HW04 for ECE 9343

Tongda XU, N18100977

November 13, 2018

1 Question 1: CLRS Exercise 22.1-3

```
TRANSPOSE(Adjlist)
new AdjlistPrime
for each node in Adjlist
    for each subnode in Adjlist(node)
        AdjlistPrime(subnode).insert(node)
        Adjlist = AdjlistPrime
```

For adjacent list: just traverse every node and rebuild one
$\Theta(E + V)$ for time and space complexity, hard to do it inplace

```
TRANSPOSE(Adjmatrix)
for each pair(i, j) in upper left Adjmatrix
    SWAP(Adjmatrix[i, j], Adjmatrix[j, i])
```

For adjacent matrix: just transpose the matrix
$\Theta(V^2)$ for time and $\Theta(1)$ for space

2 Question 2: CLRS Exercise 22.1-5

For adjacent list, it is hard. We should regard it as a BREADTH-FIRST-SEARCH(G) end at $d = 2$:

```
SQUARE(G)
for each u in G.vertices
    G.reset()
    list = ∅
    u.adjlist' = BFS-AID(G, u, list, 0)
```

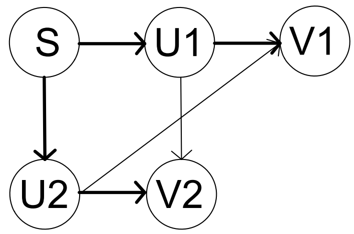

Figure 1: 22.2-6

BFS-AID$(G, u, list, dist)$
for each v in $u.adjlist$
  if $v.color = white$ and $dist \leq 2$
    $list.insert(u)$
    BFS-AID$(G, v, list, dist + 1) =$
return $list$

This could cost $\Theta(V^2 + VE)$ time and $\Theta(V + E)$ space (if optimized).

For adjacent matrix, the square process would be simple. for each index m of matrix row, if matrix[m][n] exist, calculate bool union of matrix[m] and matrix[n]:

SQUARE(G)
for each m in $G.adjMatrix$
  for each n $G.adjMatrix[m]$
    if $G.adjMatrix[m][n] == 1$
      $G^{'}.adjMatrix[m] =$ AND$(G.adjMatrix[m], G.adjMatrix[n])$
return $G^{'}$

The SQUARE(G) cost $\Theta(V^3)$ time and $\Theta(V)$ space (if optimize)

3 Question 3: CLRS Exercise 22.2-6

Consider the following condition in Figure 1:
$E_\pi =< s, u1 >, < u1, v1 >, < s, u2 >, < u2, v2 >$
In BFS Tree, $\delta(s, v1), \delta(s, v2)$ is either $< s, u1, v1 >, < s, u1, v2 >$ or $< s, u2, v1 >$ $, < s, u2, v2 >$

4 Question 4: Traverse Edge of undirected graph

According to $Theorem 22.10$, all edges are either tree edge or back edge. Modify the DFS-Visit(G, u), add a print-path(G, u) would do it. Assume a $root = u$ is selected:

```
DFS-Visit(G, u)
1  u.color = grey
2  dict[(Vertex, Vertex), edgeType] = ∅
3  for each v in u.adjList
4      if v.color == white
5          dict(u, v) = treeEdge
6          DFS-Visit(G, v)
7      else dict(u, v) = backEgde
8  print-path(G, u)
```

```
print-path(G, u)
1  print ("u")
2  for each v in u.adjList
3      if (u, v) == treeedge
4          print (" → ")
5          print-path(G, v)
6      else print (" → v")
```

line 4,6 cost same level of time as the comparison in line 3, would not change the $\Theta(V + E)$ time complexity of DFS(G)
the print path function as:
This procedure cost $\Theta(V + E)$ as well

5 Question 5: CLRS Exercise 22.3-12

Tweak the DFS-Visit(G, u) and DFS(G) would be enough:

```
DFS(G)
for each u in G.V
    u.color = white
c = 1
for each u in G.V
    if u.color = white
        DFS-Visit(G, u, c)
        c++
```

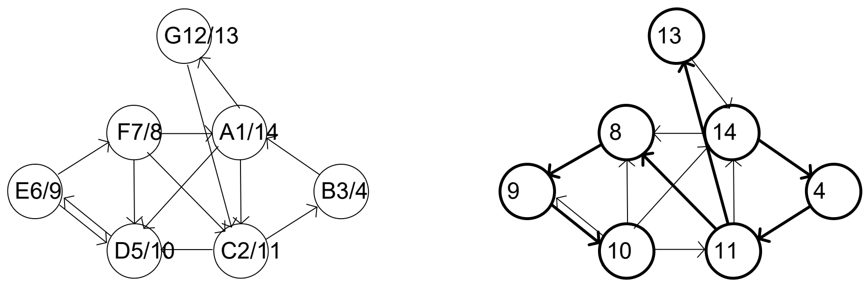

Figure 2: 22.2-6

DFS-VISIT(G, u, c)

$u.color = grey$
$u.cc = c$
for each v in $u.adjList$
    if $v.color == white$
        DFS-VISIT(G, v)

DFS(G) could be tweaked to do it as well

6 Question 6: CLRS Exercise 22.4-1

$p[27:28] \rightarrow n[21:26] \rightarrow o[22:25] \rightarrow s[23:24] \rightarrow$
$m[1:20] \rightarrow r[6:19] \rightarrow y[9:18] \rightarrow v[10:17] \rightarrow x[15:16] \rightarrow$
$w[11:14] \rightarrow z[12:13] \rightarrow u[7:8] \rightarrow q[2:5] \rightarrow t[3:4]$

7 Question 7: Show process of SCC

See Figure 2

8 Question 8: CLRS Problem Set 22.1

8.1 a-1

Suppose (v, u) is a backedge. u is ancestor elder than parent of v. This means (s, u) + forwardEdge is shorter than (s, v) produced by BFS which is $\delta(s, v)$ by **Theorem 22.5**. Same reason for forward edge.

8.2 a-2

By **Theorem 22.5** $\delta(s, v) = \delta(s, v.parent) + (v.parent, v) = \delta(s, u) + (u, v) \rightarrow v.d = u.d + 1$

8.3 a-3

$v.d \leq u.d + 1$: Same as a-1, if $v.d > u.d + 1$, $\delta(s, v) = (s, u) + cross$ instead of (s, v).
$v.d \geq u.d$: If $v.d < u.d$, (v, u) should be find out first, since this is undirected graph.

8.4 b-1

Same as a-1, the $(s, u) + backEdge$ would be shorter than (s, v)

8.5 b-2

Same as a-2, By **Theorem 22.5** $\delta(s, v) = \delta(s, v.parent) + (v.parent, v) = \delta(s, u) + (u, v) \rightarrow v.d = u.d + 1$

8.6 b-3

Only the first half of a-3. if $v.d > u.d + 1$, $\delta(s, v) = (s, u) + cross$ instead of (s, v).

8.7 b-4

By **Corollary 22.4** and By **Theorem 22.5**, we know that if v is an ancestor of u $\delta(s, u) = \delta(s, v) + k \rightarrow \delta(s, u) > \delta(s, v) \rightarrow u.d > v.d$, I did not see how $u.d = v.d$ but the statement is correct.

9 Question 8: CLRS Problem Set 22.3

9.1 1. proof

Euler tour exist $\rightarrow$ in-degree == out-degree: Suppose the cycle through i vertex n times would be $E - cycle = \{v_i, v_j, v_k, ..., v_i\}$. The in-degree of v_j would be the time of v_j appears with element in front, and out-degree of v_j would be the time of v_j appears with element in the back. If v_j is not head or tail, this is obvious that every time v_j appear, there is element in front and tail. If v_j is head, it must also be tail, which balance the in-degree and out-degree again.

in-degree == out-degree $\rightarrow$ Euler tour exist

9.2 2. implement

This is very similar to SCC, we find closed cycle first then join them with other edge set. This procedure would return a cycle, which is a list of vertex. closed cycle has $cycle.begin() = cycle.end()$, open cycle(path, not a cycle) do not has it. But if Euler tour exist, open cycle would join close cycle into a big cycle.

```
CircleFind(cycle, u, v)
ClosedCycleSet, OpenCycleSet = ∅
while alladjList! = ∅
    for v in Vertex with adjList! = ∅
        if v.adjList! = ∅
            new cycle = ∅
            CircleFindAid(cycle, u, NIL)
            if cycle.type == closed
                ClosedCycleSet.push(cycle)
            else OpenCycleSet.push(cycle)
```

```
CircleFindAid(cycle, u, v)
cycle.insert(u)
if v! = NIL
    v.adjList.erase(u)
if u == NIL
    cycle.type = open
    return
elseif u == cycle.start
    cycle.type = close
    return
else v = u
    u = u.adjList.begin()
    CircleFind(cycle, u, v)
```

It is easy to find that as we remove an edge from adjacent list once we find it, and we traverse every edge, the time complexity would be $\Theta(E)$