CMPT307: Binary Search Trees

Week 2-2

Xian Qiu

Simon Fraser University

xianq@sfu.ca

Outine

- open addressing
- trees
- binary search trees

Linear Probing

auxiliary hash function: $h' : U \to \{0, 1, \ldots, m-1\}$

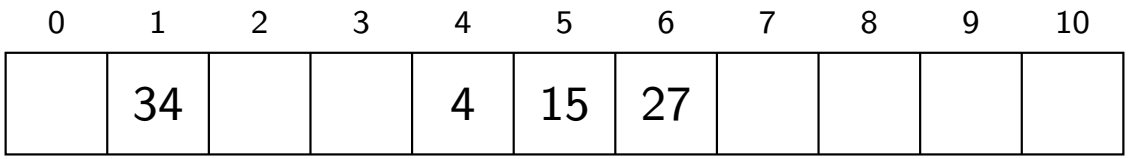
linear probing

$$h(k, i) = (h'(k) + i) \mod m, \text{ for } i = 0, 1, \ldots, m-1$$

- ▷ probe sequence $T[h'(k)], T[h'(k)+1], \ldots, T[m-1]$
- ▷ m distinct probe sequence in total
- ▷ primary clustering: successive filled slots tend to get longer

Example

- hash function $h(k, i) = (k + i) \mod 11$
- to insert $34, 4, 15, 27$



Quadratic/Double Probing

quadratic probing: $h(k,i) = (h'(k) + c_1 i + c_2 i^2) \mod m$

- c_1, c_2, m shall yield distinct $h(k,i)$ values for all i
  e.g. $c_1 = c_2 = 1/2$ and $m = 2^n$
- secondary clustering: $h(k_1, 0) = h(k_2, 0)$ implies $h(k_1, i) = h(k_2, i)$

double hashing: $h(k,i) = (h_1(k) + i \cdot h_2(k)) \mod m$

- the offset also depends on k
- example: choose m prime and let $m' = m - 1$

$$h_1(k) = k \mod m, \quad h_2(k) = 1 + (k \mod m')$$

$\Theta(m^2)$ probe sequences

Average Running Times

assume uniform hashing

Theorem

If $\alpha < 1$, unsuccessful search requires at most $1/(1-\alpha)$ probes on average.

Corollary

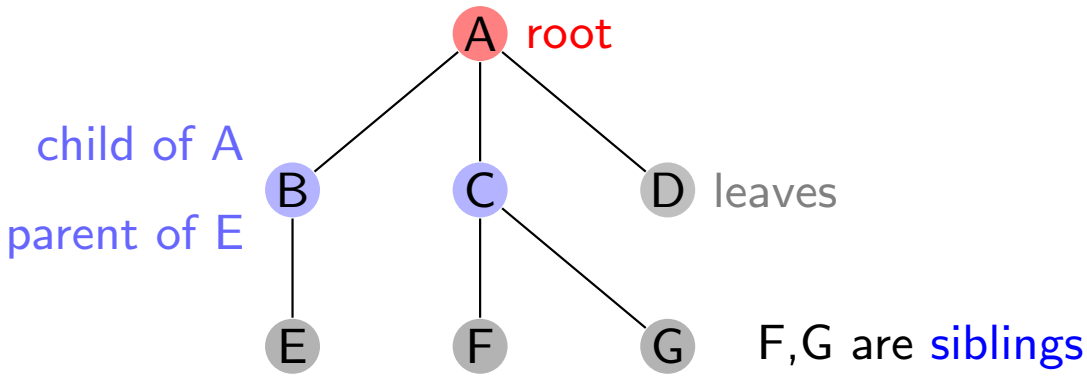
Insertion requires at most $1/(1-\alpha)$ probes on average.

Theorem

If $\alpha < 1$, successful search requires at most $\frac{1}{\alpha}\ln\frac{1}{1-\alpha}$ probes on average.

Trees

- ▷ tree: connected and acyclic graph — denoted by T
- ▷ can specify a root of a tree — denoted by r

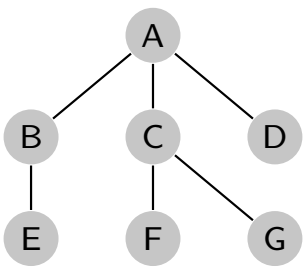


- ▷ a tree of n nodes has $n-1$ edges
  every node has a parent except for the root
- ▷ $\exists$ a unique path between any two nodes of T
- ▷ depth of $v \in T$: the length of (r, v)-path
- ▷ height of $v \in T$: length of longest path from v to all leaves
- ▷ height of T = height of r

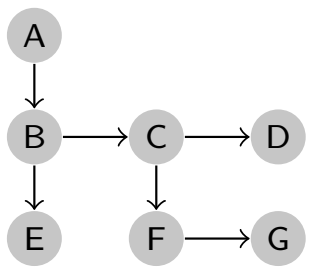
Representation

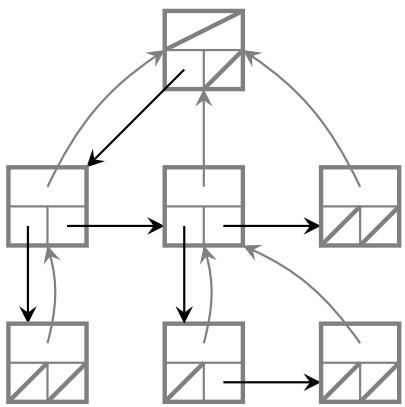
left-child, right-sibling representation (x = tree node)

- x.p $\rightarrow$ parent of x
- x.left-child $\rightarrow$ leftmost child of node x
- x.right-sibling $\rightarrow$ the sibling of x immediately to its right



binary tree



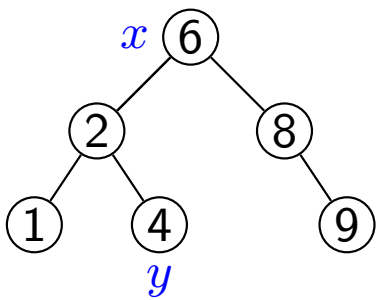


Binary Search Trees

assume each node x is assigned with a key (integer)

Property

if y is a node in the left (right) subtree of x, then y.key $\leq$ ($\geq$) x.key



x.left

x.right

x.p

how to print out keys in sorted order?

Printing Keys

INORDER-TREE-WALK(x)

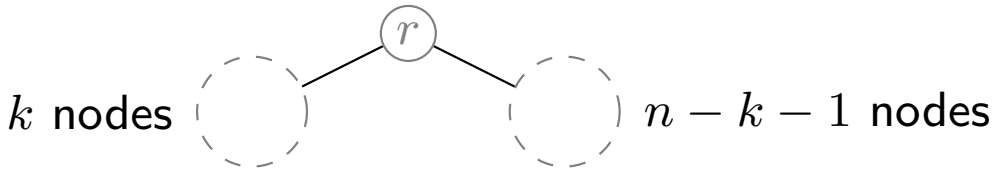
```
1 if x ≠ nil then
2     INORDER-TREE-WALK(x.left);
3     print x.key;
4     INORDER-TREE-WALK(x.right);
```

Theorem. INORDER-TREE-WALK(T.root) runs in $\Theta(n)$.

- $T(n) \in \Omega(n)$, to show $T(n) \in O(n)$ (by induction)

Proof of the Theorem

- to prove $T(n) \leq cn$ for some constant c (*)
- for $n = 1$, let $c \geq T(1)$ and (*) holds
- assume $T(m) \leq cm$, for all $m < n$
- consider n and observe



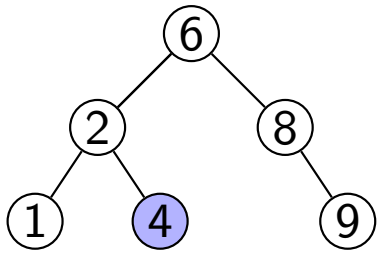
$$\begin{aligned} T(n) &\leq T(k) + T(n-k-1) + \Theta(1) \\ &\leq ck + c(n-k-1) + d \\ &= cn - c + d \end{aligned}$$

- let $c \geq d$ and get $T(n) \leq cn$, proving the claim

Searching

TREE-SEARCH(x,k)

```
1 if x == nil or k == x.key then
2     return x;
3 if k < x.key then
4     return TREE-SEARCH(x.left,k);
5 else
6     return TREE-SEARCH(x.right,k);
```



- running time $O(h)$, where h = tree height
- non-recursive way?
- maximum and minimum