Backtracking with bound pruning

Start with the code we developed in class for backtracking with bound pruning. Recall that we implemented this in an iterative way. This allowed us to easily switch the data structure being used from a Stack to a Queue to a Priority Queue. By changing this structure, we get different searches through the state-space.

Weighted Graph Coloring

The particular problem you will be coding is weighted graph coloring.

You will have to implement the State interface in order to plug weighted graph coloring into your backtracking framework. Call this class GraphColoringState. You will need a constructor that takes in basic information about the problem to be solved. In particular, we will need a representation of the graph and a list of possible colors. I am going to constrain you a bit by specifying the representation of these two items. The following is an example Main you could use for the non-weighted version:

```java
public static void main(String[] args) {
    Backtracker bt = new Backtracker();

    // Define the graph from the ppt
    boolean[][] graph =
        {{false, true, false, false, false, true},
         {true, false, true, false, false, true},
         {false, true, false, true, true, false},
         {false, false, true, false, true, false},
         {false, false, true, true, false, true},
         {true, true, false, false, true, false}};

    // Define the colors used in the ppt
    List<String> colors = new ArrayList<String>();
    colors.add("Red");
    colors.add("Green");
    colors.add("Blue");

    State s = new GraphColoringState(graph, colors);

    State result = bt.backtrack(s);
    System.out.println(result);
}
```
The Graph array is an adjacency matrix representation. The trues indicate where there is a connection between the nodes. This particular graph uses a 6x6 matrix, meaning that there are 6 nodes in the graph. This is the same graph as in the ppt slides. You can view node “A” as 0, node “B” as 1, etc. The colors List is the colors to use and the order to use them in.

To upgrade this problem to a weighted version, you will need to also include weights in your GraphColoringState representation. You may pick how this is done.

“Timing”

You are to “time” your code on a fully connected graph with 3 nodes along with colors/weights of red/2, green/3, blue/5, and yellow/2. Note that the example above is the example from the un-weighted version in the ppt slides. You will need to adjust this to be a fully connected graph of 3 nodes. The timing will not be actual runtime but instead a count of the number of nodes expanded. You should time it for depth, breath, and best-first searches. You should make sure you stop expanding nodes as early as possible given the type of search you are performing.

Submission

You are to turn in:

- A printed picture of your results for the sample runs using all three types of searches.
- A printed single page (max) document describing any problems that your code might still have if it is not complete. Or if it is complete, possible places for improvement in your code.
- A jar file of your entire package submitted to the 335 folder on the W drive.

You may work with a partner.