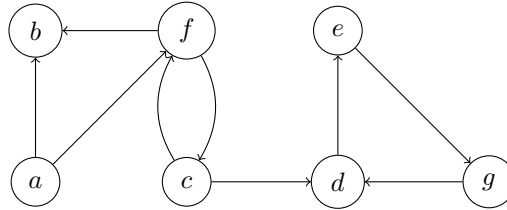# DSC 40B - Discussion 06

Problem 1.

The python code for quickselect is given below.

a) Consider a *breadth*-first search on the graph shown in the figure, starting with node c. Which nodes are visited, and in what order? Use the convention that graph.neighbors() produces successors in ascending order of label.

> **Solution:** c,d,f,e,b,g

b) Consider a *breadth*-first search on the graph shown in the figure, starting with node a. Which nodes are visited, and in what order? Use the convention that graph.neighbors() produces successors in ascending order of label.

> **Solution:** a,b,f,c,d,e,g

c) Consider a *breadth*-first search on the graph shown in the figure, starting with node g. Which nodes are visited, and in what order? Use the convention that graph.neighbors() produces successors in ascending order of label.

> **Solution:** g,d,e

Problem 2.

Consider line 6 in Algorithm 1 shown below. Whenever this line is executed, is it possible...

    a) ...that node is 'pending'?

> **Solution:** No

    b) ...that node is 'visited'?

> **Solution:** Yes

    c) ...that some other node is 'pending'?

> **Solution:** No

    d) ...that some other node is 'visited'?

> **Solution:** Yes

Figure 1: Algorithm 1

```
1.   from collections import deque
2.
3.   def full_bfs(graph):
4.       status = {node: 'undiscovered' for node in graph.nodes}
5.       for node in graph.nodes:
6.           if status[node] == 'undiscovered'
7.               bfs(graph, node, status)
8.
9.   def bfs(graph, source, status=None):
10.      """Start a BFS at `source`."""
11.      if status is None:
12.          status = {node: 'undiscovered' for node in graph.nodes}
13.
14.      status[source] = 'pending'
15.      pending = deque([source])
16.
17.      # while there are still pending nodes
18.      while pending:
19.          u = pending.popleft()
20.          for v in graph.neighbors(u):
21.              # explore edge (u,v)
22.              if status[v] == 'undiscovered':
23.                  status[v] = 'pending'
24.                  # append to right
25.                  pending.append(v)
26.          status[u] = 'visited'
```

Problem 3.

It turns out that BFS does not need to use three distinct statuses (undiscovered, pending, visited) in order to function – we just use three to help us understand the algorithm. In fact, we can delete code from bfs, and the function will still work. List below the line number(s) which can all be deleted without affecting the behavior of the algorithm.
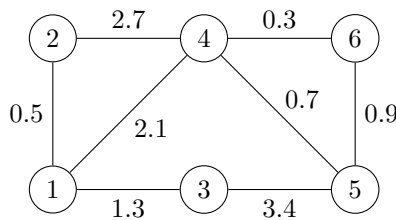
> **Solution:** Line 26

Problem 4.

An *edge weighted graph* $G = (V, E, \omega)$ is a graph along with a function $\omega : E \to \mathbb{R}$ which assigns a weight to every edge in the graph. One of the uses of edges weights is to encode dissimilarities. That is, the greater the weight of an edge, the more dissimilar the nodes at either end.

A natural task involving weighted graphs is to *cluster* the nodes of the graph into groups such that nodes in the same group are similar to one another while two nodes in different groups are dissimilar.

Here is a simple way of clustering a weighted, undirected graph $G = (V, E, \omega)$. Given a real number t, place two nodes u and v in the same cluster if (and only if) there is a path between u and v along which every edge has weight $\leq t$. For instance, consider the graph below:

If $t = 1$, there are three clusters: $\{1, 2\}$, $\{4, 5, 6\}$ and $\{3\}$. If $t = 2$, there are two clusters: $\{1, 2, 3\}$ and $\{4, 5, 6\}$. And if $t = 3$, there is one cluster containing all of the nodes.

Design an algorithm which returns the clusters given an input graph, a weight function, weights, and a threshold t. Your algorithm should take $\Theta(V + E)$ time. To receive full credit, your algorithm should not modify the graph or create a copy of it. Provide pseudocode (or Python).

> **Solution:**
>
> ```python
> def cluster(graph, weights, t):
>     status = {'undiscovered' for node in graph.nodes}
>     clusters = []
>     for node in graph.nodes:
>         if status[node] == 'undiscovered':
>             cluster = bfs_cluster(graph, node, weights, status, t)
>             clusters.append(cluster)
>
> def bfs_connected_components_with_threshold(
>         graph, source, weights, status, t):
>     cluster = [source]
>     pending = deque([source])
>     status[source] = 'pending'
>     while pending:
>         u = pending.popleft()
>         for v in graph.neighbors(u):
>             if status[v] == 'undiscovered' and weight(u, v) <= t:
> ```

3

```python
            pending.append(v)
            status[v] = 'pending'
            cluster.append(v)
    status[u] = 'visited'
    return cluster
```