Student Number (Required) ______________________

Name (Optional) ________________________________

This is a closed book test. You may not refer to any resources.

This is a 50 minute test.

Please write your answers in ink. Pencil answers will be marked but will not be re-marked under any circumstances.

The test will be marked out of 50.

<table>
<thead>
<tr>
<th>Question 1</th>
<th>/15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Question 2</td>
<td>/15</td>
</tr>
<tr>
<td>Question 3</td>
<td>/15</td>
</tr>
<tr>
<td>Question 4</td>
<td>/5</td>
</tr>
<tr>
<td>TOTAL</td>
<td>/50</td>
</tr>
</tbody>
</table>

“When you want to fool the world, tell the truth.”

Happy Birthday to Otto von Bismarck
We have studied Breadth First Search and Depth First Search. In this question we will look at a different search method called Best-First Search. Best-First Search is applicable to graphs in which each vertex has an integer value attached to it (see the example below). The search starts at some specified vertex $x$ and searches the graph by always choosing the next reachable vertex with the highest value. We will assume the graph is connected. In pseudo-code the algorithm looks like this:

```
let $P(v)$ be a function that returns the value of vertex $v$

Best_First_Search($x$):
  $A = \{x\}$
  Chosen = {}
  while |Chosen| < $n$: # $n$ is the number of vertices
    let $v$ be the vertex in $A$ with the largest $P()$ value
    Add vertex $v$ to Chosen
    Remove vertex $v$ from $A$
    for each neighbour $y$ of $v$:
      if $y$ is not in Chosen:
        Add $y$ to $A$
```

If we start at vertex $A$, the next vertex added is $E$, then $C$, then $D$, then finally $B$

If we start at vertex $D$, the next vertex added is $C$, then $B$, then $E$, then finally $A$
a) What data structure would you choose to represent the graph? Why?

Solution: Adjacency lists are most appropriate for this algorithm, since it optimizes the performance of the line “for each neighbour y of v”

Marking: part a) is worth 6 marks:

structure:
- adj list: 4
- adj matrix: 2
- heap, stack, queue etc: 1

why:
- reasonable answer: 2
- weak answer: 1
b) What data structure would you choose to help with the process of selecting the next vertex to add at each iteration of the while loop? Why?

Solution: A max-heap would be best because it lets us add new vertices to the set \( A \) and keep track of the best choice efficiently.

Marking: part b) is worth 6 marks:

structure: heap: 4
1-dimensional array: 2
queue, stack, binary tree etc: 1

why: reasonable answer: 2
weak answer: 1

---

c) How would you keep track of which vertices are in the set Chosen?

Solution: Use an array of length \( n \) – call it Chosen. Set Chosen\([v]\) to True (or 1) when vertex \( v \) is chosen.

Marking: part c) is worth 3 marks:

structure: array: 3
hash table: 2
heap, stack, queue etc: 1
Question 2 (15 marks)

Let G be an undirected graph with vertex set V and edge set E.

The complement of G is defined to get the undirected graph with vertex set V and edge set E', where E' contains exactly the edges that are not in E.

a) Suppose G is represented by an adjacency matrix. How would you construct the complement of G? (this question does not require a full page to answer)

Solution: Make a copy of the adjacency matrix. Change all the 0's to 1's, and vice versa.

In practice we would not put 1's on the diagonal of the new adjacency matrix because we do not normally allow loops in our graphs, but this detail is not important in this answer.

Marking: part a) is worth 6 marks:

The key concept is changing all the 1's to 0's and vice versa – they don't need to make a copy of the matrix first.
b) Suppose G is represented by a set of adjacency lists. How would you construct the complement of G?

Solution: Create a new set of adjacency lists for the complement, initially empty. Then for each vertex x, for every vertex y other than x, if y is not in x’s adjacency list for the original graph then add y to x’s adjacency list for the complement graph.

Marking: part b) is worth 6 marks:

There are a number of other possible solutions, such as

- create adjacency lists containing all possible edges, then remove the edges that are in G
- build the adjacency matrix for G from the adjacency lists, then use the solution from part a) to build the adjacency matrix for the complement (this is actually a very good solution)

If their solution works and is reasonably straightforward: 6 marks
If their solution works but is very inefficient (for example, if it traverses the old lists and the new lists multiple times): 4 marks
If their solution doesn't work: 1 to 3 marks, depending on how broken it is
c) Which of the two is more efficient? Even if they have the same big O complexity, which is likely to be faster?

Solution: the method in a) is more efficient: it runs in $O(n^2)$. The method in b) runs in $O(n^2)$ only if each adjacency list is sorted. If they are not, it runs in $O(n^2 \log n)$. Even if the method in b) can run in $O(n^2)$ it will probably be slower since it involves list operations.

Marking: part c) is worth 3 marks

If they show a reasonable understanding of which of their solutions is more efficient: 2 marks

If they recognize that array operations are faster than list operations: 1 mark
Question 3 (15 marks)

By definition, a spanning tree of a graph contains a connecting path for each pair of vertices in the graph.

Suppose G is represented by a set of adjacency lists and a spanning tree T of G is represented by a list of edges, where each edge is in the form of a pair of vertices. Given vertices x and y, how would you find the path in T that connects x and y?

You may use any well-defined data structures you choose (which unfortunately rules out the “magically solve Question 3 in one step” structure).

Solution:

The simplest method is to use the list of edges in T to build a new set of adjacency lists, just for T. Then we can use Breadth-First search, starting at x, to find the path from x to y.

Other methods could involve repeatedly traversing the edge list of T to find the neighbours of x, then their neighbours, etc (effectively doing a Breadth-First search). This achieves the same end but is much less efficient.

Marking:

There are many possible solutions – the key thing to look for is whether they have a way of using the given information (the list of edges that make up T) to find the path from x to y. If they recognize that this is what they need to do, they should get at least 8/15
If they give a solution that works but is very inefficient (such as the alternative solution suggested above) they should get about 11/15

If they give a good solution (which could use Depth-First Search instead of Breadth-First, or some other alternative I haven't thought of!) they should get at least 13/15 even if they don't name the method they are using
Question 4 (5 marks)

Catatonia is a large country with a population of 37 million people. Suppose 95% of all residents of Catatonia submit their tax returns honestly, and that the other 5% are known to have cheated on their taxes in the past. Each year the Infernal Revenue Service wants to audit the returns of the known cheaters, but does not want to audit the returns of the honest citizens.

The IRS does not mind if a few innocent citizens are audited, but will be very upset if any cheater is not audited.

What data structure would you recommend to the government of Catatonia for the purpose of quickly deciding whether a particular tax return should be audited or not? Why?

Solution: A Bloom Filter would be the best choice since it is fast, small, and achieves exactly the required performance.

Marking:

structure:

Bloom filter: 4 marks

anything else: 1 mark

why: 1 mark