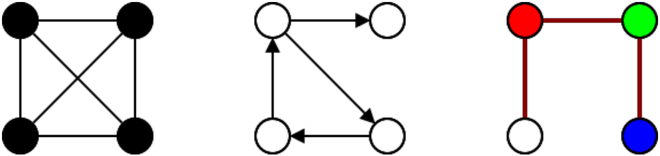
Grafos - Introdução

Pedro Ribeiro

DCC/FCUP

2016/2017

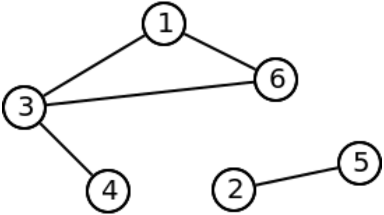


Conceito

Definição de Grafo

Formalmente, um **grafo** é:

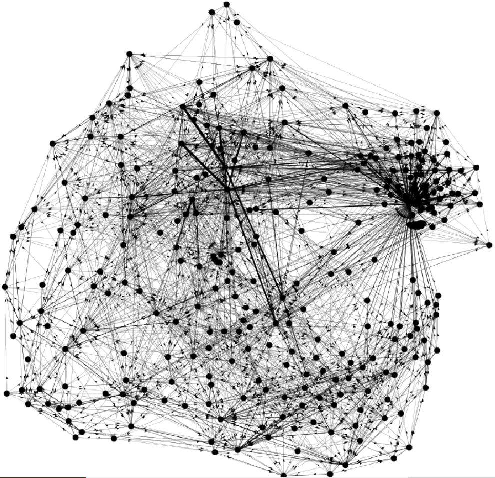
- Um conjunto de **nós**/**vértices** (**V**).
- Um conjunto de **ligações**/**arestas**/**arcos** (**E**), que consistem em pares de vértices



- $V = \{1, 2, 3, 4, 5, 6\}$
- $E = \{(1, 6), (1, 3), (3, 6), (3, 4), (2, 5)\}$

Para que servem os grafos?

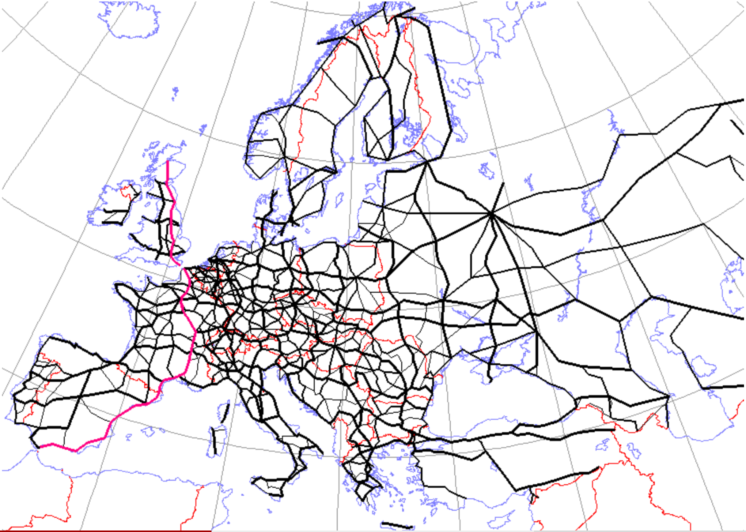
- Os grafos são **úbiquos** na Ciência de Computadores e estão presentes, implicita ou explicitamente, em muitos algoritmos.
- Podem ser usados para representar uma **multiplicidade** de coisas.



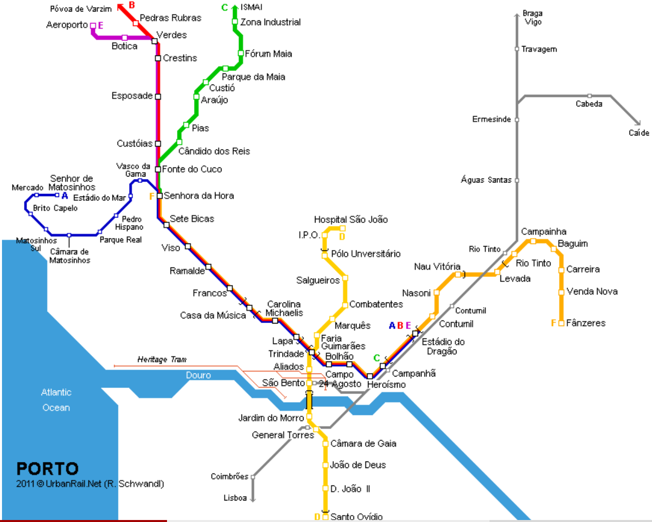
Exemplos de Grafos

Redes com existência física

- Redes de estradas



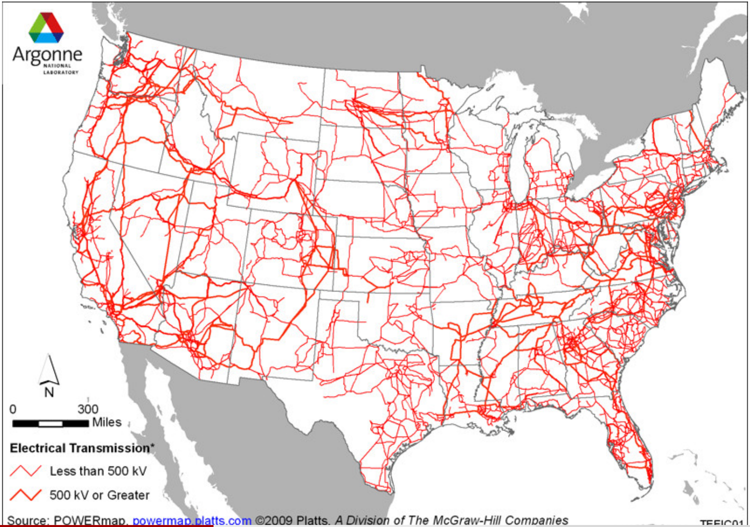
- Redes de transportes públicos (ex: metro, comboio)



Exemplos de Grafos

Redes com existência física

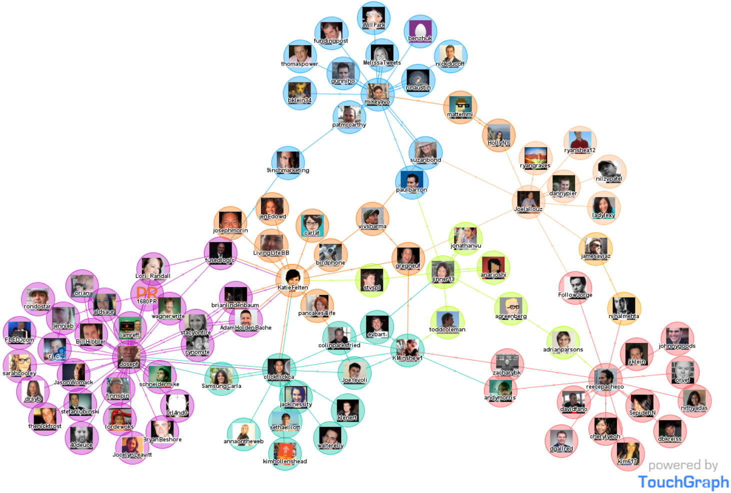
- Redes de energia eléctrica



Exemplos de Grafos
Redes com existência física

- Redes de computadores

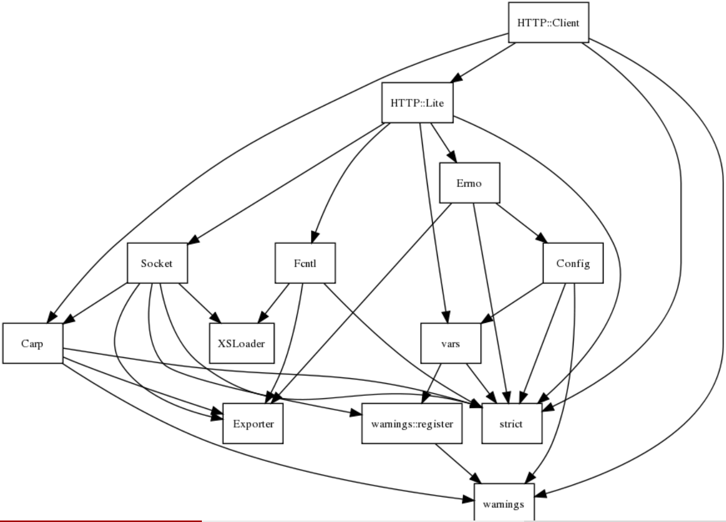
- Facebook (outros ex: Twitter, e-emails, co-autoria de artigos, ...)



Exemplos de Grafos

Redes de Software

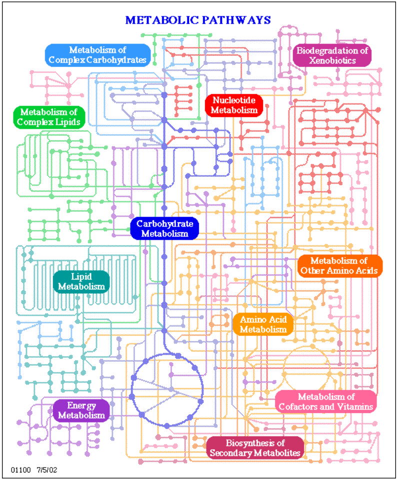
- Depêndencia entre módulos (outros: estado, fluxo de informação, ...)



Exemplos de Grafos

Redes Biológicas

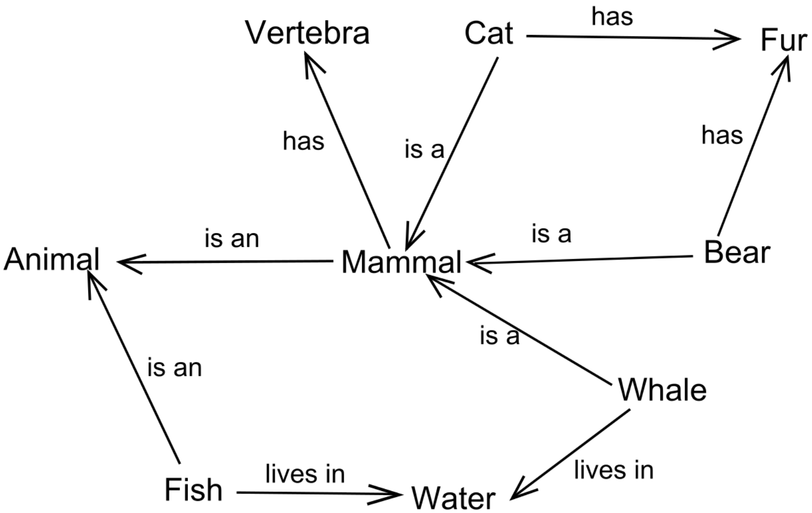
- Rede metabólica (outros exemplos: proteínas, transcrição, cerebrais, cadeias alimentares, redes filogenéticas, ...)



Exemplos de Grafos

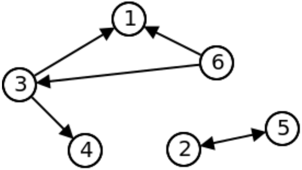
Outros Grafos

- Rede semântica (outros exemplos: links entre páginas, ...)

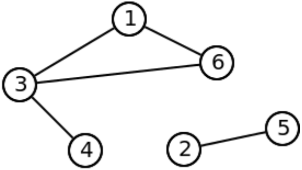


Terminologia

- Grafo **dirigido**/**direcionado**/**digrafo** - cada ligação tem um nó de partida (**origem**) e um nó de chegada **fim** (ordem interessa!). Normalmente nos desenhos usam-se setas para indicar a direção
- Grafo **não dirigido** - não existe partida e chegada, apenas uma ligação



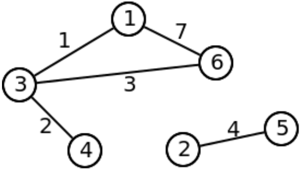
Grafo Dirigido



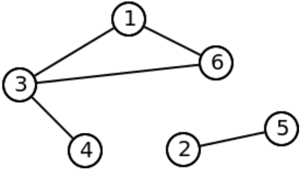
Grafo Não Dirigido

Terminologia

- Grafo **pesado** - a cada ligação está associado um valor (pode ser uma distância, um custo, ...)
- Grafo **não pesado** - não existem valores associados a cada arco



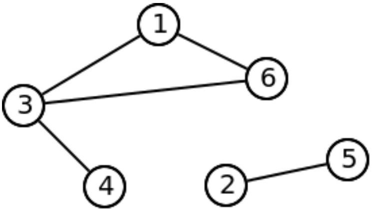
Grafo Pesado



Grafo Não Pesado

Terminologia

- **Grau** de um nó - número de ligações desse nó
- Em grafos dirigidos pode distinguir-se entre **grau de entrada** e **grau de saída**



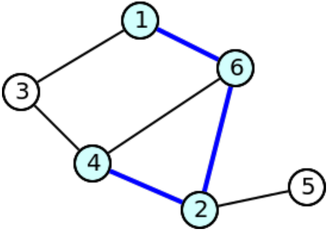
1 tem grau 2
2 tem grau 1
3 tem grau 3
4 tem grau 1
5 tem grau 1
6 tem grau 2

Terminologia

- Nó **adjacente/vizinho**: dois nós são adjacentes se tiverem uma ligação entre si
- **Grafo trivial**: grafo sem arestas e com um único nó
- **Laço** ou **lacete** (*self-loop*): ligação de um nó a si próprio
- **Grafo simples:** nó sem laços e sem ligações repetidas em E
  (em DAA vamos usar (quase) sempre grafos simples)
- **Multigrafo:** grafo não simples (com laços e/ou ligações) repetidas)
- **Grafo denso:** com muitas ligações quando comparadas com o máximo possível de ligações - $|E|$ da ordem de $O(|V|^2)$
- **Grafo esparso:** com poucas ligações quando comparadas com o máximo possivel de ligações - $|E|$ de ordem inferior a $O(|V|^2)$

Terminologia

- **Caminho**: sequência alternada de nós e arestas, de tal modo que dois nós sucessivos são ligados por uma aresta. Tipicamente em grafos simples indicam-se só os nós para definir um caminho.

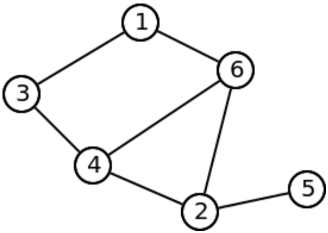


$$1 \rightarrow 6 \rightarrow 2 \rightarrow 4$$

- **Ciclo**: caminho que começa e termina no mesmo nó (ex: para o grafo de cima, $1 \rightarrow 6 \rightarrow 4 \rightarrow 3 \rightarrow 1$ é um ciclo)
- **Grafo acíclico**: grafo sem ciclos

Terminologia

- **Tamanho** de um caminho: número de arestas num caminho
- **Custo** de caminho: se for um grafo pesado podemos falar no custo, que é a soma dos pesos das arestas
- **Distância**: tamanho/custo do menor caminho entre dois nós
- **Diâmetro** de um grafo: distância máxima entre dois nós de um grafo



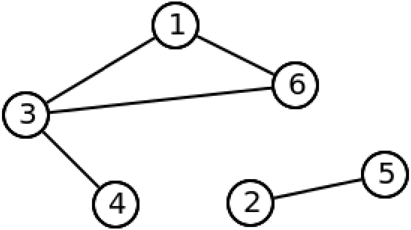
Diâmetro = 3

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

Distâncias entre nós

Terminologia

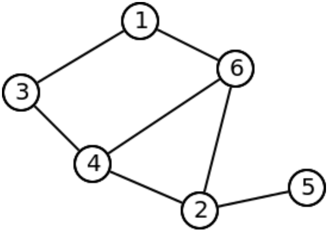
- **Componente conexa**: Subconjunto de nós onde existe pelo menos um caminho entre cada um deles
- **Grafo conexo**: Grafo com apenas uma componente conexa (existe caminho de todos para todos)



Grafo com duas componentes conexas: $\{1, 3, 4, 6\}$ e $\{2, 5\}$

Terminologia

- **Subgrafo**: subconjunto de nós e arestas entre eles
- Grafo **completo**: existem ligações entre todos os pares de nós
- **Clique**: subgrafo que é completo
- **Triângulo**: clique de 3 nós

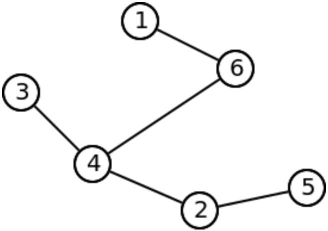


Exemplo de subgrafos: $\{1, 3\}$, $\{1, 6, 2\}$, $\{2, 4, 5, 6\}$, etc
Exemplo de clique: $\{2, 4, 6\}$ (é um triângulo)

Terminologia

- **Árvore**: grafo simples, conexo e acíclico
  (se tem n nós, então terá $n - 1$ arestas)

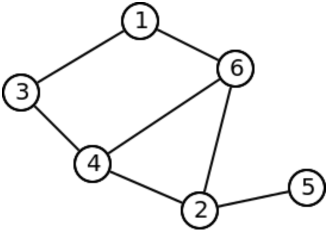
- **Floresta**: conjunto de múltiplas árvores disconexas



Representação de Grafos

Como representar um grafo?

- **Matriz de Adjacências**: matriz de $|V| \times |V|$ onde a entrada (i, j) indica se existe uma ligação entre o nó i e j (se for um grafo pesado podemos indicar o peso)
- **Lista de Adjacências**: cada nó guarda uma lista contendo os seus vizinhos (se for grafo pesado temos de guardar pares (destino,peso))



	1	**2**	**3**	**4**	**5**	**6**
1			X			X
2				X	X	X
3	X			X		
4		X	X			X
5		X				
6	X	X		X		

Matriz de Adjacências

1: 3, 6
2: 4, 5, 6
3: 1, 4
4: 2, 3, 6
5: 2
6: 1, 2, 4

Lista de Adjacências

Representação de Grafos

Algumas Vantagens/Desvantagens:

- **Matriz de Adjacências**:
  - Muito simples de implementar
  - Rápida para ver se existe ligação entre dois nós - $\mathcal{O}(\mathbf{1})$
  - Lenta para percorrer nós adjacentes - $\mathcal{O}(|\mathbf{V}|)$
  - Elevado desperdício de memória (em grafos esparsos) - $\mathcal{O}(|\mathbf{V}|^\mathbf{2})$
  - Grafo pesado implica apenas armazenar peso na matriz
  - Adicionar/remover ligações é só mudar célula da matriz - $\mathcal{O}(\mathbf{1})$

- **Lista de Adjacências**:
  - Lenta para ver se existe ligação entre nós u e v - $\mathcal{O}(\mathbf{grau(u)})$
  - Rápida para percorrer nós adjacentes - $\mathcal{O}(\mathbf{grau(u)})$
  - Memória bem aproveitada - $\mathcal{O}(|\mathbf{V}| + |\mathbf{E}|)$
  - Grafo pesado implica adicionar um campo à lista
  - Remover ligação (u, v) implica percorrer a lista - $\mathcal{O}(\mathbf{grau(u)})$

    Nota: podemos usar por exemplo BSTs (set/map) para melhorar eficiência da pesquisa/remoção para $\mathcal{O}(\log grau(u))$

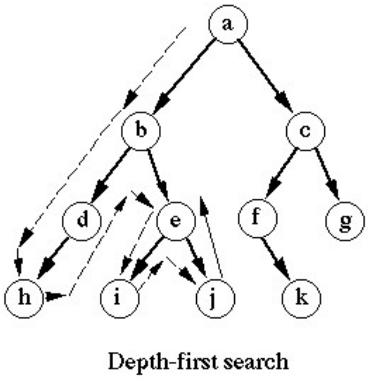
Algoritmos

Alguns dos algoritmos que vamos dar em DAA

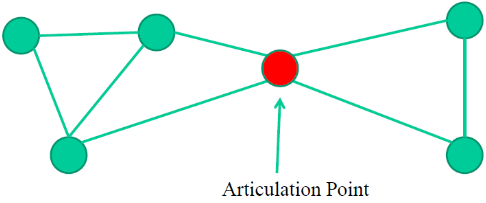
Pesquisas de um grafo:

- Em **profundidade** - DFS ($\mathcal{O}(|V| + |E|)$ com lista de adjacências)
  Alguns exemplos de aplicação:
  - Descobrir componentes conexos / *Flood-Fill*
  - Descobrir ciclos
  - Ordenação topológica
  - Descobrir pontos de articulação e/ou pontes
  - Descobrir componentes fortemente conexos
  - Saber se um grafo é bipartido
  - Pesquisa exaustiva de caminhos
- Em **largura** - BFS($\mathcal{O}(|V| + |E|)$ com lista de adjacências)
  Alguns exemplos de aplicação:
  - Quase todas as aplicações de DFS
    $+$
  - Descobrir caminho mínimo entre dois pontos (num grafo não pesado)

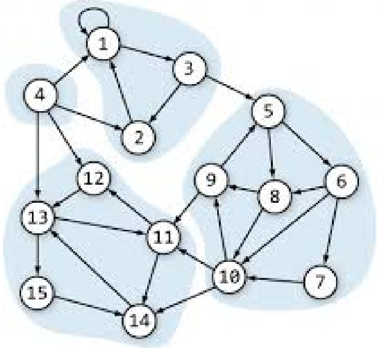
Pesquisas em profundidade e largura:



Ponto de articulação:



Componentes fortemente conexos:

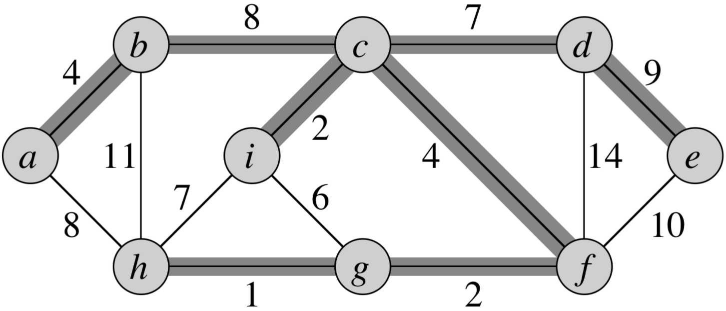


Árvores Mínimas de Suporte

Uma **árvore de suporte** é um subgrafo conexo que é uma árvore e que contém todos os vértices do grafo. Uma **árvore mínima de suporte** é uma árvore de suporte onde a soma dos pesos das arestas é mínima.

- Algoritmo de **Prim**
  - Algoritmo greedy que adiciona um nó de cada vez
  - Complexidade temporal: $\mathcal{O}(|E| \log |V|)$ com uma fila de prioridade
- Algoritmo de **Kruskal**
  - Algoritmo greedy que adiciona uma aresta de cada vez
  - Complexidade temporal: $\mathcal{O}(|E| \log |E|)$ com conjuntos disjuntos

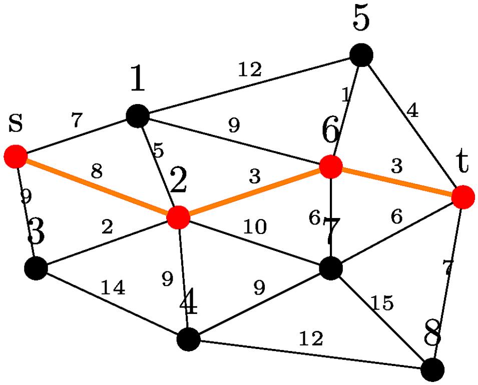
Árvore Mínima de Suporte:



Caminhos mínimos

- Algoritmo de **Dijkstra** - Distância de um nó para todos os outros (não funciona em grafos com pesos negativos)
  - Um "mix" de greedy com programação dinâmica
  - Complexidade temporal: $\mathcal{O}(|E| log |V|)$ com uma fila de prioridade
- Algoritmo de **Bellman-Ford** - Distância de um nó para todos os outros (funciona em grafos com pesos negativos)
  - Parecido com Dijkstra mas sempre com "relaxamento" de todas as arestas
  - Complexidade temporal: $\mathcal{O}(|E| \times |V|)$ com conjuntos disjuntos
- Algoritmo de **Floyd-Warshall** - Distâncias entre todos os nós
  - Usa ideias de programação dinâmica
  - Complexidade temporal: $\mathcal{O}(|V|^3)$

Caminho mínimo entre dois pontos:



Fluxos máximos

Problema de optimização que envolve descobrir fluxo máximo (não excedendo capacidades das arestas) entre dois nós.

Exemplos de aplicações:

- *Bipartite matching*
- Mínima cobertura de caminhos
- Número de caminhos sem nós e/ou arestas comuns

O algoritmo que vamos dar:

- Algoritmo de **Edmonds-Karp**
  Uma implementação do algoritmo de **Ford-Fulkerson**
  - Ir descobrindo *augmenting paths* com sucessivas pesquisas em largura
  - Complexidade temporal: $\mathcal{O}(|V| \times |E|^2)$ ()

Algoritmos

Alguns dos algoritmos que vamos dar em DAA

Fluxo máximo:

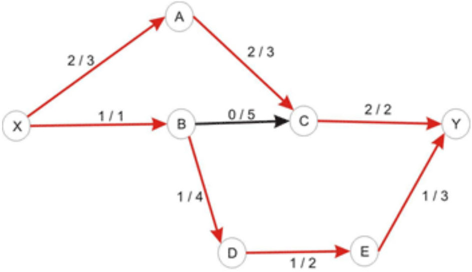


Figure 1a - Maximum Flow in a network

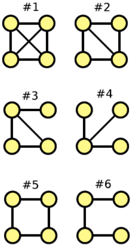
Exemplos de Grafos

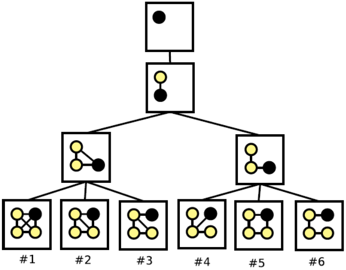
Alguns sites com datasets interessantes

- **Koblenz:** http://konect.uni-koblenz.de/
- **SNAP:** https://snap.stanford.edu/data/
- **Mark Newman:** http://www-personal.umich.edu/~mejn/netdata/
- **Pajek:** http://vlado.fmf.uni-lj.si/pub/networks/data/

Redes Complexas (Network Science / Graph Mining)

A minha área de investigação principal



Tese de Doutoramento (2011):
Efficient and Scalable Algorithms for Network Motifs Discovery

Publicações: http://www.dcc.fc.up.pt/~pribeiro/pubs_by_year.html