# CS 341: Algorithms

## Lecture 9: Dynamic programming, continued

**Éric Schost**

**based on lecture notes by many other CS341 instructors**

**David R. Cheriton School of Computer Science, University of Waterloo**

**Fall 2024**

# Maximum independent set in a tree

# The problem

**Input:**

- a tree $T$ (connected undirected graph with no cycle) with $n$ vertices

**Output:**

- a maximum cardinality **independent set** of vertices in $T$
- a subset $S$ of vertices is **independent** if there is **no edge** in $T$ connecting two elements of $S$

**Remarks:**

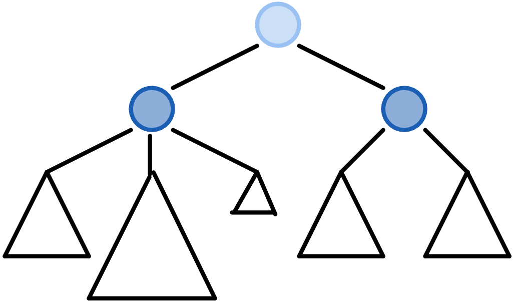
- in a general graph, INDEPENDENTSET is NP-complete
- *a priori* not a rooted tree, but we can suppose we chose a root $\boldsymbol{r}$
- vertices $= \{1, \ldots, n\}$, each vertex stores a linked list of children

# Setting up the recurrence

**Case discussion:** is the root in $S$ or not?

**If no:**

- all its children can be in $S$
- so we look (recursively) at the children of the root
- taking independent sets in children gives an independent set in $T$

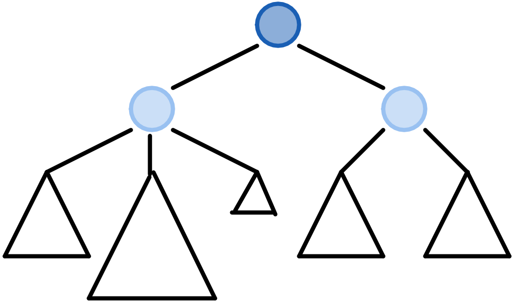


# Setting up the recurrence

**Case discussion:** is the root in $S$ or not?

**If yes:**

- none of its children can be in $S$
- so we can look (recursively) at its **grandchildren**
- taking independent sets in grandchildren gives an independent set in $T$



# Setting up the recurrence

**Finally**

$$O(T) = \max(1 + \sum_{C \text{ grandchild of } r} O(C), \sum_{C' \text{ child of } r} O(C'))$$

**Algorithm:**

- level-order traversal, get an array $V[0..h]$, $V[i]$ =linked list of vertices at level $i$
- **for** $v$ in $V[h]$, set $O[v] = 1$
- **for** $i = h - 1, \ldots, 0$, **for** $v$ in $V[i]$, use the recurrence to get $O[v]$
  (loop over children and grandchildren to get the sums)

# Setting up the recurrence

**Finally**

$$O(T) = \max(1 + \sum_{C \text{ grandchild of } r} O(C), \sum_{C' \text{ child of } r} O(C'))$$

**Runtime:** proportional to

$$\sum_{v \text{ vertex in } T} 1 + \sum_{v \text{ vertex in } T} \#\text{children}(v) + \sum_{v \text{ vertex in } T} \#\text{grandchildren}(v)$$

- second sum is number of vertices of level at least 1
- third sum is number of vertices of level at least 2
- so $\boldsymbol{\Theta(n)}$ altogether

**Exercise**

find the independent set itself

# Optimal binary search trees

# The problem

**Input:**

- integers (or something else that can be ordered) $1, \ldots, n$
- probabilities of access $p_1, \ldots, p_n$, with $p_1 + \cdots + p_n = 1$

**Output:**

- an **optimal** BST with keys $1, \ldots, n$
- **optimal:** minimizes $\sum_{i=1}^{n} p_i \text{depth}(i)$ = expected number of tests for a search (here, depths start at 1)

# The problem

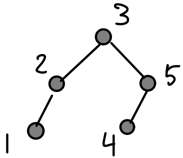
**Input:**

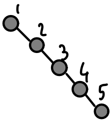
- integers (or something else that can be ordered) $1, \ldots, n$
- probabilities of access $p_1, \ldots, p_n$, with $p_1 + \cdots + p_n = 1$

**Output:**

- an **optimal** BST with keys $1, \ldots, n$
- **optimal:** minimizes $\sum_{i=1}^{n} p_i \text{depth}(i)$ = expected number of tests for a search (here, depths start at 1)

**Example:** $p_1 = p_2 = p_3 = p_4 = p_5 = 1/5$




$1 \cdot \frac{1}{5} + 2 \cdot 2 \cdot \frac{1}{5} + 2 \cdot 3 \cdot \frac{1}{5} = \frac{11}{5}$

$\frac{1}{5}(1+2+3+4+5) = 3$

# The problem

**Input:**

- integers (or something else that can be ordered) $1, \ldots, n$
- probabilities of access $p_1, \ldots, p_n$, with $p_1 + \cdots + p_n = 1$

**Output:**

- an **optimal** BST with keys $1, \ldots, n$
- **optimal:** minimizes $\sum_{i=1}^{n} p_i \text{depth}(i)$ = expected number of tests for a search (here, depths start at 1)

**See also**

- optimal static ordering for **linked lists**
- **Huffman trees**

both built using greedy algorithms

# The problem

**Input:**

- integers (or something else that can be ordered) $1, \dots, n$
- probabilities of access $p_1, \dots, p_n$, with $p_1 + \cdots + p_n = 1$

**Output:**

- an **optimal** BST with keys $1, \dots, n$
- **optimal:** minimizes $\sum_{i=1}^{n} p_i \mathrm{depth}(i)$ = expected number of tests for a search (here, depths start at 1)

**Remark**

- there are $\frac{1}{n+1}\binom{2n}{n}$ binary search trees with $n$ keys
- this is $\boldsymbol{\Theta(4^n/n^{1.5})}$

# Setting up the recurrence

**Definition** for $i, j$ in $\{1, \ldots, n\}$, we define $\boldsymbol{M[i, j]}$ by

- $M[i, j] =$ the minimal cost for items $\{i, \ldots, j\}$, if $i \leq j$
- $M[i, j] = 0$ for $j < i$

**Want:** $M[1, n]$

**Recurrence**

$$M[i,j] = \min_{i \leq k \leq j} \left( \boldsymbol{M[i, k-1]} + \sum_{\ell=i}^{k-1} \boldsymbol{p_\ell} + \boldsymbol{p_k} + \boldsymbol{M[k+1, j]} + \sum_{\ell=k+1}^{j} \boldsymbol{p_\ell} \right)$$

$$= \min_{i \leq k \leq j} \left( M[i, k-1] + M[k+1, j] \right) + \sum_{\ell=i}^{j} p_\ell$$

**check:** gives $M[i, i] = p_i$

# Algorithm

**Remark:** to get $\sum_{\ell=i}^{j} p_\ell$:

- compute $S[\ell] = p_1 + \cdots + p_\ell$, for $\ell = 1, \ldots, n$
- then $p_i + \cdots + p_j = S[j] - S[i-1]$, with $S[0] = 0$

```
OptimalBST(p_1, ..., p_n, S_0, ..., S_n)
    for i = 1, ..., n+1
        M[i, i-1] ← 0
    for d = 0, ..., n-1                    d = j - i
        for i = 1, ..., n-d
            j ← d + i
            M[i, j] ← min_{i≤k≤j}(M[i, k-1] + M[k+1, j]) + S[j] - S[i-1]
```

**Runtime** $\Theta(n^3)$

# A faster algorithm

For all $i, j$, let $\boldsymbol{k_{i,j}}$ be **the largest index that gives the min** at Step 6.

**Claim (difficult)**

For all $i, j$, with $j > i$, we have $\boldsymbol{k_{i,j-1} \leq k_{i,j} \leq k_{i+1,j}}$
(root shifts left (right) if you add elements on the left (right))

```
OptimalBST(p_1, ..., p_n, S_0, ..., S_n)
1.   for i = 1, ..., n+1
2.       M[i, i-1] ← 0
3.   for d = 0, ..., n-1                     d = j - i
4.       for i = 1, ..., n-d
5.           j ← d + i
6.           if d = 0 then range ← {i} else range ← {k_{i,j-1}, ..., k_{i+1,j}}
7.           M[i, j], k_{i,j} ← min_{k∈range}(M[i, k-1] + M[k+1, j]) + S[j] - S[i-1]
```

# Runtime, revisited

Work is proportional to

$$\begin{aligned}\sum_{d=0}^{n-1}\sum_{i=1}^{n-d}(k_{i+1,j}-k_{i,j-1}+1) &= \sum_{d=0}^{n-1}\sum_{i=1}^{n-d}(\boldsymbol{k_{i+1,i+d}-k_{i,i-1+d}}+1)\\ &\le \sum_{d=0}^{n-1}\sum_{i=1}^{n-d}(k_{i+1,i+d}-k_{i,i-1+d})+\sum_{d=0}^{n-1}\sum_{i=1}^{n-d}1\\ &\le \sum_{d=0}^{n-1}(k_{n-d+1,n}-k_{0,d-1})+\sum_{d=0}^{n-1}\sum_{i=1}^{n-d}1\\ &\le 2n^2\end{aligned}$$

**Conclusion:** $\boldsymbol{\Theta(n^2)}$

# Text segmentation

# The problem

**Input:** a string, represented as an array $A[1..n]$

**Output:**

- **true** if we can **segment** of $A$ into words from a given dictionary
- **false** otherwise

(we assume that we can test if $A[i..j]$ is a word in $O(1)$ using **is_word**$[i..j]$)

**Example:** $A$=caramelow $\rightarrow$ **true**, with **car a me low**

**Remark:** there are $2^{n-1}$ ways to segment $A$

# Subproblems and their recurrence

**Subproblems:** can we split $A[1..i]$ into words?

**Definition:** for $i = 1, \ldots, n$, let $s[i]$ be

- **true** if we can **segment** of $A[1..i]$ into words
- **false** otherwise

we set $s[0] = \mathbf{true}$

**Recurrence:** $s[i] = \mathbf{or}_{j=0}^{i-1}\Big(s[j] \text{ } \mathbf{and} \text{ } \mathbf{is\_word}(A[j+1..i])\Big)$

Algorithm could be written recursively, but we'll focus on iterative version

# A polynomial algorithm

```
IsSplittable(A[1..n])
    s[0] ← true
    for i = 1, ..., n do
        s[i] ← false
        for j = 0, ..., i − 1 do
            s[i] ← s[i] or (s[j] and is_word(A[j + 1..i]))
    return s[n]
```

**Runtime:** $\Theta(n^2)$

**Exercise**

return a valid subdivision, if there is one