Overview

- aggregate method (amortization)
  - stars
  - a generalized example (Q2)
- potential function method: binary counter
- AVL-tree review and practice

Problems

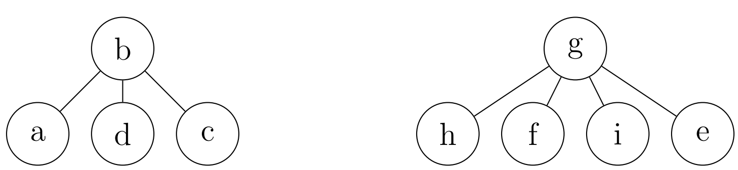
Binary counter. A *binary n-bit counter* counts upward from zero as an array n bits (the leftmost bit is least significant). It supports the operation *increment*, which adds 1 to the counter:

```
increment(A[0..n-1]):
    i = 0
    while(A[i] != 0):
        A[i] = 0
        ++i
    A[i] = 1
```

The running time for *increment* is $\Theta(k)$, where k is the final value of variable i, which is $\Theta(n)$ in the worst case. Show the amortized cost per *increment* of $\Theta(1)$.

Aggregate analysis. Suppose any sequence of n operations on a data structure has the property that the i-th operation costs $i \log i$ if i is an exact power of 2, and 1 otherwise. Show the amortized cost per operation of $O(\log n)$.

Stars. We have a data structure to maintain collection of stars (height-1 trees).



Every child knows its parent. It supports three operations:

- *new-star*(x) : creates a new star whose only member is x
- *find-star*(x) : returns a handle to the root of the star containing x

- *merge*(x, y) : merges the stars that contain x and y

new-star(x) is implemented in constant worst-case time by simply creating a new star with x as its only element. Similarly, *find-star*(x) is implemented in constant worst-case time by returning x's parent pointer.

The operation *merge*(x, y), however, can be slow: it sets the parent pointer of all elements of y's star to *find-star*(x), in time proportional to the size of y's star (i.e. the number of element's in y's star).

Let n be the number of objects currently stored.

(a) Construct a sequence of $\Theta(n)$ operations that requires $\Theta(n^2)$ time.

Hence, conclude that the amortized cost of all operations using the aggregate method is $\Theta(n)$.

(b) We may *augment* this data structure with a *size* field at the root: now every root knows the size of its star. Now rather than breaking ties arbitrarily during *merge*, we always set the parent pointers of a smaller star.

Show using the aggregate method that the amortized runtime of all operations is $O(\log n)$.

Hint: argue that any sequence of m *new-star*, *find-star*, and *merge* operations, n of which are *new-star* operations take $O(m + n \log n)$ time.

2-AVL tree. Let a 2-AVL tree be a binary search tree where for every node, the difference of heights of its left and right subtree is at most 2. Prove that a 2-AVL tree has height at most $3 \log n$ where n is the number of nodes in the tree.

Balanced BST. Recall that a binary search tree is called *perfectly balanced* if for every node v we have

$$|v.\text{left.size} - v.\text{right.size}| \leq 1,$$

i.e., the size-difference between the left and right is as small as possible. Show that in any perfectly balanced binary search tree T, the leaves are only on the bottom two levels.

Hint: First consider the case where $n = 2^k - 1$ for some integer k. Then consider the case where $n = 2^k$ for some integer k. Finally for arbitrary n, let k be the integer with $2^k \leq n < 2^{k+1}$. In all three cases, what are the sizes of the subtrees, and hence where are the leaves, relative to k?