Mini Homework:
CLRS Exercise 4.1-5

(Read and apply the clever insight I forgot at the end of last class. Figure out yourself the Good O(N) algorithm for the Max Sum of subarray problem.)
objects storing state, methods having local variables and code referring to objects. Data structures (implicitly with algorithms on them) 333+402, etc. Applications of some data structures.

Data structures are associated with the best algorithms for each, some operations change the set, others don’t. Abstract operations on Dynamic Sets:

- `Search(S, k)` - `k` is a key
- `Insert(S, x)` - `x` is an element, a pointer to `x` is a key
- `Delete(S, x)`
- `Minimum(S)`
- `Maximum(S)`
- `Successor(S, x)`
- `Predecessor(S, x)`

Cartesian product "combine in all possible ways".
Problem: Find the best algorithm for each operation on the given data str choice, and figure out its worst case running time order. 

\[ N = \text{# elements in the set } S \]

Unsorted, singly linked list

- Search \((S, k)\) \(\Theta(N)\)
- Insert \((S, x)\) \(\Theta(1)\)
- Delete \((S, x)\) \(\Theta(N)\)
- Minimum \((S)\) \(\Theta(N)\)
- Maximum \((S)\) \(\Theta(N)\)
- Successor \((S, x)\) \(\Theta(N)\)
- Predecessor \((S, x)\) \(\Theta(N)\)
Sorted, singly linked list

- `Search(S, k)` $\Theta(N)$ (or non-decreasing)
- `Insert(S, x)` $\Theta(N)$
- `Delete(S, x)` $\Theta(N)$
- `Minimum(S)` $\Theta(1)$
- `Maximum(S)` $\Theta(1)$ if no max or tail pointer!
- `Successor(S, x)` $\Theta(1)$
- `Predecessor(S, x)` $\Theta(N)$

```
 x -> [data] -> [3, 9, 5, 8, 100] ...
```

Diagram: Linked list with a node marked as the predecessor of another node. The diagram shows a chain of nodes with arrows indicating the direction of the links. The node marked as the predecessor is connected to the node it points to, and the diagram also includes a note about knowing something.
sorted, doubly linked list

**Search** \((S, k)\)

**Insert** \((S, x)\)

**Delete** \((S, x)\)

**Minimum** \((S)\)

**Maximum** \((S)\)

**Successor** \((S, x)\)

**Predecessor** \((S, x)\)

\[
\begin{align*}
\Theta(1) & \quad \text{for sorted, singly linked lists} \\
\Theta(n) & \quad \text{for sorted, doubly linked lists}
\end{align*}
\]
len Sorted, doubly linked list

\[
\text{Search}(S, k) \\
\text{Insert}(S, x) \quad \Theta(1) \\
\text{Delete}(S, x) \quad \Theta(1) \quad \text{not } \Theta(N) \\
\text{Minimum}(S) \\
\text{Maximum}(S) \\
\text{Successor}(S, x) \\
\text{Predecessor}(S, x)
\]
Stacks

Queue

Store a dynamic set in an order determined by the Insert(S, x) and a restricted Delete(S).

Insert(S, x) \rightarrow \text{at a particular spot in a stack.}

Delete(S) \text{ does not say which}

Stack LIFO

Queue FIFO

Useful for

Controlling computations
1. Stack: controls calling and returning of methods/funs calling one another, recursive or not.
2. Stack: controls evaluating nested expressions.
3. \text{... search of a natural; Depth First}
4. Queue: controls fairly access to a queue.
5. \text{... controls discrete event simulation process.}

Components in cool algorithms for cool problems
(Parsing and rendering xhtml, etc)
(Rooted) Tree data structures

Binary Tree

Unbounded Branching

A node with 2 node pointers suffices for both kinds!
Shape of tree (binary or unbounded) \n
How balanced \n
Constraints on ordering of the keys (none or Min-heap or Max-heap or Search ordered) \n
Binary Balanced Search tree \n
Unbounded Unbalanced Max-heap \n
$2 \times 2 \times 4 = 16$
Hash tables

key $k$ $\xrightarrow{h}$ number to be used as an array index

Universe of keys $\equiv$ direct address table $\{0,1,...,m-1\} = M$

$\Rightarrow \Rightarrow \Rightarrow$ hash table

$h$ should be easy to compute

$h(k)$ locates in memory where to find $k$ or where to begin looking

Clearly $k_1, h(k_1) = h(k_1)$

$\overbrace{\text{yesterday}}$ $\overbrace{\text{today}}$

But with $k_1 \neq k_2$ $h(k_1) = h(k_2)$

a collision

is possible (sad but true!)
Search(S, k)
Insert(S, x)
Delete(S, x)
Minimum(S)
Maximum(S)
Successor(S, x)
Predecessor(S, x)
Example for a family:

Those trees that might actually appear in memory are the ones with a height $H$ equal to $\max \# of parent-to-child steps to go from the root to a leaf.

A family of trees is balanced when there is some constant $C$ so for every tree $T$ in the family:

$$\text{Height}(T) \leq C \log_2(\# \text{nodes}(T))$$
ordered binary tree

all keys in the left subtree (N) < k

all keys in the right subtree of k > k
Non-Balanced Family example: All the Binary search trees that come from plain simple inserting of sequences of distinct key:

Case: deleting search
When Family is Balanced.
Search (S, h) time for it is O (log₂N)
Search \((S,k)\)
Insert \((S,x)\)
Delete \((S,x)\)
Minimum \((S)\)
Maximum \((S)\)
Successor \((S,x)\)
Predecessor \((S,x)\)
Search($S, k$)
Insert($S, x$)
Delete($S, x$)
Minimum($S$)
Maximum($S$)
Successor($S, x$)
Predecessor($S, x$)
Search \((S, k)\)
Insert \((S, x)\)
Delete \((S, x)\)
Minimum \((S)\)
Maximum \((S)\)
Successor \((S, x)\)
Predecessor \((S, x)\)
Hash tables

key $k$ \rightarrow \text{Hash Function } h \rightarrow \text{number to be used as an array index}

Universe of keys \rightarrow \text{direct address table} \rightarrow \{0, 1, \ldots, m-1\} = m

range of $h$ = range of index values

$h$ should be easy to compute

$h(k)$ locates in memory where to find $k$
or where to begin looking

Clearly $k_1 \neq h(k_1) = h(k_1)$

today

tyesterday

But with $k_1 \neq k_2$, $h(k_1) = h(k_2)$
a collision is possible (sad but true!)
Hash table with chains to resolve collisions

\[ k \rightarrow h(k) \in \{0, \ldots, m-1\} \]

\[ i = h(k_1) = h(k_2) = h(k_3) \]

Use to help find \( k \)

**MUST**

Compute \( k' = ? \) \( k \)

**MUST**

Give the actual key
General Idea

1. Open

2. Addressing in situ with $k_2 \neq k_1$

3. Calculate another order!

$k \rightarrow$ Calculate

Address
look up

look for a new key

so insert $k_{10}$ into it
The same path of addresses must be computed for the same key EVERY TIME!
look for +
maybe insert a new key
so insert k10 into it
Good idea

$F_2$ is in the DB

$F_2$ (3) Deleted

Deleted
$k_1, h(k_1) = i$

$h(k_2) = i$

$h(k_3) = \text{else}$

Bad idea:

When you need a new address, use $(\text{old address} + 1)$
B-tree $t$

min degree

$t = 2$

Every node (except for the root) has $t$ or more children and 2t or less children

A B C E G I J K L M

Diagram of a B-tree with levels and nodes labeled.
All Leaves are set at the same level!