

# Solution for Problem Set 7

Mianzhi Pan, 181240045

November 3, 2020

## 1 Problem 1

(a) We do MAKESET for  $n$  times and each of them cost  $\Theta(1)$ . Notice during each UNION, we append the list of length 1 to the longer list, so it also cost  $\Theta(1)$ . Hence the total running time is  $n + n - 1 = \Theta(n)$ .

(b)

## 2 Problem 2

(a)

## 3 Problem 3

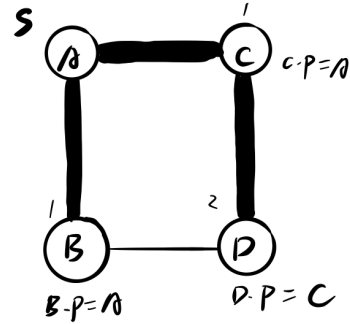
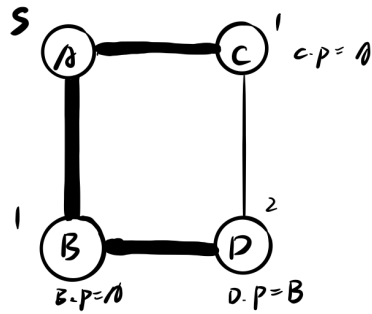
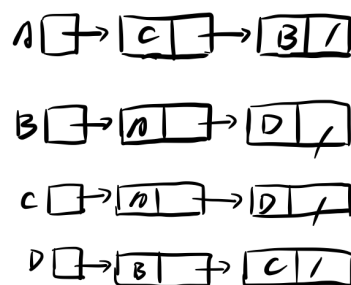
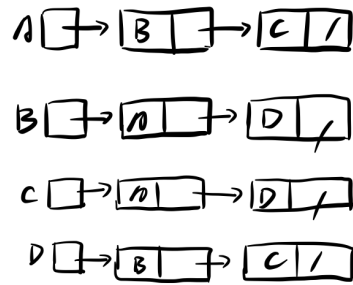
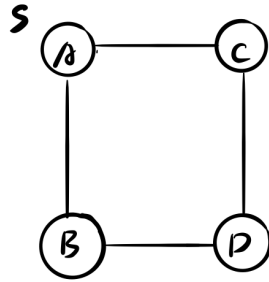
Suppose the adjacency matrix is  $A$  and the  $i^{th}$  vertex satisfying the property, then the matrix row  $A[i][1, \dots, |V|]$  must be all 0 and the column  $A[1, \dots, |V|][i]$  must be all 1 except the element  $A[i][i]$ .

Suppose we start at the left upper corner  $A[1][1]$  and then we start moving. If the current element is 0 we move right. Otherwise we will go down. At last we may finish at  $A[k][|V|]$  or  $A[|V|][k]$ , then check if the  $k^{th}$  element satisfies the property. If it does, we find the vertex we want. Otherwise the vertex we want does not exist.

## 4 Problem 4

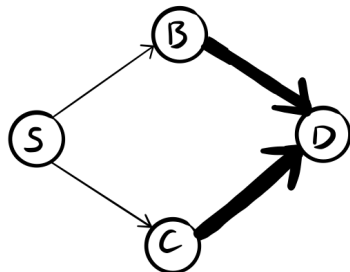
(a) Suppose the source vertex is  $s$ , then for every vertex  $u$  reachable from  $s$ , the distance  $u.d$  assigned to it must be the distance of the shortest path (Theorem 22.5, CLRS). Therefore, once the source vertex is fixed, the distance for every vertex is fixed as well, which is independent of the order in the adjacency list.

See from the following example



Suppose the source vertex is A, for adjacency lists of different ordering, the breadth-first tree is different since  $D.p = B$  in the left tree while  $D.p = C$  in the right one.

(b) The graph is shown below

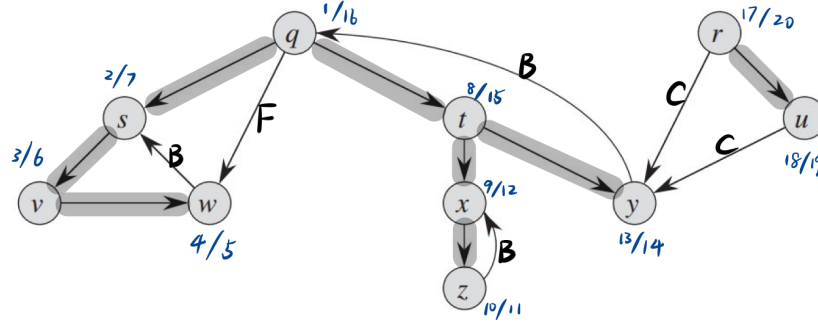


No matter B precedes C or C precedes B in the adjacency list  $Adj[s]$ , one

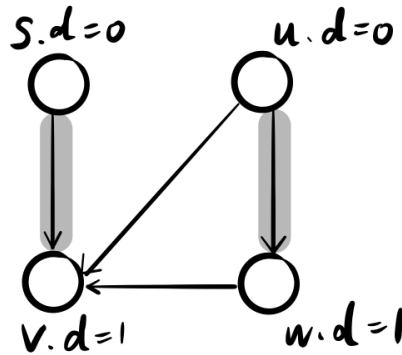
of the two bold edges will not be produced.

## 5 Problem 5

(a) The discovery and finishing time is marked near each vertex and the tree edges are bold, back edges are  $B$ , forward edges are  $F$ , cross edges are  $C$ .



(b) A counterexample is given in the following



Assume the DFS procedures the vertices in alphabetical order, we can find that  $u.d = 0 < v.d = 1$ , and graph  $G$  contains a path from  $u$  to  $v$ . However,  $v$  is not a descendant of  $u$  since the edge  $(u, v)$  is a cross edge.

## 6 Problem 6

---

### Algorithm 1 DFS( $G, s$ )

---

```

1:  $time = time + 1$ 
2:  $s.d = time$ 
3:  $s.color = GRAY$ 
4: for (each edge  $(s, v)$  in  $E$ ) do
5:   if  $v.color == BLACK$  then
6:     if  $v.d > s.d$  then
7:       print  $(s, v)$  is forward edge
8:     else
9:       print  $(s, v)$  is cross edge
10:    end if
11:  else if  $v.color == GRAY$  then
12:    print  $(s, v)$  is back edge
13:  else
14:    DFS( $G, v$ )
15:  end if
16: end for
17:  $s.color = BLACK$ 
18:  $time = time + 1$ 
19:  $s.f = time$ 

```

---

## 7 Problem 7

(a) Inspired by the concept of implicit graph, we can build a new graph to solve the problem.

Suppose the origin directed graph is  $G = (V, E)$ , we make three copies  $V_0, V_1, V_2$  of the vertices. For each edge  $(u, v) \in E$  in the origin graph, we create new edges  $(u_i, v_j)$  in the new graph where  $u_i$  is the corresponding vertex of  $u$  in  $V_i$ ,  $v_j$  is the corresponding vertex of  $v$  in  $V_j$  and  $j = (i + 1) \bmod 3$  ( $i = 0, 1, 2$ ). i.e. each edge  $(u, v)$  has three corresponding edges  $(u_0, v_1), (u_1, v_2), (u_2, v_0)$  in the new graph.

Then we comes to the problem. Given two vertices  $s^0$  and  $s^k$  in the origin graph  $G$  and a path connected them of length  $k$ , assume all vertices along the path is  $s^0, s^1, \dots, s^{k-1}, s^k$ , and the corresponding vertices in the new graph is  $s_m^0, s_{(m+1) \bmod 3}^1, \dots, s_n^k$ . Obviously,  $n = (m + k) \bmod 3$ . If  $k$  is divisible by 3, then  $n = m$ . Samely, we can prove the inverse proposition is true as well.

Deduce from the above proposition, if there exists a path from  $s_m$  to  $t_m$ , (e.g.  $s_0$  and  $t_0$ ) in the new graph, which can be implement by BFS, then there must exists a walk from  $s$  to  $t$  whose length is divisible by 3 in the origin graph, and the algorithm will return **true**.