CS 61B Fall 2026

Hashing, Midterm Review

Discussion 10: November 2, 2026

1 A Side of Hash Browns

We want to map food items to their yumminess. We want to be able to find this information in constant time, so we've decided to use Java's built-in **`HashMap`** class! Here, the key is an **`String`** representing the food item and the value is an **`int`** yumminess rating.

For simplicity, let's say that here a **`String`**'s hashcode is the first letter's position in the alphabet (A = 0, B = 1... Z = 25). For example, the **`String "Hashbrowns"`** starts with "H", and "H" is 7th letter in the alphabet (0 indexed), so the hashCode would be 7. Note that in reality, a **`String`** has a much more complicated **`hashCode`** implementation.

Our **`HashMap`** will compute the index as the key's hashcode value modulo the number of buckets in our **`HashMap`**. Assume the initial size is 4 buckets, and we double the size of our **`HashMap`** as soon as the load factor reaches 3/4. If we try to put in a duplicate key, simply replace the value associated with that key with the new value.

(a) Draw what the **`HashMap`** would look like after the following operations.

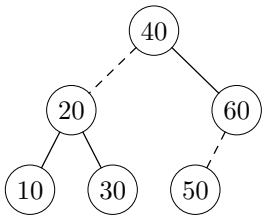
```
HashMap<String, Integer> hm = new HashMap<>();
hm.put("Hashbrowns", 7);
hm.put("Dim sum", 10);
hm.put("Escargot", 5);
hm.put("Brown bananas", 1);
hm.put("Burritos", 2);
hm.put("Buffalo wings", 8);
hm.put("Banh mi", 9);
hm.put("Burritos", 10);
```

(b) Do you see a potential problem here with the behavior of our **`HashMap`**? How could we solve this?

2 LLRBs

from Fall 2022 Midterm 2

Consider the below integer LLRB. Like typical LLRBs, assume all values are unique.



(a) Draw the corresponding 2-3 tree.

(b) Give all integers which, when inserted into the LLRB, result in a single `rotateLeft` operation and no further cascading operations (i.e. no `rotateLeft`s, `rotateRight`s or `colorFlip`s). If this is not possible, write "impossible".

If it is possible, use **inclusive square bracket notation**, e.g. if your answer is "`[62, 64]` and `[66, 68]`", this is equivalent to saying 62, 63, 64, 66, 67, 68.

(c) Give all integers which, when inserted into the LLRB results in a single `rotateRight` operation and no further cascading operations (i.e. no `rotateLeft`s, `rotateRight`s or `colorFlip`s). If this is not possible, write "impossible". If it is possible, use inclusive square bracket notation.

(d) At most, how many additional inserts can be performed into the LLRB above without triggering any `colorFlip` operations? Do not count the operation that results in the first color flip.

3 I Am Speed

For each code block below, fill in the blank(s) so that the function has the desired runtime. If the answer is impossible, just write "impossible" in the blank. Assume that `System.out.println` runs in constant time. You may use Java's `Math.pow(x, y)` to raise `x` to the power of `y`.

(a) Desired Runtime: $\Theta(N)$

```
public static void f1(int N) {
    for (int i = 1; i < N; _____){
        System.out.println("hi Selina");
    }
}
```

(b) Desired Runtime: $\Theta(\log N)$

```
public static void f2(int N) {
    for (int i = 1; i < N; _____) {
        System.out.println("howdy Caitlin");
    }
}
```

(c) Desired Runtime: $\Theta(1)$

```
public static void f3(int N) {
    for (int i = 1; _____; i += 1) {
        System.out.println("hello Dawn");
    }
}
```

(d) Desired Runtime: $\Theta(2^N)$

This one is tricky! *Hint: think about the dominating term in* $1 + 2 + 4 + 8 + ... + f(N)$

```
public static void f4(int N) {
    for (int i = 1; _____; i *= 2) {
        for (int j = 0; j < i; j += 1) {
            System.out.println("what's up Alyssa");
        }
    }
}
```

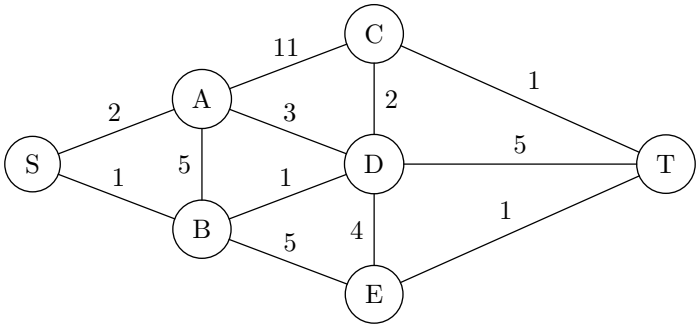
4 Lost in the Ceryneian H.I.N.D.

from Spring 2024 Final

After being repeatedly hacked in past exams, you've finally tracked down the hackers responsible as members of the Ceryneian H.I.N.D. With the assistance of an investigator (codenamed Heracles), you have managed to infiltrate their internal network. Help Heracles catch the Ceryneian H.I.N.D.

The internal network is a **weighted, undirected graph**, with start vertex S and target vertex T. You want to delete one edge from the graph, such that when deleted, the shortest path length from S to T is increased by as much as possible.

For example, consider the graph below:



Before deleting an edge, the shortest path from S to T has length 5. If we delete edge BS, the shortest path from S to T will instead have length 8; deleting any other edge will result in a shortest path from S to T that is shorter than 8. Thus, we should delete edge BS.

Design an algorithm (in English or pseudocode) that, given a graph, returns the edge that should be deleted. If multiple edges yield the same maximal shortest-path length, your algorithm may return any of them. You may assume that:

- All edges have positive integer edge weights.
- $S \neq T$
- The input graph is connected, and cannot be disconnected by removing any one edge.
- You have access to a working black-box implementation of Dijkstra's algorithm, which runs in $\Theta(E \log V)$ time.

For credit, your algorithm must run in $O(VE \log(V))$ time, where V and E are the number of vertices and edges in the graph, respectively.

Hint: Try to devise a naive solution first, then optimize that solution down to the specified runtime bound.