Lecture 12: Dynamic Programming II

Michael Dinitz

October 7, 2021
601.433/633 Introduction to Algorithms

Introduction

Today: two more examples of dynamic programming

- *Longest Common Subsequence* (strings)
- *Optimal Binary Search Tree* (trees)

Important problems, but really: more examples of dynamic programming

Both in CLRS (unlike Weighted Interval Scheduling)

Longest Common Subsequence

Definitions

String: Sequence of elements of some *alphabet* ($\{0,1\}$, or $\{A - Z\} \cup \{a - z\}$, etc.)

Definition: A sequence $Z = (z_1, \dots, z_k)$ is a *subsequence* of $X = (x_1, \dots, x_m)$ if there exists a strictly increasing sequence $(i_1, i_2, \dots, i_k)$ such that $x_{i_j} = z_j$ for all $j \in \{1, 2, \dots, k\}$.

Example: (B, C, D, B) is a subsequence of (A, B, C, B, D, A, B)

- Allowed to skip positions, unlike substring!

Definitions

String: Sequence of elements of some *alphabet* ($\{0,1\}$, or $\{A-Z\} \cup \{a-z\}$, etc.)

Definition: A sequence $Z = (z_1, \dots, z_k)$ is a *subsequence* of $X = (x_1, \dots, x_m)$ if there exists a strictly increasing sequence $(i_1, i_2, \dots, i_k)$ such that $x_{i_j} = z_j$ for all $j \in \{1, 2, \dots, k\}$.

Example: (B, C, D, B) is a subsequence of (A, B, C, B, D, A, B)

- Allowed to skip positions, unlike substring!

Definition: In *Longest Common Subsequence* problem (LCS) we are given two strings $X = (x_1, \dots, x_m)$ and $Y = (y_1, \dots y_n)$. Need to find the longest Z which is a subsequence of both X and Y.

Subproblems

First and most important step of dynamic programming: define subproblems!

- Not obvious: **X** and **Y** might not even be same length!

Subproblems

First and most important step of dynamic programming: define subproblems!

- Not obvious: $\mathbf{X}$ and $\mathbf{Y}$ might not even be same length!

Prefixes of strings

- $\mathbf{X_i} = (\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_i})$ (so $\mathbf{X} = \mathbf{X_m}$)
- $\mathbf{Y_j} = (\mathbf{y_1}, \mathbf{y_2}, \ldots, \mathbf{y_j})$ (so $\mathbf{Y} = \mathbf{Y_n}$)

Subproblems

First and most important step of dynamic programming: define subproblems!

- Not obvious: $\mathbf{X}$ and $\mathbf{Y}$ might not even be same length!

Prefixes of strings

- $\mathbf{X_i = (x_1, x_2, \ldots, x_i)}$ (so $\mathbf{X = X_m}$)
- $\mathbf{Y_j = (y_1, y_2, \ldots, y_j)}$ (so $\mathbf{Y = Y_n}$)

Definition: Let $\mathbf{OPT(i, j)}$ be longest common subsequence of $\mathbf{X_i}$ and $\mathbf{Y_j}$

So looking for optimal solution $\mathbf{OPT = OPT(m, n)}$

- Last time $\mathbf{OPT}$ denotes value of solution, here denotes solution. Be flexible in notation

Subproblems

First and most important step of dynamic programming: define subproblems!

- Not obvious: $\mathbf{X}$ and $\mathbf{Y}$ might not even be same length!

Prefixes of strings

- $\mathbf{X_i = (x_1, x_2, \ldots, x_i)}$ (so $\mathbf{X = X_m}$)
- $\mathbf{Y_j = (y_1, y_2, \ldots, y_j)}$ (so $\mathbf{Y = Y_n}$)

Definition: Let $\mathbf{OPT(i,j)}$ be longest common subsequence of $\mathbf{X_i}$ and $\mathbf{Y_j}$

So looking for optimal solution $\mathbf{OPT = OPT(m,n)}$

- Last time $\mathbf{OPT}$ denotes value of solution, here denotes solution. Be flexible in notation

Two-dimensional table!

Optimal Substructure

Second step of dynamic programming: prove optimal substructure

- Relationship between subproblems: show that solution to subproblem can be found from solutions to smaller subproblems

Optimal Substructure

Second step of dynamic programming: prove optimal substructure

- Relationship between subproblems: show that solution to subproblem can be found from solutions to smaller subproblems

Theorem

Let $\mathbf{Z} = (\mathbf{z_1}, \ldots, \mathbf{z_k})$ *be an LCS of* $\mathbf{X_i}$ *and* $\mathbf{Y_j}$ *(so* $\mathbf{Z} = \mathbf{OPT(i, j)}$*).*

1. *If* $\mathbf{x_i} = \mathbf{y_j}$*:*

Optimal Substructure

Second step of dynamic programming: prove optimal substructure

- Relationship between subproblems: show that solution to subproblem can be found from solutions to smaller subproblems

Theorem

Let $\mathbf{Z} = (\mathbf{z_1}, \ldots, \mathbf{z_k})$ *be an LCS of* $\mathbf{X_i}$ *and* $\mathbf{Y_j}$ *(so* $\mathbf{Z} = \mathbf{OPT(i, j)}$*).*

1. *If* $\mathbf{x_i} = \mathbf{y_j}$*: then* $\mathbf{z_k} = \mathbf{x_i} = \mathbf{y_j}$ *and* $\mathbf{Z_{k-1}} = \mathbf{OPT(i-1, j-1)}$

Optimal Substructure

Second step of dynamic programming: prove optimal substructure

- Relationship between subproblems: show that solution to subproblem can be found from solutions to smaller subproblems

Theorem

Let $\mathbf{Z} = (\mathbf{z}_1, \ldots, \mathbf{z}_k)$ *be an LCS of* $\mathbf{X}_i$ *and* $\mathbf{Y}_j$ *(so* $\mathbf{Z} = \mathbf{OPT}(i, j)$*).*

1. *If* $\mathbf{x}_i = \mathbf{y}_j$*: then* $\mathbf{z}_k = \mathbf{x}_i = \mathbf{y}_j$ *and* $\mathbf{Z}_{k-1} = \mathbf{OPT}(i-1, j-1)$
2. *If* $\mathbf{x}_i \neq \mathbf{y}_j$ *and* $\mathbf{z}_k \neq \mathbf{x}_i$*:*

Optimal Substructure

Second step of dynamic programming: prove optimal substructure

- Relationship between subproblems: show that solution to subproblem can be found from solutions to smaller subproblems

Theorem

Let $\mathbf{Z} = (\mathbf{z_1}, \ldots, \mathbf{z_k})$ *be an LCS of* $\mathbf{X_i}$ *and* $\mathbf{Y_j}$ *(so* $\mathbf{Z} = \mathbf{OPT(i,j)}$*).*

1. *If* $\mathbf{x_i} = \mathbf{y_j}$*: then* $\mathbf{z_k} = \mathbf{x_i} = \mathbf{y_j}$ *and* $\mathbf{Z_{k-1}} = \mathbf{OPT(i-1, j-1)}$
2. *If* $\mathbf{x_i} \neq \mathbf{y_j}$ *and* $\mathbf{z_k} \neq \mathbf{x_i}$*: then* $\mathbf{Z} = \mathbf{OPT(i-1, j)}$

Optimal Substructure

Second step of dynamic programming: prove optimal substructure

- Relationship between subproblems: show that solution to subproblem can be found from solutions to smaller subproblems

Theorem

Let $\mathbf{Z} = (\mathbf{z_1}, \ldots, \mathbf{z_k})$ *be an LCS of* $\mathbf{X_i}$ *and* $\mathbf{Y_j}$ *(so* $\mathbf{Z} = \mathbf{OPT(i,j)}$*).*

1. *If* $\mathbf{x_i} = \mathbf{y_j}$*: then* $\mathbf{z_k} = \mathbf{x_i} = \mathbf{y_j}$ *and* $\mathbf{Z_{k-1}} = \mathbf{OPT(i-1, j-1)}$
2. *If* $\mathbf{x_i} \neq \mathbf{y_j}$ *and* $\mathbf{z_k} \neq \mathbf{x_i}$*: then* $\mathbf{Z} = \mathbf{OPT(i-1, j)}$
3. *If* $\mathbf{x_i} \neq \mathbf{y_j}$ *and* $\mathbf{z_k} \neq \mathbf{y_j}$*:*

Optimal Substructure

Second step of dynamic programming: prove optimal substructure

- Relationship between subproblems: show that solution to subproblem can be found from solutions to smaller subproblems

Theorem

Let $\mathbf{Z} = (\mathbf{z_1}, \ldots, \mathbf{z_k})$ *be an LCS of* $\mathbf{X_i}$ *and* $\mathbf{Y_j}$ *(so* $\mathbf{Z} = \mathbf{OPT(i, j)}$*).*

1. *If* $\mathbf{x_i} = \mathbf{y_j}$*: then* $\mathbf{z_k} = \mathbf{x_i} = \mathbf{y_j}$ *and* $\mathbf{Z_{k-1}} = \mathbf{OPT(i-1, j-1)}$
2. *If* $\mathbf{x_i} \neq \mathbf{y_j}$ *and* $\mathbf{z_k} \neq \mathbf{x_i}$*: then* $\mathbf{Z} = \mathbf{OPT(i-1, j)}$
3. *If* $\mathbf{x_i} \neq \mathbf{y_j}$ *and* $\mathbf{z_k} \neq \mathbf{y_j}$*: then* $\mathbf{Z} = \mathbf{OPT(i, j-1)}$

Optimal Substructure: Proof (I)

Case 1: If $\mathbf{x_i = y_j}$, then $\mathbf{z_k = x_i = y_j}$ and $\mathbf{Z_{k-1} = OPT(i-1, j-i)}$

Proof Sketch.

Contradiction.

Optimal Substructure: Proof (I)

Case 1: If $\mathbf{x_i = y_j}$, then $\mathbf{z_k = x_i = y_j}$ and $\mathbf{Z_{k-1} = OPT(i-1, j-i)}$

Proof Sketch.

Contradiction.

Part 1: Suppose $\mathbf{x_i = y_j = a}$, but $\mathbf{z_k \neq a}$.

Optimal Substructure: Proof (I)

Case 1: If $\mathbf{x_i = y_j}$, then $\mathbf{z_k = x_i = y_j}$ and $\mathbf{Z_{k-1} = OPT(i-1, j-i)}$

Proof Sketch.

Contradiction.

Part 1: Suppose $\mathbf{x_i = y_j = a}$, but $\mathbf{z_k \neq a}$. Add **a** to end of **Z**, still have common subsequence, longer than LCS. Contradiction

Optimal Substructure: Proof (I)

Case 1: If $x_i = y_j$, then $z_k = x_i = y_j$ and $Z_{k-1} = OPT(i-1, j-i)$

Proof Sketch.

Contradiction.

Part 1: Suppose $x_i = y_j = a$, but $z_k \neq a$. Add a to end of Z, still have common subsequence, longer than LCS. Contradiction

Part 2: Suppose $Z_{k-1} \neq OPT(i-1, j-1)$.

Optimal Substructure: Proof (I)

Case 1: If $\mathbf{x_i} = \mathbf{y_j}$, then $\mathbf{z_k} = \mathbf{x_i} = \mathbf{y_j}$ and $\mathbf{Z_{k-1}} = \mathbf{OPT(i-1, j-i)}$

Proof Sketch.

Contradiction.

Part 1: Suppose $\mathbf{x_i} = \mathbf{y_j} = \mathbf{a}$, but $\mathbf{z_k} \neq \mathbf{a}$. Add $\mathbf{a}$ to end of $\mathbf{Z}$, still have common subsequence, longer than LCS. Contradiction

Part 2: Suppose $\mathbf{Z_{k-1}} \neq \mathbf{OPT(i-1, j-1)}$.

$\Longrightarrow$ $\exists \mathbf{W}$ LCS of $\mathbf{X_{i-1}}, \mathbf{Y_{j-1}}$ of length $> \mathbf{k-1} \Longrightarrow \geq \mathbf{k}$

$\Longrightarrow$ $\mathbf{(W, a)}$ common subsequence of $\mathbf{X_i}, \mathbf{Y_j}$ of length $> \mathbf{k}$

- Contradiction to $\mathbf{Z}$ being LCS of $\mathbf{X_i}$ and $\mathbf{Y_j}$ $\square$

Optimal Substructure: Proof (II)

Case 2: If $\mathbf{x_i} \neq \mathbf{y_j}$ and $\mathbf{z_k} \neq \mathbf{x_i}$ then $\mathbf{Z} = \mathbf{OPT(i-1, j)}$

Optimal Substructure: Proof (II)

Case 2: If $\mathbf{x_i} \neq \mathbf{y_j}$ and $\mathbf{z_k} \neq \mathbf{x_i}$ then $\mathbf{Z = OPT(i-1, j)}$

Proof.

Since $\mathbf{z_k} \neq \mathbf{x_i}$, $\mathbf{Z}$ a common subsequence of $\mathbf{X_{i-1}, Y_j}$

Optimal Substructure: Proof (II)

Case 2: If $\mathbf{x_i \neq y_j}$ and $\mathbf{z_k \neq x_i}$ then $\mathbf{Z = OPT(i-1, j)}$

Proof.

Since $\mathbf{z_k \neq x_i}$, $\mathbf{Z}$ a common subsequence of $\mathbf{X_{i-1}, Y_j}$

$\mathbf{OPT(i-1, j)}$ a common subsequence of $\mathbf{X_i, Y_j}$

$\implies |\mathbf{OPT(i-1, j)}| \leq |\mathbf{OPT(i, j)}| = |\mathbf{Z}|$ (def of $\mathbf{OPT(i, j)}$ and $\mathbf{Z}$)

Optimal Substructure: Proof (II)

Case 2: If $\mathbf{x_i} \neq \mathbf{y_j}$ and $\mathbf{z_k} \neq \mathbf{x_i}$ then $\mathbf{Z} = \mathbf{OPT(i-1, j)}$

Proof.

Since $\mathbf{z_k} \neq \mathbf{x_i}$, $\mathbf{Z}$ a common subsequence of $\mathbf{X_{i-1}, Y_j}$

$\mathbf{OPT(i-1, j)}$ a common subsequence of $\mathbf{X_i, Y_j}$

$\implies |\mathbf{OPT(i-1, j)}| \leq |\mathbf{OPT(i, j)}| = |\mathbf{Z}|$ (def of $\mathbf{OPT(i, j)}$ and $\mathbf{Z}$)

$\implies \mathbf{Z} = \mathbf{OPT(i-1, j)}$ □

Optimal Substructure: Proof (III)

Case 3: If $\mathbf{x_i \neq y_j}$ and $\mathbf{z_k \neq y_j}$ then $\mathbf{Z = OPT(i, j-1)}$

Proof.

Symmetric to Case 2. □

Structure Corollary

Corollary

$$\mathrm{OPT}(i,j) = \begin{cases} \varnothing & \textit{if } i = 0 \textit{ or } j = 0, \\ \mathrm{OPT}(i-1, j-1) \circ x_i & \textit{if } i,j > 0 \textit{ and } x_i = y_j \\ \max(\mathrm{OPT}(i, j-1), \mathrm{OPT}(i-1, j)) & \textit{if } i,j > 0 \textit{ and } x_i \neq y_j \end{cases}$$

Structure Corollary

Corollary

$$\mathbf{OPT(i,j)} = \begin{cases} \varnothing & \textit{if } \mathbf{i = 0} \textit{ or } \mathbf{j = 0}, \\ \mathbf{OPT(i-1, j-1) \circ x_i} & \textit{if } \mathbf{i, j > 0} \textit{ and } \mathbf{x_i = y_j} \\ \mathbf{\max(OPT(i, j-1), OPT(i-1, j))} & \textit{if } \mathbf{i, j > 0} \textit{ and } \mathbf{x_i \neq y_j} \end{cases}$$

Gives obvious recursive algorithm

- Can take exponential time (good exercise at home!)

Dynamic Programming!

- Top-Down: are problems getting "smaller"? What does "smaller" mean?
- Bottom-Up: two-dimensional table! What order to fill it in?

Dynamic Programming Algorithm

```
LCS(X,Y) {
    for($\mathbf{i = 0}$ to $\mathbf{m}$) $\mathbf{M[i,0] = 0}$;
    for($\mathbf{j = 0}$ to $\mathbf{n}$) $\mathbf{M[0,j] = 0}$;
    for($\mathbf{i = 1}$ to $\mathbf{m}$) {
        for($\mathbf{j = 1}$ to $\mathbf{n}$) {
            if($\mathbf{x_i = y_j}$)
                $\mathbf{M[i,j] = 1 + M[i-1,j-1]}$;
            else
                $\mathbf{M[i,j] = max(M[i,j-1], M[i-1,j])}$;
            }
        }
    return $\mathbf{M[m,n]}$;
}
```

Dynamic Programming Algorithm

```
LCS(X,Y) {
    for($\mathbf{i = 0}$ to $\mathbf{m}$) $\mathbf{M[i, 0] = 0}$;
    for($\mathbf{j = 0}$ to $\mathbf{n}$) $\mathbf{M[0, j] = 0}$;
    for($\mathbf{i = 1}$ to $\mathbf{m}$) {
        for($\mathbf{j = 1}$ to $\mathbf{n}$) {
            if($\mathbf{x_i = y_j}$)
                $\mathbf{M[i, j] = 1 + M[i-1, j-1]}$;
            else
                $\mathbf{M[i, j] = max(M[i, j-1], M[i-1, j])}$;
        }
    }
    return $\mathbf{M[m, n]}$;
}
```

Running Time: O(mn)

Correctness

Theorem

$M[i, j] = |OPT(i, j)|$

Correctness

Theorem

$M[i,j] = |OPT(i,j)|$

Proof.

Induction on $i + j$ (or could do on iterations in the algorithm)

Correctness

Theorem

$\mathbf{M[i,j] = |OPT(i,j)|}$

Proof.

Induction on $\mathbf{i + j}$ (or could do on iterations in the algorithm)

Base Case: $\mathbf{i + j = 0 \implies i = j = 0 \implies M[i,j] = 0 = |OPT(i,j)|}$

Correctness

Theorem

$\mathbf{M[i,j] = |OPT(i,j)|}$

Proof.

Induction on $\mathbf{i+j}$ (or could do on iterations in the algorithm)

Base Case: $\mathbf{i+j=0 \implies i=j=0 \implies M[i,j]=0=|OPT(i,j)|}$

Inductive Step: Divide into three cases

1. If $\mathbf{i=0}$ or $\mathbf{j=0}$, then $\mathbf{M[i,j]=0=|OPT(i,j)|}$

Correctness

Theorem

$\mathbf{M[i,j] = |OPT(i,j)|}$

Proof.

Induction on $\mathbf{i+j}$ (or could do on iterations in the algorithm)

Base Case: $\mathbf{i+j=0 \implies i=j=0 \implies M[i,j]=0=|OPT(i,j)|}$

Inductive Step: Divide into three cases

1. If $\mathbf{i=0}$ or $\mathbf{j=0}$, then $\mathbf{M[i,j]=0=|OPT(i,j)|}$
2. If $\mathbf{x_i = y_j}$, then $\mathbf{M[i,j] = 1 + M[i-1,j-1] = 1 + |OPT(i-1,j-1)| = |OPT(i,j)|}$

Correctness

Theorem

$\mathbf{M[i,j] = |OPT(i,j)|}$

Proof.

Induction on $\mathbf{i+j}$ (or could do on iterations in the algorithm)

Base Case: $\mathbf{i+j=0 \implies i=j=0 \implies M[i,j]=0=|OPT(i,j)|}$

Inductive Step: Divide into three cases

1. If $\mathbf{i=0}$ or $\mathbf{j=0}$, then $\mathbf{M[i,j]=0=|OPT(i,j)|}$
2. If $\mathbf{x_i=y_j}$, then $\mathbf{M[i,j]=1+M[i-1,j-1]=1+|OPT(i-1,j-1)|=|OPT(i,j)|}$
3. If $\mathbf{x_i \neq y_j}$, then

$$\begin{aligned}
\mathbf{M[i,j]} &= \mathbf{\max(M[i,j-1],M[i-1,j])} && \text{(def of algorithm)}\\
&= \mathbf{\max(|OPT(i,j-1)|,|OPT(i-1,j)|)} && \text{(induction)}\\
&= \mathbf{|OPT(i,j)|} && \text{(structure thm/corollary)}
\end{aligned}$$

Computing a Solution

Like we talked about last lecture: backtrack through dynamic programming table.

Details in CLRS 15.4

Optimal Binary Search Trees

Problem Definition

Input: probability distribution / search frequency of keys

- $\mathbf{n}$ distinct keys $\mathbf{k_1 < k_2 < \cdots < k_n}$
- For each $\mathbf{i} \in [\mathbf{n}]$, probability $\mathbf{p_i}$ that we search for $\mathbf{k_i}$ (so $\sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{p_i} = \mathbf{1}$)

What's the best binary search tree for these keys and frequencies?

Problem Definition

Input: probability distribution / search frequency of keys

- n distinct keys $k_1 < k_2 < \cdots < k_n$
- For each $i \in [n]$, probability p_i that we search for k_i (so $\sum_{i=1}^{n} p_i = 1$)

What's the best binary search tree for these keys and frequencies?

Cost of searching for k_i in tree T is $\text{depth}_T(k_i) + 1$ (say depth of root = 0)

$\implies E[\text{cost of search in } T] = \sum_{i=1}^{n} p_i(\text{depth}_T(k_i) + 1)$

Problem Definition

Input: probability distribution / search frequency of keys

- $\mathbf{n}$ distinct keys $\mathbf{k_1 < k_2 < \cdots < k_n}$
- For each $\mathbf{i} \in [\mathbf{n}]$, probability $\mathbf{p_i}$ that we search for $\mathbf{k_i}$ (so $\sum_{i=1}^{n} \mathbf{p_i} = \mathbf{1}$)

What's the best binary search tree for these keys and frequencies?

Cost of searching for $\mathbf{k_i}$ in tree $\mathbf{T}$ is $\mathbf{depth_T(k_i) + 1}$ (say depth of root = $\mathbf{0}$)
$\Longrightarrow$ $\mathbf{E}[\text{cost of search in } \mathbf{T}] = \sum_{i=1}^{n} \mathbf{p_i(depth_T(k_i) + 1)}$

Definition: $\mathbf{c(T) = \sum_{i=1}^{n} p_i(depth_T(k_i) + 1)}$

Problem: Find search tree $\mathbf{T}$ minimizing cost.

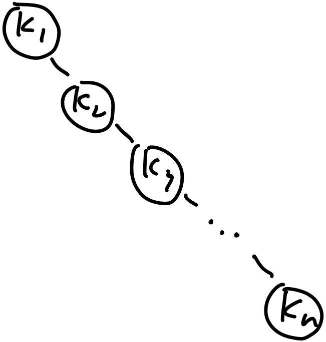
Obvious Approach

Natural approach: greedy (make highest probability key the root). Does this work?

Obvious Approach

Natural approach: greedy (make highest probability key the root). Does this work?

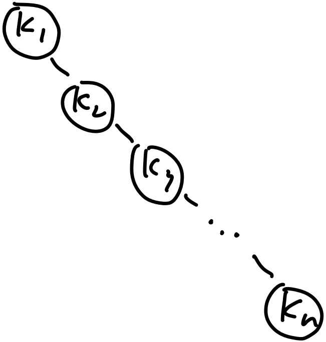
Set $\mathbf{p_1} > \mathbf{p_2} > \ldots \mathbf{p_n}$, but with $\mathbf{p_i} - \mathbf{p_{i+1}}$ extremely small (say $\mathbf{1/2^n}$)



Obvious Approach

Natural approach: greedy (make highest probability key the root). Does this work?

Set $\mathbf{p_1} > \mathbf{p_2} > \ldots \mathbf{p_n}$, but with $\mathbf{p_i} - \mathbf{p_{i+1}}$ extremely small (say $\mathbf{1/2^n}$)

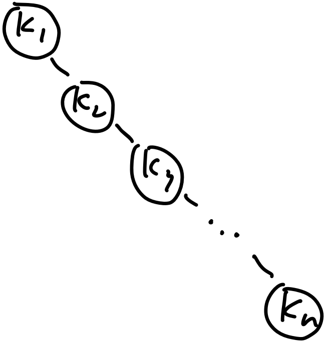


$\mathbf{E}[\text{cost of search in } \mathbf{T}] \geq \mathbf{\Omega(n)}$

Obvious Approach

Natural approach: greedy (make highest probability key the root). Does this work?

Set $\mathbf{p_1} > \mathbf{p_2} > \ldots \mathbf{p_n}$, but with $\mathbf{p_i} - \mathbf{p_{i+1}}$ extremely small (say $\mathbf{1/2^n}$)



$\mathbf{E}[\text{cost of search in } \mathbf{T}] \geq \mathbf{\Omega(n)}$

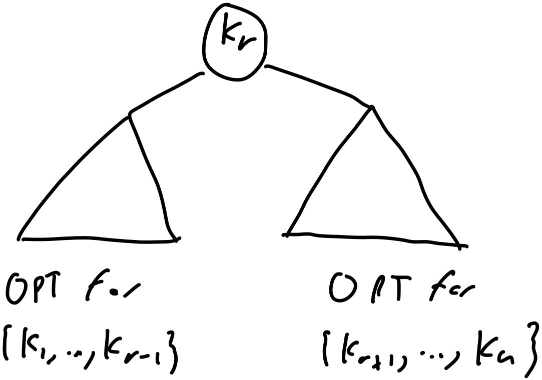
Balanced search tree: $\mathbf{E}[\text{cost}] \leq \mathbf{O(\log n)}$

Intuition

Suppose root is $\mathbf{k_r}$. What does optimal tree look like?

Intuition

Suppose root is $\mathbf{k_r}$. What does optimal tree look like?



Subproblems

Definition

Let $\mathbf{OPT(i,j)}$ with $\mathbf{i \le j}$ be optimal tree for keys $\{\mathbf{k_i, k_{i+1}, \ldots, k_j}\}$: tree $\mathbf{T}$ minimizing $\mathbf{c(T) = \sum_{a=i}^{j} p_a(depth_T(k_a) + 1)}$

By convention, if $\mathbf{i > j}$ then $\mathbf{OPT(i,j)}$ empty
So overall goal is to find $\mathbf{OPT(1,n)}$.

Subproblems

Definition

Let $\mathbf{OPT(i,j)}$ with $\mathbf{i \le j}$ be optimal tree for keys $\{\mathbf{k_i, k_{i+1}, \dots, k_j}\}$: tree $\mathbf{T}$ minimizing $\mathbf{c(T) = \sum_{a=i}^{j} p_a(depth_T(k_a) + 1)}$

By convention, if $\mathbf{i > j}$ then $\mathbf{OPT(i,j)}$ empty
So overall goal is to find $\mathbf{OPT(1,n)}$.

Theorem (Optimal Substructure)

Let $\mathbf{k_r}$ *be the root of* $\mathbf{OPT(i,j)}$*. Then the left subtree of* $\mathbf{OPT(i,j)}$ *is* $\mathbf{OPT(i,r-1)}$*, and the right subtree of* $\mathbf{OPT(i,j)}$ *is* $\mathbf{OPT(r+1,j)}$*.*

Proof Sketch of Optimal Substructure

Definitions:

- Let $\mathbf{T} = \mathbf{OPT(i, j)}$, $\mathbf{T_L}$ its left subtree, $\mathbf{T_R}$ its right subtree.
- Suppose for contradiction $\mathbf{T_L} \neq \mathbf{OPT(i, r-1)}$, let $\mathbf{T'} = \mathbf{OPT(i, r-1)}$

  $\implies$ $\mathbf{c(T') < c(T_L)}$ (def of $\mathbf{OPT(i, r-1)}$)
- Let $\mathbf{\hat{T}}$ be tree get by replacing $\mathbf{T_L}$ with $\mathbf{T'}$

Proof Sketch of Optimal Substructure

Definitions:

- Let $\mathbf{T} = \mathbf{OPT(i, j)}$, $\mathbf{T_L}$ its left subtree, $\mathbf{T_R}$ its right subtree.
- Suppose for contradiction $\mathbf{T_L} \neq \mathbf{OPT(i, r-1)}$, let $\mathbf{T'} = \mathbf{OPT(i, r-1)}$

  $\implies$ $\mathbf{c(T') < c(T_L)}$ (def of $\mathbf{OPT(i, r-1)}$)
- Let $\mathbf{\hat{T}}$ be tree get by replacing $\mathbf{T_L}$ with $\mathbf{T'}$

Whole bunch of math (see lecture notes): get that $\mathbf{c(\hat{T}) < c(T)}$

Contradicts $\mathbf{T} = \mathbf{OPT(i, j)}$

Proof Sketch of Optimal Substructure

Definitions:

- Let $\mathbf{T} = \mathbf{OPT(i,j)}$, $\mathbf{T_L}$ its left subtree, $\mathbf{T_R}$ its right subtree.
- Suppose for contradiction $\mathbf{T_L \neq OPT(i, r-1)}$, let $\mathbf{T' = OPT(i, r-1)}$

  $\Longrightarrow$ $\mathbf{c(T') < c(T_L)}$ (def of $\mathbf{OPT(i, r-1)}$)
- Let $\mathbf{\hat{T}}$ be tree get by replacing $\mathbf{T_L}$ with $\mathbf{T'}$

Whole bunch of math (see lecture notes): get that $\mathbf{c(\hat{T}) < c(T)}$

Contradicts $\mathbf{T = OPT(i,j)}$

Symmetric argument works for $\mathbf{T_R = OPT(r+1, j)}$

Cost Corollary

Corollary

$c(OPT(i,j)) = \sum_{a=i}^{j} p_a + \min_{i \le r \le j}(c(OPT(i,r-1)) + c(OPT(r+1,j)))$

Let k_r be root of $OPT(i,j)$

$$\begin{aligned} c(OPT(i,j)) &= \sum_{a=i}^{j} p_a(\text{depth}_{OPT(i,j)}(k_a)+1) \\ &= \sum_{a=i}^{r-1}(p_a(\text{depth}_{OPT(i,r-1)}(k_a)+2)) + p_r + \sum_{a=r+1}^{j} p_a(\text{depth}_{OPT(r+1,j)}(k_a)+2) \\ &= \sum_{a=i}^{j} p_a + \sum_{a=i}^{r-1}(p_a(\text{depth}_{OPT(i,r-1)}(k_a)+1)) + \sum_{a=r+1}^{j} p_a(\text{depth}_{OPT(r+1,j)}(k_a)+1) \\ &= \sum_{a=i}^{j} p_a + c(OPT(i,r-1)) + c(OPT(r+1,j)). \end{aligned}$$

Cost Corollary

Corollary

$c(OPT(i,j)) = \sum_{a=i}^{j} p_a + \min_{i \le r \le j}(c(OPT(i,r-1)) + c(OPT(r+1,j)))$

Let k_r be root of $OPT(i,j)$

$$\begin{aligned} c(OPT(i,j)) &= \sum_{a=i}^{j} p_a(\text{depth}_{OPT(i,j)}(k_a)+1) \\ &= \sum_{a=i}^{r-1}(p_a(\text{depth}_{OPT(i,r-1)}(k_a)+2)) + p_r + \sum_{a=r+1}^{j} p_a(\text{depth}_{OPT(r+1,j)}(k_a)+2) \\ &= \sum_{a=i}^{j} p_a + \sum_{a=i}^{r-1}(p_a(\text{depth}_{OPT(i,r-1)}(k_a)+1)) + \sum_{a=r+1}^{j} p_a(\text{depth}_{OPT(r+1,j)}(k_a)+1) \\ &= \sum_{a=i}^{j} p_a + c(OPT(i,r-1)) + c(OPT(r+1,j)). \end{aligned}$$

Same logic holds for any possible root $\implies$ take min

Algorithm

Fill in table **M**:

$$\mathbf{M}[\mathbf{i},\mathbf{j}] = \begin{cases} \mathbf{0} & \text{if } \mathbf{i} > \mathbf{j} \\ \mathbf{min}_{\mathbf{i} \le \mathbf{r} \le \mathbf{j}} \left(\sum_{\mathbf{a}=\mathbf{i}}^{\mathbf{j}} \mathbf{p}_{\mathbf{a}} + \mathbf{M}[\mathbf{i}, \mathbf{r}-\mathbf{1}] + \mathbf{M}[\mathbf{r}+\mathbf{1}, \mathbf{j}] \right) & \text{if } \mathbf{i} \le \mathbf{j} \end{cases}$$

Algorithm

Fill in table **M**:

$$\mathbf{M}[\mathbf{i},\mathbf{j}] = \begin{cases} \mathbf{0} & \text{if } \mathbf{i} > \mathbf{j} \\ \mathbf{min}_{\mathbf{i} \le \mathbf{r} \le \mathbf{j}} \left(\sum_{\mathbf{a}=\mathbf{i}}^{\mathbf{j}} \mathbf{p}_{\mathbf{a}} + \mathbf{M}[\mathbf{i},\mathbf{r}-\mathbf{1}] + \mathbf{M}[\mathbf{r}+\mathbf{1},\mathbf{j}] \right) & \text{if } \mathbf{i} \le \mathbf{j} \end{cases}$$

Top-Down (memoization): are problems getting smaller?

Algorithm

Fill in table **M**:

$$\mathbf{M}[\mathbf{i},\mathbf{j}] = \begin{cases} \mathbf{0} & \text{if } \mathbf{i} > \mathbf{j} \\ \mathbf{min}_{\mathbf{i} \le \mathbf{r} \le \mathbf{j}} \left(\sum_{\mathbf{a}=\mathbf{i}}^{\mathbf{j}} \mathbf{p}_{\mathbf{a}} + \mathbf{M}[\mathbf{i}, \mathbf{r}-\mathbf{1}] + \mathbf{M}[\mathbf{r}+\mathbf{1}, \mathbf{j}] \right) & \text{if } \mathbf{i} \le \mathbf{j} \end{cases}$$

Top-Down (memoization): are problems getting smaller? Yes! $\mathbf{j} - \mathbf{i}$ decreases in every recursive call.

Algorithm

Fill in table $\mathbf{M}$:

$$\mathbf{M}[\mathbf{i},\mathbf{j}] = \begin{cases} \mathbf{0} & \text{if } \mathbf{i} > \mathbf{j} \\ \min_{\mathbf{i} \le \mathbf{r} \le \mathbf{j}} \left(\sum_{\mathbf{a}=\mathbf{i}}^{\mathbf{j}} \mathbf{p_a} + \mathbf{M}[\mathbf{i}, \mathbf{r}-\mathbf{1}] + \mathbf{M}[\mathbf{r}+\mathbf{1}, \mathbf{j}] \right) & \text{if } \mathbf{i} \le \mathbf{j} \end{cases}$$

Top-Down (memoization): are problems getting smaller? Yes! $\mathbf{j} - \mathbf{i}$ decreases in every recursive call.

Correctness. Claim $\mathbf{M}[\mathbf{i},\mathbf{j}] = \mathbf{c}(\mathbf{OPT}(\mathbf{i},\mathbf{j}))$. Induction on $\mathbf{j} - \mathbf{i}$.

Algorithm

Fill in table **M**:

$$\mathbf{M}[\mathbf{i},\mathbf{j}] = \begin{cases} \mathbf{0} & \text{if } \mathbf{i} > \mathbf{j} \\ \min_{\mathbf{i} \le \mathbf{r} \le \mathbf{j}} \left(\sum_{\mathbf{a}=\mathbf{i}}^{\mathbf{j}} \mathbf{p}_{\mathbf{a}} + \mathbf{M}[\mathbf{i}, \mathbf{r}-\mathbf{1}] + \mathbf{M}[\mathbf{r}+\mathbf{1}, \mathbf{j}] \right) & \text{if } \mathbf{i} \le \mathbf{j} \end{cases}$$

Top-Down (memoization): are problems getting smaller? Yes! $\mathbf{j} - \mathbf{i}$ decreases in every recursive call.

Correctness. Claim $\mathbf{M}[\mathbf{i},\mathbf{j}] = \mathbf{c}(\mathbf{OPT}(\mathbf{i},\mathbf{j}))$. Induction on $\mathbf{j} - \mathbf{i}$.

- Base case: if $\mathbf{j} - \mathbf{i} < \mathbf{0}$ then $\mathbf{M}[\mathbf{i},\mathbf{j}] = \mathbf{OPT}(\mathbf{i},\mathbf{j}) = \mathbf{0}$

Algorithm

Fill in table **M**:

$$\mathbf{M}[\mathbf{i},\mathbf{j}] = \begin{cases} \mathbf{0} & \text{if } \mathbf{i} > \mathbf{j} \\ \min_{\mathbf{i}\le \mathbf{r}\le \mathbf{j}} \left(\sum_{\mathbf{a}=\mathbf{i}}^{\mathbf{j}} \mathbf{p}_\mathbf{a} + \mathbf{M}[\mathbf{i},\mathbf{r}-\mathbf{1}] + \mathbf{M}[\mathbf{r}+\mathbf{1},\mathbf{j}] \right) & \text{if } \mathbf{i} \le \mathbf{j} \end{cases}$$

Top-Down (memoization): are problems getting smaller? Yes! $\mathbf{j}-\mathbf{i}$ decreases in every recursive call.

Correctness. Claim $\mathbf{M}[\mathbf{i},\mathbf{j}] = \mathbf{c}(\mathbf{OPT}(\mathbf{i},\mathbf{j}))$. Induction on $\mathbf{j}-\mathbf{i}$.

- Base case: if $\mathbf{j}-\mathbf{i}<\mathbf{0}$ then $\mathbf{M}[\mathbf{i},\mathbf{j}] = \mathbf{OPT}(\mathbf{i},\mathbf{j}) = \mathbf{0}$
- Inductive step:

$$\begin{aligned} \mathbf{M}[\mathbf{i},\mathbf{j}] &= \min_{\mathbf{i}\le \mathbf{r}\le \mathbf{j}} \left(\sum_{\mathbf{a}=\mathbf{i}}^{\mathbf{j}} \mathbf{p}_\mathbf{a} + \mathbf{M}[\mathbf{i},\mathbf{r}-\mathbf{1}] + \mathbf{M}[\mathbf{r}+\mathbf{1},\mathbf{j}] \right) && \text{(alg def)} \\ &= \min_{\mathbf{i}\le \mathbf{r}\le \mathbf{j}} \left(\sum_{\mathbf{a}=\mathbf{i}}^{\mathbf{j}} \mathbf{p}_\mathbf{a} + \mathbf{c}(\mathbf{OPT}(\mathbf{i},\mathbf{r}-\mathbf{1})) + \mathbf{c}(\mathbf{OPT}(\mathbf{r}+\mathbf{1},\mathbf{j})) \right) && \text{(induction)} \\ &= \mathbf{c}(\mathbf{OPT}(\mathbf{i},\mathbf{j})) && \text{(cost corollary)} \end{aligned}$$

Algorithm: Bottom-up

What order to fill the table in?

- Obvious approach: for($\mathbf{i} = \mathbf{1}$ to $\mathbf{n} - \mathbf{1}$) for($\mathbf{j} = \mathbf{i} + \mathbf{1}$ to $\mathbf{n}$) Doesn't work!

Algorithm: Bottom-up

What order to fill the table in?

- Obvious approach: for($i = 1$ to $n - 1$) for($j = i + 1$ to n) Doesn't work!
- Take hint from induction: $j - i$

```
OBST {
    Set $M[i, j] = 0$ for all $j > i$;
    Set $M[i, i] = p_i$ for all $i$
    for($\ell = 1$ to $n - 1$) {
        for($i = 1$ to $n - \ell$) {
            $j = i + \ell$
            $M[i, j] = \min_{i \le r \le j} \left( \sum_{a=i}^{j} p_a + M[i, r - 1] + M[r + 1, j] \right)$;
        }
    }
    return $M[1, n]$;
}
```

Analysis

Correctness: same as top-down

Running Time:

Analysis

Correctness: same as top-down

Running Time:

- # table entries:

Analysis

Correctness: same as top-down

Running Time:

- # table entries: $\mathbf{O(n^2)}$

Analysis

Correctness: same as top-down

Running Time:

- # table entries: $\mathbf{O(n^2)}$
- Time to compute table entry $\mathbf{M[i,j]}$:

Analysis

Correctness: same as top-down

Running Time:

- # table entries: $\mathbf{O(n^2)}$
- Time to compute table entry $\mathbf{M[i,j]}$: $\mathbf{O(j-i) = O(n)}$

Analysis

Correctness: same as top-down

Running Time:

- # table entries: $\mathbf{O(n^2)}$
- Time to compute table entry $\mathbf{M[i,j]}$: $\mathbf{O(j-i) = O(n)}$

Total running time: $\mathbf{O(n^3)}$