Sample Solution for Problem Set 7

Data Structures and Algorithms, Fall 2020

December 29, 2020

Contents

1 Problem 1

(a) We call MAKE-SET n times, which contributes $\Theta(n)$. In each union, the smaller set is of size 1, so each of these takes $\Theta(1)$ time. Since we union $n-1$ times, the runtime is $\Theta(n)$.

(b) Since the loop and the assignment runs in $O(n)$ time, we only need to prove that the operation sequence $Find(node[1]), \ldots, Find(node[n])$ runs in $O(n)$ time in worst case.

Let T be the original forest. Let T' be the forest that results from the operation sequence $Find(v)$ which v is a node. In T' all nodes on the path from v to the root of the tree are now children of the root. Suppose that $Find(w)$ is done later. If the path in T from w to the root meets the path in T from v to the root, say at x, then $Find(w)$ follows only one link for the overlapping path segment since in T' x is a child of the root. Thus the total amount of work done by the $Find$s is bounded by a multiple of the number of distinct branches in T. Since there are n nodes, the total amount of work done by the $Find$s is in $O(n)$.

2 Problem 2

WLOG, we may assume that we will only paint white cells into black.

Given array $X = [x_1, x_2, \ldots, x_n]$, consider $S_1, S_2, \ldots, S_k$ as a partition of $[n]$, satisfying that for each $i \in [k]$, S_i forms a continous segment, and $x_j = 0$ if and only if there exists an $i \in [k]$, such that $j = \max S_i$. We may directly see from the definition that there's a one to one corresponding between X and partition of $[n]$ satisfying above constraints. The crucial observation is the following lemma.

Lemma 1. *For a given binary array X, there exists unique (up to permutation) partition $S_1, S_2, \ldots, S_k$ satisfying above constraints, and*

- $Blacken(i)$ *operator for array X is exactly $Union(i, i+1)$ operator for corresponding partition $S_1, S_2, \ldots, S_k$.*
- $NextWhite(i)$ *query for array X is exactly finding the maximum value of S_j where $i \in S_j$.*

Therefore, we may solve this task via DSU.

(a)

Create two auxiliary array Y, Z, representing size of its component, and leader element(in this case, $\max S_j$). We may merge the smaller set into the larger one in merge process, and output Y_i in query process. Note that for every position i, its information will only be updated $O(\log n)$ times.

(b)

We may use the classical DSU data structure with path compression and union by rank technique. However, we should notice that the leader element in this scenario may not be the largest index in this components. This can be fixed by tracking the largest index in each component.

(c, bonus)

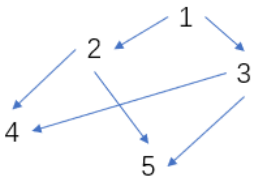
Note that union operation will be operated only on ends of two segments, and concatenate them. Therefore, we may store index of leader and range of segment for both ends of segments. It can be seen that information can be updated efficiently(in $O(1)$) during merge process.

3 Problem 3

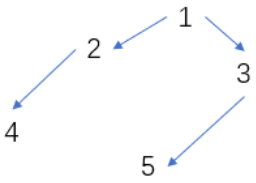
Start by examining position $(1, 1)$ in the adjacency matrix. When examining position (i, j), if a 1 is encountered, examine position $(i + 1, j)$. If a 0 is encountered, examine position $(i, j + 1)$. Once either i or j is equal to $|V|$, terminate.

We claim that if the graph contains a universal sink, then it must be at vertex i. To see this, suppose that vertex k is a universal sink. Since k is a universal sink, row k in the adjacency matrix is all 0's, and column k is all 1's except for position (k, k) which is a 0. Thus, once row k is hit, the algorithm will continue to increment j until $j = |V|$. To be sure that row k is eventually hit, note that once column k is reached, the algorithm will continue to increment i until it reaches k. This algorithm runs in $O(V)$ and checking whether or not i in fact corresponds to a sink is done in $O(V)$. Therefore the entire process takes $O(V)$.

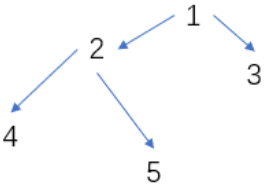
4 Problem 4



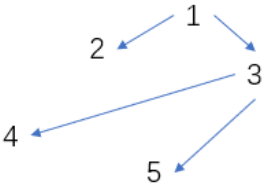
G=<V,E>



G_π=<V,E_π>



Breadth-First-Tree 1



Breadth-First-Tree 2

(a)

- $u.d$ represents the shortest distance from the root node to u, which must be unique.
- As $G =< V, E >$ in the figure above, different breadth-first trees (1 and 2) will be generated based on the visited order of nodes 2 and 3.

(b)

- As $G_\pi =< V, E_\pi >$ in the figure above, No matter which node you visit first in node 2 and 3, you can't generate a G_π.

vertex	d	f
q	1	16
r	17	20
s	2	7
t	8	15
u	18	19
v	3	6
w	4	5
x	9	12
y	13	14
z	10	11

5 Problem 5

(a)

- tree edge: (q,s), (s,v), (v,w), (q,t), (t,x), (x,z), (t,y), (r,u)
- back edge: (w,s), (z,x), (y,q)
- forward edge: (q,w)
- cross edge: (r,y), (u,y)

(b)

Counterexample:

- $G =< V, E >, V = \{x, y, z\}, E = \{(x, y), (y, x), (x, z)\}$
- $x.d < y.d < z.d$
- There is a path from y to z, but z is not a descendant of y.

6 Problem 6

```
DFS-STACK(G)
    for each vertex u ∈ G.V
        u.color = WHITE
        u.π = NIL
    time = 0
    S = Ø
    for each vertex u ∈ G.V
        if u.color == WHITE
            time = time + 1
            u.d = time
            u.color = GRAY
            PUSH(S, v)
            while !STACK-EMPTY(S)
                v = TOP(S)
                isNeighborhoodsAllDiscovered = true
                if v.color == GRAY
                    for each vertex w ∈ G.Adj[v]
                        if w.color == WHITE
                            time = time + 1
                            w.d = time
                            w.color = GRAY
                            PUSH(S, w)
                            isNeighborhoodsAllDiscovered = false
                            break
                if isNeighborhoodsAllDiscovered
                    time = time + 1
                    v.f = time
                    v.color = BLACK
                    POP(S)
```

7 Problem 7

Suppose $G = (V, E)$ and $V = \{v_1 = s, ..., v_n = t\}$.

(a) Create a new graph $G' = (V', E')$, where $V = \{v_{i,j}\}_{1 \le i \le n, 0 \le j \le 2}$, and $E' = \{(v_{i,k}, v_{j,(k+1) \mod 3} \mid (v_i, v_j) \in E\}$. The algorithm is: do BFS or DFS on G' to find whether there is a path from $v_{1,0}$ to $v_{n,0}$. If there is a path, the output yes, otherwise output no. The complexity is $O(|V| + |E|)$ since the number of vertice and edges are multiplied by a constant.

To prove the correctness, suppose there is a path $\{v_{a_1,b_1}, v_{a_2,b_2}, ...\}$ in G' from $v_{1,0}$ to $v_{n,0}$, then the path $\{v_{a_1}, v_{a_2}, ...\}$ is the path with the same length (divided by 3 since and edge will increase b_i by 1, i.e., $b_{i+1} = b_i + 1 \mod 3$) G' from v_1 to v_n. Conversely, if there is a path in G: $\{v_{a_1}, v_{a_2}, ...\}$ such that the length is divided by 3, then the path $\{v_{a_1,0}, v_{a_2,1}, v_{a_3,2}, v_{a_4,0}...\}$ is the path from $v_{1,0}$ to $v_{n,0}$ with the same length.

(b) Similar to **(a)**, we copy the graph for 5 times, creating graph $G' = (V', E')$ where $V = \{v_{i,j}\}_{1 \le i \le n, 0 \le j \le 4}$. E' is composed by 8 part:

- $(v_{i,0}, v_{j,1})$ where (v_i, v_j) is a blue edge.
- $(v_{i,1}, v_{j,2})$ where (v_i, v_j) is a blue edge.
- $(v_{i,0}, v_{j,3})$ where (v_i, v_j) is a red edge.
- $(v_{i,3}, v_{j,4})$ where (v_i, v_j) is a red edge.
- $(v_{i,1}, v_{j,3})$ where (v_i, v_j) is a red edge.
- $(v_{i,2}, v_{j,3})$ where (v_i, v_j) is a red edge.
- $(v_{i,3}, v_{j,1})$ where (v_i, v_j) is a blue edge.
- $(v_{i,4}, v_{j,1})$ where (v_i, v_j) is a blue edge.

Intuitively, the graph is composed by 5 layers, the layer 0 are the initial layer; layer 1 and 2 are the blue layer such that the node has already pass 1 or 2 consecutive blue edges; the layer $3, 4$ are the red layer where the node has already pass 1 or 2 consecutive red edges. Each layer can be seen as a "state" of a node, and the edges in E' is all the possible change between states.

Now run BFS to find the shortest path from $v_{1,0}$ to $v_{n,i}$ where i is arbitrary. The length of the shortest path in G' is the lenght of the required path in G. The proof is similar to **(a)**: paths between two graph can map from each other.