All-pairs shortest paths

Suprakash Datta

March 11, 2011

Last week

Single source shortest paths:

Shortest paths from a given node to all nodes.
Dijkstra's algorithm:

- Complexity (using heaps): $O((V+E)\log V)$.
- Complexity (using Fibonacci heaps): $O(V\log V+E)$.
- Cannot handle negative edges

Single-source shortest paths with negative edges:
Bellman-Ford Algorithm

- may be covered later, covered in CSE3101
- Complexity: $\Theta(VE)$

All pairs shortest paths

Compute shortest paths between all pairs of nodes.

Possible approaches:

1. Run Dijkstra from each node
   - Complexity (using heaps): $O((V^2 + VE)\log V)$
     Complexity (using Fibonacci heaps): $O(V^2 \log V + VE)$.
   - Cannot handle negative edges
2. Run Bellman-Ford from each node
   - Complexity: $O(V^2 E)$ – recall that E may be $\Theta(V^2)$
   - Simple code, works with negative edges
3. Floyd-Warshall algorithm
   - Complexity: $O(V^3)$
   - Simple code, works with negative edges

Specifications/Assumptions

INPUT: a directed graph $G = (V, E)$.

- Nodes are numbered $1 \ldots n$.
- Each edge $\langle i, j \rangle$ has a real-valued weight w_{ij} [1]
- Assume that all cycles have non-negative cost.

OUTPUT: A matrix $D = [d_{ij}]$, $d_{ij} =$ the (cost of the) shortest path from i to j

[1] formally $w : E \to \mathbb{R}$.

The Floyd Warshall algorithm - Intuition

Definition

$c_{ij}^{(k)}$: the cost of the shortest path from i to j, with the intermediate nodes being restricted to the set $\{1, 2, \ldots, k\}$.

- Therefore

$$\begin{aligned} c_{ij}^{(0)} &= 0 \text{ if } i = j && (1) \\ &= w_{ij} \text{ if } i \neq j \text{ and } (i, j) \in E, && (2) \\ &= \infty \text{ otherwise} && (3) \end{aligned}$$

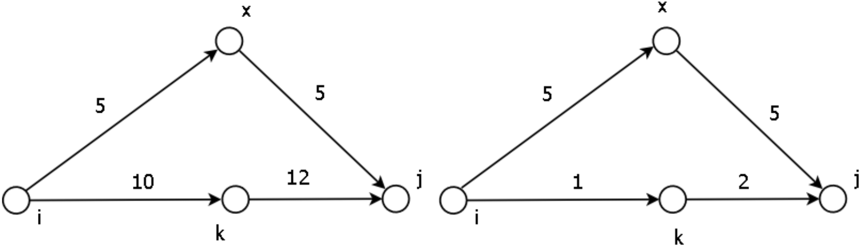
- $$c_{ij}^{(k)} = \min[c_{ij}^{(k-1)}, c_{ik}^{(k-1)} + c_{kj}^{(k-1)}]$$

- The (cost of the) shortest path from i to j is $d_{ij} = c_{ij}^{(n)}$.

The Floyd Warshall algorithm - Intuition

$$c_{ij}^{(k)} = \min[c_{ij}^{(k-1)}, c_{ik}^{(k-1)} + c_{kj}^{(k-1)}]$$

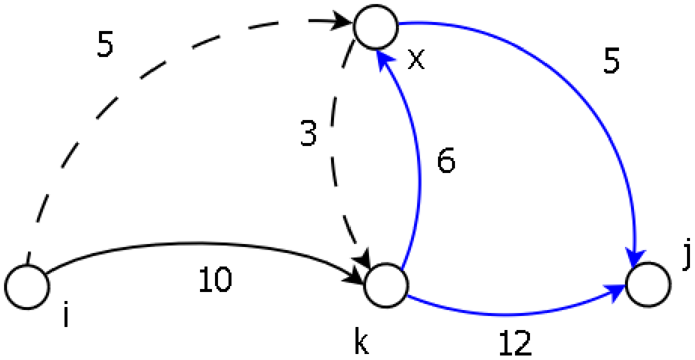
Figure: Two possibilities for node k



The Floyd Warshall algorithm - questions

In the expression $c_{ik}^{(k-1)} + c_{kj}^{(k-1)}$, there is nothing that says the intermediate nodes in $c_{ik}^{(k-1)}$ are disjoint from those in $c_{kj}^{(k-1)}$. What if there is overlap?

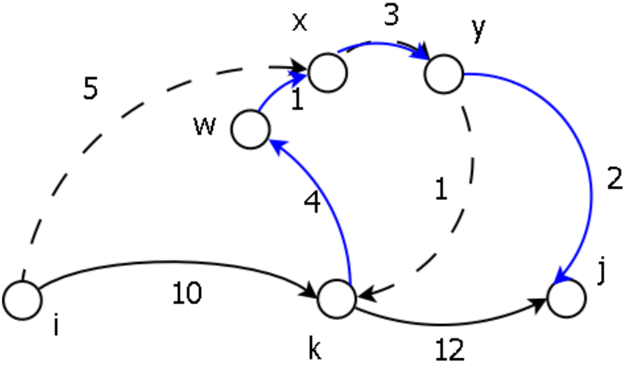
Figure: Scenario 1



The Floyd Warshall algorithm - questions

In the expression $c_{ik}^{(k-1)} + c_{kj}^{(k-1)}$, there is nothing that says the intermediate nodes in $c_{ik}^{(k-1)}$ are disjoint from those in $c_{kj}^{(k-1)}$. What if there is overlap?

Figure: Scenario 2



The Floyd Warshall algorithm - steps

Input: $W = [w_{ij}]$
Output: $D = [d_{ij}]$
$\textsc{Floyd-Warshall}(W)$

$n \leftarrow rows(W)$
$D^{(0)} \leftarrow W$
for $k \leftarrow 1$ **to** n
do $D^{(k)} \leftarrow D^{(k-1)}$
    for $i \leftarrow 1$ **to** n
    do for $j \leftarrow 1$ **to** n
        do if $D^{(k)}[i,j] > D^{(k-1)}[i,k] + D^{(k-1)}[k,j]$
            then $D^{(k)}[i,j] \leftarrow D^{(k-1)}[i,k] + D^{(k-1)}[k,j]$
return $D^{(n)}$

Compare with: $c_{ij}^{(k)} = \min[c_{ij}^{(k-1)}, c_{ik}^{(k-1)} + c_{kj}^{(k-1)}]$.

The Floyd Warshall algorithm - simplified

```
FLOYD-WARSHALL(W)
  n ← rows(W)
  D ← W
  for k ← 1 to n
  do for i ← 1 to n
       do for j ← 1 to n
            do if D[i, j] > D[i, k] + D[k, j]
                 then D[i, j] ← D[i, k] + D[k, j]
  return D
```

Notes

- recall: $w_{ij} = 0$, if $i = j$, weight of edge $\langle i, j \rangle$ if it exists, otherwise ∞
- the pseudocode assumes the use of ∞ – use a suitable guard in your code. For today, all edges are non-negative; so -1 can be a guard value.

The Floyd Warshall algorithm - computing paths

Maintain a matrix called *via*

Floyd-Warshall(W)

```
n ← rows(W)
D ← W
via ← nil //initialize all entries
for k ← 1 to n
do for i ← 1 to n
    do for j ← 1 to n
        do if D[i, j] > D[i, k] + D[k, j]
            then D[i, j] ← D[i, k] + D[k, j]
                via[i, j] ← k
return D, via
```

The Floyd Warshall algorithm - printing paths

```
PrintFWPath(i, j)
if via[i, j] = nil
   then print(i, j)
   else PrintFWPath(i, via[i, j])
        PrintFWPath(via[i, j], j)
```

If you want to store the path (instead of print) insert into a queue instead.

The Floyd Warshall algorithm - notes

- Example of **dynamic programming**
  1. When applicable, often provides very efficient solutions to optimization problems
  2. Many contest problems require this technique
  3. covered in detail in CSE 3101 (Design and Analysis of Algorithms).
- The efficiency arises from the clever formulation – the more obvious dynamic programming formulation yields a $\Theta(V^4)$-time algorithm.
- Proofs of correctness are skipped here