Advanced Algorithm Design and Analysis (Lecture 7)

SW5 fall 2004

Simonas Šaltenis

E1-215b

simas@cs.aau.dk

All-pairs shortest paths

- Main goals of the lecture:
  - *to go through one more* ***example*** *of* ***dynamic programming*** *– to solve the all-pairs shortest paths and transitive closure of a weighted graph (the* ***Floyd-Warshall*** *algorithm);*
  - *to see how algorithms can be adapted to work in different settings (idea for reweighting in* ***Johnson****'s algorithm)*
  - *to be able to compare the applicability and efficiency of the different algorithms solving the all-pairs shortest paths problems.*

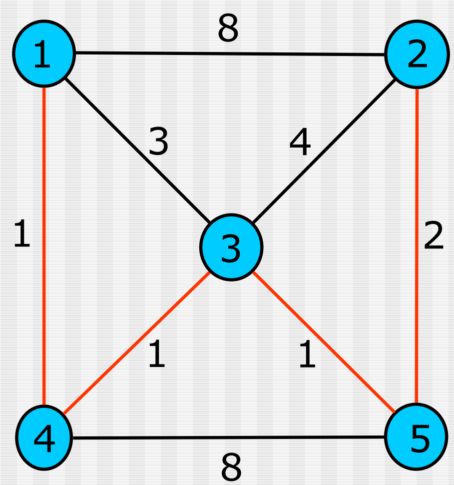
Input/Output

- What is the input and the output in the *all-pairs shortest path problem*?
  - *What are the popular memory representations of a weighted graph*?
  - Input: **adjacency matrix**
    - Let $n = |V|$, then $W=(w_{ij})$ is an n x n matrix, where
    - w_{ij}=0, if $i = j$;
    - w_{ij}=weight of the edge (i,j) or ∞, if $(i,j) \notin E$
  - Output:
    - **Distance matrix**
    - **Predecessor matrix**

Input/Output

- Output:
  - **Distance matrix**
    - $D=(d_{ij})$ is an n x n matrix, where $d_{ij} = \delta(i,j)$ – weight of the shortest path between vertices i and j.
  - **Predecessor matrix**
    - $P=(p_{ij})$ is an n x n matrix, where $p_{ij} = nil$, if $i = j$ or there is no shortest path from i to j, otherwise p_{ij} is the predecessor of j on a shortest path from i.
    - The i-th row of this matrix *encodes* the shortest-path tree with root i.

Example graph



- *Write an adjacency matrix for this graph*.
- *Give the first row of the predecessor matrix (to encode the shown shortest path tree).*

Sub-problems

- *What are the sub-problems? Defined by which parameters?*
- Options:
  - $L^{(m)}(i,j)$ – minimum weight of a path between i and j containing at most m edges.
  - $d^{(k)}(i,j)$ – minimum weight of a path where the only *intermediate* vertices (not i or j) allowed are from the set $\{1, \ldots, k\}$.
- Floyd-Warshall algorithm uses $d^{(k)}(i,j)$ as a sub-problem
  - $d^{(n)}(i,j)$ is the solution to the whole problem

Solving sub-problems

- *How are sub-problems solved? Which choices have to be considered?*
  - Let p be the shortest path from i to j containing only vertices from the set {1, …, k}. Optimal sub-structure:
    - If vertex k **is not** in p then a shortest path with intermediate vertices in {1, …, k-1} is also a shortest path with intermediate vertices in {1, …, k}.
    - If k is an intermediate vertex in p, then we break down p into p_1(i to k) and p_2(k to j), where p_1 and p_2 are shortest paths with intermediate vertices in {1, …, k-1}.
  - Choice – either we include k in the shortest path or not!

Trivial Problems, Recurrence

- *What are the trivial problems*?
  - $d^{(0)}(i,j) = w_{ij}$
- Recurrence:

$$d^{(k)}(i,j) = \begin{cases} w_{ij} & \text{if } k = 0 \\ \min\left(d^{(k-1)}(i,j),\ d^{(k-1)}(i,k) + d^{(k-1)}(k,j)\right) & \text{if } k \geq 1 \end{cases}$$

- *What order have to be used to compute the solutions to sub-problems*?
  - Increasing k
  - Can use one matrix D – no danger of overwriting old values as $d^{(k)}(i,k) = d^{(k-1)}(i,k)$ and $d^{(k)}(k,j) = d^{(k-1)}(k,j)$

The Floyd-Warshall algorithm

```
Floyd-Warshall(W[1..n][1..n])
D ← W    // D^(0)
for k ← 1 to n do // compute D^(k)
   for i ←1 to n do
      for j ←1 to n do
        if D[i][k] + D[k][j] < D[i][j] then
           D[i][j] ←D[i][k] + D[k][j]
return D
```

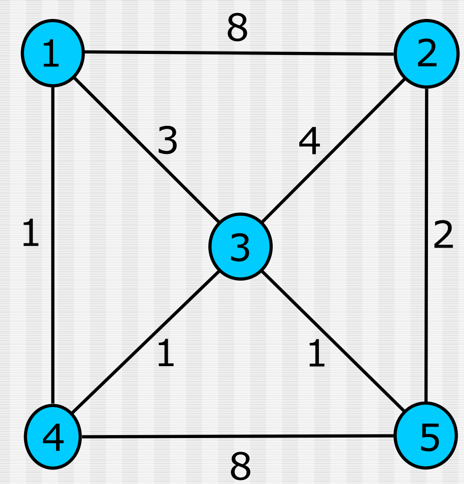
Computing predecessor matrix

- *How do we compute the predecessor matrix?*
  - Initialization: $p^{(0)}(i,j) = \begin{cases} nil & \text{if } i = j \text{ or } w_{ij} = \infty \\ i & \text{if } i \neq j \text{ and } w_{ij} < \infty \end{cases}$
  - Updating:

```
Floyd-Warshall(W[1..n][1..n])
…
for k ← 1 to n do // compute D(k)
   for i ←1 to n do
      for i ←1 to n do
         if D[i][k] + D[k][j] < D[i][j] then
            D[i][j] ←D[i][k] + D[k][j]
            P[i][j] ← P[k][j]
return D
```

Analysis, Example

- *When does it make sense to run Floyd-Warshall*?
  - Running time: $O(V^3)$
  - Graphs with and without negative edges
  - Sparse and dense graphs
  - Constants behind the O notation
- *Run the first iteration of the algorithm (*k=1*), show both D and P matrices.*



Transitive closure of the graph

- Input:
  - Un-weighted graph G: $W[i][j] = 1$, if $(i,j) \in E$, $W[i][j] = 0$ otherwise.
- Output:
  - $T[i][j] = 1$, if there is a path from i to j in G, $T[i][j] = 0$ otherwise.
- Algorithm:
  - Just run Floyd-Warshall with weights 1, and make $T[i][j] = 1$, whenever $D[i][j] < \infty$.
  - More efficient: use only Boolean operators

Transitive closure algorithm

```
Transitive-Closure(W[1..n][1..n])
T ← W    // T^(0)
for k ← 1 to n do // compute T^(k)
   for i ←1 to n do
      for i ←1 to n do
         T[i][j] ← T[i][j] ∨ (T[i][k] ∧ T[k][j])
return T
```

Sparse graphs

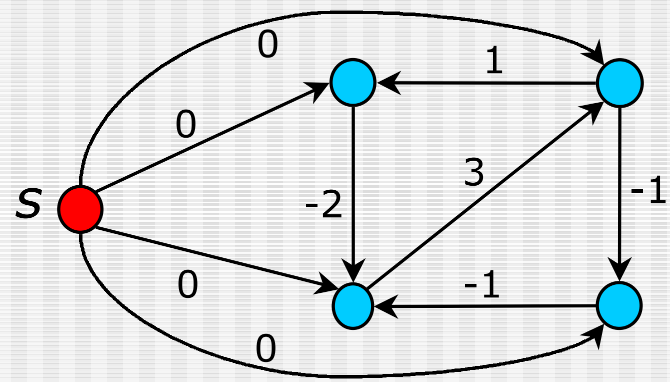
- *What if the graph is sparse*?
  - If no negative edges – run repeated Dijkstra's
  - If negative edges – let us somehow change the weights of all edges (to w') and then run repeated Dijkstra's
- Requirements for ***reweighting***:
  - *Non-negativity*: for each (u,v), $w'(u,v) \geq 0$
  - *Shortest-path equivalence*: for all pairs of vertices u and v, a path p is a shortest path from u to v using weights w *if and only if* p is a shortest path from u to v using weights w'.

Reweighting theorem

- *Rweighting does not change shortest paths*
  - Let h: V $\rightarrow$ **R** be any function
  - For each $(u,v) \in E$, define
    $w'(u,v) = w(u,v) + h(u) - h(v)$.
  - Let $p = (v_0, v_1, \ldots, v_k)$ be any path from v_0 to v_k
  - *Then*: $w(p) = \delta(v_0, v_k) \Leftrightarrow w'(p) = \delta'(v_0, v_k)$

Choosing reweighting function

- *How to choose function h*?
- The idea of Johnson:
  - 1. Augment the graph by adding vertex *s* and edges (*s*,*v*) for each vertex *v* with 0 weights.



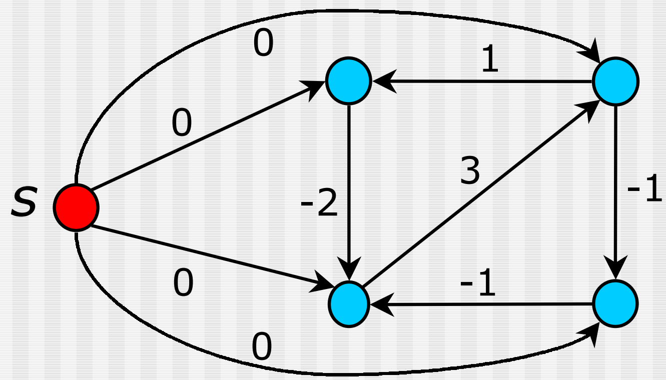
  - 2. Compute the shortest paths from *s* in the augmented graph (using Belman-Ford).
  - 3. Make $h(v) = \delta(s, v)$

Johnson's algorithm

- *Why does it work*?
  - By definition of the shortest path: for all edges (u,v), $h(u) \leq h(v) + w(u,v)$
  - Thus, $w(u,v) + h(u) - h(v) \geq 0$
- Johnson's algorithm:
  - 1. Construct augmented graph
  - 2. Run Bellman-Ford (possibly report a negative cycle), to find $h(v) = \delta(s, v)$ for each vertex v
  - 3. Reweight all edges:
    - $w'(u,v) \leftarrow w(u,v) + h(u) - h(v)$.
  - 4. For each vertex u:
    - Run Dijkstra's from u, to find $\delta'(u, v)$ for each v
    - For each vertex v: $D[u][v] \leftarrow \delta'(u, v) + h(v) - h(u)$

Example, Analysis

- *Do the reweighting on this example:*



- *What is the running time of Johnson's?*