Problem Set 8 and Last!

Assigned: Nov. 28
Due: Dec. 5

In all of the following, it may be assumed that the hash table operations \( \text{put}(k,v) \) and \( \text{get}(k) \) execute in constant time. This is not guaranteed to be true in the worst case, but is true with very high probability.

Problem 1

The class notes on “Combining Data Structures”, in the section entitled “Adding a hash table to pretty much anything” describe a data structure that combines a hash table with a maxHeap.

Show how the following two operations can be implemented with the specified running time. You do not have to change the general definition of the data structure, though of course each of the operations will modify the particular data structure. Your implementations must of course leave the data structures in a consistent state.

- **rename(Job j, String name)** — Change the name of \( j \) to \( \text{name} \). Running time: \( O(1) \).
- **revalue(Job j, int value)** — Change the value of \( j \) to \( \text{value} \). Running time: \( O(\log n) \).

Problem 2

A. Show the adjacency array for the subgraph of the above graph with vertices \{ A, B, C, D, E \}

B. Show how to compute the shortest paths from vertex B to all the other vertices using breadth-first search. Specify the sequence of adds and pops from the queue; the back-pointers from each vertex; and the distance of each vertex from B.
Problem 3

Suppose that you want to implement a class `StringSet`. Each `StringSet` will represent a set of `Strings`, and each `StringSet` will have a name. The name is not an element of the set; for instance, you might have a `Set` with the name “States” whose elements are { “Alabama”, “Alaska” … “Wyoming” }. Assume that any two different `Set` objects have different names; however, they may represent the same set of `Strings`.

Describe an implementation for `StringSet` that supports each the following operations in the specified run time. (This must be a single implementation that supports all the operations, not a different implementation for each operation.)

You may define additional classes, if you find that helpful.

- `StringSet emptySet(String name)` — Create an empty set of the specified name. Running time: $O(1)$.
- `StringSet findSet(String name)` — Find the `Set` of the specified name. Running time: $O(1)$.
- `void add(StringSet s, String x)` — Add `String x` to `StringSet s`. Running time: $O(1)$.
- `boolean element(String x, StringSet s)` — Check whether `x` is an element of `s`. Running time: $O(1)$.
- `void delete(StringSet s, String x)` — Delete `String x` from `StringSet s`. Running time: $O(1)$.
- `int count(StringSet s)` — Return the number of elements in `s`. Running time: $O(1)$.
- `int setCount(String x)` — Return the number of `StringSets` that contain `x`. Running time: $O(1)$.
- `void vaporize(String x)` — Remove `x` from all the sets that contain it. Running time: $O(x.setCount())$.
- `void vaporizeSet(Set s)` — Entirely eliminate `s`. Running time: $O(s.count())$. 