Tutorial 8

Interpolation search, bisection method, ternary search
CS 240E W23
University of Waterloo
Monday, March 13

1. **Ancestors in min-oriented heap.** We are given a tree (not necessarily binary) with the min-heap ordering property. At every node, we store a list of its children (as with a trie).

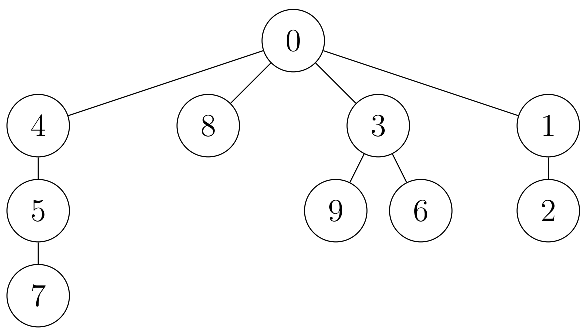

Figure 1: An example tree.

The problem is to answer Q offline queries of the form: find the ancestor of x with key at least k that is *closest to the root* (given x and k). Give an algorithm to solve this problem in $O(n + Q \log n)$ time.

2. **Bisection method.** Given a function f that:

- takes an *integral* argument,
- is *monotone* on $\{a, \ldots, b\}$ (for given $a \leq b$),
- has the property that $f(a) = 0$ and $f(b) = 1$;

we would like to find the smallest $x \in \{a, \ldots, b\}$ such that $f(x) \geq 1$.

Suppose we are able to compute $f(v)$ for any $v \in \{a, \ldots, b\}$ in constant time. Give an algorithm to find x in time $O(\log(b - a))$.

3. **Ternary search.** Given a function f that:

- takes a *floating point* argument,
- is *unimodal* on $[lo, hi]$;

we want to find $lo \leq x \leq hi$ such that $f(x)$ is minimum. Give an algorithm to achieve find x in $\Theta(\log(hi - lo))$ assuming we can compute $f(v)$ in constant time for any $v \in [lo, hi]$.

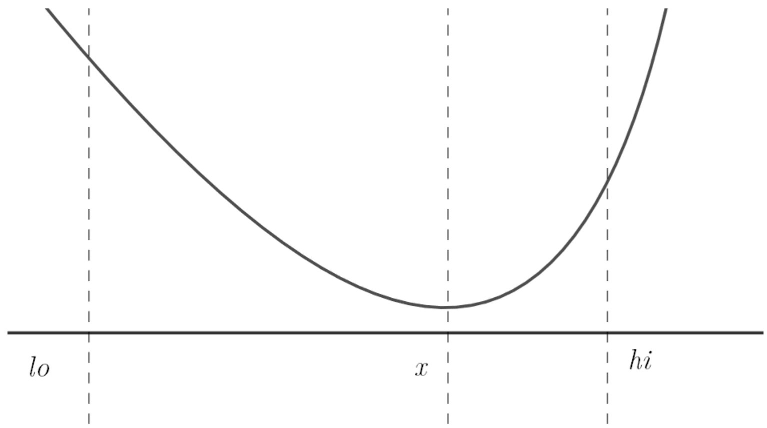

Figure 2: A function unimodal on $[lo, hi]$.

Note: we say f is unimodal on $[lo, hi]$ if:

- for all a, b with $lo \leq a < b \leq x$, we have $f(a) > f(b)$, and
- for all a, b with $x \leq a < b \leq hi$, we have $f(a) < f(b)$.

4. **Improving interpolation search.** Our goal for this problem is to improve the worst-case runtime of interpolation search, and to simplify its analysis.

   (a) Give an instance with 10 elements such that interpolation search makes a comparison at every element.

   (b) Give an instance of size n that achieves runtime $\Omega(n)$.

   We know from lecture, the average case runtime of interpolation search (if keys are uniformly distributed) is in $\Theta(\log \log n)$. In tutorial, we will improve interpolation search to achieve $\Theta(\sqrt{n})$ worst-case runtime and $\Theta(\log \log n)$ average-case runtime. The material we will discuss is in Section 6.1.4 of [Biedl].