Dynamic Order Statistics

Some of the slides are courtesy of
Charles Leiserson and Carola Wenk

More Data structure ????
Isn't it an Algorithm course ???

If you want to feel poetic
"Data Structures are Algorithms frozen in Time"

Anonymous

By now you are familiar with several data structures that supports the following operations on a dynamic set S

Insert (x, S):	inserts x into S
Delete (x, S):	deletes x from S
Find (x, S):	finds x in S
Succ(x, S):	find smallest element larger than x in S

Popular implementation uses any balanced search tree (not necessarily binary) or Skiplist. Each operation takes $O(\log n)$ time.

Balanced search trees

Balanced search tree: A search-tree data structure for which a height of $O(\log n)$ is guaranteed when implementing a dynamic set of n items.

Examples:

- AVL trees
- 2-3 trees
- 2-3-4 trees
- B-trees
- Red-black trees
- SkipList (only expected time bounds)
- Splay tress (Amortized time)

Dynamic order statistics

Need a DS that supports the following operations in addition to Insert(x,S), Delete(x,S), Find(x), Succ(x,S)

OS-SELECT(i, S): returns the element with rank i in the dynamic set S.

Smallest key has rank 0. Largest has rank $n-1$.

OS-RANK(x, S): returns the rank of $x \in S$ in the sorted order of S' s elements.

(many other problems could be solved in a similar techniques)

First Try: Each key stores its rank. So we only need to find the key (takes O(log n) in most data structures) and retrieve the rank.

So OS-Rank(x, S) takes O(log n)

Dynamic order statistics-cont

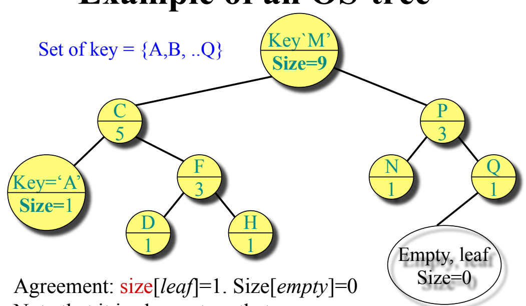
- **Second Try:** (just for the protocol)
- Store all keys in a sorted array.
- The index is the rank.
- So great for a **static** structure, less so for dynamic structure.

Dynamic order statistics-cont

Third Idea: (actually working) Use a balanced binary search tree for storing the set *S*, but each node *v* has an extra field *size[v]* storing the number of keys in the subtree rooted at *v*

Notation for nodes: 

Example of an OS-tree



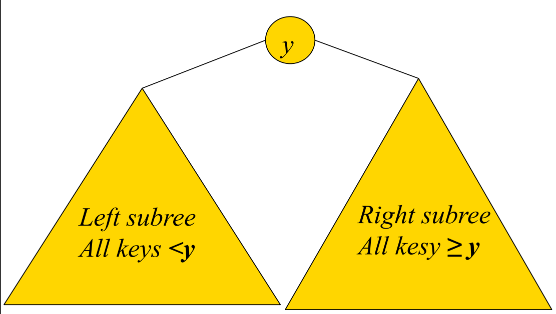
Agreement: size[*leaf*]=1. Size[*empty*]=0

Note that it is always true that

size[*x*] = *size*[*left*[*x*]] + *size*[*right*[*x*]] + 1

We will use it in the algorithm (wait for it)

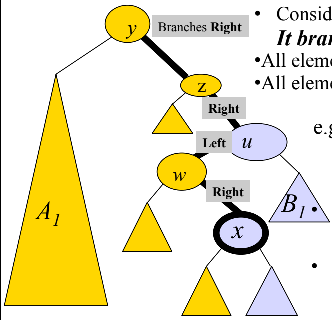
Quick Reminder



How to answer OS-Rank(*x,S*)

Returns the number of keys in the tree which are strictly smaller than *x*.

- Assume we already performed **find(x,S)**.
- Consider the search path from root to ***x***. ***It branches left and right.***
- All elements in yellow subtrees are ≤ ***x***.
- All elements in blue trees are > ***x***.

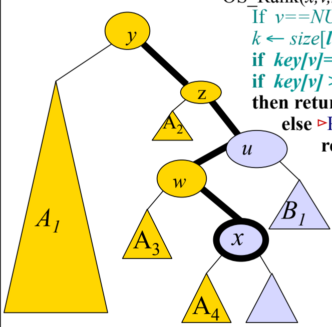


e.g. in A_1, all *keys* < ***y*** <***x***.
in B_1 , all *keys* > ***u***> ***x***.

- Need to sum numbers of all keys in yellow **trees** + # yellow **nodes**.
- They are the left subtrees of every node where the search path **branches right** **plus** # these nodes**.**

How to answer OS-Rank(x,S) (cont)

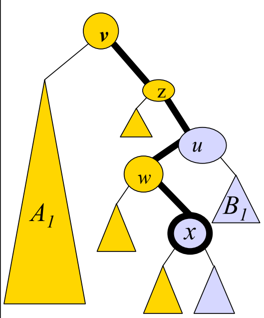
Returns the number of keys in the tree which are strictly smaller than *x*.



OS_Rank(*x,v,S*)
 If *v==NULL* return *0* ▹ empty subtree
 k ← *size*[***left***[*v*]]
 if ***key[v]==x*** **then return** *k.* **else**
 if ***key[v]*** **>*x*** *0* ▹ Path branches **LEFT**
 then return OS_Rank(*x, left*[*v*], *S*)
  else ▹Path branches **RIGHT**
   return
    k+1+OS_Rank(*x, right*[*x*], *S*)

How to answer OS-Select(*i*,S)

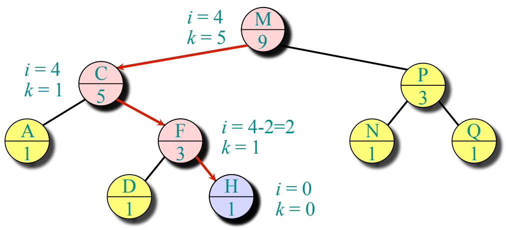
Returns the *i*'th smallest key
(e.g. OS-Select*(0,S)* returns the first. OS-Select*(n-1,S)* return last)



OS-SELECT(*v*, *i*)
▹ return *i* th smallest element in the ▹
 subtree rooted at *v*
k ← *size*[*left*[*v*]] + 1
if *i* == *k* **then return** *v.* **else**
if *i* < *k*
 then return OS-SELECT(*left*[*v*], ***i***)
else
  return
   OS-SELECT(*right*[*v*], ***i* – (*k*+1)**)

Example

OS-SELECT(*root*, 4)



Running time = $O(h) = O(\lg n)$ for BSTs.

Data structure maintenance

Q. Why not keep the ranks themselves in the nodes instead of subtree sizes?

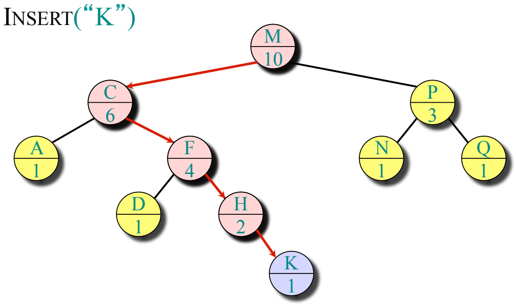
A. They are hard to maintain when the BST is modified.

Modifying operations: INSERT and DELETE.

Strategy: Update subtree sizes when inserting or deleting.

Example of insertion

INSERT("K")

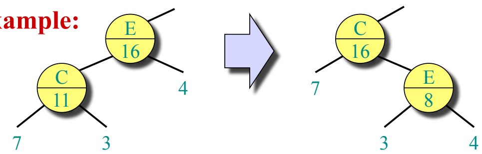


Handling rebalancing

Don't forget that BST-INSERT and BST-DELETE may also need to modify the binary search tree in order to maintain balance.

• *Rotations*: fix up subtree sizes in $O(1)$ time.

Example:



∴ BST-INSERT and BST-DELETE still run in $O(\lg n)$ time.

Data-structure augmentation

Methodology: (*e.g., order-statistics trees*)

1. Choose an underlying data structure (*binary search trees, e.g. AVL or red-black trees*).
2. Determine additional information to be stored in the data structure (*subtree sizes*).
3. Verify that this information can be maintained for modifying operations (*BST-INSERT, BST-DELETE — don't forget rotations*).
4. Develop new dynamic-set operations that use the information (*OS-SELECT and OS-RANK*).

These steps are guidelines, not rigid rules.