COMP 355
Advanced Algorithms

All-Pairs Shortest Paths
Floyd-Warshall Algorithm
Section 25.2 (CLRS): Not in KT

1

All-Pairs Shortest Paths

- Generalization of single-source shortest path: computing shortest path between all pairs of vertices
- Let G = (V, E) be a directed graph with edge weights.
- Find the cost of the shortest path between all pairs of vertices in G.

2

Possible Algorithms

- If no negative weights:
  - Run Dijkstra's with each vertex as the source
  - Runtime: O(VE lg V) (if we use binary min-heap implementation)
- If negative weights, but no negative cycles:
  - Run Bellman-Ford algorithm once from each vertex
  - Runtime: $O(V^2E)$ (on a dense graph = $O(V^4)$)
- Can we do better (assuming negative edges)?
  - Yes! $O(V^3)$ using dynamic programming

Rhodes College

Input/Output

- Input Format:
  - input is an *n x n* matrix *w* of edge weights, which are based on the edge weights in the digraph.
  - We let w_{ij} denote the entry in row *i* and column *j* of *w*.

$$w_{ij} = \begin{cases} 0 & \text{if } i = j, \\ w(i,j) & \text{if } i \neq j \text{ and } (i,j) \in E, \\ +\infty & \text{if } i \neq j \text{ and } (i,j) \notin E. \end{cases}$$

- Output Format:
  - n x n distance matrix D = d_{ij} where d_{ij} = δ(i, j), the shortest path from vertex i to vertex j.
  - To recover the actual shortest path, we can compute an auxillary matrix mid[i, j] where the value of mid[i, j] will be a vertex that is somewhere along the path from i to j. (null if no such vertex exists)

Rhodes College

Observations

- A shortest path does not contain the same vertex more than once.
- For a shortest path from i to j such that any intermediate vertices on the path are chosen from the set $\{1, 2, ..., k\}$, there are two possibilities:
  - 1. k is not a vertex on the path, so the shortest such path has length d_{ij}^{k-1}
  - 2. k is a vertex on the path, so the shortest such path is $d_{ik}^{k-1} + d_{kj}^{k-1}$
- So we see that we can recursively define $d_{ij}^{(k)}$ as

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

5

Floyd-Warshall Algorithm

Floyd-Warshall Algorithm

```
Floyd_Warshall(int n, int w[1..n, 1..n]) {
    array d[1..n, 1..n]
    for i = 1 to n do {                          // initialize
        for j = 1 to n do {
            d[i,j] = W[i,j]
            mid[i,j] = null
        }
    }
    for k = 1 to n do                            // use intermediates {1..k}
        for i = 1 to n do                        // ...from i
            for j = 1 to n do                    // ...to j
                if (d[i,k] + d[k,j]) < d[i,j]) {
                    d[i,j] = d[i,k] + d[k,j]     // new shorter path length
                    mid[i,j] = k                 // new path is through k
                }
    return d                                     // matrix of distances
}
```

Running Time: $\Theta(n^3)$
Space Required: $\Theta(n^2)$

Rhodes College

6

Example (on board)

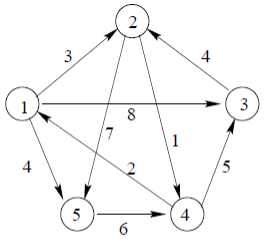


Figure 1: Example Graph

Consider the graph in Figure 1. For this graph, we would initialize D and P to be:

$$D = \begin{pmatrix} 0 & 3 & 8 & \infty & 4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & 5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad P = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & \text{NIL} & 4 & \text{NIL} & \text{NIL} \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

and our final values for D and P are:

$$D = \begin{pmatrix} 0 & 3 & 8 & 4 & 4 \\ 3 & 0 & 6 & 1 & 7 \\ 7 & 4 & 0 & 5 & 11 \\ 2 & 5 & 5 & 0 & 6 \\ 8 & 11 & 11 & 6 & 0 \end{pmatrix} \quad P = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ 4 & \text{NIL} & 4 & 2 & 2 \\ 4 & 3 & \text{NIL} & 2 & 2 \\ 4 & 1 & 4 & \text{NIL} & 1 \\ 4 & 1 & 4 & 5 & \text{NIL} \end{pmatrix}$$

Rhodes College

Floyd-Warshall Algorithm: Example

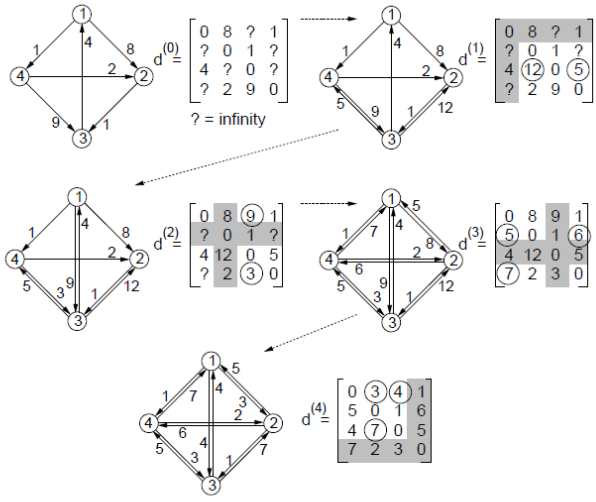
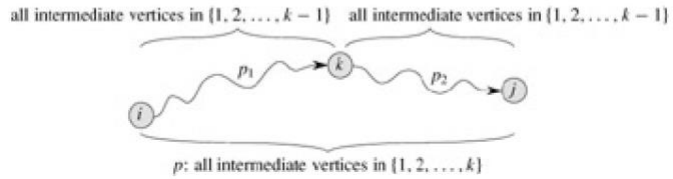


Fig. 42: Floyd-Warshall Example. Newly updates entries are circled.

Proof of Correctness

Inductive Hypothesis

Suppose that prior to the kth iteration it holds that for $i, j \in V$, d_{ij} contains the length of the shortest path Q from i to j in G containing only vertices in the set $\{1, 2, ..., k-1\}$, and π_{ij} contains the immediate predecessor of j on path Q.



Rhodes College

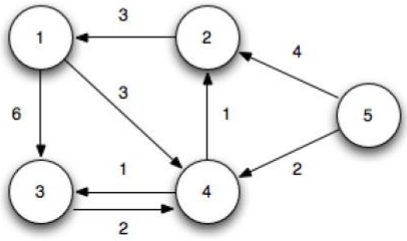
Applications

- Detecting the Presence of a Negative Cycle
- Transitive Closure of a Directed Graph

Rhodes College

Other All-Pairs Shortest Paths Algorithms

- Dynamic Programming Approach Based on Matrix Multiplication
- Johnson's Algorithm for Sparse Graphs

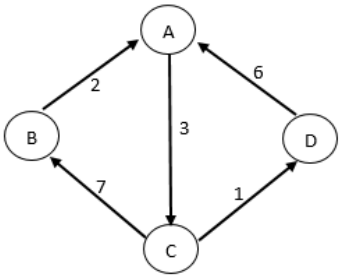


Rhodes College

11

Practice

Apply the Floyd-Warshall algorithm, which finds the shortest paths and their lengths, to the following graph.



Apply the Bellman-Ford algorithm to this graph assuming that the start node is A.

12