# Problem Set 2

Brian Fox

1. *22.1-3* The ***transpose*** of a directed graph $G = (V, E)$ is the graph $G^T = (V, E^T)$ , where $E^T = \{(v, u) \in V \times V : (u, v) \in E\}$. Thus,$G^T$ is $G$ with all its edges reversed. Describe efficient algorithms for computing $G^T$ from $G$, for both the adjacency-list and adjacency-matrix representations of $G$. Analyze the running times of your algorithms.

   Adjacency-list($G$)

   ```
   1  for each list L in G
   2          add first element of L to G^T
   3  i = 0
   4  for each list L in G
   5          i = i+1
   6          for each element e in L
   7                  if not first element of e
   8                          append i to G^T[e]
   ```

   Adjacency-matrix($G$)

   ```
   1  r = 1
   2  while r ≤ rows in G
   3          c = 1
   4          while c ≤ columns in G
   5                  G^T(c, r) = G(r, c)
   6                  c = c+1
   7          r = r+1
   ```

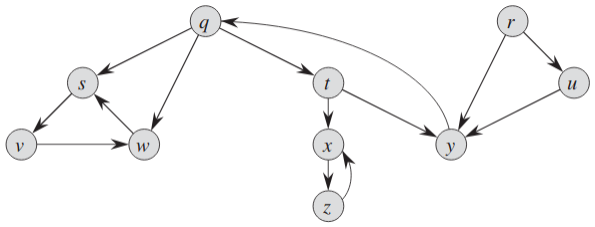
   Adjacency-list($G$) will run $V + E$ iterations which makes it $O(E)$ assuming the graph is connected. Adjacency-matrix($G$) is $O(V^2)$

2. *22.2-4* What is the running time of BFS if we represent its input graph by an adjacency matrix and modify the algorithm to handle this form of input?

   To queue the connected vertices of the vertex we pop, we will have to iterate over a list of length $V$. This will make the run time $O(V^2)$

3. *22.3-2* Show how depth-first search works on the graph of Figure 22.6. Assume that the for loop of lines 5–7 of the DFS procedure considers the vertices in alphabetical order, and assume that each adjacency list is ordered alphabetically. Show the discovery and finishing times for each vertex, and show the classification of each edge.

   fig 22.6:




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

4. *22.3-7* Rewrite the procedure DFS, using a stack to eliminate recursion. DFS($G$)

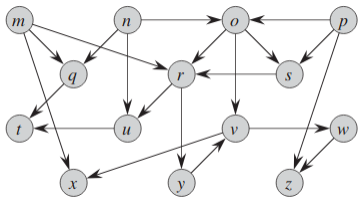
```
1   for each vertex u ∈ G.V
2       u.color = WHITE
3       u.π = NIL
4   time = 0
5   for each vertex u ∈ G.V
6       if u.color == WHITE
7           DFS-VISIT(G, u)
```

DFS-Visit($G, u$)

```
1   time = time + 1
2   u.d = time
3   u.color = GRAY
4   S = Stack(G)
5   while S is not empty
6       for each v ∈ G.Adj[u]
7           if v.color == WHITE
8               v.π = u
9               S.add(v)
10          u.color = BLACK
11          time = time + 1
12          u.f = time
```

5. *22.4-1* Show the ordering of vertices produced by TOPOLOGICAL-SORT when it is run on the dag of Figure 22.8, under the assumption of Exercise 22.3-2.

fig 22.8:




$$p \to n \to o \to s \to m \to r \to y \to v \to x \to w \to z \to u \to q \to t$$

6. *23-4* In this problem, we give pseudocode for three different algorithms. Each one takes a connected graph and a weight function as input and returns a set of edges $T$. For each algorithm, either prove that $T$ is a minimum spanning tree or prove that $T$ is not a minimum spanning tree. Also describe the most efficient implementation of each algorithm, whether or not it computes a minimum spanning tree.

   a) Maybe-MST-A $(G, w)$

```
1  sort the edges into nonincreasing order of edge weights w
2  T = E
3  for each edge e, taken in nonincreasing order by weight
4        if T − {e} is a connected graph
5              T = T − {e}
6  return T
```

   T is a MST. By using an adjacency list you need to sort $E$ items while an adjacency matrix would need to sort $V^2$ items. the rest of the algorithm is independent of the data structure used since it uses a list of edges to construct the MST.

   b) Maybe-MST-B$(G, w)$

```
1  T = ∅
2  for each edge e, taken in arbitrary order
3        if T ∪ {e} has no cycles
4              T = T ∪ {e}
5  return T
```

   T is a tree but is not a MST. Checking for cycles will be best done with a DFS. the worst case run time with be the same as a BFS but the average will be better if the cycles contain many vertices.

   c) Maybe-MST-C$(G, w)$

```
1  T = ∅
2  for each edge e, taken in arbitrary order
3        T = T ∪ {e}
4        if T has a cycle c
5              let e′ be a maximum-weight edge on c
6              T = T − {e′}
7  return T
```

   T is not necessarily a MST or a tree.