G205
Fundamentals of Computer Engineering
CLASSES 18, Wed. Nov. 10 2004
Stefano Basagni
Fall 2004
M-W, 1:30pm-3:10pm
All-Pairs Shortest Paths

Finding shortest path between all pairs of vertices in a graph

Input:
- \( G=(V,E) \)
- \( w:E \rightarrow R \)

Output: For each pair of vertices \( u \) and \( v \) in \( V \) we want the least weight path from \( u \) to \( v \)
Representation of Input

For APSP graph represents by an adjacency matrix \( W = (w_{ij}) \)

- \( w_{ij} = 0 \) if \( i = j \)
- \( w_{ij} = w(i,j) \) if \( i \neq j \) and \((i,j) \in E\)
- \( w_{ij} = \infty \) if \( i \neq j \) and \((i,j) \notin E\)

Negative-weight edges \( \rightarrow \) OK
Negative-weight cycles \( \rightarrow \) not OK
Representation of Output

- n x n matrix \( D = (d_{ij}) \)
  - \( d_{ij} \) = shortest path weight from \( i \) to \( j \)
  - At termination: \( d_{ij} = d(i,j) \)

- Actual shortest paths: Predecessor matrix \( \Pi = (\pi_{ij}) \)
  - \( \pi_{ij} \) = NIL if \( i=j \) or there is no path from \( i \) to \( j \)
  - \( \pi_{ij} \) = predecessor of \( j \) on some path from \( i \) to \( j \) otherwise
Predecessor Subgraph

- The subgraph induced by the i-th row of the matrix $\Pi$ is a shortest path tree with root $i$.

- For each vertex $i \in V$ we define the predecessor subgraph $G_{\pi,i} = (V_{\pi,i}, E_{\pi,i})$:
  - $V_{\pi,i} = \{ j \in V : \pi_{ij} \neq \text{NIL} \} \cup \{i\}$
  - $E_{\pi,i} = \{ (\pi_{ij}, j) : j \in V_{\pi,i} \backslash \{i\} \}$
Printing APSPs

\textbf{Print-APSP}(\Pi,i,j)\
if i = j then print i\
else if \(\pi_{ij} = \text{NIL}\)\
then print no i-j path\
else Print-APSP(\Pi,i,\pi_{ij})\
print j
Some Notation

- Graph $G$ has $|V|=n$ vertices
- Matrix are denoted in uppercase $D,W,L$
- Matrix elements: $d_{ij}$, $w_{ij}$, $l_{ij}$
- Iterates of matrices: $D^{(m)} = (d_{ij}^{(m)})$
Shortest Paths and Matrix Multiplication

Dynamic Programming approach:

- Characterize the structure of an optimal solution
- Define its value recursively
- Compute a solution in a bottom-up fashion
- Constructing an optimal solution
Structure of a Shortest Path

- All subpaths of a shortest path are shortest paths.
- Graph represented by adjacency matrix $W = (w_{ij})$.
- Let $p$ be a shortest path with at most $m$ edges.
- If $i = j$ then $p$ has no edges and weight 0.
- If $i \neq j$ then $p = i \sim k \sim j$ with $i \sim k$ with at most $m - 1$ edges and $d(i, j) = d(i, k) + w_{kj}$. 
Recursive Solution

$\ell^{(m)}_{ij}$ is the minimum weight of any path from $i$ to $j$ with at most $m$ edges

When $m=0$

- $\ell^{(0)}_{ij} = 0$ if $i = j$
- $\ell^{(0)}_{ij} = \infty$ if $i \neq j$

When $m \geq 1$

- $\ell^{(m)}_{ij} = \min(\ell^{(m-1)}_{ij}, \min_{1 \leq k \leq n} \{\ell^{(m-1)}_{ik} + w_{ki}\}) = \min_{1 \leq k \leq n} \{\ell^{(m-1)}_{ik} + w_{ki}\}$ (since $w_{jj} = 0$ for each $j$)
The Shortest Path Weight

- No negative-weight cycles $\rightarrow$ shortest paths have at most $n-1$ edges
- A path from $i$ to $j$ for which $d(i,j) < \infty$ is simple and with $\leq n-1$ edges or otherwise it cannot have weight $\leq d(i,j)$
- Actual shortest path weight:
  $$d(i,j) = l^{(n-1)}_{ij} = l^{(n)}_{ij} = l^{(m+1)}_{ij} = \ldots$$
Computing weights bottom-up

- **Input:** $W = (w_{ij})$
- **We compute:** $L^{(1)}, L^{(2)}, \ldots, L^{(n-1)}$ with $L^{(m)} = (l^{(m)}_{ij}), m=1,2,\ldots,n-1$
- $L^{(n-1)}$ contains the shortest-path weights
- **By definition of $L^{(m)}$ is** $L^{(1)} = W$
- **Basic step:** Extending shortest paths edge by edge: Given $L^{(m-1)}$ and $W$ we obtain $L^{(m)}$
Extending Shortest Paths

Extend-Shortest-Paths(L,W)

for i = 1 to n do
    for j = 1 to n do
        \( l'_{ij} = \infty \)
        for k = 1 to n do
            \( l'_{ij} = \min(l'_{ij}, l_{ik} + w_{kj}) \)
    return \( L' \)
Matrix Multiplication Analogy

- Extend-Shortest-Paths costs $O(n^3)$
- It is like multiplying $n \times n$ matrices:
  - $C = A \times B \rightarrow c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$
- Here:
  - $l^{(m-1)} = a$
  - $w = b$
  - $l^{(m)} = c$
  - $\min = +$
  - $+ = \ast$
  - $\infty = 0$
Computing Shortest-Paths Weights

- We extend shortest paths edge by edge
- We compute the sequence:
  - \( L^{(1)} = L^{(0)} \times W = W \)
  - \( L^{(2)} = L^{(1)} \times W = W^2 \)
  - \( L^{(2)} = L^{(2)} \times W = W^3 \)
  - ...
  - \( L^{(n-1)} = L^{(n-2)} \times W = W^{n-1} \)
A Slow APSP algorithm

Slow-APSP(W)

\[ L^{(1)} = W \]

for \( m = 2 \) to \( n - 1 \) do

\[ L^{(m)} = \text{Extend-Shortest-Paths}(L^{(m-1)}, W) \]

return \( L^{(n-1)} \)

Since \( \text{Extend-Shortest-Paths} \) is \( O(n^3) \)

Slow-APSP(W) is \( O(n^4) \)
Improving the Running Time

Goal: Compute \( L^{(n-1)} \) and not the whole sequence \( L^{(1)}, L^{(2)}, \ldots, L^{(n-1)} \)

Recall: \( L^{(m)} = L^{(n-1)} \) for each \( m \geq n - 1 \)

Repeated Squaring:

- \( L^{(1)} = L^{(0)} \times W = W \)
- \( L^{(2)} = W^2 = W \times W \)
- \( L^{(4)} = W^4 = W^2 \times W^2 \)
- \( \ldots \)
- \( L^{(2\log(n-1))} = W^{2\log(n-1)} = W^{(2\log(n-1)-1)} \times W^{(2\log(n-1)-1)} = L^{(n-1)} \)
A Faster APSP Algorithm

Faster-APSP(W)

$L^{(1)} = W$

$m = 1$

while $m \leq n-1$ do

$L^{(2m)} = \text{Extend-Shortest-Paths}(L^{(m)}, L^{(m)})$

$m = 2m$

return $L^{(m)}$

\textbf{Faster-APSP(W) is } $O(n^3 \log n)$
The Floyd-Warshall Algorithm

Another dynamic programming formulation for All-Pairs Shortest Path

- The structure of a shortest path
  - Uses intermediate vertices of a shortest path
- Recursive solution to the ASPS problem
- Computing the shortest path weights bottom up
Structure of a Shortest Path, 1

- An intermediate vertex of a simple path $p = \langle v_1, v_2, \ldots, v_l \rangle$ is any vertex of $p$ in $\{v_2, \ldots, v_{l-1}\}$

- Consider $G = (V, E)$ with $V = \{1, \ldots, n\}$

- Consider $K = \{1, \ldots, k\}$, for some $k$

- For each $i, j \in V$ consider paths with vertices only from $K$

- Let $p$ be a shortest paths among them
Structure of a Shortest Path, 2

Relationship of p and the $i \sim j$ shortest path with vertices from $K-1=\{1, \ldots, k-1\}$

- k is not intermediate in p $\rightarrow$ the int. vertices of p are in $K-1$ $\rightarrow$ shortest path $i \sim j$ with vertices in $K-1$ has also vertices in K
- If k is an intermediate vertex in p then:
  $p=p_1p_2$ where $p_1=i \sim k$ and $p_2=k \sim j$ where $p_1$ and $p_2$ have int. vertices in $K-1$
Recursive Solution to ASPS

Let $d^{(k)}_{ij}$ the weight of a $i \sim j$ shortest path with all int. vertices in $K$

- $k = 0 \rightarrow d^{(0)}_{ij} = w_{ij}$
- $k \geq 1 \rightarrow \min\{d^{(k-1)}_{ij}, d^{(k-1)}_{ik} + d^{(k-1)}_{kj}\}$

Since for any path all intermediate vertices are in $V$ the matrix $D^{(n)} = (d^{(n)}_{ij})$ is such that $d^{(n)}_{ij} = d(i,j)$, for each $i, j \in V$
Computing the Shortest-Paths Weights Bottom Up

Floyd-Warshall(W)

\[ D^{(0)} = W \]

for \( k = 1 \) to \( n \) do

for \( i = 1 \) to \( n \) do

for \( j = 1 \) to \( n \) do

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

return \( D^{(n)} \)
Running Time and Space

- The running time is clearly $\Theta(n^3)$ since the min operation and the sum takes $O(1)$ time.
- Space needed is $\Theta(n^3)$: Each of the $n$ $D^{(k)}$ needs $\Theta(n^2)$ space.
- Dropping all superscript leads to a solution that works in $\Theta(n^2)$ space.
C++ implementation of Floyd-Warshall

```cpp
void FW( int n, matrix< int > &fw ) {
    matrix< int > t( n, n ) = fw;
    for( int k = 0; k < n; k++ ) {
        for( int i = 0; i < n; i++ )
            for( int j = 0; j < n; j++ )
                fw[ i ][ j ] = min( t[ i ][ j ],
                                    t[ i ][ k ] + t[ k ][ j ] );
        t = fw;
    }
}
```
Assignments

♦ Textbook, Chapter 25, pages 620—640

♦ Updated information on the class web page:

www.ece.neu.edu/courses/eceg205/2004fa