All Sources Shortest Path: The Floyd-Warshall Algorithm

All Sources Shortest Path Problem

Shortest path network.

- Directed graph $G = (V, E, w)$.
- Weight w_e = weight of edge e.

Shortest path problem: for all pairs of vertices (u,v), find shortest directed path from u to v.

Option #1: if all edge weights are non-negative, just run Dijkstra n times ($n = |V|$)

- Each iteration of Dijkstra takes $O(n^2)$ for array-based or $O(m \log n)$ for heap-based
- Total complexity is either $O(n^3)$ or $O(mn \log n)$
- This is a case where just repeatedly using a solution to a simpler problem works out fine.

All Sources Shortest Path Problem

Shortest path network.

- Directed graph $G = (V, E, w)$.
- Weight w_e = weight of edge e.

Shortest path problem: for all pairs of vertices (u,v), find shortest directed path from u to v.

Option #2: if some edge weights are negative

- Use Bellman-Ford single source shortest path algorithm
  - However, it has complexity $O(nm)$ for a single source.
  - So all sources solution is $O(n^2m)$, which is $O(n^4)$ for dense graphs

All Sources Shortest Path Problem

Shortest path network.
- Directed graph $G = (V, E, w)$.
- Weight w_e = weight of edge e.

Shortest path problem: for all pairs of vertices (u,v), find shortest directed path from u to v.

Option #3: if some edge weights are negative
- Floyd-Warshall algorithm
  - Dynamic programming solution to compute all sources shortest paths
  - Works with negative weights (or without) – we assume no negative cycles, however
  - Complexity $O(n^3)$

Floyd-Warshall Algorithm

Floyd-Warshall algorithm.

- Build a 3-dimensional dynamic programming array (hence the $O(n^3)$ complexity) that keeps track of the shortest path between any two vertices, using only some subset of the entire collection of vertices as intermediate steps along the path.

- S[i][j][k] := shortest path from vertex i to vertex j, using only vertices 1,2,…,k as intermediate vertices along the path

- Solution: for each i,j: S[i][j][n] *gives* the shortest path from i to j allowing all vertices at intermediate steps

Floyd-Warshall Algorithm

Floyd-Warshall algorithm.

- Number the vertices 1,2,…,n
- Consider subset 1,2,…,k

- For any i,j ∈ V, consider all paths from i to j whose intermediate vertices are restricted to 1,2,…,k
  - Let p be a shortest path among these.
  - p is a simple path (it has no cycles)
    - All cycles are assumed non-negative weight,
      - There can't be a positive weight cycle in a shortest path (we could just remove it and have a better path)
      - Any zero-weight cycle can be removed without affecting the shortest path
    - This means each vertex appears at most once along path p

Floyd-Warshall Algorithm

Floyd-Warshall algorithm.

- Recurrence relation: consider if vertex k is part of path p
  - If not, then all intermediate vertices are in 1,…,k-1, so the best solution for shortest path from i to j using 1,2,…k will be the same as using 1,2,…,k-1
  - If yes, p can be split into two subpaths – p_1, the path from i to k, and p_2, the path from k to j
    - p_1 and p_2 are themselves shortest paths
      - Why? If not, we could form a better path from i to k than the path p by using the better subpaths
  - Optimal subproblems:
    - p_1 is a shortest path from i to k using 1,2,…,k-1 (because no vertex is used twice in the simple path)
    - p_2 is a shortest path from k to j using 1,2,…,k-1

Floyd-Warshall Algorithm

Floyd-Warshall algorithm.

- Recurrence relation

- $S[i][j][k] = \min(S[i][j][k-1], S[i][k][k-1] + S[k][j][k-1])$ for $k > 0$

- $S[i][j][0] = w_{ij}$ if there is an edge e directed from i to j
  0 if $i=j$
  ∞ otherwise

- We can build up from the bottom, considering more and more vertices along the intermediate path

Floyd-Warshall Algorithm

Floyd-Warshall algorithm.

- Complexity: $O(n^3)$, small constant factor makes practical use possible even for moderate size of n
- Initialization weights w(i,j) are 0 if i=j and ∞ if no edge exists

```
Floyd-Warshall ( G=(V,E,w) )

For i=1 to |V| do
    For j=1 to |V| do
        S[i,j,0] = w(i,j)
For k=1 to |V| do
    For i=1 to |V| do
        For j=1 to |V| do
            S[i,j,k] = min {
                S[i,j,k-1],
                S[i,k,k-1]+S[k,j,k-1] }
Return S[:,:,n]  # return 2d array with n = |V|
```