A hierarchical combinatorial structure

A hierarchical combinatorial structure:

1. A rooted tree is a directed graph
   - A rooted tree is a directed graph
   - A rooted tree is a directed graph

2. A rooted tree has a root node
   - A rooted tree has a root node
   - A rooted tree has a root node

Ordered tree

- An ordered tree is a tree in which the children of each node are ordered
- An ordered tree is a tree in which the children of each node are ordered
- An ordered tree is a tree in which the children of each node are ordered

Example: description of a book

- A book is a hierarchical structure
- A book is a hierarchical structure
- A book is a hierarchical structure

The dictionary problem

- Maintain (distinct) items with keys from a totally ordered universe
- Maintain (distinct) items with keys from a totally ordered universe
- Maintain (distinct) items with keys from a totally ordered universe

- Subject to the following operations
- Subject to the following operations
- Subject to the following operations
The ADT

- Insert\((x, D)\)
- Delete\((x, D)\)
- Find\((x, D)\): Returns a pointer to \(x\) if \(x \in D\), and a pointer to the successor or predecessor of \(x\) if \(x\) is not in \(D\)

- successor\((x, D)\)
- predecessor\((x, D)\)
- Min\((D)\)
- Max\((D)\)

The ADT

- catenate\((D_1, D_2)\): Assume all items in \(D_1\) are smaller than all items in \(D_2\)
- split\((x, D)\): Separate to \(D_1, D_2\)
  - \(D_1\) with all items greater than \(x\) and
  - \(D_2\) smaller than \(x\)

Reminder from “mavo”

- We have seen solutions using unordered lists and ordered lists.
- Worst case running time \(O(n)\)
- We also defined Binary Search Trees (BST)

Binary search trees

- A representation of a set with keys from a totally ordered universe
- We put each element in a node of a binary tree subject to:
  - If \(y\) is in the left subtree of \(x\) then \(y.key < x.key\)
  - If \(y\) is in the right subtree of \(x\) then \(y.key > x.key\)
BST

```
x.left
x
x.parent
x.key
x.right
```

Find(x, T)

```
Y ← null
z ← T.root
While z ≠ null
do y ← z
  if x = z.key return z
  if x < z.key then z ← z.left
  else z ← z.right
return y
```

Find(5, T)

```
Y ← null
z ← T.root
While z ≠ null
do y ← z
  if x = z.key return z
  if x < z.key then z ← z.left
  else z ← z.right
return y
```

Find(5, T)

```
Y ← null
z ← T.root
While z ≠ null
do y ← z
  if x = z.key return z
  if x < z.key then z ← z.left
  else z ← z.right
return y
```

Find(5, T)

```
Y ← null
z ← T.root
While z ≠ null
do y ← z
  if x = z.key return z
  if x < z.key then z ← z.left
  else z ← z.right
return y
```
Find(5, T)

```
Y ← null
z ← T.root
While z ≠ null
  do y ← z
    if x = z.key return z
    if x < z.key then z ← z.left
    else z ← z.right
return y
```

Find(6, T)

```
Y ← null
z ← T.root
While z ≠ null
  do y ← z
    if x = z.key return z
    if x < z.key then z ← z.left
    else z ← z.right
return y
```

Min(T)

```
Min(T.root)

min(z):
While (z.left ≠ null)
  do z ← z.left
return z
```

Insert(x, T)

```
n ← new node
n.key ← x
n.left ← n.right ← null
y ← find(x, T)
n.parent ← y
if x < y.key then y.left ← n
else y.right ← n
```
\[
\text{Insert}(6,T) \\
\begin{align*}
n &\leftarrow \text{new node} \\
n\text{.key} &\leftarrow x \\
n\text{.right} &\leftarrow \text{null} \\
y &\leftarrow \text{find}(x,T) \\
n\text{.parent} &\leftarrow y \\
\text{if } x < y\text{.key then } y\text{.left} &\leftarrow n \\
\text{else } y\text{.right} &\leftarrow n
\end{align*}
\]

\[
\text{Delete}(6,T) \\
\begin{align*}
n &\leftarrow \text{new node} \\
n\text{.key} &\leftarrow x \\
n\text{.left} &\leftarrow n\text{.right} \\
n\text{.right} &\leftarrow \text{null} \\
y &\leftarrow \text{find}(x,T) \\
n\text{.parent} &\leftarrow y \\
\text{if } x < y\text{.key then } y\text{.left} &\leftarrow n \\
\text{else } y\text{.right} &\leftarrow n
\end{align*}
\]

\[
\text{Delete}(8,T) \\
\begin{align*}
n &\leftarrow \text{new node} \\
n\text{.key} &\leftarrow x \\
n\text{.left} &\leftarrow n\text{.right} \\
n\text{.right} &\leftarrow \text{null} \\
y &\leftarrow \text{find}(x,T) \\
n\text{.parent} &\leftarrow y \\
\text{if } x < y\text{.key then } y\text{.left} &\leftarrow n \\
\text{else } y\text{.right} &\leftarrow n
\end{align*}
\]

\[
\text{Delete}(8,T) \\
\begin{align*}
n &\leftarrow \text{new node} \\
n\text{.key} &\leftarrow x \\
n\text{.left} &\leftarrow n\text{.right} \\
n\text{.right} &\leftarrow \text{null} \\
y &\leftarrow \text{find}(x,T) \\
n\text{.parent} &\leftarrow y \\
\text{if } x < y\text{.key then } y\text{.left} &\leftarrow n \\
\text{else } y\text{.right} &\leftarrow n
\end{align*}
\]
Delete(2, T)

Switch 5 and 2 and delete the node containing 5

Delete(2, T)

Switch 5 and 2 and delete the node containing 5

delete(x, T)

q ← find(x, T)
If q.left = null or q.right = null then z ← q
else z ← min(q.right)
q.key ← z.key

delete(x, T)

q ← find(x, T)
If q.left = null or q.right = null then z ← q
else z ← min(q.right)
q.key ← z.key

delete(x, T)

q ← find(x, T)
If q.left = null or q.right = null then z ← q
else z ← min(q.right)
q.key ← z.key

delete(x, T)

q ← find(x, T)
If q.left = null or q.right = null then z ← q
else z ← min(q.right)
q.key ← z.key
\begin{enumerate}
\item \texttt{delete(x,T)}
\begin{itemize}
\item \texttt{q ← find(x,T)}
\item If \texttt{q.left = null or q.right = null} then \texttt{z ← q}
\item else \texttt{z ← min(q.right)}
\item \texttt{q.key ← z.key}
\item If \texttt{z.left = null then} \texttt{y ← z.left}
\item else \texttt{y ← z.right}
\end{itemize}
\end{enumerate}
Variation: Items only at the leaves

- Keep elements only at the leaves
- Each internal node contains a number to direct the search

Implementation is simpler (e.g. delete)
Costs space

Balance

\[ h = O(\log n) \]

How do we keep the tree balanced through insertions and deletions?

Analysis

- Each operation takes \( O(h) \) time, where \( h \) is the height of the tree
- In general \( h \) may be as large as \( n \)
- Want to keep the tree with small \( h \)

Applications of search trees

1) Order statistics

- rank and select

Select\((i,D)\)

- \( \text{Select}(i,D) \): Returns the \( i \)th element in our predefined set:

An element \( x \) such that \( i-1 \) elements are smaller than \( x \)
We can use binary trees for this

For each node $v$ store # of leaves in the subtree of $v$
Select(7, T)

Select(3, )

Select(1, )

O(\log n) worst case time for balanced trees

Rank(x, T)

• Return the index of x in T

Sum up the sizes of the subtrees to the left of the path

Need to return 9