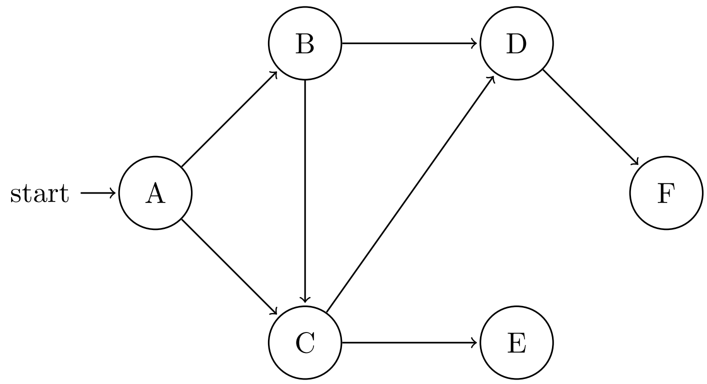
1. Suppose we are talking about the depth-first search (DFS) algorithm. Nodes are added to the data structures in alphabetical order.

   (a) What underlying data structure does this algorithm use?

   (b) Given the following graph, state the DFS traversal order and show the data structure at each step. Node `A` is the start node, and `F` is the destination node.



   (c) What path from `A` to `F` does the DFS algorithm return?

2. Now consider a BFS algorithm, again populating data structures in alphabetical order.

   (a) What changes would need to be made to a DFS implementation to turn it into a breadth-first search (BFS)?

   (b) Using the graph as described in Question 1, what is the BFS traversal order? Show the data structure at each step.

   (c) What path from `A` to `F` results from the BFS algorithm?

3. **Searching a Graph**

   (a) Write a recursive algorithm that (given a graph, start vertex, and goal vertex), determines whether or not there is a path to the goal vertex.

   Assume you are provided with a `Graph` class with a `getNeighbors( int vertex )` method, which returns a `Set<Integer>` representing the numbers corresponding to neighboring vertices. Assume `visited` is a `Set` keeping track of all visited vertices.
   (Note: Your algorithm should return a Boolean value, not an actual path!)

```
boolean hasPathToRec(Graph g, int start, int goal, Set<Integer> visited) {

}
```

   (b) Rewrite your algorithm to be iterative instead.
   (Hint: What data structure do you need to use if you no longer have recursion?)

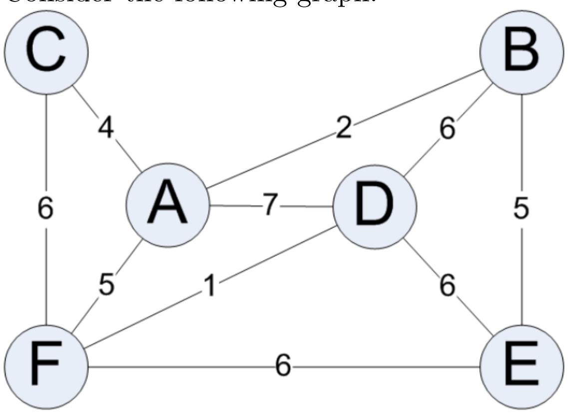
```
boolean hasPathToIter(Graph g, int start, int goal, Set<Integer> visited) {

}
```

4. When is a vertex's sum weight finalized in Dijkstra's algorithm?

5. In Dijkstra's algorithm, what role does the priority queue play in finding the shortest path? When do we use it?

6. Why does Dijkstra's algorithm not work correctly on graphs with negative edge weights?

7. Consider the following graph.



(a) Perform Dijkstra's algorithm to find the shortest path between C and E.

Finalized	A	B	C	D	E	F
–	(∞, None)	(∞, None)	**(0, None)**	(∞, None)	(∞, None)	(∞, None)

(b) In general, when using Dijkstra's algorithm to find the shortest path between nodes, do you need to use every row of the table? Why or why not?

8. Explain the Model-View-Controller design pattern.

9. Describe each of the following layout managers:

   (a) `FlowPane`

   (b) `BorderPane`

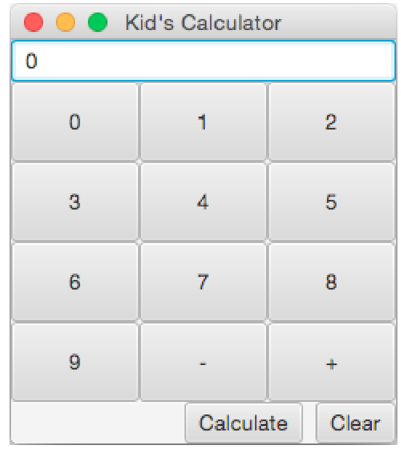
   (c) `GridPane`

   (d) `HBox and VBox`

10. Write a function or routine which creates a `Button` with the text "Click me!" that prints out a message of your choosing when it is clicked, using `System.out.println()`. When you've reached a solution, think about how you might be able to reorganize the code which tells the button what do when it is pressed in a different way.

11. Create a class that constructs and displays the following GUI. If you can't remember exactly how to implement a certain part of the GUI in code, explain what the component is and how it would fit in with the rest of the calculator. (*Hint: draw out the GUI's **component hierarchy**.*)

The buttons within the GUI do not need to be functional. You may or may not need the following: `Scene`, `BorderPane`, `FlowPane`, `HBox`, `TextField`, `Button`. The window should fit to all of the components and have a title.



12. Briefly explain the differences between the three kinds of exceptions: checked exceptions, runtime exceptions, and errors.

13. Is there anything wrong with the following exception handler as written? Will this code work as intended?

```
try {
    this.epicFail();
} catch (Exception e) {
    ...
} catch (ArithmeticException a) {
    ...
}
```

14. What is the output when `LookAtDatMagic`'s main is executed?

```java
public class HeySteve{
    public int bananza(int in) throws NewException{
        if ( in == 7 ){
            throw new NewException("HeySteve, cut that out!");
        }
        return in;
    }
}

public class NewException extends Exception{
    public NewException(String message){
        super(message);
    }

}

public class WakaWaka{
    public String BeachBash(Object a, Object b) throws NewException{
        if ( a.equals(b) ){
            throw new NewException("It's a Beach-bash! WakaWaka!");
        }
        return "Da-nanananan";
    }
}

public class LookAtDatMagic{
    public void magic() throws NewException{
        int maraca = 5;
        try{
            HeySteve steve = new HeySteve();
            maraca = steve.bananza(7);
        }catch(NewException e){
            System.out.println(e.getMessage());
        }finally{
            WakaWaka waka = new WakaWaka();
            System.out.println(waka.BeachBash(maraca, 5));
        }
    }

    public static void main(String[] args){
        try{
            LookAtDatMagic ladm = new LookAtDatMagic();
            ladm.magic();
        }catch(NewException e){
            System.out.println(e.getMessage());
        }
    }
}
```

15. Consider a simple backtracking algorithm.

   (a) What are the four core components of any backtracking `solve` function?

   (b) Write a generic `solve()` function for a given configuration, which returns either the solution configuration or `null` (if no solution exists):
   (*Hint: make up a function name for each of the parts above, if necessary*)

```
public Configuration solve(Configuration config) {

}
```