CS 61B Discussion 10 Spring 2017

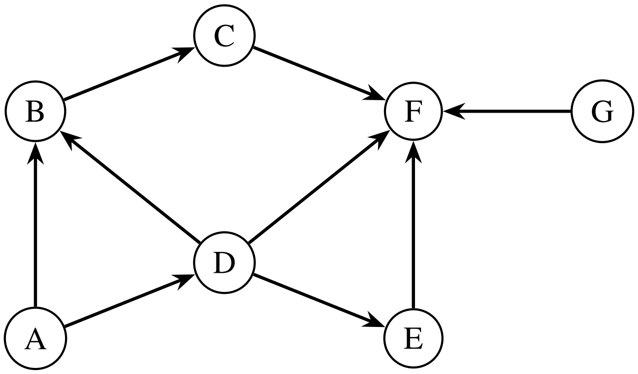
1 Heaps of fun

(a) Assume that we have a binary min-heap (smallest value on top) data structure called `Heap` that stores integers, and has properly implemented `insert` and `removeMin` methods. Draw the heap and its corresponding array representation after each of the operations below:

```
Heap h = new Heap(5); // Creates a min-heap with 5 as the root
h.insert(7);
h.insert(3);
h.insert(1);
h.insert(2);
h.removeMin();
h.removeMin();
```

(b) Your friend Sahil Finn-Garng challenges you to quickly implement an integer max-heap data structure - "Hah! I'll just use my min-heap implementation as a template to write max-heap.java", you think to yourself. Unfortunately, two Destroyer Penguins manage to delete your `MinHeap.java` file. You notice that you still have `MinHeap.class`. Can you still complete the challenge before time runs out? **Hint:** you can still use methods from MinHeap.

2 Graph Representations

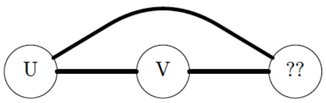


Write the graph above as an adjacency matrix, then as an adjacency list.

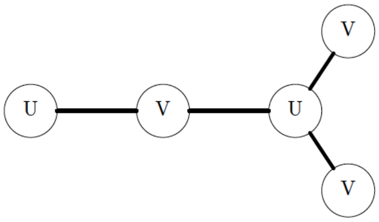
Give the DFS preorder, DFS postorder, and BFS order of the graph starting from vertex *A*. Break ties alphabetically.

3 Graph Algorithm Design: Bipartite Graphs

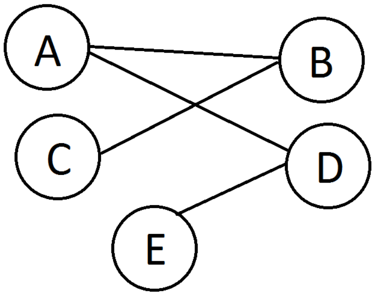
An undirected graph is said to be bipartite if all of its vertices can be divided into two disjoint sets U and V such that every edge connects an item in U to an item in V. For example, the graphs in the center and on the right are bipartite, whereas the graph on the left is not. Provide an algorithm which determines whether or not a graph is bipartite. What is the runtime of your algorithm?



Not bipartite



Bipartite



Bipartite

4 Extra for Experts: Shortest Directed Cycles

Provide an algorithm that finds the shortest directed cycle in a graph in $O(EV)$ time and $O(E)$ space, assuming $E > V$.

5 Extra for Experts: DFS Gone Wrong

Consider the following implementation of DFS, which contains a crucial error:

```
create the fringe, which is an empty Stack
  push the start vertex onto the fringe and mark it
  while the fringe is not empty:
    pop a vertex off the fringe and visit it
    for each neighbor of the vertex:
      if neighbor not marked:
        push neighbor onto the fringe
        mark neighbor
```

Give an example of a graph where this algorithm may not traverse in DFS order