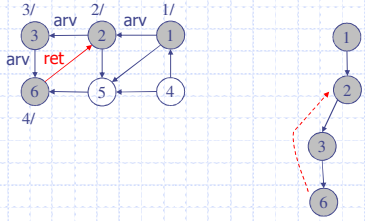
6

Teste para Verificar se Grafo é Acíclico



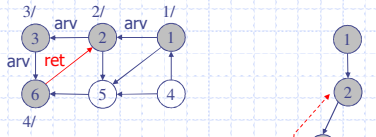
7

Teste para Verificar se Grafo é Acíclico



8

Teste para Verificar se Grafo é Acíclico



O grafo é cíclico

9

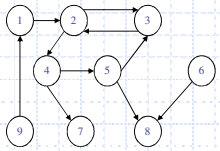
Teste para Verificar se Grafo é Acíclico

- A complexidade do algoritmo para verificar se um grafo direcionado é acíclico é a mesma da busca em profundidade. Ou seja:
  - $O(|V|+|V|^2)$ para matrizes de adjacência e;
  - $O(|V|+|A|)$ para listas de adjacência.

10

Exercício

- Mostre como a busca em profundidade que o seguinte gráfico é cíclico.



11

Componentes Conexos

- Componente conexo de um grafo é um conjunto maximal de vértices tal que existe um caminho entre todos os pares de vértices.
- Existe problemas aparentemente complexos, como testar se 15-puzzle ou cubo mágico podem ser solucionados a partir de qualquer posição, que se resumem a um teste de conectividade.
- Um teste simples para indicar se um grafo é um componente conexo é utilizar uma busca em profundidade ou largura e verificar se todos os vértices podem ser alcançados.

12

Componentes Conexos

- Um segundo problema é encontrar um componente conexo dado um grafo.
- Componentes conexos podem ser facilmente encontrados a partir de uma busca em largura ou profundidade.
- Basta iniciar a busca e verificar se existem vértices não descobertos. Reiniciar a busca a partir de um vértice não descoberto até que todos sejam descobertos.

13

Componentes Conexos

```
void componentes_conexo(tgrafo *grafo) {
    tvertice v;
    int cor[MAXNUMVERTICES], c;

    c = 1;
    for (v = 0; v < grafo->num_vertices; v++)
        cor[v] = BRANCO;
    for (v = 0; v < grafo->num_vertices; v++)
        if (cor[v] == BRANCO) {
            printf("Componente conexo: %d", c++);
            visita_dfs(v, cor, grafo);
        }

}
```

14

Componentes Conexos

```
void visita_dfs(tvertice v, int cor[], tgrafo *grafo) {
    tvertice w;
    tapontador p;
    tpeso peso;

    printf("%d ", v);
    cor[v] = CINZA;
    p = primeiro_adj(v, grafo);
    while (p != NULO) {
        recupera_adj(v, p, &w, &peso, grafo);
        if (cor[w] == BRANCO)
            visita_dfs(w, cor, grafo);
        p = proximo_adj(v, p, grafo);
    }
    cor[v] = PRETO;
}
```

15

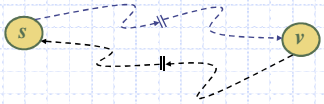
Componentes Fortemente Conexos

- Termo utilizado em grafos direcionados.
- Um teste de conexão forte é realizar uma busca em largura ou profundidade múltiplas vezes, verificando se a partir de cada vértice tomado como origem todos os demais vértices são alcançáveis ou não.
- Complexidade:
  - $O(|V| (|V| + |V|^2))$ para matrizes de adjacências;
  - $O(|V| (|V| + |A|))$ para listas de adjacências.

16

Componentes Fortemente Conexos

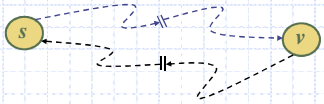
- Teste de Conexão Forte Eficiente: Basta que exista um único vértice de um dígrafo G que alcance qualquer outro e que seja alcançável por qualquer outro para que todos os vértices de G possuam essa mesma propriedade através dele $\Rightarrow$ G fortemente conexo.
  - Note que um vértice s alcança qualquer outro v e é alcançável por v se e somente se s alcança v em ambos os dígrafos G e G^T (transposto), pois o ciclo direcionado entre s e v se mantém (invertido) em G^T.
  - Logo, basta tomar qualquer vértice e executar DFS ou BFS duas vezes, uma sobre o dígrafo original G e a outra sobre G^T: $O(|V| + |A|)$ em listas de adjacências.



17

Componentes Fortemente Conexos

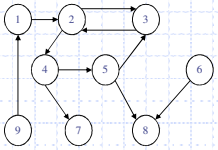
- Componentes Fortemente Conexos: Podemos utilizar a propriedade anterior também para calcular os componentes fortemente conexos de G:
  - Toma-se um vértice v e calcula-se o componente fortemente conexo que inclui v como todos aqueles vértices (e as respectivas arestas) que são alcançados por v em ambos os dígrafos G e G^T.
  - Faz-se isso sucessivas vezes, sempre a partir de um vértice não presente no componente fortemente conexo anterior.



18

Exercício

- Mostre os componentes fortemente conexos do seguinte grafo:



19

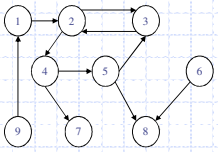
Fechamento Transitivo

- Fechamento Transitivo: é simples calcular o fechamento transitivo de um dígrafo G via busca em profundidade ou largura executando o caminhamento a partir de cada vértice s de G e inserindo uma aresta direcionada adicional ligando a origem s a cada vértice alcançável a partir de s (se esta aresta já não existir).
  - Tempo = $|V|$ caminhamentos $\Rightarrow$ $O(|V|(|V| + |A|))$ para listas de adjacências.
  - Superior a algoritmo de Floyd-Warshall ($O(|V|^3)$) se G não for denso.

20

Exercício

- Mostre o fechamento transitivo do seguinte grafo:



21

Ordenação Topológica

- Um grafo direcionado acíclico é também chamado de **dag** (*directed acyclic graph*).
- Um dag é diferente de uma árvore, uma vez que as árvores são não direcionadas.
- Dags podem ser utilizados, por exemplo, para indicar precedências entre eventos.

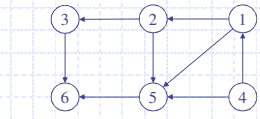
22

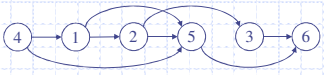
Ordenação Topológica

- A ordenação topológica é uma ordenação linear de todos os vértices, tal que se *G* contém uma aresta *(u,v)* então *u* aparece antes de *v*.
- Pode ser vista como uma ordenação de seus vértices ao longo de uma linha horizontal de tal forma que todas as arestas estão direcionadas da esquerda para a direita.

23

Ordenação Topológica





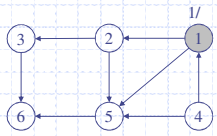
24

Ordenação Topológica

- A ordenação topológica de um dag pode ser obtida utilizando-se uma busca em profundidade.
- Para isso deve-se fazer o seguinte algoritmo:
  1. Faça uma busca em profundidade;
  2. Quando um vértice é pintado de preto, insira-o na cabeça de uma lista de vértices;
  3. Retorne a lista de vértices.

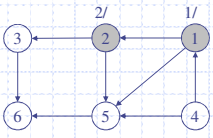
25

Ordenação Topológica



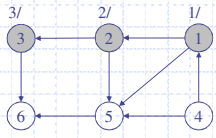
26

Ordenação Topológica



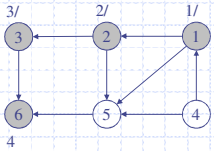
27

Ordenação Topológica



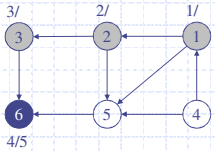
28

Ordenação Topológica



29

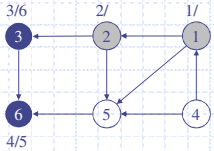
Ordenação Topológica



6

30

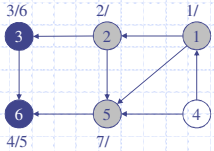
Ordenação Topológica



3 6

31

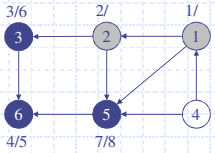
Ordenação Topológica



3 6

32

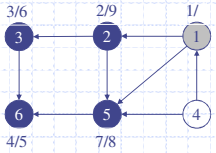
Ordenação Topológica



5 3 6

33

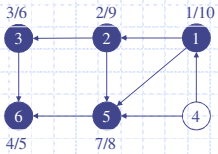
Ordenação Topológica



2 5 3 6

34

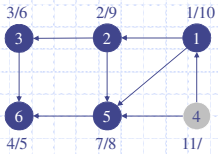
Ordenação Topológica



1 2 5 3 6

35

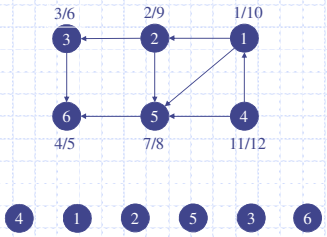
Ordenação Topológica



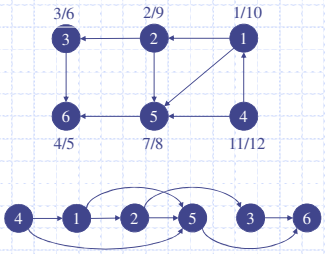
1 2 5 3 6

36

Ordenação Topológica



Ordenação Topológica



Ordenação Topológica

- A complexidade do algoritmo de ordenação topológica em um dag é a mesma da busca em profundidade, ou seja:
  - $O(|V|+|V|^2)$ para matrizes de adjacência, e;
  - $O(|V|+|A|)$ para listas de adjacência.
- Inserir um elemento na cabeça da lista é $O(1)$.

Exercício

- Dado o seguinte grafo, mostre uma possível ordenação topológica.