Computer Science B63
University of Toronto Scarborough

Homework Exercise # 3

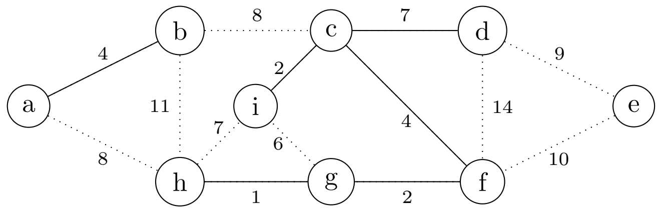
Handing in and marking

For this exercise, 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 exercise counts for 15% of the course grade.

Question 1. [15 MARKS]

Recall the example graph we used to find minimum spanning trees in class.

a. Consider the snapshot of running Kruskal's algorithm below, in which the edge `(h,i,7)` has already been processed, and we are about to process the edge `(a,h,8)`:

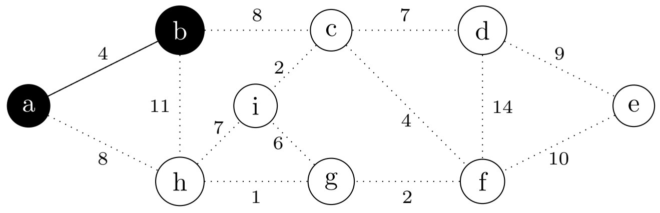


L: [(g,h,1), (c,i,2), (f,g,2), (c,f,4), (a,b,4), (g,i,6), (c,d,7), (h,i,7), (a,h,8), (b,c,8), (d,e,9), (e,f,10), (b,h,11), (d,f,14)]
Clusters: {a,b}, {e}, {d,c,i,f,g,h}
T: { (g,h), (c,i), (f,g), (c,f), (a,b), (c,d) }

Your task is to carefully verify every bullet point in the proof of correctness of Kruskal's algorithm with this snapshot. We started the process for you below:

1. Assume each of the clusters {a,b}, {e}, {d,c,i,f,g,h} is a tree.
2. Assume $T \subseteq T_{min}$ for some MST T_{min}.
   Hint: what are the options for T_{min} in this snapshot?

b. Consider the snapshot of running Prim's algorithm below, in which vertices `a` and `b` have already been dequeued, and we are about to dequeue the vertex `h`:



PQ:

vertex	h	c	d	e	f	g	i
priority	8	8	∞	∞	∞	∞	∞
pred	a	b					

T: { (a,b) }

Your task is to carefully verify every bullet point in the proof of correctness of Prim's algorithm with this snapshot. We started the process for you below:

1. Assume T contains vertices `a,b` and PQ contains vertices `c,d,e,f,g,h,i`.
2. Assume for each v in PQ, $priority(v)$ = minimum weight of any edge between v and T.
3. Assume $T \subseteq T_{min}$ for some MST T_{min}.
   Hint: what are the options for T_{min} in this snapshot?

Question 2. Negative Weights [20 MARKS]

In the lectures, edge weights in weighted graphs were non-negative numbers. In this question, we explore allowing negative weights. If we think of a positive weight as paying a price, then a negative weight means getting a rebate!

a. Prove or disprove: the modified Prim's algorithm below outputs a minimum spanning tree, if weights may be negative.

```
0. modified Prim's:
1.   minimum_weight := minimum weight over all edges
2.   c := max(-minimum_weight, 0)
3.   define new weights weight' as follows:
4.     for all u,v: weight'(u,v) := weight(u,v) + c
5.   run original Prim's using new weights weight'(u,v) for all u,v
6.   claim the output is also correct for original weights weight(u,v)
```

b. For shortest-path problems, before we even consider any algorithm, we need to think through the problem statement itself. If the problem statement does not require *simple* paths (i.e., if it allows paths going through some vertex or edge multiple times), what can go wrong with negative weights? Include a small concrete example to illustrate your explanation.

c. Prove or disprove: Dijkstra's algorithm outputs single-source shortest simple paths, if weights may be negative.

d. Prove or disprove: the modified Dijkstra's algorithm below outputs single-source simple paths, if weights may be negative.

```
0. modified Dijkstra's:
1.   minimum_weight := minimum weight over all edges
2.   c := max(-minimum_weight, 0)
3.   define new weights weight' as follows:
4.     for all u,v: weight'(u,v) := weight(u,v) + c
5.   run original Dijkstra's using new weights weight'(u,v) for all u,v
6.   claim the output is also correct for original weights weight(u,v)
```

Question 3. Maximum Problems [10 MARKS]

In this question, we explore the problems of *maximum* spanning trees and *longest* simple paths; this means we seek to *maximise* total weights instead of minimising. For longest paths, we allow only simple paths obviously. Edge weights are non-negative in this question for simplicity.

a. Give an algorithm of the following form for computing a maximum spanning tree, and prove that it is correct.

   1. compute new weights w' from original weights w
   2. run Kruskal's or Prim's *as a black box* with w' for weights
   3. claim its output is a maximum spanning tree for original w

   The correctness proof should be about why a minimum spanning tree under w' is a maximum spanning tree under w, and should not depend on whether stage 2 uses Kruskal's, Prim's, or any other algorithm for minimum spanning trees.

b. Prove or disprove: the modified Dijkstra's algorithm below computes single-source longest simple paths.

- change min-heap to max-heap
- change `extract-min` to `extract-max`
- change `decrease-priority(v, d')` to `increase-priority(v, d')`
- for vertices v other than the start vertex, initial priority and $v.d$ is $-\infty$
- change $d' < v.d$ to $d' > v.d$

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.