Teoria dos Grafos
Aula 8

Aula passada

- Prova 1

Aula de hoje

- Caminho mais curto entre todos os pares
- Algortimo de Floyd-Warshall

Distância em Grafos

- **Problema**: Dado G, com pesos nas arestas, qual é o menor caminho entre dois vértices?
- dado uma métrica para distância (ex. soma dos pesos)

Como resolver este problema?

- **Problema**: Menor caminho entre todos os pares de vértices?

Dijkstra n vezes!

Pesos Negativos

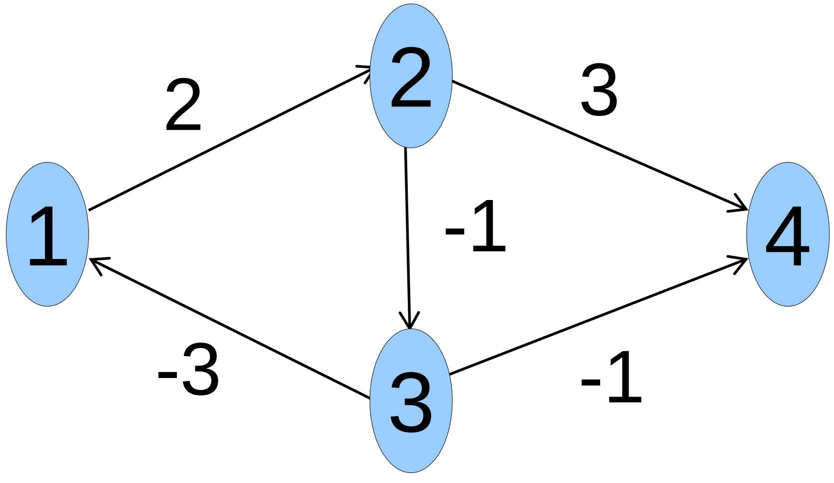
- **Problema**: Dijkstra funciona apenas para grafos com pesos postivos nas arestas
  - Contra-exemplo com pesos negativos?
  - Por que Dijkstra falha?

- **Problema**: Considerar grafos com pesos negativos nas arestas
- Ex. aplicações financeiras

Como resolver este problema?

Ciclos Negativos

- Ciclos com pesos negativos



- Peso do menor caminho entre 1 e 4?
- **Indefinido!**
- Caminho mais curto definido apenas quando não há ciclos negativos

Distância em Grafos

- Considerar grafo com pesos negativos, distânca é soma dos pesos
- **Problema 1**: calcular distância entre todos os pares de vértices
- **Problema 2**: calcular caminho mínimo entre todos os pares de vértices
- **Problema 3**: detectar ciclos negativos
- no final, assumir isto no momento

Algoritmo (eficiente) para este problema?

Caminho Mínimo

- Considere um par de vértices i, j

O que é a solução ótima?

- Caminho mais curto de i até j
- $p = (i, v_1, v_2, ..., j)$
- $v_1, v_2, ...$: vértices intermediários

O que podemos dizer sobre p?

- Considere o último vértice do grafo, ou seja, *n*
- Ou *n* pertence a *p* ou *n* não pertence a *p*

Análise do Caminho Mínimo

- Se *n* não pertence a *p*

O que podemos afirmar?

- $p = (i, v_1, v_2, ..., j)$ é caminho mínimo também para o grafo sem vértice n

- Se *n* pertence a *p*

O que podemos afirmar?

- $p = (i, ..., n , ..., j)$ é caminho mínimo
- $p_1 = (i, ..., n)$ e $p_2 = (n, ..., j)$ são caminhos mínimos

Análise do Caminho Mínimo

- Se p_1 = (i, ..., n) e p_2 = (n, ..., j) são caminhos mínimos

O que podemos afirmar sobre distâncias?

- d(i, j) = distância entre i e j?
- d(i, j) = d(i, n) + d(n, j)
- Além disso:
- p_1 = (i, ..., n) é mínimo usando como vértices intermediários 1, ..., n-1
- p_2 = (n, ..., j) é mínimo é mínimo usando como vértices intermediários 1, ..., n-1

Construindo uma Recursão

- p_1 = (i, ..., n) e p_2 = (n, ..., j) são caminhos mínimos sem utilizar n como intermediário
- Problema menor, com um vértice a menos!

Como construir recursão?

- Usar variável para indicar quais vértices intermediários estão sendo considerados na construção do caminho mínimo
- d(i,j,n) : distância entre i e j quando consideramos os vértices 1,..., n como intermediários
- d(i,j,n) é ótimo (distância é o valor do caminho mais curto)

Construindo uma Recursão

- Se vértice n não pertence ao caminho mínimo, então temos
- d(i,j,n) = d(i,j,n-1)
- Se vértice n pertence ao caminho mínimo, então temos
- d(i,j,n) = d(i,n,n-1) + d(n,j,n-1)

- Qual caso devemos usar?
- Menor deles!
- d(i,j,n) = min (d(i,j,n-1), d(i,n,n-1) + d(n,j,n-1))

Generalizando

- Considere o conjunto de vértices intermediários 1,2, ..., k.
- Considere o par de vértices i, j
- Considere a solução ótima (distância) usando apenas estes vértices
- Se vértice k não pertence ao ótimo, então temos
- $d(i,j,k) = d(i,j,k-1)$
- Se vértice k pertence ao ótimo, então temos
- $d(i,j,k) = d(i,k,k-1) + d(k,j,k-1)$
- Logo, temos
- $d(i,j,k) = \min(\ d(i,j,k-1),\ \ d(i,k,k-1) + d(k,j,k-1)\ )$

Algoritmo

- Algoritmo para calcular distâncias?
- iterativo, não recursivo (mas utilizando recursão)

```
Floyd-Warshall(A)
        Array d[1,...,n ; 1,...,n; 0,...,n]
  d[i,j,0] = A[i,j]; // peso das arestas
  d[i,i,0] = 0;      // peso zero

        for k = 1, ..., n
                for i = 1, ..., n
                   for j = 1, ..., n
                                d[i,j,k] = min(d[i,j,k-1] ,
                   d[i,k,k-1] + d[k,j,k-1] )
        return d[*,*,n]
```

- Complexidade?
- Memória e tempo de execução?

Algoritmo de Floyd-Warshall

- Descoberto por Floyd e Warshall em 1962 de maneira independente
- Determina distância mínima entre todos os pares de vértices de um grafo
- Complexidade Q(n^3)
- **Impressionante,** uma vez que grafo pode ter Q(n^2) arestas e *todos* os caminhos são considerados entre *todos* os Q(n^2) pares de vértices
- Poder de fogo da programação dinâmica

Algoritmo Melhorado

- Como reduzir uso de memória – $Q(n^3)$?
- Manter apenas 1 matriz de distâncias, atualizar na própria matriz

```
Floyd-Warshall(A)
        Array d[1,...,n; 1,...,n]
  d[0,i,j] = A[i,j]; // peso das arestas
  d[0,i,i] = 0;      // peso zero

        for k = 1, ..., n
                for i = 1, ..., n
                   for j = 1, ..., n
                                 d[i,j] = min(d[i,j], d[i,k]+d[k,j] )
        return d;
```

- Convencer que está correto!

Detectando Ciclos Negativos

Como detectar ciclos negativos?

- **Idéia**: se grafo tem ciclo negativo, custo para ir do vértice a ele mesmo é menor do que zero!
- Algoritmo atualiza todas as distâncias, inclusive entre o par de vértices i, i
- d(i,i) é considerada a cada passo k
- atualizada somente se d(i,k) + d(k,i) < 0, possível apenas se grafo tem ciclo negativo!
- Ao final do algoritmo, se d(i,i) < 0 para algum i, então temos ao menos um ciclo negativo!

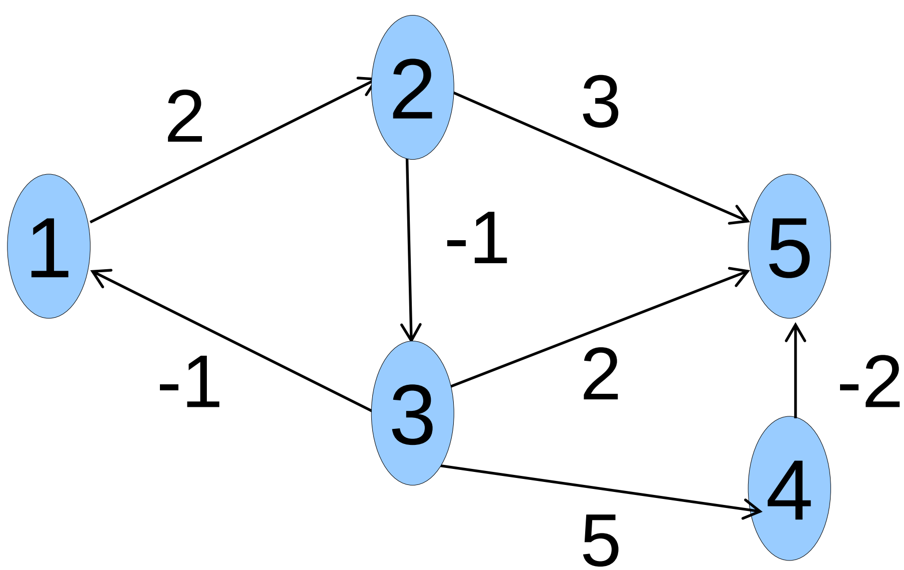
Mantendo Menor Caminho

Como obter o menor caminho?

- **Idéia**: manter sequencia de pais para cada par de vértices i,j
- parecido com Dijkstra
- pred(i, j) : pai do vértice j no caminho mínimo de i para j
- Atualizar pred(i, j) toda vez que distância entre i e j for atualizado passando por vértice k
- pred(i, j) = ?
- No final, usar recursão para imprimir caminho mínimo

Caminho Mínimo

- Dado grafo direcionado G, com pesos negativos
- e vértice qualquer t
- **Problema**: Calcular caminho mínimo e distância de todos os vértices de G ***até*** t



- Ex. caminho mais curto até 5
- Árvore geradora induzida

Algoritmo p/ Caminho Mínimo

- Abordagem via programação dinâmica
- caminho mínino entre um par de vértices
- Como definir a solução ótima?
- Custo do caminho mínimo (ótimo) P
- P = (v_1, v_2, ..., t)

- Que variáveis usar para definir subproblemas?
- encontrar P através de soluções ótimas para subproblemas menores
- Vértice de origem v, *número de arestas* que podem ser usadas no caminho, i = 0, ..., n-1

Analisando Solução Ótima

- Maior comprimento possível (em arestas) do caminho mínimo P entre *v* e *t*?
- n-1 arestas (caminho simples)
- 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

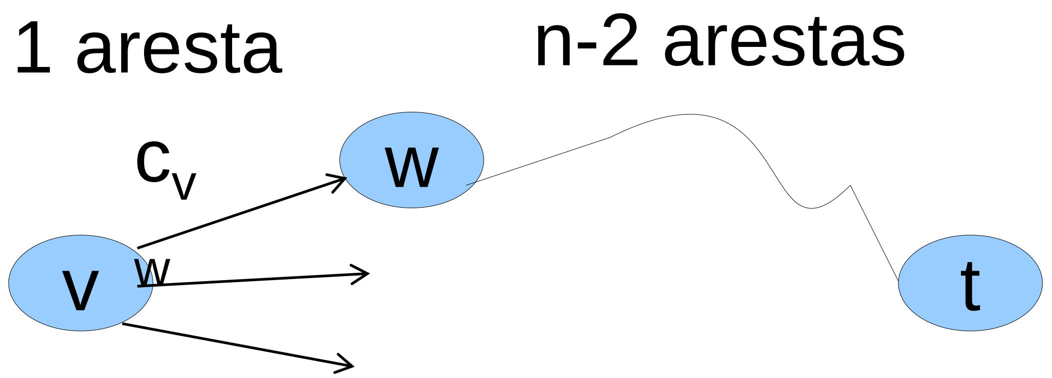
Analisando Solução Ótima

- 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
- Custo da solução ótima será a menor entre todos os caminhos com n-2 arestas, com w sendo vértice inicial, vizinho de v
- $OPT(n-1, v) = \min_w (OPT(n-2, w) + c_{vw})$

Analisando Solução Ótima

- Graficamente



- Qual vizinho w de v, P (com exatamente n-2 arestas) irá ter?
- o de menor custo

Generalizando

- Supor P caminho ótimo de v para t com exatamente i arestas
- Caminho ótimo P é dado pelo mínimo entre os dois casos
- P possuir exatamente i arestas ou
- P possuir menos de i arestas
- $OPT(i, v) = \min (OPT(i-1, v), \min_w (OPT(i-1, w) + c_{vw}))$

Algoritmo de Bellman-Ford

- Algoritmo para calcular OPT(n-1, *)?
- iterativo, não recursivo (mas utilizando recursão)

```
Shortest-Path(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

- Tempo de execução do algoritmo?
- Cada elemento da matriz M tempo proporcional ao grau do vértice
- Soma dos graus: O(m)
- Complexidade O(m n)
- se grafo for denso: $O(n^3)$
- Memória?
- Matriz M, $O(n^2)$

Melhorias Práticas (1)

- Redução na quantidade de memória
- Manter vetor M[v], ao invés de matriz M
- Não precisamos de caminhos com menos arestas
- M[v] : custo do caminho mais curto entre v e t que conhecemos até agora
- Custo: O(m + n)
- pois precisamos representar o grafo

Melhorias Práticas (2)

- Redução no tempo de execução
- Atualizar M[v] apenas quando M[w] for atualizado no passo anterior
- Terminar algoritmo quando nenhum M[v] for atualizado
- Nada mais irá mudar!
- Não reduz complexidade de pior caso
- mas faz diferença prática, pois caminhos mais curtos tem muito menos do que n-1 arestas

Algoritmo de Roteamento

- Cada vértice é um *roteador* (ou uma rede)
- Arestas representam enlaces (*links*) entre roteadores (ou entre redes)
- Enlaces possuem custos (ex. tempo de propagação)

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

Algoritmo de Roteamento

- **Soluções?**
- 1) Centralizada
- Cada roteador obtém visão global da rede (ex, todos roteadores, links e pesos)
- Executa Dijkstra (árvore geradora define para qual vizinho enviar)
- 2) Distribuída
- Cada roteador conhece apenas seus vizinhos
- “Descobre” caminho através dos vizinhos
- Como? Algoritmo anterior (Bellman-Ford)

Algoritmo de Roteamento

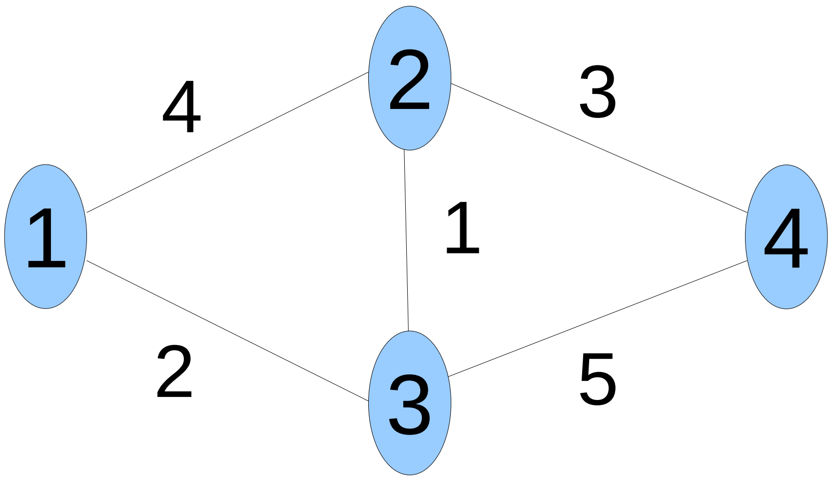
- Vetor M[] do algoritmo anterior está *distribuído* entre os roteadores
  - cada roteador v é responsável por manter M[v]
- Ordem de execução do loop no algoritmo anterior não é importante
- Roteador *v* informa aos vizinhos sempre que M[v] diminuir
  - vizinhos atualizam M[] e então informam aos seus vizinhos
- Opções de melhor caminho se *propagam* pela rede

Distance Vector Protocols

- Classe de algortimo de roteamento
- algoritmos distribuídos
- Cada roteador v, mantém M[v] para cada *destino* da rede (outros rotedores)
- Mantém melhor caminho via cada vizinho
- Propaga melhor caminho quando atualizado
- Algoritmos de roteamento utilizado na Internet
- BGP é variação desta idéia

Distance Vector Protocols

- Exemplo



- Cada vértice mantém tabela
- Para cada destino da rede (linhas)
- Para cada vizinho (colunas)
- Custo do melhor caminho
- Informa vizinhos do melhor caminho quando custo é atualizado