SIN110 Algoritmos e Grafos

Aula 10

Técnicas de Projeto de algoritmos

- Backtracking (E06)

Grafos

- *conceito, modelagem, aplicações*
- representação computacional
- pesquisa em grafos
- Busca em Profundidade

SIN110 Algoritmos e Grafos

Backtraking

E06 - solução

1) A equipe de um programa de TV quer realizar um evento de grande impacto junto ao público: *reunir o maior número possível de casais em uma mesma cerimônia nupcial*. De início, podemos imaginar uma solução trivial como inscrever todos os casais da cidade, ou região, que estão com seus papeis em andamento nos cartórios. Porém é aí que começa o desafio: os futuros pares deverão ser formados a partir de um encontro promovido pelo programa entre desconhecidos! Para promover os encontros, a produção da TV, lançou uma campanha publicitária "convocando" os solitários a participar do programa através de uma inscrição, via Internet, onde os candidatos e candidatas preenchem um conjunto de informações sobre suas características e preferências. Encerrado o período de inscrições, foram confeccionadas duas tabelas de dimensão $n \times n$, uma para os homens H e, outra para as mulheres M, correspondendo a n homens e n mulheres. Em cada tabela foram quantificadas as preferências de tal forma que o elemento H[i,j] correspondem ao nível de preferência do homem *i* em relação à mulher *j* e de forma análoga o elemento M[i,j] quantifica a preferência da mulher *i* em relação ao homem *j*. Entrando na reta final do projeto, a TV precisa maximizar o número de encontros que resultem em casamento e para isso vai adotar a estratégia de formar pares utilizando as tabelas M e H de forma a *maximizar a soma dos produtos obtidos com as preferências dos casais formados.* Assim poderíamos ter uma solução do tipo:

$$S = H[1,37]*M[37,1] + H[2,k]*M[k,2] + ... + H[n,78]*M[78,n] \text{ e } S \text{ é máxima.}$$

Esboce um algoritmo para efetuar essa tarefa.

Esboce um algoritmo para efetuar essa tarefa.

Backtraking:

Entrada: matrizes M e H, de dimensão n x n, com níveis de preferência

Saída: n pares cuja soma dos produtos de interesses é máxima

Processamento:

1) Criar a matriz P = M X H, com todos os produtos de preferências.

2) Criar uma pilha de pares (h,m) para armazenar os casais, e um vetor que indica alocação de casais, A[i] = j indica homem i e mulher j.

3) Pesquisar os maiores valores na matriz P e, empilhar o elemento (h, m, Val).

4) Para os conflitos, i ou j, já alocados, deve-se procurar os maiores valores restantes para i e j, verificando novas parcerias disponíveis, para tentar alocar, avaliando o valor do par já alocado em comparação de novas alocações com i e j, e escolhendo aquele que tiver maior soma. Por exemplo, se temos o par (1,37) com valor 45, e uma nova alocação com (13,37) de valor 33 entra em conflito, verificamos os maiores valores envolvendo (13, j) e (i, 37) para avaliar quem terá a maior soma: (1,37) + (13, j≠37) ou (1, j≠37) + (13,37).

5) Ao final indica qual a maior soma encontrada.

2) *O problema das quatro rainhas*: oito rainhas devem ser colocadas em um tabuleiro de xadrez, de tal modo que nenhuma delas ataque, ou seja atacada por nenhuma outra. Implemente um algoritmo para resolver este problema considerando como dado de entrada uma casa C_{ij} do tabuleiro, que pode ser interpretado como uma matriz **4×4**, e a partir daí o algoritmo deverá solucionar o problema.

> *Entrada: matriz M (tabuleiro 4×4) e posição inicial (i,j) - casa Cij do tabuleiro.*
> *Saída: posições encontradas indicando a linha de cada uma das colunas (1 a 4), por exemplo: 3 1 2 4.*

```
Alg4Rainhas(C_ij)
 1. Crie Pilha                                  {armazena rainha no tabuleiro}
 2. Crie Marca[1..4]                            {vetor de alocações}
 3. Crie Tab[1..4, 1..4]                        {tabuleiro com alocações}
 4. Pilha( )← (i,j)                             {aloca primeira rainha}
 5. para k ←1 até 4 faça  Marca[k] ←0           {controle de alocações}
 6. para l ←1 até 4                             {ocupação do tabuleiro}
 7.       para k ←1 até 4 faça  Tab[l,k] ←0
 8. Marca[j] ←i, N ←1, TAb[i,j] ←N              {anota posição inicial}
 9. lin ←i+1, col ←j+1,                         {auxiliares de posição}
10. enquanto N < 4 faça                         {preenchimento do tabuleiro}
11.     (l,k) ←Localiza(Tab[ ], lin, col)
12.    se (l,k) = (-1,-1)
13.      então (l,k) ←Retira-Topo-Pilha()       {backtraking}
14.            Marca[k] ←0, N ←N-1              {desmarca decisão anterior}
15.           (lin, col) ←(l, k+1)              {recomeça na coluna seguinte}
16.     senão  Marca[k] ←l, Anota(lin,col,N)
17.            lin ←l+1, col ←k+1, N ←N+1
18.           se lin > 4 então lin ←1
19.           se col > 4 então col ←1
20. devolve Marca[1..n]                         {saída com rainhas alocadas}
```

```
Localiza(Tab[ ], lin, col)
1. se col > 1 então aux ← col, Lim ← 5
2.              senão aux = 0
3. (l,k) ← Verifica(Tab[ ], lin, col, Lim]
4. se k = Lim e E aux ≠ 0 então col ← 1, Lim ← aux
5.                                  (l,k) ← Verifica(Tab[ ], lin, col, Lim]
6. se k = Lim então devolve (-1, -1)
7.              senão devolve (l, k)
```

```
Verifica(Tab[ ], lin, col, Lim)
1. enquanto col <  Lim  faça
 2.    para k ←1 até 4 faça
 3.       se Tab[lin,k] ≠ 0   então lin ←  lin + 1, k ←4
 4.    para k ←1 até 4 faça
 5.          se Tab[k,col] ≠ 0  então col ←col + 1
 6.    para l ←lin até 1 faça
 7.          para k ←col até col-1in+1  faça            {diag principal acima}
 8.                se Tab[l,k] ≠ 0   então col ←col + 1
 9.          para k ←col+1 até 4 faça          {diag secundária acima}
10                 se Tab[l,k] ≠ 0   então col ←col + 1
11.   para l ←4 até lin+1 faça
12.         para k ←4 até col+1  faça          {diag princ abaixo}
13.               se Tab[l,k] ≠ 0  então col ←col + 1
14 .       para k ←col-lin+1 até col+1 faça            {diag sec abaixo}
15.               se Tab[l,k] ≠ 0    então col ←col + 1
```

SIN110 Algoritmos e Grafos

GRAFOS

- *Modelagem*
- *Representação computacional*

Grafos

Abstração que permite codificar relacionamentos entre pares de objetos

Que objetos?

Qualquer um! Ex. pessoas, cidades, empresas, países, páginas web, filmes, etc...

Que relacionamentos?

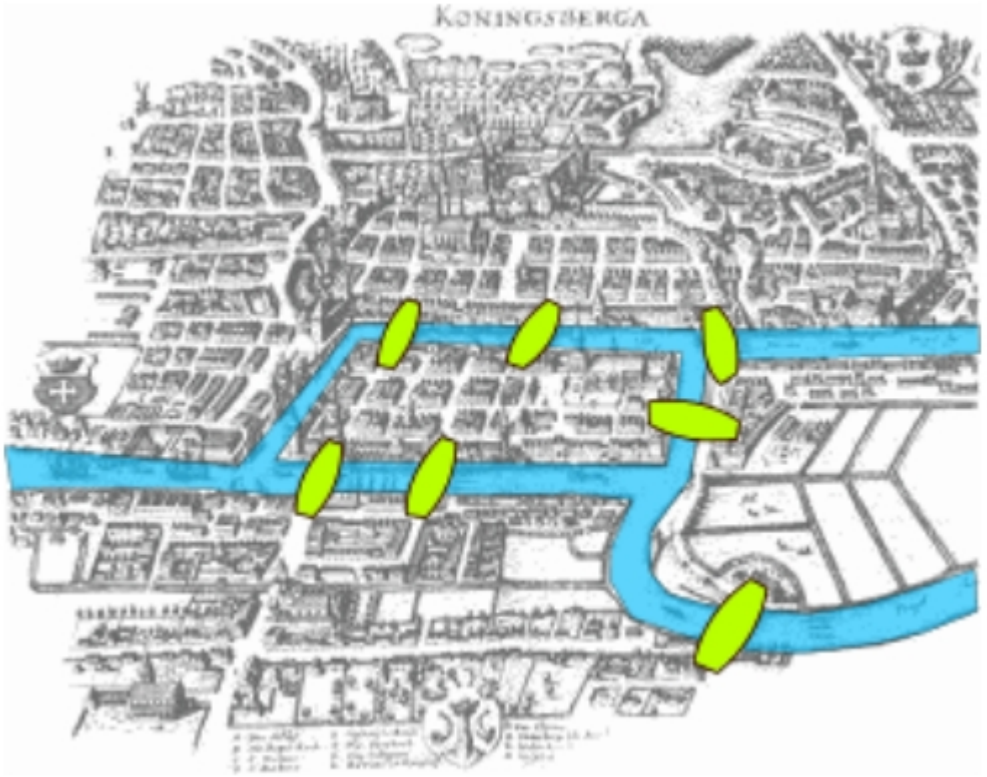
Qualquer um! Ex. amizade, conectividade, produção, língua falada, etc.

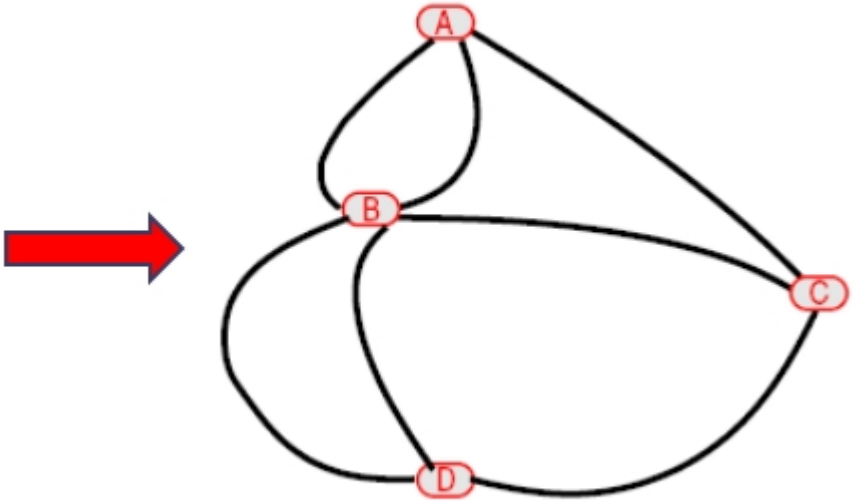
Abstração que permite codificar relacionamentos entre pares de objetos

Objetos → vértices do grafo

Relacionamentos → arestas do grafo

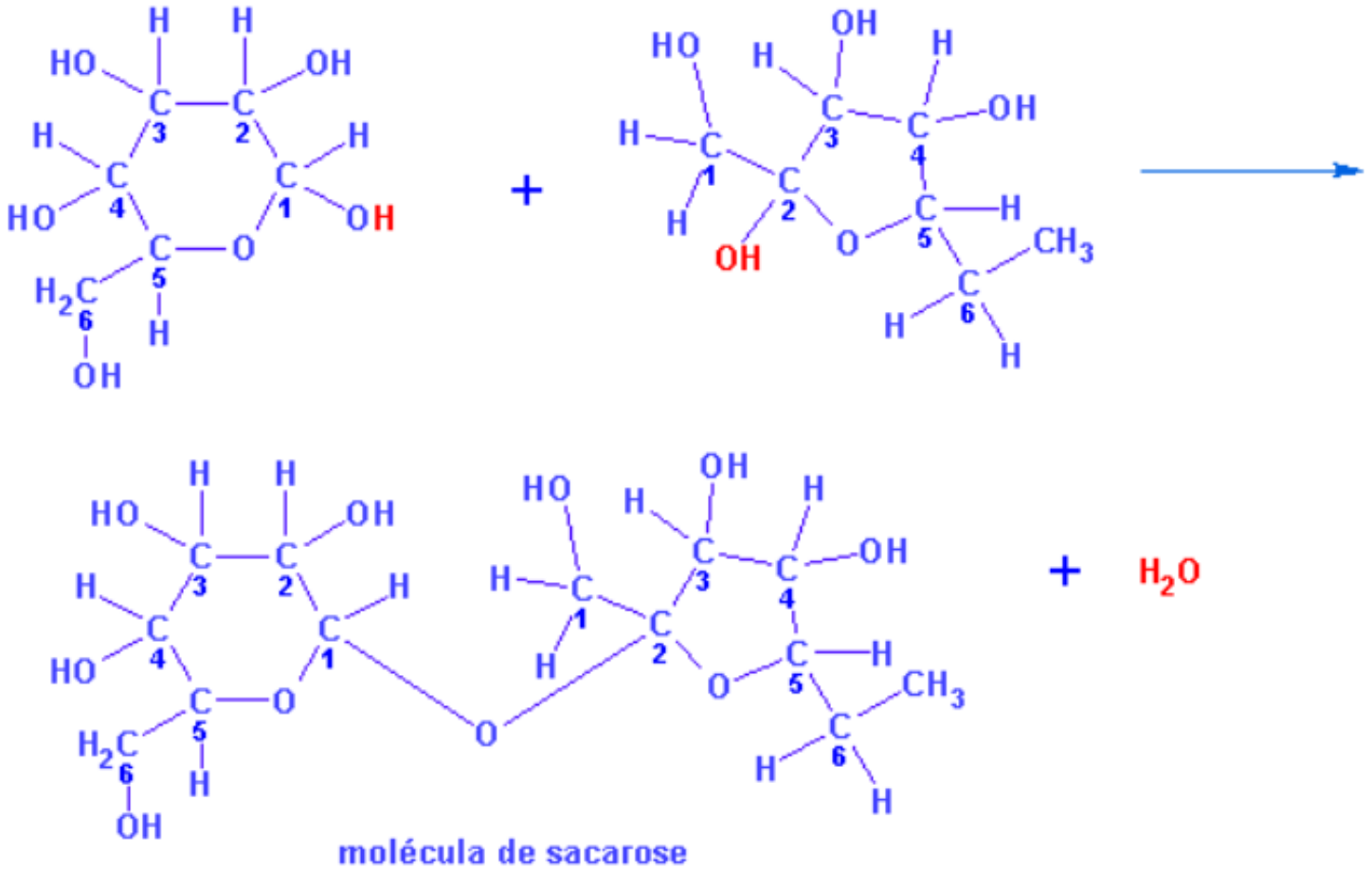
As Sete Pontes de Königsberg





Química molecular

- Representação bidimensional de moléculas utilizando grafos...



Redes de computadores

- Redes de computadores utilizam tabelas de encaminhamento para o roteamento de pacotes...

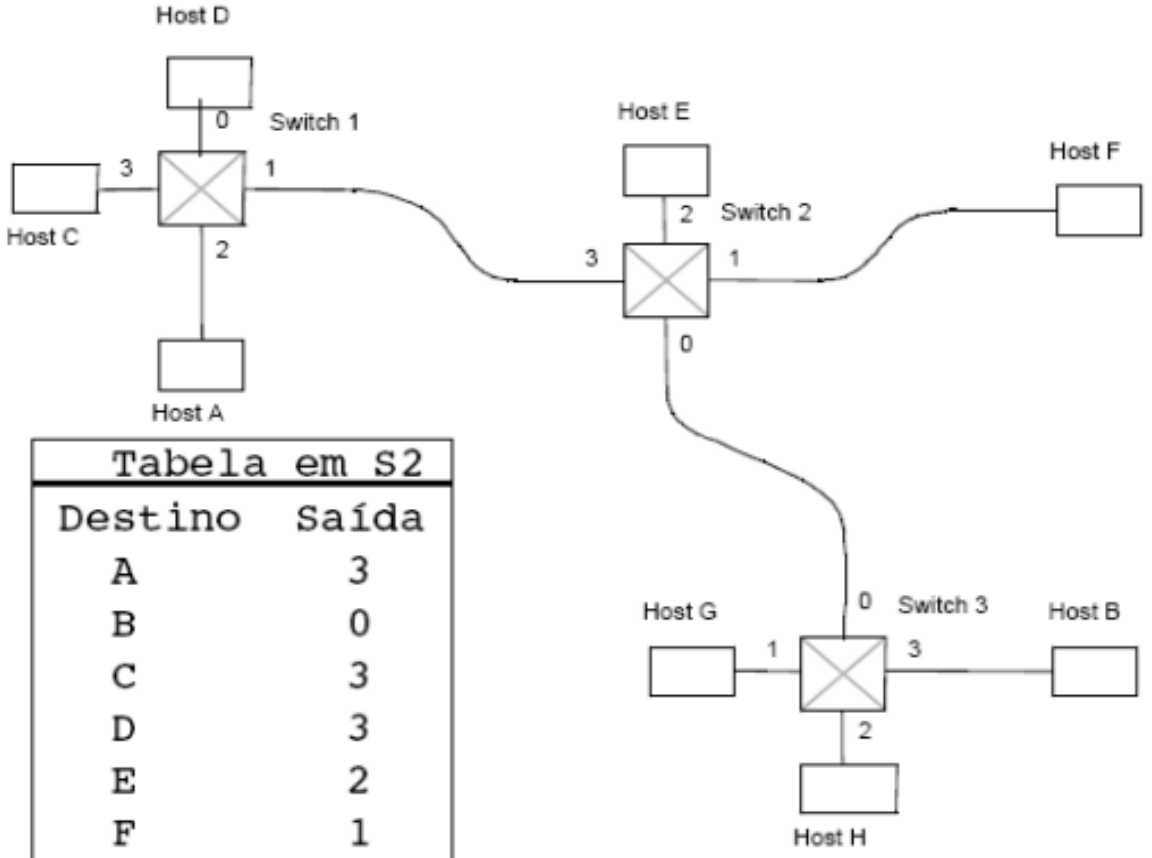


Tabela em S2	
Destino	Saída
A	3
B	0
C	3
D	3
E	2
F	1
G	0
H	0

conceitos e definições

GRAFO

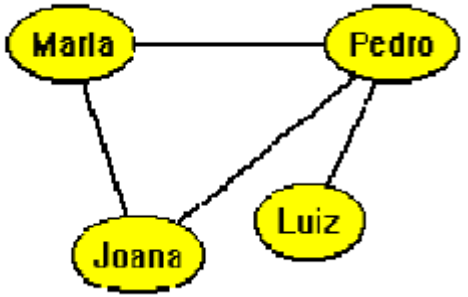
Um grafo *G(V,E)* é definido pelo par de conjuntos *V* e *E*, onde:

V - conjunto não vazio: os **vértices** ou **nodos** do grafo;
E - conjunto de pares ordenados *e* = (v,w), v e w $\in$ V: as **arestas** do grafo.

Seja, por exemplo, o grafo G(V,E) dado por:

V = { p | p é uma pessoa } e E = { (v,w) | < v é amigo de w > }

G_1:



DIGRAFO (Grafo Orientado)

Considere, agora, o grafo definido por:

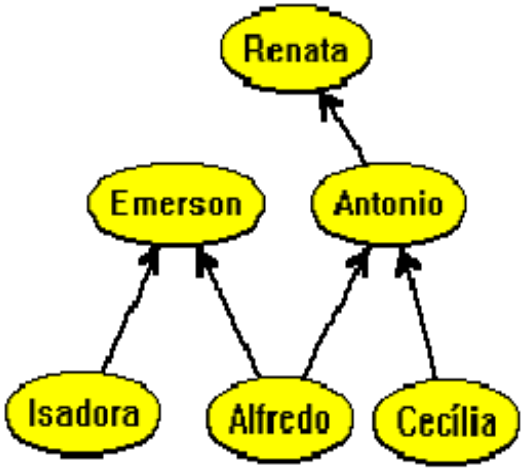
$V = \{ p \mid p \text{ é uma pessoa da família Castro} \}$ e $E = \{ (v,w) \mid < v \text{ é pai/mãe de } w > \}$

Um exemplo de deste grafo (ver G_2) é:

V = { Emerson, Isadora, Renata, Antonio, Rosane, Cecília, Alfredo }

E = {(Isadora, Emerson), (Antonio, Renata), (Alfredo, Emerson), (Cecília, Antonio), (Alfredo, Antonio)}

G_2:



A relação definida por *E* **não é simétrica**, pois se

<v é pai/mãe de w>, não é o caso de *<w é pai/mãe de v>*.

Há, portanto, uma orientação na relação, com um correspondente efeito na representação gráfica de G.

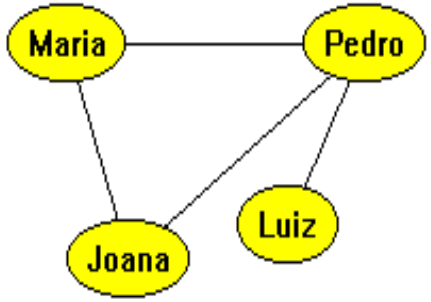
O grafo acima é dito ser um **grafo orientado** (ou **dígrafo**), sendo que as conexões entre os vértices são chamadas de **arcos**.

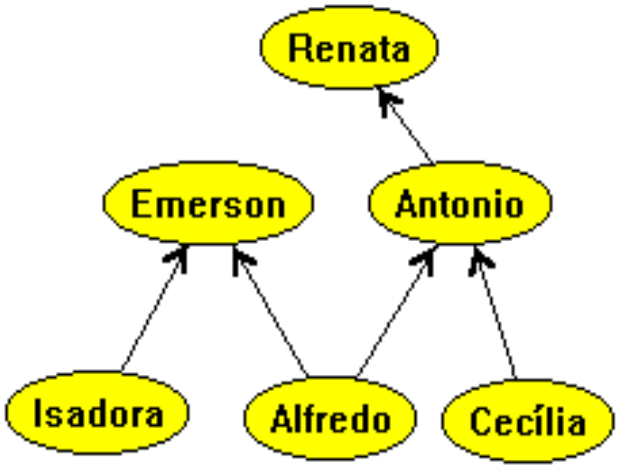
ADJACÊNCIA

Em um grafo simples (a exemplo de G_1) dois vértices *v* e *w* são adjacentes (ou vizinhos) se há uma aresta $a=(v,w)$ em G. Está aresta é dita ser incidente a ambos, *v* e *w*. É o caso dos vértices *Maria* e *Pedro* em G_1. No caso do grafo ser dirigido (a exemplo de G_2), a adjacência (vizinhança) é especializada em:

Sucessor: um vértice *w* é sucessor de *v* se há um arco que parte de *v* e chega em *w*. Em G_2, por exemplo, diz-se que *Emerson* e *Antonio* são sucessores de *Alfredo*.

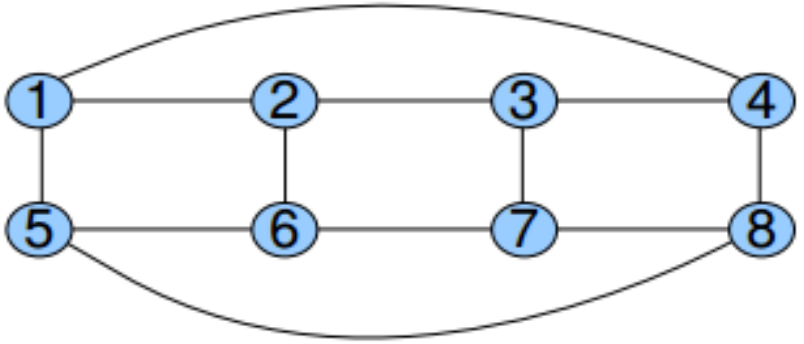
Antecessor: um vértice *v* é antecessor de *w* se há um arco que parte de *v* e chega em *w*. Em G_2, por exemplo, diz-se que *Alfredo* e *Cecília* são antecessores de *Antonio*.





Caminho Simples

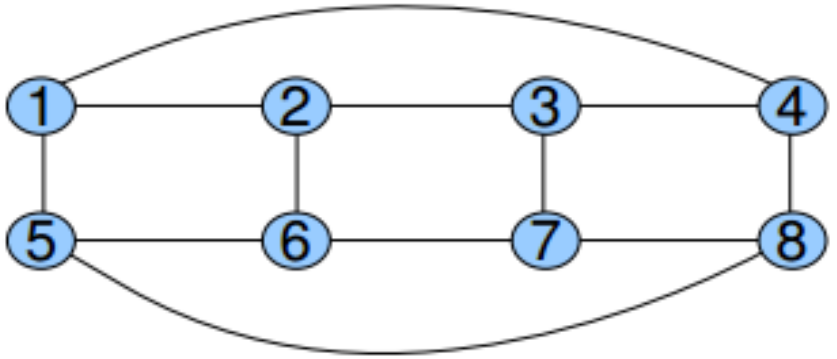
- Vértices do caminho são distintos
  - não há “voltas”



- Ex. caminho entre 1 e 7?
  - 1, 5, 6, 2, 6, 7 ← não é caminho simples
  - 1, 5, 8, 7 ← é caminho simples
- Comprimento do caminho
  - ńumero de arestas que o forma
- Dado G = (V, E)
  - qual é o menor caminho simples entre dois vértices?
  - qual é o maior?

Ciclo

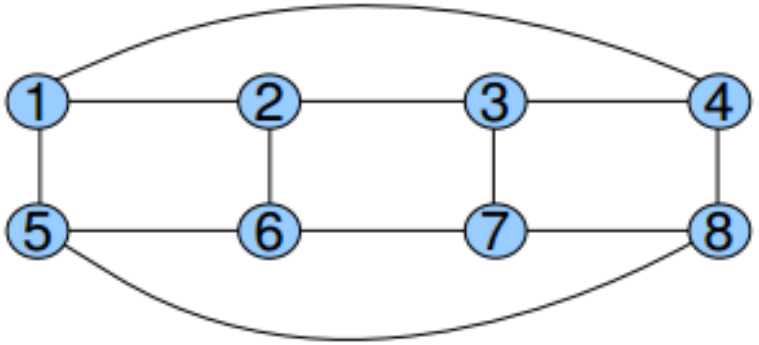
- Caminho simples que começa e termina no mesmo vértice
  - $v_1 = v_k$
- Ex. ciclo em 5?
  - 5, 1, 2, 3, 7, 6, 5
  - 5, 1, 4, 8, 5



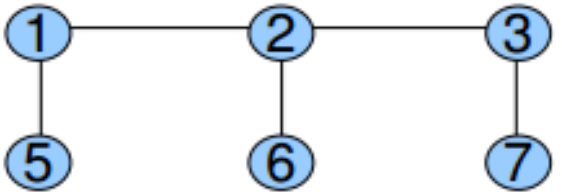
- Comprimento do ciclo
  - ńumero de arestas que o forma
- Dado G = (V, E)
  - qual é o maior ciclo?
  - n vértices, ciclo hamiltoniano

Subgrafo

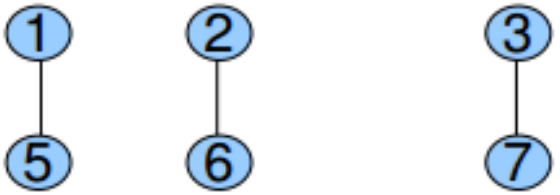
- Dado G = (V, E)



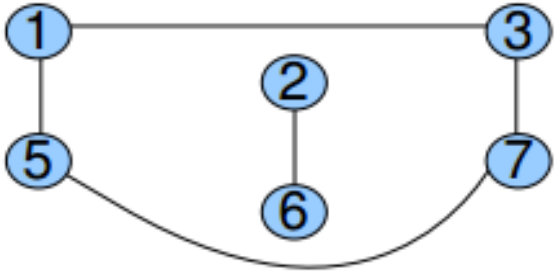
É subgrafo de G?



É subgrafo de G?

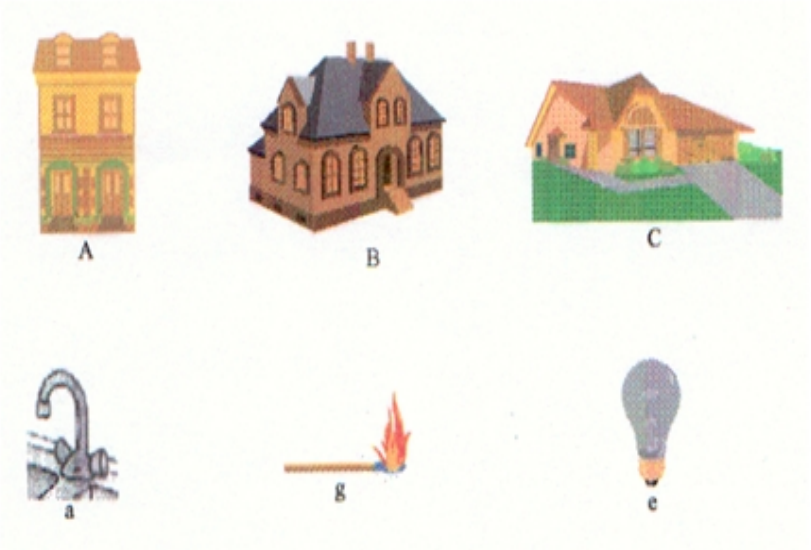


É subgrafo de G?



Exemplo de aplicação

Pretendemos ligar três casas A, B e C, a três utilitários, gás (g) , água (a) e electricidade (e). Por razões de segurança convém que as ligações não se cruzem. Quantas ligações terão de ser feitas?



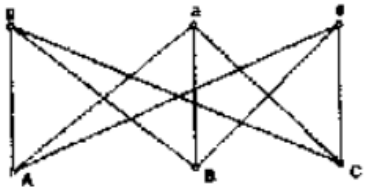
Exemplo de aplicação

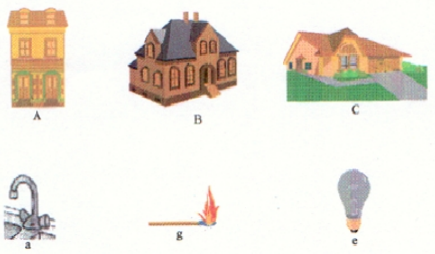
Resolução:

Representação em grafo:

Vértices: A, B, C, g, a ,e

Arestas: (A, g); (A, a); (A, e); (B, g); (B, e); (B, a); (C, g); (C, a); (C, e)

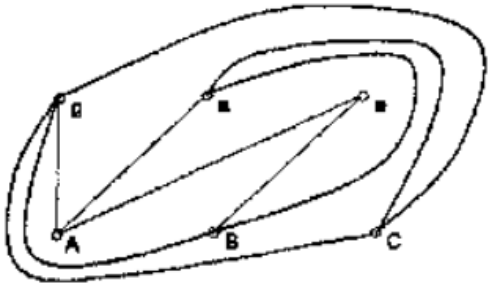




Trata-se de um grafo regular: todos os vértices têm grau três

É bipartido, basta considerar a partição $V_1=\{A, B, C\}$ $V_2=\{g, a, e\}$

É planar porque ,embora tenha sido desenhado com as arestas a intersectarem-se pode



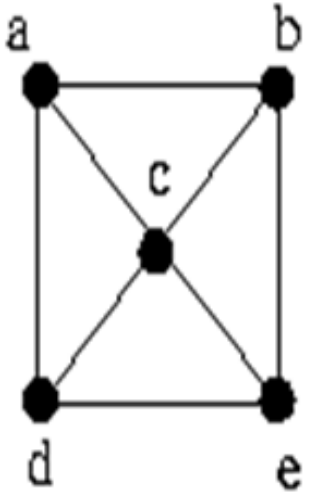
ser desenhado sem que tal aconteça.

Conclusão: Não é possível efetuar todas as ligações sem cruzar as linhas

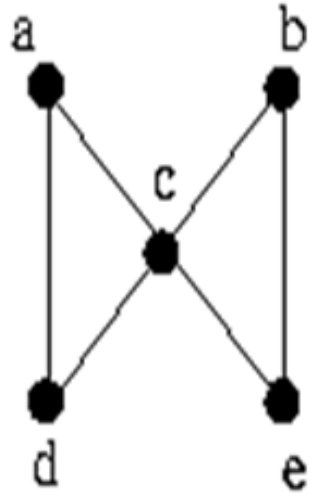
Ciclo hamiltoniano

... é um caminho hamiltoniano que começa e termina no mesmo vértice.

O grafo da figura (a) é hamiltoniano porque contém pelo menos um circuito de Hamilton (por exemplo: a, b, e, c, d, a), enquanto que o da figura (b) não é hamiltoniano porque para termos um ciclo temos que passar mais que uma vez pelo vértice c.



(a)



(b)

Exemplo de aplicação

Problema do Cavalo no jogo de xadrez

Seguindo as regras de movimento do cavalo, é possível que um cavalo parta de uma casa qualquer, percorra todo o tabuleiro visitando cada casa uma e somente uma única vez e retorne à casa inicial?

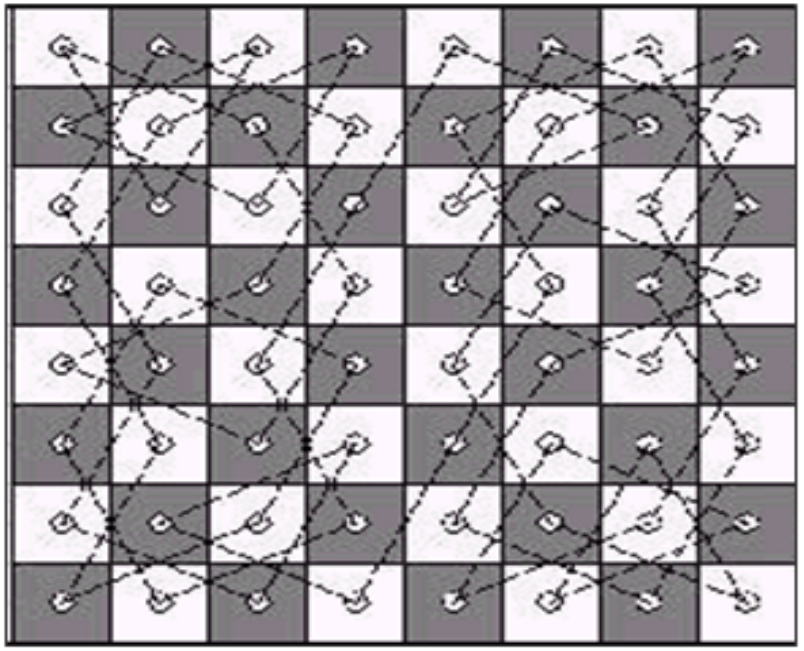
Um modelo para este problema é definir o grafo **G(V,A)** como:

V = { c| c é uma casa do tabuleiro de xadrez}

A = { (c_1,c_2) | a casa c_2 pode ser atingida a partir da casa c_1 por um único movimento de cavalo}

A solução deste problema passa por verificar se o grafo **G** é hamiltoniano.

Este grafo tem 64 vértices e 168 arestas, e, na realidade, contém vários ciclos hamiltonianos, um os quais:

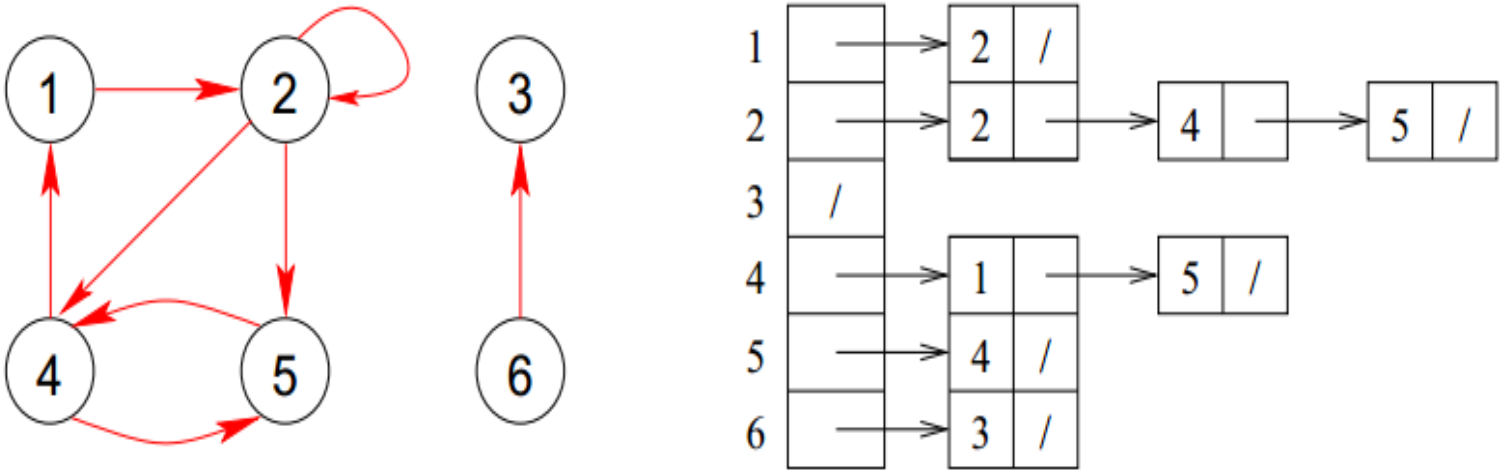


representação computacional

Listas de adjacências

Como representar um grafo orientado?

A representação mais usual é como um conjunto de listas de adjacências. Trata-se de uma representação compacta, útil para grafos em que o número de arcos $|E|$ seja pequeno (muito menor que $|V|^2$) – grafos *esparsos*.



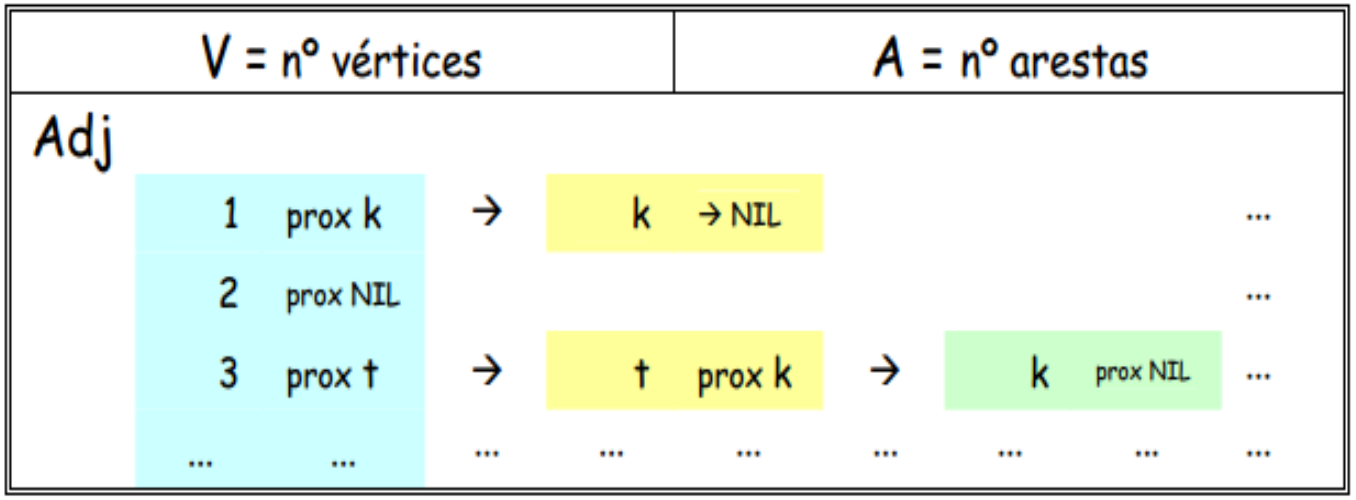
Consiste num vector Adj de $|V|$ listas ligadas. A lista $Adj[i]$ contém todos os vértices j tais que $(i, j) \in E$, i.e., todos os vértices adjacentes a i.

Representação computacional

A estrutura de dados deve informar:

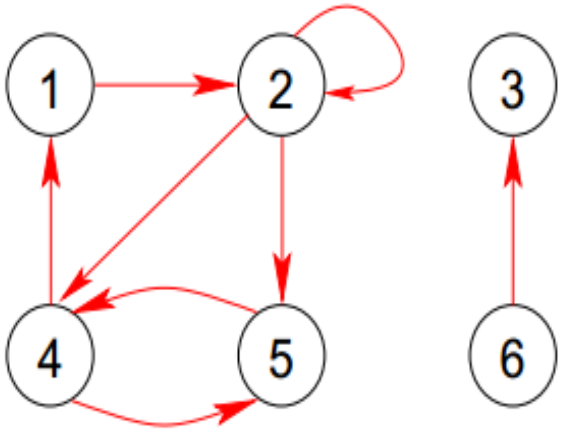
- O número de vértices;
- A quantidade de arestas (ou arcos);
- As listas de adjacências
- Valores e/ou atributos associados a cada aresta (arco)

Ilustrando a estrutura teríamos o Tipo Abstrato de Dado (TAD) grafos (ou dígrafos):



Matriz de adjacências

Uma outra possibilidade é a representação por uma matriz de adjacências. Trata-se de uma representação estática, e por isso apropriada para grafos *densos* – em que $|E|$ se aproxima de $|V|^2$.



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

Trata-se de uma matriz de dimensão $|V| \times |V|$, $A = (a_{ij})$ com $a_{ij} = 1$ se $(i, j) \in E$; $a_{ij} = 0$ em caso contrário.

Representação computacional

A estrutura de dados deve informar:

- O número de vértices;
- A quantidade de arestas (ou arcos);
- A matriz de adjacências
- Valores e/ou atributos associados a cada aresta (arco)

Ilustrando a estrutura teríamos o Tipo Abstrato de Dado (TAD) grafos (ou dígrafos):

V = nº vértices A = nº arestas

Adj

(1,1)	(1,2)	(1,3)	...
(2,1)	(2,2)	(2,3)	...
(3,1)	(3,2)	(3,3)	...
...	...	...	...

modelagem

Modelo usando grafos
Vegetarianos e Canibais (1)

- Seja uma região formada por vegetarianos e canibais.
- Inicialmente, dois vegetarianos e dois canibais estão na margem esquerda (ME) de um rio.
- Existe um barco que pode transportar no máximo duas pessoas e sempre atravessa o rio com pelo menos uma pessoa.
- O objetivo é achar uma forma de transportar os dois vegetarianos e os dois canibais para a margem direita (MD) do rio.
- Em nenhum momento, o número de canibais numa margem do rio pode ser maior que o número de vegetarianos, caso contrário, . . .

Modelo usando grafos
Vegetarianos e Canibais (2)

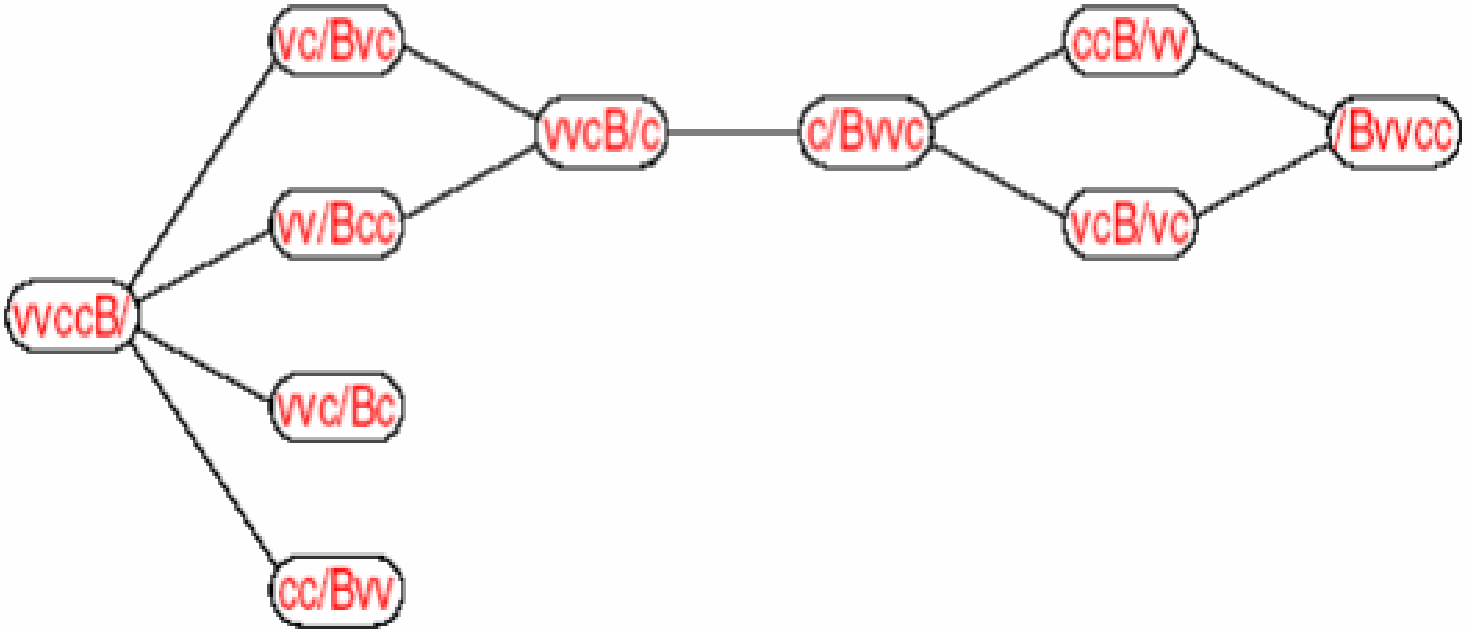
- Solução:
  - Notação para representar cada cenário possível.
  - Modelo para representar a mudança de um cenário em outro válido.

- Notação: ME/MD
  - vvccB/ $\rightarrow$ ME: 2v, 2c e o barco (B); MD: –.
  - vc/Bvc $\rightarrow$ ME: 1v, 1c; MD: B, 1v e 1c.

- Modelo: grafo
  - Vértice: cenário válido.
  - Aresta: transição válida de um dado cenário em outro.

Modelo usando grafos
Vegetarianos e Canibais (3)

Uma possível seqüência válida de cenários é:



Exercícios: modelagem em grafos

1. Três missionários e três canibais devem atravessar um rio. Para isso dispõe de uma canoa que pode transportar no máximo duas pessoas de cada vez. Durante a travessia, se o número de canibais for maior que o de missionários em qualquer das margens os canibais devora os missionários. Determinar um plano de travessia para que nenhum missionário seja devorado empregando um grafo.

2. Suponha um cruzamento de trânsito em "T", como ilustra a figura com os sentidos de trânsito demonstrados. Um veículo vindo de B pode ir em frente ou virar para C, outro veículo vindo de C só pode virar para B, e assim por diante. O cruzamento deve ser controlado por semáforos que exibem as cores verde e vermelho, não há o amarelo. Suponha que há sensores que determinam o sentido dos veículos. Por exemplo, há um sensor que determina se existe veículo se dirigindo de B para C, há um sensor que determina se há veículos se deslocando de A para B, etc. Modele o funcionamento dos semáforos através de um grafo.

Algoritmos de pesquisa

busca em grafos

Busca em Grafos

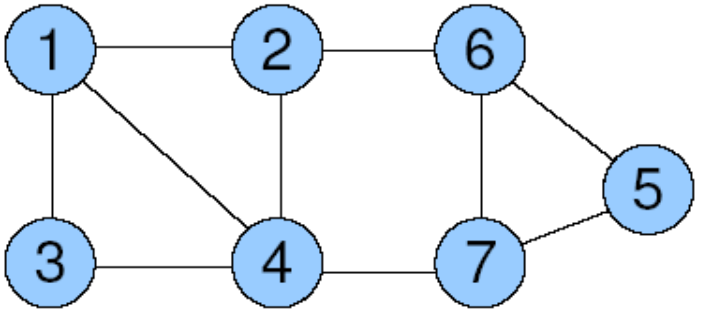
- Problema fundamental em grafos

Como explorar um grafo de forma sistemática?

- Muitas aplicações são abstraídas como problemas de busca
- Muitos algoritmos utilizam fundamentos similares

Busca em Grafos

- Como explorar o grafo abaixo?
- Como saber se dois vértices estão “conectados”?
  - de maneira eficiente



- **Idéia**: evitar explorar vértices já explorados
  - marcar os vértices!
  - vértice: *descoberto* ou *explorado*

Busca em Grafos

- Definir vértice inicial (origem ou raiz)
- Explorar e marcar vértices
  - *Descoberto*: vértice foi descoberto (visitado pela primeira vez)
  - *Explorado*: todas as arestas incidentes ao vértice foram exploradas
- Algoritmo genérico?

Algoritmo Genérico

- Passo inicial
  - desmarcar todos os vértices
  - selecionar origem e marcá-lo *descoberto*
- Passo geral (enquanto houver vértice descoberto)
  - Selecionar vértice descoberto, u
  - Considerar aresta não explorada, (u, v)
  - Se v não estiver marcado, marcar v como *descoberto*
  - Marcar u *explorado* quando não houver mais arestas incidentes a u a serem exploradas

Ordenação da Exploração

- Ordem de visita dos vértices e arestas?

Algoritmo genérico não estabelece ordem!

- Qual é uma possível ordem?
  - Sistemática de exploração
- Duas abordagens
  - Explorar o vértice *descoberto* "mais antigo"
  - Explorar o vértice *descoberto* "mais recente"

Busca em profundidade

Depth - First Search

Busca em Profundidade (DFS)

- Explorar vértices descobertos mais recentes primeiro
- **Interpretação**
  - procurar uma saída de um labirinto
  - vai fundo atrás da saída (tomando decisões a cada encruzilhada)
  - volta a última encruzilhada quando encontrar um beco sem saída (ou lugar já visitado)

Pesquisa primeiro na profundidade

A busca em profundidade – do inglês depth-first search: DFS

Na busca em profundidade, as arestas são exploradas a partir do vértice visitado mais recentemente.

Sempre que um vértice v é descoberto durante a busca na lista de adjacências de um outro vértice já visitado u, busca em profundidade (DFS) memoriza este evento ao definir o predecessor de v - pred(v) - como u.

O vetor de predecessores forma uma árvore, o vetor predecessor da busca em profundidade pode ser composto de várias árvores.

O algoritmo DFS recebe um grafo cujos vértices são coloridos durante a busca:

- BRANCO: antes da busca;
- CINZA: quando o vértice for visitado e,
- PRETO: quando os vértices adjacentes já foram visitados.

DFS emprega uma variável global "tempo", que marca as visitas em cada vértice em dois instantes:

- d(v) $\leftarrow$ tempo, no instante em que v foi visitado (pintado de CINZA) ,
- f(v) $\leftarrow$ tempo, no instante em que a busca pelos vértices na lista de adjacências de v foi completada sendo então pintado de PRETO.

Algoritmo:

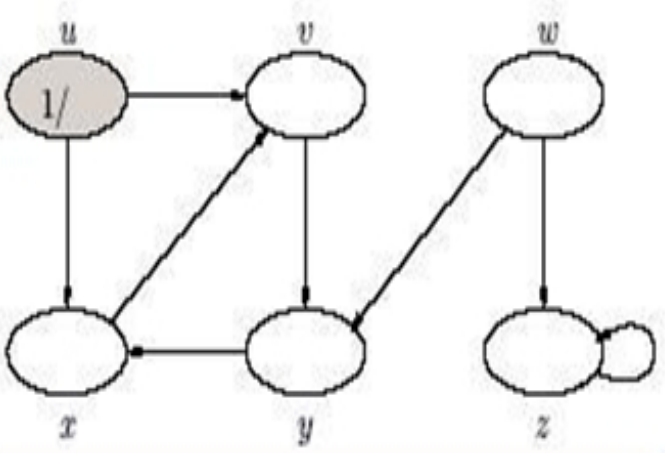
DFS(G)

1. para cada vértice u em G faça
2. cor[u] = BRANCO
3. pred(u) $\leftarrow$ NIL
4. tempo $\leftarrow$ 0
5. para cada vértice u em G faça
6. se cor(u) = branco
7. então Visita_DFS(u)

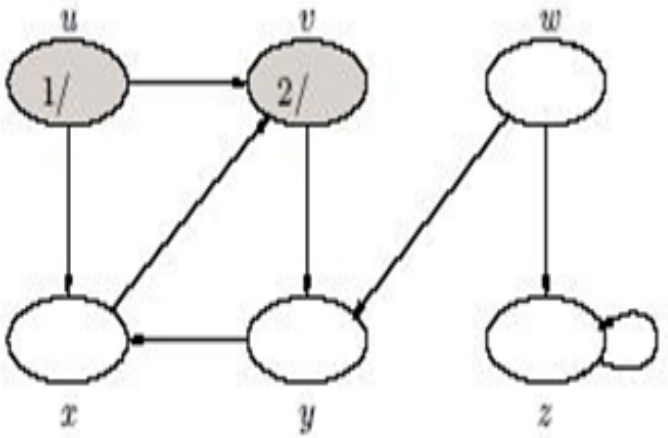
Visita_DFS(u)

```
1.  cor[u] ← CINZA
2.  tempo ← tempo + 1
3.  d(u) ← tempo
4.  para cada v em Adj(u) faça
5.          se cor[v] = BRANCO
6.                  então pred(v) ← u
7.                          Visita_DFS(v)
8.  cor[u] ← PRETO
9.  tempo ← tempo + 1
10. f(u) ← tempo
```

Exemplo:

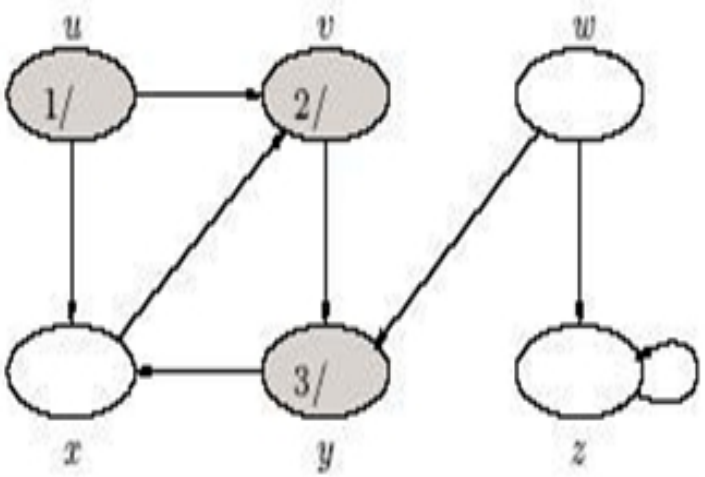


(i)

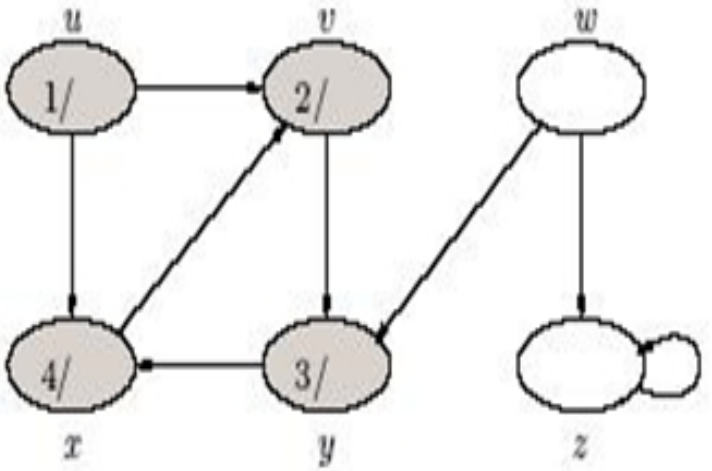


(ii)

Exemplo:

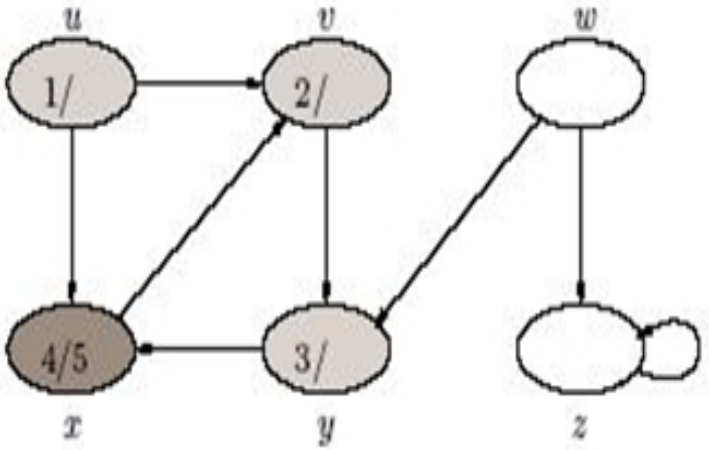


(iii)

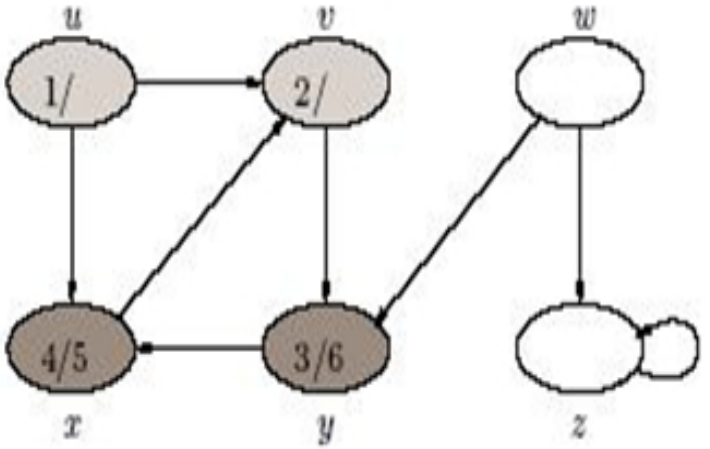


(iv)

Exemplo:

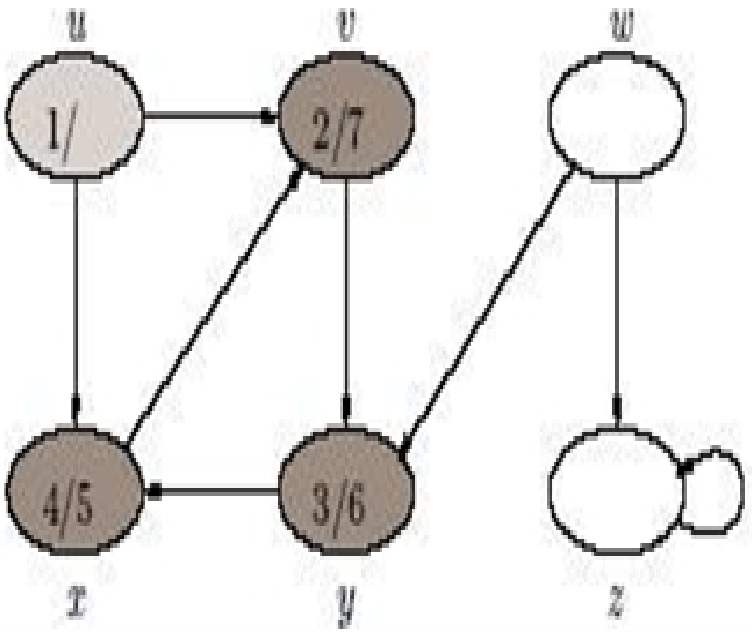


(v)

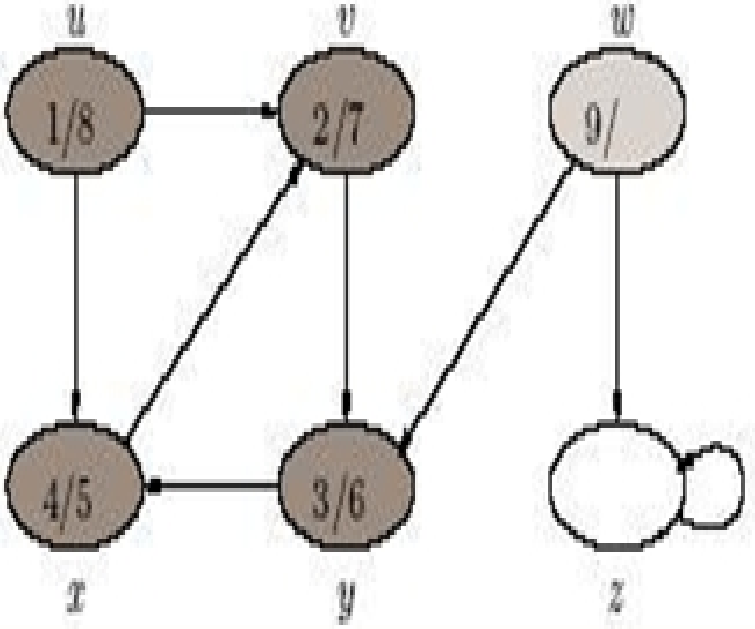


(vi)

Exemplo:

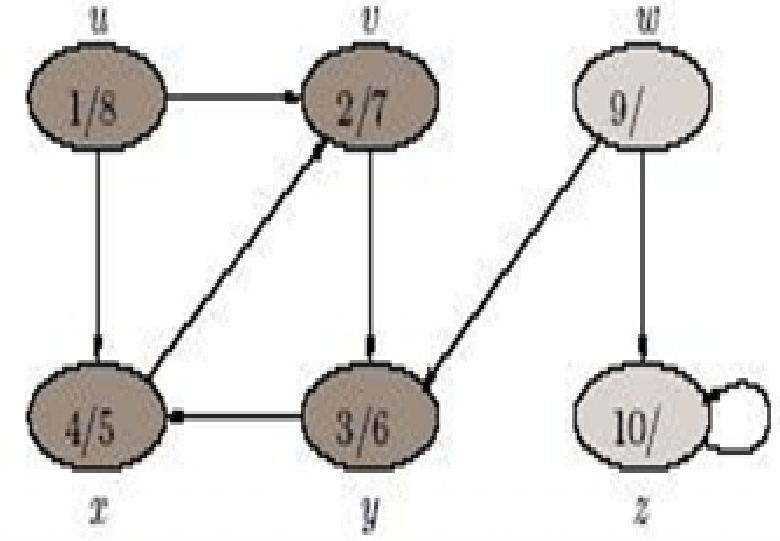


(vii)

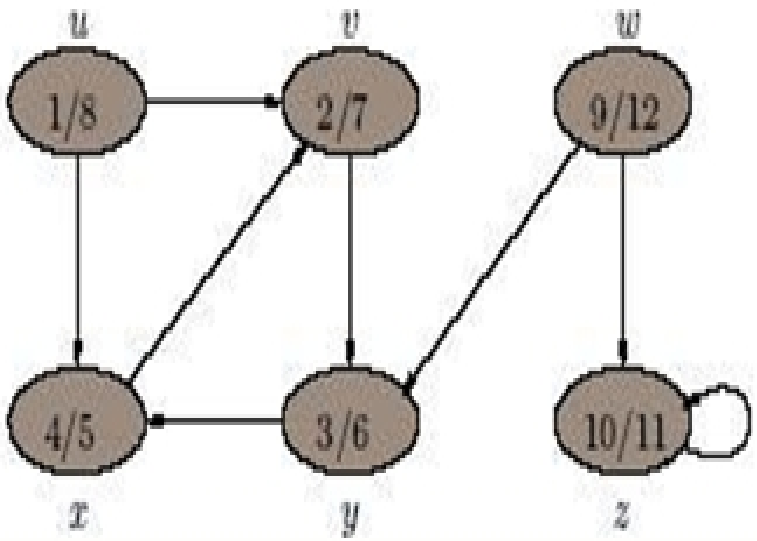


(viii)

Exemplo:



(ix)



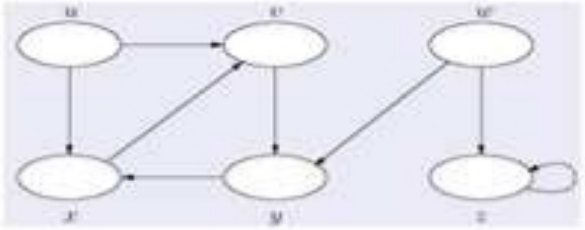
(x)

DFS(G)

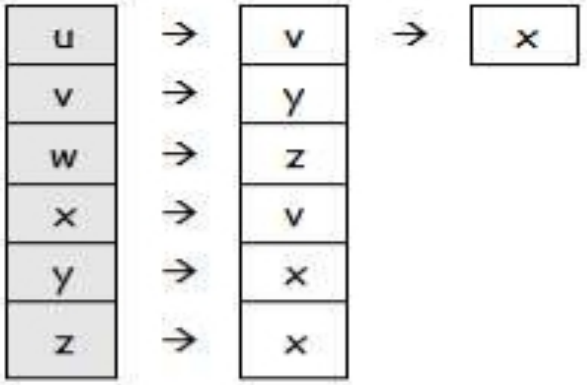
```
1.  para cada vértice u em G faça
2.        cor[u] = BRANCO
3.        pred[u] ← NIL
4.  tempo ← 0
5.  para cada vértice u em G faça
6.           se cor(u) = BRANCO
7.                 então Visita_DFS(u)
8.  devolve pred[1..n]
```

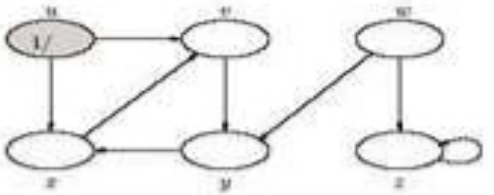
Visita_DFS(u)

```
1.   cor[u] ← CINZA
2.  tempo ← tempo + 1
3.  d(u)  ← tempo
4.  para cada v em Adj(u) faça
5.           se cor[v] = BRANCO
6.                    então pred[v] ← u
7.                          Visita_DFS(v)
7.  cor[u] ← PRETO
8.  tempo ← tempo + 1
9.  f(u)  ← tempo
```

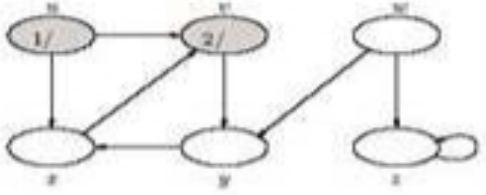


Dígrafo e suas listas adj:

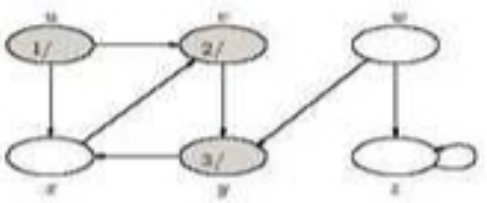




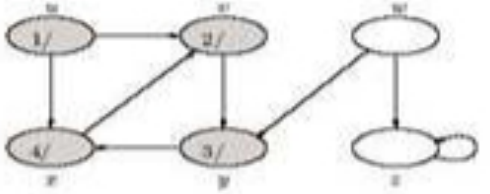
(i)



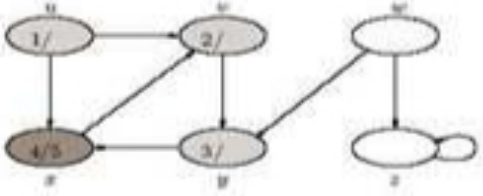
(ii)



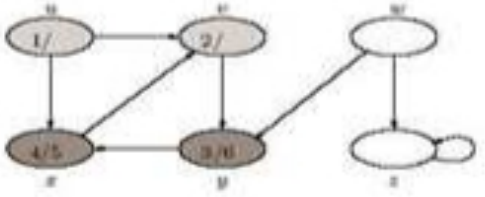
(iii)



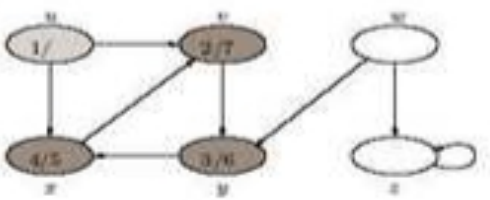
(iv)



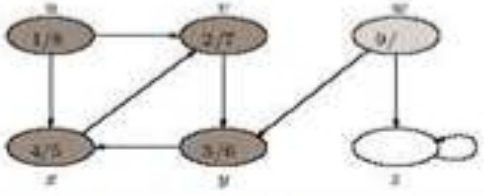
(v)



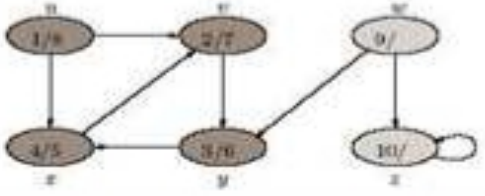
(vi)



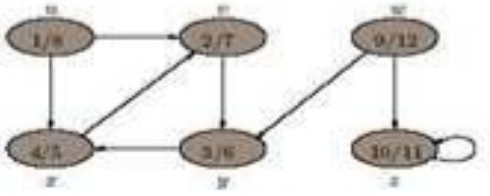
(vii)



(viii)

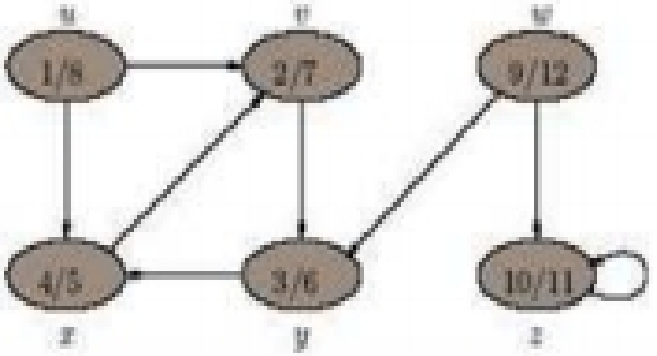


(ix)

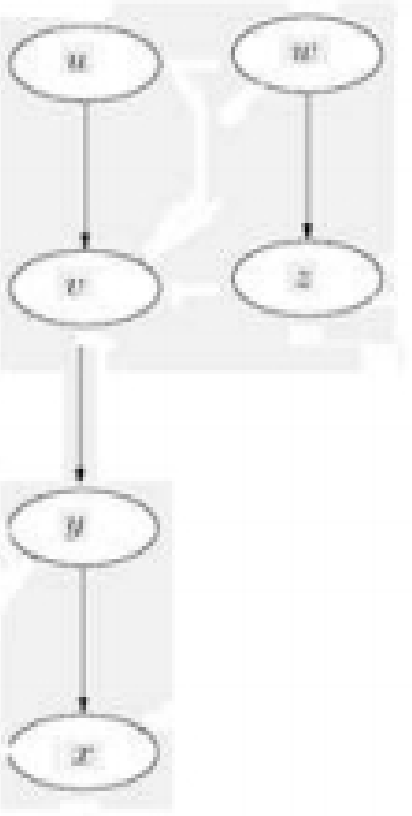


(x)

DFS



Arborescências:



pred:

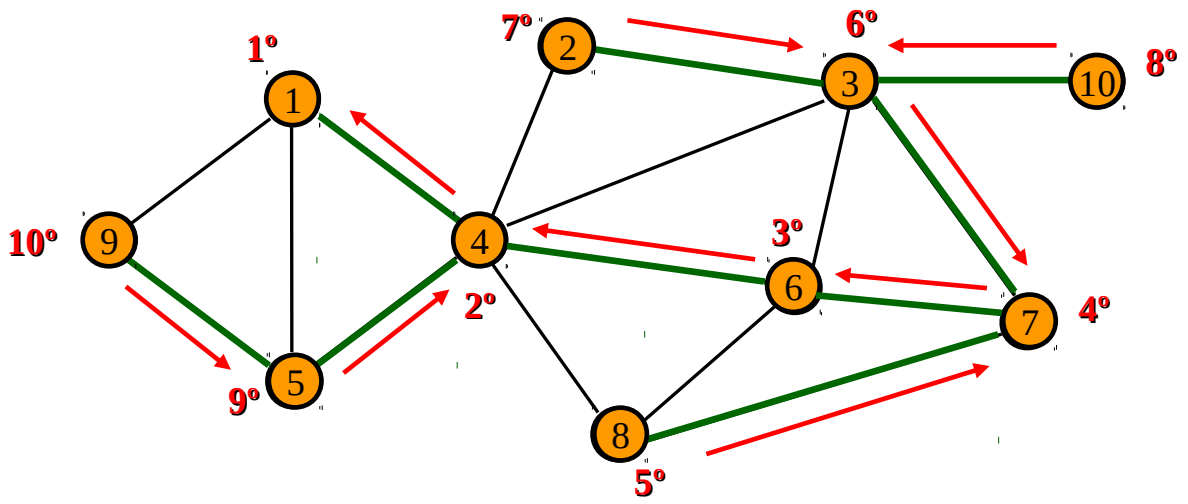
[u]	[v]	[w]	[x]	[y]	[z]
-	u	-	y	v	w

Busca em profundidade

- A ordem em que os nós e arestas são visitados depende:
  - Do vértice inicial
  - Da ordem em que os vértices e arestas aparecem na lista de adjacências

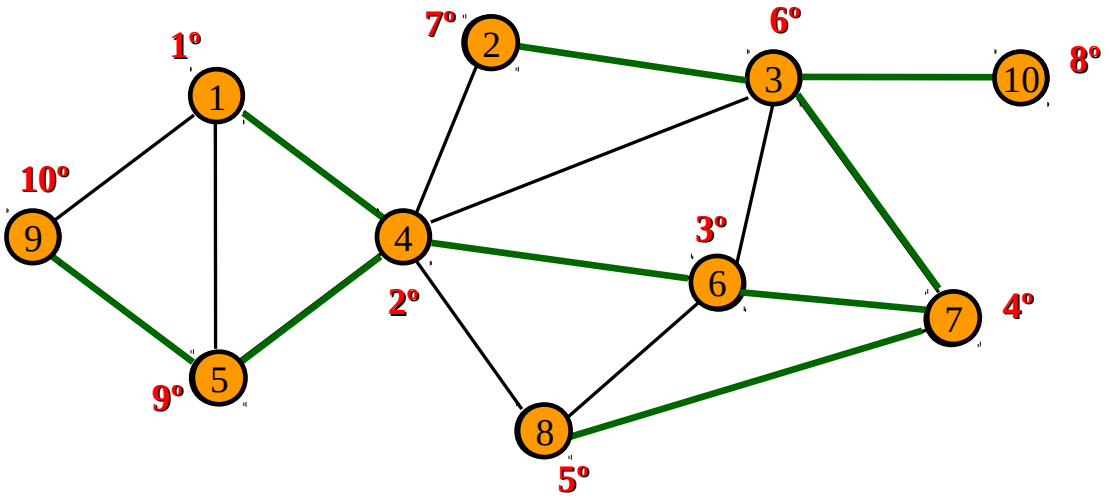
Busca em profundidade

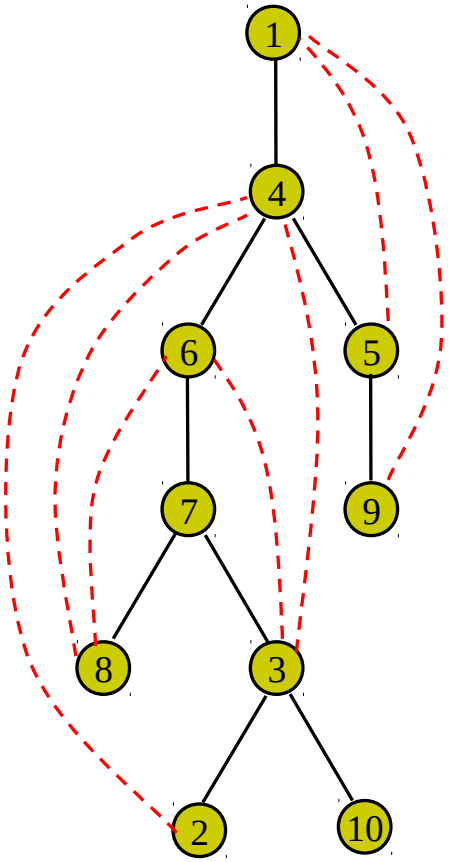
- Funcionamento básico
  - Enquanto for possível, aprofundar-se no grafo. Quando não for mais possível, recuar até que seja possível continuar a aprofundar-se.



Busca em profundidade

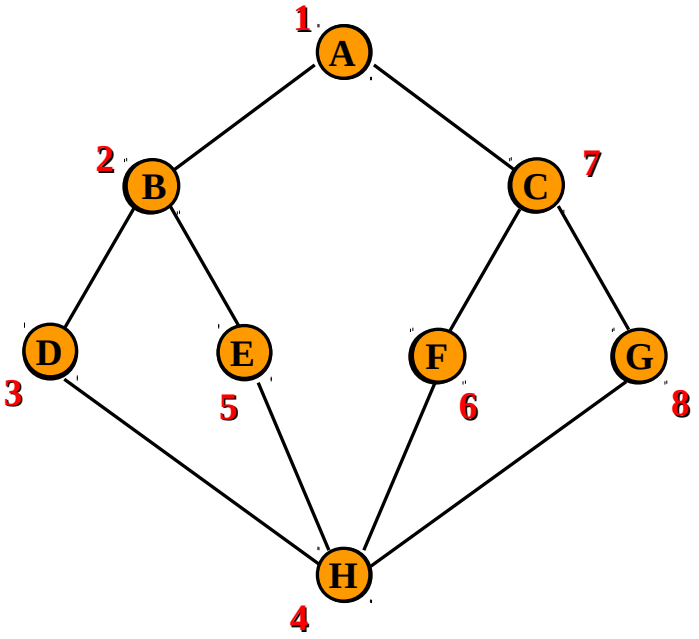
- Esse processo de busca pode ser representado por meio de uma árvore binária

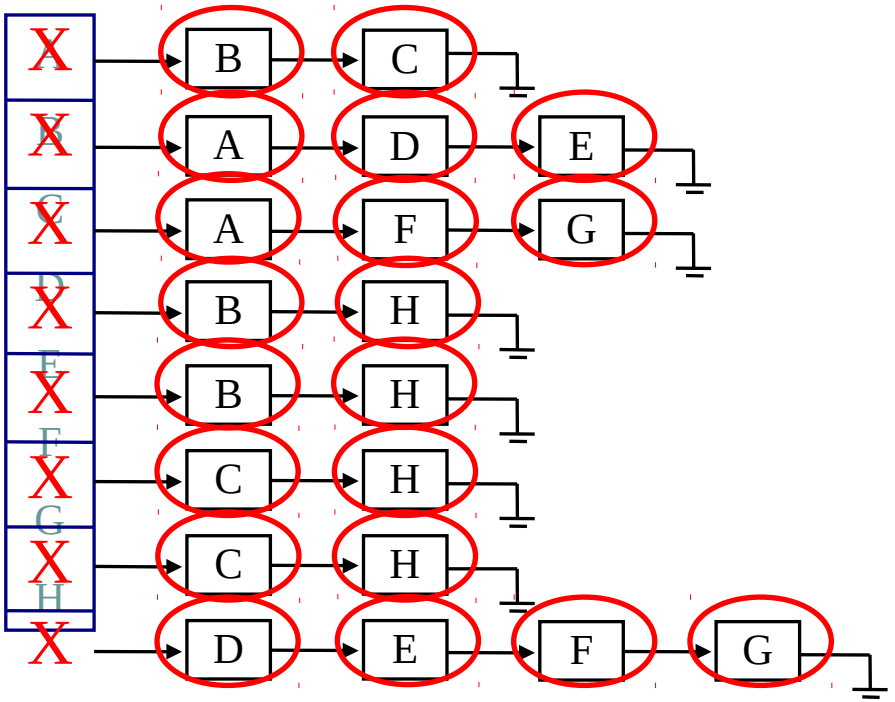




Busca em profundidade

- Caminhamento na lista de adjacências



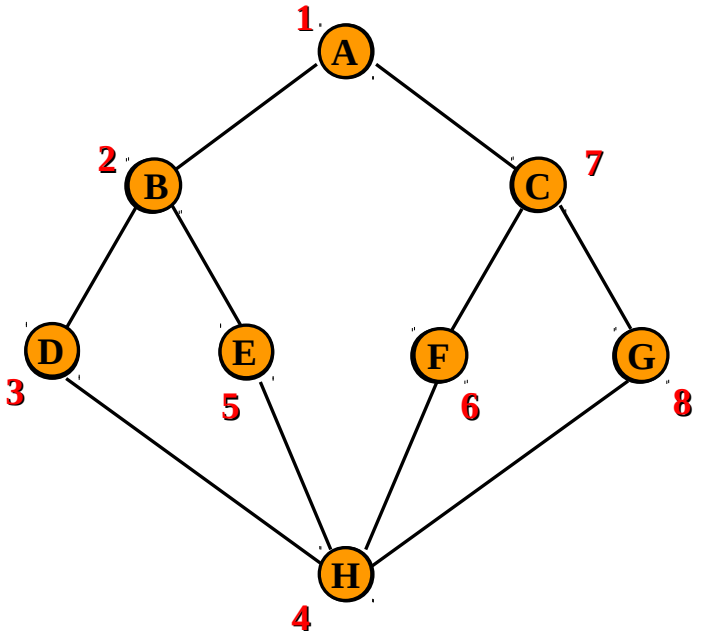


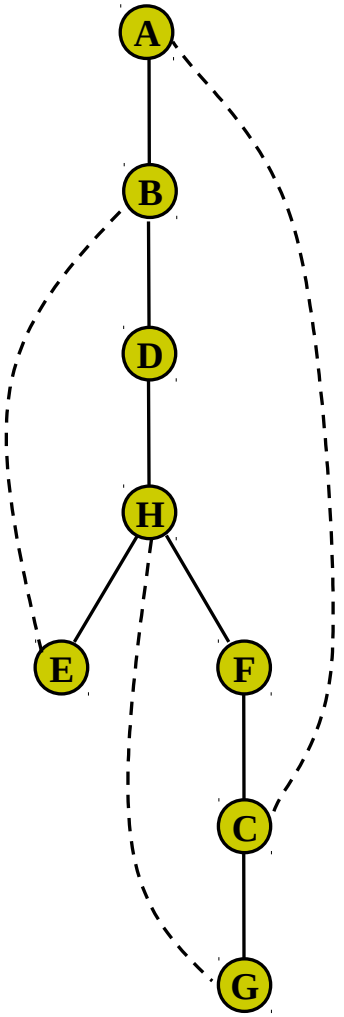
Não visitado

visitado

Busca em profundidade

- A árvore de busca em profundidade é gerada através de uma pilha





+ exemplos DFS

1) Aplique a **busca em profundidade**, ao *dígrafo* definido pelos arcos: 3-7 1-4 7-8 0-5 5-2 3-8 2-9 0-6 4-9 2-6 6-4, considerando sua representação em listas de adjacências e, depois em matriz de adjacências.

2) Repita o exercício considerando que temos um *grafo*, observando que o conjunto de arestas é montado na ordem listada.

+ exemplo ...

grafo