Teoria dos Grafos
Aula 5

Aula passada

- Grafos direcionados
- Busca em grafos direcionados
- Ordenação topológica

Aula de hoje

- Grafos com pesos
- Dijkstra
- Implementação
- Fila de prioridades e Heap
- Dijkstra (o próprio)

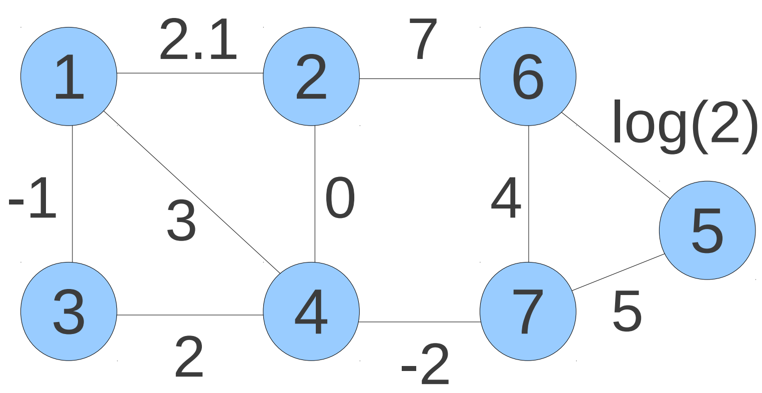
Relacionamentos de Peso

- Relacionamentos entre objetos nem sempre são idênticos (mesma intensidade)
- **Exemplos?**
- Amizade
  - mais ou menos amigo (Orktut)
- Distância física
  - perto ou longe
- Tempo de translado
  - mais ou menos tempo

Como representar tais relacionamentos?

Grafos com Pesos

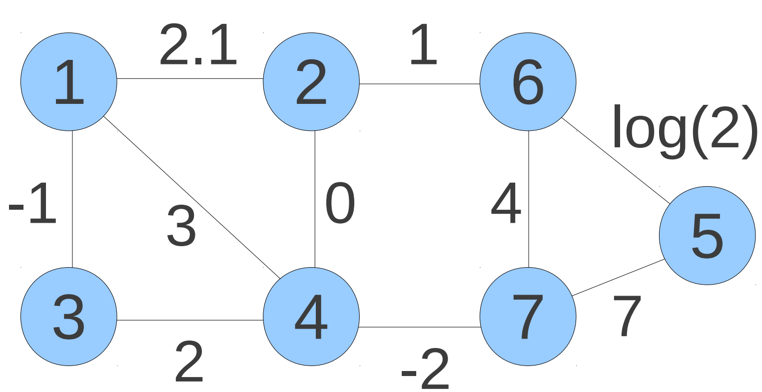
- Anotar arestas do grafo com “intensidade” do relacionamento
  - peso da aresta (*weight*)
  - função w(*e*) retorna peso da aresta *e*
  - Ex. $w : E \rightarrow \Re$
- Graficamente



w((2,6)) = 7
w((5,6)) = log(2)
w((2,4)) = 0

Distância com Pesos

- Comprimento de um caminho
  - **soma** dos pesos das arestas que definem o caminho
- Distância
  - comprimento do **menor** caminho simples de entre dois vértices
- Graficamente



Comprimento do caminho 1,2,6,7?
Comprimento do caminho 1,3,4,7?
Comprimento do caminho 1,4,7?
Distância entre 1 e 7?
Distância entre 3 e 5?

Grafos Direcionados com Peso

- Relacionamentos assimétricos com pesos (diferentes intensidades)
- Exemplo?
- Mesma idéia: anotar arestas do grafo com “intensidade” do relacionamento
  - peso da aresta (*weight*)
  - função w(*e*) retorna peso da aresta *e*
  - aresta direcionada, pesos potenciamente diferentes

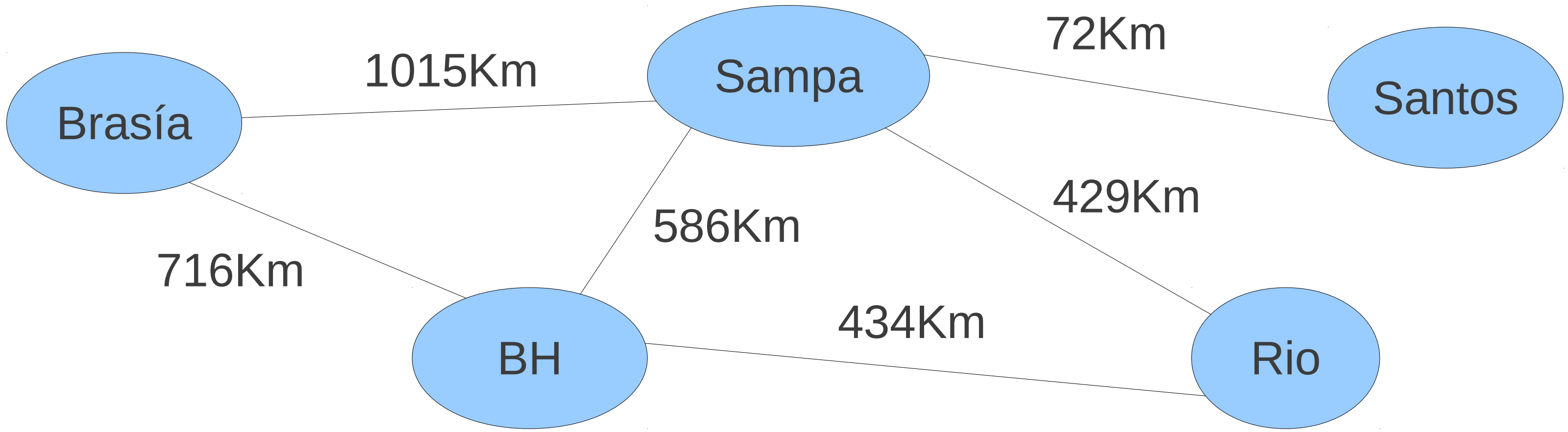
Viagem entre Cidades

- Cidades brasileiras
- Estradas entre cidades

- **Problema 1:** Como saber se duas cidades estão "conectadas" por estradas?
- **Problema 2:** Qual é o menor (melhor) caminho entre duas cidades?

Viagem entre Cidades

- Abstração via grafos com pesos



- **Problema 1:** Como sa… …stão “conec…

Resolvido!

- **Problema 2:** Qual é o menor (melhor) caminho entre duas cidades?

Distância em Grafos com Peso

- Calcular caminho mais curto entre cidades é calcular a distância em grafos com peso
  - pesos sempre maior que zero

- Dado G, com pesos
- Qual é a distância do vértice *s* ao *d*?

Como resolver este problema?

- Como resolvemos o problema sem pesos?
- Podemos adaptar algumas idéias?

Distância em Grafos com Peso

- **Idéia**
- Partindo de *s*, expandir os caminhos, incluindo vértices

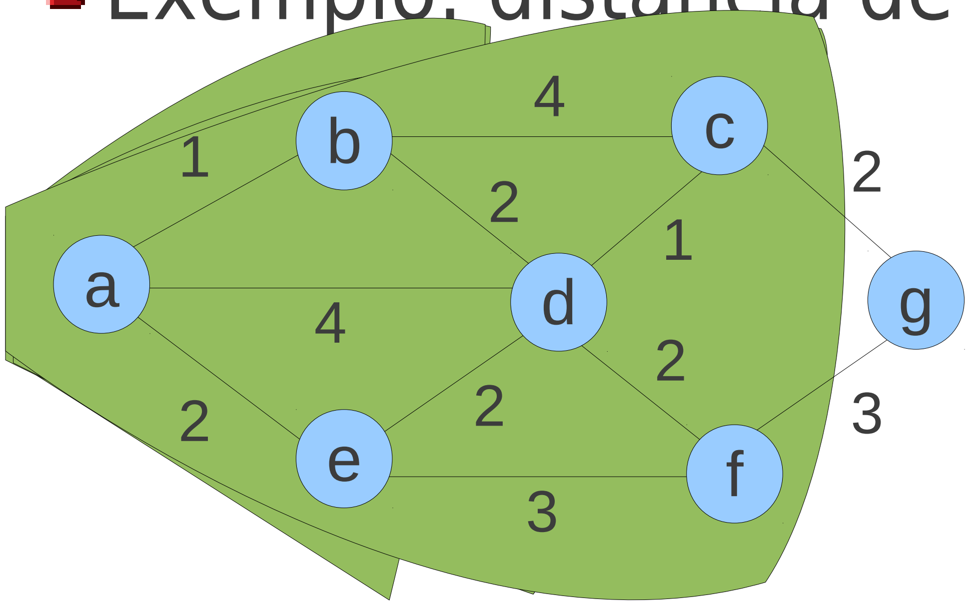
Mas em que ordem?

  - Na ordem em que caminhos mínimos sejam garantidos!

- Expandir caminhos mínimos até chegar em *d* de maneira gulosa!

Distância em Grafos com Peso

- Exemplo: distância de *a* à *g*?



Algoritmo de Dijkstra!

- Começar em a, expandir
- Qual próximo vértice?
- Qual vértice nos dá caminho mínimo garantido?

Algoritmo de Dijkstra

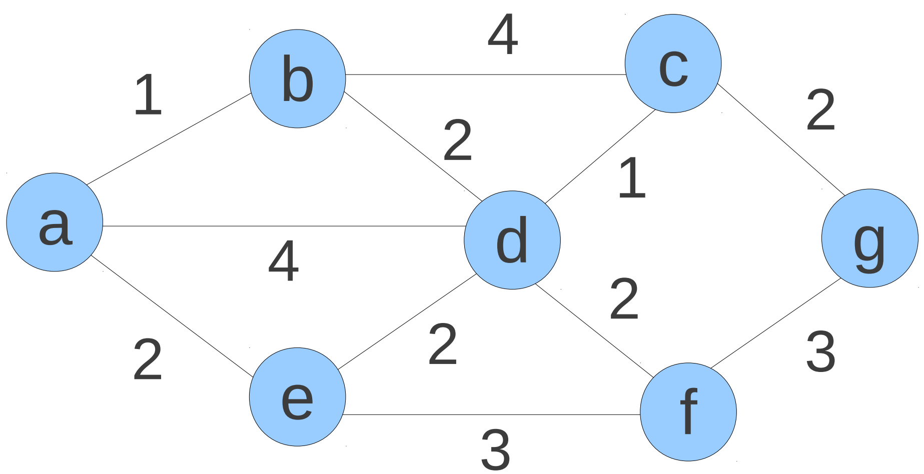
- Como tornar a idéia em algoritmo?
  - adicionar o vértice para o qual temos o menor caminho
- **Idéias:**
  - Manter dois conjuntos de vértices
  - Manter comprimento do menor caminho conhecido até o momento para cada vértice
  - Adicionar o vértice de menor caminho
  - Atualizar distâncias

Algoritmo de Dijkstra

```
Dijkstra(G, s)
Para cada vértice v
   dist[v] = infinito
Define conjunto S = 0 // vazio
dist[s] = 0
Enquanto S != V
   Selecione u em V-S, tal que dist[u] é mínima
   Adicione u em S
   Para cada vizinho v de u faça
      Se dist[v] > dist[u] + w((u,v)) então
         dist[v] = dist[u] + w((u,v))
```

- Como o algoritmo executa?

Executando o Algoritmo



- Manter tabela com passos e distâncias

Conjunto S	d(a)	d(b)	d(c)	d(d)	d(e)	d(f)	d(g)
{}	0	inf	inf	inf	inf	inf	inf
{a}	-	1	inf	4	2	inf	inf
{a,b}		-	5	3	2	inf	inf
{a,b,e}			5	3	-	5	inf
{a,b,e,d}			4	-		5	inf
{a,b,e,d,c}			-			5	6
{a,b,e,d,c,f}						-	6
{a,b,e,d,c,f,g}							-

Analizando o Algoritmo

- Provar que algoritmo sempre produz resultado correto – caminho mínimo entre dois vértices

Teorema:

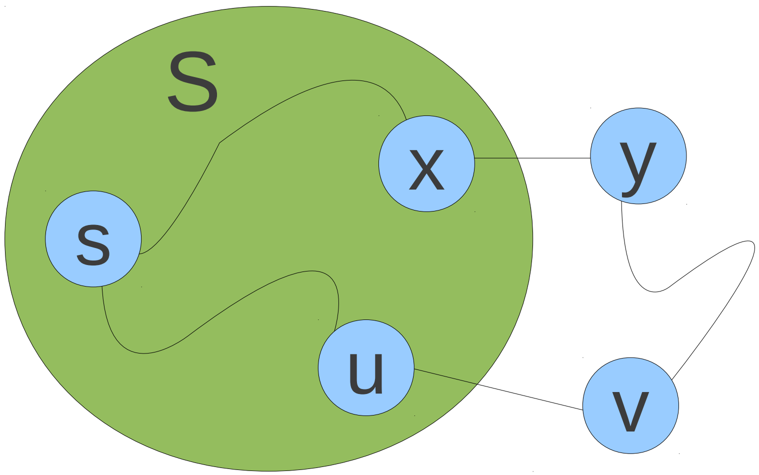
- Considere um vértice *u* pertencente ao conjunto *S* em qualquer ponto do algoritmo. Temos que “dist[u]” é a distância entre *s* e *u*.

Prova

- Prova por indução no tamanho de S
- Caso base: |S| = 1
  - S = {s}, dist[s] = 0, devido inicialização
- Hipótese: |S| = k
  - Para |S| = k, assuma que dist[u] é igual a distância entre *s* e *u*
- Caso geral: |S| = k + 1
  - k+1 adiciona *v* à S, dando origem a P_v
  - seja (u,v) última aresta no caminho P_v
  - Pela hipótese, P_u é caminho mínimo s-u

Prova

- Considere outro caminho s-v, P, que não passa por u
- Precisamos provar que P é maior (ou igual) a P_v
- P passa pela aresta (x,y), com x em S e y fora de S (para algum x e y qualquer)
- Situação:



- No passo k+1, algortimo escolhe *v* para adicionar a S (e não *y*)
- Então, caminho s—y é maior ou igual a P_v
- Como dist(y,v) >=0, caminho P será maior ou igual a P_v
- Logo, P_v é caminho mínimo e dist[v] será distância até v

Complexidade

- Qual é a complexidade do algoritmo?

```
Dijkstra(G, s)
Para cada vértice v
   dist[v] = infinito
Define conjunto S = 0 // vazio
dist[s] = 0
Enquanto S != V
   Selecione u em V-S, tal que dist[u] é mínima
   Adicione u em S
   Para cada vizinho v de u faça
      Se dist[v] > dist[u] + w((u,v)) então
         dist[v] = dist[u] + w((u,v))
```

- Depende do tempo para selecionar u tal que dist[u] seja mínima!

Complexidade

- Algoritmo simples
  - Percorre vértices e encontra dist[u] mínimo
  - Complexidade? ← $O(n^2)$

Outra idéia?

- Fila de prioridades – muito usada em grafos!
  - Chave: distâncias
  - Extrair o mínimo (vértice com menor distância)
  - Atualizar chaves (distâncias)

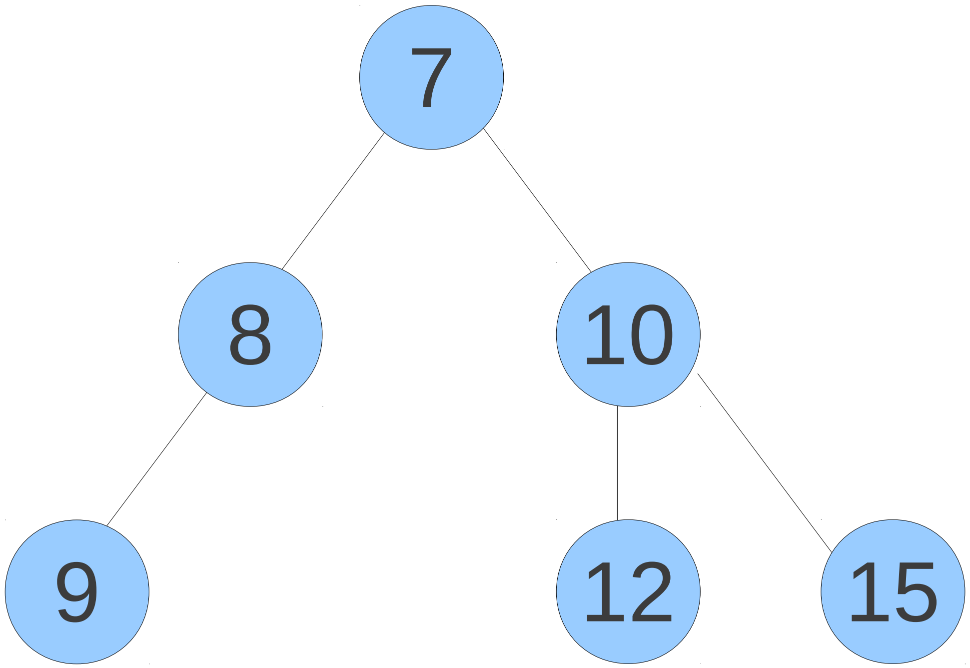
Fila de Prioridade

- Estrutura de dados poderosa *(priority queue)*
- Mantém um conjunto de elementos S
- Cada elemento possui uma chave (número de prioridade)
- Permite inserir, remover e modificar elementos de S através de sua chave
- Permite remover menor chave (maior prioridade)
- Complexidade para inserir ou modificar elemento: O(log n)

Heap

- Como implementar Fila de Prioridade?
- **Heap!**
- O que é um heap?
- Estrutura de dados em árvore
- Armazena conjunto de elementos, todos associados a uma chave
- Chave dos elementos define árvore
- *Heap order*: chave(u) <= chave (v), quando u é pai de v

Heap - Exemplo



- Chaves podem inseridas, removidas ou mudar de valor
- Heap precisa ser reajustado – *heapify*
  - custo O(log n)
- Manter árvore “balanceada”

Complexidade

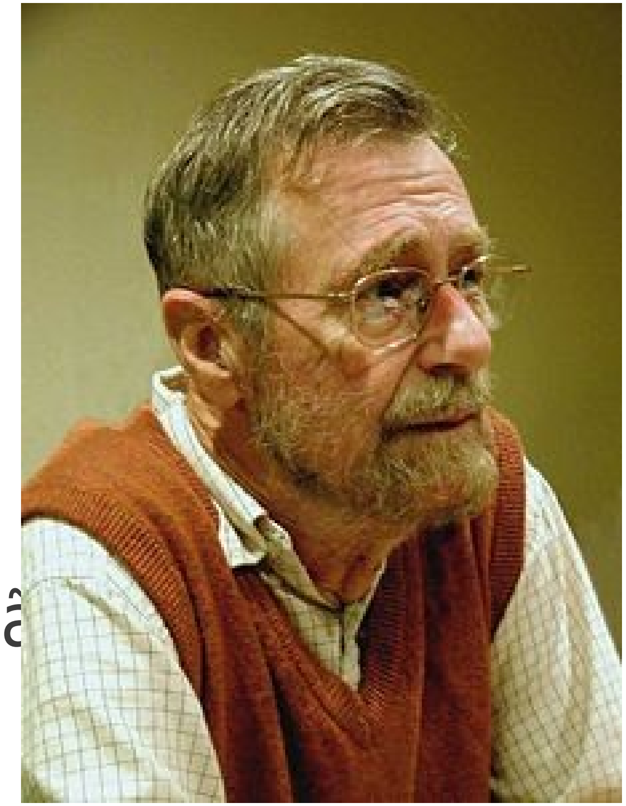
- Usando um heap?

```
Dijkstra(G, s)
Para cada vértice v
    dist[v] = infinito
Define conjunto S = 0 // vazio
dist[s] = 0
Enquanto S != V
    Selecione u em V-S, tal que dist[u] é mínima
    Adicione u em S
    Para cada vizinho v de u faça
        Se dist[v] > dist[u] + w((u,v)) então
            dist[v] = dist[u] + w((u,v))
```

- Cada operação no heap: $O(\log n)$
  - n operações de remoção
  - m operações de atualização
- Complexidade: $O((n+m)\log n) = O(m \log n)$

Dijkstra, o Próprio

- Edsger Wybe Dijkstra
- Professor e pesquisador na área de Ciência da Computação
- Recebeu Turing Award 1972
  - mais renomado prêmio da Computaçã
- Contribuições fundamentais em ling. de programação e verificação formal
- Algoritmo de Dijkstra utilizado em vários sistemas (redes, GPS, etc)



11/5/1930 – 6/8/2002