600.363/463 Homework 5 Solution
Posted: 10/16/03
Due: Wednesday, 10/22/03

## 22.1-5

Let $A$ be the adjacency matrix of the graph $G$ and $B$ the adjacency matrix of the graph $G^2$. $B[i,j] = 1$ iff $\exists k s.t. A[i,k] = 1 and A[k,j] = 1$ by definition. To compute $B$, multiply $A$ by itself using mattrix multiplication with the change that addition is the Boolean OR operation and multiplication is the Boolean AND operation. The running time is $O(|V|^3)$ as computing each entry of $B$ involves $|V|$ multiplications and $|V|$ additions. The pseudocode is given below:

1. $n \leftarrow |V|$
2. for $i = 1$ to $n$
3. for $j = 1$ to $n$
4. $B[i,j] = 0$
5. for $i = 1$ to $n$
6. for $j = 1$ to $n$
7. for $k = 1$ to $n$
8. $B[i,j] = \bigwedge(A[i,k] \bigvee A[k,j])$

The above algorithm computes $B$ and hence $G^2$. The proof of correctness would argue that $B[i,j] = 1$ iff there is a $k$ in the $i^t h$ row of $A$ such that $A[i,k] = 1$ and $A[k,j] = 1$. This implies that $j$ is at a distance of 2 from $i$ via the path $(i,k) \rightarrow (k,j)$.

## 22.1-7

$BB^T(i,j) = \sum_{e \in E} b_{ie} b^T_{je} = \sum_{e \in E} b_{ie} b_{ej}$
So, if $i = j$, then $b_{ie} b_{ej} = 1$ ( it is $1 \cdot 1$ or $(-1) \cdot (-1)$) whenever $e$ enters or leaves vertex $i$, and 0 otherwise.
If $i \neq j$, then $b_{ie} b_{ej} = -1$ when $e = (i,j)$ or $e = (j,i)$, and 0 otherwise.
Thus:

$$BB^T(i,j) = \begin{cases} \text{degree of } i\text{=in-degree + out-degree} & \text{if } i = j \\ -(\#\text{ deges connecting } i \text{ and } j) & \text{if } i \neq j \end{cases}$$

**22.2-4**

The correctness proof for the breadth-first-search algorithm shows that $d[u] = \delta(s, u)$, and the algorithm doesn't assume that the adjacency lists are in any particular order.
Consider the example of 22.3, if we visit $x$ before visiting $t$, then edge$(x, u)$ will belong to the BFS tree instead of edge $(t, u)$.

**22.3-11**

The comments in the following pseudocode show the changes to DFS to assign the desired $cc$ label to vertices.

```
DFS(G)
    for each vertex u ∈ V[G]
        do color[u] ← WHITE
            π[u] ← NIL
    time ← 0
    counter ← 0          % New conter
    for each vertex u ∈ V[G]
        do if color[u] = WHITE
            then counter ← counter + 1        %Increment counter
                DFS-VISIT(u, counter)        %Pass counter argument

DFS-VISIT(u, counter)          %Counter is argument.
    color[u] ← GRAY
    cc[u] ← counter          %Label the vertex.
    d[u] ← time ← time + 1
    for each v ∈ Adj[u]
        do if color[v] = WHITE
            then π[v] ← u
                DFS-VISIT(v, counter).        %Pass unchanged counter.
    color[u] ← BLACK
    f[u] ← time ← time + 1
```

This DFS increments a counter each time DFS-CISIT is called to grow a new tree in the DFS forest. Every vertex visited (and added to the tree) by DFS-VISIT is labeled with that same counter value. Thus $cc[u] = cc[v]$ if and only id $u$ and $v$ are visited in the same call to DFS-VISIT from DFS, and the final value of the counter is the number of calls that were made to DFS-VISIT by DFS. Also, since

every vertex is visited eventually, every vertex is labeled.
Thus all we need to show is that the vertices visited by each call to DFS-VISIT from DFS are exactly the certices in one connected component of $G$.

- All vertices in a connected component are visited by one call to DFS-VISIT from DFS:
  Let $u$ be the first vertex in component $C$ visited by DFS-VISIT. Since a vertex becomes non-WHITE only when it is visited, all vertices in $C$ are white when DFS-VISIT is called for $u$. THus, by the White-path theorem, all vertices in $C$ become descendants of $u$ in the forest, which means that all vertices in $C$ are visited (by recursive calls to DFS-VISIT) before DFS-VISIT returns to DFS.

- ALl Vertices visited by one call to DFS-VISIT from DFS are in the same connected component:
  if two vertices are visited in the same call to DFS-VISIT from DFS, they in the same connected component, because vertices are visited only by following paths in $G$ (by following edges found in adjacency lists, starting from some vertex).