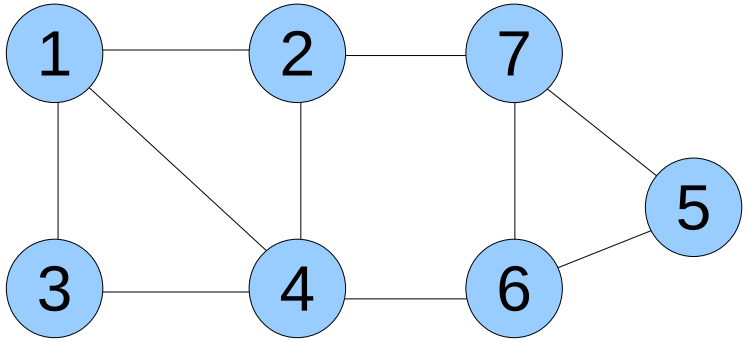
Grafos – Aula 5

Roteiro

- BFS – implementação
- Complexidade
- Busca em profundidade (DFS)
- Implementação
- Complexidade

Busca em Largura (BFS)

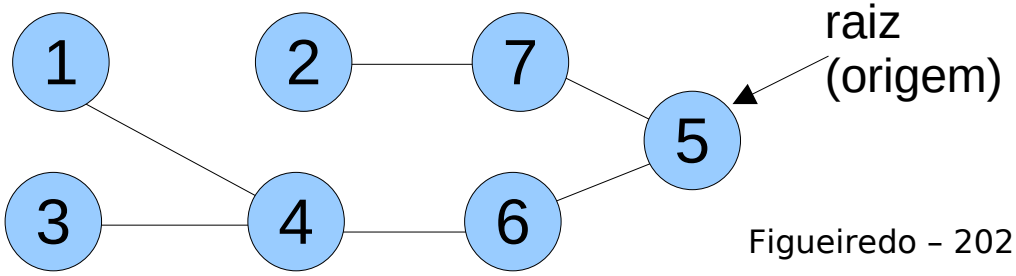
- Explorar vértices descobertos mais antigos primeiro



- Origem: vértice 5
- Em que ordem os vértices são *descobertos*?

Assumir que arestas são analisadas em ordem crescente dos vértices adjacentes

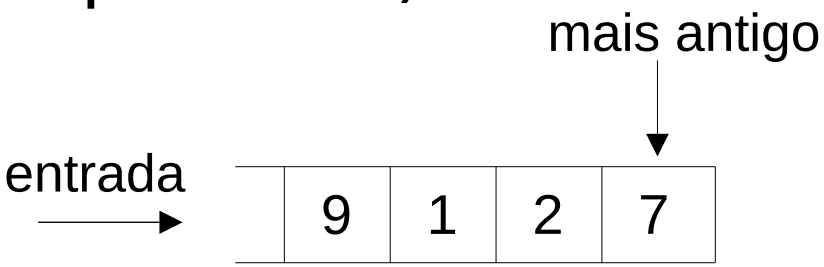
- Ordem: 5,6,7,4,2,1,3
- Árvore geradora induzida pela BFS



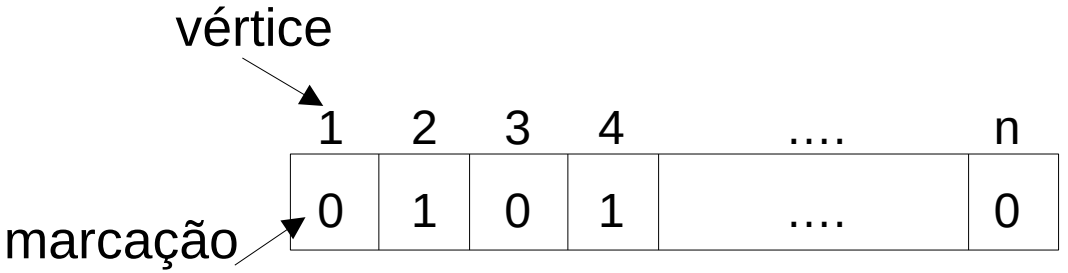
Implementação

- Como implementar a busca em largura?
- Estruturas de dados auxiliares

1) Manter ordem de descobrimento dos vértices (mais antigo primeiro)



2) Marcação dos vértices descobertos / desconhecidos



- **Fila**
  - inserir e remover elementos em tempo constante, O(1)
- **Vetor**
  - verificar e trocar marcação em tempo constante, O(1)

Implementação

- Algoritmo simples, sem árvore geradora, somente para percorrer o grafo


```
1.BFS(s)
2.   Desmarcar todos os vértices
3.   Definir fila Q vazia
4.   Marcar s e inserir s na fila Q
5.   Enquanto Q não estiver vazia
6.       Retirar v de Q
7.       Para todo vizinho w de v faça
8.          Se w não estiver marcado
9.             marcar w
10.            inserir w em Q
```

Significado e Analogia

- Três conjuntos da aula passada
  - desconhecido, descoberto, explorado

- Vértice não marcado = vértice desconhecido
- Vértice marcado e na fila = vértice descoberto
- Vértice marcado e fora da fila = vértice explorado

- Algoritmo constroi uma representação eficiente dos três conjuntos

Complexidade

- Qual é a complexidade deste algoritmo?
  - utilizando lista de adjacência para representar o grafo

```
1.Desmarcar todos os vértices
2.Definir fila Q vazia
3.Marcar s e inserir s na fila Q
4.Enquanto Q não estiver vazia
5.      Retirar v de Q
6.      Para todo vizinho w de v faça
7.         Se w não estiver marcado
8.            marcar w
9.            inserir w em Q
```

percorre os n vértices

quantos vértices em Q?

Percorre vizinhos do vértice, para cada vértice

$$\sum_{v \in V} d(v) = 2m$$

Complexidade

- O(n + m)
- Complexidade linear
  - mesma ordem de grandeza do tempo necessário para ler o grafo
- Se grafo for denso, $m \sim n^2$

Recordação

- O que é O(n + m)?
- Por que não O(n) ou O(m) ou $O(n^2)$?
- O(n + m) representa o **máximo** entre n e m ← "+" = max

Conectividade

- **Problema**: Como saber se existe caminho entre dois vértices?
  - ex. ir do Rio à Xapuri de carro?

Usar BFS para resolver este problema!

- Como?
- Marca s como raiz
- Realiza BFS
- Ao terminar a BFS se t estiver marcado, então há caminho, caso contrário, não há

Grafo Conexo

- **Problema:** como determinar se um grafo G é conexo?

Aplicações?

- Transporte aéreo comercial
  - voar de qualquer cidade para qualquer cidade

Idéia: utilizar BFS!

- Escolher vértice *v* qualquer de G
- Executar BFS à partir de *v*
- Verificar se todos vértices foram marcados

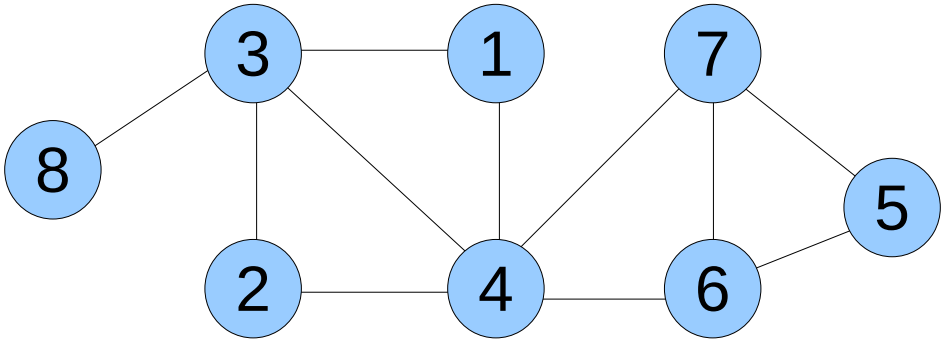
Busca em Profundidade (DFS)

- Explorar vértices descobertos mais recentes primeiro
  - Depth First Search (DFS)
- Oposto de BFS: explora mais antigo 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 apenas lugares já visitado)

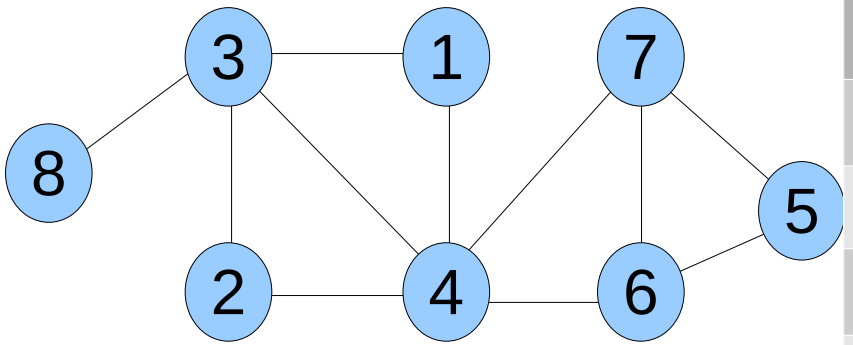
Busca em Profundidade

- Explorar o grafo abaixo usando DFS
  - início: vértice 4
  - vizinhos encontrados em ordem crescente



- Ordem de descobrimento dos vértices?

Exemplo



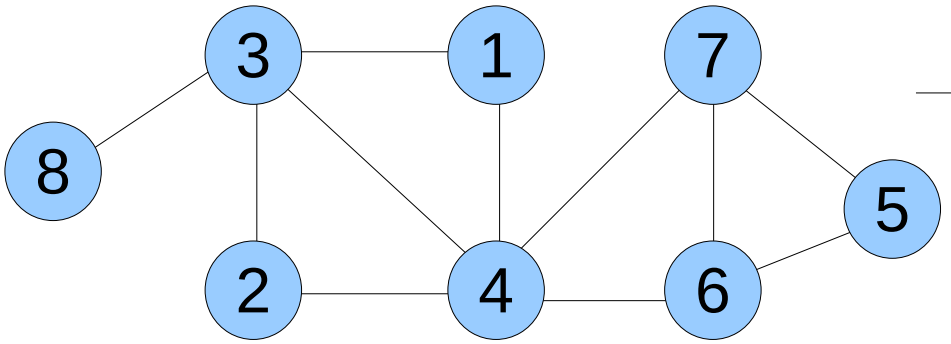
- Iniciar com vértice 4
- Analisar mais recente primeiro
  - analisa vizinhos em ordem crescente
- Ordem que vértices são adicionados no conjunto descoberto
- 4,1,3,2,8,6,5,7

Passo	Descoberto	Explorado	Desconhecido
0	4	-	1,2,3,5,6,7,8
1	4,1	-	2,3,5,6,7,8
2	4,1,3	-	2,5,6,7,8
3	4,1,3,2	-	5,6,7,8
4	4,1,3	2	5,6,7,8
5	4,1,3,8	2	5,6,7
6	4,1,3	2,8	5,6,7
7	4,1	2,8,3	5,6,7
8	4	2,8,3,1	5,6,7
9	4,6	2,8,3,1	5,7
10	4,6,5	2,8,3,1	7
11	4,6,5,7	2,8,3,1	-
12	4,6,5	2,8,3,1,7	-
13	4,6	2,8,3,1,7,5	-
14	4	2,8,3,1,7,5,6	-
15	-	2,8,3,1,7,5,6,4	-

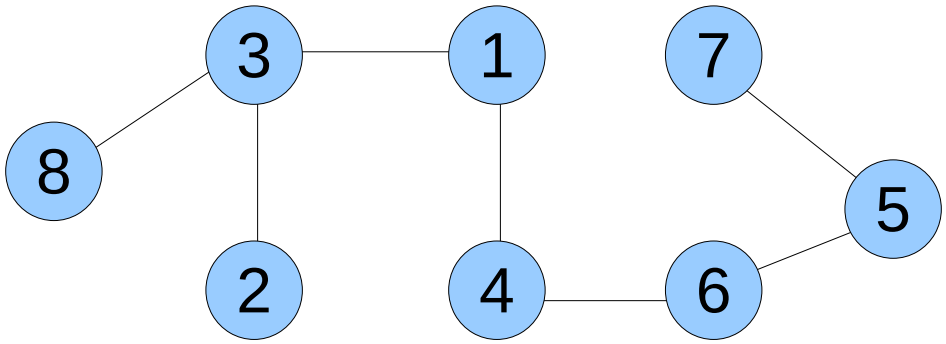
Árvore de Profundidade

- Árvore *induzida* pela busca em profundidade
  - Raiz: vértice de origem
  - Pai de v: nó que levou à descoberta de v

raiz: nó 4



Árvore de profundidade



- Ordem da busca *define* árvore

Implementação

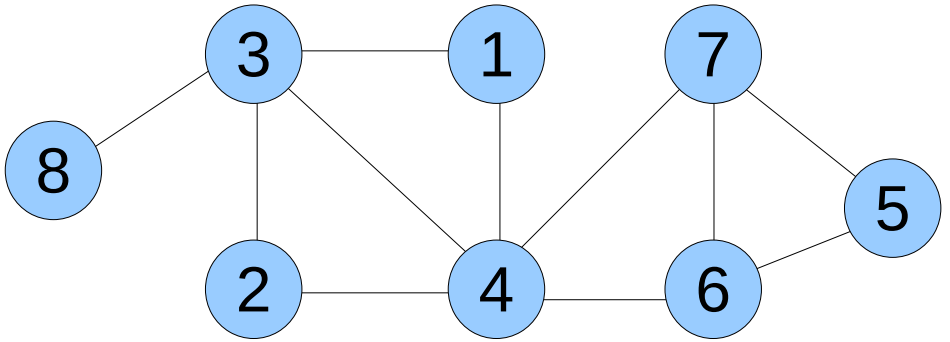
- Como implementar DFS?
  - utilizando algoritmo recursivo

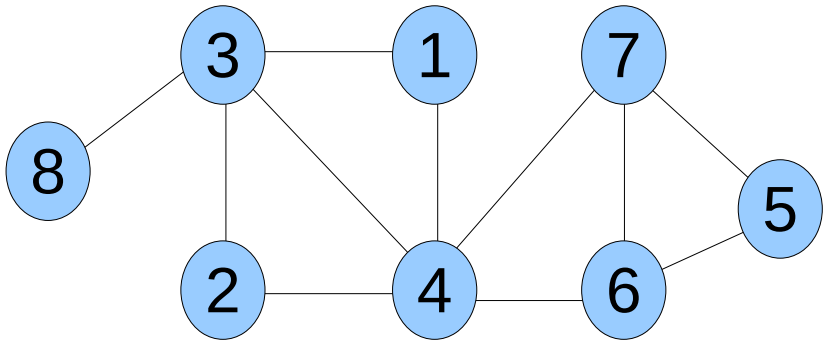
```
1.DFS(u)
2.  Marcar u
3.  Para cada aresta (u,v) incidente a u
4.    Se v não estiver marcado
5.      DFS(v)  // chamada recursiva
6.
7.Desmarcar todos os vértices
8.Escolher vértice inicial s
9.DFS(s)
```

DFS Recursiva

- Desvantagem da implementação?
- Empilha muitas chamadas de função em tempo de execução do programa
  - pode estourar a pilha do SO!

```
1.DFS(u)
2.  Marcar u
3.  Para cada aresta (u,v)
4.    Se v não estiver marcado
5.      DFS(v)
6.
7.Desmarcar todos os vértices
8.Escolher vértice inicial s
9.DFS(s)
```





Exemplo

DFS(1) gera quais chamadas?

```
-- DFS(1)
---- DFS(3)
------ DFS(2)
-------- DFS(4)
---------- DFS(6)
------------ DFS(5)
-------------- DFS(7)
---------------- Retorna // DFS(7), vizinhos 4, 5 e 6 marcados
-------------- Retorna // DFS(5), vizinhos 6 e 7 marcados
------------ Retorna // DFS(6), vizinhos 4, 5 e 7 marcados
---------- Retorna // DFS(4), vizinhos 1, 2, 3, 6 e 7 marcados
-------- Retorna // DFS(2), vizinhos 3 e 4 marcados
------ DFS(8)
-------- Retorna // DFS(8), vizinho 3 marcado)
------ Retorna // DFS(3), vizinhos 1, 2, 4 e 8 marcados)
---- Retorna // DFS(1), vizinhos 3 e 4 marcados)
```

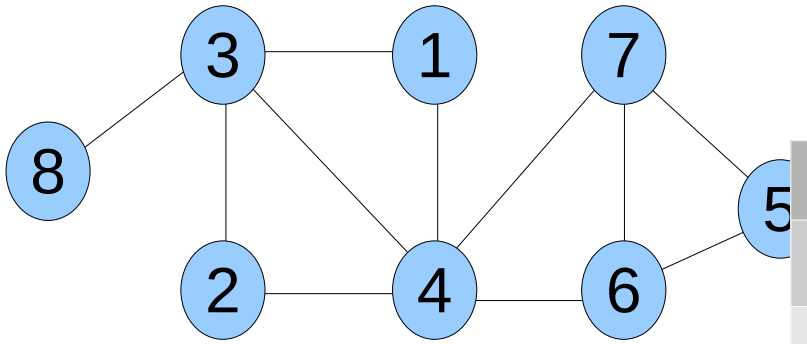
Sete chamadas na pilha de execução do processo!

Implementação

- Como implementar DFS?
  - utilizando uma pilha (sua)


```
1.DFS(s)
2.Desmarcar todos os vértices
3.Definir pilha P com um elemento s
4.Enquanto P não estiver vazia
5.    Remover u de P // no topo da pilha
6.    Se u não estiver marcado
7.       Marcar u
8.       Para cada aresta (u,v) incidente a u
9.          Adicionar v em P // no topo
```

Exemplo



- Chamada DFS(1)
- Percorrer vizinhos em ordem decrescente
  - menor valor fica no topo da pilha, ou seja menor valor primeiro
- Ordem que vértices são marcados (explorados)
- 1,3,2,4,6,5,7,8

Passo	Pilha	Marcado	Desmarcado
0	1	-	1,2,3,4,5,6,7,8
1	4,3	1	2,3,4,5,6,7,8
2	4,8,4,2,1	1,3	2,4,5,6,7,8
3	4,8,4,2	1,3	2,4,5,6,7,8
4	4,8,4,4,3	1,3,2	4,5,6,7,8
5	4,8,4,4	1,3,2	4,5,6,7,8
6	4,8,4,7,6,3,2,1	1,3,2,4	5,6,7,8
7	4,8,4,7,6,3,2	1,3,2,4	5,6,7,8
8	4,8,4,7,6,3	1,3,2,4	5,6,7,8
9	4,8,4,7,6	1,3,2,4	5,6,7,8
10	4,8,4,7,7,5,4	1,3,2,4,6	5,7,8
11	4,8,4,7,7,5	1,3,2,4,6	5,7,8
12	4,8,4,7,7,7,6	1,3,2,4,6,5	7,8
13	4,8,4,7,7,7	1,3,2,4,6,5	7,8
14	4,8,4,7,7,6,5,4	1,3,2,4,6,5,7	8
15	4,8,4,7,7,6,5	1,3,2,4,6,5,7	8
16	...		

Complexidade

- Qual é a complexidade deste algoritmo?

```
1.Desmarcar todos os vértices
2.Definir pilha P com um elemento s
3.Enquanto P não estiver vazia
4.      Remover u de P // no topo da pilha
5.      Se u não estiver marcado
6.         Marcar u
7.         Para cada aresta (u,v) incidente a u
8.            Adicionar v em P // no topo
```

- Custo para desmarcar vértices?
- Quantos vértices adicionados em P?
  - vértice adicionado em P mais de uma vez?
- Quantas vezes uma aresta é examinada?

Complexidade

- O(n + m)
- Complexidade linear
  - mesma ordem de grandeza do tempo necessário para ler o grafo
- Se grafo for denso, $m \sim n^2$
- Mesma complexidade que BFS!

Árvore Geradoras

- Como obter árvore geradora induzida pela BFS ou DFS?
- Modificar algoritmo para gerar a árvore
- **Idéia**
  - Utilizar vetor pai[] para indicar o pai de um vértice *v* na árvore
    - todo vértice tem exatamente um pai
  - Idéia independe do tipo de busca (BFS, DFS, etc)

Exercício da lista