Teoria dos Grafos
Aula 23

Aula passada

- Apresentação de trabalhos
- Discussão da prova
- *Subset sum*
- Problema da mochila

Aula de hoje

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

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

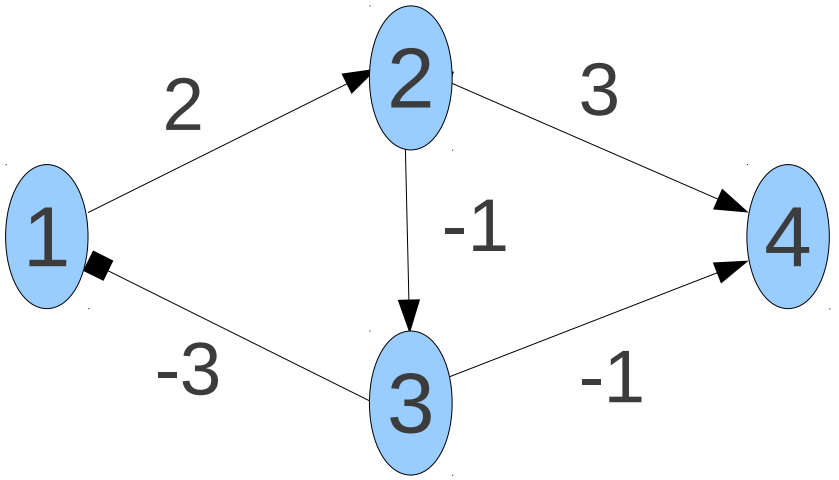
- **Problema**: Dijkstra funciona apenas para grafos com pesos postivos nas arestas
- Contra-exemplo com pesos negativos?
- Por que Dijkstra falha?

- **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
  - $p = (i, v_1, v_2, \ldots, j)$
  - $v_1, v_2, \ldots$: 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 $\Theta(n^3)$
- **Impressionante,** uma vez que grafo pode ter $\Theta(n^2)$ arestas e *todos* os caminhos são considerados entre *todos* os $\Theta(n^2)$ pares de vértices
- Poder de fogo da programação dinâmica

Algoritmo Melhorado

- Como reduzir uso de memória – $\Theta(n^3)$?
- 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