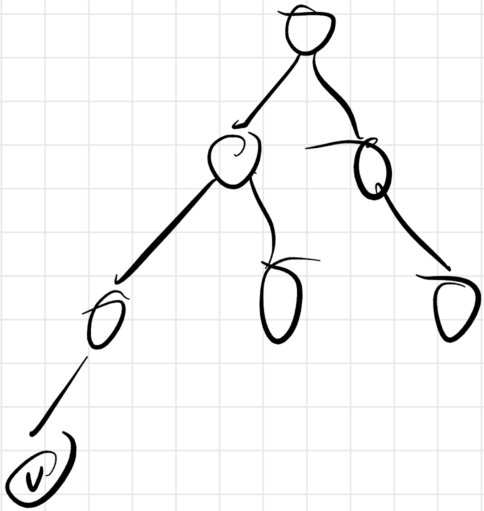


If we do many many searches, how to minimize total search time?

If node x is a more frequent search than y, we want x to have the lower depth.

If some nodes are searched for much more frequently, best tree could have depth $\Omega(n)$.

Optimal Binary Search Tree

Given: 1) A sorted array $A[1..n]$ of keys for the nodes.

2) An array $f[1..n]$ of access frequencies.

We search for $A[i]$ a total of $f[i]$ times.

Goal: Build best static BST to minimize total search time.

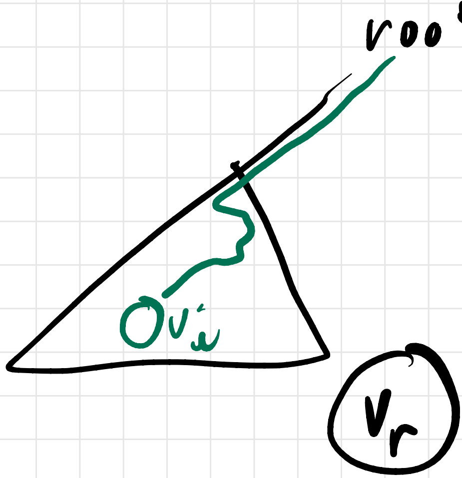
For a given BST T, let $v_1, v_2, \ldots, v_n$ be its nodes in sorted order so v_i stores $A[i]$.

$$\text{Cost}(T, f[1..n]) := \sum_{i=1}^{n} f[i] \cdot \left(\#\text{ancestors of } v_i \text{ in } T\right)$$

(node's depth + 1)

(root has one ancestor: itself)

binary search tree:



right subtree

v_r: root node

If $i < r$, all ancestors of v_i except v_r are in left subtree.

$$\mathrm{Cost}(T, f[1..n]) = \sum_{i=1}^{n} f[i] \cdot 1$$

$$+ \sum_{i=1}^{r-1} f[i] \cdot \left(\begin{array}{c}\#\text{ ancestors in} \\ \text{left}(T)\end{array}\right)$$

$$+ \sum_{i=r+1}^{n} f[i] \cdot \left(\begin{array}{c}\#\text{ ancestors in} \\ \text{right}(T)\end{array}\right)$$

$$Cost(T, f[1..n]) = \sum_{i=1}^{n} f[i] + Cost(left(T), f[1..r-1]) + Cost(right(T), f[r+1..n])$$

$$Cost(T, f[1..0]) = 0$$

OptCost(i,k): optimal cost for BST over $A[i..k]$.

$$\text{OptCost}(i,k) = \begin{cases} 0 & \text{if } i > k \\ \displaystyle\sum_{j=i}^{k} f[j] + \min_{i \le r \le k} \left\{ \begin{array}{l} \text{OptCost}(i, r-1) \\ + \text{OptCost}(r+1, k) \end{array} \right\} & \text{o.w.} \end{cases}$$

Final goal: Compute OptCost$(1, n)$

$$F(i,k) := \sum_{j=i}^{k} f[j].$$

$$F(i,k) = \begin{cases} 0 & \text{if } i > k \\ F(i,k-1) + f[k] & \text{o.w.} \end{cases}$$

Goal: Fill array $F[1..n, 1..n]$.

$O(n^2)$ subproblems in $O(1)$ time each.

```
InitF(f[1..n]):
  for i ← 1 to n
      F[i, i−1] ← 0
      for k ← i to n
          F[i, k] ← F[i, k−1] + f[k]
```

$$OptCost(i,k) = \begin{cases} 0 & \text{if } i > k \\ F[i,k] + \min\limits_{i \le r \le k} \left\{ \begin{array}{l} OptCost(i, r-1) \\ \quad + OptCost(r+1, k) \end{array} \right\} & \text{otherwise} \end{cases}$$

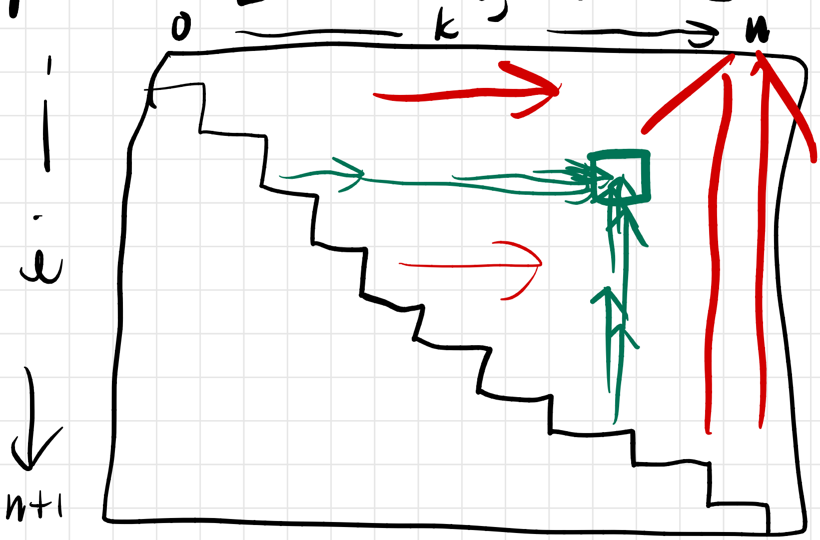
Memoization:

subproblems: $\underline{1} \le i \le \underline{n+1}$

$0 \le k \le n$

memo data structure:

OptCost[1.. n+1, 0...n]



```
ComputeOptCost(i, k):
    OptCost[i, k] ← ∞
    for r ← i to k
        tmp ← OptCost[i, r − 1] + OptCost[r + 1, k]
        if OptCost[i, k] > tmp
            OptCost[i, k] ← tmp
    OptCost[i, k] ← OptCost[i, k] + F[i, k]
```

```
OptimalBST2(f[1..n]):
    InitF(f[1..n])
    for i ← n + 1 downto 1
        OptCost[i, i − 1] ← 0
        for j ← i to n
            ComputeOptCost(i, j)
    return OptCost[1, n]
```

$O(n^3)$ time

$O(n^2)$ subproblems

$O(n)$ per problem

$O(n^2)$ with more work!

[Erickson D.3]

Independent Set of a graph: set of vertices that have no shared edges

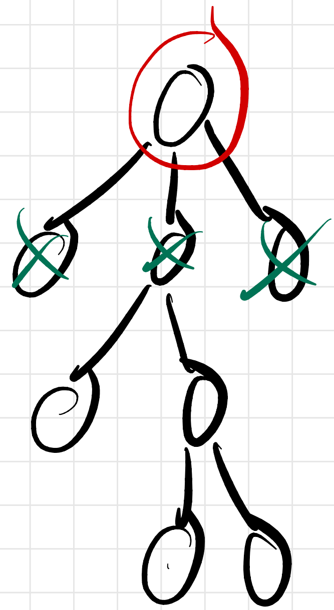
max ind set: given a graph, find max size ind. set

REALLY HARD

(for arbitrary input graphs)

So, say we're given a tree T.

Root it...



Do we include the root?

if no, find max ind sets in each subtree independently

if yes... find sets in <u>grandchild</u> subtrees!

MIS(v): size of MIS in subtree rooted at v

w↓v: "w is a child of v"

a child of v".

$$MIS(v) = \max\left\{\sum_{w\downarrow v} MIS(w),\ 1 + \sum_{w\downarrow v}\sum_{x\downarrow w} MIS(x)\right\}$$

no root

yes root

Next time: how to memoize!