**Warm-Up**: Draw the string "PINHEAD" as a max heap. Then perform an `extract-max` operation. What is the string representation after you do so?

**Question 1.** Suppose you have two min-heaps, $A$ and $B$, with a total of $n$ elements between them. You want to discover if $A$ and $B$ have a key in common. Give a solution to this problem that takes time $\mathcal{O}(n \log n)$. For this problem, do not use the fact that heaps are arrays. Rather, use the API of a heap (e.g., calls to `A.min()` or `B.removeMin()`). For full credit, your solution should use $\mathcal{O}(1)$ additional memory.

Give a brief explanation for why your code has the required running time.

We will then cover Dijkstra's Algorithm; see the back for that. However, for space reasons, I am listing the extra practice questions on this side.

## Extra Practice

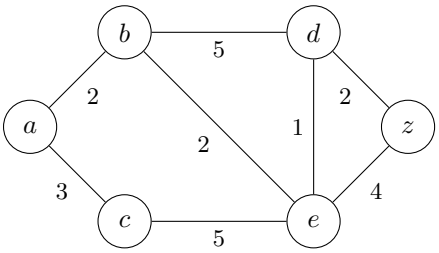
I encourage you to do the following problems from the textbook of Goodrich and Tamassia as extra practice. R-8.12, R-8.16, R-8.21, R-8.22, R-8.24, C-8.10, C-8.14

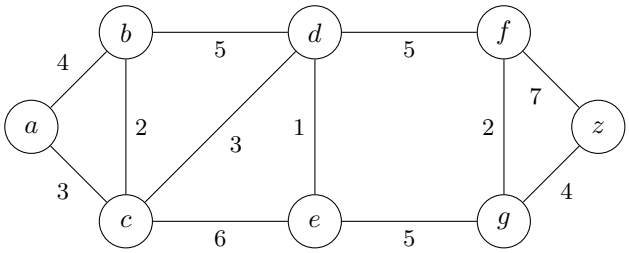
In addition, here are some more problems:

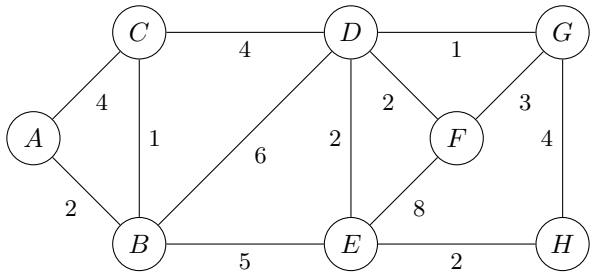
1. How do we sort (put into order) a vector using a heap? Can you do this with only $\mathcal{O}(1)$ additional memory rather than $\mathcal{O}(n)$?

   *We will discuss this in the next unit, during the lectures after exam three. However, you may want to think about it in the meantime, see if you can discover the so-called "HeapSort" on your own!*

2. If you wish to practice Dijkstra's Algorithm on some more graphs, here are a few you can use. You can also reuse the graph from earlier if you need more, and select a different starting point.






Dijkstra's Algorithm takes as input a weighted graph with positive edge weights [1] and a designated vertex $s$ to signify the "starting point." The output is a **tree** such that the unique path from $s$ to any other vertex $v$ in the tree is the shortest (lowest-cost) path in the original graph.

```
Dijkstra's (Single-Source, Shortest Path Tree) Algorithm
  for each vertex v do
    intree(v) = false
    parent(v) = N/A
    dist(v) = ∞
  end for
  dist(s) = 0
  while ∃ vertex u with intree(u) = false do
    u ← vertex with intree(u) = false and smallest dist(u)
    intree(u) = true
    for each vertex v ∈ adj[u] do
      if dist(v) > dist(u) + w(u, v) then
        dist(v) = dist(u) + w(u, v)
        parent(v) = u
      end if
    end for
  end while
```

This algorithm keeps track of three pieces of information for each vertex: whether or not it's in the shortest path tree so far, which vertex either is its parent in that tree, or would be if it were added right now (using the best path found so far), and what the distance from the start vertex to it is (along edges viewed so far).

| v | intree(v) | parent(v) | dist(v) |
|---|---|---|---|
| A | | N/A | 0 |
| B | | | |
| C | | | |
| D | | | |
| E | | | |
| F | | | |
| G | | | |
| H | | | |

[1] the way to solve this when edge weights can be negative is more complicated.