Graph Algorithms

- Topological Sort
- Single Source Shortest Path
- All Pairs Shortest Path

Topology Sort

- Input: Directed Acyclic Graph (DAG)
- Output: Vertices of G sorted to satisfy the following condition: if there is a path from u to v in G then u occurs after v in the list.

Topological Sort – Take 1

\[
\text{TopoSort}(G) \\
\text{If } G \text{ is empty return} \\
\text{Choose vertex } v \text{ with in-degree } 0. \\
\text{Return } v, \text{ TopoSort}(G-v)
\]

Topological Sort – Take 2

\[
\text{TopoSort}(G) \\
\text{Perform DFS with timestamp} \\
\text{Output vertices by ____________________}
\]

Single Source Shortest Path

- Input: Graph G with a designated start vertex s.
- Output: For each vertex v, the distance between s and v.
Def: Path Length, Distance

• In an unweighted graph
  – the length of a path is the number of edges in the path
  – the distance between two vertices is the length of a shortest path between the vertices
• We use \( d_G(u,v) \) to denote the distance between \( u \) and \( v \) in \( G \)

Example: Distance

\[
d_G(s,s) = 0, \quad d_G(s,u) = 1, \quad d_G(s,v) = 2
\]

Shortest path tree for \( s \)
A spanning tree of \( G \) such that the path between \( s \) and a vertex \( v \) in \( T \) is a shortest path in \( G \).

Does such a tree always exist?

Proof of Existence:
Shortest path tree for \( s \)

• Order the vertices of \( G \) by distance from \( s \): \( v_0 = s, v_1, v_2, \ldots, v_n \)
• Claim: There is a subtree \( T \) of \( G \) on vertices \( \{v_0, \ldots, v_i\} \) such that for every \( v_j \),
  \[ d_T(s,v_j) = d_G(s,v_j) \]

Base Case

• When \( i = 0 \) the claim holds.

Inductive Hypothesis

• There is a tree \( T \) such that \( d_T(s,v) = d_G(s,v) \) for each \( v \) in \( \{s=v_0, \ldots, v_{i+1}\} \)
Now add \( v_i \)

- We need to exhibit \( T' \) such that \( d(s,v_i) = d_G(s,v_i) \) for each \( v \) in \( \{s,v_0, \ldots, v_{i-1}, v_i \} \).

\[ \begin{align*}
T & \quad \bullet v_1 \\
v_2 & \quad \bullet v_3 \\
v_4 &
\end{align*} \]

Observe

- Let \( s, \ldots, u, v_i \) be a shortest path between \( s \) and \( v_i \) in \( G \).
- Since \( i > 0 \) \( u \neq v_i \).
- Thus \( d_G(s,v_i) = 1 + d_G(s,u) > d_G(s,u) \).
- Therefore \( u \) precedes \( v_i \) in the ordering of vertices; i.e. \( u = v_j \) for some \( j < i \).

\[ \begin{align*}
T' & = T + (u,v_i) \\
v_i & \quad \bullet v_4
\end{align*} \]

Proof of Claim

- So \( u \) is already in \( T \) and, by our induction hypothesis, \( d_T(s,u) = d_G(s,u) \).

\[ \begin{align*}
T & \quad \bullet u \\
v_2 & \quad \bullet v_3 \\
v_1 & \quad \bullet v_4
\end{align*} \]

\[ \begin{align*}
T' & = T + (u,v_i) \\
v_i & \quad \bullet v_4
\end{align*} \]

\[ \begin{align*}
T' & = T + (u,v_i) \\
v_i & \quad \bullet v_4
\end{align*} \]

QED

\[ \begin{align*}
T & \quad \bullet v_2 \\
v_1 & \quad \bullet v_4
\end{align*} \]

Shortest path tree for \( s \)

The path between \( s \) and \( v \) in \( T \) is a shortest path in \( G \).

The edges traversed in Breadth-First(s) form shortest path tree for \( s \).

\[ \begin{align*}
S & \quad \bullet v_1 \\
& \quad \bullet v_2 \\
& \quad \bullet v_3 \\
& \quad \bullet v_4
\end{align*} \]

Breadth-first(s) tree:

All vertices and the edges traversed

\[ \begin{align*}
S & \quad \bullet v_1 \\
& \quad \bullet v_2 \\
& \quad \bullet v_3 \\
& \quad \bullet v_4
\end{align*} \]
Breadth-first(s)

Q=s

Breadth-first(s)

Q=2,3,4

Breadth-first(s)

Q=3,4,5,6

Breadth-first(s)

Q=4,5,6,7

Breadth-first(s)

Q=5,6,7,8

Single Source Shortest Path

• Input: Graph G with a designated start vertex s.
• Output: For each vertex v, the length of the shortest path between s and v.
• Algorithm: Modify Breadth-first to compute d(s,v) along the way.
Breadth-first search
Compute distance from s

Running Time
• The modified breadth-first algorithm for single source shortest path in an unweighted graph is: ___________________

What if G is weighted?
Single Source Shortest Path

- Input: Weighted graph G with a designated start vertex s. Weights are positive!
- Output: For each vertex v, the length of the shortest path between s and v.

Path Length

- In a weighted graph the length of a path is the sum of the weights of the edges of the path.

Shortest path tree for s

The path between s and v in T is a shortest path in G.

Shortest path tree for s

- Is it clear such a tree exists? YES by same argument.

Shortest path tree for s

- Claim: Let the vertices of G be sorted by distance from s. Then there is a subtree T of G on vertices \{v_0, \ldots, v_i\} such that \(d_T(s,v) = d_G(s,v)\) for each v in T.
- If you have a tree for \{v_0, \ldots, v_i\} can you find the tree for \{v_0, \ldots, v_{i+1}\}?

Shortest path tree for s

- Suppose you don’t know the ordering?
- At each step find the vertex v \(\not\in T\) that minimizes \(\min_{u\not\in T} d_T(s,u)+w(u,v)\).

11/07/01  CS140 Lec 15
Dykstra’s Algorithm

Number in node $u$ indicate $d_G(s,u)$
Dykstra’s Algorithm

Number in node u indicate $d_G(s,u)$

Does this sound familiar?
- Prim’s algorithm for MST is VERY similar.
- The implementation details are almost identical.

All Pairs Shortest Path
(directed version)
- Input: Weighted digraph $G$
- Output: For each pair of vertices $x, y$ the distance between $x$ and $y$ in $G$
What is $d(b,a)$?

Three Cases:
- There is no path from $a$ to $b$
- There is a path from $a$ to $b$ but no shortest path
- There is a shortest path from $a$ to $b$

Shortest path algorithm should
- Determine which case holds
  - There is no path from $a$ to $b$
  - There is a path from $a$ to $b$ but no shortest path
  - There is a shortest path from $a$ to $b$
- Find the length of the shortest path when one exists

All Pairs Shortest Path
Inductive definition
K-limited paths

- A path from \( v_i \) to \( v_j \) is k-limited if the intermediate vertices in the path are numbered k or less

3-limited paths

```
  o-------o       3
 /       /        /  \
 v       v        v
  o-------o       2
 /       /        /  \
 v       v        v
  o-------o       1
```

“No exit” nodes

What is the shortest 5-limited path from \( v_1 \) to \( v_2 \)?

```
  o-------o       5
 /       /        /  \
 v       v        v
  o-------o       4
 /       /        /  \
 v       v        v
  o-------o       3
 /       /        /  \
 v       v        v
  o-------o       2
 /       /        /  \
 v       v        v
  o-------o       1
```

Floyd-Warshall algorithm

- \( D^k(i,j) \) is the length of a shortest k-limited path from \( v_i \) to \( v_j \)
- \( D^k(i,j) = \min(D^{k-1}(i,j), D^{k-1}(i,k) + D^{k-1}(k,j)) \)
- \( D^0(i,j) = w(<v_i,v_j>) \)

\( D^0 \)

```
\begin{array}{c|ccccc}
 i & 1 & 2 & 3 & 4 & 5 \\
\hline
 1 & 0 & \infty & \infty & 2 & \infty \\
 2 & \infty & 0 & \infty & \infty & 3 \\
 3 & \infty & \infty & 0 & \infty & \infty \\
 4 & \infty & \infty & 4 & 0 & 4 \\
 5 & \infty & \infty & 5 & \infty & 0 \\
\end{array}
```
\[ D^1(i,j) = \min(D^0(i,j), D^0(i,1) + D^0(1,j)) \]

\[ D^2(i,j) = \min(D^1(i,j), D^1(i,2) + D^1(2,j)) \]

And so on …
And so on …

**Floyd-Warshall algorithm**

- $D^0(i,j) = w(v_i, v_j)$
- For $i=1$ to $n$
  - Compute $D^i$ from $D^{i-1}$
- Return $D^n$

**Running Time**

- $n$ Tables
- Each is $n \times n$
- Each table entry takes $O(1)$
- $O(n^3)$

![Diagram of the Floyd-Warshall algorithm](image)