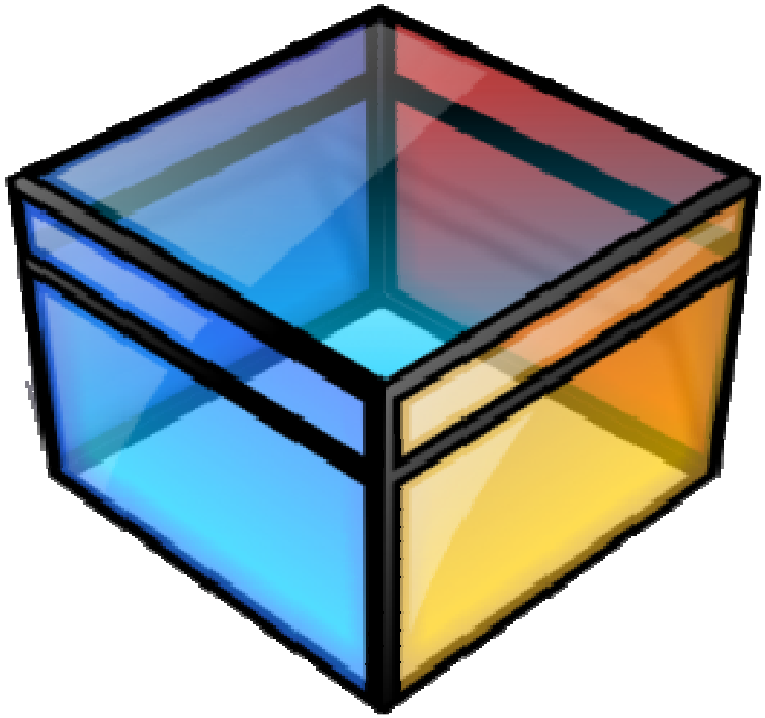
Grafos



COM11087-Tópicos Especiais em Programação II

edmar.kampke@ufes.br

Introdução

- Grafos são estruturas muito estudadas na Ciência da Computação para modelagem de problemas

- Euler (1736) em Königsberg (antiga Prússia), com o problema das 7 pontes sobre o Rio Pregel

- Definição: Um grafo $G = (V,E)$ é formado por um conjunto V de vértices e um conjunto E de pares (u,v), onde cada par representa uma aresta que liga os vértices u e v.

Introdução

- Exemplos:

1) Mapa rodoviário, as cidades seriam os vértices e as estradas as arestas

2) Rede social, ou para estudar os relacionamentos humanos, as pessoas seriam os vértices e as relações interpessoais as arestas

3) Centro de Distribuição de Água, cada centro de bombeamento é um vértice e as tubulações são as arestas

Conceitos

- Grau de vértice
- Grafo não direcionado / Grafo direcionado (Dígrafo)
- Ponderados / Não-Ponderado
- Trilha/Caminhos/Circuitos
- Cíclico / Acíclico
- Simples / Não-simples

Conceitos

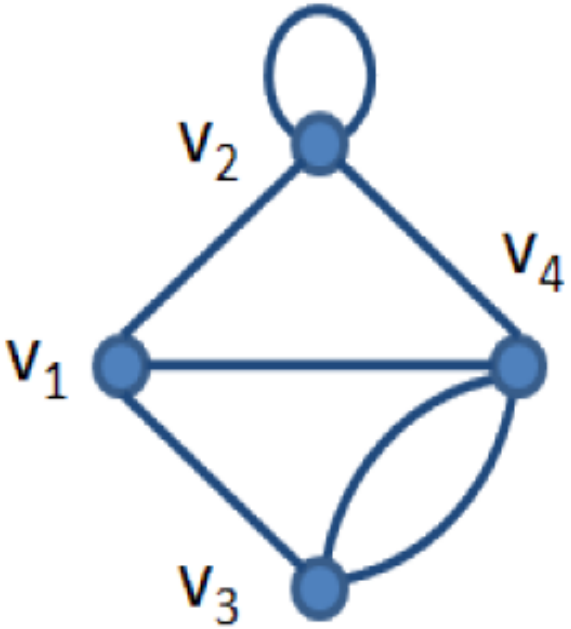
- Planaridade
- Isomorfismo
- Subgrafo e Complemento
- Conexidade
- Árvores
- Grafo Completo
- Grafo k-partido

Representação Computacional

- Matriz de Incidência
  - ✓ Para todo grafo G existe uma matriz $|V| \times |A|$ chamada matriz de incidência
  - ✓ Seja $V = \{v_1, v_2, \ldots, v_n\}$ e $A = \{a_1, a_2, \ldots, a_m\}$
  - ✓ A matriz de incidência de G é a matriz $\mathbf{M(G)} = [m_{ij}]$, sendo m_{ij} o número de vezes (0, 1 ou 2) que o vértice v_i e a aresta a_j são incidentes

Representação Computacional

- Matriz de Incidência



M(*G*)

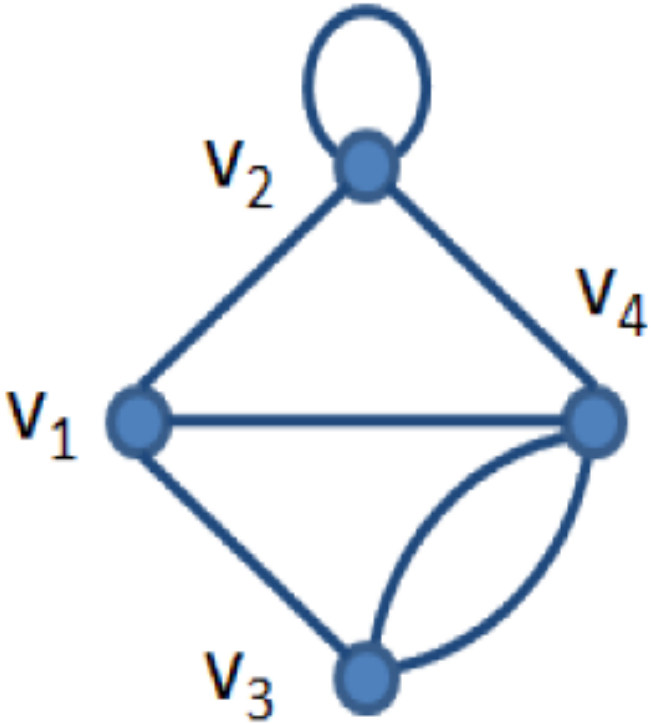
	a_1	a_2	a_3	a_4	a_5	a_6	a_7
v_1	1	1	1	0	0	0	0
v_2	1	0	0	1	2	0	0
v_3	0	1	0	0	0	1	1
v_4	0	0	1	1	0	1	1

Representação Computacional

- Matriz de Adjacência
  - ✓ Para todo grafo *G* existe uma matriz |V|×|V| chamada matriz de adjacência
  - ✓ Seja V = $\{v_1, v_2, \ldots, v_n\}$
  - ✓ A matriz de adjacência de *G* é a matriz **A(*G*)** = $[a_{ij}]$, sendo a_{ij} o número de arestas unindo os vértices v_i e v_j

Representação Computacional

- Matriz de Adjacência



$\mathbf{A}(G)$

	v_1	v_2	v_3	v_4
v_1	0	1	1	1
v_2	1	1	0	1
v_3	1	0	0	2
v_4	1	1	2	0

Representação Computacional

- A matriz de adjacência é geralmente consideravelmente menor que a matriz de incidência (por que?)

- Por isso a de adjacência é mais usada que a de incidência para armazenar grafos em computadores

M(*G*)

	a_1	a_2	a_3	a_4	a_5	a_6	a_7
v_1	1	1	1	0	0	0	0
v_2	1	0	0	1	2	0	0
v_3	0	1	0	0	0	1	1
v_4	0	0	1	1	0	1	1

A(*G*)

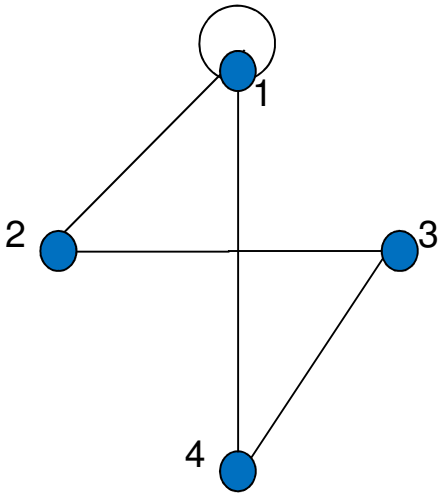
	v_1	v_2	v_3	v_4
v_1	0	1	1	1
v_2	1	1	0	1
v_3	1	0	0	2
v_4	1	1	2	0

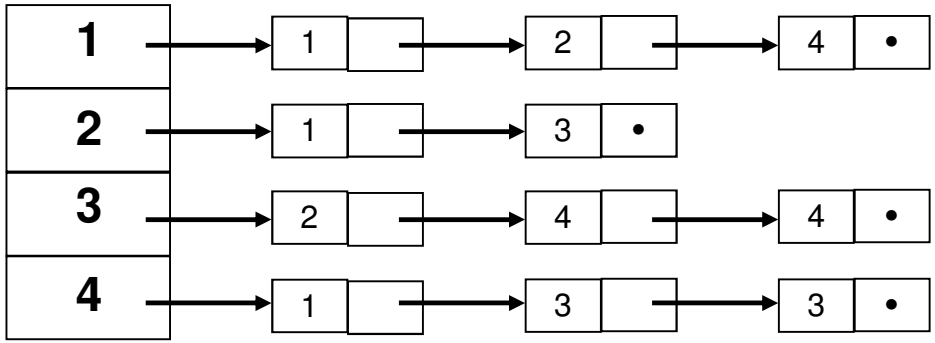
Representação Computacional

- Lista de Adjacência
  - ✓ Simples
  - ✓ Lista de listas de vértices
  - ✓ Cada lista: formada por um vértice e seus adjacentes
  - ✓ Adequada na representação de grafos esparsos
  - ✓ Ineficiente na busca de uma aresta no grafo
  - ✓ A lista associada a um vértice pode ser vazia.
  - ✓ Em grafos não orientados, pode-se evitar a repetição na representação de arestas adotando-se algum critério de ordenação

Representação Computacional

- Lista de Adjacência





Representação Computacional

	Matriz de Incidências	Matriz de Adjacências	Lista de Adjacências
Uso Memória	O(nm)	$O(n^2)$	O(n+m)
Procurar todos os vértices adjacentes de um vértice *v*	O(nm)	O(n)	O(grau(*v*))
Verificar se dois vértices *u e v são* adjacentes	O(m)	O(1)	O(grau(*u*))
Visitar todas as arestas	O(nm)	$O(n^2)$	O(m)
Calcular o grau de um vértice *u*	O(m)	O(n)	O(grau(*u*))

Note que para grafos completos, a estrutura mais eficiente é matriz de adjacências. Para grafos esparsos, a estrutura de listas de adjacências é mais eficiente.

Código-fonte do Livro

- Representação através de vetor
  - Número Máximo de Vértices e Número Máximo de grau
  - Matriz de Adjacencias e Vetor de Grau's
  - Número de Arestas e Número de Vértices

- Grafo Direcionado

A aresta (x,y) será inserida colocando y na lista de adjacências de x

- Leitura de um Grafo

- Inserindo Aresta

Busca em Profundidade

- Complexidade:

✓ Se o grafo tem mais arestas que vértices a complexidade de tempo será $O(m)$
✓ Caso contrário, temos a complexidade $O(n)$

Busca em Largura

- Para cada vértice do grafo visitam-se os vértices adjacentes sempre que possível

- A Busca em Largura está intimamente relacionada com o conceito de distância entre vértices. Quando aplicada a uma árvore, a busca faz uma varredura "por níveis"

- Principal diferença entre Busca em Profundidade e Busca em Largura:
  - ✓ Busca em Profundidade usa Pilha (administrada pela recursão)
  - ✓ Busca em Largura usa Fila

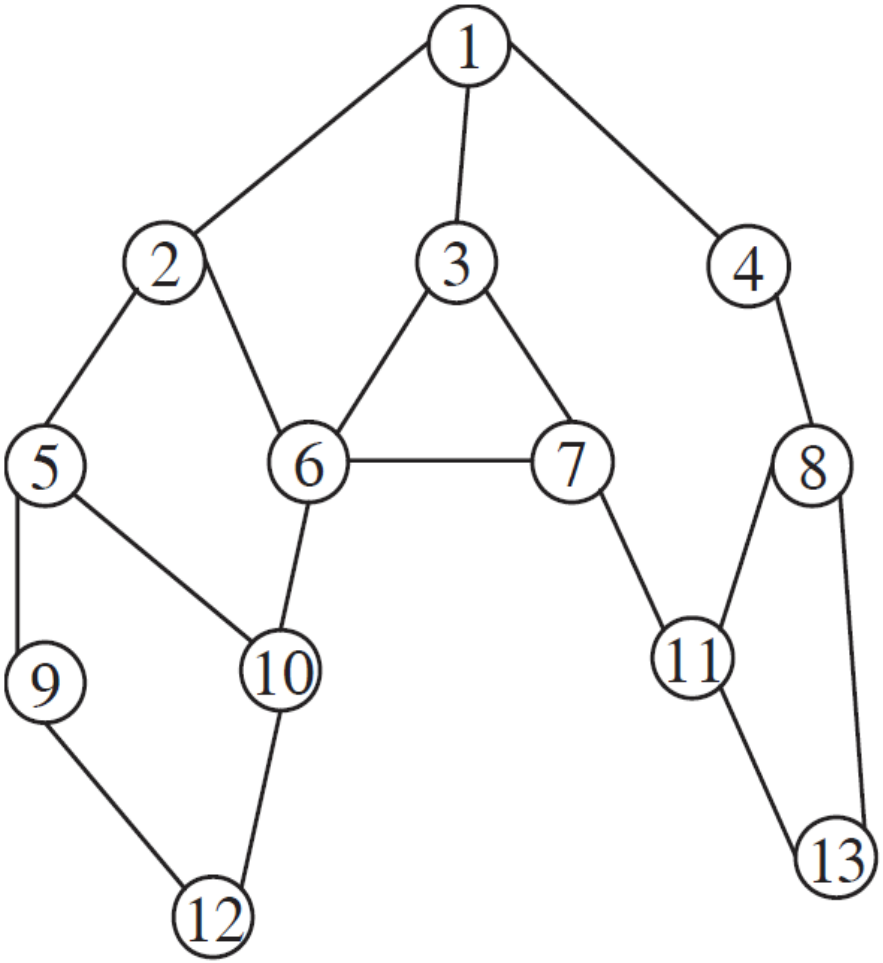
Diferenças

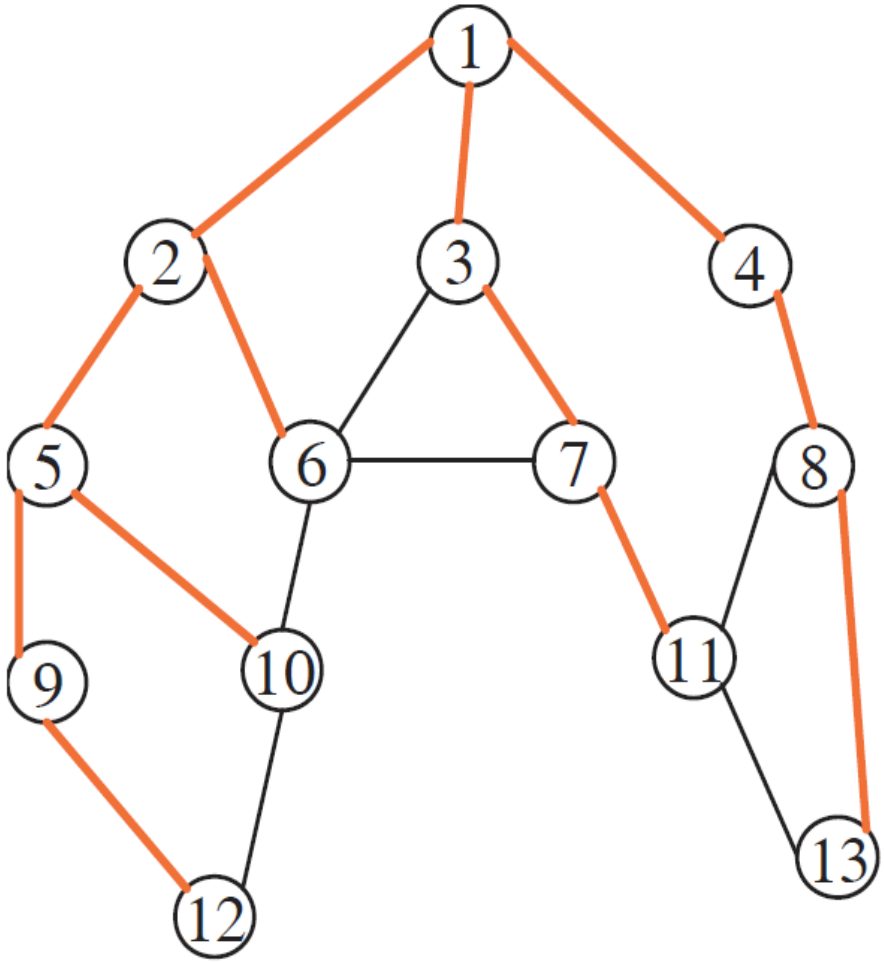
- Outras diferenças entre os dois algoritmos de busca:
  - ✓ Na BP o próprio algoritmo escolhe o vértice inicial, enquanto que na BL o vértice inicial é escolhido pelo usuário
  - ✓ A BP visita todos os vértices do grafo, enquanto que a BL visita apenas os vértices que podem ser atingidos pelo vértice inicial
  - ✓ BP é, usualmente, recursivo, enquanto a BL é iterativo

- A BP e BL são diferentes e tem aplicações muito diferentes

Busca em Largura

- Exemplo





- Ordem de visita dos vértices: 1,2,3,4,5,6,7,8,9,10,11,13,12

Algoritmo BL

- Na Busca em Largura define-se uma fila F para armazenar os vértices adjacentes de um vértice já visitado

- Usamos as seguintes operações de fila

✓ **F.InitFila();** //Inicializa a fila

✓ **F. FilaVazia();**

//Retorna *true* se a fila está vazia e *false* caso contrário

✓ **F.InsereNaFila(*v*);**

//Insere vértive v na fila

✓ **F.RemoveDaFila();**

//Retorna o vértive removido da fila

Algoritmo BL


```
BLargura (G, v, Visit[])
{
    Fila F;
    cont ← 0;
    Para cada vértice u do grafo façafaça
        Visit[u] ← -1;
    Visit[v] ← ++cont;
    F.InitFila();
    F.InsereNaFila(v);
    Escreva(v);
    Enquanto !F.FilaVazia() faça
    {
        x ← F.RemoveDaFila();
        Para cada vértice y adjacente a x

            Se Visit[y] == -1 então
            {
                F.InsereNaFila(y);
                Visit[y] ← ++cont;
                Escreva(y);
             }
        }
    }
```

Busca em Largura

- Complexidade:

✓Em relação ao tempo de execução a BL possui a mesma complexidade que a BP.

✓ Porém, em termos de memória, o desempenho da BL é pior.

Aplicações dos Algoritmos

- Determinar o número de componentes conexas de um grafo
- Determinar um caminho entre dois vértices
- Determinar um ciclo de um grafo
- Determinar as pontes de um grafo
- Determinar as articulações de um grafo
- Determinar se um grafo é bipartido

Aplicações dos Algoritmos

➢ Determinar o número de componentes conexas de um grafo

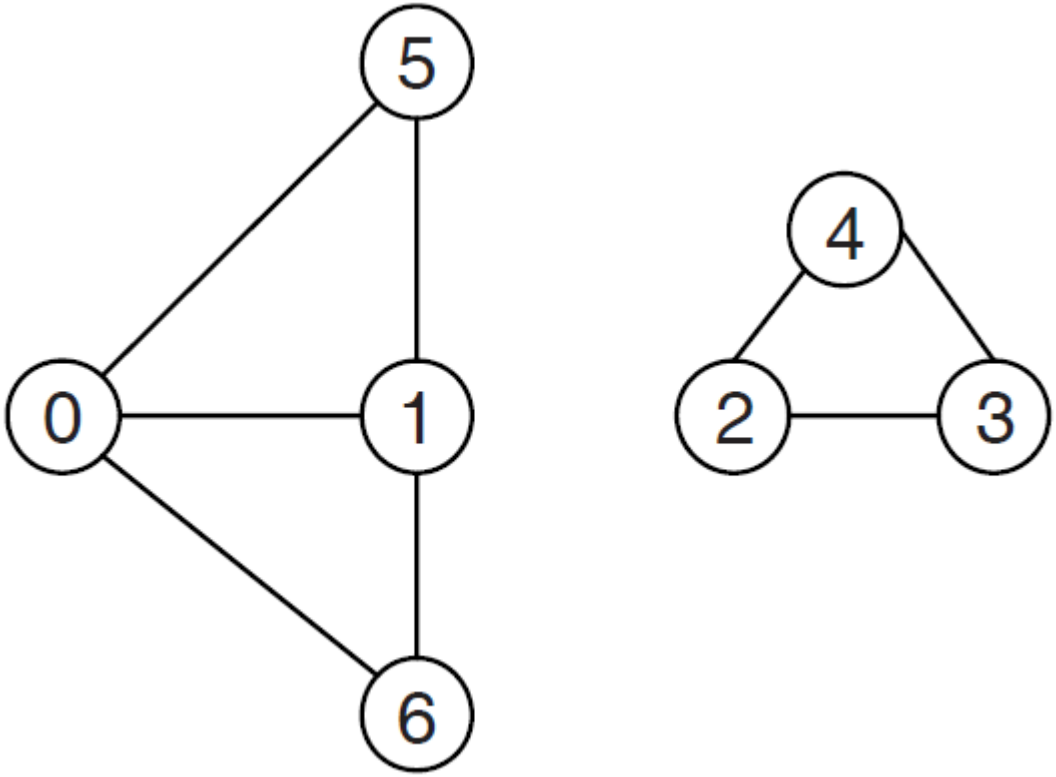
✓Além de contar o número de componentes do grafo, o algoritmo deve atribuir um rótulo *Visit*[*v*] a cada vértice *v* de tal modo que dois vértices tenham o mesmo rótulo se e somente se estão na mesma componente.

Aplicações dos Algoritmos

```
BProfundidade (G)
{
    cont ← 0;
    Para cada vértice v do grafo faça
        Visit[v] ← -1;
    Para cada vértice v do grafo faça
        Se Visit[v] == -1 então {
            cont++;
            BP(G, v, Visit, cont);
        }
     retorne cont;
}
BP (G, v, Visit[], &cont)
{
    Visit[v] ← cont;
     Para cada vértice u adjacente a v faça
        Se Visit[u] == -1 então
            BP(G, u, Visit, cont);
}
```

Aplicações dos Algoritmos

- Exemplo



Aplicações dos Algoritmos

- Para este grafo, o algoritmo retorna 2 (número de componentes do grafo)
- No início do algoritmo, o vetor *Visit*

-1	-1	-1	-1	-1	-1	-1
0	1	2	3	4	5	6

- No fim do algoritmo, o vetor *Visit*

1	1	2	2	2	1	1
0	1	2	3	4	5	6

- Os vértices 0, 1, 5 e 6 estão na componente 1; os vértices 2, 3 e 4 estão na componente 2

Código-fonte do Livro

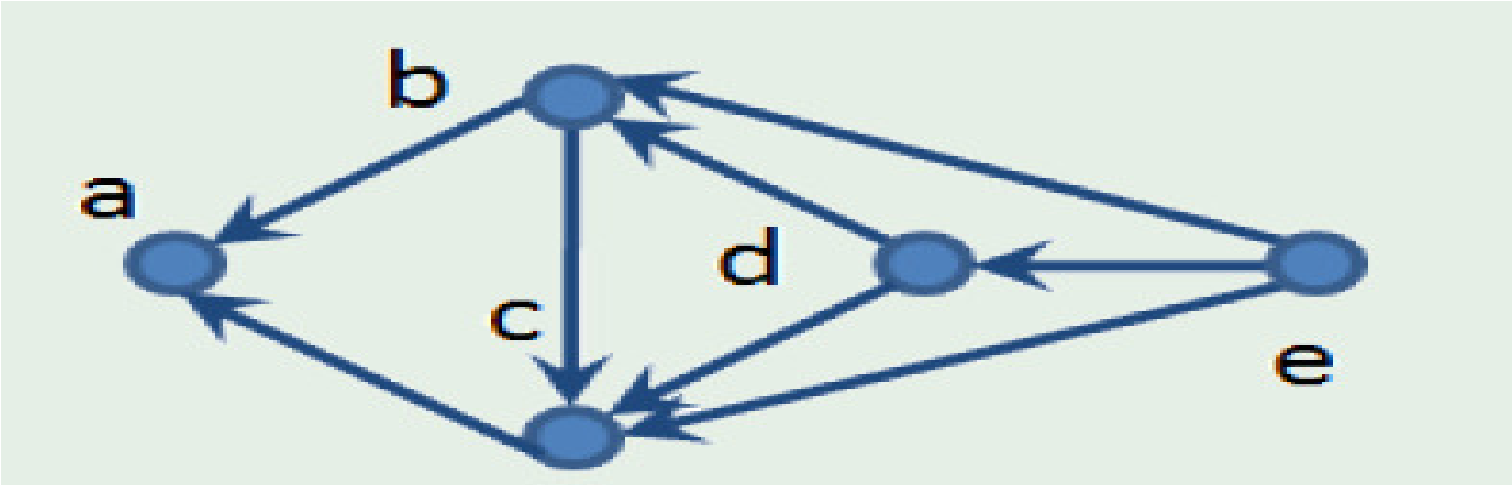
- Encontrando Caminhos (somente no livro)
- Componentes Conexas
- Encontrando Ciclos

Ordenação Topológica

➢ Definição

Um DAG (*directed acyclic graph*) é um grafo dirigido sem ciclos direcionados

✓ Cuidado! Não é necessariamente árvore nem conectado

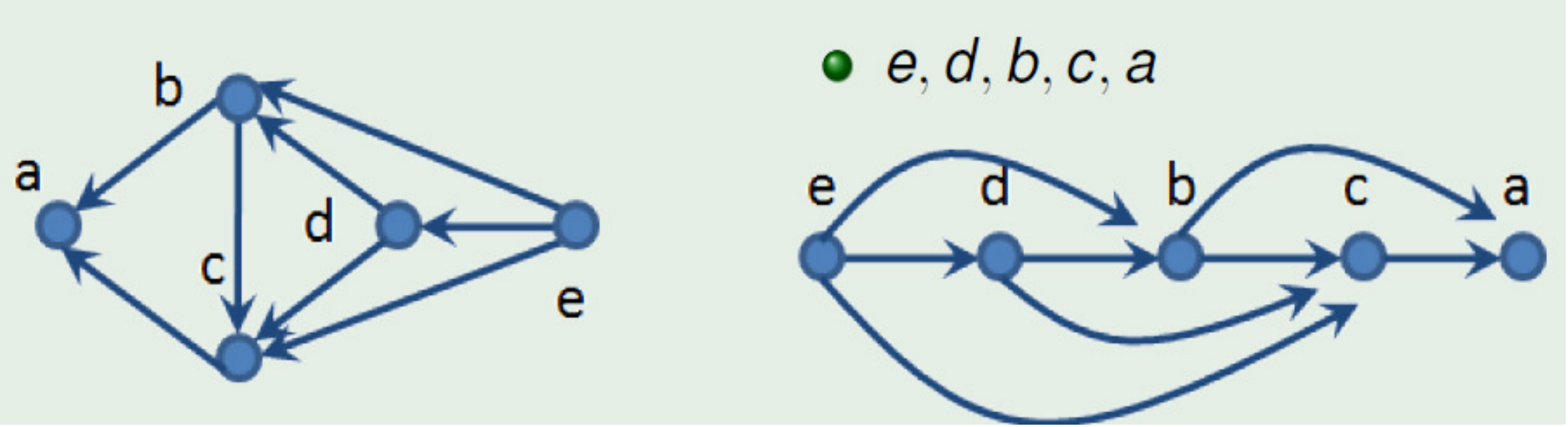


Ordenação Topológica

- Definição

Uma ordenação topológica de um DAG é uma sequência de vértices tal que não exista arco de um vértice a algum anterior na sequência

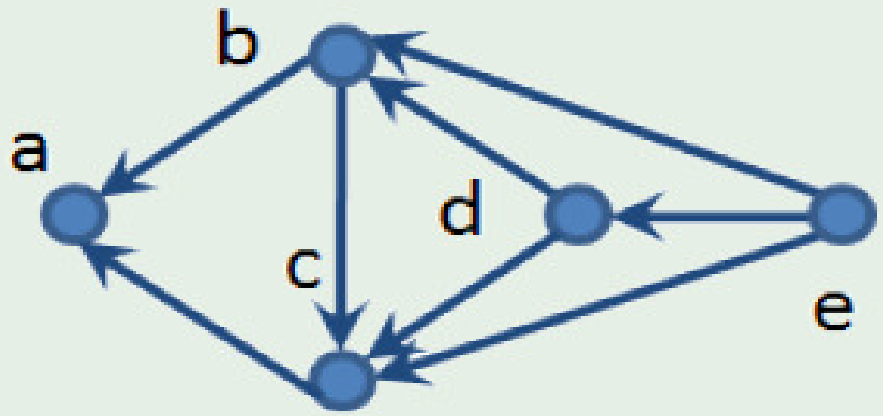
- Busca em profundidade pode ser adaptada



Ordenação Topológica

- Construir a ordenação topológica $s_1 s_2 \ldots s_n$ do DAG
- Algoritmo

para i = 1...n
    $s_i \leftarrow$ algum vértice $v \in D$ com $\delta^-(v) = 0$
    $D \leftarrow D - v$



Ordenação: e, d, b, c, a

Ordenação Topológica

- Construir a ordenação topológica $s_1 s_2 \ldots s_n$ do DAG
- Algoritmo

para i = 1...n
  $s_i \leftarrow$ algum vértice $v \in D$ com $\delta^-(v) = 0$
  $D \leftarrow D - v$

- Note que só funciona em DAG
- Se houver ciclo direcionado, faltará v com $\delta^-(v) = 0$

Ordenação Topológica

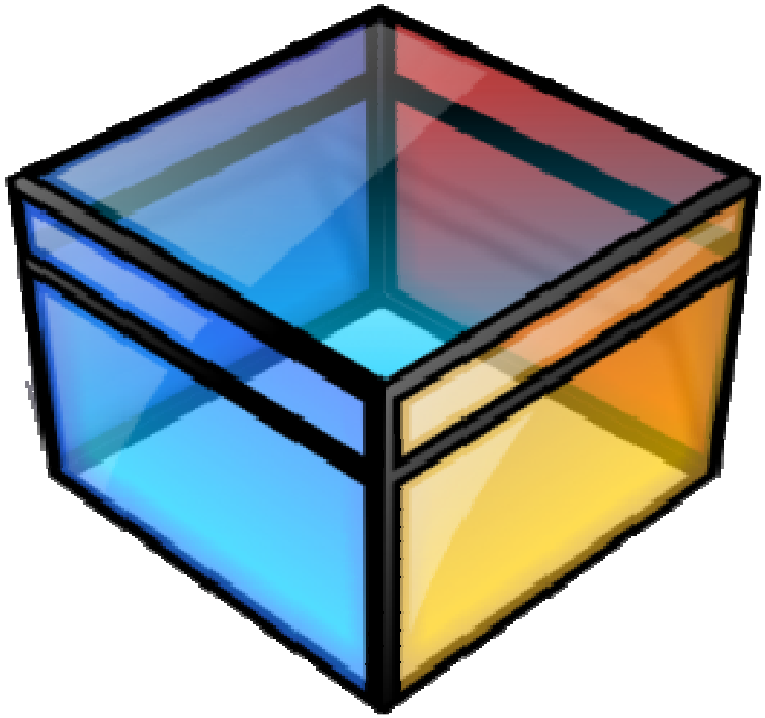
- Construir a ordenação topológica $s_1 s_2 \ldots s_n$ de um dígrafo
- Algoritmo

para $i = 1 \ldots n$
    se não há vértice $v \in D$ com $\delta^-(v) = 0$
    Erro! Não é DAG; break;
    $s_i \leftarrow$ algum vértice $v \in D$ com $\delta^-(v) = 0$
    $D \leftarrow D - v$

Exercícios

- *Bicoloring*
- *Playing With Wheels*
- *Slash Maze*
- *Hanoi Tower Troubles Again!*
- *The Tourist Guide*

Grafos



COM11087-Tópicos Especiais em Programação II

edmar.kampke@ufes.br