Algorithms

Exercise 7: Graph algorithms II

Answer SECTION 1 and TWO other sections.

1 vEB Trees

1. Draw a proto-vEB node, labelling each of its fields and briefly explaining what each field does. (2 marks)
2. Draw a vEB node, labelling each of its fields and briefly explaining what each field does. (2 marks)
3. Draw a complete and legible vEB tree holding the values $\{0, 2, 4, 8, 9, 10, 13, 14\}$. The correctness of the structure and the accuracy of all fields of all nodes is important. Once done, write each of the values under the leaf record in which it is logically stored. (6 marks)
4. Consider the task of inserting a value v into a proto-vEB or vEB tree whose root node is r. Assume the value is in range and not already in the tree. Write two pieces of high-level pseudocode for $insertInProtoVEB(r, v)$ and $insertInVEB(r, v)$ respectively. Clarity (insert comments where appropriate) will count more than perfect low-level accuracy. Derive the computational complexity of your two procedures using the appropriate recurrence formulae (but solving the recurrences is not required). Explain what specific features of the vEB tree make it faster than the proto-vEB tree for this particular task. (10 marks)

2 Maximum flow problem

5. The pseudocode below is a first attempt at a recursive algorithm to enumerate all the paths from the source to sink, in the context of a maximum flow problem.

```
def allPaths(graph, source, sink) :
  // Each path is a list of vertices from source to sink, e.g. [2,4,7]
  // The result is a list of paths, e.g. [[2,4,7],[2,7]], initially empty
  result = []

  if source == sink :
    result.append([source])
  else :
    for v in graph.verticesAdjacentTo(source) :
      for path in allPaths(graph, v, sink) :
        // Reject paths that revisit the source, else infinite loops
        if source not in path :
          result.append([source] + path)
  return result
```

(a) Point out all the bugs you can find, highlighting the failures with test cases. (5 marks)

(b) Correct all the bugs you found, clearly explaining your fixes. Rewrite a corrected and clearly commented version of the pseudocode. (10 marks)

(c) Provide a correctness proof for your new version. (5 marks)

3 Dijkstra's shortest-path algorithm

6. Describe Dijkstra's shortest-path algorithms, making the priority queue operations explicit. (4 marks)

7. Provide a small example demonstrating that Dijkstra's shortest-path algorithm will not work correctly when negative weights are used on some arcs. (3 marks)

8. Suppose some arcs in a directed graph have negative weights, and that $-W$ is the least negative weight among all arcs. Suppose that we add W to all arcs in the graph to obtain a new graph with non-negative arc weights. Will the resulting graph have the same shortest paths as the original graph? Explain your answer. (3 marks)

9. For each of the data structures listed below, describe the computational complexity of Dijkstra's shortest-path algorithm when this data structure is used to implement the algorithm's priority queue. Justify your answers.

   (a) An unsorted array, indexed by node number. (2 marks)

   (b) A linked list, sorted by key (in this case a distance estimate). (2 marks)

   (c) A binary heap. (2 marks)

   (d) A binomial heap. (2 marks)

   (e) A Fibonacci heap. (2 marks)

4 Noughts and Crosses

10. Write a program that allows a human player to play noughts and crosses against the computer, using game trees and a suitable algorithm to ensure that the computer never loses.

    The default size of a noughts and crosses board is 3, but it should be possible to change this to values greater than 3. The human player and the computer take turns, but it should be possible to change who goes first.

    Solutions which are particularly elegant and are *algorithmically* faster than my model solution will be awarded a prize. (20 marks)