Last time:
\[ m_0 = 0 \]
\[ m_1 = 1 \]
\[ h > 1 \colon m_h = m_{h-2} + m_{h-1} + 1 \]

Fibonacci sequence:
\[ F_0 = 0 \]
\[ F_1 = 1 \]
\[ F_n = F_{n-2} + F_{n-1} \quad (n > 1) \]

Fact: For all \( h \geq -1 \),
\[ m_h = F_{h+3} - 1. \]

Proof: Let \( f(h) = F_{h+3} - 1 \).
\[ f(-1) = F_2 - 1 = 1 - 1 = 0 \]
\[ f(0) = F_3 - 1 = 2 - 1 = 1 \]

\( h > 0 \colon \)
\[ f(h) = F_{h+3} - 1 \]
\[ = F_{h+1} + F_{h+2} - 1 \]
\[ = (F_{h+1} - 1) + (F_{h+2} - 1) + 1 \]
\[ = f(h-2) + f(h-1) + 1 \]

\( f \) satisfies same recurrence as \( m_h \) and same base case values. So \( m_h = f(h) = F_{h+3} - 1 \)
Ex: \( F_h = \Theta(\phi^h) \)

where \( \phi = \text{golden ratio} \)

\[
\phi = \frac{1 + \sqrt{5}}{2}
\]

\( = 1.6 \ldots \)

So,

\[
m_h = F_{h+3} - 1
\]

\[
= \Theta(\phi^{h+3})
\]

\[
= \Theta(\phi^3 \phi^h)
\]

\[
m_h = \Theta(\phi^h)
\]

\[
\log_\phi m_h = h \pm O(1)
\]

\[
= \Theta(h)
\]

So, \( h = \Theta(\log_\phi m_h) \)

\[
= \Theta(\log m_h)
\]

\[
= O(\log n)
\]

for any AVL tree of size \( n \), height \( h \).

Augmented Structures

Ex: dynamic order stats.
Data Structure to support:
1. Insert (x)
2. Delete (x)
3. Find i’th smallest element.

All ops can take O(lgn) for the following implementation.

Start with a balanced BST (e.g., AVL or RB tree). Insert, delete take O(lgn) worst case time. Maintain extra info at each node: at node x, keep $\text{size}[x] = \text{#nodes in the subtree rooted at } x$.

Selection (T, i)
i’th smallest element:

\[ \begin{align*}
\text{Check that info can be updated quickly.}
\end{align*} \]
recompute \text{size}[x] \text{ and } \text{size}[y]\text{ from roots of } \ A, B, C.

\underline{Dynamic Programming}

\underline{Greedy Algorithms}

Sometimes, recursive solution is not efficient:

\[
Fib(n) = \\
\text{if } n = 0 \text{ then } 0 \\
\text{else if } n = 1 \text{ then } 1 \\
\text{else } \\
\frac{Fib(n-1) + Fib(n-2)}{Fib(5)} \\
\]

\[
Fib(5) = \\
4 = \\
3 + 2 = \\
2 + 1 = \\
1 + 0
\]
Fib \( (n) \) once
9 once
8 twice
7 3 times
6 5 "
5 8 "
4 13 "
3 21 

# calls to Fib \( (n-i) \)
= \( F_{i+1} \)

Time is exponential in \( n \)

Faster way:
- small number of intermediate results
- only compute an intermediate result once.

work from bottom up:

\[ A = \begin{array}{cccccc}
0 & 1 & 1 & 2 & 3 & 5 \\
\end{array} \]

\[ A[0] = 0 \quad \text{ \( \Theta(n) \)} \]
\[ A[1] = 1 \]

For \( i := 2 \) to \( n \) do
[ 

// \( A[i] = Fib(i) \)
moving window:
\[ x := 0 \]
\[ y := 1 \]

\[ \text{for } i := 2 \text{ to } n \text{ do} \]
\[ \rightarrow \text{temp} := x + y \]
\[ x := y \]
\[ y := \text{temp} \]

\[ \text{// } (x, y) = (Fib(i-1), Fib(i)) \]
\[ \text{return } y. \]

**Knapsack Problem (easy version):**

Given integer \( t \geq 0 \) and list of integers \( s_1, s_2, \ldots, s_k \geq 0 \) do there exist \( a_1, a_2, \ldots, a_k \in \{0, 1\} \) such that \( \sum_{i=1}^{k} a_i s_i = t \)?

**Recursive solution (uses backtracking):**

\[ t = 0 \text{ then yes} \]
\[ t > 0 \]
\( t > 0 \):
For \( i = 1 \) to \( k \) do
\( t' = t - s_i \):
if recursive call to
\( \text{Knapsack}(t', s_1, s_2, \ldots, s_{i-1}, s_{i+1}, \ldots, s_k) \)
answer "yes" then
answer yes and quit
end for
answer "no".

Time is exponential in \( t \)

\[
\begin{array}{c}
\text{Dynamic Prog solution} \\
\text{in time } O(tk): \\
\end{array}
\]

array of Boolean values
// \( A[i] = \text{true} \) iff can fill a knapsack of size \( i \)
from \( \langle s_1, \ldots, s_k \rangle \).

For the \( j \)th pass, \( A[i] = \text{true} \)
iff can fill knapsack of size \( i \) with
\( \langle s_1, \ldots, s_j \rangle \).
\[\begin{aligned}
&\text{For } i := 0 \text{ to } t \text{ do} \\
&\quad A[i] = \text{false} \\
&\text{end for} \\
&\quad A[0] = \text{true} \\
&\quad \ldots \\
&\quad \ldots
\end{aligned}\]