CS2040 Tutorial 8

Week 10, starting 17 Oct 2022

Q1 AVL Tree

n(h) = n(h-1) + n(h-2) + 1

(a) What is the minimum number of nodes that an AVL tree of height 5 can have?

0->1;1 -> 2; 2 -> 4; 3 -> 7; 4->12; 5-> 20

(b) Because of the AVL property, the longest (simple) path from root to a leaf, minus the shortest (simple) path from root to a leaf cannot be 2 or more. True or false? F

12?

(c) What is the minimal AVL tree height needed for deletion of one node to cause 2 rebalancing operations? 3 rebalancing operations? A rebalancing operation may involve one or two rotations, to resolve AVL property violations at a given node.

Q2 Heap Update

You are given a binary min heap implementation below:

```
class MinHeap {
   ArrayList<Integer> data;

   MinHeap() { this(new ArrayList<>()); }
   MinHeap(ArrayList<Integer> from) {
      data = new ArrayList<>(from); // create new object, preserves orig.
      for (int idx = data.size()/2 - 1; idx >= 0; idx--) bubbleDown(idx);
   }

   int size() { return data.size(); }
   boolean isEmpty() { return data.size() == 0; }
   int peek() { return data.get(0); }

   void offer(int key) {
      data.add(key);
      bubbleUp(data.size() - 1);
   }
   int poll() {
      if (data.size() == 1) return data.remove(0);
      int ans = data.get(0);
      data.set(0, data.remove(data.size() - 1));
      bubbleDown(0);
      return ans;
   }
   void update(int idx, int newKey) {} // to implement
```

```
  private void bubbleUp(int idx) {
    while (idx != 0 && data.get(parent(idx)) > data.get(idx)) {
      swap(parent(idx), idx);
      idx = parent(idx);
    }
  }
  private void bubbleDown(int idx) {
    while (left(idx) < data.size()) {
      int smallerIdx = left(idx);
      if (smallerIdx + 1 < data.size() && // check if rightC exists
            data.get(smallerIdx) > data.get(smallerIdx + 1))
        smallerIdx++; // and if rightC is smaller

      if (data.get(idx) <= data.get(smallerIdx))
        break;

      swap(idx, smallerIdx);
      idx = smallerIdx;
    }
  }

  private int parent(int idx) { return (idx - 1) / 2; }
  private int left(int idx) { return 2 * idx + 1; }
  private int right(int idx) { return 2 * idx + 2; }

  private void swap(int left, int right) {
    int newLeft = data.get(right);
    data.set(right, data.get(left));
    data.set(left, newLeft);
  }

  void print() { System.out.println(data.size() + ": " + data); }
}
```

(a) Efficiently implement the `void update(int idx, int newKey)` method in the MinHeap class that replaces the element at the given index `idx` with the `newKey` and maintaining the minimum heap by the end of the operation.

O(h)

(b) Would `void update(int oldKey, int newKey)` which replaces any one occurrence of the element `oldKey` with `newKey` run in the same time complexity? If not, how can the MinHeap class be modified such that this new operation's efficiency is improved, supposing all heap elements are distinct?

hashmap key -> index?

Q3 Greed is Good

You are given a positive integers **k**, **N** (k ≤ **N** and **k**, **N** could be large) cards in a line, each with an integer number, face up. You are given as many *turns* as you want to pick a set of cards with the largest sum, and output this sum. Each turn, you may only pick a card that is within **k** cards from the left of the line.

e.g. when **k** = 2, **N** = 5, cards = [2, -10, 2, -6, 5], the output will be 4
e.g. when **k** = 5, **N** = 6, cards = [-1, -1, -1, -1, -1, 10], the output will be 9

Design and implement a solution to **efficiently** compute this largest sum, and state its time complexity.

Question 4 (Online Discussion) – Best Team

There are **N** candidates being considered for selection in a competition. Each student **s** has a:

- individual skill $\mathbf{D_s}$
- team-player skill $\mathbf{T_s}$

both being positive integers.

You are given a positive integer **L** (**L** ≤ **N** and **L**, **N** could be large), which is the upper limit on the number of candidates that can be selected, as well as the skill levels of the **N** candidates $\mathbf{D_1}$ $\mathbf{T_1}$ $\mathbf{D_2}$ $\mathbf{T_2}$ … $\mathbf{D_N}$ $\mathbf{T_N}$

Some people just cannot work with others and pull the whole team down. A team is as good as its weakest link, so you judge that the best team will have the highest value of:

$$\sum_{i \in Team} D_i \times \min_{i \in Team} T_i$$

Efficiently find and output this highest value.

e.g. when **L** = 4, **N** = 5, candidates(Ds, Ts) = [(1, 3), (8, 1), (2, 3), (4, 2), (3, 3)], the best team score is 20
e.g. when **L** = 4, **N** = 5, candidates(Ds, Ts) = [(1, 3), (99, 1), (2, 3), (4, 2), (3, 3)], the best team score is 108
e.g. when **L** = 2, **N** = 5, candidates(Ds, Ts) = [(1, 3), (8, 1), (2, 3), (4, 2), (3, 3)], the best team score is 15