Optimal Binary Search Trees

Given a sorted array $A[1..n]$ of **keys** & an array $f[1..n]$ of **access frequencies**.

Want is to find

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

OptCost(i,k): optimal cost of any BST over $A[i..k]$ using frequencies $f[i..k]$.

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

Original goal: compute

$$\mathrm{OptCost}(1,n)$$

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

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

$O(n^2)$ values in $O(1)$ time each
$\Rightarrow O(n^2)$ time

```
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]
```

$$OptCost(i,k) =$$

$$\begin{cases} 0 & \text{if } i > k \\ F[i,k] + \min\limits_{i \le r \le k} \{OptCost(i, r-1) + OptCost(r+1, k)\} & \text{o.w.} \end{cases}$$

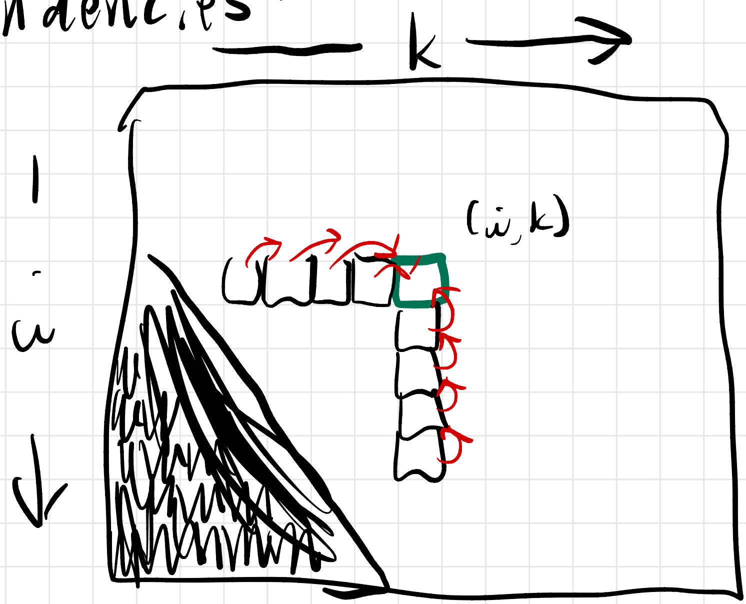
Subproblems: $1 \le i \le n+1$

$0 \le k \le n$

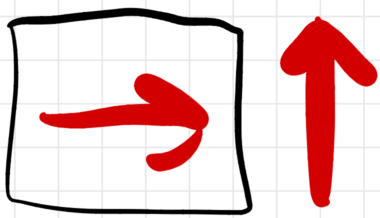
Data structure: 2D array

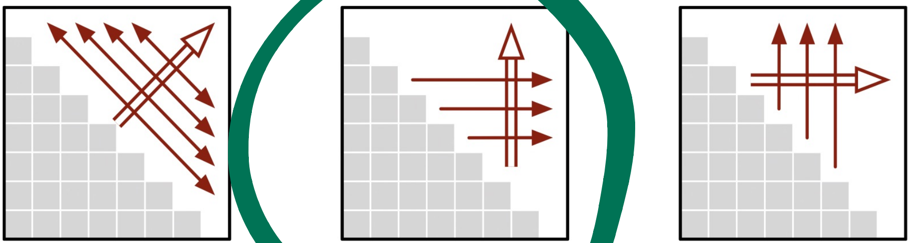
OptCost[1..n+1, 0..n]

Dependencies:



Eval order:





Space: $O(n^2)$

Time: $O(n^2) \cdot O(n) = O(n^3)$

computes & stores $OptCost[i,k]$

<u>COMPUTEOPTCOST(i,k):</u>
 $OptCost[i,k] \leftarrow \infty$
 for $r \leftarrow i$ to k
  $tmp \leftarrow OptCost[i,r-1] + OptCost[r+1,k]$
  if $OptCost[i,k] > tmp$
   $OptCost[i,k] \leftarrow tmp$
 $OptCost[i,k] \leftarrow OptCost[i,k] + F[i,k]$

```
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]
```

```
OptimalBST3(f[1..n]):
    InitF(f[1..n])
    for j ← 0 to n+1
        OptCost[j+1, j] ← 0
        for i ← j downto 1
            ComputeOptCost(i, j)
    return OptCost[1, n]
```

```
OptimalBST(f[1..n]):
    InitF(f[1..n])
    for i ← 1 to n+1
        OptCost[i, i−1] ← 0
    for d ← 0 to n−1
        for i ← 1 to n−d          ⟨⟨...or whatever⟩⟩
            ComputeOptCost(i, i+d)
    return OptCost[1, n]
```

Maximum Independent Set on Trees

Given a graph $G=(V,E)$.
An <u>independent set</u> is a subset of V s.t.

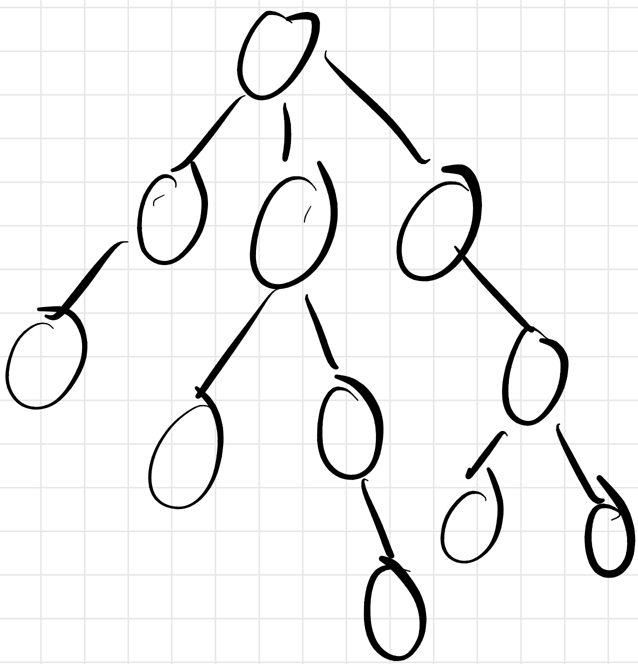
no pair in the subset share an edge.

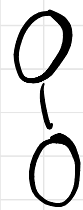
Max Independent Set: find an ind. set of max size.

Really Hard!

But what about on a tree?

Given a rooted tree T on n vertices. Want to know size of max ind. set in T.





T is a root + zero or more subtrees not sharing edges.

If we don't take root, subtrees can do whatever. So recursively find their max indi sets.

If we do take root, we can't take children so recurse on grandchildren.

MIS(v): size of max ind. set in the subtree rooted at v.

children of v

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

(could have written as $\sum_{x \downarrow\downarrow v}$)

Subproblems: vertice v of T

Data structure: Store MIS(v)

as v.MIS.

Dependencies: children & grandchildren

Eval order: in post order.

Space: $O(n)$

Time: Each node appears in the summations ≤ 2 times, so $O(n)$

computes v.MIS & return it

TREEMIS(v):
  $skipv \leftarrow 0$
  for each child w of v
    $skipv \leftarrow skipv + \text{TREEMIS}(w)$
  $keepv \leftarrow 1$
  for each grandchild x of v
    $keepv \leftarrow keepv + x.MIS$
  $v.MIS \leftarrow \max\{keepv, skipv\}$
  return $v.MIS$

$MIS_{yes}(v)$: size max ind set in v's subtree. You <u>must</u> take v.

$MIS_{no}(v)$: Size max ind. set in v's subtree. You <u>must not</u> take v.

Want to return $\max\{MIS_{yes}(r), MIS_{no}(r)\}$ where r is root of T.

$$MIS_{yes}(v) = 1 + \sum_{w \downarrow v} MIS_{no}(w)$$

$$MIS_{no}(v) = \sum_{w \downarrow v} \max\{MIS_{yes}(w), MIS_{no}(w)\}$$

Nearly the same dynamic programming steps including $O(n)$ time

returns "$MIS(v)$"

```
TreeMIS2(v):
    v.MISno ← 0
    v.MISyes ← 1
    for each child w of v
        v.MISno ← v.MISno + TreeMIS2(w)
        v.MISyes ← v.MISyes + w.MISno
    return max{v.MISyes, v.MISno}
```

Subset Sum

Given set X of positive integers & a int. T,

Is there a subset of X that sums to T?

If $X = \{2, 5, 8\}$ & $T = 10$ ✓

If & $T = 11$ ✗

If $T = 0$ Yes, $\sum_{\emptyset} = 0$

If $(T < 0)$ or $(T > 0$ & $X = \emptyset)$ then no!

In general...

Take any $x \in X$. If a subset exists...

- if x not in subset, we want answer for $X \setminus \{x\}$ + T.

- if x in subset, we want answer for $X \setminus \{x\}$ + $T - x$.

Order X arbitrarily as $X[1..n]$.

$SS(i, t)$: True iff a subset

of $X[1..i]$ sums to t.

Want to know $SS(n, T)$.

$$SS(i,t) = \begin{cases} \text{True}, & \text{if } t=0 \\ \text{False}, & \text{if } i=0, t\neq 0 \\ SS(i-1,t) & \text{if } i>0 \text{ + } 0<t<X[i] \\ SS(i-1, t-X[i]) \vee SS(i-1,t) & \end{cases}$$

$0 \le i \le n$, $0 \le t \le T$

So store in $SS[0..n, 0..T]$

Do $i \leftarrow 0$ to n, whatever for t

Space: $O(nT)$. Time: $O(nT)$

```
FastSubsetSum(X[1 .. n], T):
    for i ← 0 to n
        SS[i, 0] ← True
    for t ← 1 to T
        SS[0, t] ← False

    for i ← 1 to n
        for t ← 1 to X[i] − 1
            SS[i, t] ← SS[i − 1, t]
        for t ← X[i] to T
            SS[i, t] ← SS[i − 1, t − X[i]] ∨ SS[i − 1, t]

    return SS[n, T]
```