CISC 320 Introduction to Algorithms  
Fall 2005

Lecture 15
Single Source Shortest Paths (Dijkstra’s Algorithm)  
All Pairs Shortest Paths (Floyd-Warshall Algorithm)

---

Dijkstra’s Shortest-Path Algorithm

Single-source shortest path Problem  
Given a weighted graph \( G = (V, E, W) \) and a source vertex \( s \), find a shortest path from \( s \) to each vertex \( v \).

Growing a shortest-path tree
- Start at source vertex \( s \) and “branches out” by selecting edges that lead to new vertices
- For each vertex \( z \) in the fringe, there is at least one tree vertex \( v \) such that \( vz \) is an edge of \( G \). why?
  (otherwise how can \( z \) be in the fringe)
- Choose \( v \) such that \( d(s, v) + W(vz) \) is minimized

---

Theorem
Let \( G = (V,E,W) \), \( V' \) is a subset of \( V \), and \( V' \) contains the source \( s \). Let \( d(s,y) \) be the shortest distance in \( G \) from \( s \) to \( y \), for each \( y \) in \( V' \). If edge \( yz \) is chosen to minimize \( d(s,y) + W(yz) \) over all edges with \( y \) in \( V' \) and \( z \) in \( V-V' \), then the path consisting of a shortest path from \( s \) to \( y \) followed by the edge \( yz \) is a shortest path from \( s \) to \( z \) in \( V \).

Proof: For any other path \( P' \) from \( s \) to \( z \), we have

\[
W(P) = d(s,y) + W(yz) \leq d(s,r) + W(rt) \leq W(P')
\]

That is why vertex \( z \) is chosen by the algorithm to expand \( V' \)

---

Dijkstra Algorithm
Graph \( G \), weights \( w \), source \( s \)

Dijkstra(G,w,s)
1. Initialize all vertices as unseen
2. Start the tree with the specified source vertex \( s \); reclassify it as tree
3. Define \( d[s,s] = 0 \)
4. Reclassify all vertices adjacent to \( s \) as fringe
5. while there are fringe vertices
6. select a tree vertex \( t \) and a fringe vertex \( v \)
7. such that \( d(s,t)+W(tv) \) is minimum
8. reclassify \( v \) as tree; add edge \( tv \) to the tree
9. define \( d[s,v] = d[s,t]+W(tv) \)
10. reclassify all unseen vertices adjacent to \( v \) as fringe
Dijkstra Algorithm

Time analysis
- Initialization of priority queue (as a binary heap): \(O(V)\)
- Extract-min is called \(|V|\) times
  - Each Extract-Min takes \(O(\log V)\) time
  - For each adjacent vertex, update its distance (with Decrease-Key operation: \(O(\log v)\))

Total running time: \(O((V+E)\log(V))\)

Note: weights have to be non-negative.

---

All-Pairs Shortest Paths

For all pairs in a graph,
- Is there a path from \(u\) to \(v\)?
- What is the shortest path from \(u\) to \(v\)?

\(W\) is the weight matrix of graph \(G = (V,E,W)\).

\[ w_{ij} = \begin{cases} W(v_i v_j) & \text{if } i \neq j \text{ and } v_i v_j \in E \\ \infty & \text{if } i \neq j \text{ and } v_i v_j \notin E \\ 0 & \text{if } i = j \end{cases} \]

\(D\) with entry \(d_{ij}\) is the shortest-path distance from \(v_i\) to \(v_j\).

---

Lemma: If a shortest path from \(v_i\) to \(v_j\) goes through an intermediate vertex \(v_k\), then the segments of that path from \(v_i\) to \(v_k\), and from \(v_k\) to \(v_j\) are themselves shortest paths.

Path \(p\) is the shortest path from \(i\) to \(j\). Path \(p_1\), the portion of path \(p\) from \(i\) to \(k\), is the shortest path from \(i\) to \(k\). Suppose \(p'1\) is the shortest path from \(i\) to \(k\), then \(p'1\) and \(p\)
Floyd-Warshall algorithm

Floyd-Warshall(W)
1. \( D^{(0)} \leftarrow W \)
2. for \( k \leftarrow 1 \) to \( n \)
   3. for \( i \leftarrow 1 \) to \( n \)
      4. for \( j \leftarrow 1 \) to \( n \)
         5. \( D^{(k)}[i][j] \leftarrow \min(D^{(k-1)}[i][j], D^{(k-1)}[j][k] + D^{(k-1)}[k][j]) \)
6. return \( D^{(n)} \)

Time analysis:
\( \Theta(n^3) \)

Transitive closure of a directed graph
- Transitive closure is also called reachability relation. R matrix is a \( n \times n \) matrix
  \( r_{ij} = 1 \) if there is a path from \( s_i \) to \( s_j \)
  \( r_{ij} = 0 \) otherwise

Transitive-Closure(G)
1. for \( i \leftarrow 1 \) to \( n \)
   2. for \( j \leftarrow 1 \) to \( n \)
      3. if \( i = j \) or \((i, j) \in E[G]\) then \( r^{(0)}[i][j] \leftarrow 1 \)
      4. else \( r^{(0)}[i][j] \leftarrow 0 \)
   5. for \( k \leftarrow 1 \) to \( n \)
      6. for \( i \leftarrow 1 \) to \( n \)
         7. for \( j \leftarrow 1 \) to \( n \)
            8. \( r^{(k)}[i][j] \leftarrow r^{(k-1)}[i][j] \lor (r^{(k-1)}[i][k] \land r^{(k-1)}[k][j]) \)
   9. return \( r^{(n)} \)

Find a connected component of a graph
- Undirected graph
  - Corresponds to a depth-first search tree
- Directed graph
  - A depth-first search tree does not guarantee to give a connected component
  - A connected component \( \neq \) Transitive closure
    - For each vertex \( u \), do DFS starting \( u \), update \( r_{uv} \) for all reachable vertex \( v \).
    - Running time: \( O(V^2 + VE) \)

Define \( t^{(k)}_{ij} = 1 \) if there exists a path in graph \( G \) from vertex \( i \) to vertex \( j \) with all intermediate vertices in the set \( \{1, 2, \ldots, k\} \), and 0 otherwise.

\[
\begin{align*}
  t^{(0)}_{ij} &= 1 \text{ if } i = j \text{ or } (i, j) \in E \\
  &= 0 \quad \text{if } i \neq j \text{ and } (i, j) \notin E \\
  t^{(k)}_{ij} &= t^{(k-1)}_{ij} \lor (t^{(k-1)}_{ik} \land t^{(k-1)}_{kj})
\end{align*}
\]