Lecture 12: Overview

- All Pairs Shortest Path
- Read Chapter 25
All Pairs Shortest Path

- For all vertices, i,j, want the shortest path between them
- Could simply use multiple invocations of Belman-Ford
- $O(V^2E)$ - for dense graph this is roughly $O(V^4)$
All Pairs Shortest Path

- matrix $W = (w_{ij})$
- Recursive solutions
  - matrix $L = (l_{ij})$
  - $l_{ij}^{(m)}$ = minimum path weight $i$ to $j$ containing at most $m$ edges
    
    $l_{ij}^{(0)} = \begin{cases} 
    0 & \text{if } i = j \\
    \infty & \text{if } i \neq j 
    \end{cases}$

- for $m^{th}$ iteration, want to RELAXPAIR$(i,k,j)$
- if a path from $i$ to $k$ in $m-1$ steps, and $k$ to $j$ in 1 step is shorter than the current best path from $i$ to $j$, then take path $i...k->j$
All Pairs Shortest Path

\[ l_{ij}^{(m)} = \min(l_{ij}^{(m-1)}, \min\{l_{ik}^{(m-1)} + w_{kj}\}) \]
\[ = \min\{l_{ik}^{(m-1)} + w_{kj}\} \]

- Note that \[ l_{ij}^{(n-1)} = l_{ij}^{(n)} = l_{ij}^{(n+1)} = \ldots \]
Extend-Shortest-Paths(L,W)

1 n <- rows[L]
2 let L’=(l_{ij}') be an n x n matrix
3 for i <- 1 to n
4 do for j <- 1 to n
5 do l_{ij}' <- \infty
6 for k <- 1 to n
7 do l_{ij}' <- \min(l_{ij}', l_{ik} + w_{kj})
8 return L’
Computing Shortest Paths

- Can repeatedly use this to compute shortest path
- But $O(V^4)$ - same cost as Belman Ford
- Draw inspiration from matrix multiple example
- $W$ gives shortest paths of length 1
- First use of extend-shortest paths gives shortest paths up to length 2
- Can use this instead of $W$ to give shortest paths up to length 4...
FAST-ALL-PAIRS-SHORTEST-PATHS(W)

1 n <- rows[W]
2 \(L^{(1)} \leftarrow W\)
3 m <- 1
4 while m < n-1
5 do \(L^{(2m)} \leftarrow \text{EXTEND-SHORTEST-PATHS}(L^{(m)},L^{(m)})\)
6 m <- 2m
7 return \(L^{(m)}\)
Floyd-Warshall Algorithm

- Idea: consider vertices \{1,\ldots,k\}
- Shortest path from i to j with intermediate vertices in set \{1,\ldots,k\}
- Two cases:
  - Doesn’t include k, then it is the same as shortest path in set \{1,\ldots,k-1\}
  - Includes k, then it can be composed with the shortest path from i to k in \{1,\ldots,k-1\} and the shortest path from k to j in \{1,\ldots,k-1\}
- Recursion relation:

\[
d_{ij}^{(k)} = \begin{cases} 
  w_{ij} & \text{if } k = 0 \\
  \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) & \text{if } k \geq 1
\end{cases}
\]
FLOYD-WARSHALL(W)

1 n <- rows[W]
2 D^{(0)} <- W
3 for k <- 1 to n
4    do for i <- 1 to n
5        do for j <- 1 to n
6            do d_{ij}^{(k)} <- min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})
7    return D^{(n)}
Johnson’s Algorithm

- What if we have a sparse graph?
- All previous algorithms work on adjacency matrix
- Johnson’s algorithm works on adjacency lists
Reweighting Graphs

• Lemma: Given a graph $G$ with weight function $w$. Let $h$ be any function mapping vertices to reals. For each edge $(u,v) \in E$, define $w'(u,v) = w(u,v) + h(u) - h(v)$

We have the shortest paths using $w'$.

• Why: All $h$’s in path cancel except first and last ones
Idea

• Re-weight graph to eliminate negative weights on edges
• Repeatedly use Dijkstra’s algorithm to calculate distances
• Compute weighting by adding vertex s with 0-weighted edges to the remaining vertices
• Run Belman-Ford
• Use distances as h function
Johnson(G)

1 compute G’, where V[G’]=V[G] U \{s\},
   E[G’]=E[G] U \{(s,v):v \text{ in } V[G]\}, and
   w(s,v)=0 \text{ for all } v \text{ in } V[G]
2 if BELLMAN-FORD(G’, w, s)=FALSE
3 then print “negative cycle”
4 else for each vertex v in V[G’]
5     do set h(v) to value of d[s,v] computed in line 2
6     for each edge (u,v) in E[G’]
7     do w’(u,v) <- w(u,v) +h(u) - h(v)
8     for each vertex v in V[G]
9     do run DIJKSTRA(G, w’, u) to compute d’(u,v)
10    for each vertex v in V[G]
11    do d_{uv} <- d’(u,v) +h(v) – h(u)
12 return D