Grafos – Aula 14

Roteiro

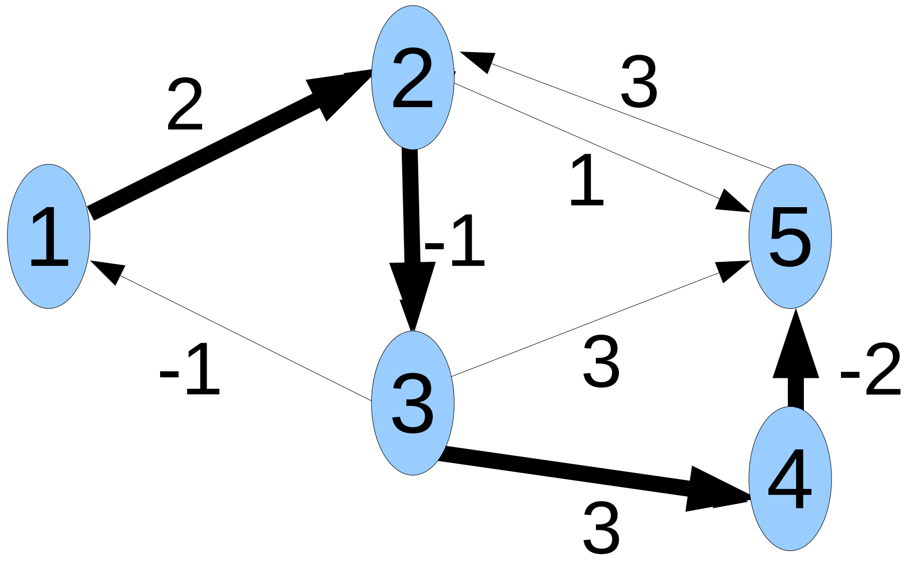
- Caminho mínimo com pesos negativos
- Programação dinâmica
- Algoritmo de Bellman-Ford
- Melhorias

Aula passada

- Grafos com pesos negativos
- Caminho mais curto entre todos os pares
- Programação dinâmica
- Algortimo de Floyd-Warshall
- Melhorias

Caminho Mínimo

- Dado grafo direcionado G, com pesos negativos
- **Problema**: Encontrar caminho mínimo e distância de todos os vértices para um vértice destino *t* (fixado)



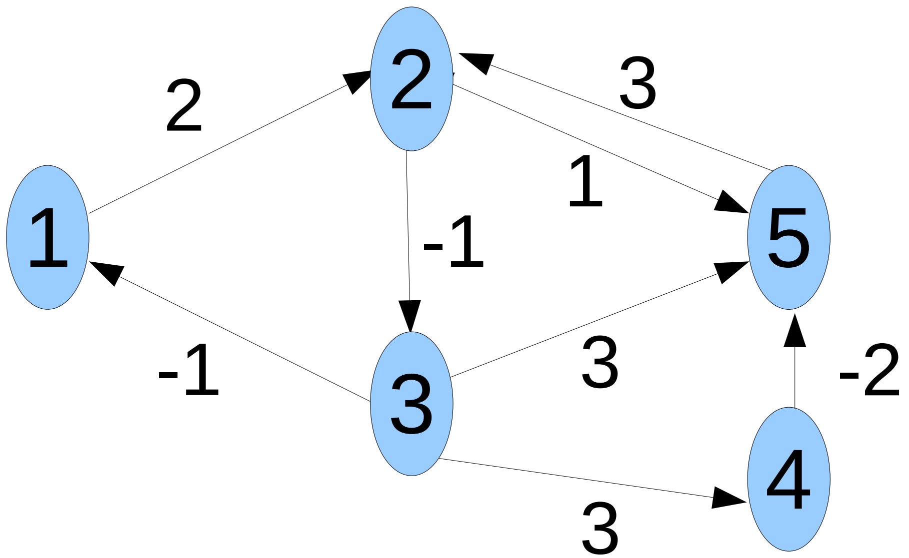
- Ex. caminho mais curto para t = 5
- Árvore geradora com caminhos mínimos

Programação Dinâmica

- Considere o caminho mínimo P entre v e t
  - $P = (v, v_1, v_2, \ldots, t)$
- Decompor o problema em subproblemas
  - encontrar P através de soluções ótimas para subproblemas menores
- **Ideia:** número de arestas de P
  - P pode ter 1, 2, ..., n-1 arestas
- Usar número de arestas para decompor o problema
  - caminho mínimo passando por até i arestas
  - qualquer i arestas

Função de Recursão

- OPT(i, v) : custo do caminho mínimo P entre v e t usando no máximo i arestas
- Ex. t = 5
  - OPT(1, 3) = 3
  - OPT(2, 3) = 1
  - OPT(3, 3) = 1
  - OPT(1, 2) = 1
  - OPT(2, 2) = 1
  - OPT(3, 2) = 0
  - OPT(1, 1) = infinito
  - OPT(2, 1) = 3



Analisando Solução Ótima

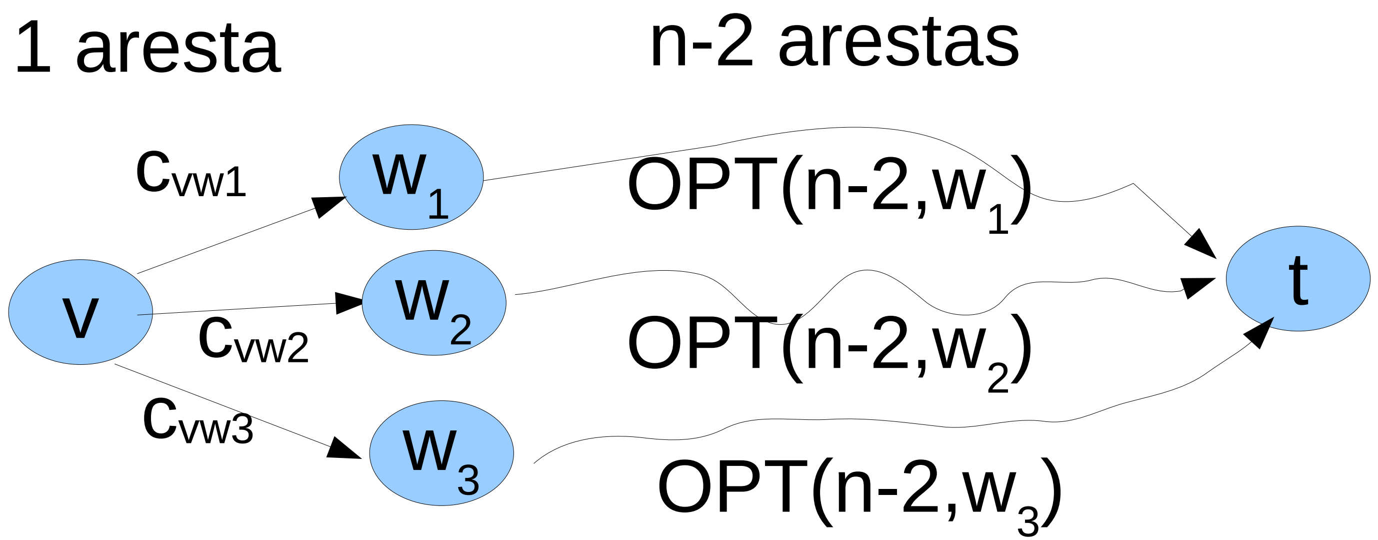
- Maior comprimento possível (em arestas) do caminho mínimo P entre *v* e *t*?
  - n-1 arestas (caminho simples, pois não temos ciclo negativos)
- O que podemos dizer sobre o caminho mais curto P entre v e t?
- 1) Ou possui exatamente n-1 arestas
- 2) Ou possui menos arestas

Analisando Solução Ótima

- Se P possui menos do que n-1 arestas
  - Custo da solução ótima com n-2 arestas será o mesmo
  - $OPT(n-1, v) = OPT(n-2, v)$

- Se P possui exatamente n-1 arestas
  - P passa por algum vizinho de v
  - caminho do vizinho até t tem exatamente n-2 arestas
  - escolher vizinho w com menor caminho somando o peso da aresta (v, w), denotado por c_{vw}
  - $OPT(n-1, v) = \min_w (OPT(n-2, w) + c_{vw})$

Graficamente



- Por qual vizinho de v o caminho mínimo P irá passar?
  - o de menor custo, considerando o peso da aresta com o vizinho
  - $\min_w$ (OPT(n-2, w) + c_{vw}), onde w pertence aos vizinhos de saída de v

Generalizando

- Supor P caminho ótimo de v para t usando i arestas ou menos
- Podemos decompor P em dois casos
  - P possuir exatamente i arestas
  - P possuir menos de i arestas
- Mesmo raciocínio de antes

peso da aresta (v, w)

- $OPT(i, v) = \min(\, OPT(i-1, v),\ \min_w (OPT(i-1, w) + c_{vw})\,)$

w pertence aos vizinhos de saída e *v*

Algoritmo de Bellman-Ford

- Algoritmo iterativo para calcular OPT(n-1, *)

```
Bellman_Ford(G, t)
  Array M[0,...,n-1 ; 1,...,n]
  M[0, v] = oo para todo v
  M[0, t] = 0

  For i = 1, ..., n-1
    For v = 1, ..., n
      M[i, v] = M[i-1, v]
      Para cada vizinho w de v
         M[i, v] = min (M[i, v], M[i-1, w] + c_vw)

  Retorna M[n-1,*]
```

- M retorna distância

Complexidade

- Loop mais interno passa por todos os vértices
  - para cada vértice, percorre vizinhos (grau)
- Soma dos graus é igual 2m, número de arestas
- Loop externo tem n-1 passos
- Complexidade O(n m)
  - se grafo for muito denso: $O(n^3)$
- Memória?
  - Armazenar a matriz M, $O(n^2)$

Melhorias Práticas (1)

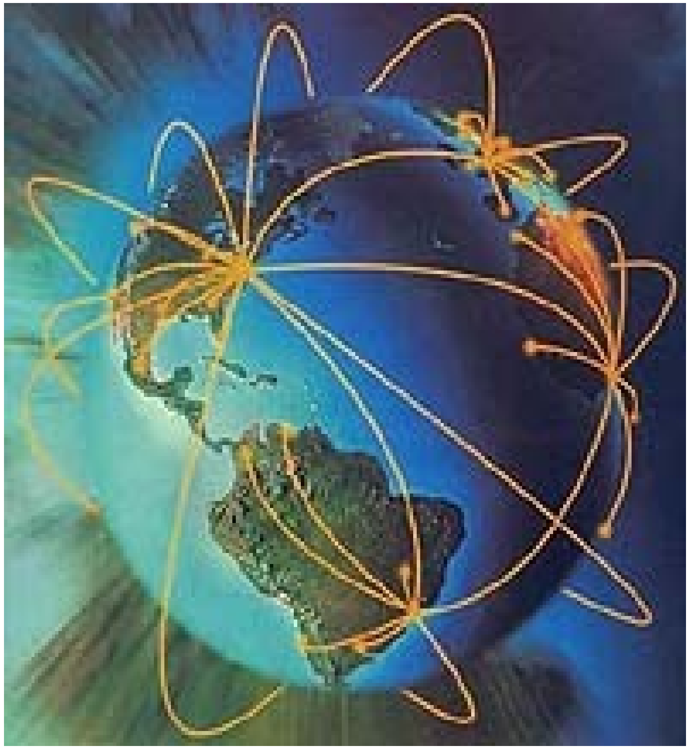
- Redução na quantidade de memória
- Manter vetor M[v], ao invés de matriz M[i,v]
  - não precisamos guardar distâncias de caminhos com menos arestas (já vimos este truque)
- M[v] : custo do menor caminho entre v e t que conhecemos até agora
  - número de arestas não importa
- Custo de memória: O(m + n)
  - armazenar os pesos das m arestas

Melhorias Práticas (2)

- Redução no tempo de execução
- Atualizar M[v] apenas quando M[w] for atualizado no passo anterior
  - para algum vizinho w de v
- Terminar algoritmo quando nenhum M[v] for atualizado
  - nenhuma distância vai reduzir
- Não reduz complexidade de pior caso
  - diferença na prática caminhos mínimos tem bem menos que n-1 arestas

Roteamento em Redes

- Rede global de comunicação
- Cada vértice é um *roteador*
- Arestas representam enlaces (*links*) entre roteadores
- Enlaces possuem custos (ex. tempo de propagação)
  - conhecidos pelo roteador



- **Problema**: cada roteador deve encontrar *melhor* rota (caminho) para todos os outros roteadores da rede

Roteamento em Redes

- 1) Solução Centralizada
  - Cada roteador obtém visão global da rede (ex. passa a conhecer o grafo da rede, com pesos)
  - Executa Dijkstra: árvore geradora com raiz no roteador define para qual vizinho enviar
- 2) Solução Distribuída
  - Cada roteador conhece apenas seus vizinhos
  - Descobre caminho mínimo através dos vizinhos (trocando mensagens)
  - Versão distribuída do Bellman-Ford

Bellman-Ford Distribuído

- Vetor M[] do algoritmo está *distribuído* entre os roteadores
  - cada roteador *v* é responsável por manter o valor M[v], para cada destino *t* (vetor com todos os destinos)
- Ordem de execução do loop interno do algoritmo não é importante
- Roteador *w* informa aos vizinhos sempre que M[w] diminuir
  - vizinho v atualiza M[v] e informa aos seus vizinhos
- Opções de melhor caminho se *propagam* pela rede
- Precisa executar para cada destino *t* da rede
  - mas pode ser executado em paralelo

Implementado por protocolos de redes