R12 Bellman-Ford

3.30/3.31 https://github.com/GUMI-21/MIT6.006_note

Self summary of Bellman-Ford

1. If graph is a DAG, we can directed use DAG Relaxation in topological order to solve SSSP.
2. If graph does not have negative-weight cycles, shortest path from S to V must be simple (proof is in note)
3. simple path contain at most |V| - 1 edges
4. If $\delta_{|V|}(s, v) < \delta_{|V|-1}(s, v)$, then $\delta(s, v) = -\infty$, if A vertex V has this property, then V is a witness, can see a example which vertex at the end of a negative cycle.
5. Why need |V| - 1 iterate?
   If you are confused with Lec and the recitation note's code, I think this may be help you to understand Bellman-Ford.
   Because we input graph vertex array may not in a topological order, means the second index of array may not connect to source vertex array[0], then in worest case we need loop |V|-1 times to finally update all vertex estimate distance to be shortest weighted path. May we can associate this to Full-DFS, think just like Full-Relaxation. After all nodes be relaxed, then we can do one more relaxation to detect if there is a negative cycle. if any vertex still can be relaxed, then there must be a negative cycle ingraph.
   A important point is do |V|-1 loop is to ensure every vertex can be full-relaxed, not mean to detect negative cycle. The operations are two things, I think it is a good way to relax the confusion.

Rectation

The original Bellman-Ford algorithm is easier to state but is a little less powerful.
The algorithm is straight-forward:
-initialize distance estimates, and then relax every edge in the graph in |V |−1 rounds
-if the graph does not contain negative-weight cycles, d(s, v) = δ(s, v) for all v ∈ V at termination;
-otherwise if any edge still relaxable (i.e., still violates the triangle inequality), the graph contains a negative weight cycle.

```
def bellman_ford(Adj, w, s):
    # initialization
    infinity = float('inf')   # number greater than sum of all + weights
```

```
    d = [infinity for _ in Adj]  # shortest path estimates d(s,v)
    parent = [None for _ in Adj] # initialize parent pointers
    d[s], parent[s] = 0,s        # initizalize source
    # construct shortest paths in rounds
    V = len(Adj)                # number of vertices
    for k in range(V - 1):      # relax all edges in (V - 1) rounds
        for u in range(V):
            for v in Adj[u]:
                # try to relax edges. All unreachable from S throw
                if d[u] != infinity and d[v] > d[u] + w[u][v]:
                    d[v] = d[u] +w[u][v]
                    parent[v] = u

    # check for negative weight cycles accessible from s
    for u in range(V):
        for v in Adj[u]:
            if d[u] != infinty and d[v] > d[u] + w[u][v]:   # if edge relax-
able, report cycle
                raisse Excaption(`Ack! There is a negative weight cycle`)
    return d,parent
```

the algorithm relaxes every edge of the graph in a series of |V| - 1 rounds

Correctness

- Lemma 1
  At the end of relaxation round i of Bellman-Ford, d(s, v) = δ(s, v) for any vertex v that has a shortest path from s to v which traverses at most i edges
  Proof by indcution.
  base case: i = 0, d(s, s) = 0 = δ(s, s). correct
  induction step:
  Now suppose the claim is true at the end of round i − 1. Let v be a vertex containing a shortest path from s traversing at most i edges.
  d(s, v) != δ(s, v) prior to round i, and let u be the second to last vertex visited along some shortest path from s to v which traverses exactly i edges. Some shortest path from s to u traverses at most i − 1 edges, so d(s, u) = δ(s, u) prior to round i. Then after the edge from u to v is relaxed during round i, d(s, v) = δ(s, v) as desired. correct.
- *runtime*
  This algorithm runs |V | rounds, where each round performs a constant amount of work for each edge in the graph, so Bellman-Ford runs in O(|V||E|) time.
- supplement
  Note that if edges are processed in a topological sort order with respect to a shortest path tree from s, then Bellman-Ford will correctly compute shortest paths from s after its first round;

of course, it is not easy to find such an order.

However, for many graphs, significant savings can be obtained by stopping Bellman-Ford after any round for which no edge relaxation is modifying.