All Pairs Shortest Path

Floyd-Warshall Algorithm

Joseph Chuang-Chieh Lin (林莊傑)

Department of Computer Science & Engineering,
National Taiwan Ocean University

Fall 2024

Outline

Outline

1. Introduction
2. Floyd-Warshall algorithm
3. Transitive Closure

All Pairs Shortest Paths

Goal

Find the shortest path between all pairs of vertices in the graph.

- Naïve approach:

All Pairs Shortest Paths

Goal

Find the shortest path between all pairs of vertices in the graph.

- Naïve approach:
  Running Bellman-Ford algorithm for n vertices

All Pairs Shortest Paths

Goal

Find the shortest path between all pairs of vertices in the graph.

- Naïve approach:
  Running Bellman-Ford algorithm for n vertices $\Rightarrow$

All Pairs Shortest Paths

Goal

Find the shortest path between all pairs of vertices in the graph.

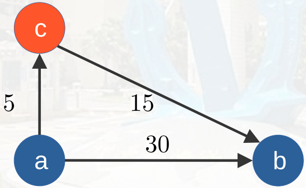
- Naïve approach:
  Running Bellman-Ford algorithm for n vertices $\Rightarrow n \cdot O(ne) = O(n^2 e)$ time.
  - $|V| = n, |E| = e$.

Outline

Floyd-Warshall algorithm ($O(n^3)$ time)

Idea

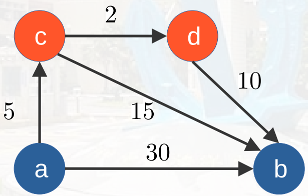
We can gradually consider "all" intermediate routes between two vertices i and j.



Floyd-Warshall algorithm ($O(n^3)$ time)

Idea

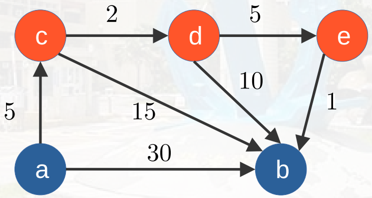
We can gradually consider "all" intermediate routes between two vertices i and j.



Floyd-Warshall algorithm ($O(n^3)$ time)

Idea

We can gradually consider "all" intermediate routes between two vertices i and j.



Floyd-Warshall algorithm (contd.)

Idea

Compute $\text{dist}^k[i][j]$: shortest path (length) from i to j routing through vertices $\{0, 1, \ldots, k-1, k\}$.

Floyd-Warshall algorithm (contd.)

Idea

Compute $\text{dist}^k[i][j]$: shortest path (length) from i to j routing through vertices $\{0, 1, \ldots, k-1, k\}$.

- Initialization: $\text{dist}^0[i][j] = \text{length}[i][j]$ for all i, j.

Floyd-Warshall algorithm (contd.)

Idea

Compute $\text{dist}^k[i][j]$: shortest path (length) from i to j routing through vertices $\{0, 1, \ldots, k-1, k\}$.

- Initialization: $\text{dist}^0[i][j] = \text{length}[i][j]$ for all i, j.
- For $k \geq 1$:

Floyd-Warshall algorithm (contd.)

Idea

Compute $\text{dist}^k[i][j]$: shortest path (length) from i to j routing through vertices $\{0, 1, \ldots, k-1, k\}$.

- Initialization: $\text{dist}^0[i][j] = \text{length}[i][j]$ for all i, j.
- For $k \geq 1$:

$$\text{dist}^k[i][j] = \min\{\text{dist}^{k-1}[i][j], \text{dist}^{k-1}[i][k] + \text{dist}^{k-1}[k][j]\}.$$

- Effectively reuse the calculated information (DP).

Initial Setup

```
for (i=0; i<n; i++) {
    for (j=0; j<n; j++) {
        dist[i][j] = length[i][j]
        if (length[i][j] != INT_MAX) {
            next[i][j] = j; // for constructing the path
            // next step from i to j: go to j
        }
    }
}
```

The Main Function of FW

```
for (k=0; k<n; k++) {
    for (i=0; i<n; i++) {
        for (j=0; j<n; j++) {
            if (dist[i][k] + dist[k][j] < dist[i][j]) {
                dist[i][j] = dist[i][k] + dist[k][j];
                next[i][j] = next[i][k];
                // next step from i to j: go to k
            }
        }
    }
}
// Then we deal with negative cycles!
dist = catchNegativeCycles(dist, n);
```

Dealing with negative cycles

We need to "propagate" the negative cycles.

```
catchNegativeCycles(dist, n) {
    for (k=0; k<n; k++) {
        for (i=0; i<n; i++) {
            for (j=0; j<n; j++) {
                if (dist[i][k] + dist[k][j] < dist[i][j]) {
                // the distance is still updatable by a negative cylcle
                    dist[i][j] = -INT_MAX;
                    next[i][j] = -1
                }
            }
        }
    }
    return dist; // return the final distance matrix
}
```

Path Reconstruction

```
reconstructPath(dist, next, start, end) {
    node queue path[n];
    if (dist[start][end] == INT_MAX) return path;

    int current = start;
    while (current != end) {
        if (current == -1) return NULL;
        else
            enqueue(path, current)
        current = next[current][end]
    }
    if (next[current][end] == -1) return NULL;
    else enqueue(path, end);
    return path;
}
```

Outline

1. Introduction
2. Floyd-Warshall algorithm
3. Transitive Closure

Transitive Closure (1/3)

Question

Given a directed graph G with *unweighted* edges, determine if there is a path from i to j for all vertex pairs i, j.

Transitive Closure (1/3)

Question

Given a directed graph G with *unweighted* edges, determine if there is a path from i to j for all vertex pairs i, j.

- case (i): positive path lengths;
- case (ii): nonnegative path lengths.

Transitive Closure (2/3)

Transitive closure matrix

The transitive closure matrix A^+ of a directed graph G is a matrix such that $A^+[i][j] = 1$ if there is a path of length > 0 from i to j (otherwise $A^*[i][j] = 0$).

Reflexive transitive closure matrix

The reflexive transitive closure matrix A^* of a directed graph G is a matrix such that $A^*[i][j] = 1$ if there is a path of length ≥ 0 from i to j (otherwise $A^*[i][j] = 0$).

Transitive Closure (3/3)

- We can utilize the all-pairs-shortest-path algorithm (FW).
- Note:
  - $\text{length}[i][j] = 1$ if (i, j) is an edge in G.
  - $\text{length}[i][j] = +\infty$ if (i, j) is not in G.
- Change the if-statement in FW algorithm (distance updating):

```
dist[i][j] = (dist[i][j] || dist[i][k] && dist[k][j])
```

- Initialize the distance to be the adjacency matrix of the graph.

Discussions