Data Structures and Algorithms

Week 11 - Bellman-Ford SSSP Algorithm, Floyd-Warshalls APSP

Subodh Sharma and Rahul Garg

{svs,rahulgarg}@iitd.ac.in.

Bellman-Ford

The algorithm

```
Bellman-ford(source, V, E):
  // initialise d[v], p[v], and d[source]=0
  for |V| -1 times repeat:
    foreach (u,v) ∈ E with weight c_uv:
      alt = d[u] + c_uv
      If (alt < d[v]):
        d[v] = alt
        p[v] = u
```

```
foreach (u,v) ∈ E with weight c_uv:
  if d[u] + c_uv <  d[v]
    Error "Negative wt cycle found"
```

Time complexity: $O(|V|.|E|)$

Bellman-Ford

Proof of correctness

- Lemma 1: The longest path without a cycle in G = (V, E) can be of almost $|V| - 1$ edges.
  - Proof: Assume there is a path in G w/o a cycle that has length $|V|$. This means it has $|V| + 1$ vertices.
    - By Pigeonhole principle, at least one vertex is repeated ⇒ the path has a cycle. This is a contradiction.
- Lemma 2: Assume a G with no negative cycles. Let $p = \langle v_0, v_1, \ldots, v_j \rangle$ be the shortest path from v_0 to v_j. Any sequence of calls that include in-order relaxations of $(v_0, v_1), (v_1, v_2), \ldots, (v_{j-1}, v_j)$ produces $d[v_j] = \delta[v_j]$ after all the relations and at all times afterwards.

Bellman-Ford

Proof of correctness

- Lemma 2: Assume a G with no negative cycles. Let $p = \langle v_0, v_1, \ldots, v_j \rangle$ be the shortest path from v_0 to v_j. Any sequence of calls that include in-order relaxations of $(v_0, v_1), (v_1, v_2), \ldots, (v_{j-1}, v_j)$ produces $d[v_j] = \delta[v_j]$ after all the relations and at all times afterwards.
  - Proof: Base case trivial. Assume IH until k-1 and $d[v_{k-1}] = \delta[v_{k-1}]$. Eventually we will relax the edge (v_{k-1}, v_k) at least once after the call (v_{k-2}, v_{k-1}). At the time of this call $d[v_k] \geq \delta[v_k]$. We also know that $\delta[v_k] = \delta[v_{k-1}] + w(v_{k-1}, v_k)$ because this path is the shortest. Therefore, $d[v_k] \geq d[v_{k-1}] + w(v_{k-1}, v_k)$. After the relaxation call, $d[v_k] = d[v_{k-1}] + w(v_{k-1}, v_k) = \delta[v_k]$.

Bellman-Ford

Proof of correctness

- Thm1: For all $v \in V$ reachable from s, Bellman-Ford produces $d[v] = \delta[v]$
  - Proof: Let $p = \langle v_0 = s, v_1, \ldots, v_j = v \rangle$ be the shortest acyclic path. p cannot contain more than $|V| - 1$ edges (Lemma 1). Assuming we consider a relaxation sequence where (s, v_1) gets relaxed in the 1st iteration, then (v_1, v_2) in the next and so on.
    - From Lemma 2, it implies that after $|V| - 1$ many iterations $d[v] = \delta[v]$.

All-pairs Shortest Path Algorithm

Also called Floyd-Warshall

- On directed weighted graphs with **no** negative weight cycles
- The algorithm finds uses in computing **Transitive closure** of a relation and **widest path problems**
- **ShortestPath**(i, j, k): returns the length of the shortest path from i to j using vertices only from the set $\{1,2,\ldots,k\}$ as intermediate points.
  - Note: ShortestPath$(i, j, k) \leq$ ShortestPath$(i, j, k-1)$. **Think Why?**
  - Now, if ShortestPath$(i, j, k) <$ ShortestPath$(i, j, k-1)$, then there must exist a path from i to j using vertices $\{1,2,\ldots,k\}$ that is shorter than any path that does not use k

All-pairs Shortest Path Algorithm

Also called Floyd-Warshall

- **ShortestPath**(i, j, k): returns the length of the shortest path from i to j using vertices only from the set $\{1,2,\ldots,k\}$ as intermediate points.
  - Note: ShortestPath$(i, j, k) \leq$ ShortestPath$(i, j, k-1)$. **Think Why?**
  - Now, if ShortestPath$(i, j, k) <$ ShortestPath$(i, j, k-1)$, then there must exist a path from i to j using vertices $\{1,2,\ldots,k\}$ that is shorter than any path that does not use k
    - This path can be decomposed as ShortestPath(i, j, k) = ShortestPath$(i, k, k-1)$ + ShortestPath$(k, j, k-1)$
- **Recursive formulation:**
  ShortestPath$(i, j, k) = \min($ShortestPath$(i, j, k-1),$ ShortestPath$(i, k, k-1) +$ ShortestPath$(k, j, k-1))$

All-pairs Shortest Path Algorithm

Also called Floyd-Warshall

- **Base case:** ShortestPath$(i, j, 0)$ = $w(i, j)$ where $w(i, j)$ = weight of the edge between i and j if one exists, otherwise ∞
- Pseudocode:

```
// initialise d[u][v] matrix with w[u][v] or \infty, d[v][v] = 0

for k from 1 to |V|:

    for i from 1 to |V|:

        for j from 1 to |V|:

            if   d[i][j] > d[I][k] + d[k][j]:

                d[i][j] := d[I][k] + d[k][j]
```

Time Complexity: $\Theta(|V|^3)$

On Implementation of Floyd-Warshall All

- The $\Theta(|V|^3)$ complexity works when the graphs are dense
- For sparse graphs the asymptotic complexity can be reduced
  - Hint: you Dijkstra with binary heaps for each vertex
    - Time complexity?
      - $O(|E|.|V|.log|V| + |V|^2 log|V|)$
      - The above is smaller than $O(|V|^3)$ when $|E| \ll |V|^2$
- Fasten matrix multiplication — Strassen's Algorithm
  - The optimal # of arithmetic ops to multiply two square n x n matrices is still an open problem!

Some applications of the discussed algorithms

- Find the shortest path between two nodes while avoiding any strongly connected components
  - Tarjan’s + Dijkstra’s Alg.
- Given a network of cities and roads, which each city allowing a certain number of goods to flow through it. Find the path for each city’s flow such that the maximum time taken for any flow is minimised.
  - Floyd-warshall + binary search on max flow time; then check feasibility of a path through Dijkstra’s
- **Optimal meeting point**:
  - n people need to meet in a restaurant in a city which minimises the distance for everyone.