Projeto e Análise de Algoritmos (PAA)

Caminhos Mínimos de Todos para Todos, Algoritmos de Floyd-Warshall e de Johnson

O problema dos caminhos mínimos de todos para todos

Neste problema recebemos como entrada:

- um grafo orientado (ou dirigido) G=(V,E),
- com custo c(e) em cada aresta 'e' em E.

E queremos encontrar:

- o valor do caminho mínimo de 'u' até 'v' para todo par u, v em V.
- Também gostaríamos que esses caminhos fossem devolvidos.

No caso de haver um circuito negativo na entrada

- o valor dos caminhos mínimos não está bem definido.
  - Por isso, tal fato deve ser reportado.

Com o conhecimento que já temos, podemos resolver esse problema.

- Quiz1: Como? Que conhecimento?

Uma estratégia para resolver o problema é

- executar um algoritmo de caminhos mínimos a partir de cada vértice.

- Este procedimento leva tempo
  - O(n * tempo do algoritmo de caminhos mínimos utilizado).

Se os custos da entrada são não negativos

- podemos usar o algoritmo de Dijkstra, que leva tempo Theta(m log n).

- Portanto, nosso algoritmo para encontrar
  - caminhos mínimos de todos para todos levará tempo
    - n * Theta(m log n) = Theta(n m log n).

- Se o grafo é esparso, i.e., m = Theta(n) temos
  - Theta(n m log n) = Theta (n^2 log n).
- o que é muito bom, já que, apenas para devolver os valores das soluções,
  - precisamos preencher n^2 posições.

- Se o grafo for denso, i.e., m = Theta(n^2) temos
  - Theta(n m log n) = Theta (n^3 log n)
- o que é razoável, já que gastamos pouco mais que tempo linear
  - por caminho mínimo encontrado.

Se os custos da entrada podem ser negativos temos que

- usar o algoritmo de Bellman-Ford, que leva tempo Theta(n m). $\Theta(n \cdot m)$

- Portanto, nosso algoritmo para encontrar
  - caminhos mínimos de todos para todos levará tempo
    - n * Theta(n m) = Theta(n^2 m). $\Theta(n^2 \cdot m)$

- Se o grafo é esparso, i.e., m = Theta(n) temos Theta(n^2 m) = Theta (n^3),
  - o que é razoável, já que gastamos tempo linear
    - por caminho mínimo encontrado.

$m = \Theta(n) \Rightarrow$ tempo $\Theta(n^3)$

- Se o grafo for denso, i.e., m = Theta(n^2) temos Theta(n^2 m) = Theta (n^4),
  - o que já não é tão bom, embora seja polinomial,
    - i.e., muito melhor que buscas exaustivas exponenciais.

$m = \Theta(n^2)$ → tempo $\Theta(n^4)$

Será que conseguimos fazer melhor?

- Ao olhar para caminhos mínimos entre todos os pares de vértices,
  - percebemos que subcaminhos ótimos podem ser compartilhados,
    - o que evita recálculos.
- Seguindo essa intuição, veremos um algoritmo de programação dinâmica
  - que usa uma subestrutura ótima diferente
    - daquela do algoritmo de Bellman-Ford
- e resolve o problema em tempo Theta(n^3).

os pares (a,c), (a,d), (b,c) e (b,d) compartilham o mesmo subcaminho ótimo

Subestrutura ótima para o algoritmo de Floyd-Warshall

No problema dos caminhos mínimos de todos para todos
- não temos uma origem fixa.

- Por isso, para definir cada subproblema precisamos
  - pelo menos de dois parâmetros, i.e., origem 'i' e destino 'j'.

Além disso, também é crítico definir claramente
- quais subproblemas são menores e quais são maiores.

- No algoritmo de Bellman-Ford usamos um parâmetro
  - que determinava o número de arestas (ou vértices) máximo
    - que o caminho podia ter.
- Nessa análise de subestrutura ótima,
    - que dará origem ao algoritmo de Floyd-Warshall,
  - vamos usar algo um pouco mais forte.

Dada uma ordem arbitrária dos vértices, i.e., V = {1, 2, ..., n},
- usaremos um parâmetro 'k' que determina que
  - apenas os primeiros 'k' vértices da ordem
    - podem ser vértices intermediários do caminho,
  - i.e., os vértices intermediários estão em V(k) = {1, 2, ..., k}.

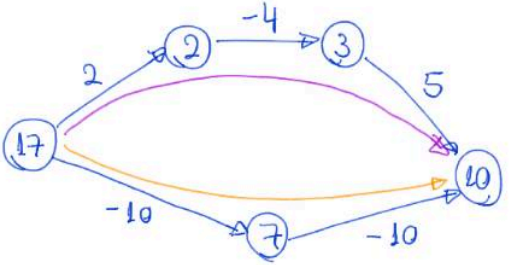
Lembra o que usamos no problema da mochila

Definidos os parâmetros i, j, k, temos L(i, j, k)

- que corresponde ao problema de encontrar
  - o menor caminho de 'i' até 'j'
- que só usa vértices intermediários de V(k) = {1, 2, ..., k}.

Exemplos:

Origem $i = 17$
destino $j = 10$
limitador $k = 5$



$i = 17$
$j = 10$
$k = 8$

$A[17, 10, 5] = 3$

$A[17, 10, 8] = -20$

$i = 17 \quad j = 10 \quad k = 2$

$A[17, 10, 2] = +\infty$

desenha

Sendo P a solução ótima para L(i, j, k)

- vamos analisar sua subestrutura ótima em relação ao vértice 'k'.

- Temos duas possibilidades: $k \in P$ ou $k \notin P$

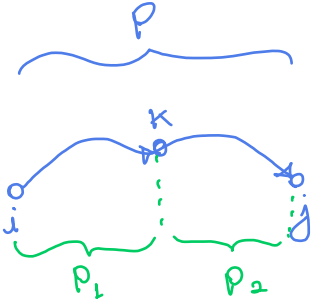
$k \notin P$

Caso 1) Se o vértice 'k' não é um vértice intermediário de P
- então P é uma solução ótima de L(i, j, k-1).

} P é sol. ótima de $L(i, j, k-1)$

$k \in P$

Caso 2) Se o vértice 'k' é um vértice intermediário de P então seja
- P1 a parte de P que vai de 'i' até 'k',
- P2 a parte de P que vai de 'k' até 'j'.



- Como não temos ciclos negativos, P não tem circuitos.
- Portanto, o vértice 'k' não aparece no meio de P1,
  - que é uma solução de L(i, k, k-1).
- De modo similar, o vértice 'k' não aparece no meio de P2,
  - que é uma solução de L(k, j, k-1).

P_1 é sol. ótima de $L(i, k, k-1)$

P_2 é sol. ótima de $L(k, j, k-1)$

Resta mostrar que, tanto P1 quanto P2
- são soluções ótimas de seus respectivos subproblemas.

- Para tanto, basta fazer um argumento por contradição e mostrar que
  - se houvesse uma solução melhor para algum dos subproblemas,
- obteríamos uma solução melhor que P para L(i, j, k),
  - o que é uma contradição.

Recorrência → $A[i,j,k] = \min\{A[i,j,k-1], A[i,k,k-1] + A[k,j,k-1]\}$

Valor do cam. mín. de i até j que só usa vértices intermediários em $V_k = \{1,2,...,k\}$

Dessa subestrutura ótima derivamos a seguinte recorrência

- A[i, j, k] = min{A[i, j, k-1], A[i, k, k-1] + A[k, j, k-1]}

- sendo que A[i, j, k] corresponde ao valor do caminho mínimo
  - de 'i' para 'j' que só usa vértices em V(k) = {1, 2, ..., k}.

Os valores de 'i' e 'j' variam sobre todos os vértices, i.e, de 1 até 'n',

- e 'k' varia de '0' até 'n',
  - i.e., V0 = emptyset até Vn = V. $V_0 = \emptyset$ e $V_n = V$

$i \in \{1, ..., n\}$
$j \in \{1, ..., n\}$
$k \in \{0, 1, ..., n\}$

Os casos base ocorrem quando k = 0. Nestes casos

- A[i, j, 0] = 0 se i = j,
  - pois existe caminho de 'i' para 'i' sem arestas e custo '0'. } $i = j \Rightarrow A[i,j,0] = 0$

- A[i, j, 0] = c(i, j) se (i, j) \in E,
  - pois existe caminho de 'i' para 'j' com apenas a aresta (i, j). } $(i,j) \in E \Rightarrow A[i,j,0] = c(i,j)$

- A[i, j, 0] = +\infinity se i != j e (i, j) \not \in E,
  - pois não existe caminho de 'i' para 'j' sem vértices intermediários.

$i \neq j$ e $(i,j) \notin E \Rightarrow A[i,j,0] = +\infty$

Algoritmo de Floyd-Warshall

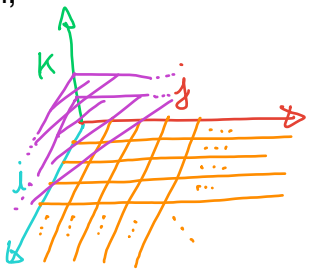
```
algFloydWarshall(G = (V, E), c) {

    // Casos Base
    para i = 1 até n:
        para j = 1 até n:
            se i = j: então A[i, j, 0] = 0
            senão se (i, j) \in E: então A[i, j, 0] = c(i, j)
            senão A[i, j, 0] = +\infinity

    para k = 1 até n:
        para i = 1 até n:
            para j = 1 até n:
                A[i, j, k] = min{A[i, j, k-1], A[i, k, k-1] + A[k, j, k-1]}
}
```

Eficiência: Theta(n^3), três laços aninhados, cada um indo de 1 até n,

- e resolver a recorrência leva tempo constante.

Quiz2: Posso inverter a ordem dos laços aninhados? Por que?

Quiz3: Em que células da matriz estão as soluções ótimas?



Bônus: Como detectar circuito negativo?

Basta percorrer a diagonal A[i, i, n], para i = 1, ..., n.

- Se não houver circuito negativo ela estará preenchida de zeros.
    - Se houver, ao menos um posição terá valor negativo.

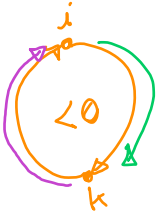
∃ caminho de v p/ v c/ valor menor que 0

Para obter alguma intuição do por que isso ocorre,

- considere que existe um circuito negativo.
- Seja ‘k’ o vértice de mais alto índice na nossa ordem,
    - i.e., o último vértice do circuito a ser permitido num subproblema.
- Seja ‘i’ um outro vértice do circuito.

Considere a iteração do algoritmo em que A[i, i, k] é calculado.

- Pela recorrência, temos que A[i, k, k-1] + A[k, i, k-1] será considerado.
    - Mas este é exatamente o valor do circuito negativo.
- Assim, nesta iteração A[i, i, k] receberá um valor negativo,
    - o qual será passado adiante até o final do algoritmo.



$A[i,i,k] \leq A[i,k,k-1] + A[k,i,k-1]$

Bônus: Como reconstruir a solução?

Como nossa recorrência basicamente decide

- se um vértice intermediário é ou não usado
  - no caminho mínimo entre dois pontos,
- para reconstruir a solução podemos usar uma matriz auxiliar B[i, j]
  - cujo valor é o vértice intermediário de mais alto índice
    - usado no caminho mínimo de 'i' até 'j'.

Assim, para reconstruir o caminho de 'i' até 'j', colocamos

  - o vértice em B[i, j] no "meio" do caminho sendo reconstruído.
- Então, recursivamente reconstruímos
  - a primeira parte do caminho de 'i' até B[i, j],
- e, também recursivamente reconstruímos
  - a segunda parte do caminho de B[i, j] até 'j'.

função imprime os nós intermediários

```
recSol(B, i, j):
  se B[i,j] ≠ NULL:
    v = B[i,j]
    recSol(B, i, v)      1°s.p.
    imprime(v)
    recSol(B, v, j)      2°s.p.
```

Quiz: adicionar linhas p/ preencher B[] no código do alg.

Quiz: faça código do alg. que recebe B[] preenchida e reconstrói o caminho de i até j.