Homework 8 Problem 1

Yucheng Jin (yucheng9)
Wenyan Fan (wenyanf2)
Hanxiao Lu (hl4)

Solution

The following two subroutines in Jeff's *Algorithms* have not changed in (a).

```
InitSSSP(s):
    dist(s) ← 0
    pred(s) ← Null
    for all vertices v ≠ s
        dist(v) ← ∞
        pred(v) ← Null
```

An edge u → v is tense if dist(u) + w(u → v) < dist(v).

```
Relax(u → v):
    dist(v) ← dist(u) + w(u → v)
    pred(v) ← u
```

(a)

```
ModifiedBellmanFord(s):
    InitSSSP(s)
    repeat V - 1 times
        for every edge u → v
            if u → v is tense
                Relax(u → v)
    for every edge u → v
        # if any such cycle is reachable from s
        if u → v is tense
```

```
            cycle ← []

            # go backward from v along the predecessor chain
            cycle.push(v)
            repeat V - 1 times

                v ← pred(v)

                # until a cycle is found
                if v in cycle
                    # return the cycle
                    cycle.push(v)
                    return cycle
                else
                    cycle.push(v)
    # if there is no such cycle, return the tree*
    return pred
```

* "The predecessor pointers automatically define a tentative shortest-path tree rooted at s", refer to "8.3 The Only SSSP Algorithm" in "Notes on Shortest Paths" (08-sssp.pdf).

The Bellman–Ford algorithm should run in O(V E) time.
Regardless of any such cycle is reachable from s or there is no such cycle, the modified part checks each edge one by one, so the modified part should run in O(E) time.
Thus, the modified algorithm should run in O(V E) time.

(b)
this_dist is a temporary record of distance before checking negative cycles.

```
ModifiedBellmanFord(s):
    InitSSSP(s)
    repeat V - 1 times

        for every edge u → v

            if u → v is tense

                Relax(u → v)

    for every vertex v

        this_dist[v] ← dist[v]

    for every edge u → v

        if u → v is tense
```

```
            RELAX(u → v)

    for every vertex v
        # if any walk from s to v contains a negative cycle
        if this_dist[v] > dist[v]

            # our algorithm should end with dist(v) = ∞

            dist(v) ← ∞

    # otherwise, dist(v) should contain the length of the shortest path from s to v
```

The Bellman–Ford algorithm should run in O(V E) time.
Regardless of whether any walk from s to v contains a negative cycle, the modified part checks each vertex and each edge one by one, so the modified part should run in O(V) + O(E) time. Thus, the modified algorithm should run in O(V E) time.

Homework 8 Problem 2
Yucheng Jin (yucheng9)
Wenyan Fan (wenyanf2)
Hanxiao Lu (hl4)

Solution

For the given directed graph, we do not assign time cost to each edge until we traverse the directed tree with Dijkstra's algorithm. Each vertex represents the bus stop and each directed $v \to w$ means that there is a bus going from bus station v to the bus station w.

s is the starting vertex, t is the destination vertex and t_0 is the starting time. $l(u,v)$ means the time needed to travel from u to v, $f(u,v)$ mean the first time the bus leaves from u to v and $delta(u,v)$ means the time between such bus.

We modify the Dijkstra algorithm as follows:

```
Dijkstra($s$):
    INITSSSP($s$)
    Insert($s, t_0$)
    While the priority queue is not empty:
        $u \leftarrow$ ExtractMin( )
        ComputeNeighbourEdge($u$)
        // compute all the directed edge from u to its neighbour before
        checking for tense edge
        If $u \to v$ is tense:
            Relax($u \to v$)
            If $v$ is in the priority queue:
                DecreaseKey($v, Dist(v)$)
            Else:
                Insert($v, Dist(v)$)
```

Where the subroutines INITSSSP(s) and ComputeNeighbourEdge(u) are:

```
INITSSSP($s$):
    $Dist(s) \leftarrow t_0$    // assign the start vertex with the start time
    $Pred(s) \leftarrow$ null
    For every $v$ such that $v \neq s$:
        $Dist(v) \leftarrow \infty$
        $Pred(s) \leftarrow$ null
    For every directed edge $E$ in $G$:   // assign the edges to ∞ when initialize
        $E \leftarrow \infty$
```

```
ComputeNeighbourEdge($u$):
    For all edges $u \to v$:
        $k \leftarrow 0$    // k is the counter
        While ( $f(u,v) + k \times delta(u,v) - Dist(u) < 0$ ):
            $k \leftarrow k + 1$
        $u \to v \leftarrow f(u,v) + k \times delta(u,v) - Dist(u) + l(u,v)$
// use k to find the smallest time to wait for the bus from u to v and assign the
  edge with the time waiting plus the length of travel
```

Because the time complexity of ComputeNeighbourEdge(u) is $O(deg(u))$, which can be seen as $O(1)$, therefore, when we implement the Dijkstra algorithm, the overall time complexity is $O(E \cdot \log V)$.