Final exam schedule

<table>
<thead>
<tr>
<th>Section</th>
<th>Date and time, Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>901</td>
<td>6/11, 5:45-8:45, Daley 512</td>
</tr>
<tr>
<td>902</td>
<td>6/10, 5:45-8:45, CDM 819</td>
</tr>
<tr>
<td>910,911</td>
<td>One of the above, or a 3-hour period between 6/6 and 6/11 with a proctor</td>
</tr>
</tbody>
</table>

Other exam information

The exam is open book and open notes. It will have a written portion and a coding portion, similar in format to the midterm exam. You may use your laptop or a PC in the lab or provided by your proctor. You may use Eclipse and Java to help you write code for problems which require coding. Online students who do not have access to a laptop or a lab PC may take the exam on paper.

Topics

The exam will emphasize, but not be limited to, topics that we have covered since the midterm. These include:

- **Hashing.** HashMaps and HashSets, and their implementations and basic operations. Collision strategies.
- **Priority queues.** Basic operations; implementation using arrays/binary heaps; applications.
- **Graphs.**
  - Representations as adjacency matrices and adjacency lists.
  - Depth-first and breadth-first graph search.
  - Cheapest-path search: Dijkstra's algorithm and A*.
- **String matching,** including regular expressions and finite-state automata. I may ask you to write a regular expression or a simple FSA, but you will not be responsible for the algorithms to convert between regular expressions to FSAs, or from NFAs to DFAs.

There may also be a question about which data structure is best for a task, such as question 1 below. And of course, we have discussed complexity throughout the quarter.
• Practice problems

1. Explain which of the data structures from CSC 402-403 would be most useful in each of the following applications. You may choose from the following data structures: stack, queue, list, balanced tree, hash table, priority queue, or graph.
   a. You are building a system which processes orders that have been placed in an online store. The orders should be processed in the order that they were made.
   b. You are writing a text editor, and wish to implement an "undo" feature, which behaves that way that the Control-Z keystroke does in many editors.
   c. You are building a system that maintains a wait-list of passengers on a flight who are waiting for an upgrade to first class. The passenger with the most frequent flier miles is the first to be upgraded when a first class seat becomes available.
   d. You are building a system that will be used to maintain a list of people who are employed by a company. The system should maintain the list in alphabetical order by the employees' last names, and should support operations to add and remove employees from the list as they are hired by or leave the the company.
   e. You are building a system which maintains a collection of products sold by a store. Products are looked up by serial number, where serial numbers are represented as strings.
   f. You are building a tool for a social networking company on which people can "friend" each other. The tool suggests new friends to users. A suggested friend is determined as follows: if W is X's friend, X is Y's friend, and Y is Z's friend, then the tool suggests X, Y, and Z to W.

2. Show the results of inserting 37, 32, 17, 19, 34, and 20 into a hash table of size 13, and a hash function H(x) = x%13, using the following techniques:
   - Linear probing
   - Quadratic probing
   - Double hashing, assuming the second hash function is 11-(x%11)

3. Assume that the contents of a priority queue are stored in an array, as discussed in class. What are the contents of the array after the following items are added to the queue: 7, 8, 3, 4, 2, 1, 6, 5? Assume that lower numbers have first priority.

4. Represent the graph below as (a) an adjacency matrix, and (b) using an adjacency list.

![Graph Image]
5. Show how Dijkstra's algorithm finds the shortest path from 0 to 6 in the above graph. Assume that the algorithm is implemented to use a table which, for each vertex \( v \), stores information about (a) the cost to reach a vertex (from 0) of the cheapest path found so far; (b) the predecessor to \( v \) in this path; and (c) whether or not this path is known to be the cheapest path to \( v \). The table is initialized as follows:

\[
\begin{array}{ccc}
0 & 0 & f \\
1 & \infty & f \\
2 & \infty & f \\
3 & \infty & f \\
4 & \infty & f \\
5 & \infty & f \\
6 & \infty & f \\
7 & \infty & f \\
\end{array}
\]

Initially, the costs to all vertices except 0 are infinity, and each vertex except 0 has no known predecessor (0 definitely has no predecessor, since it is the start of the path). \( k = f \) means that it is not yet known if the cheapest path to a vertex has been found.

Show the state of the table as the algorithm progresses.

6. Consider this graph:

\[
\begin{array}{c}
0 \quad 2 \\
\downarrow & 1 \\
1 \quad 2 \\
\uparrow & 3 \\
2 \quad 1 \\
\downarrow & 2 \\
3 \quad 2 \\
\end{array}
\]

Assume that A* is used to find the cheapest path from 0 to 4. The estimate for the cheapest path from \( v \) to \( w \) is the "Manhattan distance" between them. For example, the estimate of the cost from 1 to 4 is 2.

Again, show the state of the table as the search progresses. Recall that the A* table has an additional column \( e \) for the estimate of future cost.
7. Write Java regular expressions to match strings according to each of the descriptions below

   Strings containing a number (any sequence of digits), followed by a hyphen and then any sequence of (at least one) alphabetic lower case letters.
   a. Strings containing a number, followed by a hyphen and then any sequence of alphanumeric characters (digits, lower or upper case letters) and/or hyphens. Strings may not end with a hyphen.
   b. Any integer (positive, zero, or negative) or floating point number.
   c. Any number between 420 and 599.

8. What strings are accepted by the Finite State Automaton below, assuming that state 3 is the accepting state (there is a loop on state 1 labeled 'b')?

   Is the above automaton deterministic or nondeterministic?

9. Draw an FSA which accepts strings over \{a, b, c\} which contain more than 2 a's.

10. Consider the `Fraction` class below. Write methods for this class so that `Fraction` objects could be used in HashMaps and TreeSets. Indicate if one or more interfaces is required and their name(s).

    ```java
    public class Fraction { // implements ... ?
        private int numerator;
        private int denominator;
    }
    ```
11. The DiGraph class below is intended to represent an unweighted, directed graph. Notice that the class has a method called hasCycle. This method should return true if a graph has a cycle or false otherwise. Note that it calls path, which is intended to return true if there is a path from v to w, or false otherwise. path is also passed a set of vertices which have already been visited in the search for a path from v to w.

Complete the path method.

```java
public class DiGraph {
    // the number of vertices in the graph
    private final int V;
    // the number of edges
    private int E;
    // adj[v] is a list of vertices w
    // such that an edge (v,w) is in the graph
    private final List[] adj;

    // methods to create a graph are not shown

    public boolean hasCycle() {
        // for every vertex v in the graph, try to find
        // a path from v to itself. If there is such
        // a path, return true
        for (int v=0; v<V; v++)
            for (int w : adj[v]) {
                Set visited = new HashSet();
                if (path(w, v, visited))
                    return true;
            }
        // if there is no path (v,...,v) for any vertex v
        // in the graph, then there are no cycles.
        return false;
    }

    // should return true is there is a path from v to w
    // probably easiest to write recursively
    public boolean path(int v, int w, Set visited) {
        // replace this
        return false;
    }
}
```

12. The DFA class below is intended to represent a deterministic finite state automaton. It differs somewhat from the implementation that I presented during lecture, primarily in that the edges which originate from a state (i.e., a vertex) are represented as a list rather than a map.

Write an accept method for this implementation. The method should determine whether or not a string is accepted by the automaton. You may assume that vertex 0 is the start state.
public class DFA {
    class State {
        public List adj; // the edges that originate from a state
        public boolean accept; // is the state an accept state?
    }
    class Edge {
        public char label; // the label on an edge
        public State to; // the state which the edge leads to
        public Edge(char c, int w) {
            label = c;
            to = states[w];
        }
    }
    // the states in the FSA.  Assume states[0] is start state
    private State[] states;
    // create a DFA with V states
    public DFA(int V) {
        states = new State[V];
        for (int i=0; i<V; i++) {
        }
    }
    public void markAsAccepting(int a) {
        states[a].accept = true;
    }
    public void addEdge(int v, int w, char c) {
        State state = states[v];
        states[v].adj.add(new Edge(c, w));
    }
    // write this method
    public boolean accept(String string) {
        // replace this
        return false;
    }
    // creates a simple DFA that should accept
    // strings over {a,b} that end with 'a'
    public static void main(String[] args) {
        DFA dfa = new DFA(2);
        dfa.addEdge(0, 1, 'a');
        dfa.addEdge(0, 0, 'b');
        dfa.addEdge(1, 1, 'a');
        dfa.addEdge(1, 0, 'b');
        dfa.markAsAccepting(1);
        System.out.println(dfa.accept("aba")); // true
        System.out.println(dfa.accept("abab")); // false
    }
}