



# Teoria dos Grafos e Computabilidade

— Graph traversing —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF

Laboratory of Image and Multimedia Data Science – IMScience

Pontifical Catholic University of Minas Gerais – PUC Minas

# Teoria dos Grafos e Computabilidade

— Depth-First search —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF

Laboratory of Image and Multimedia Data Science – IMScience

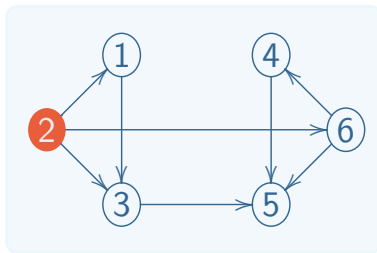
Pontifical Catholic University of Minas Gerais – PUC Minas

# Caminhamento em grafos

Na **busca em profundidade**, deve-se caminhar no grafo visitando todos os seus vértices sempre procurando o **vértice mais profundo**.

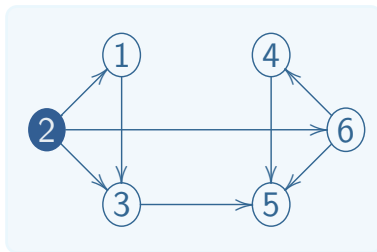
# Caminhamento em grafos

Na **busca em profundidade**, deve-se caminhar no grafo visitando todos os seus vértices sempre procurando o **vértice mais profundo**.



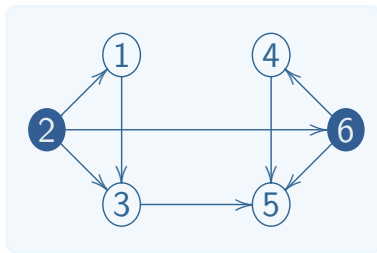
# Caminhamento em grafos

Na **busca em profundidade**, deve-se caminhar no grafo visitando todos os seus vértices sempre procurando o **vértice mais profundo**.



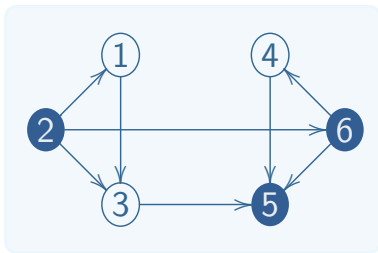
# Caminhamento em grafos

Na **busca em profundidade**, deve-se caminhar no grafo visitando todos os seus vértices sempre procurando o **vértice mais profundo**.



# Caminhamento em grafos

Na **busca em profundidade**, deve-se caminhar no grafo visitando todos os seus vértices sempre procurando o **vértice mais profundo**.



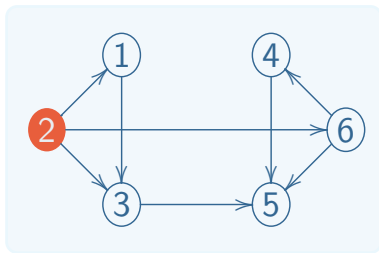
# Caminhamento em grafos

Na **busca em largura**, deve-se expandir o conjunto de vértices de forma uniforme em que são visitados todos os vértices de **mesma distância** ao início antes de visitar outros níveis.



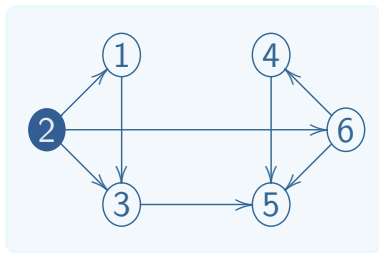
# Caminhamento em grafos

Na **busca em largura**, deve-se expandir o conjunto de vértices de forma uniforme em que são visitados todos os vértices de **mesma distância** ao início antes de visitar outros níveis.



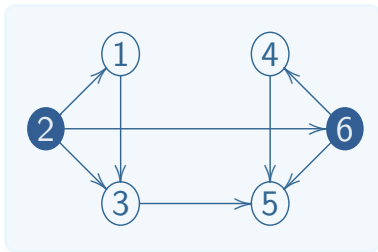
# Caminhamento em grafos

Na **busca em largura**, deve-se expandir o conjunto de vértices de forma uniforme em que são visitados todos os vértices de **mesma distância** ao início antes de visitar outros níveis.



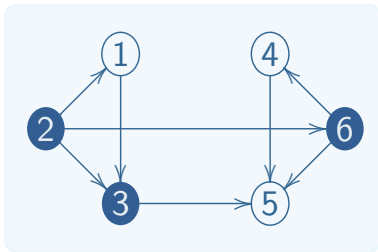
# Caminhamento em grafos

Na **busca em largura**, deve-se expandir o conjunto de vértices de forma uniforme em que são visitados todos os vértices de **mesma distância** ao início antes de visitar outros níveis.

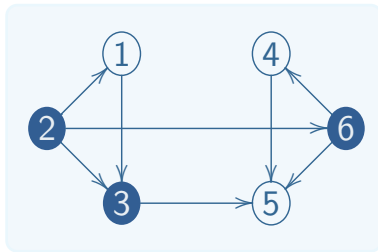
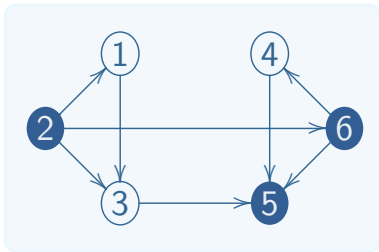


# Caminhamento em grafos

Na **busca em largura**, deve-se expandir o conjunto de vértices de forma uniforme em que são visitados todos os vértices de **mesma distância** ao início antes de visitar outros níveis.



# Diferença entre os caminhamentos

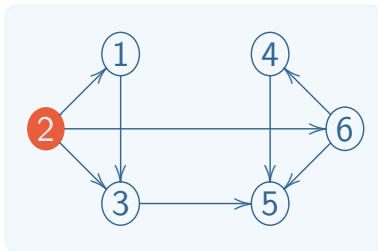


# Busca em profundidade

- As arestas são exploradas a partir do vértice  $v$  mais recentemente descoberto que ainda tem arestas não descobertas saindo dele;

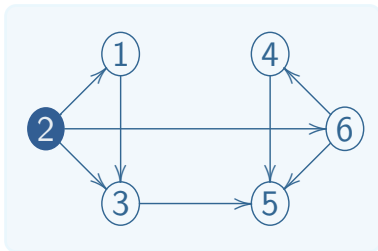
# Busca em profundidade

- As arestas são exploradas a partir do vértice  $v$  mais **recentemente descoberto** que ainda tem arestas não descobertas saindo dele;



# Busca em profundidade

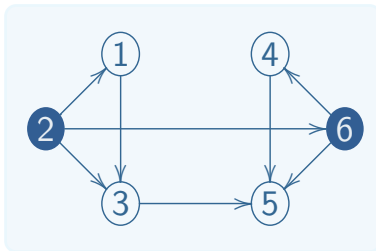
- As arestas são exploradas a partir do vértice  $v$  mais **recentemente descoberto** que ainda tem arestas não descobertas saindo dele;





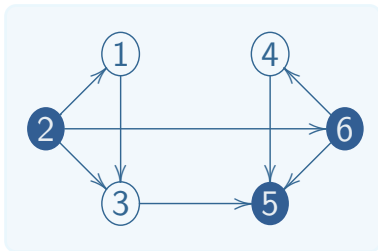
# Busca em profundidade

- As arestas são exploradas a partir do vértice  $v$  mais **recentemente descoberto** que ainda tem arestas não descobertas saindo dele;



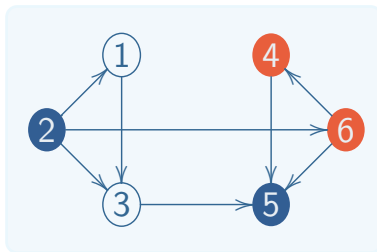
# Busca em profundidade

- As arestas são exploradas a partir do vértice  $v$  mais **recentemente descoberto** que ainda tem arestas não descobertas saindo dele;



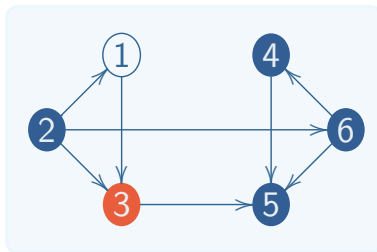
# Busca em profundidade

- ▶ As arestas são exploradas a partir do vértice  $v$  mais **recentemente descoberto** que ainda tem arestas não descobertas saindo dele;
- ▶ Quando todas as arestas de  $v$  tiverem sido exploradas **volta-se** até para explorar arestas que saem do vértice a partir do qual  $v$  foi descoberto.



# Busca em profundidade

- ▶ As arestas são exploradas a partir do vértice  $v$  mais **recentemente descoberto** que ainda tem arestas não descobertas saindo dele;
- ▶ Quando todas as arestas de  $v$  tiverem sido exploradas **volta-se** até para explorar arestas que saem do vértice a partir do qual  $v$  foi descoberto.

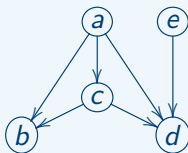


# Busca em profundidade

- ▶ Todos os vértice são inicializados com **branco**
- ▶ Quando um vértice é visitado pela primeira vez ele torna-se **azul**
- ▶ Quando sua lista de adjacentes foi totalmente explorada ele torna-se **vermelho**

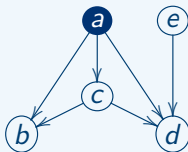
# Busca em profundidade

- ▶ Todos os vértice são inicializados com **branco**
- ▶ Quando um vértice é visitado pela primeira vez ele torna-se **azul**
- ▶ Quando sua lista de adjacentes foi totalmente explorada ele torna-se **vermelho**



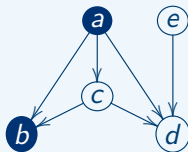
# Busca em profundidade

- ▶ Todos os vértice são inicializados com **branco**
- ▶ Quando um vértice é visitado pela primeira vez ele torna-se **azul**
- ▶ Quando sua lista de adjacentes foi totalmente explorada ele torna-se **vermelho**



# Busca em profundidade

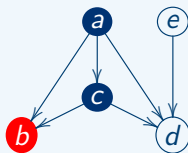
- ▶ Todos os vértice são inicializados com **branco**
- ▶ Quando um vértice é visitado pela primeira vez ele torna-se **azul**
- ▶ Quando sua lista de adjacentes foi totalmente explorada ele torna-se **vermelho**





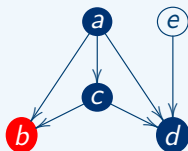
# Busca em profundidade

- ▶ Todos os vértice são inicializados com **branco**
- ▶ Quando um vértice é visitado pela primeira vez ele torna-se **azul**
- ▶ Quando sua lista de adjacentes foi totalmente explorada ele torna-se **vermelho**



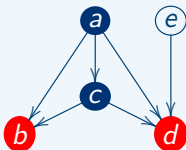
# Busca em profundidade

- ▶ Todos os vértice são inicializados com **branco**
- ▶ Quando um vértice é visitado pela primeira vez ele torna-se **azul**
- ▶ Quando sua lista de adjacentes foi totalmente explorada ele torna-se **vermelho**



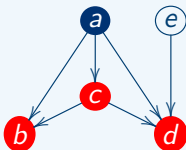
# Busca em profundidade

- ▶ Todos os vértice são inicializados com **branco**
- ▶ Quando um vértice é visitado pela primeira vez ele torna-se **azul**
- ▶ Quando sua lista de adjacentes foi totalmente explorada ele torna-se **vermelho**



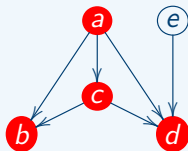
# Busca em profundidade

- ▶ Todos os vértice são inicializados com **branco**
- ▶ Quando um vértice é visitado pela primeira vez ele torna-se **azul**
- ▶ Quando sua lista de adjacentes foi totalmente explorada ele torna-se **vermelho**



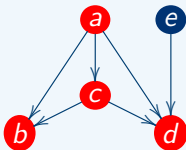
# Busca em profundidade

- ▶ Todos os vértice são inicializados com **branco**
- ▶ Quando um vértice é visitado pela primeira vez ele torna-se **azul**
- ▶ Quando sua lista de adjacentes foi totalmente explorada ele torna-se **vermelho**



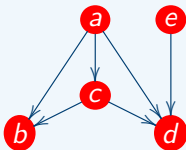
# Busca em profundidade

- ▶ Todos os vértice são inicializados com **branco**
- ▶ Quando um vértice é visitado pela primeira vez ele torna-se **azul**
- ▶ Quando sua lista de adjacentes foi totalmente explorada ele torna-se **vermelho**



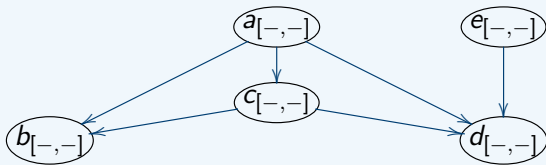
# Busca em profundidade

- ▶ Todos os vértice são inicializados com **branco**
- ▶ Quando um vértice é visitado pela primeira vez ele torna-se **azul**
- ▶ Quando sua lista de adjacentes foi totalmente explorada ele torna-se **vermelho**



# Busca em profundidade

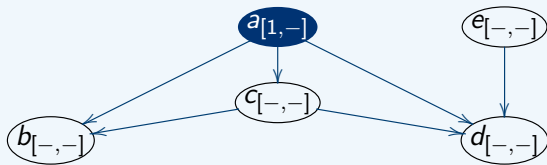
- ▶ O tempo de **descoberta**  $d[v]$  é o momento em que o vértice  $v$  foi visitado pela **primeira vez**
- ▶ O tempo de **término** do exame da lista de adjacentes  $t[v]$  é o momento em que a visita a **toda lista** de vértices adjacentes a  $v$  foi concluída.
- ▶  $d[v]$  e  $t[v]$  são inteiros entre 1 e  $2V$ , onde  $V$  é o número de vértices do grafo





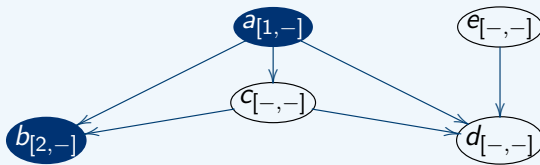
# Busca em profundidade

- ▶ O tempo de **descoberta**  $d[v]$  é o momento em que o vértice  $v$  foi visitado pela **primeira vez**
- ▶ O tempo de **término** do exame da lista de adjacentes  $t[v]$  é o momento em que a visita a **toda lista** de vértices adjacentes a  $v$  foi concluída.
- ▶  $d[v]$  e  $t[v]$  são inteiros entre 1 e  $2V$ , onde  $V$  é o número de vértices do grafo



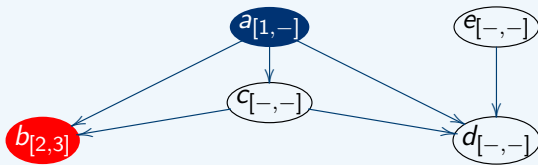
# Busca em profundidade

- ▶ O tempo de **descoberta**  $d[v]$  é o momento em que o vértice  $v$  foi visitado pela **primeira vez**
- ▶ O tempo de **término** do exame da lista de adjacentes  $t[v]$  é o momento em que a visita a **toda lista** de vértices adjacentes a  $v$  foi concluída.
- ▶  $d[v]$  e  $t[v]$  são inteiros entre 1 e  $2V$ , onde  $V$  é o número de vértices do grafo



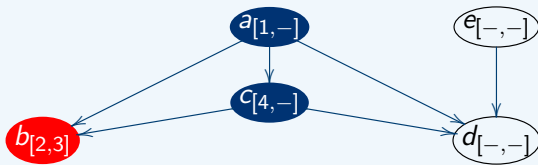
# Busca em profundidade

- ▶ O tempo de **descoberta**  $d[v]$  é o momento em que o vértice  $v$  foi visitado pela **primeira vez**
- ▶ O tempo de **término** do exame da lista de adjacentes  $t[v]$  é o momento em que a visita a **toda lista** de vértices adjacentes a  $v$  foi concluída.
- ▶  $d[v]$  e  $t[v]$  são inteiros entre 1 e  $2V$ , onde  $V$  é o número de vértices do grafo



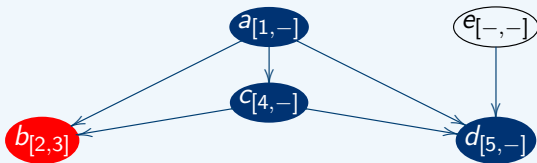
# Busca em profundidade

- ▶ O tempo de **descoberta**  $d[v]$  é o momento em que o vértice  $v$  foi visitado pela **primeira vez**
- ▶ O tempo de **término** do exame da lista de adjacentes  $t[v]$  é o momento em que a visita a **toda lista** de vértices adjacentes a  $v$  foi concluída.
- ▶  $d[v]$  e  $t[v]$  são inteiros entre 1 e  $2V$ , onde  $V$  é o número de vértices do grafo



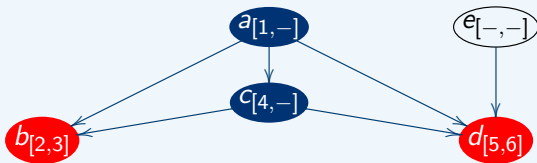
# Busca em profundidade

- ▶ O tempo de **descoberta**  $d[v]$  é o momento em que o vértice  $v$  foi visitado pela **primeira vez**
- ▶ O tempo de **término** do exame da lista de adjacentes  $t[v]$  é o momento em que a visita a **toda lista** de vértices adjacentes a  $v$  foi concluída.
- ▶  $d[v]$  e  $t[v]$  são inteiros entre 1 e  $2V$ , onde  $V$  é o número de vértices do grafo



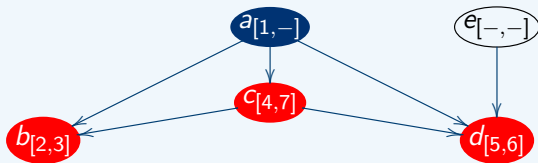
# Busca em profundidade

- ▶ O tempo de **descoberta**  $d[v]$  é o momento em que o vértice  $v$  foi visitado pela **primeira vez**
- ▶ O tempo de **término** do exame da lista de adjacentes  $t[v]$  é o momento em que a visita a **toda lista** de vértices adjacentes a  $v$  foi concluída.
- ▶  $d[v]$  e  $t[v]$  são inteiros entre 1 e  $2V$ , onde  $V$  é o número de vértices do grafo



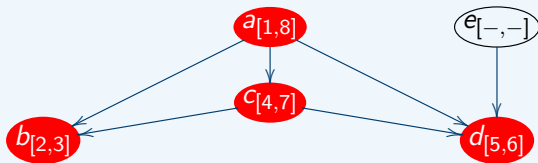
# Busca em profundidade

- ▶ O tempo de **descoberta**  $d[v]$  é o momento em que o vértice  $v$  foi visitado pela **primeira vez**
- ▶ O tempo de **término** do exame da lista de adjacentes  $t[v]$  é o momento em que a visita a **toda lista** de vértices adjacentes a  $v$  foi concluída.
- ▶  $d[v]$  e  $t[v]$  são inteiros entre 1 e  $2V$ , onde  $V$  é o número de vértices do grafo



# Busca em profundidade

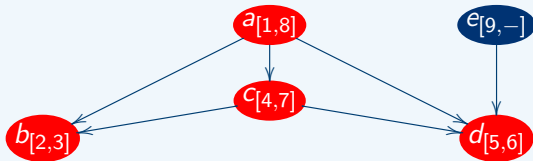
- ▶ O tempo de **descoberta**  $d[v]$  é o momento em que o vértice  $v$  foi visitado pela **primeira vez**
- ▶ O tempo de **término** do exame da lista de adjacentes  $t[v]$  é o momento em que a visita a **toda lista** de vértices adjacentes a  $v$  foi concluída.
- ▶  $d[v]$  e  $t[v]$  são inteiros entre 1 e  $2V$ , onde  $V$  é o número de vértices do grafo





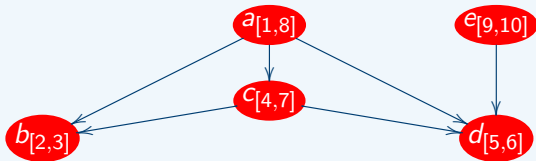
# Busca em profundidade

- ▶ O tempo de **descoberta**  $d[v]$  é o momento em que o vértice  $v$  foi visitado pela **primeira vez**
- ▶ O tempo de **término** do exame da lista de adjacentes  $t[v]$  é o momento em que a visita a **toda lista** de vértices adjacentes a  $v$  foi concluída.
- ▶  $d[v]$  e  $t[v]$  são inteiros entre 1 e  $2V$ , onde  $V$  é o número de vértices do grafo



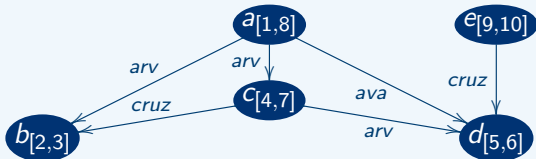
# Busca em profundidade

- ▶ O tempo de **descoberta**  $d[v]$  é o momento em que o vértice  $v$  foi visitado pela **primeira vez**
- ▶ O tempo de **término** do exame da lista de adjacentes  $t[v]$  é o momento em que a visita a **toda lista** de vértices adjacentes a  $v$  foi concluída.
- ▶  $d[v]$  e  $t[v]$  são inteiros entre 1 e  $2V$ , onde  $V$  é o número de vértices do grafo



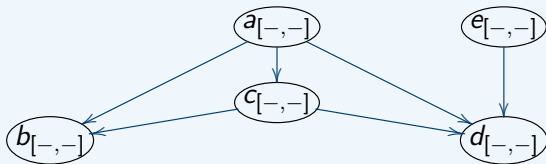
# Classificação das arestas

- ▶ De árvore: uma aresta  $(u,v)$  é de **árvore** se o vértice  $v$  foi visitado a primeira vez passando pela aresta  $(u,v)$
- ▶ De retorno: uma aresta  $(u,v)$  é uma aresta de **retorno** se esta conecta um vértice  $u$  com um predecessor  $v$  já presente em uma árvore de busca
- ▶ De avanço: Não pertencem a árvore de busca em profundidade mas conectam um vértice a um **descendente** que pertence a árvore de busca
- ▶ De cruzamento: conectam vértice de uma **mesma árvore** de busca ou de árvores diferentes



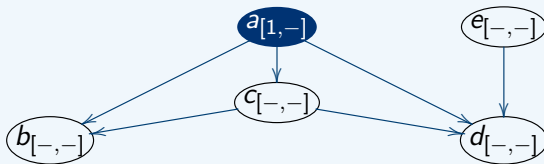
# Busca em profundidade

- ▶ As arestas  $e = (u, v)$  podem ser classificadas pela cor do vértice  $v$  que é alcançado quando se passa pela aresta e primeira vez
  - ▶ Branco : aresta de árvore
  - ▶ Azul : aresta de retorno
  - ▶ Vermelho : (i) Se  $u$  é visitado antes de  $v$  então  $e$  é uma aresta de avanço; (ii) Se  $v$  é visitado antes de  $u$  então  $e$  é de cruzamento



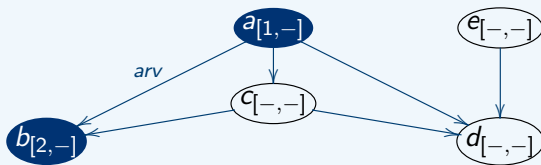
# Busca em profundidade

- ▶ As arestas  $e = (u, v)$  podem ser classificadas pela cor do vértice  $v$  que é alcançado quando se passa pela aresta e primeira vez
  - ▶ Branco : aresta de árvore
  - ▶ Azul : aresta de retorno
  - ▶ Vermelho : (i) Se  $u$  é visitado antes de  $v$  então  $e$  é uma aresta de avanço; (ii) Se  $v$  é visitado antes de  $u$  então  $e$  é de cruzamento



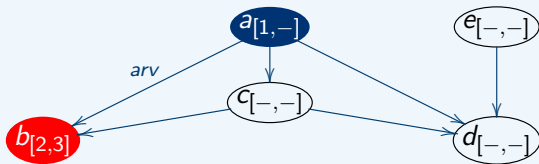
# Busca em profundidade

- ▶ As arestas  $e = (u, v)$  podem ser classificadas pela cor do vértice  $v$  que é alcançado quando se passa pela aresta e primeira vez
  - ▶ Branco : aresta de árvore
  - ▶ Azul : aresta de retorno
  - ▶ Vermelho : (i) Se  $u$  é visitado antes de  $v$  então  $e$  é uma aresta de avanço; (ii) Se  $v$  é visitado antes de  $u$  então  $e$  é de cruzamento



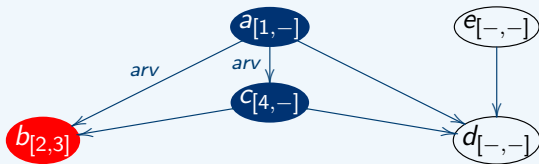
# Busca em profundidade

- ▶ As arestas  $e = (u, v)$  podem ser classificadas pela cor do vértice  $v$  que é alcançado quando se passa pela aresta e primeira vez
  - ▶ Branco : aresta de árvore
  - ▶ Azul : aresta de retorno
  - ▶ Vermelho : (i) Se  $u$  é visitado antes de  $v$  então  $e$  é uma aresta de avanço; (ii) Se  $v$  é visitado antes de  $u$  então  $e$  é de cruzamento



# Busca em profundidade

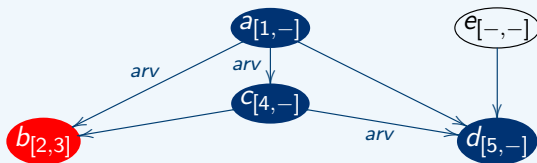
- ▶ As arestas  $e = (u, v)$  podem ser classificadas pela cor do vértice  $v$  que é alcançado quando se passa pela aresta e primeira vez
  - ▶ Branco : aresta de árvore
  - ▶ Azul : aresta de retorno
  - ▶ Vermelho : (i) Se  $u$  é visitado antes de  $v$  então  $e$  é uma aresta de avanço; (ii) Se  $v$  é visitado antes de  $u$  então  $e$  é de cruzamento





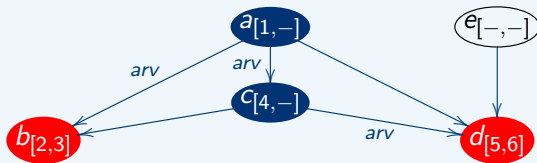
# Busca em profundidade

- ▶ As arestas  $e = (u, v)$  podem ser classificadas pela cor do vértice  $v$  que é alcançado quando se passa pela aresta e primeira vez
  - ▶ Branco : aresta de árvore
  - ▶ Azul : aresta de retorno
  - ▶ Vermelho : (i) Se  $u$  é visitado antes de  $v$  então  $e$  é uma aresta de avanço; (ii) Se  $v$  é visitado antes de  $u$  então  $e$  é de cruzamento



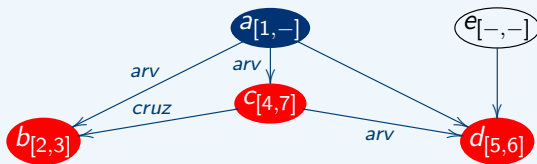
# Busca em profundidade

- ▶ As arestas  $e = (u, v)$  podem ser classificadas pela cor do vértice  $v$  que é alcançado quando se passa pela aresta e primeira vez
  - ▶ Branco : aresta de árvore
  - ▶ Azul : aresta de retorno
  - ▶ Vermelho : (i) Se  $u$  é visitado antes de  $v$  então  $e$  é uma aresta de avanço; (ii) Se  $v$  é visitado antes de  $u$  então  $e$  é de cruzamento



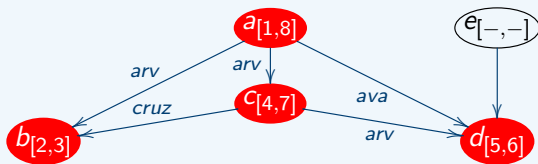
# Busca em profundidade

- ▶ As arestas  $e = (u, v)$  podem ser classificadas pela cor do vértice  $v$  que é alcançado quando se passa pela aresta e primeira vez
  - ▶ Branco : aresta de árvore
  - ▶ Azul : aresta de retorno
  - ▶ Vermelho : (i) Se  $u$  é visitado antes de  $v$  então  $e$  é uma aresta de avanço; (ii) Se  $v$  é visitado antes de  $u$  então  $e$  é de cruzamento



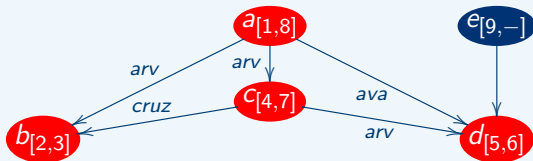
# Busca em profundidade

- ▶ As arestas  $e = (u, v)$  podem ser classificadas pela cor do vértice  $v$  que é alcançado quando se passa pela aresta e primeira vez
  - ▶ Branco : aresta de árvore
  - ▶ Azul : aresta de retorno
  - ▶ Vermelho : (i) Se  $u$  é visitado antes de  $v$  então  $e$  é uma aresta de avanço; (ii) Se  $v$  é visitado antes de  $u$  então  $e$  é de cruzamento



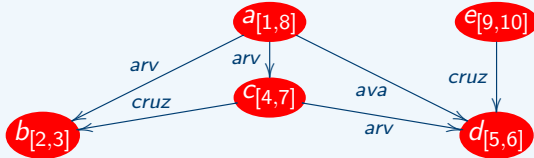
# Busca em profundidade

- ▶ As arestas  $e = (u, v)$  podem ser classificadas pela cor do vértice  $v$  que é alcançado quando se passa pela aresta e primeira vez
  - ▶ Branco : aresta de árvore
  - ▶ Azul : aresta de retorno
  - ▶ Vermelho : (i) Se  $u$  é visitado antes de  $v$  então  $e$  é uma aresta de avanço; (ii) Se  $v$  é visitado antes de  $u$  então  $e$  é de cruzamento



# Busca em profundidade

- ▶ As arestas  $e = (u, v)$  podem ser classificadas pela cor do vértice  $v$  que é alcançado quando se passa pela aresta e primeira vez
  - ▶ Branco : aresta de árvore
  - ▶ Azul : aresta de retorno
  - ▶ Vermelho : (i) Se  $u$  é visitado antes de  $v$  então  $e$  é uma aresta de avanço; (ii) Se  $v$  é visitado antes de  $u$  então  $e$  é de cruzamento



## Exemplo 1

Como verificar se há um circuito em um grafo direcionado?



Programa de  
Pós-graduação em

informática



PUC Minas



# Teoria dos Grafos e Computabilidade

— Breadth-First search —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF

Laboratory of Image and Multimedia Data Science – IMScience

Pontifical Catholic University of Minas Gerais – PUC Minas

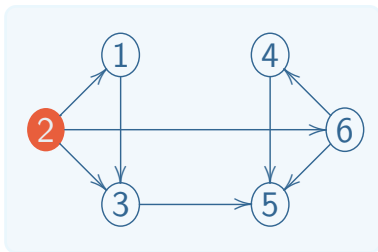


# Busca em largura

- ▶ Na busca em largura, deve-se **expandir** o conjunto de vértices de forma uniforme em que são visitados todos os vértices de **mesma distância** ao início antes de visitar outros níveis.

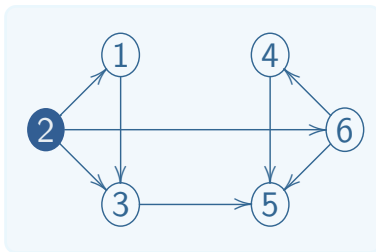
# Busca em largura

- Na busca em largura, deve-se **expandir** o conjunto de vértices de forma uniforme em que são visitados todos os vértices de **mesma distância** ao início antes de visitar outros níveis.



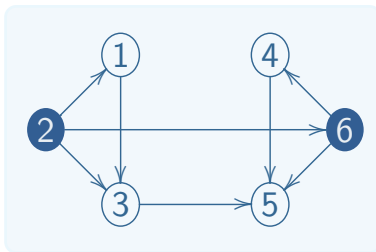
# Busca em largura

- ▶ Na busca em largura, deve-se **expandir** o conjunto de vértices de forma uniforme em que são visitados todos os vértices de **mesma distância** ao início antes de visitar outros níveis.



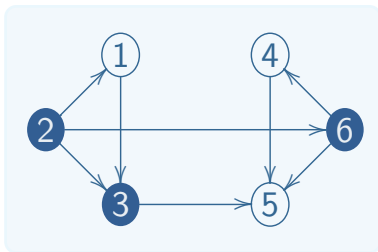
# Busca em largura

- Na busca em largura, deve-se **expandir** o conjunto de vértices de forma uniforme em que são visitados todos os vértices de **mesma distância** ao início antes de visitar outros níveis.



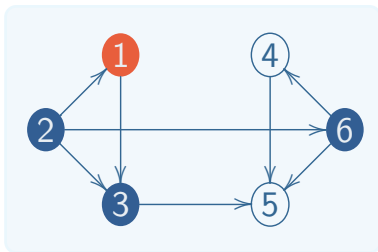
# Busca em largura

- ▶ Na busca em largura, deve-se **expandir** o conjunto de vértices de forma uniforme em que são visitados todos os vértices de **mesma distância** ao início antes de visitar outros níveis.



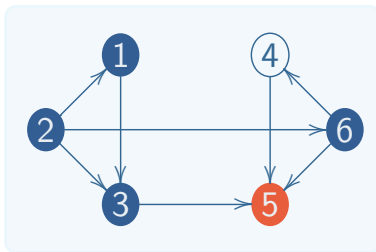
# Busca em largura

- Na busca em largura, deve-se **expandir** o conjunto de vértices de forma uniforme em que são visitados todos os vértices de **mesma distância** ao início antes de visitar outros níveis.



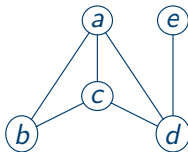
# Busca em largura

- ▶ Na busca em largura, deve-se **expandir** o conjunto de vértices de forma uniforme em que são visitados todos os vértices de **mesma distância** ao início antes de visitar outros níveis.
- ▶ Na busca em largura o algoritmo descobre todos os vertices a uma distância  $k$  do vértice de origem antes de descobrir os que estão a uma distância  $k + 1$



# Busca em largura

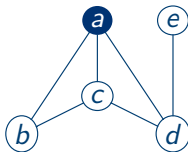
- ▶ Cada vértice é colorido de branco, azul ou vermelho
- ▶ Todos os vértices são inicializados com branco
- ▶ Quando um vértice é descoberto pela primeira vez ele torna-se azul
- ▶ Vértices cujos adjacentes são todos descobertos tornam-se vermelhos. Atenção, pois não será necessário todos os descendentes serem descobertos para se tornar vermelho.
- ▶ Se  $(u, v) \in A$  e o vértice  $u$  é vermelho, então  $v$  tem que ser azul ou vermelho
- ▶ Vértices azul podem ter adjacentes brancos.





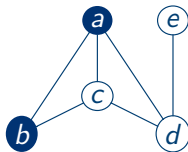
# Busca em largura

- ▶ Cada vértice é colorido de branco, azul ou vermelho
- ▶ Todos os vértices são inicializados com branco
- ▶ Quando um vértice é descoberto pela primeira vez ele torna-se azul
- ▶ Vértices cujos adjacentes são todos descobertos tornam-se vermelhos. Atenção, pois não será necessário todos os descendentes serem descobertos para se tornar vermelho.
- ▶ Se  $(u, v) \in A$  e o vértice  $u$  é vermelho, então  $v$  tem que ser azul ou vermelho
- ▶ Vértices azul podem ter adjacentes brancos.



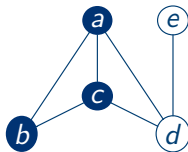
# Busca em largura

- ▶ Cada vértice é colorido de branco, azul ou vermelho
- ▶ Todos os vértices são inicializados com branco
- ▶ Quando um vértice é descoberto pela primeira vez ele torna-se azul
- ▶ Vértices cujos adjacentes são todos descobertos tornam-se vermelhos. Atenção, pois não será necessário todos os descendentes serem descobertos para se tornar vermelho.
- ▶ Se  $(u, v) \in A$  e o vértice  $u$  é vermelho, então  $v$  tem que ser azul ou vermelho
- ▶ Vértices azul podem ter adjacentes brancos.



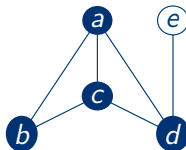
# Busca em largura

- ▶ Cada vértice é colorido de branco, azul ou vermelho
- ▶ Todos os vértices são inicializados com branco
- ▶ Quando um vértice é descoberto pela primeira vez ele torna-se azul
- ▶ Vértices cujos adjacentes são todos descobertos tornam-se vermelhos. Atenção, pois não será necessário todos os descendentes serem descobertos para se tornar vermelho.
- ▶ Se  $(u, v) \in A$  e o vértice  $u$  é vermelho, então  $v$  tem que ser azul ou vermelho
- ▶ Vértices azul podem ter adjacentes brancos.



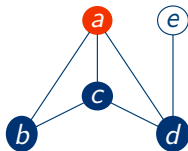
# Busca em largura

- ▶ Cada vértice é colorido de branco, azul ou vermelho
- ▶ Todos os vértices são inicializados com branco
- ▶ Quando um vértice é descoberto pela primeira vez ele torna-se azul
- ▶ Vértices cujos adjacentes são todos descobertos tornam-se vermelhos. Atenção, pois não será necessário todos os descendentes serem descobertos para se tornar vermelho.
- ▶ Se  $(u, v) \in A$  e o vértice  $u$  é vermelho, então  $v$  tem que ser azul ou vermelho
- ▶ Vértices azul podem ter adjacentes brancos.



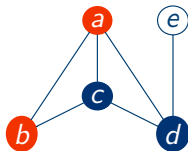
# Busca em largura

- ▶ Cada vértice é colorido de branco, azul ou vermelho
- ▶ Todos os vértices são inicializados com branco
- ▶ Quando um vértice é descoberto pela primeira vez ele torna-se azul
- ▶ Vértices cujos adjacentes são todos descobertos tornam-se vermelhos. Atenção, pois não será necessário todos os descendentes serem descobertos para se tornar vermelho.
- ▶ Se  $(u, v) \in A$  e o vértice  $u$  é vermelho, então  $v$  tem que ser azul ou vermelho
- ▶ Vértices azul podem ter adjacentes brancos.



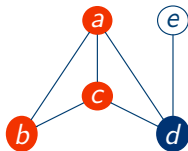
# Busca em largura

- ▶ Cada vértice é colorido de branco, azul ou vermelho
- ▶ Todos os vértices são inicializados com branco
- ▶ Quando um vértice é descoberto pela primeira vez ele torna-se azul
- ▶ Vértices cujos adjacentes são todos descobertos tornam-se vermelhos. Atenção, pois não será necessário todos os descendentes serem descobertos para se tornar vermelho.
- ▶ Se  $(u, v) \in A$  e o vértice  $u$  é vermelho, então  $v$  tem que ser azul ou vermelho
- ▶ Vértices azul podem ter adjacentes brancos.



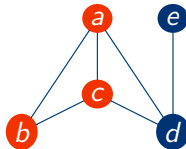
# Busca em largura

- ▶ Cada vértice é colorido de branco, azul ou vermelho
- ▶ Todos os vértices são inicializados com branco
- ▶ Quando um vértice é descoberto pela primeira vez ele torna-se azul
- ▶ Vértices cujos adjacentes são todos descobertos tornam-se vermelhos. Atenção, pois não será necessário todos os descendentes serem descobertos para se tornar vermelho.
- ▶ Se  $(u, v) \in A$  e o vértice  $u$  é vermelho, então  $v$  tem que ser azul ou vermelho
- ▶ Vértices azul podem ter adjacentes brancos.



# Busca em largura

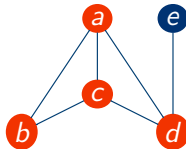
- ▶ Cada vértice é colorido de branco, azul ou vermelho
- ▶ Todos os vértices são inicializados com branco
- ▶ Quando um vértice é descoberto pela primeira vez ele torna-se azul
- ▶ Vértices cujos adjacentes são todos descobertos tornam-se vermelhos. Atenção, pois não será necessário todos os descendentes serem descobertos para se tornar vermelho.
- ▶ Se  $(u, v) \in A$  e o vértice  $u$  é vermelho, então  $v$  tem que ser azul ou vermelho
- ▶ Vértices azul podem ter adjacentes brancos.





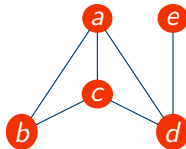
# Busca em largura

- ▶ Cada vértice é colorido de branco, azul ou vermelho
- ▶ Todos os vértices são inicializados com branco
- ▶ Quando um vértice é descoberto pela primeira vez ele torna-se azul
- ▶ Vértices cujos adjacentes são todos descobertos tornam-se vermelhos. Atenção, pois não será necessário todos os descendentes serem descobertos para se tornar vermelho.
- ▶ Se  $(u, v) \in A$  e o vértice  $u$  é vermelho, então  $v$  tem que ser azul ou vermelho
- ▶ Vértices azul podem ter adjacentes brancos.



# Busca em largura

- ▶ Cada vértice é colorido de branco, azul ou vermelho
- ▶ Todos os vértices são inicializados com branco
- ▶ Quando um vértice é descoberto pela primeira vez ele torna-se azul
- ▶ Vértices cujos adjacentes são todos descobertos tornam-se vermelhos. Atenção, pois não será necessário todos os descendentes serem descobertos para se tornar vermelho.
- ▶ Se  $(u, v) \in A$  e o vértice  $u$  é vermelho, então  $v$  tem que ser azul ou vermelho
- ▶ Vértices azul podem ter adjacentes brancos.



## Caminho mais curto

- ▶ A busca em largura encontra o **caminho mais curto** entre dois vértice  $u$  e  $v$ .
- ▶ O caminho entre dois vertices quaisquer fica armazenado no vetor antecessor

## Caminho mais curto

- ▶ A busca em largura encontra o **caminho mais curto** entre dois vértice  $u$  e  $v$ .
- ▶ O caminho entre dois vertices quaisquer fica armazenado no vetor antecessor

## Ordenação topológica

- ▶ Grafos direcionados **acíclicos** pode ser usados para indicar precedência de eventos
- ▶ Uma aresta direcionada  $(u, v)$  indica que a atividade  $u$  tem que ocorrer antes da atividade  $v$
- ▶ Os vértices ordenados topologicamente aparecem em **ordem inversa** aos seus tempos de término na busca em profundidade

# Algoritmo menor caminho

Sejam  $G = (V, A)$  um grafo direcionado e  $c_{ij}$  as distâncias associadas à aresta  $(i, j) \in A$ . Espera-se encontrar o **caminho mais curto** entre dois vértices  $s$  e  $t$ !!

# Algoritmo menor caminho

Sejam  $G = (V, A)$  um grafo direcionado e  $c_{ij}$  as distâncias associadas à aresta  $(i, j) \in A$ . Espera-se encontrar o **caminho mais curto** entre dois vértices  $s$  e  $t$ !!!

O **comprimento de um caminho** é igual à **soma** dos comprimentos (distâncias) das arestas que formam o caminho. A **distância** ou **comprimento** de uma aresta pode ter diversas interpretações dependendo da aplicação: custos, distâncias, consumo de combustível, etc.

# Algoritmo menor caminho

Sejam  $G = (V, A)$  um grafo direcionado e  $c_{ij}$  as distâncias associadas à aresta  $(i, j) \in A$ . Espera-se encontrar o **caminho mais curto** entre dois vértices  $s$  e  $t$ !!!

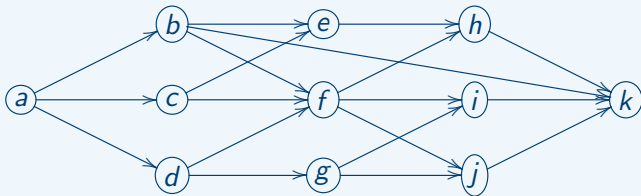
O **comprimento de um caminho** é igual à **soma** dos comprimentos (distâncias) das arestas que formam o caminho. A **distância** ou **comprimento** de uma aresta pode ter diversas interpretações dependendo da aplicação: custos, distâncias, consumo de combustível, etc.

## Exemplo 2

Dado um mapa rodoviário, determinar a **rota mais curta** de uma cidade a outra (rota mais rápida, rota com menor consumo de combustível, rota com menor valor de pedágio)

## Exemplo 3

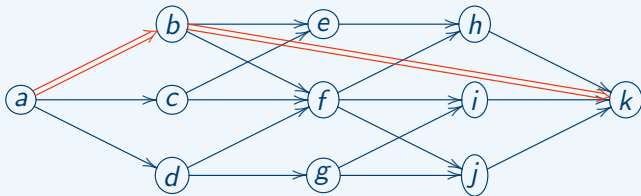
Construção de uma estrada entre duas cidades A e K, representado pelo grafo G com diversos trechos possíveis. Determinar o trajeto ótimo (corresponde a achar o **caminho mais curto** de A a K em relação a estes custos).





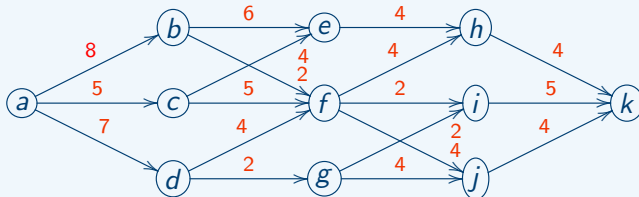
## Exemplo 3

Construção de uma estrada entre duas cidades A e K, representado pelo grafo G com diversos trechos possíveis. Determinar o trajeto ótimo (corresponde a achar o **caminho mais curto** de A a K em relação a estes custos).



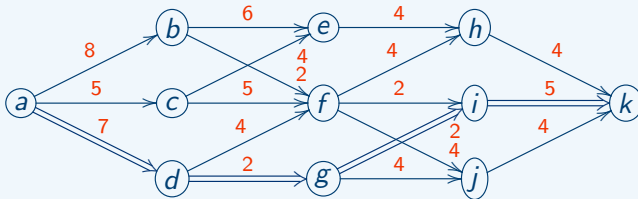
## Exemplo 4

Construção de uma estrada entre duas cidades A e K, representado pelo grafo G com diversos trechos possíveis e o custo de construção de cada um. Determinar o trajeto ótimo cujo custo de construção seja mínimo (corresponde a achar o caminho mais curto de A a K em relação a estes custos).



## Exemplo 4

Construção de uma estrada entre duas cidades A e K, representado pelo grafo G com diversos trechos possíveis e o custo de construção de cada um. Determinar o trajeto ótimo cujo custo de construção seja mínimo (corresponde a achar o caminho mais curto de A a K em relação a estes custos).



# Teoria dos Grafos e Computabilidade

— Distances over the graph —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF

Laboratory of Image and Multimedia Data Science – IMScience

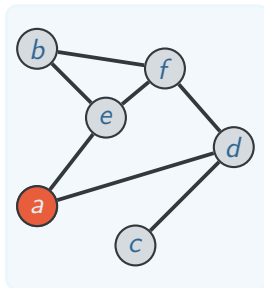
Pontifical Catholic University of Minas Gerais – PUC Minas

# Distances

The **distance** between two vertices in a graph is the **number** of edges in a **shortest path** connecting them.

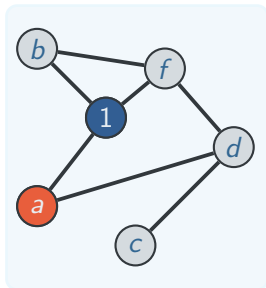
# Distances

The **distance** between two vertices in a graph is the **number** of edges in a **shortest path** connecting them.



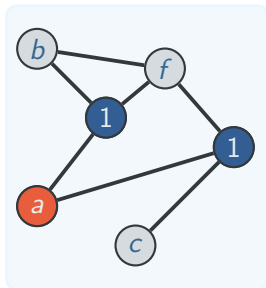
# Distances

The **distance** between two vertices in a graph is the **number** of edges in a **shortest path** connecting them.



# Distances

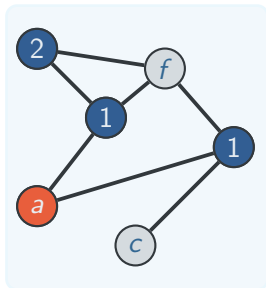
The **distance** between two vertices in a graph is the **number** of edges in a **shortest path** connecting them.





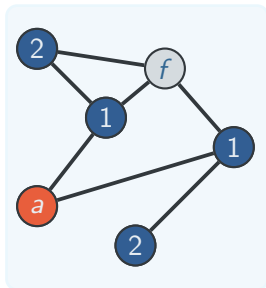
# Distances

The **distance** between two vertices in a graph is the **number** of edges in a **shortest path** connecting them.



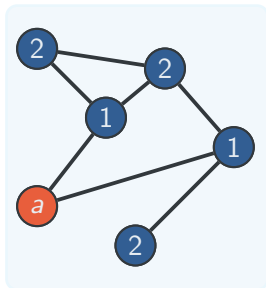
# Distances

The **distance** between two vertices in a graph is the **number** of edges in a **shortest path** connecting them.



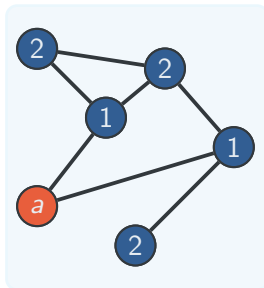
# Distances

The **distance** between two vertices in a graph is the **number** of edges in a **shortest path** connecting them.



# Distances

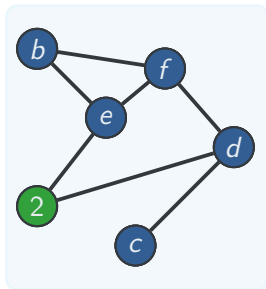
The **distance** between two vertices in a graph is the **number** of edges in a **shortest path** connecting them.



The **eccentricity** of a vertex  $v$  **greatest distance** between  $v$  and any other vertex

# Distances

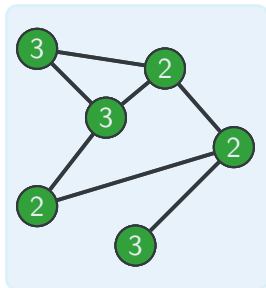
The **distance** between two vertices in a graph is the **number** of edges in a **shortest path** connecting them.



The **eccentricity** of a vertex  $v$  **greatest distance** between  $v$  and any other vertex

# Distances

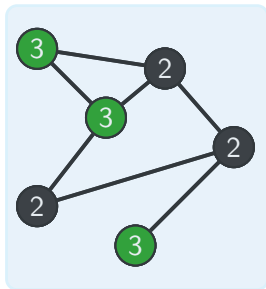
The **distance** between two vertices in a graph is the **number** of edges in a **shortest path** connecting them.



The **eccentricity** of a vertex  $v$  **greatest distance** between  $v$  and any other vertex

# Distances

The **distance** between two vertices in a graph is the **number** of edges in a **shortest path** connecting them.

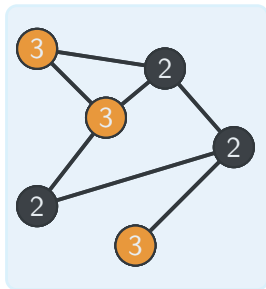


$$r = 2$$

The **radius**  $r$  of a graph is the **minimum eccentricity** of any vertex. The central vertex is one whose the eccentricity is  $r$ .

# Distances

The **distance** between two vertices in a graph is the **number** of edges in a **shortest path** connecting them.



$$d = 3$$

The **diameter**  $d$  of a graph is the **maximum eccentricity** of any vertex. The peripheral vertex is one whose the eccentricity is  $d$ .



# Teoria dos Grafos e Computabilidade

— Special graphs —

Silvio Jamil F. Guimarães

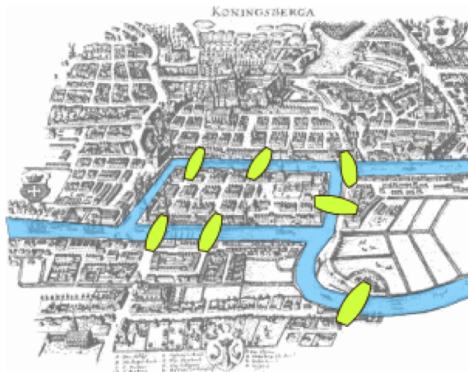
Graduate Program in Informatics – PPGINF

Laboratory of Image and Multimedia Data Science – IMScience

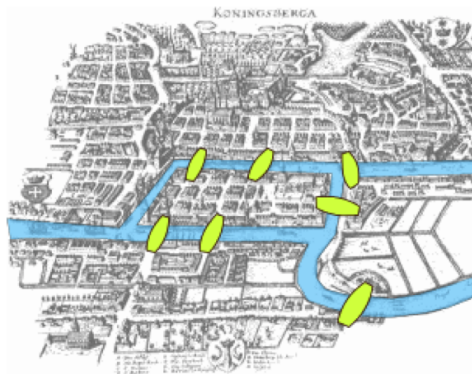
Pontifical Catholic University of Minas Gerais – PUC Minas

# Eulerian graph - bridges of Königsberg

Is there a **path** that crosses each bridge **(exactly) once**?

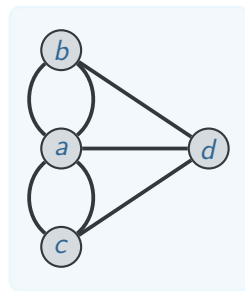
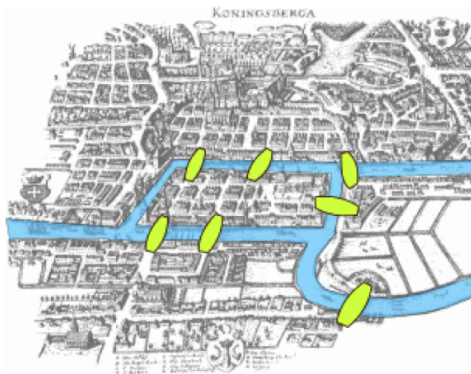


# Eulerian graph - bridges of Königsberg



Is there a **path** that crosses each bridge (exactly) once?

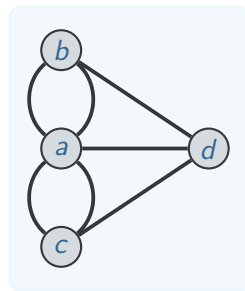
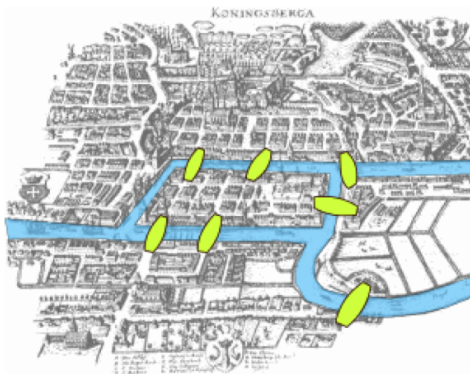
# Eulerian graph - bridges of Königsberg



edges – bridges

Is there a **path** that crosses each bridge (exactly) once ?

# Eulerian graph - bridges of Königsberg



edges – bridges

Is there a **path** that crosses each bridge (exactly) once?

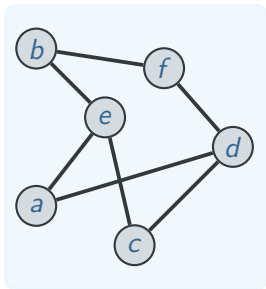
**Eulerian tour**

# Eulerian graph

**Eulerian tour** (or path): a path in a graph that passes through every edge exactly **once**.

# Eulerian graph

**Eulerian tour** (or path): a path in a graph that passes through every edge exactly **once**.

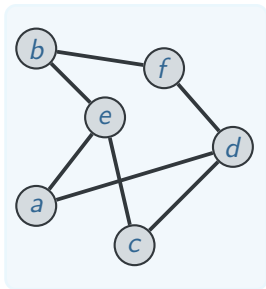


e-a-d-c-e-b-f-d

# Eulerian graph

**Eulerian tour** (or path): a path in a graph that passes through every edge exactly **once**.

**Eulerian cycle** (or circuit): a path in a graph that pass through every edge exactly once and **starts and ends** on the same vertex.



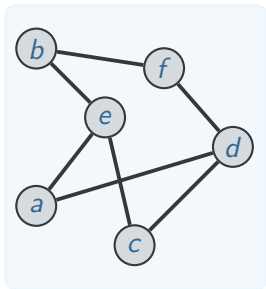
e-a-d-c-e-b-f-d



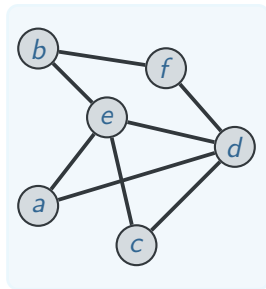
# Eulerian graph

**Eulerian tour** (or path): a path in a graph that passes through every edge exactly **once**.

**Eulerian cycle** (or circuit): a path in a graph that pass through every edge exactly once and **starts and ends** on the same vertex.



e-a-d-c-e-b-f-d



e-a-d-c-e-b-f-d-e

Does the given undirected graph  $G=(V,E)$  contain an Euler tour?

# Algorithmic issues - Eulerian tour

Does the given undirected graph  $G=(V,E)$  contain an Euler tour?

Easy to decide

Does the given undirected graph  $G=(V,E)$  contain an Euler tour?

Easy to decide

Find an Euler tour for the given undirected graph  $G=(V,E)$ , if possible.

# Algorithmic issues - Eulerian tour

Does the given undirected graph  $G=(V,E)$  contain an Euler tour?

Easy to decide

Find an Euler tour for the given undirected graph  $G=(V,E)$ , if possible.

Easy to compute

A **Hamiltonian tour** is a path where each **vertex** occurs exactly once

# Algorithmic issues - Hamiltonian tour

A **Hamiltonian tour** is a path where each **vertex** occurs exactly once

Does the given undirected graph  $G=(V,E)$  **contain** an Hamiltonian tour?

# Algorithmic issues - Hamiltonian tour

A **Hamiltonian tour** is a path where each **vertex** occurs exactly once

Does the given undirected graph  $G=(V,E)$  **contain** an Hamiltonian tour?

Not easy to decide



# Algorithmic issues - Hamiltonian tour

A **Hamiltonian tour** is a path where each **vertex** occurs exactly once

Does the given undirected graph  $G=(V,E)$  **contain** an Hamiltonian tour?

Not easy to decide

**Find** an Hamiltonian tour for the given undirected graph  $G=(V,E)$ , if possible.

# Algorithmic issues - Hamiltonian tour

A **Hamiltonian tour** is a path where each **vertex** occurs exactly once

Does the given undirected graph  $G=(V,E)$  **contain** an Hamiltonian tour?

Not easy to decide

**Find** an Hamiltonian tour for the given undirected graph  $G=(V,E)$ , if possible.

Not easy to compute

# Light Switches – an exercise

A light bulb is connected to 3 switches in such a way that it lights up only when all the switches are in the proper position. But you don't know what the proper position of each switch is!



# Light Switches – an exercise

A light bulb is connected to 3 switches in such a way that it lights up only when all the switches are in the proper position. But you don't know what the proper position of each switch is!

What's the minimum number of **single flips** of a switch to guarantee that the light bulb turns on?

1. 4
2. 8
3. 16
4. 64



# Light Switches – an exercise

A light bulb is connected to 3 switches in such a way that it lights up only when all the switches are in the proper position. But you don't know what the proper position of each switch is!



# Light Switches – an exercise

A light bulb is connected to 3 switches in such a way that it lights up only when all the switches are in the proper position. But you don't know what the proper position of each switch is!

## Exemplo 5

Given a collection of short DNA strings.

**Find** longer DNA string that **includes them all**.

## Exemplo 5

Given a collection of short DNA strings.

Find longer DNA string that includes them all.

Possible solutions

- ▶ as a Hamiltonian tour
- ▶ as an Eulerian tour



## Exemplo 5

Given a collection of short DNA strings.

Find longer DNA string that includes them all.

Possible solutions

- ▶ as a Hamiltonian tour
- ▶ as an Eulerian tour

How to model over a graph the following instance for the DNA reconstruction problem?

$$S = \{ATG, AGG, TGC, TCC, GTC, GGT, GCA, CAG\}$$

# DNA Reconstruction: as a Hamiltonian tour

# DNA Reconstruction: as a Hamiltonian tour

# DNA Reconstruction: as a Hamiltonian tour

# DNA Reconstruction: as an Eulerian tour

# DNA Reconstruction: as an Eulerian tour

# DNA Reconstruction: as an Eulerian tour

# Finding Eulerian tours

Let  $G = (V, E)$  be a graph with  $n$  vertices being  
undirected and connected



# Finding Eulerian tours

Let  $G = (V, E)$  be a graph with  $n$  vertices being  
undirected and connected

Determine whether  $G$  has an Eulerian tour.

# Finding Eulerian tours

Let  $G = (V, E)$  be a graph with  $n$  vertices being undirected and connected

Determine whether  $G$  has an Eulerian tour.

**Theorem** If  $G$  has an Eulerian tour,  $G$  has at most two odd degree vertices.

# Finding Eulerian tours

Let  $G = (V, E)$  be a graph with  $n$  vertices being undirected and connected

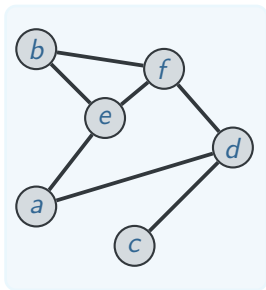
Determine whether  $G$  has an Eulerian tour.

**Theorem** If  $G$  has an Eulerian tour,  $G$  has at most two odd degree vertices.

If yes, find the tour itself

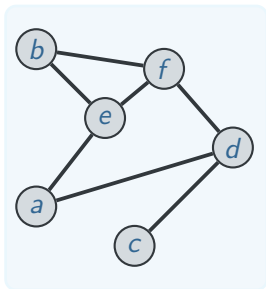
# Finding Eulerian tours – Bridges

Let  $G=(V,E)$  be an undirected and connected graph. A **bridge** is an edge, which, if removed, would cause  $G$  to be **disconnected**.



# Finding Eulerian tours – Bridges

Let  $G=(V,E)$  be an undirected and connected graph. A **bridge** is an edge, which, if removed, would cause  $G$  to be **disconnected**.



Which of the edges in this graph are bridges?

1. All of them.
2. (d,a)
3. (d,c)
4. (e,a) and (f,g)

# Finding Eulerian tours – Bridges

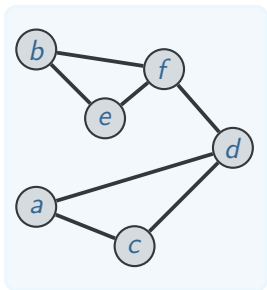
Let  $G=(V,E)$  be an undirected and connected graph. A **bridge** is an edge, which, if removed, would cause  $G$  to be **disconnected**.

In an Eulerian tour, we have to visit **every edge** on one side of the bridge before we **cross** it (because there is no coming back).

# Finding Eulerian tours – Bridges

Let  $G=(V,E)$  be an undirected and connected graph. A **bridge** is an edge, which, if removed, would cause  $G$  to be **disconnected**.

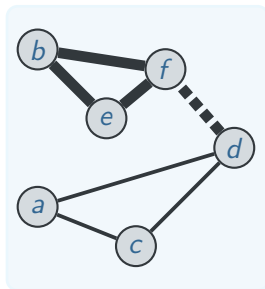
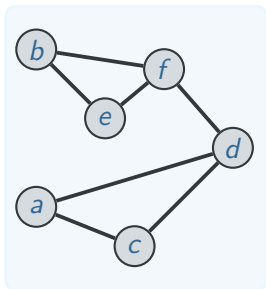
In an Eulerian tour, we have to visit **every edge** on one side of the bridge before we **cross** it (because there is no coming back).



# Finding Eulerian tours – Bridges

Let  $G=(V,E)$  be an undirected and connected graph. A **bridge** is an edge, which, if removed, would cause  $G$  to be **disconnected**.

In an Eulerian tour, we have to visit **every edge** on one side of the bridge before we **cross** it (because there is no coming back).





# Finding Eulerian tours – Fleury's Algorithm

---

**Algorithm:** Eulerian tours – Fleury's Algorithm

---

**input** : An undirected and connected graph  $G = (V, E)$ .

**output:** The Eulerian tour, if so.

- 1 Check that  $G$  has at most 2 odd degree vertices;
  - 2 Start at vertex  $v$ , an odd degree vertex if possible;
  - 3 **while**  $E \neq \emptyset$  **do**
  - 4     **if** *there is more than one edge incident on  $v$*  **then**
  - 5         Cross any edge incident  $e$  on  $v$  that is not a bridge;
  - 6     **else**
  - 7         Cross the only edge  $e$  available from  $v$ ;
  - 8     **end**
  - 9      $E = E - \{e\}$ ;
  - 10 **end**
-

# Finding Eulerian tours – Fleury's Algorithm

---

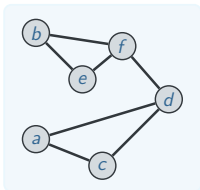
## Algorithm: Eulerian tours – Fleury's Algorithm

---

**input** : An undirected and connected graph  $G = (V, E)$ .

**output**: The Eulerian tour, if so.

- 1 Check that  $G$  has at most 2 odd degree vertices;
  - 2 Start at vertex  $v$ , an odd degree vertex if possible;
  - 3 **while**  $E \neq \emptyset$  **do**
  - 4     **if** *there is more than one edge incident on  $v$*  **then**
  - 5         Cross any edge incident  $e$  on  $v$  that is not a bridge;
  - 6     **else**
  - 7         Cross the only edge  $e$  available from  $v$ ;
  - 8     **end**
  - 9      $E = E - \{e\}$ ;
  - 10 **end**
- 



# Finding Eulerian tours – Fleury's Algorithm

---

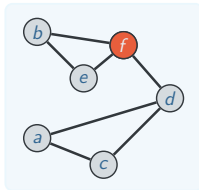
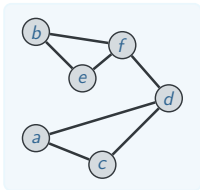
## Algorithm: Eulerian tours – Fleury's Algorithm

---

**input** : An undirected and connected graph  $G = (V, E)$ .

**output**: The Eulerian tour, if so.

- 1 Check that  $G$  has at most 2 odd degree vertices;
  - 2 Start at vertex  $v$ , an odd degree vertex if possible;
  - 3 **while**  $E \neq \emptyset$  **do**
  - 4     **if** *there is more than one edge incident on  $v$*  **then**
  - 5         Cross any edge incident  $e$  on  $v$  that is not a bridge;
  - 6     **else**
  - 7         Cross the only edge  $e$  available from  $v$ ;
  - 8     **end**
  - 9      $E = E - \{e\}$ ;
  - 10 **end**
- 



# Finding Eulerian tours – Fleury's Algorithm

---

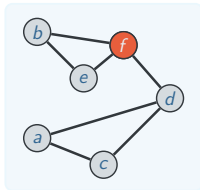
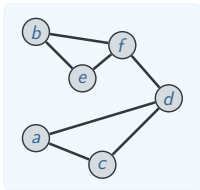
**Algorithm:** Eulerian tours – Fleury's Algorithm

---

**input** : An undirected and connected graph  $G = (V, E)$ .

**output:** The Eulerian tour, if so.

- 1 Check that  $G$  has at most 2 odd degree vertices;
  - 2 Start at vertex  $v$ , an odd degree vertex if possible;
  - 3 **while**  $E \neq \emptyset$  **do**
  - 4     **if** *there is more than one edge incident on  $v$*  **then**
  - 5         Cross any edge incident  $e$  on  $v$  that is not a bridge;
  - 6     **else**
  - 7         Cross the only edge  $e$  available from  $v$ ;
  - 8     **end**
  - 9      $E = E - \{e\}$ ;
  - 10 **end**
- 



# Finding Eulerian tours – Fleury's Algorithm

---

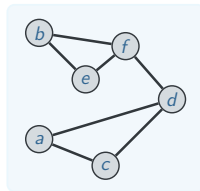
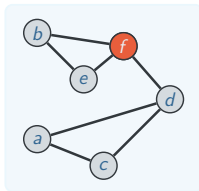
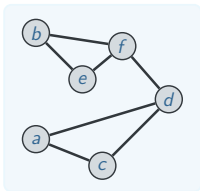
**Algorithm:** Eulerian tours – Fleury's Algorithm

---

**input** : An undirected and connected graph  $G = (V, E)$ .

**output**: The Eulerian tour, if so.

- 1 Check that  $G$  has at most 2 odd degree vertices;
  - 2 Start at vertex  $v$ , an odd degree vertex if possible;
  - 3 **while**  $E \neq \emptyset$  **do**
  - 4     *if there is more than one edge incident on  $v$  then*
  - 5         Cross any edge incident  $e$  on  $v$  that is not a bridge;
  - 6     *else*
  - 7         Cross the only edge  $e$  available from  $v$ ;
  - 8     *end*
  - 9      $E = E - \{e\}$ ;
  - 10 **end**
- 



# Finding Eulerian tours – Fleury's Algorithm

---

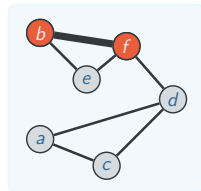
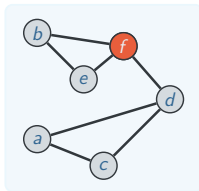
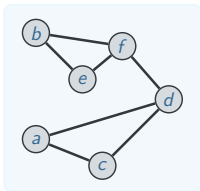
**Algorithm:** Eulerian tours – Fleury's Algorithm

---

**input** : An undirected and connected graph  $G = (V, E)$ .

**output:** The Eulerian tour, if so.

- 1 Check that  $G$  has at most 2 odd degree vertices;
  - 2 Start at vertex  $v$ , an odd degree vertex if possible;
  - 3 **while**  $E \neq \emptyset$  **do**
  - 4     **if** *there is more than one edge incident on  $v$*  **then**
  - 5         Cross any edge incident  $e$  on  $v$  that is not a bridge;
  - 6     **else**
  - 7         Cross the only edge  $e$  available from  $v$ ;
  - 8     **end**
  - 9      $E = E - \{e\}$ ;
  - 10 **end**
- 



# Finding Eulerian tours – Fleury's Algorithm

---

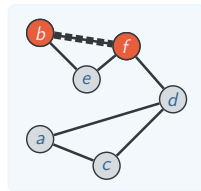
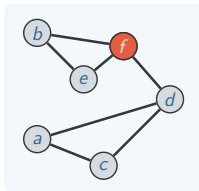
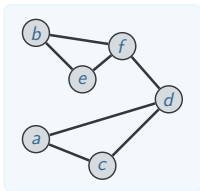
**Algorithm:** Eulerian tours – Fleury's Algorithm

---

**input** : An undirected and connected graph  $G = (V, E)$ .

**output:** The Eulerian tour, if so.

- 1 Check that  $G$  has at most 2 odd degree vertices;
  - 2 Start at vertex  $v$ , an odd degree vertex if possible;
  - 3 **while**  $E \neq \emptyset$  **do**
  - 4     **if** *there is more than one edge incident on  $v$*  **then**
  - 5         Cross any edge incident  $e$  on  $v$  that is not a bridge;
  - 6     **else**
  - 7         Cross the only edge  $e$  available from  $v$ ;
  - 8     **end**
  - 9      $E = E - \{e\}$ ;
  - 10 **end**
- 



# Finding Eulerian tours – Fleury's Algorithm

---

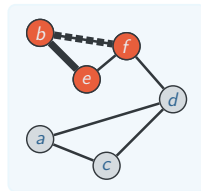
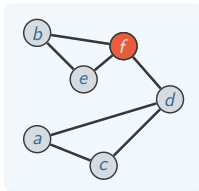
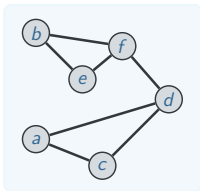
**Algorithm:** Eulerian tours – Fleury's Algorithm

---

**input** : An undirected and connected graph  $G = (V, E)$ .

**output**: The Eulerian tour, if so.

- 1 Check that  $G$  has at most 2 odd degree vertices;
  - 2 Start at vertex  $v$ , an odd degree vertex if possible;
  - 3 **while**  $E \neq \emptyset$  **do**
  - 4     **if** *there is more than one edge incident on  $v$*  **then**
  - 5         Cross any edge incident  $e$  on  $v$  that is not a bridge;
  - 6     **else**
  - 7         Cross the only edge  $e$  available from  $v$ ;
  - 8     **end**
  - 9      $E = E - \{e\}$ ;
  - 10 **end**
- 





# Finding Eulerian tours – Fleury's Algorithm

---

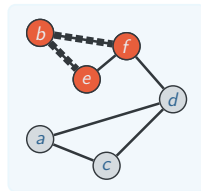
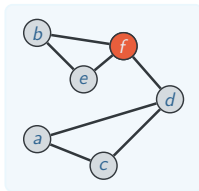
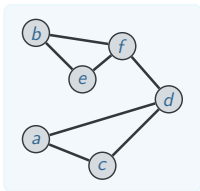
## Algorithm: Eulerian tours – Fleury's Algorithm

---

**input** : An undirected and connected graph  $G = (V, E)$ .

**output**: The Eulerian tour, if so.

- 1 Check that  $G$  has at most 2 odd degree vertices;
  - 2 Start at vertex  $v$ , an odd degree vertex if possible;
  - 3 **while**  $E \neq \emptyset$  **do**
  - 4     **if** *there is more than one edge incident on  $v$*  **then**
  - 5         Cross any edge incident  $e$  on  $v$  that is not a bridge;
  - 6     **else**
  - 7         Cross the only edge  $e$  available from  $v$ ;
  - 8     **end**
  - 9      $E = E - \{e\}$ ;
  - 10 **end**
- 



# Finding Eulerian tours – Fleury's Algorithm

---

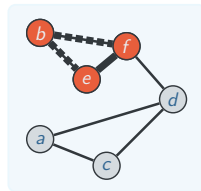
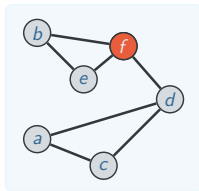
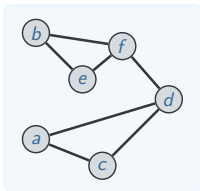
## Algorithm: Eulerian tours – Fleury's Algorithm

---

**input** : An undirected and connected graph  $G = (V, E)$ .

**output**: The Eulerian tour, if so.

- 1 Check that  $G$  has at most 2 odd degree vertices;
  - 2 Start at vertex  $v$ , an odd degree vertex if possible;
  - 3 **while**  $E \neq \emptyset$  **do**
  - 4     **if** *there is more than one edge incident on  $v$*  **then**
  - 5         Cross any edge incident  $e$  on  $v$  that is not a bridge;
  - 6     **else**
  - 7         Cross the only edge  $e$  available from  $v$ ;
  - 8     **end**
  - 9      $E = E - \{e\}$ ;
  - 10 **end**
- 



# Finding Eulerian tours – Fleury's Algorithm

---

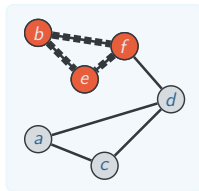
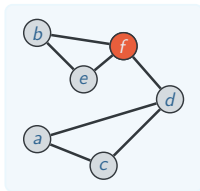
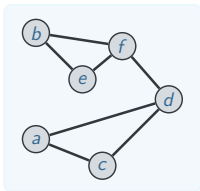
## Algorithm: Eulerian tours – Fleury's Algorithm

---

**input** : An undirected and connected graph  $G = (V, E)$ .

**output**: The Eulerian tour, if so.

- 1 Check that  $G$  has at most 2 odd degree vertices;
  - 2 Start at vertex  $v$ , an odd degree vertex if possible;
  - 3 **while**  $E \neq \emptyset$  **do**
  - 4     **if** *there is more than one edge incident on  $v$*  **then**
  - 5         Cross any edge incident  $e$  on  $v$  that is not a bridge;
  - 6     **else**
  - 7         Cross the only edge  $e$  available from  $v$ ;
  - 8     **end**
  - 9      $E = E - \{e\}$ ;
  - 10 **end**
- 



# Finding Eulerian tours – Fleury's Algorithm

---

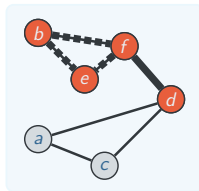
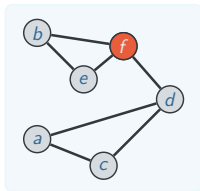
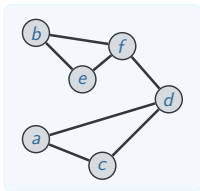
## Algorithm: Eulerian tours – Fleury's Algorithm

---

**input** : An undirected and connected graph  $G = (V, E)$ .

**output**: The Eulerian tour, if so.

- 1 Check that  $G$  has at most 2 odd degree vertices;
  - 2 Start at vertex  $v$ , an odd degree vertex if possible;
  - 3 **while**  $E \neq \emptyset$  **do**
  - 4     **if** *there is more than one edge incident on  $v$*  **then**
  - 5         Cross any edge incident  $e$  on  $v$  that is not a bridge;
  - 6     **else**
  - 7         Cross the only edge  $e$  available from  $v$ ;
  - 8     **end**
  - 9      $E = E - \{e\}$ ;
  - 10 **end**
- 



# Finding Eulerian tours – Fleury's Algorithm

---

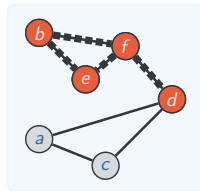
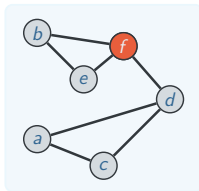
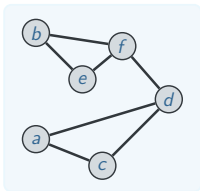
## Algorithm: Eulerian tours – Fleury's Algorithm

---

**input** : An undirected and connected graph  $G = (V, E)$ .

**output**: The Eulerian tour, if so.

- 1 Check that  $G$  has at most 2 odd degree vertices;
  - 2 Start at vertex  $v$ , an odd degree vertex if possible;
  - 3 **while**  $E \neq \emptyset$  **do**
  - 4     **if** *there is more than one edge incident on  $v$*  **then**
  - 5         Cross any edge incident  $e$  on  $v$  that is not a bridge;
  - 6     **else**
  - 7         Cross the only edge  $e$  available from  $v$ ;
  - 8     **end**
  - 9      $E = E - \{e\}$ ;
  - 10 **end**
- 



# Finding Eulerian tours – Fleury's Algorithm

---

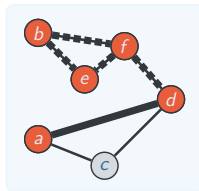
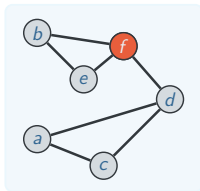
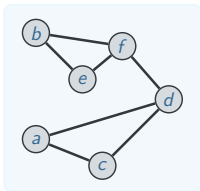
## Algorithm: Eulerian tours – Fleury's Algorithm

---

**input** : An undirected and connected graph  $G = (V, E)$ .

**output**: The Eulerian tour, if so.

- 1 Check that  $G$  has at most 2 odd degree vertices;
  - 2 Start at vertex  $v$ , an odd degree vertex if possible;
  - 3 **while**  $E \neq \emptyset$  **do**
  - 4     **if** *there is more than one edge incident on  $v$*  **then**
  - 5         Cross any edge incident  $e$  on  $v$  that is not a bridge;
  - 6     **else**
  - 7         Cross the only edge  $e$  available from  $v$ ;
  - 8     **end**
  - 9      $E = E - \{e\}$ ;
  - 10 **end**
- 



# Finding Eulerian tours – Fleury's Algorithm

---

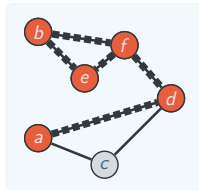
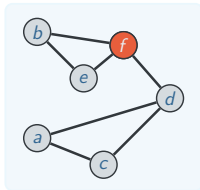
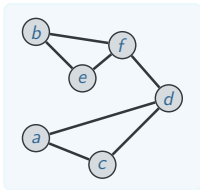
## Algorithm: Eulerian tours – Fleury's Algorithm

---

**input** : An undirected and connected graph  $G = (V, E)$ .

**output**: The Eulerian tour, if so.

- 1 Check that  $G$  has at most 2 odd degree vertices;
  - 2 Start at vertex  $v$ , an odd degree vertex if possible;
  - 3 **while**  $E \neq \emptyset$  **do**
  - 4     **if** *there is more than one edge incident on  $v$*  **then**
  - 5         Cross any edge incident  $e$  on  $v$  that is not a bridge;
  - 6     **else**
  - 7         Cross the only edge  $e$  available from  $v$ ;
  - 8     **end**
  - 9      $E = E - \{e\}$ ;
  - 10 **end**
- 



# Finding Eulerian tours – Fleury's Algorithm

---

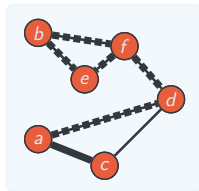
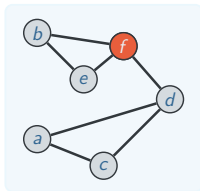
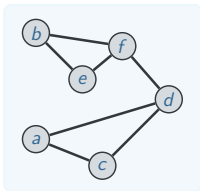
## Algorithm: Eulerian tours – Fleury's Algorithm

---

**input** : An undirected and connected graph  $G = (V, E)$ .

**output**: The Eulerian tour, if so.

- 1 Check that  $G$  has at most 2 odd degree vertices;
  - 2 Start at vertex  $v$ , an odd degree vertex if possible;
  - 3 **while**  $E \neq \emptyset$  **do**
  - 4     **if** *there is more than one edge incident on  $v$*  **then**
  - 5         Cross any edge incident  $e$  on  $v$  that is not a bridge;
  - 6     **else**
  - 7         Cross the only edge  $e$  available from  $v$ ;
  - 8     **end**
  - 9      $E = E - \{e\}$ ;
  - 10 **end**
- 





# Finding Eulerian tours – Fleury's Algorithm

---

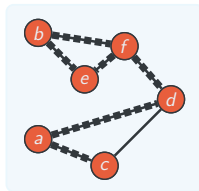
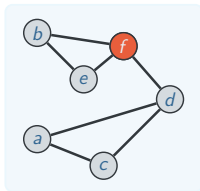
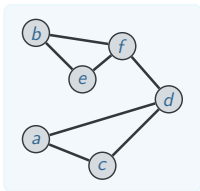
## Algorithm: Eulerian tours – Fleury's Algorithm

---

**input** : An undirected and connected graph  $G = (V, E)$ .

**output**: The Eulerian tour, if so.

- 1 Check that  $G$  has at most 2 odd degree vertices;
  - 2 Start at vertex  $v$ , an odd degree vertex if possible;
  - 3 **while**  $E \neq \emptyset$  **do**
  - 4     **if** *there is more than one edge incident on  $v$*  **then**
  - 5         Cross any edge incident  $e$  on  $v$  that is not a bridge;
  - 6     **else**
  - 7         Cross the only edge  $e$  available from  $v$ ;
  - 8     **end**
  - 9      $E = E - \{e\}$ ;
  - 10 **end**
- 



# Finding Eulerian tours – Fleury's Algorithm

---

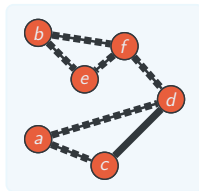
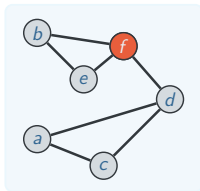
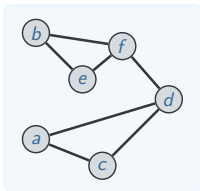
## Algorithm: Eulerian tours – Fleury's Algorithm

---

**input** : An undirected and connected graph  $G = (V, E)$ .

**output**: The Eulerian tour, if so.

- 1 Check that  $G$  has at most 2 odd degree vertices;
  - 2 Start at vertex  $v$ , an odd degree vertex if possible;
  - 3 **while**  $E \neq \emptyset$  **do**
  - 4     **if** *there is more than one edge incident on  $v$*  **then**
  - 5         Cross any edge incident  $e$  on  $v$  that is not a bridge;
  - 6     **else**
  - 7         Cross the only edge  $e$  available from  $v$ ;
  - 8     **end**
  - 9      $E = E - \{e\}$ ;
  - 10 **end**
- 



# Finding Eulerian tours – Fleury's Algorithm

---

**Algorithm:** Eulerian tours – Fleury's Algorithm

---

**input** : An undirected and connected graph  $G = (V, E)$ .

**output:** The Eulerian tour, if so.

- 1 Check that  $G$  has at most 2 odd degree vertices;
  - 2 Start at vertex  $v$ , an odd degree vertex if possible;
  - 3 **while**  $E \neq \emptyset$  **do**
  - 4     **if** *there is more than one edge incident on  $v$*  **then**
  - 5         Cross any edge incident  $e$  on  $v$  that is not a bridge;
  - 6     **else**
  - 7         Cross the only edge  $e$  available from  $v$ ;
  - 8     **end**
  - 9      $E = E - \{e\}$ ;
  - 10 **end**
- 

