CS 61BL Summer 2021

Final Exam Review

Monday August 9, 2021

1 Hashing Asymptotics

Suppose we set the `hashCode` and `equals` methods of the `ArrayList` class as follows.

```
/* Returns true iff the lists have the same elements in the same ordering */
@Override
public boolean equals(Object o) {
    if (o == null || o.getClass() != this.getClass() || o.size() != this.size()) {
        return false;
    }
    ArrayList<T> other = (ArrayList<T>) o;
    for (int i = 0; i < this.size(); i++) {
        if (other.get(i) != this.get(i)) {
            return false;
        }
    }
    return true;
}

/* Returns the sum of the hashCodes in the list. Assume the sum is a cached instance variable. */
@Override
public int hashCode() {
    return sum;
}
```

(a) Give the best and worst case runtime of `hashContents` in $\Theta(.)$ notation as a function of `N`, where `N` is initial size of the `list`. Assume the length of `set's` underlying array is `N` and the `set` does **not** resize. Assume the `hashCode` of an `Integer` is itself. Admittedly, the `ArrayList` class does not have the method `removeLast`, but assume it does for this problem, and is implemented the same as in Project 1. Finally, assume `f` accepts two **int**s, returns an unknown **int**, and runs in constant time.

```
static void hashContents(HashSet<ArrayList<Integer>> set, ArrayList<Integer> list) {
    if (list.size() <= 1) {
        return;
    }
    int last = list.removeLast();
    list.set(0, f(list.get(0), last));
    set.add(list);
    hashContents(set, list);
}
```

Best Case: $\Theta($ $\quad$ $)$, Worst Case: $\Theta($ $\quad$ $)$

(b) Continuing from the previous part, how can we define `f` to **ensure** the worst case runtime? How can we define `f` to **ensure** the best case runtime? There may be multiple possible answers.

1. Worst case:

```
int f(int first, int last) {
    return __________________________;
}
```

2. Best case:

```
int f(int first, int last) {
    return __________________________;
}
```

2 Sorted Runtimes

We want to sort an array of N **unique** numbers in ascending order. Determine the best case and worst case runtimes of the following sorts:

(a) Once the runs in merge sort are of $size <= N/100$, we perform insertion sort on them.

Best Case: $\Theta(\quad)$, Worst Case: $\Theta(\quad)$

(b) We can only swap adjacent elements in selection sort.

Best Case: $\Theta(\quad)$, Worst Case: $\Theta(\quad)$

(c) We use a linear time median finding algorithm to select the pivot in quicksort.

Best Case: $\Theta(\quad)$, Worst Case: $\Theta(\quad)$

(d) We implement heapsort with a min-heap instead of a max-heap. You may modify heapsort but must maintain constant space complexity.

Best Case: $\Theta(\quad)$, Worst Case: $\Theta(\quad)$

(e) We run an optimal sorting algorithm of our choosing knowing:

- There are at most N inversions

  Best Case: $\Theta(\quad)$, Worst Case: $\Theta(\quad)$

- There is exactly 1 inversion

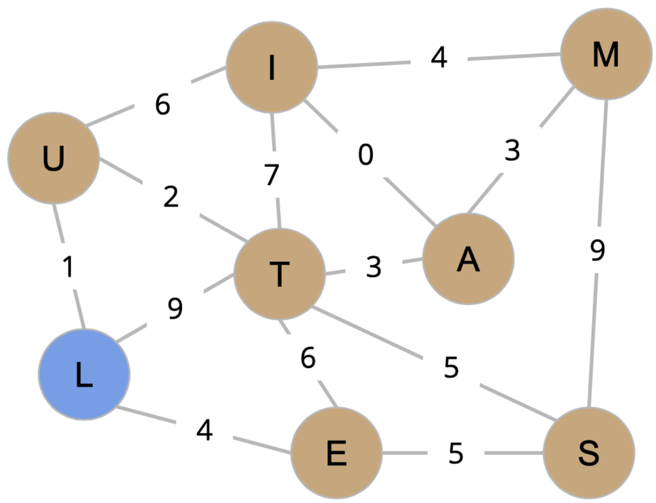
  Best Case: $\Theta(\quad)$, Worst Case: $\Theta(\quad)$

- There are exactly $(N^2 - N)/2$ inversions

  Best Case: $\Theta(\quad)$, Worst Case: $\Theta(\quad)$

3 Dijkstra's and A*

Given the graph below, answer the following questions:



(a) What edges are in the shortest paths tree (SPT) starting from **L**?

(b) Decreasing **which edge** by 2 changes the SPT from **L**? Assume the SPT tree was created by running Dijkstra's from **L**. There may be more than one correct answer, determine **all**!

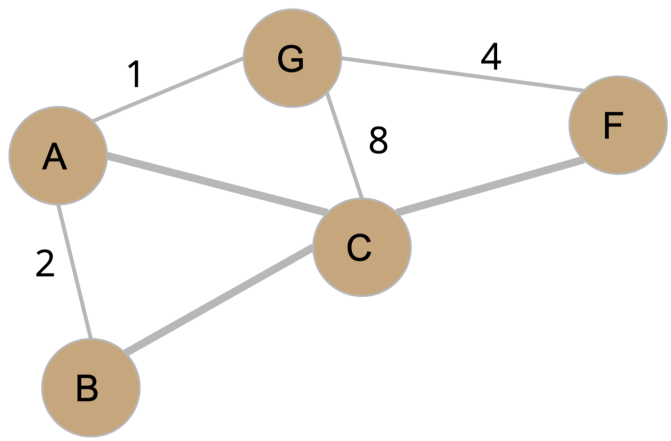
(c) We will define the heuristic of a vertex v as the shortest distance from v to I. For instance, the heuristic of T is 3.

Given that I is the end vertex, what start vertex would visit the most vertices on one run of A*? Recall that A* terminates after removing the goal. If multiple answers produce the maximum, select all.

4 Prim's

(a) In an arbitrary graph, Prim's can change the priority of a vertex v in the priority queue a **maximum** of ______ times and a **minimum** of ______ times. Assume **v** is not the start vertex and the graph is connected and undirected. Give **tight** bounds specific to **v**. Assume we set all priorities to infinity initially.

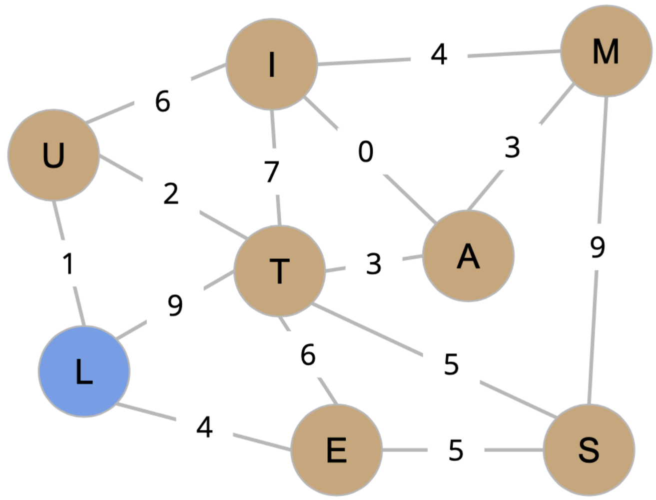
(b) Suppose we run Prim's from A on the graph below.



Fill in the missing edges in the graph to the right so that

1. The priority of C is changed the **maximum** number of times, i.e. the first blank from above.
2. The priority of every vertex is changed the **minimum** number of times, i.e. the second blank from above.

5 Kruskal's



(a) We want to run Kruskal's, but we have no cycle detection, so we terminate upon inserting $V - 1$ edges. Will this produce a valid MST on the graph above? If not, determine which edge(s) need to be changed, and to what. If there are many possibilities, choose the one that involves the minimum added/removed weight.

Assume ties are broken alphabetically, and edges are written in alphabetical order, and compared as such. For instance, if edges (A, Z) and (E, H) are equal, (A, Z) would be chosen before (E, H).

(b) After completing the previous part, Sohum wondered if it's possible to run Kruskal's with limited cycle detection. More specifically, he pondered: what if we can only detect a maximum of **k** cycles during one run of Kruskal's?

Looking at the specific instance of a 6 vertex graph, what is the **minimum** value of **k** for which we can ensure that Kruskal's will always work?

6 Graph Algorithm Design

Given a **undirected**, **weighted** graph G with **positive**, **integer** edge weights, we want to find a path from **u** to **v** that minimizes the total cost. For each "catch" below, find the path of optimal cost no slower than $O(ElogV)$.

(a) Excluding the start and end vertex, we partition the vertices into 5 subsets, and we must visit vertices in order of their subset. That is, if we are in subset **k**, the next vertex we visit must be in subset $\mathbf{k+1}$.

(b) We must visit two designated vertices s and k on our path.

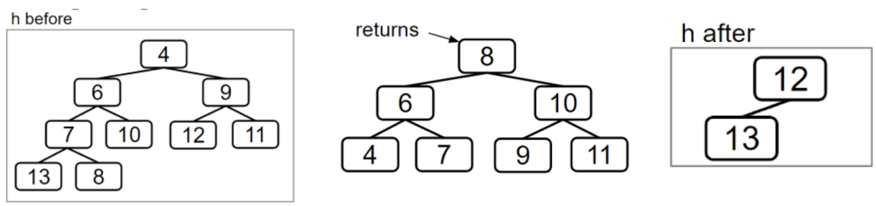
(c) If two paths from u to v are of the same cost, we will choose the path with fewer edges.

(d) Instead of starting from u and ending at v, we can start from any vertex in a subset of vertices and end at any vertex in a subset of vertices. Each subset is of size k.

7 Yggdrasil (Heaps)

This problem was taken from Spring 2019 Midterm 2.

This is a very challenging problem. Write a function that takes an integer `k` and a min-heap `h` (in tree representation) and removes the **k smallest values** and returns them organized into **valid perfectly balanced BST**. For example, if we call `heapToBBST(7, h)` on the MinHeap in the left figure, it returns the Tree in the middle figure, and as a side-effect, `h` becomes the MinHeap in the right figure.



This should be done in-place, i.e., reusing the TreeNodes from the min-heap. Your function should complete in $O(NlogN)$ time, where N is the number of items in the min-heap, and use no more than $O(logN)$ additional memory while it is running. For full credit, it must work for arbitrary values for k, but you can earn almost full credit if your solution works for $k = 2^H - 1$ (i.e. powers of 2 minus 1).

```
public class TreeNode {
    public int item;
    public TreeNode left;
    public TreeNode right;
}
public class MinHeap {
    /* Even though a MinHeap is made up of TreeNodes, the instance
    * variables are private. You cant directly access them. */

    /* removes the minimum node and returns it */
    public TreeNode removeMin() { /* ... */ }
}
public static TreeNode heapToBBST(int k, MinHeap h) {
    if (k == 0) {
        return null;
    }
    _________________________________
    _________________________________
    _________________________________
    _________________________________
    _________________________________
    _________________________________
    _________________________________
    _________________________________
    _________________________________
}
```