

Problem Solving: Homework 3.1

Name: Chen Shaoyuan

Student ID: 161240004

September 4, 2017

1 [TC] Problem 24.1-2

We claim that, after k passes of relaxation, $v.d < \infty$ holds if $d(s, v) \leq k$. This can be proved by mathematical induction: for the base step, only the source s satisfies $s.d = 0 < \infty$, and the conclusion is true; for the induction step, assume that before the i -th pass of relaxation, for every vertex v such that $d(s, v) < i$, $v.d < \infty$. When performing relaxation, an estimate $v.d$ is changed from infinity to a finite number, if and only if there exists a directed edge (u, v) , such that $u.d < \infty$ and $v.d = \infty$. Therefore, for all vertices v such that $d(s, v) = i$, $v.d < \infty$. By mathematical induction, we prove the correctness of the claim.

Note that, if there exists a path from s to v , then $d(s, v) < |V|$ always holds, and Bellman-Ford algorithm performs $|V| - 1$ passes of relaxation in total. If for all vertices v , there exists a path from s to v , then the algorithms must terminate with $v.d < \infty$. Otherwise, if there exists a vertex v , such that v is unreachable from s , then it remains $v.d = \infty$ when algorithm terminates. This is because, $v.d$ is always an upper bound of $\delta(s, v)$ which is infinity, so every relaxation attempting to tighten $v.d$ will not succeed.

2 [TC] Problem 24.1-3

We use a flag to record whether a relaxation succeeds in a pass of relaxation. When no relaxation succeeds in a pass, we terminate the algorithm. The total number of passes executed is $m + 1$.

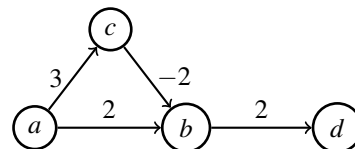
3 [TC] Problem 24.1-4

```
BELLMAN-FORD-MODIFIED( $G, w, s$ )
1  INITIALIZE-SINGLE-SOURCE( $G, s$ )
2  for  $i = 1$  to  $|G.V| - 1$ 
3      for each edge  $(u, v) \in G.E$ 
4          RELAX( $u, v, w$ )
5  Let  $Q$  be a new queue of vertices
6  for each vertex  $v \in G.V$ 
7       $v.flag = \text{FALSE}$ 
8  for each edge  $(u, v) \in G.E$ 
9      if  $v.d > u.d + w(u, v)$ 
10          $Q.enqueue(v)$ 
11          $v.flag = \text{TRUE}$ 
12  while  $Q$  is not empty
13      $u = Q.dequeue()$ 
14     for each  $v \in u.adj$ 
15          $v.d = -\infty$ 
16         if  $v.flag == \text{FALSE}$ 
17              $Q.enqueue(v)$ 
18          $v.flag = \text{TRUE}$ 
```

4 [TC] Problem 24.2-2

After changing, only the last vertex in topological order is not taken. However, the last vertex does not have succeeding vertex, therefore, the procedure remains correct.

5 [TC] Problem 24.3-2



Consider the graph shown above, if we take a as the source and run Dijkstra's algorithm, the procedure adds a, b, c, d to set S and relax the edges from them, respectively, and it finally terminates with $a.d = 0$, $b.d = 1$, $c.d = 3$, $d.d = 4$. Note that for vertex d , we have path $\langle a, c, b, d \rangle$ whose length is 3, which means Dijkstra's algorithm gives the wrong answer.

In the **maintenance** part of the proof, formula (24.2) holds because all edges are non-negative. If negative edges are allowed, the formula no longer holds and the $u.d = \delta(s, u)$ might not hold for vertex u added to set S .

6 [TC] Problem 24.3-4

DIJKSTRA-CHECKER(G, w, s)

```

1  if  $s.d \neq 0$  or  $\pi.s \neq \text{NIL}$ 
2    return FALSE
3  for each vertex  $v \in G.V$ 
4     $v.d' = \infty$ 
5   $s.d' = 0$ 
6  for each edge  $(u, v) \in G.E$ 
7     $v.d' = \min(v.d, u.d + w(u, v))$ 
8  for each vertex  $v \in G.V$ 
9    if  $v \neq s$ 
10     if  $v.d \neq v.d'$ 
11       return FALSE
12     if  $v.\pi \neq \text{NIL}$ 
13       if  $v.d \neq v.\pi.d + w(v.\pi, v)$ 
14         return FALSE
15     elseif  $v.d \neq \infty$ 
16       return FALSE
17  return TRUE

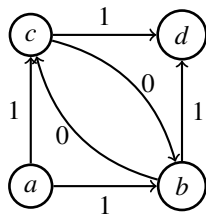
```

7 [TC] Problem 24.3-7

G' has $|V| + \sum_{e \in E} [w(e) - 1]$ vertices.

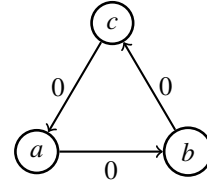
In breadth first-first search, the vertices v are colored black in the order of $d_{G'}(s, v)$. For vertices $v \in V$, $d_{G'}(s, v) = \delta_G(s, v)$. In Dijkstra's algorithm, a vertex v with smallest $v.d$ is extracted, and we've proved that $v.d = \delta_G(s, v)$. So, in both breadth-first search and Dijkstra's algorithm, vertices $v \in V$ are colored or extracted in the order of $\delta_G(s, v)$, and since $\delta_G(s, v)$ are distinct, the two orders are same.

8 [TC] Problem 24.5-2



Consider the graph above, take a as the source, then ytree s consisting of edges $(a, b), (b, d), (b, c)$ and $(a, c), (c, d), (c, b)$ are both shortest-paths trees rooted at a , and for every edge $e \in E$, exactly one of the two trees contains e .

9 [TC] Problem 24.5-5



Consider the graph above, if we take $a.\pi = c$, $b.\pi = a$, $c.\pi = b$, then they form a cycle.

10 [TC] Problem 24-2

- a. If box A with dimensions (x_1, x_2, \dots, x_d) nests in box B with dimensions (y_1, y_2, \dots, y_d) , box B nests in box C with dimensions (z_1, z_2, \dots, z_d) , then there exist permutations π_1, π_2 on $\{1, 2, \dots, d\}$, such that

$$x_{\pi_1(1)} < y_1, x_{\pi_1(2)} < y_2, \dots, x_{\pi_1(d)} < y_d$$

$$y_{\pi_2(1)} < z_1, y_{\pi_2(2)} < z_2, \dots, y_{\pi_2(d)} < z_d$$

then

$$x_{\pi_1(\pi_2(1))} < z_1, x_{\pi_1(\pi_2(2))} < z_2, \dots, x_{\pi_1(\pi_2(d))} < z_d$$

note that $\pi_1 \circ \pi_2$ is still a permutation, so box A nests in box C , i.e. the nesting relation is transitive.

- b. Given boxes A with dimensions (x_1, x_2, \dots, x_d) and B with dimensions (y_1, y_2, \dots, y_d) . Sort (x_1, x_2, \dots, x_d) and (y_1, y_2, \dots, y_d) in increasing order respectively, then check whether there exists $i (1 \leq i \leq d)$, such that $x_i \geq y_i$. If exists, then A does not nest in B ; otherwise, A nests in B .

Assume that array (y_1, y_2, \dots, y_n) is sorted in increasing order. For array (x_1, x_2, \dots, x_n) which is not sorted in increasing order, there must exist an inversion pair (x_i, x_j) ($x_i > x_j, i < j$). If $x_i < y_i$ and $x_j < y_j$, since $x_i > x_j, y_i < y_j$, we have $x_j < y_i$ and $x_i < y_j$, so after exchanging x_i with x_j , the condition still holds. Note that any finite permutation can be generated by swapping, greedy algorithm applies here and the method mentioned before is correct.

- c. The pseudocode is shown in the next page.

LONGEST-NESTED-BOXES(B, n)

```

1   $maxd = 0$ 
2  for  $i = 1$  to  $n$ 
3      sort dimensions of  $B_i$  in increasing order
4       $B_i.d = 0$ 
5       $B_i.\pi = \text{NIL}$ 
6  for  $i = 1$  to  $n$ 
7      for  $j = 1$  to  $n$ 
8          if  $B_i$  is nested in  $B_j$ 
9               $B_i.adj.insert(j)$ 
10 for every  $B_i$  in  $B$  in topological order
11     for every  $j$  in  $B_i.adj$ 
12         if  $B_j.d < B_i.d + 1$ 
13              $B_j.d = B_i.d + 1$ 
14              $B_j.\pi = B_i$ 
15             if  $B_j.d > maxd$ 
16                  $maxd = B_j.d$ 
17                  $ret = B_j$ 
18 return  $ret$ 

```

PRINT-LONGEST-NESTED-BOXES(X)

```

1  if  $X == \text{NIL}$ 
2      return
3  PRINT-LONGEST-NESTED-BOXES( $X.\pi$ )
4  print  $X$ 

```

FIND-SEQUENCE(G)

```

1  for every vertex  $v \in G.V$ 
2       $v.flag = 0$ 
3   $cnt = 0$ 
4  for every vertex  $v \in G.V$ 
5       $u = v$ 
6       $cnt = cnt + 1$ 
7      while  $u \neq \text{NIL}$  and  $u.cnt == 0$ 
8           $u.flag = cnt$ 
9           $u = u.\pi$ 
10         if  $u \neq \text{NIL}$  and  $u.flag == cnt$ 
11             print  $u$ 
12             PRINT-SEQUENCE( $u.\pi, u$ )
13         return TRUE
14 return FALSE

```

PRINT-SEQUENCE(X, u)

```

1  if  $X == u$ 
2      return
3  PRINT-SEQUENCE( $X.\pi, u$ )
4  print  $X$ 

```

This algorithm first builds a graph G , in which edge (u, v) means that u is nested in v . Then it calculates the longest nested boxes ending in v in topological order. Since we presorted the dimensions of all the boxes in $O(nd \log d)$, determining whether one box nests in another takes only $O(d)$ time, thus building the graph takes $O(n^2 d)$ in all. Calculating the longest nested boxes takes $O(|V| + |E|) = O(n^2)$ time. Therefore, the total running time is $O(nd \log d + n^2 d)$.

11 [TC] Problem 24-3

- a. First, build a graph $G = (V, E)$, where $V = \{c_1, c_2, \dots, c_n\}$, $E = \{(i, j) | i, j \in V\}$, and $w(i, j) = \ln R[i, j]$. Then we run Bellman-Ford algorithm on G to determine whether negative-weight cycle. Such sequence exists if and only if G contains negative-weight cycle. The running time is $O(|V||E|) = O(n^3)$.
- b. In the following procedure, assume that Bellman-Ford algorithm has been performed on G .