Some properties of binary trees

Defn: A proper binary tree is one in which each node has either 0 or 2 children.

Thm: If $T$ is a (proper) binary tree of height $h$ and with $n$ nodes

1) $h+1 \leq \text{# ext nodes} \leq 2^h - 1$
2) $h \leq \text{# int nodes} \leq 2^{h-1}$
3) $2^{h-1} \leq \text{# nodes} \leq 2^{h-1} + 1$
4) $\log(n+1) - 1 \leq h \leq \log(n)/2$
Proof 9(i) by induction

14. A tree of ht ≥ h−1 has ≥ k int nodes

15. Consider a tree of ht ≥ h

By 14. K has ≥ h−1 int nodes

So T has int nodes

= 1 + #ht(L) + #ht(R)
Lower bound tree

Upper bound
Prop: In a proper binary tree $T$, $\# \text{ext} = \# \text{int} + 1$.

If:

$\# \text{ext}(L) = \# \text{int}(L) + 1$

$\# \text{ext}(R) = \# \text{int}(R) + 1$

$\# \text{ext}(T) = \# \text{ext}(L) + \# \text{ext}(R)$

$= \# \text{int}(L) + \# \text{int}(R) + 2$

$\# \text{int}(T) = \# \text{int}(L) + \# \text{int}(R) + 1$

Euler Traversal.
30.08.2018

Given a set \( X \), a relation \( R \) is a subset of \( X \times X \).
- A relation \( R \) is called
  i) reflexive if \( (x,x) \in R \) \( \forall x \in X \)
  ii) antisymmetric if \( (x,y) \in R \text{ and } (y,x) \in R \Rightarrow x = y \)
  iii) transitive if \( (x,y) \in R \text{ and } (y,z) \in R \Rightarrow (x,z) \in R \)
- A relation \( R \) is called a partial order if it has
  i) \( R \)
  ii) \( R \)
  iii) \( R \)
- A partial order \( R \) is called a total order if it has
  i) \( (x,y) \in R \text{ or } (y,x) \in R \)
  ii) \( \forall x \in X \)
Priority Queue

\[ N = A \times T \]

\[ \text{where } T \text{ is a totally ordered set.} \]

A is the ground set of some ADT.

\[ (a, k) \]

\[ a \text{ data, } k \text{ priority or key.} \]

\[ \text{A set } T \text{ in } \]
\[ \text{called a T.O.} \]
\[ \text{Set of } \leq \text{ relation} \]
\[ \text{is called a} \]
\[ \text{total order on } T. \]
Operations:

1. Insert \( (a, k) \)

2. Remove Min \( (P) = \text{remove} (a_i, k_i) \)
   where \( i = \arg \min_k k_j \)

3. Insert Min \( (P) \)

\[\begin{align*}
\frac{1}{2} \bigcup \{0, 0.5, 1\}, \{0, 0.3, 0.1\} \\
\end{align*}\]
Sort using a PQ

*Given a set \( H \) of numbers*

\* - Insert all \( |H| \) numbers in PQ as \((i,i)\)

\* - Remove Min \( |H| \) times.

1) Insert \( O(|i|) \)

Remove \( O(|i|) \)

**InPlace** SelectionSort

Insert: \( \sum i \)

Remove: \( \frac{|H| \cdot (|H| + 1)}{2} \)

\( \Theta (|H|^2) \)

\( \Theta (|H|^2) \)
Heap order property:

Nodes: Key at node ≤ keys at all children

Remove min:
- Return root and
  empty the root
- Move min of children to root
  and recursively fix the subtree whose
  root now empty.

Time ≤ \( h \times \log_2(\frac{n}{h}) \)
Inert

- Attach new key at leaf.
- If new key > parent key oth

swap with parent and

Time $O(\log n)$

Heap structure propery
Remove min
- remove last node on lowest level
- put it in the empty root
- bubble down
- WLT the key violates the
  10 property swap with
  min of children

\[ \text{Time} = O(\text{ht}(T) \cdot \#\text{child}(T)) \]

\[ \text{ht}(T) = \lceil \log \left( \frac{\#\text{nodes}(T)}{2} + 1 \right) \rceil - 1 \]

\[ = \Theta(\log m) \quad m = \#\text{nodes}(T) \]
\[ L_{\text{child}}(i) = 2i + 1 \]
\[ L_{\text{child}}(i) = 2i + 2 \]

\[ \sum_{i=0}^{d-1} 2^i = 2^d - 1 + kH! \]

\[ = 2^d - 1 + kH! \]

\[ = 2^d - 1 + k - 1 \]
Heapsort
- Successively insert $n$ items
- Successively delete $n$ items

\[ \sum_{i=1}^{n} (\log i + 1) \geq \sum_{i=n/2}^{n} \log i \]

\[
\geq \frac{n}{2} \log \frac{n}{2}
\]
\[ c \sum_{k=1}^{h} 2^{h-h'} \leq 2^h \sum_{k=1}^{h} 2^{h-h'} \leq 2^h \leq 2^{h+1} = \Theta(n) \]
04. 09. 2018

Post minor rework

1. How do computer scientists compare \( f, g : \mathbb{N} \to \mathbb{R}^+ \)?

A. Three Options

i) \( f(n) \in \Theta(g(n)) \) — same (why?)

ii) \( f(n) \in o(g(n)) \) — \( f(n) \) is slower/faster than \( g(n) \) (why?)

iii) \( g(n) \in o(f(n)) \) — Conceptually the same.
2. \( T' = \text{parent}(\text{root}(T)) \)

\[
\text{int Tree} := \text{int Tree Node} \mid \text{Empty Tree}
\]

\[
\text{int Linked List} := \text{int Node} \mid \text{Empty List}
\]
Dictionary ADT: A - Dictionary set isNotAdj

Operations: Given \( x \in X \), \( D \in X \cdot A \cdot \text{dictionary} \)
- Insert \( (x, D) \)
- Delete \( (x, D) \)
- Find \( (x, D) \)

\( X \cdot A \leftarrow \text{unordered} \) \[ \text{isEqual} \quad \text{exists} \]
\( X \cdot A \leftarrow \text{ordered} \) \[ \leq \quad \text{contains} \]
\( \text{and in a Total order} \)
Implementation 1: Log files

Each \( i \in A \) has a key \( k \) and data \( d \) associated with it. The \( k \) comes from an ADT that has isEq \( = \) defined.

\[(k_1, d_1), (k_2, d_2), \ldots, (k_n, d_n)\]
Implementation 2: Hash maps.

Dictionary: BucketArray

Storage: "Bucket" array

\[ f: \mathbb{N} \rightarrow \{0, \ldots, n-1\} \]

Given \( a \in \mathbb{N} \):

- \( f(a) \) is the index of the bucket used to store \( a \)

Insert \((a, D) = \)

\[ \text{Bucket} - \text{Insert}(C[f(a)], x) \]

If Bucket is implemented with lists, then A-list insert \((C[f(a)], x)\)
\textbf{Find} \ (x, D) : A \text{-} \text{List} \ - \ A \text{-} \text{Member} \ (C \left(f(x)\right), x)

\text{Ex} \quad X_A = \text{named in Roman script} \\
\begin{array}{c}
\begin{array}{c}
\text{0} \\
\text{C} \\
\text{25}
\end{array}
\end{array}

f(x) = \text{letter position A first letter of x.}

f(\text{"ANKIT"}) = 0 \quad f(\text{"ROHIT"}) = 18

\text{insert (D, "ANKIT")}
\text{insert (D, "ROHIT")}
\text{insert (D, "ANKIT")}

\text{insert (D, "ROHIT")}
hash function

Computing a hash code \( h : X \rightarrow S \) takes at least \( \log S \) bit operations (why?)
Hash codes:

- Casting to an integer: \((a^x) \mod n\) where \(a\) is of some type A.

- Summing components: \(a = (a_1, a_2, \ldots, a_n)\)
  \[ h(a) = \sum a_i \]

- Polynomial: \(a = (a_1, a_2)\)
  \[ h(a) = \sum a_i a^i \]

Compression maps:

- \(h(a) \mod n\)
- \((a \cdot h(a) + b) \mod n\)
If \( x, y \) have the property that \( f(x) = f(y) \) we say that a "collision" has occurred.

Collision handling

1. Checking: If \( f(x) = f(y) \)

2. Load factor: Then we make

3. Given set size = \( M \) a linked list/cast

Array size = \( n \) /hash map

\( \lfloor \frac{M}{n} \rfloor \) \( \text{the load factor} \) and store \( x \rightarrow C[i] \).
Load factor is a lower bound on worst case query time for all dictionary operations.

Open addressing

1. Linear probing

\[ a : f(a) = 2 \]
\[ b : f(b) = 4 \]
\[ j = f(j) = 2 \]
\[ z = f(z) = 2 \]
\[ w = f(w) = 2 \]
\[ x = f(x) = 3 \]

Find \( x \):

- Go to \( f(w) \) and move
- Till you find \( x \) or empty

No

Yes

[Diagram: A table with entries 0, 2, 3, and 7, with arrows indicating the probing sequence.]
1. Insertion: Go to \( C(f(n)) \) and insert if it is empty or less than \( x \), else go right till you find empty / \( x \) or return to \( f(n) \) (Exception)

2. Delete \( x \): Find \( x \) and replace with * if found.

Variation (A Quadratic probing)

\[ f(n) = 0, f(n) + 1, f(n) + 2 \ldots \]

(b) Double hashing

PS (ii.9)

In case the load factor gets very high, we may need to reduce (more uniformly)
Ordered dictionaries

1. Look-up table
   - Maintain keys in sorted order: \( k \), \( k \), \( k \), \( k \)...

   **Binary search**: Find \((k, x)\) in \( A[\log_2 n] \) and \( A[2 \log_2 n] \)
   - \( O(\log_2 n) \)
   - \( O(\log_2 n) \)

   \[ n, \frac{n}{2}, \frac{n}{4}, \ldots, \frac{n}{2^i} \leq 1 \]

   For any search target: \((k, x)\) \( \log_2 n = \frac{k \cdot 2^i}{\log_2 n} \)
\[
\frac{\log_2 n}{\log_2 k} \leq \frac{\log_2 n}{\log_2 k} - 1
\]

\[
\frac{\log k}{\log 2}
\]

Find (8, A)

\rightarrow \text{Not Found (7, 12)}

\begin{align*}
\text{a}^- &= \text{greatest key on which } i \leq \text{a}^- \in A \\
\text{a}^+ &= \text{least key } x \text{ which is } x \in A
\end{align*}

\text{P.S.: Maintaining the sorted array is } \Theta(n) \text{ in order to insert and delete.}
Can we do binary search with linked lists (or some modification of linked lists) so that insert/delete becomes easier?

\[ \text{Find}(D, 4), \text{Find}(D, 10) \]
14.09.2018

How to insert in a skip-list.

Insert $(s, x)$. 

1. Decide $\text{ht}(x) > 0$.
2. Insert $x$ with height $\text{ht}(x)$.

Where $\text{ht}(x) = \max \{ k : x \in E_k \}$. 
Insert as follows:
- Find \((s, a)\) "with a stack"
- \(i = 0\)
- Until Stack empty and \(i \leq \text{ht}(n)\)
  - Pop (Stack) and get \((k, x_k, x_{k+1})\)
  - Insert \(a\) between \(x_k, x_{k+1}\)
    in \(L_k\)
  - \(i++\)
- While \(i \leq \text{ht}(n)\)
  - Make a new list \(L_i\) and insert \(a\) in it
  - \(i++\)
How to choose \( h_t(x) \): Randomly.

Pick \( p \in (0,1) \)
- \( x \) is in \( l_0 \) with probability 1
- For all \( i > 0 \)
  - if \( x \notin l_i \), then put \( x \) in \( l_i \) w/prob \( p \).

Height analysis: Given \( n \) elements \( a_1, \ldots, a_n \)

\[
P \left( \bigcup_{i=1}^{n} h_t(a_i) \geq k \right) = p^k \sum_{k=2^{\log p}}^{n} \binom{n}{k} \leq \frac{n^k}{k!} \leq \frac{e^k}{k!} 
\]

Given event \( A \cup B \)
\[
P(A \cup B) = P(A) + P(B) - P(A \cap B) 
\]

\[
k = 2 \log n \approx \log_{10} n + o(n) 
\]
Distance travelled backwards before encountering first key promoted to level 1:

\[ k \sim \text{Geometric}(p) \]

\[ E[\text{# of items seen in list } 0] = \frac{1}{p} \]

\[ E[\text{# of items seen in list } i] = \frac{1}{p^i} \]
$X$: No. of items in level $i$ met before first at $m/2$ level ≥ 1

$P[X = k] = (1-p)^{k-1}p$

$E[X] = \sum_{k=1}^{\infty} k (1-p)^{k-1} = 1 \cdot \frac{1}{\frac{1}{1-(1-p)^2}} = \frac{1}{p}$

Time taken to find in expectation = $\frac{1}{b} \log(s)$

$= \Theta\left(\frac{1}{b} \log(1/p)\right)$
Binary Search Trees (BST)

A BinTree where
A is a totally ordered datatype with comparison operator $\leq$.

Def: An A-BinTree $t$ at every node $q$, the key $v$ in
the subtree of $q$ are $\leq$ the key at $q$ and the keys in
the $\geq$ subtree of $q$ are $> v$.
Find $\alpha$(19)

- If $\alpha = \text{Empty}$ say No
  - If $\alpha = \text{key(root(T) = } \alpha$
    say Yes
  - Else if $\alpha < \text{key(root(T) = }$
    find($\alpha$, \text{left-sub(T(root(T))})
    Else
    find($\alpha$, \text{right-sub(T(root(T))})
Find (31)

30^+ = 32
30^- = 30

Define: in order successor
in order predecessor
of each node
Case 1: Delete leaf

Case 2: Delete node with 1 child
   (attach subtree to parent of deleted node)

Case 3: Pop up inorder predecessor and then delete it's node (Case 1 or 2)
AVL Trees (Adelson-Velckii, Landis, 1957)

\[ |h_l - h_r| \leq 1 \]

A binary search tree is called an AVL tree if it is a BST and the difference in heights of subtrees is at most one at every node.

"structural invariant"

Find operation:
- Exactly like BST
- (no structural invariant is broken so nothing extra to do)
Height analysis:

\[ n(h) : \text{no of nodes in AVL tree of height } h \]

Convention:

- \( h = \min \) 

\[ \bar{n}(h) = \text{max no of nodes of AVL tree of height } h \]

\[ n(h) = \min \]

\[ \bar{n}(h) = \max \]

\[ n(0) = 1 \]

Claim: \( \bar{n}(h) \) increases with \( h \).

Proof: True since \( \bar{n}(h) = 2^{h+1} - 1 \)

- \( n(0) = 2 \)
- \( n(1) = 3 \)
- \( n(2) = 7 \)
- \( n(3) = 15 \)

Claim: \( n(h) \) increases with \( h \)
Claim: $n(h)$ increases with $h$.

Proof:
- Consider a tree of height $h$ that has $n(h)$ nodes.
- Remove all nodes of depth $h$.
- The resulting tree is an AVL tree of height $h-1$.

- The number of nodes:
  $n(h) = n(h-1) + 2n(h-2) + 1$

- By induction, $n(h) = n(h-1) + 2n(h-2) + 1$.

Thus, $n(h) > n(h-1)$.
Claim: \( n(h) = 1 + 2n(h-1) \geq n(h-2) \)

LH: Follows since \( n(h) \) increases with \( h \)

Soln 1: Claim 3 \( \Rightarrow \) \( n(h) \geq 1 + 2n(h-2) \)

\[
\text{RHS} = \sum_{i=0}^{\sqrt{2/h}} 2^i \geq 1 + 2 \left( 1 + 2n(h-1) \right)
\]

\[
\Rightarrow n(h) \geq 2^\sqrt{2/h} - 1 \quad \text{and} \quad 2^\sqrt{2/h} \leq n(h)
\]

\[
\Rightarrow 2^{\frac{h+1}{2}} \leq \log_2 (n(h) + 1) \leq \log_2 n(h)
\]

\[
\Rightarrow h \leq \left( \log_2 (n(h) + 1) - 1 \right) = O(\log_2 n(h))
\]
Given an \( m \): let \( h(n) \) be the set of heights of AVL trees on \( n \) nodes. \( h(n) = \{ h_1, h_2, \ldots, h_k \} \).

\( h_i \in O(\log^2 n) \)

Function:
- Start with simple BST insertion.
- Move up (using parent pointer) updating height as required, stopping either when height doesn't change or when AVL property is violated.
Stage 1: BST insertion
Stage 2: Moving up from inserted node update
- Height check for balance property violation,
- Stop if no further height update poss. violation found

Stage 3: If stop in stage 2 happened
with no invariant violation => Quit.
Since it cannot be \( h-3 \), since violation would be shown at \( C \) before \( A \).

Since it cannot be \( h-1 \), increment.
Restructuring procedure (trinode restructuring)

1. Identify 3 nodes
   - \( z \): 1st node at which imbalance is observed
   - \( y \): higher child of \( z \) in terms of subtree height
   - \( \alpha \): higher child of \( y \) in terms of subtree height

\[ \text{If } \alpha \leq y \text{, make } \alpha \text{ the root} \]
\[ \text{If } \alpha > y \text{, make } y \text{ the root} \]