Computer Science B63
University of Toronto Scarborough

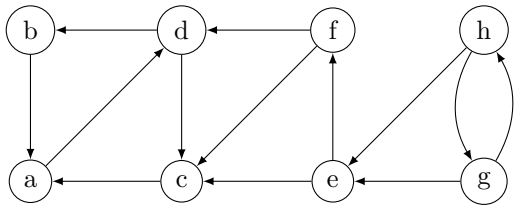
Homework Assignment # 3

Handing in and marking

For this assignment, you need to submit your solutions to the pencil-and-paper exercises on crowdmark and your solutions to the programming question on MarkUs. Your pencil-and-paper solutions will be marked with respect to correctness, clarity, brevity, and readability. Your code will be marked with respect to correctness, efficiency, program design and coding style, clarity, and readability. This assignment counts for 15% of the course grade.

Question 1. Strongly Connected Components [10 MARKS]

Consider the following graph:



1. Use the algorithm from the lecture to find the strongly connected components of this graph. Show all steps.
2. Let the vertex that you did the first DFS be u. Perform the algorithm again using some vertex v that lies in a different SCC than u. Show that you get the same SCCs. Show all steps.

Question 2. Graphs [20 MARKS]

1. Consider a directed unweighted graph $G = (V, E)$ where each vertex $v \in V$ represents a task to be done when baking a cake. An edge $(u, v) \in E$ represents that the task u must happen before the task v can be done. In other words, u is a prerequisite of v. Give an efficient algorithm that gives an ordering of the tasks such that all prerequisites of any task are already complete before the corresponding task is started. Justify that your algorithm is correct.
2. Consider a directed graph $G = (V, E)$ on which each edge $(u, v) \in E$ has an associated value $r(u, v)$, which is a real number in the range $0 \leq r(u, v) \leq 1$ that represents the reliability of a communication channel from vertex u to vertex v. Interpret $r(u, v)$ as the probability that the channel from u to v will not fail, and assume that these probabilities are independent. Give an efficient algorithm to find the most reliable path between two given vertices. Justify that your algorithm is correct.

Question 3. Minimum Spanning Trees [15 MARKS]

A student wanted to create a new algorithm to find the minimum spanning tree of a graph $G = (V, E)$.
Following is their algorithm:

```
0. T := all edges in E
1. L := edges sorted in non-increasing order by weight
4. for each (u, v) in L:
5.   T' = T - (u,v)
6.   if T' is connected:
7.     T = T'
8. return T
```

1. Prove or disprove if this algorithm correctly finds a minimum spanning tree of graph G.
2. The algorithm is very similar to the Kruskal's algorithm for finding MST that we saw in the lecture. Compare the two algorithms based on their time complexity.

Question 4. Implementing Graph Algorithms [75 MARKS]

We provided you with the starter code for implementing an undirected graph and several algorithms for it. Your task is to implement the functions declared in `minheap.h`, `graph.h`, and `graph_algos.h` that are not already implemented in `minheap.c`, `graph.c`, and `graph_algos.c`. Note that `.c` files are the only files you will submit, so make sure your implementation works with the original provided `.h` files. Make sure to **carefully study the starter code** before you begin to add your own. The marking scheme for this question is as follows:

- Correctness:
  - `MinHeap` functions `extractMin`, `insert`, `decreasePriority`: 3 marks each
  - `Graph` functions required in `graph.h`: 6 marks
  - `Edge* getMSTprim(Graph* graph, int startVertex)`: 15 marks
  - `Edge* getDistanceTreeDijkstra(Graph* graph, int startVertex)`: 15 marks
  - `EdgeList** getShortestPaths(Edge* distTree, int numVertices, int startVertex)`: 15 marks
- Program design (modular implementation, self-explanatory code, clear logic, no repeated code, no unnecessary code): 10 marks
- Readability and coding style (good use of white space, good naming, etc.): 5 marks
  - You are encouraged (but not required) to use a `linter` to help check and/or auto-format your code.

The runtime complexity requirements for your functions are as follows ($n = |V|$ and $m = |E|$):

- `extractMin`, `insert`, `decreasePriority`: $\mathcal{O}(\log n)$ time.
- `getMSTprim`: $\mathcal{O}((n+m)\log n)$ time.
- `getDistanceTreeDijkstra`: $\mathcal{O}((n+m)\log n)$ time.

Question 5. Amortized Analysis [15 MARKS]

Functional programming languages like Haskell and Scala provide immutable data items. An immutable linked list in such a language will have nodes of which the pointer values cannot change. Adding to such a list is easy (insert in the beginning), however appending to the end of such a list is problematic (since, node's next field cannot be modified). To implement a queue using such immutable data, a typical solution is to use two lists. The two lists are pointed to by `front` and `back` which are initially empty and the process of enqueue and dequeue is as follows:

```
enqueue(Q,x):
0. n = NewNode(x, back)
1. back = n
```

```
dequeue(Q):
0. if front == NULL:
1.   while back != NULL:
2.     front = NewNode(back.data, front)
3.     back = back.next
4. if front == NULL:
5.   Error "Queue underfull"
6. result = front.data
7. front = front.next
8. return result
```

`dequeue(Q)` is a $\Theta(n)$ operation if there are n items currently in the queue. Prove that both `enqueue(Q,x)` and `dequeue(Q)` are amortized $\mathcal{O}(1)$ using the 3 techniques we covered in the lecture.

1. Aggregate method
2. Accounting method
3. Potential method

Question 6. Fibonacci Heaps [20 MARKS]

1. Starting with an empty Fibonacci Heap, show a sequence of operations that results in a heap with a root node being marked.
2. Starting from an empty Fibonacci Heap, show a sequence of operations that results in a heap that is in the form of a linked list (parent-child links only) (must contain at least 4 nodes).
3. Show the result of running `extract-min` on the following Fibonacci Heap: