ENG. INFORMÁTICA E DE COMPUTADORES

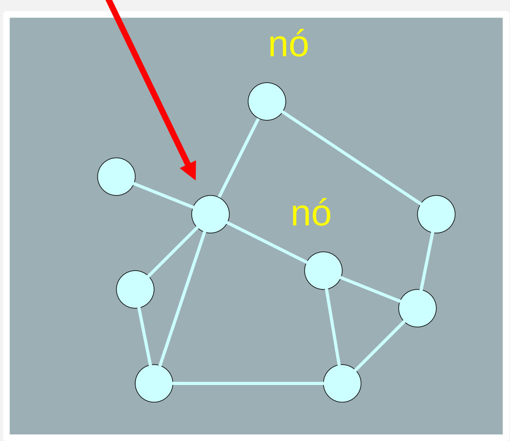
Algoritmos e Estruturas de Dados

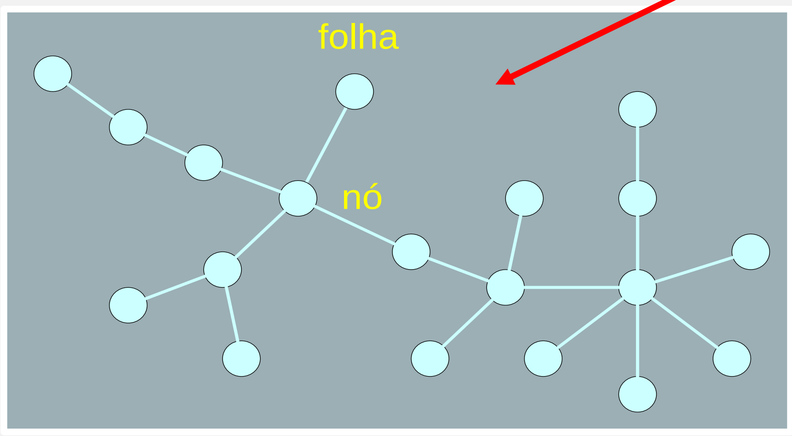
(parte 15)

2º Semestre 2021/2022
Instituto Superior de Engenharia de Lisboa
Paula Graça

GRAFOS

- Árvore Livre
  - É uma colecção não vazia de nós e arestas. Um nó é um objecto que pode conter informação associada (chave). Uma aresta é uma ligação entre dois nós
  - Num grafo existe mais do que um caminho de ligação entre qualquer par de nós
  - Se existir apenas um caminho, não se trata de um grafo mas de uma árvore

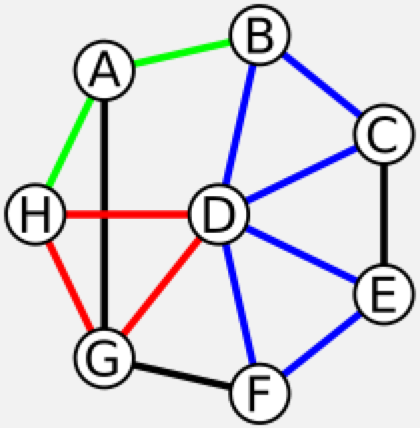




GRAFOS

- Um grafo é uma colecção de pontos num plano designados por vértices (*vertex*) ou nós, alguns dos quais ligados por arestas (*edges*) ou arcos

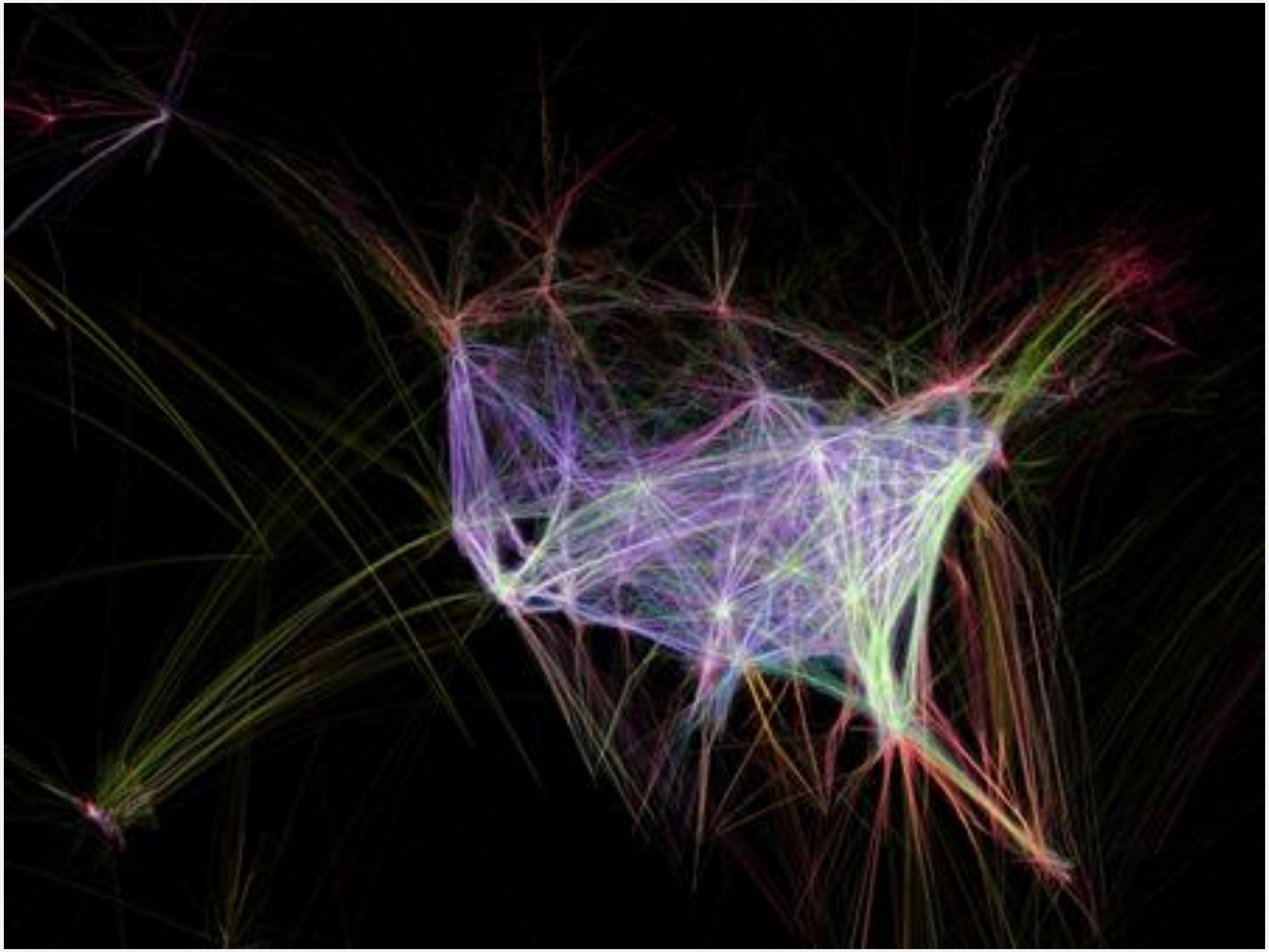
- Formalmente, um grafo
  - G = (V, E)
  - É definido por dois conjuntos:
    - um conjunto finito V de itens vertex
    - um conjunto finito E de pares de itens vertex, designados por edges



EXEMPLOS DE GRAFOS

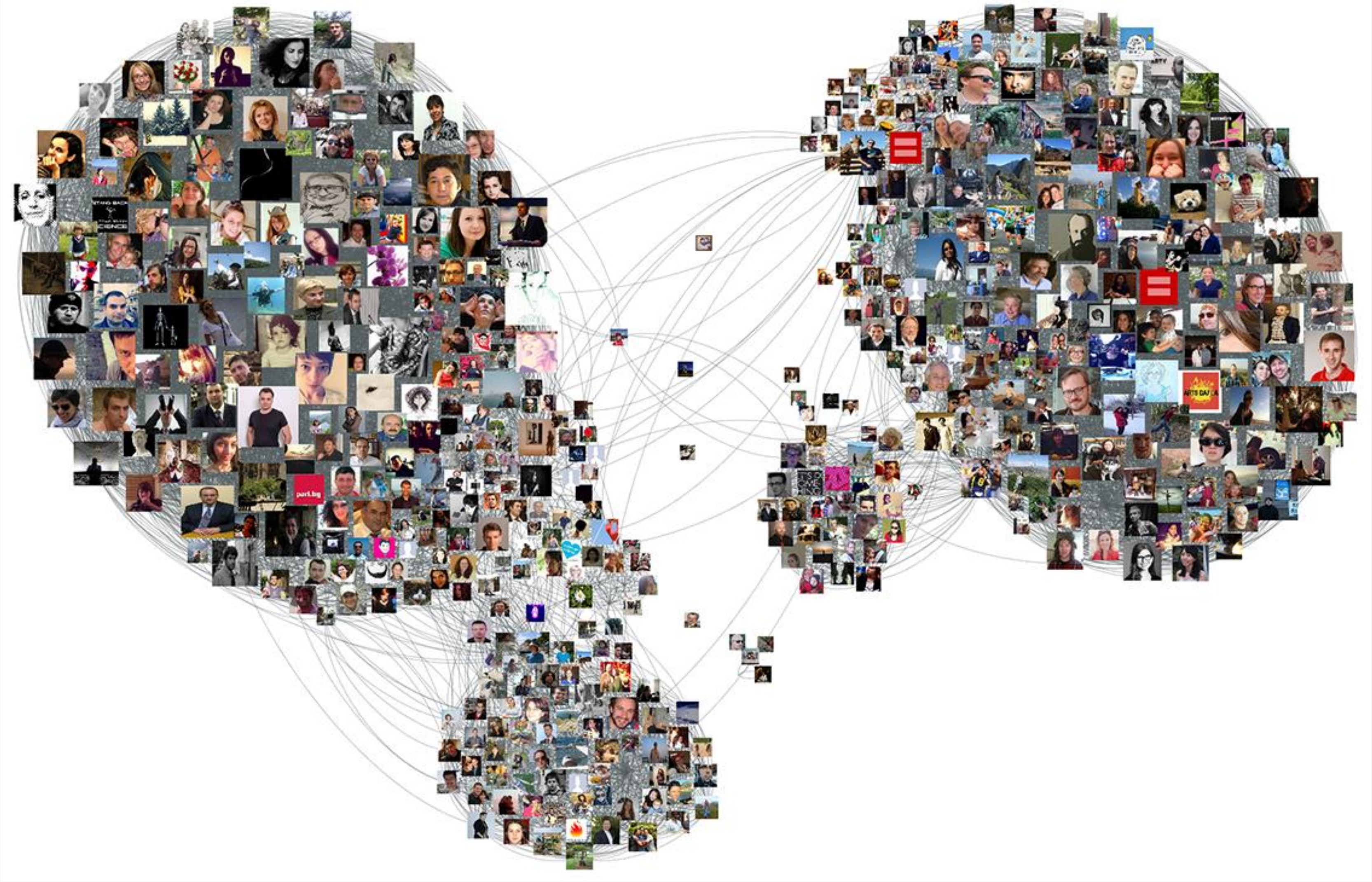
Rede de metro de Lisboa

EXEMPLOS DE GRAFOS



Grafo de tráfego aéreo

EXEMPLOS DE GRAFOS

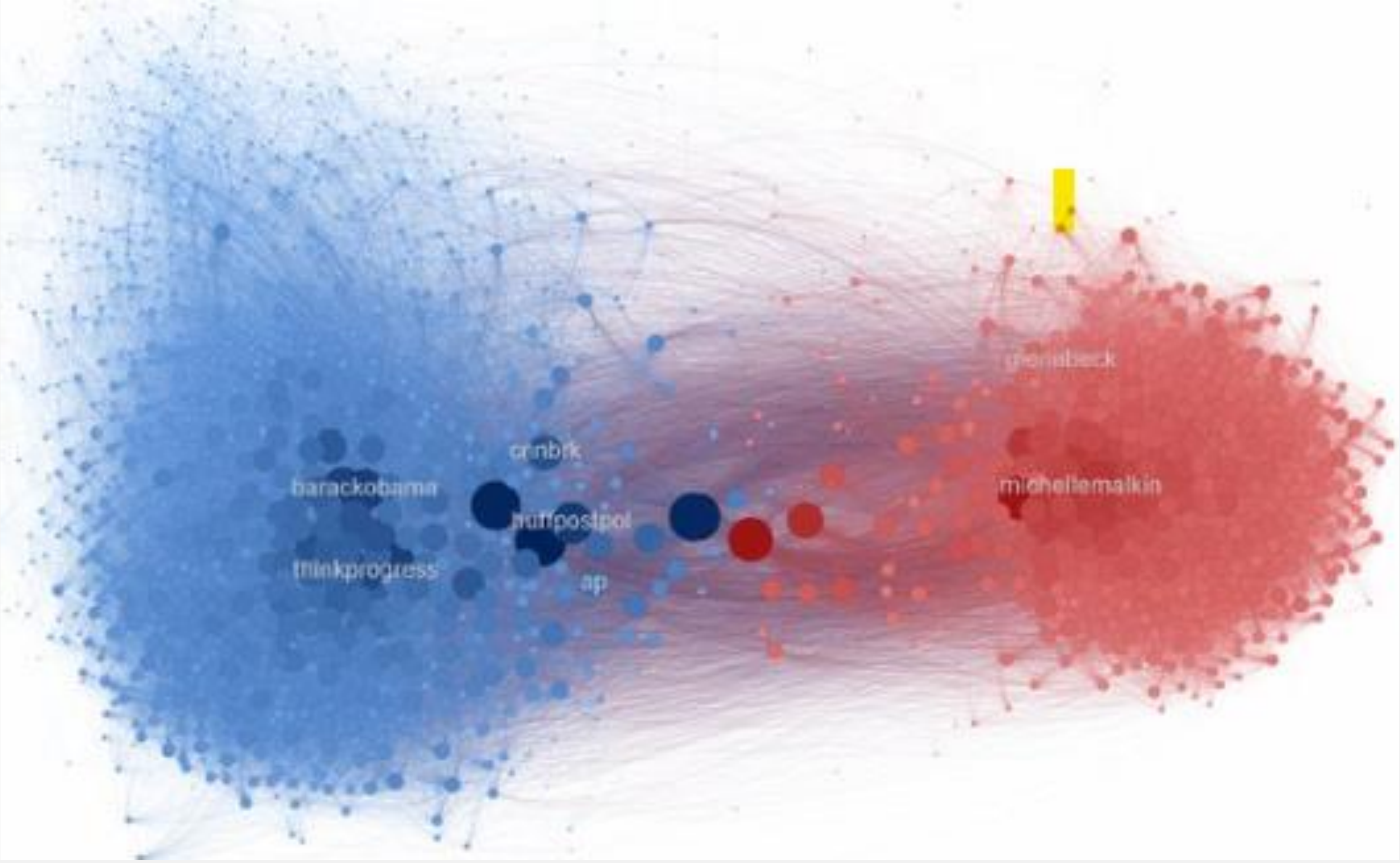


Facebook – Identificação de agrupamentos de redes de amizade

EXEMPLOS DE GRAFOS

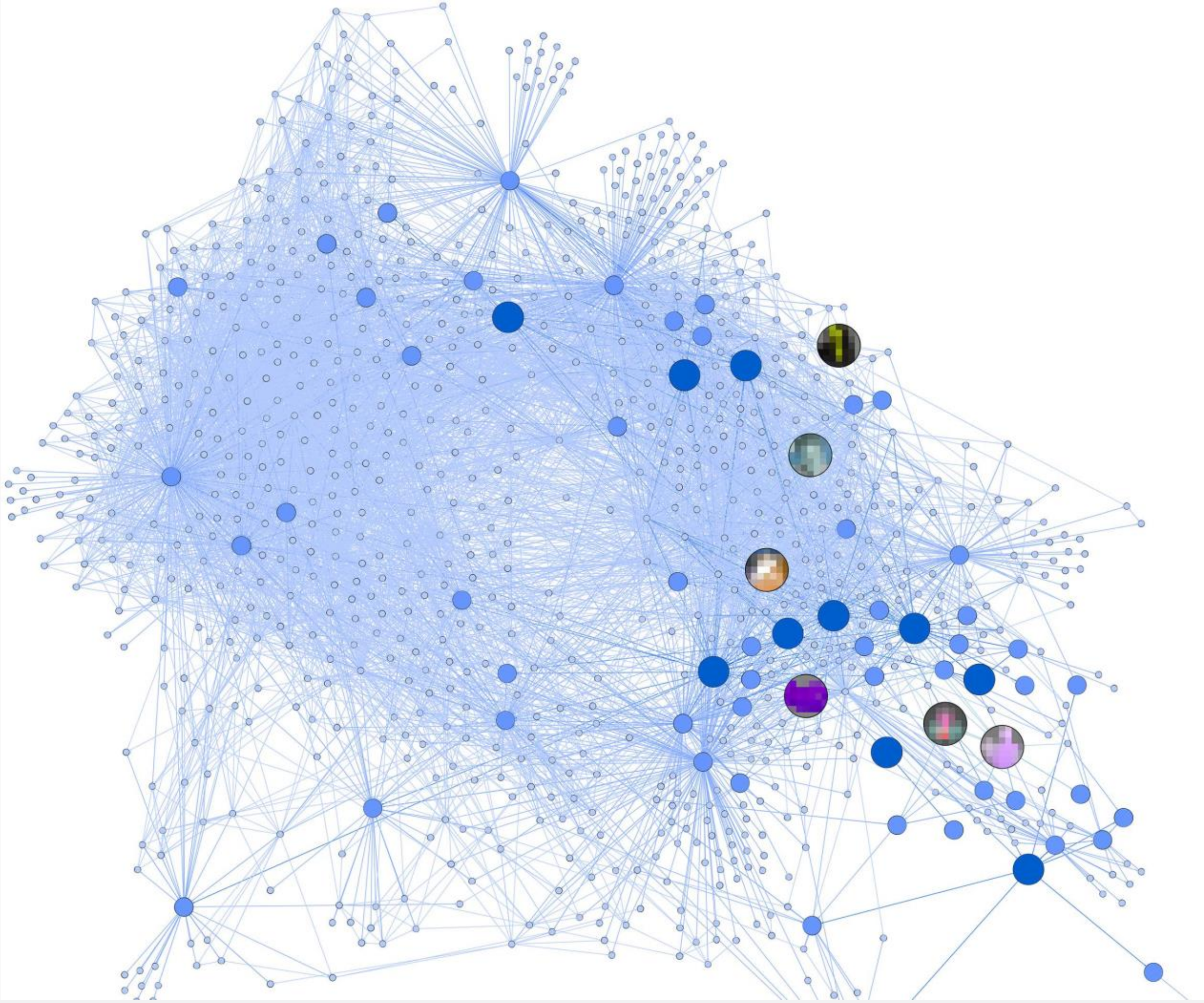
Twitter – Mapas de conversação ("*tweets/retweets*")

EXEMPLOS DE GRAFOS



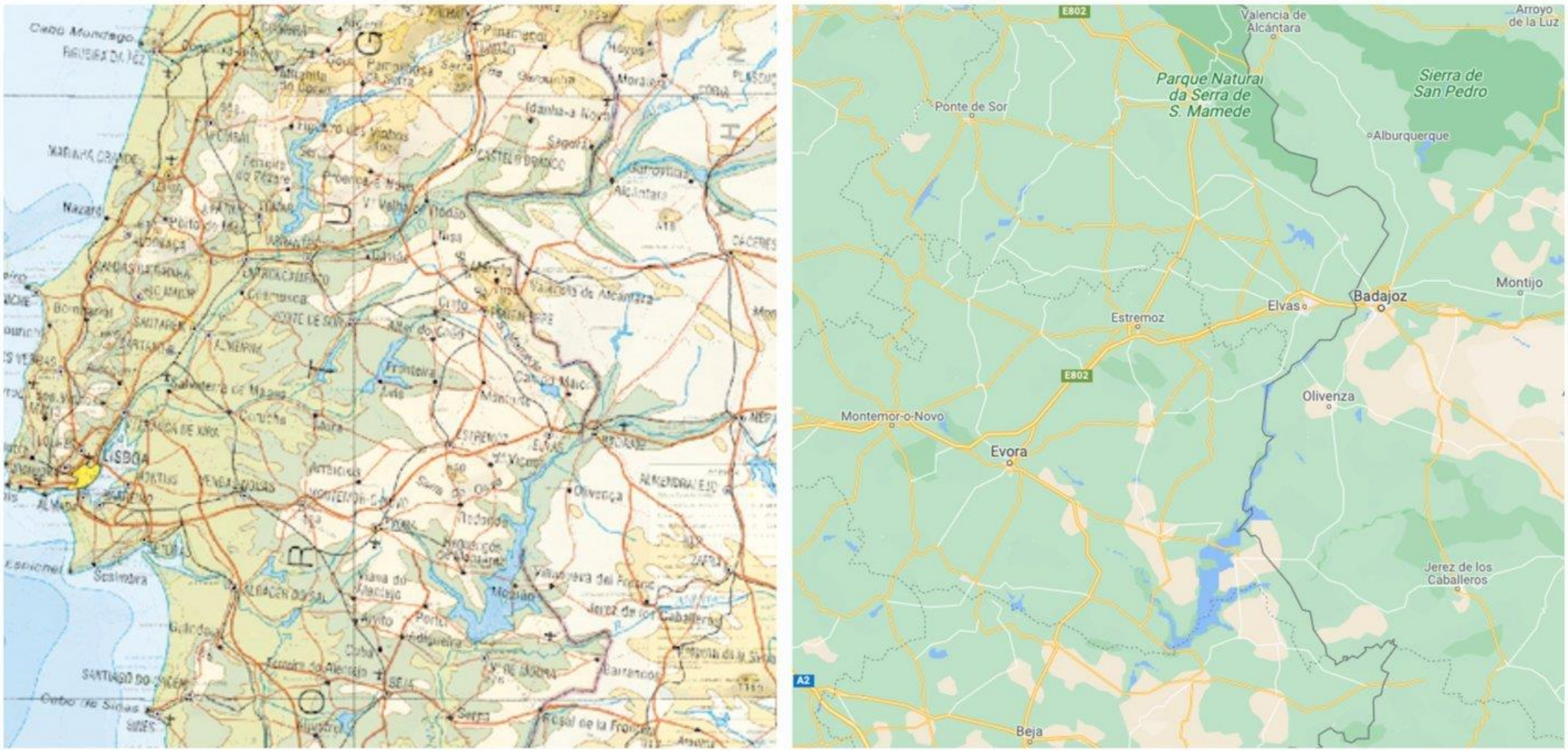
Twitter – Identificação de orientação política liberal/conservadora de "*retweets*" durante um debate

EXEMPLOS DE GRAFOS



You Tube – Identificação de "*influencers*"

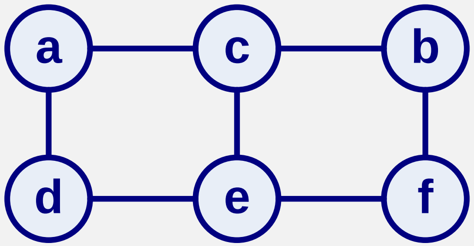
EXEMPLOS DE GRAFOS



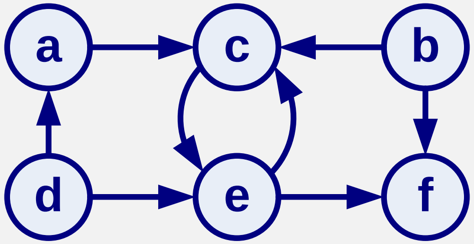
Google Maps – Mapa de estradas

GRAFOS

- Grafo não dirigido
  - Os pares de vértices não estão ordenados
  - i.e. o par de vértices (v,u) é igual ao par (u,v)

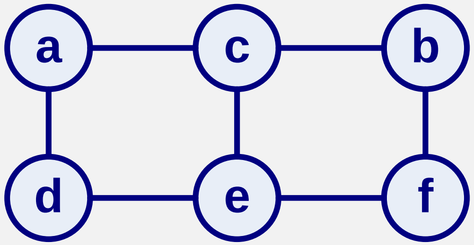


- Grafo dirigido ou dígrafo
  - Os pares de vértices estão ordenados
  - i.e. a aresta (v,u) está direcionada de v para u



GRAFOS

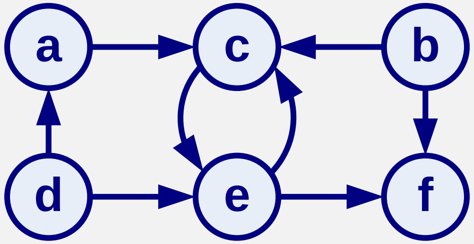
- O seguinte grafo não dirigido, tem 6 vértices e 7 arestas



V = { a, b, c, d, e, f }

E = { (a, c), (a, d), (b, c), (b, f), (c, e),
(d, e), (e, f) }

- O seguinte grafo dirigido, tem 6 vértices e 8 arestas direcionadas

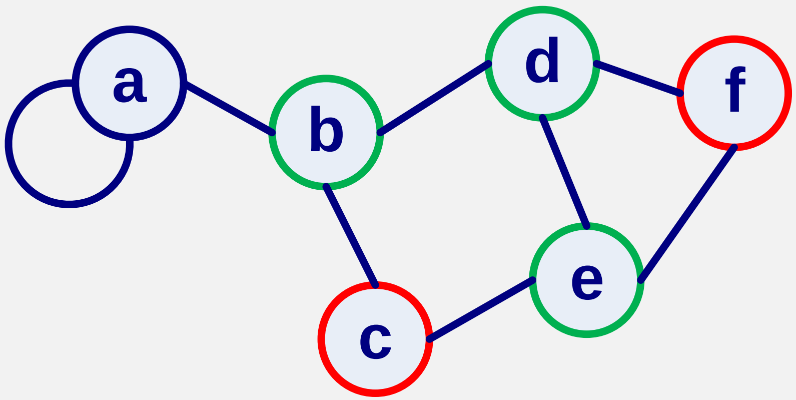


V = { a, b, c, d, e, f }

E= { (a, c), (b, c), (b, f), (c, e), (d, a),
(d, e), (e, c), (e, f) }

GRAU E ADJACÊNCIA

- O grau (ou valência) de um vértice é o número de arestas que partem do vértice

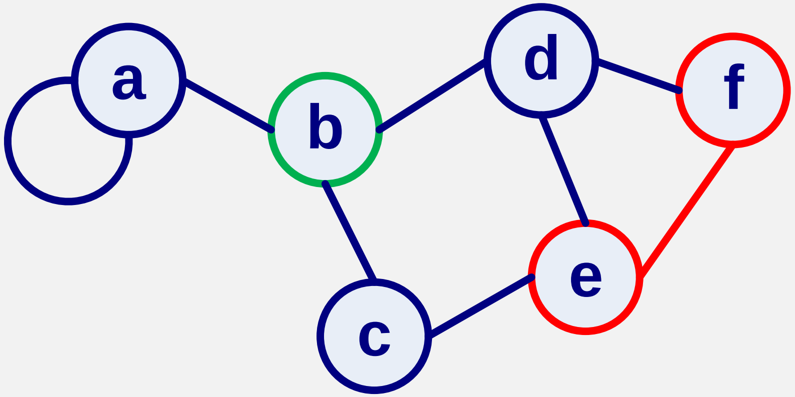


Os vértices **(f, c)** têm grau **2**

Os vértices **(b, d, e)** têm grau **3**

GRAU E ADJACÊNCIA

- Dois vértices são considerados adjacentes se existe uma aresta entre eles

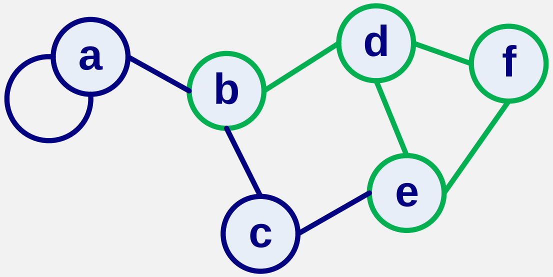


Os vértices **(f, e)** são adjacentes

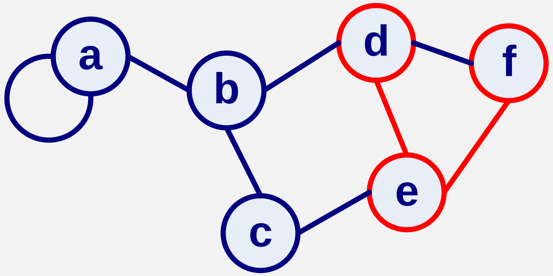
Os vértices **(b, e)** não são adjacentes

CAMINHOS

- Um caminho é uma sequência de vértices conectados por arestas, sem repetição de arestas
- O comprimento (ou distância) do caminho é o número de arestas usadas desde o vértice origem ao destino



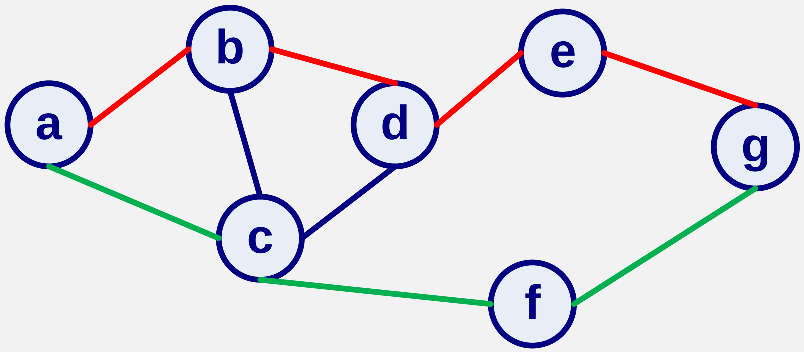
(d, f, e, d, b) caminho de comprimento **4**



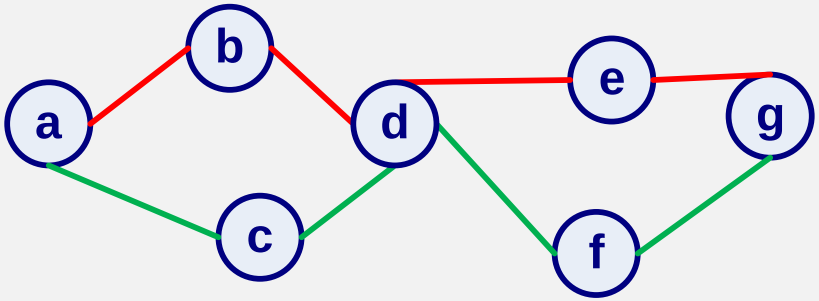
(d, e, f) caminho de comprimento **2**

CAMINHOS

- Dois caminhos entre o vértice origem e destino são vértice-independentes se não existe nenhum vértice interno comum
- Dois caminhos entre o vértice origem e destino são aresta-independentes se não existe nenhuma aresta comum



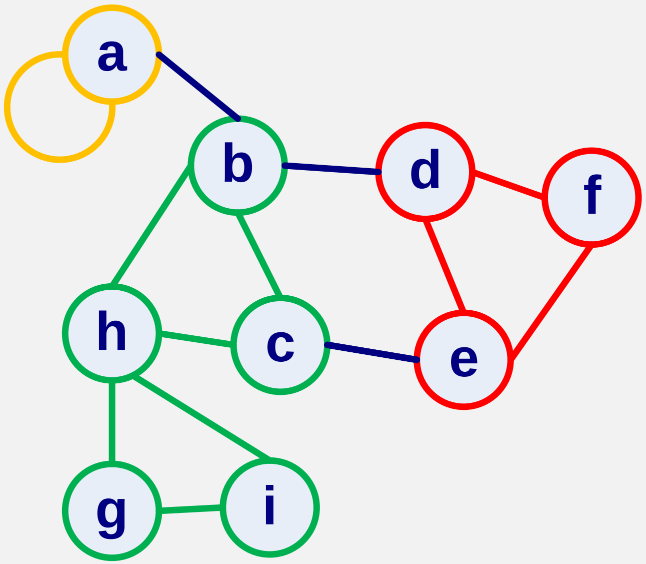
(a, b, d, e, g) e **(a, c, f, g)** são caminhos vértice-independentes e aresta-independentes



(a, b, d, e, g) e **(a, c, d, f, g)** são caminho aresta-independentes

CIRCUITOS/CICLOS

- Um circuito (ou ciclo) é um caminho que começa e acaba no mesmo vértice
  - Circuitos de comprimento 1 são laços
  - Um circuito (não laço) é um caminho fechado que tem um comprimento de pelo menos 3 arestas



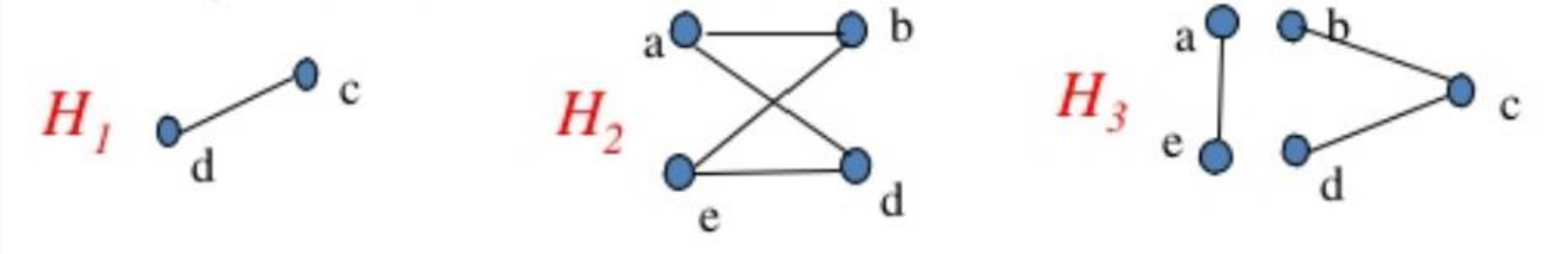
(a) laço

(f, d, e, f) circuito de comprimento 3

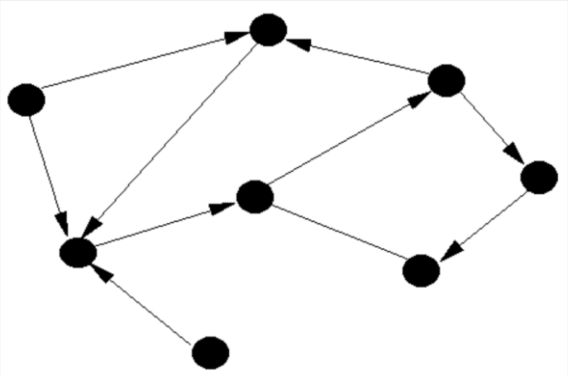
(b, h, g, i, h, c, b) circuito de comprimento 6

CIRCUITOS/CICLOS

- Um grafo é ligado (ou conexo), se for possível estabelecer um caminho de qualquer vértice para qualquer outro vértice
  - H1 e H2 são grafos ligados
  - H3 é um grafo não ligado
  - H4 é um grafo não ligado

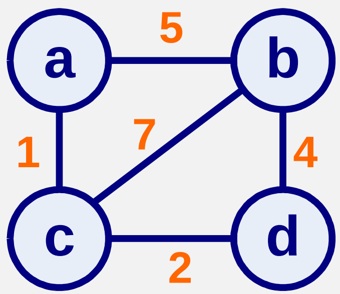


H_4



GRAFO PESADO

- Um grafo pesado é um grafo em que cada aresta tem associado um peso
  - Pode descrever a natureza do vértice/aresta: distância, custo, capacidade, tempo, etc.

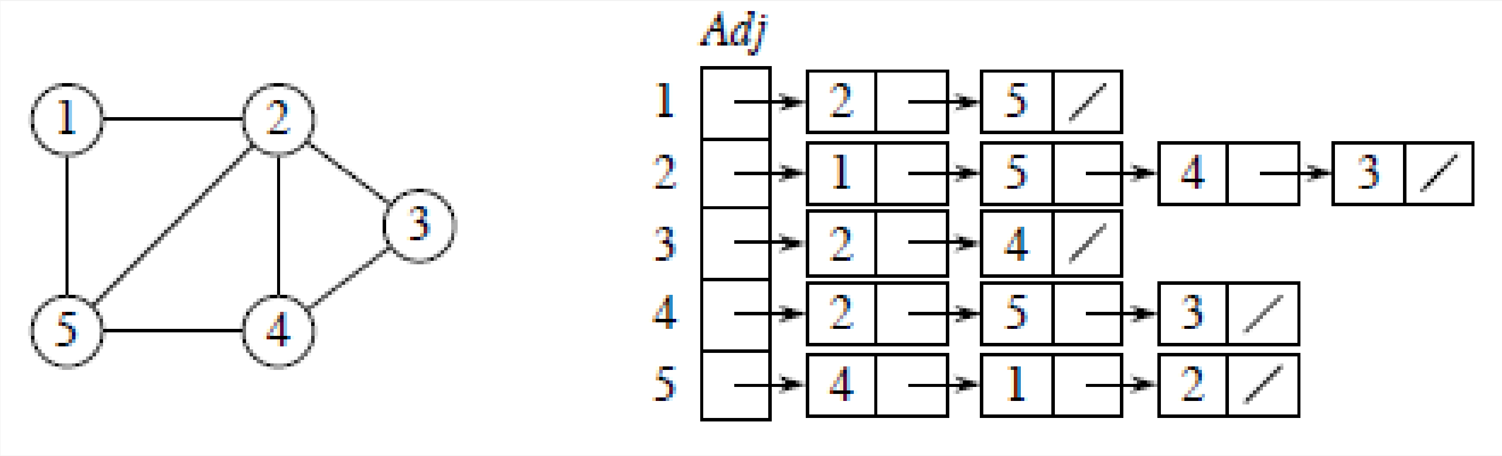


(a, b, d) caminho de custo 9

- O custo de um caminho num grafo pesado é soma dos custos das arestas percorridas

REPRESENTAÇÃO DE GRAFOS

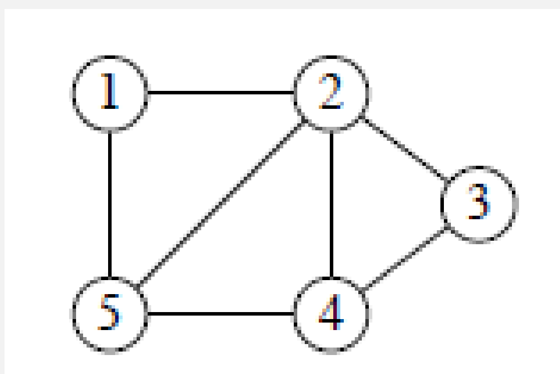
- Os grafos são representados em algoritmos de duas formas principais
  - Listas de adjacência
    - As listas de adjacência podem ser representadas por conjuntos de vértices, pois a ordem entre eles é indiferente



Exemplo de um grafo não dirigido

REPRESENTAÇÃO DE GRAFOS

- Matriz de adjacência
  - A matriz de adjacência de um grafo com n vértices, é uma matriz n-por-n, com uma linha e coluna por cada vértice
    - O elemento na linha i e coluna j, é igual a 1, se existir uma aresta do vértice i para o vértice j
    - Caso contrário, é igual a 0

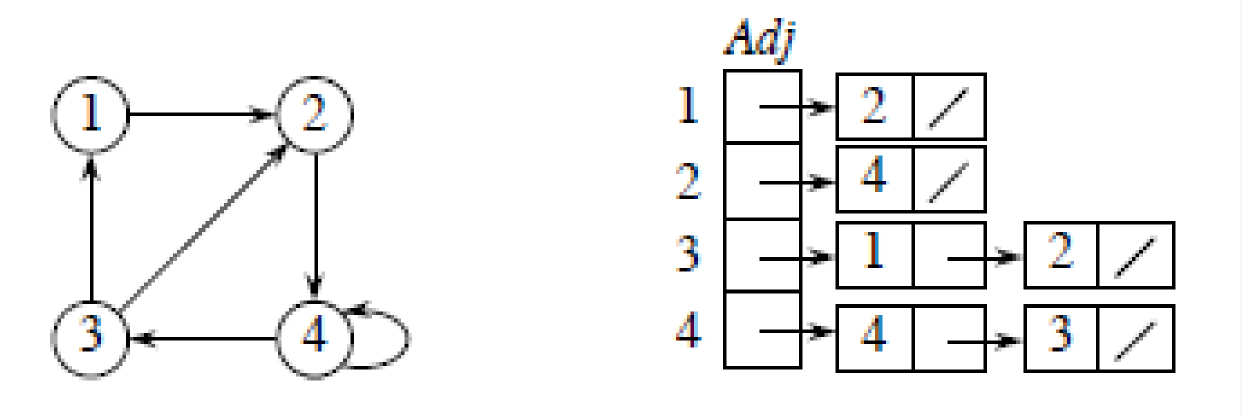


	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

Exemplo de um grafo não dirigido

REPRESENTAÇÃO DE GRAFOS

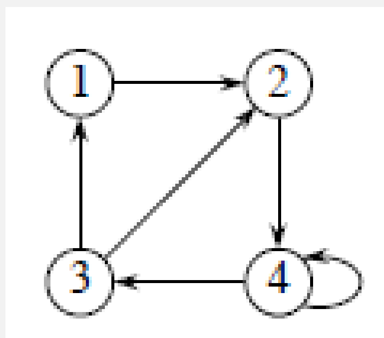
- Listas de adjacência
  - As listas de adjacência podem ser representadas por conjuntos de vértices, pois a ordem entre eles é indiferente



Exemplo de um grafo dirigido

REPRESENTAÇÃO DE GRAFOS

- Matriz de adjacência

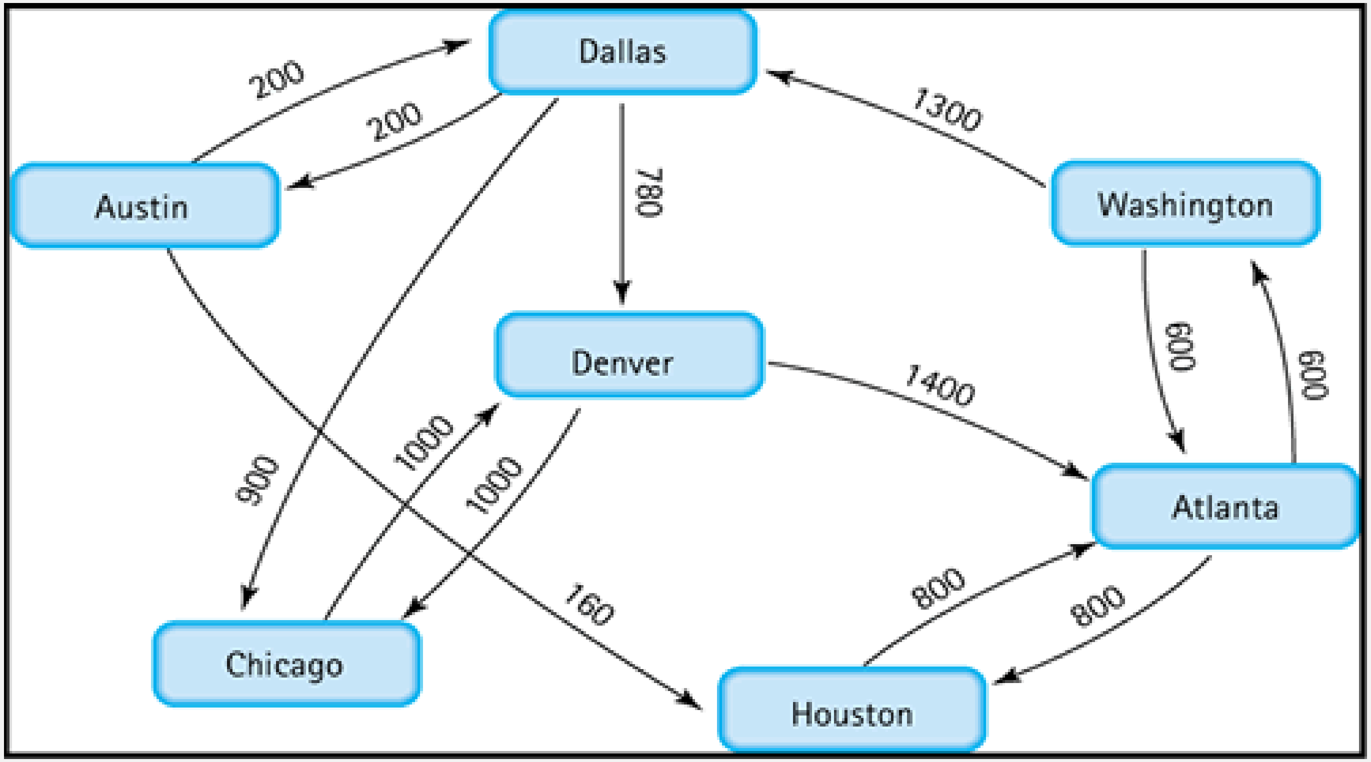


	1	2	3	4
1	0	1	0	0
2	0	0	0	1
3	1	1	0	0
4	0	0	1	1

Exemplo de um grafo dirigido

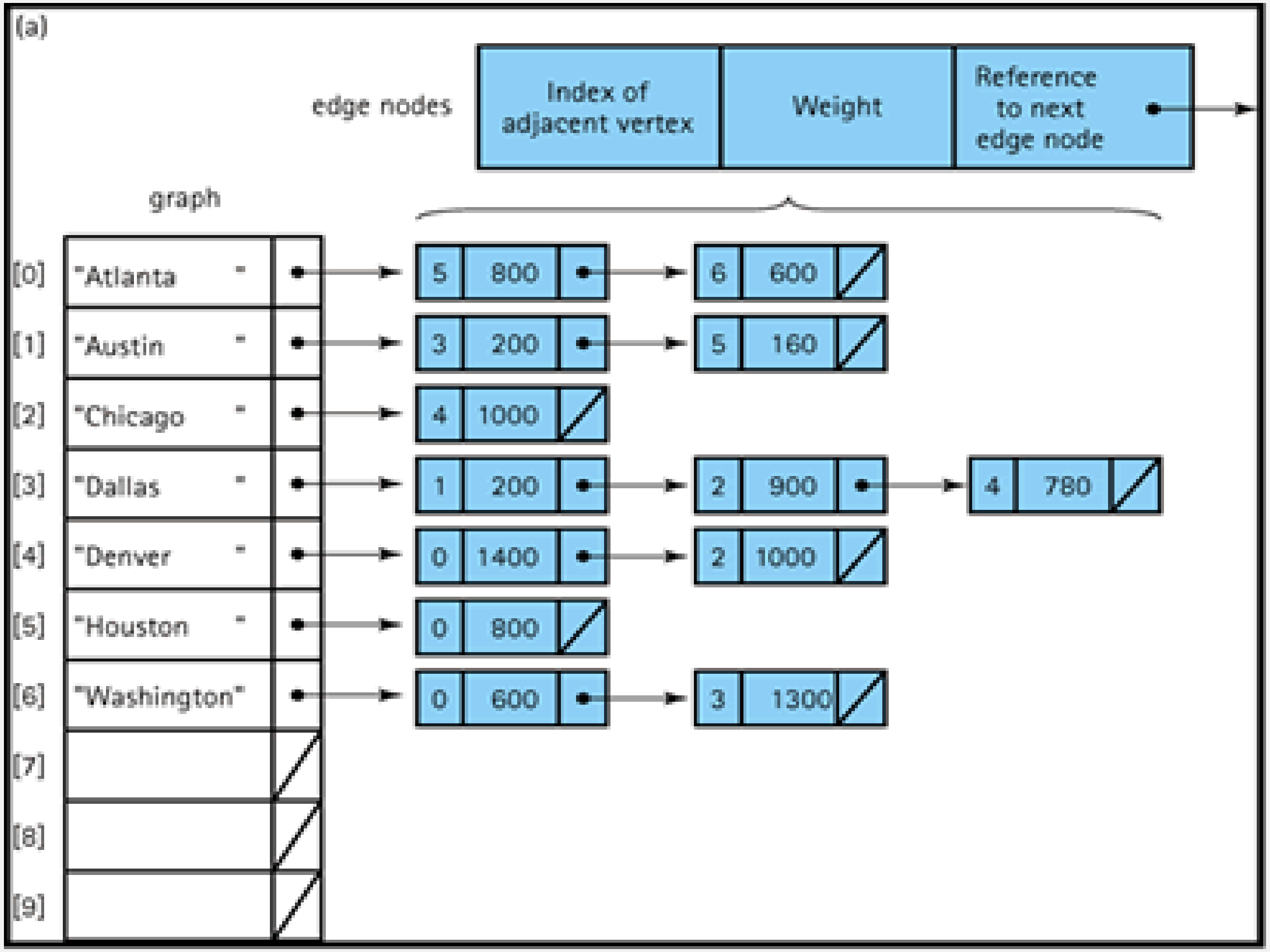
REPRESENTAÇÃO DE GRAFOS

- Representação de grafos pesados
  - Exemplo de um grafo representativo das ligações aéreas, incluindo as respetivas distâncias, entre algumas das principais capitais dos Estados Unidos da América



REPRESENTAÇÃO DE GRAFOS

- Representação do grafo através das listas de adjacência. Os pesos (distâncias) são colocados nas listas



REPRESENTAÇÃO DE GRAFOS

- Representação do grafo através da matriz de adjacência. Os pesos (distâncias) são colocados na matriz, na posição linha e coluna que corresponde à ligação

graph
.num Vertices 7
.vertices

[0]	"Atlanta "
[1]	"Austin "
[2]	"Chicago "
[3]	"Dallas "
[4]	"Denver "
[5]	"Houston "
[6]	"Washington"
[7]	
[8]	
[9]	

.edges

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
[0]	0	0	0	0	0	800	600	•	•	•
[1]	0	0	0	200	0	160	0	•	•	•
[2]	0	0	0	0	1000	0	0	•	•	•
[3]	0	200	900	0	780	0	0	•	•	•
[4]	1400	0	1000	0	0	0	0	•	•	•
[5]	800	0	0	0	0	0	0	•	•	•
[6]	600	0	0	1300	0	0	0	•	•	•
[7]	•	•	•	•	•	•	•	•	•	•
[8]	•	•	•	•	•	•	•	•	•	•
[9]	•	•	•	•	•	•	•	•	•	•

(Array positions marked '•' are undefined)

REPRESENTAÇÃO DE GRAFOS

- Listas de adjacência - Vantagens
  - Inicialização é proporcional a V
  - Utiliza sempre espaço proporcional a V+E
    - Adequado para grafos esparsos
    - Algoritmos que assentem na análise de arestas em grafos
  - Adição de arestas é feita de forma eficiente

- Listas de adjacência - Desvantagens
  - Remoção e pesquisa de arestas
    - Pode levar um tempo proporcional a V
  - Não aconselhável para
    - Grafos de grande dimensão que não podem ter arestas paralelas
    - Grande utilização de remoção de arestas

REPRESENTAÇÃO DE GRAFOS

- Matriz de adjacência - Vantagens
  - Representação mais adequada quando
    - Há espaço disponível
    - Grafos são densos
- Adição e remoção de arestas é feita de forma eficiente
- Fácil detetar a existência de arestas paralelas (repetidas)
- Fácil determinar se dois vértices estão ou não ligados

- Matriz de adjacência – Desvantagens
  - Os grafos requerem espaço de memória proporcional a V^2
  - Os grafos muito esparsos com um número muito elevado de vértices exige muito espaço de memória desperdiçado

REPRESENTAÇÃO DE GRAFOS

- Desempenho

	Matriz de Adjacência	Lista de Adjacência
Espaço	$O(V^2)$	$O(V + E)$
Inicialização	$O(V^2)$	$O(V)$
Inserir aresta	$O(1)$	$O(1)$
Procurar aresta	$O(1)$	$O(V)$
Remover aresta	$O(1)$	$O(V)$

GRAFOS - VARRIMENTO

- Existem vários algoritmos para percorrer de uma forma sistemática todos os vértices de um grafo, seguindo as ligações definidas pelas arestas:
- Pesquisa em Largura (BFS-Breadth-First-Search)
  - O objectivo do algoritmo é percorrer o grafo tão próximo quanto possível do vértice inicial de varrimento
  - Usa uma Fila (Queue) na implementação, sendo um protótipo para alguns importantes algoritmos sobre grafos:

    Prim e Kruskal (árvore geradora mínima)

    Dijkstra (caminho mais curto com fonte única)

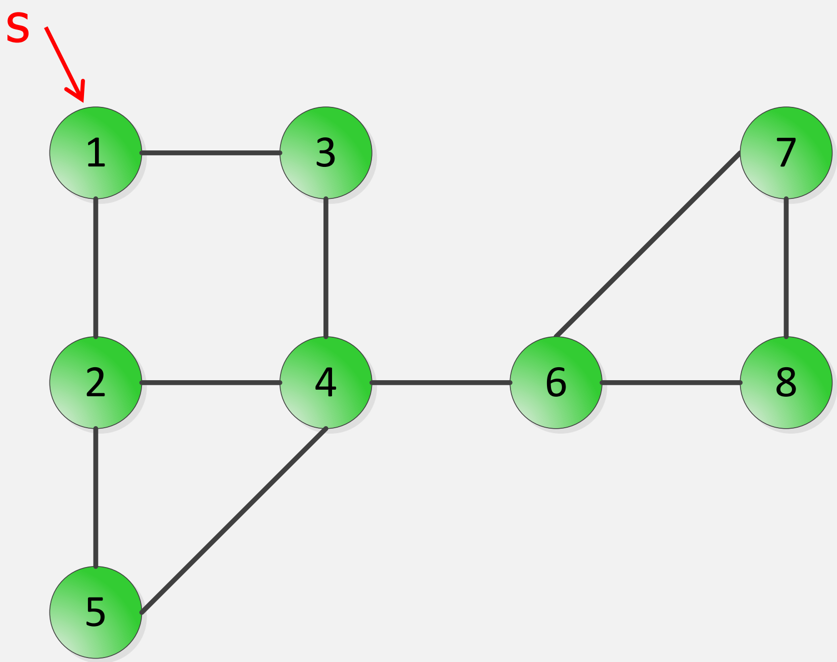
- Pesquisa em Profundidade (DFS-Depth-First-Search)

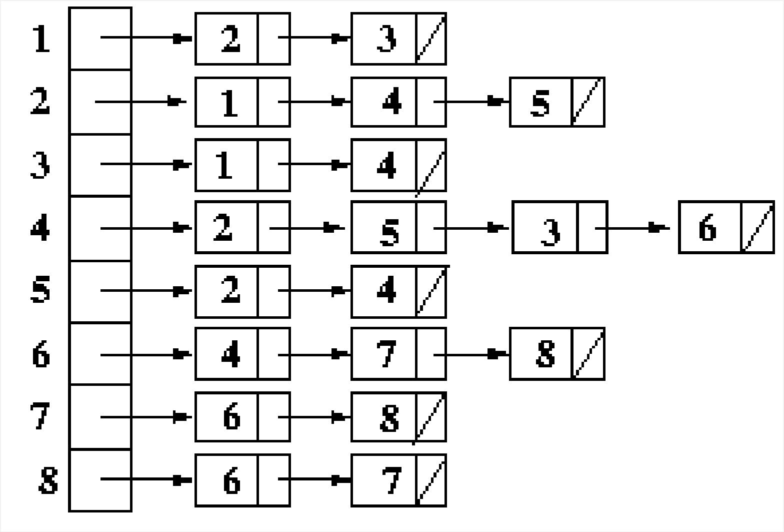
  - O objectivo do algoritmo é percorrer o grafo de tal forma que tenta ir o mais longe possível do vértice inicial de varrimento

  - Usa uma Pilha (Stack) na implementação
    - O algoritmo de Euler usa esta pesquisa (circulo/caminho Euleriano)
    - Permite verificar se uma grafo é acíclico, ou se tem um ou mais ciclos
    - Ordenação topológica em grafos orientados acíclicos (indica uma sequência válida de vértices)

GRAFOS - VARRIMENTO

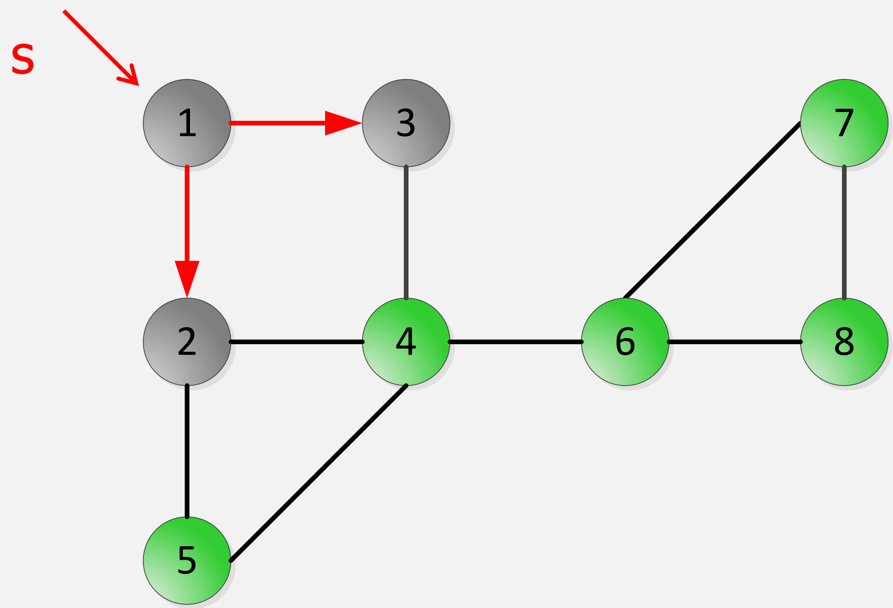
- Considere-se o seguinte grafo G = (V, E), representado pelas listas de adjacência, e o nó inicial S





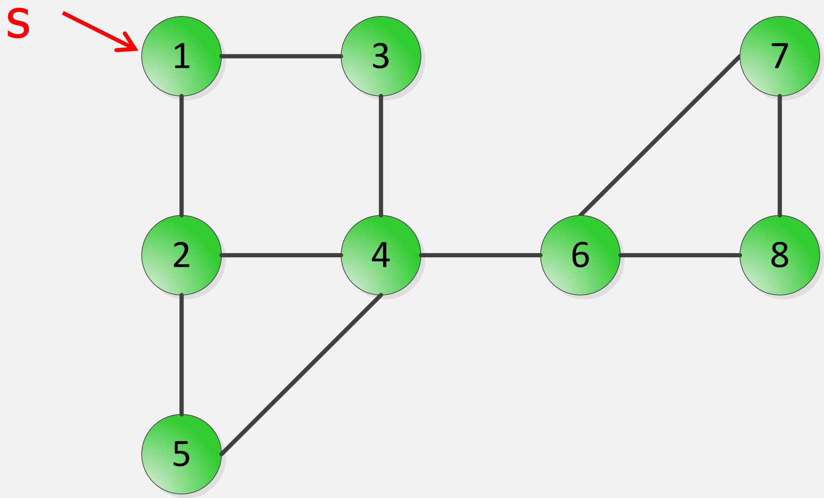
PESQUISA EM LARGURA

- BFS (Breadth-First-Search)
  - Dado um grafo G = (V, E) e um vértice inicial S (*source*), as arestas são sistematicamente exploradas de forma a visitar todos os vértices adjacentes de S
  - Assim sucessivamente, para os vértices adjacentes de S ainda não visitados

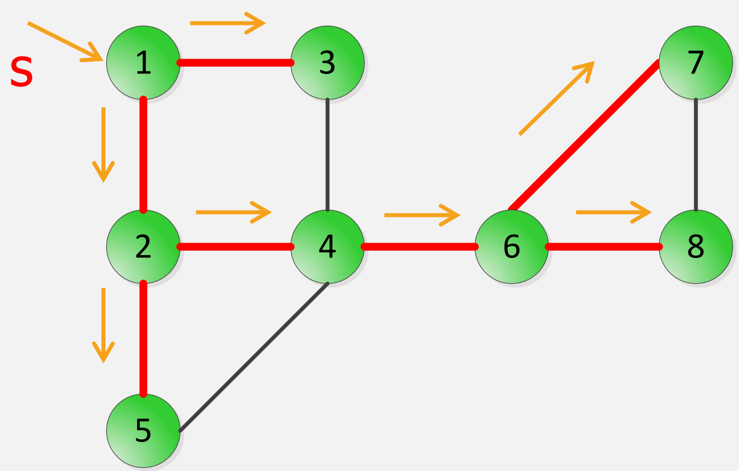


PESQUISA EM LARGURA

- BFS – Árvore de pesquisa
  - O algoritmo produz uma árvore de pesquisa em largura (breadth-first tree) com raiz em S e que contém todos os vértices alcançáveis a partir de S



Breath-first tree



PESQUISA EM LARGURA

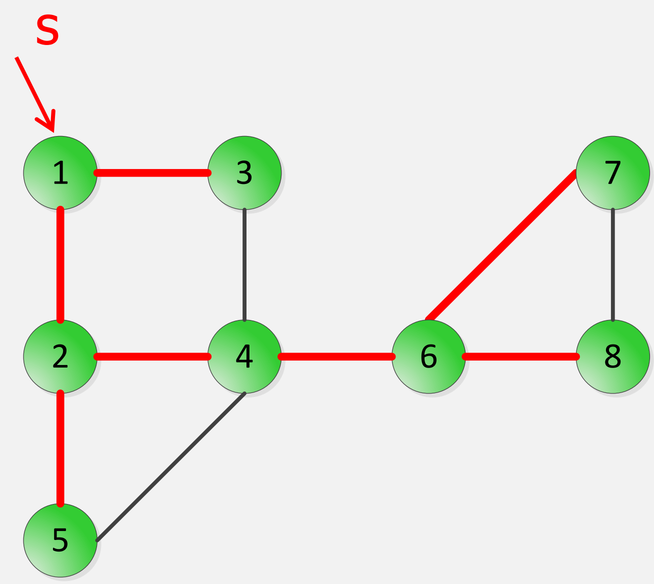
- BFS – Implementação
  - Dado um grafo G(V, E) e um nó inicial de varrimento s (source), têm que ser associados os seguintes atributos adicionais a cada vértice u:
    - u.color – indica se o vértice já foi visitado
      - branco – cor inicial dos vértices, ou seja, ainda não visitados
      - cinzento – quando o vértice é visitado
      - preto – todos os seus vértices adjacentes já foram visitados
    - u.p – predecessor de u (no sentido de s para u)

      se u não tiver predecessor, u.p = NULL
    - u.d – distância de s a u (número de arestas percorridas entre s e u)

PESQUISA EM LARGURA

```
BFS(G, s)
    for each vertex u ϵ G.V – {s}
        u.color = WHITE
        u.d = ∞
        u.p = NULL
    s.color = GRAY
    s.d = 0
    s.p = NULL
    Q = Ø
    Enqueue(Q, s)
    while Q ≠ Ø
        u = Dequeue(Q)
        for each v ϵ G.Adj[u]
            if v.color = WHITE
                v.color = GRAY
                v.d = u.d + 1
                v.p = u
                Enqueue(Q, v)
        u.color = BLACK
```

BFS – Algoritmo



Ordem do varrimento:

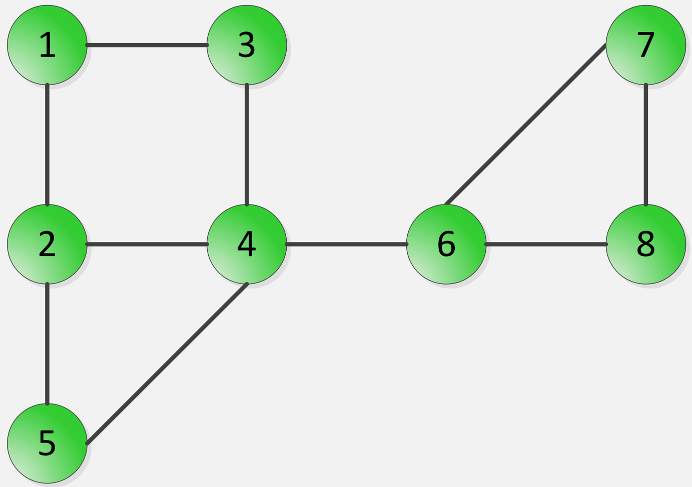
1 2 3 4 5 6 7 8

PESQUISA EM LARGURA

- BFS – Análise
  - Dado um grafo $G = (V, E)$ a sua inicialização consome tempo $\Theta(V)$
  - Quando um vértice é visitado é inserido na Queue no máximo uma vez
    - Tempo de execução de *enqueue* e *dequeue* - $\Theta(1)$
    - Tempo total das operações sobre a fila - $O\ (V)$
  - Numa lista de adjacência cada vértice é percorrido apenas uma vez
    - A soma do comprimento da listas é $\Theta(E)$
    - O tempo total a percorrer as listas é $O\ (E)$
  - Complexidade do BFS – $O\ (V + E)$

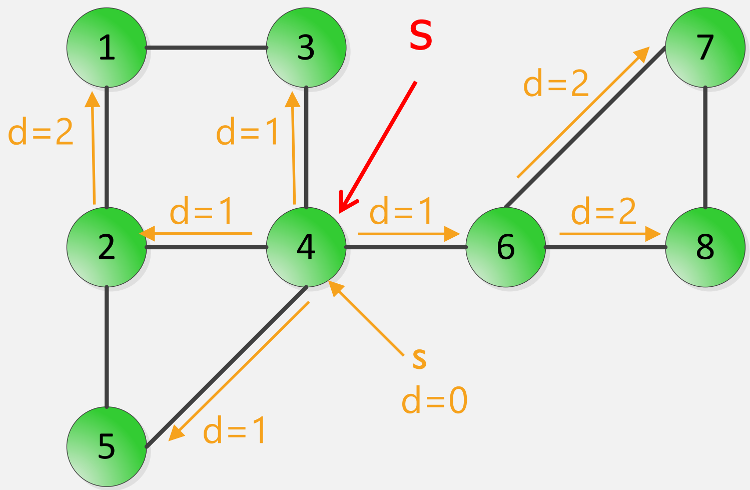
EXEMPLO DE PESQUISA EM LARGURA

- Vizinhos à distância N
  - Encontrar o número de vizinhos a uma distância N de um dado vértice
  - Exemplo: Quantos vértices estão à distância 2 do vértice (4)?



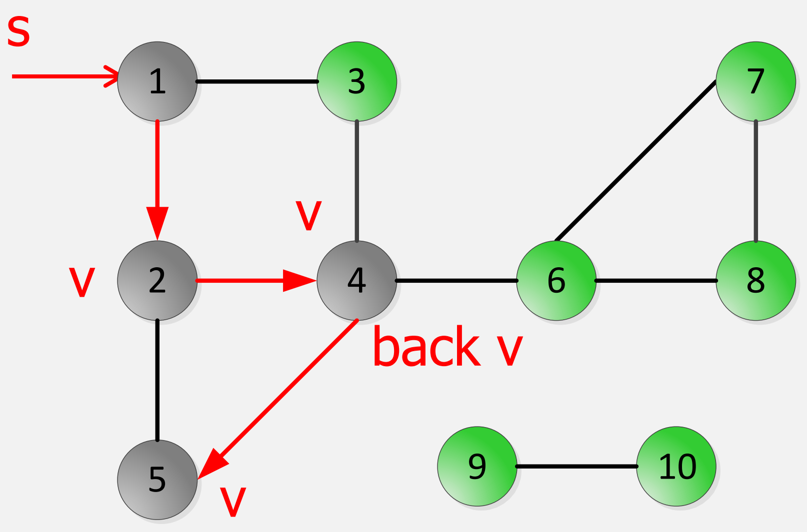
VIZINHOS À DISTÂNCIA N

- É feita uma pesquisa em largura a partir do vértice (4), considerando que existindo mais alternativas, os vértices são visitados por ordem crescente
- Resultado: Existem 3 vértices à distância 2 de (4) - (1), (7) e (8)



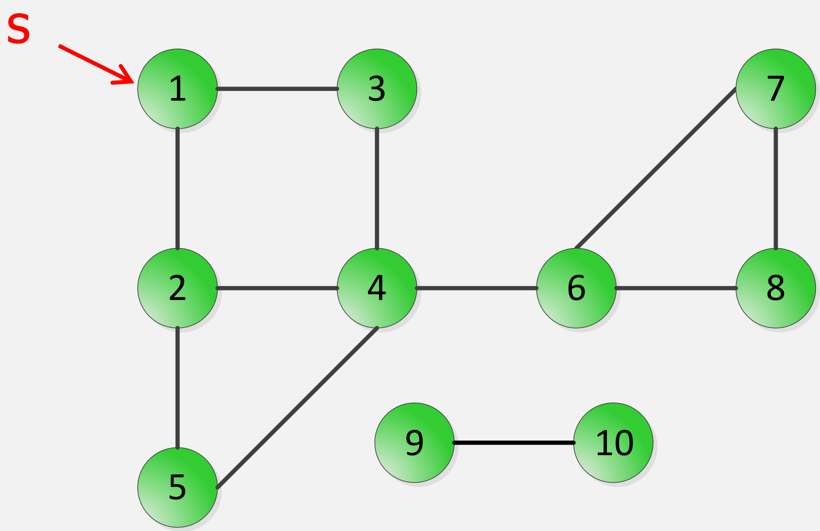
PESQUISA EM PROFUNDIDADE

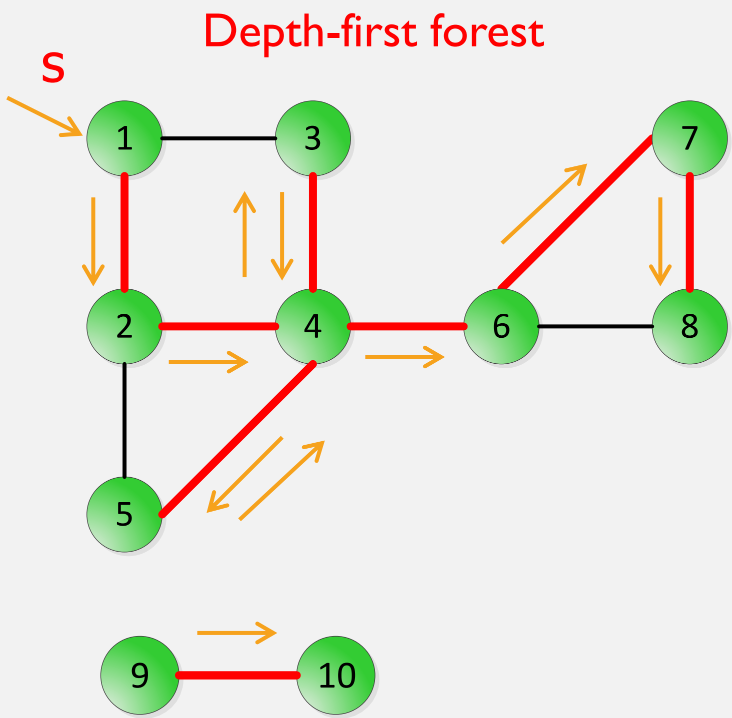
- DFS (Depth-First-Search)
  - Dado um grafo G = (V, E), iniciando em S, as arestas são exploradas sucessivamente a partir do vértice v mais recentemente visitado e que ainda tenha arestas por explorar
  - No caso de não existirem, a pesquisa retrocede continuando no próximo vértice por visitar



- O algoritmo utiliza uma pilha (*stack*), suportada pela implementação recursiva

- DFS – Árvore de pesquisa
  - O algoritmo pode produzir várias árvores de pesquisa (depth-first trees), se nem todos os vértices são atingíveis a partir de S. Nesse caso é formada uma floresta (depth-first forest)





PESQUISA EM PROFUNDIDADE

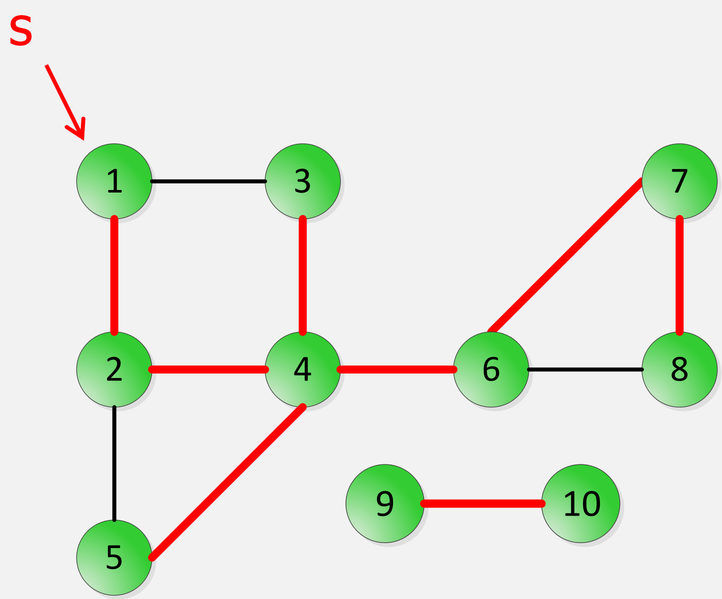
- DFS – Implementação
  - A pesquisa em profundidade reutiliza os atributos da pesquisa em largura e usa atributos adicionais para o *timestamp* dos vértices. Cada vértice u tem dois *timestamps*
    - u.d – (reutilizado com outro significado) regista quando o vértice u é visitado pela primeira vez (quando fica cinzento)
    - u.f – regista quando todos os vértices adjacente de u já foram examinados (quando fica a preto)
    - Estes *timestamps* providenciam informação importante relativa à estrutura do grafo, ajudando a compreender a pesquisa em profundidade

PESQUISA EM PROFUNDIDADE

```
DFS(G)
    for each vertex u ϵ G.V
        u.color = WHITE
        u.p = NULL
    time = 0
    for each vertex u ϵ G.V
        if (u.color == WHITE)
            DFS-Visit(G, u)
```

```
DFS-Visit(G, u)
    time = time + 1
    u.d = time
    u.color = GRAY
    for each vertex v ϵ G.adj[u]
        if v.color == WHITE
            v.p = u
            DFS-Visit(G, v)
    u.color = BLACK
    time = time + 1
    u.f = time
```

- DFS – Algoritmo
  - O algoritmo pode ser recursivo, usando assim uma pilha (*stack*) implícita



Ordem do varrimento:

1 2 4 3 5 6 7 8 9 10

PESQUISA EM PROFUNDIDADE

- DFS – Análise

  - Dado um grafo G = (V, E), a função *DFS-Visit* é executada exactamente uma vez para cada vértice v ϵ V
    - Tempo de execução de DFS-Visit - Θ(V)

  - Durante a execução de *DFS-Visit* , o ciclo de repetição *for*, é executado para cada vértice tantas vezes quantas a dimensão da sua lista de adjacência
    - Tempo total do ciclo de repetição de DFS-Visit - |Adj(V)|

  - Tempo total de execução do DFS – Θ(V + E)

EXEMPLO DE PESQUISA EM PROFUNDIDADE

- Ordenação topológica
  - A ordenação topológica de um grafo dirigido acíclo G = (V, E), é uma ordenação de todos os vértices tal que se G contém uma aresta (u, v), então u aparece antes de v na ordenação
  - Se o grafo contém algum circuito, então a ordenação topológica não é possível

```
Topological-Sort(G)
    Call DFS(G) to compute finishing times v.f for each vertex v
    As each vertex is finished, insert it onto the front of a linked list
    Return the linked list of vertices
```

ORDENAÇÃO TOPOLÓGICA

- O professor Bumstead ordena a sua roupa topologicamente antes de se vestir, construindo primeiro um grafo dirigido com as precedências das peças de roupa