DSC 40B - Hashing

Problem 1.

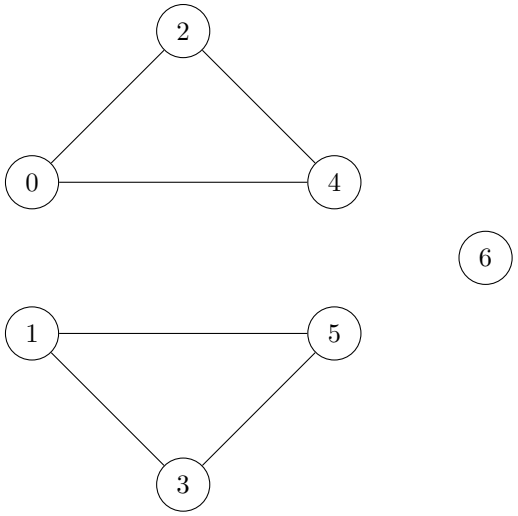
In the following, let

$$V = \{0, 1, 2, 3, 4, 5, 6\},$$
$$E = \{(0, 2), (1, 5), (3, 1), (2, 4), (0, 4), (5, 3)\}.$$

For this problem, you do not need to show your work.

a) Draw the *undirected* graph $G = (V, E)$. Remember that when writing the edges of an undirected graph, we often abuse notation and write (u, v) when we really mean $\{u, v\}$; we have done so here.

Solution:



b) Draw the graph $G = (V, E)$, assuming that G is directed.

c) Write down the connected components of G, assuming that G is undirected.

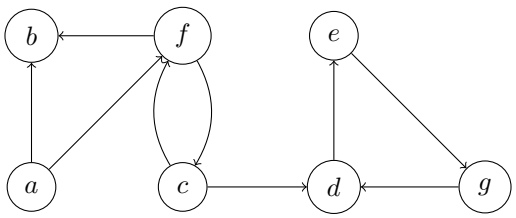
Solution: From the above drawing we see that the connected components are:

$$\{0, 2, 4\}, \{1, 3, 5\}, \{6\}$$

d) Write the adjacency matrix representation of G, assuming that G is undirected.

e) Write the adjacency matrix representation of G, assuming that G is directed.

Problem 2.



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 3.

Consider the following modified BFS which has two new lines of code: lines 21 and 22. What is the asymptotic time complexity of `foo` in terms of $|V|$ and $|E|$?

```
1  from collections import deque
2
3  def foo(graph):
4      status = {'undiscovered' for node in graph.nodes}
5      for u in graph.nodes:
6          if status[u] == 'undiscovered':
7              bar(graph, u, status)
8
9  def bar(graph, source, status):
10     status[source] = 'pending'
11     pending = deque([source])
12
13     # while there are still pending nodes
14     while pending:
15         u = pending.popleft()
16         for v in graph.neighbors(u):
17             # explore edge (u,v)
18             if status[v] == 'undiscovered':
19                 status[v] = 'pending'
20                 pending.append(v)
21             for v in graph.nodes:
22                 print(v)
23         status[u] = 'visited'
24
25     return predecessor, distance
```

Solution: The new lines of code take $\Theta(V)$ time every time that an edge is visited. This means that the total time complexity has an extra $\Theta(VE)$. So the time is $\Theta(V + E + VE)$. This simplifies to $\Theta(V + VE)$, since $|E| \leq |V| \cdot |E|$.

Note that it isn't quite right to say that $|V| \cdot |E|$ is larger than $|V|$ and thus the time complexity is $\Theta(VE)$. For instance, suppose $|E| = 0$. Then $|V| \cdot |E|$ is zero, and therefore not larger than $|V|$!.