CS473 - Algorithms I

Other Dynamic Programming Problems

View in slide-show mode

Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Problem 1
Subset Sum

Subset-Sum Problem

Given:

- a set of integers $X = \{x_1, x_2, \ldots, x_n\}$, and
- an integer B

Find:

- a subset of X that has **maximum sum not exceeding** B.

Notation: $S_{n,B} = \{x_1, x_2, \ldots, x_n: B\}$ is the subset-sum problem

- *The integers to choose from:* $x_1, x_2, \ldots, x_n$
- *Desired sum:* B

Subset-Sum Problem

Example:

$$\begin{array}{l} \quad\;\; \color{red}{x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6 \quad x_7 \quad x_8 \quad x_9 \quad x_{10} \quad x_{11} \quad x_{12}} \\ \color{blue}{S_{12,99}: \{20,\ 30,\ 14,\ 70,\ 40,\ 50,\ 15,\ 25,\ 80,\ 60,\ 10,\ 95: 99\}} \end{array}$$

Find a subset of X with maximum sum not exceeding 99.

An optimal solution:

$$\begin{array}{l} \qquad\quad \color{red}{x_1 \quad x_3 \quad x_5 \quad x_8} \\ \color{blue}{N_{opt} = \{20,\ 14,\ 40,\ 25\}} \end{array}$$

with sum 20 + 14 + 40 + 25 = 99

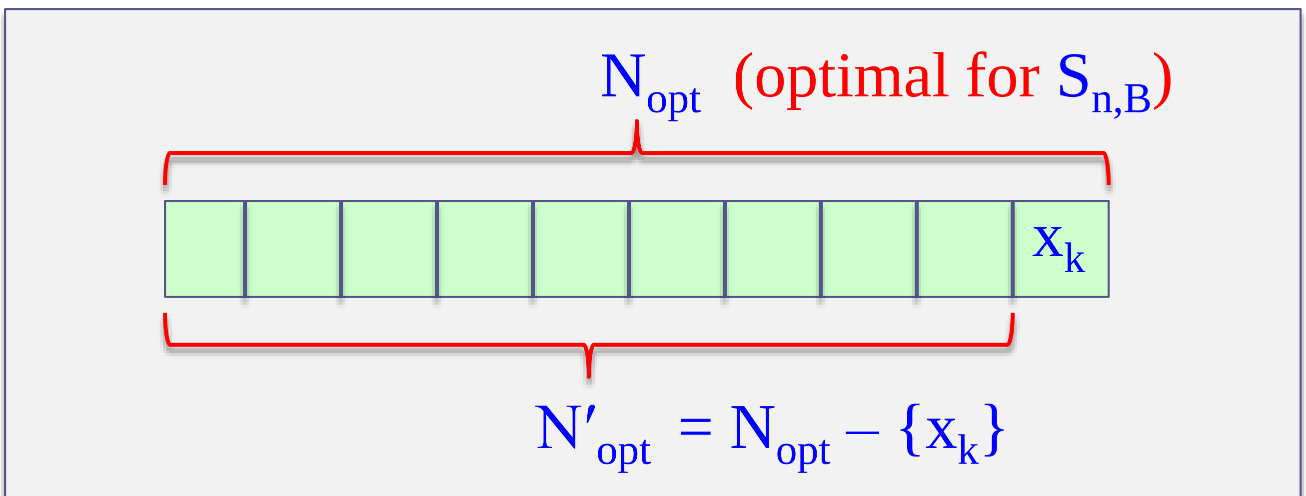
Optimal Substructure Property

- Consider the solution as a sequence of n decisions:

  i^{th} decision: whether we pick number x_i or not

Let N_{opt} be an optimal solution for $S_{n,B}$

Let x_k be the highest-indexed number in N_{opt}



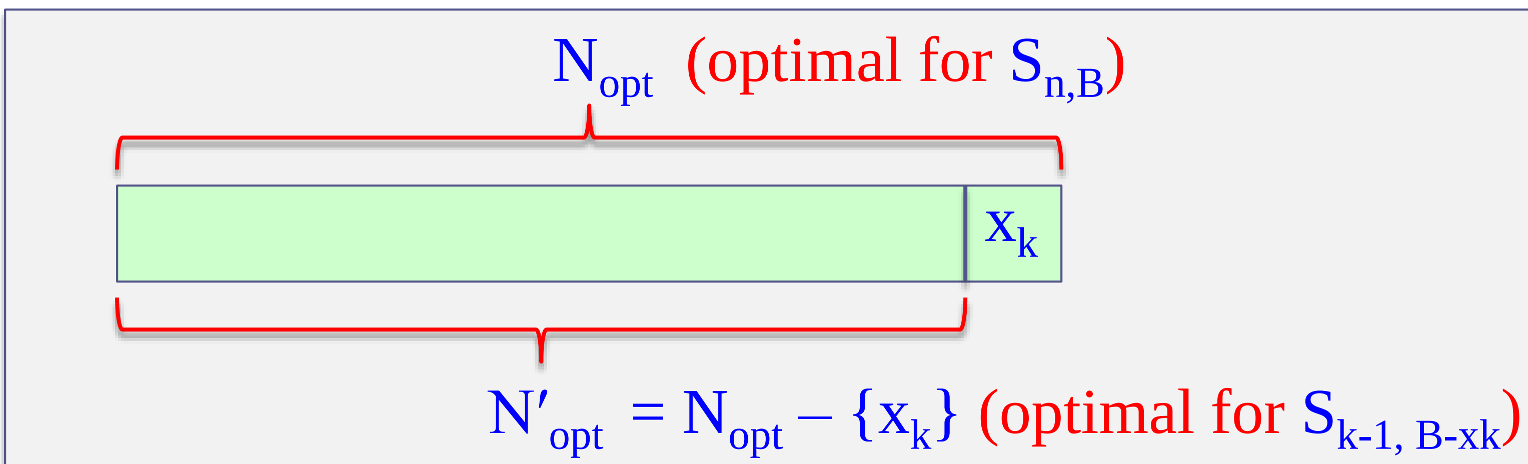
Optimal Substructure Property

Lemma: $N'_{opt} = N_{opt} - \{x_k\}$ is an optimal solution for the subproblem $S_{k-1,B-xk} = \{x_1, x_2, \ldots, x_{k-1}: B-x_k\}$

and

$$c(N_{opt}) = x_k + c(N'_{opt})$$

where $c(N)$ is the sum of all numbers in subset N



Optimal Substructure Property - Proof

Proof: By contradiction, assume that there exists another solution A' for $S_{k-1,\, B - xk}$ for which:

$$c(A') > c(N'_{opt}) \text{ and } c(A') \leq B - x_k$$

i.e. A' is a better solution than N'_{opt} for $S_{k-1,\, B-xk}$

Then, we can construct $A = A' \cup \{x_k\}$ as a solution to $S_{k,\, B}$.

We have:

$$c(A) = c(A') + x_k$$
$$> c(N'_{opt}) + x_k = c(N_{opt})$$

Contradiction! N_{opt} was assumed to be optimal for $S_{k,B}$.

Proof complete.

Optimal Substructure Property - Example

Example:

$$\begin{array}{l} \quad\quad\; x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6 \quad x_7 \quad x_8 \quad x_9 \quad x_{10} \quad x_{11} \quad x_{12} \\ S_{12,99}: \{20,\ 30,\ 14,\ 70,\ 40,\ 50,\ 15,\ 25,\ 80,\ 60,\ 10,\ 95: 99\} \end{array}$$

$$\begin{array}{l} \quad\quad\;\; x_1 \quad x_3 \quad x_5 \quad x_8 \\ N_{opt} = \{20,\ 14,\ 40,\ 25\} \end{array}$$ is optimal for $S_{12,\,99}$

$N'_{opt} = N_{opt} - \{x_8\} = \{20, 14, 40\}$ is optimal for

the subproblem $S_{7,74} = \{20, 30, 14, 70, 40, 50, 15: 74\}$ (elements labelled $x_1, x_2, x_3, x_4, x_5, x_6, x_7$)

and

$c(N_{opt}) = 25 + c(N'_{opt}) = 25 + 74 = 99$

Recursive Definition an Optimal Solution

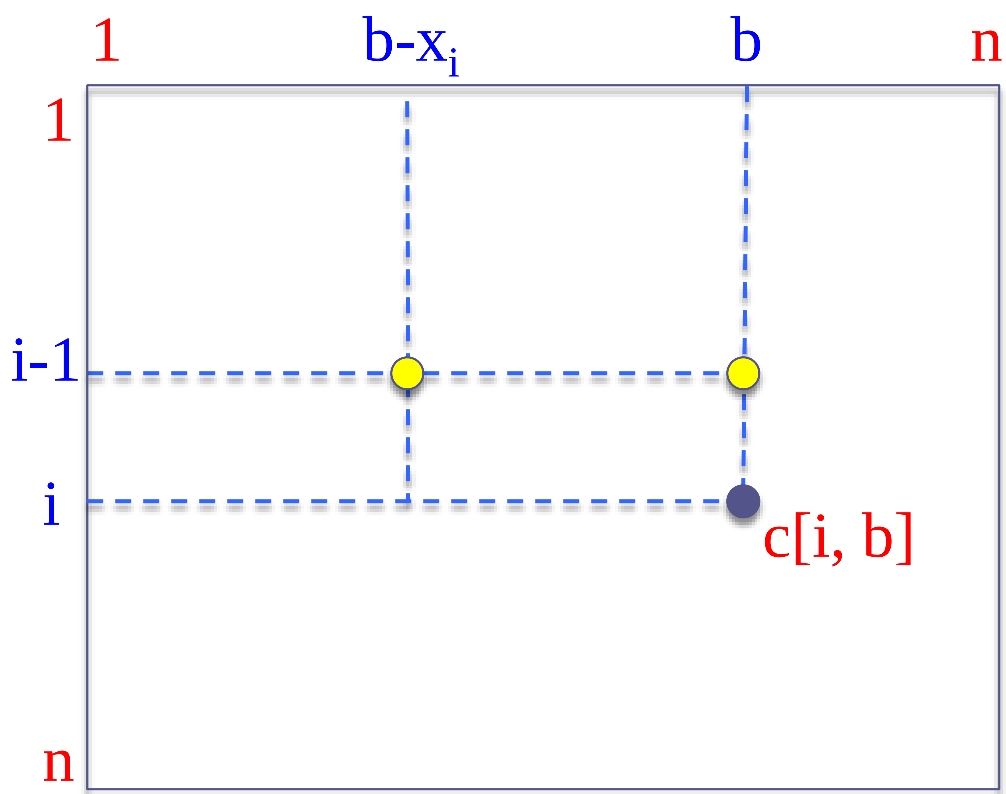
$c[i, b]$: the value of an optimal solution for $S_{i,b} = \{x_1, \ldots, x_i : b\}$

$$c[i,b] = \begin{cases} 0 & \text{if } i = 0 \text{ or } b = 0 \\ c[i-1, b] & \text{if } x_i > b \\ Max\{x_i + c[i-1, b-x_i], c[i-1, b]\} & \text{if } i > 0 \text{ and } b \geq x_i \end{cases}$$

According to this recurrence, an optimal solution $N_{i,b}$ for $S_{i,b}$:

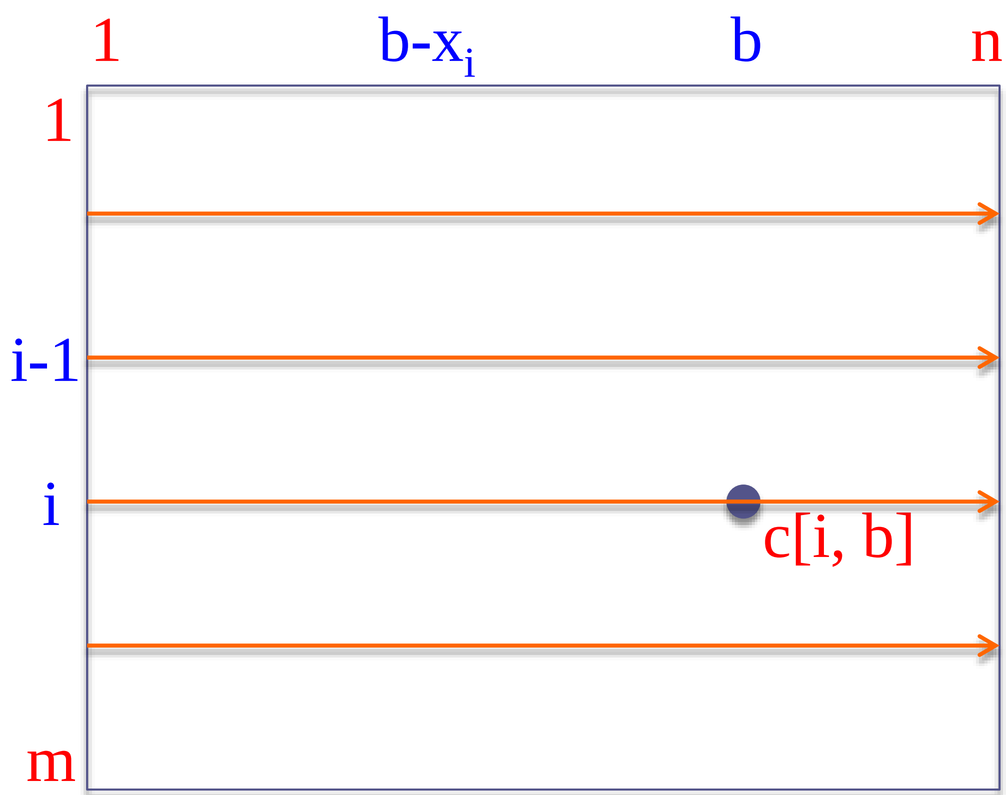
- either contains x_i $\Rightarrow c(N_{i,b}) = x_i + c(N_{i-1,\, b-x_i})$
- or does not contain x_i $\Rightarrow c(N_{i,b}) = c(N_{i-1,\, b})$

$$c[i,b] = \begin{cases} 0 & \textit{if } \text{i} = 0 \text{ or } \text{b} = 0 \\ c[i-1,b] & \textit{if } \text{x}_i > b \\ Max\{x_i + c[i-1,b-x_i], c[i-1,b]\} & \textit{if } \text{i} > 0 \text{ and } \text{b} \geq \text{x}_i \end{cases}$$



Need to process:
c[i, b]
after computing:
c[i-1, b],
c[i-1, b-x_i]

$$c[i,b] = \begin{cases} 0 & \textit{if } \mathrm{i} = 0 \text{ or } \mathrm{b} = 0 \\ c[i-1,b] & \textit{if } \mathrm{x}_i > b \\ Max\{x_i + c[i-1, b-x_i], c[i-1,b]\} & \textit{if } \mathrm{i} > 0 \text{ and } \mathrm{b} \geq \mathrm{x}_i \end{cases}$$



for i ← 1 **to** m
    for b ← 1 **to** n
        ….
        ….
        c[i, b] =

Computing the Optimal Subset-Sum Value

SUBSET-SUM (x, n, B)

```
for b ← 0 to B do
    c[0, b] ← 0
for i ← 1 to n do
    c[i, 0] ← 0
for i ← 1 to n do
    for b ← 1 to B do
        if x_i ≤ b then
            c[i, b] ← Max{x_i + c[i-1, b-x_i], c[i-1, b]}
        else
            c[i, b] ← c[i-1, b]
return c[n, B]
```

Finding an Optimal Subset

SOLUTION-SUBSET-SUM (x, b, B, c)

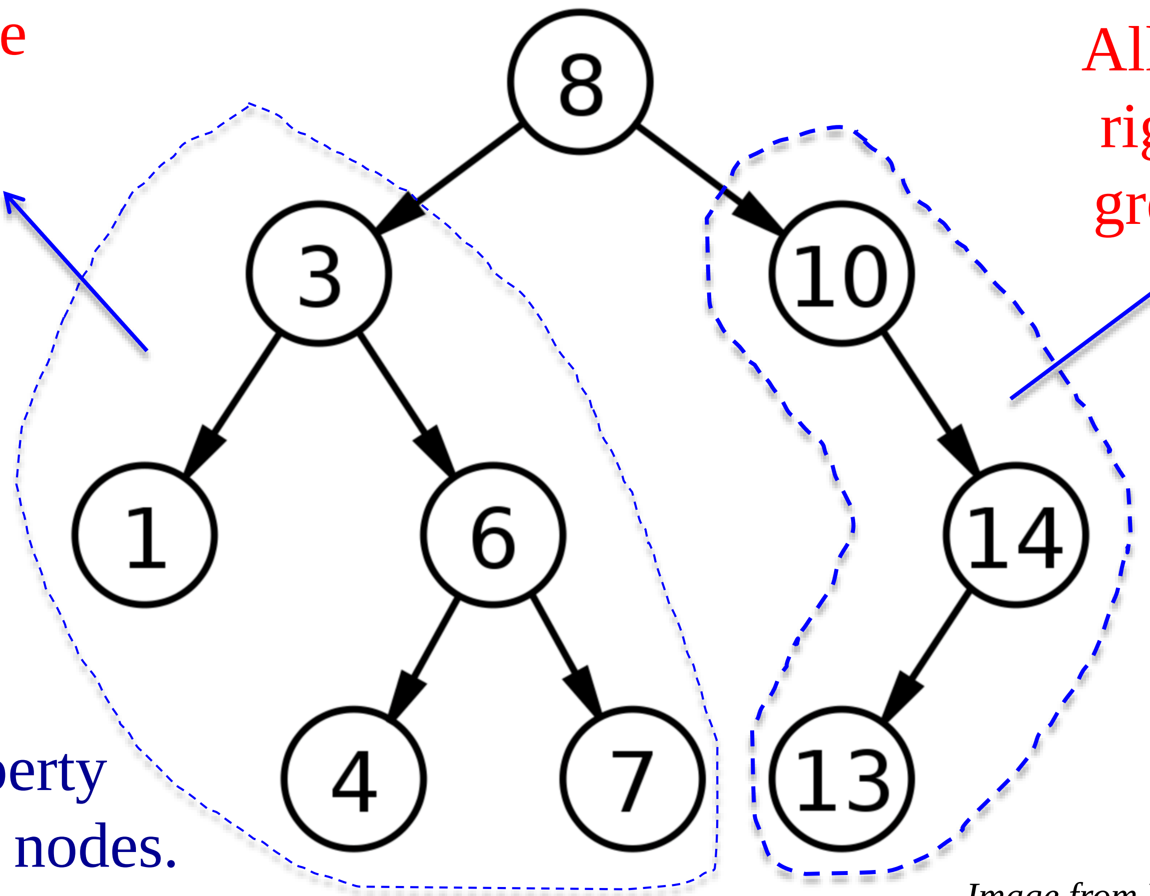
```
i ← n
b ← B
N ← Ø
while i > 0 do
    if c[i, b] = c[i-1, b] then
        i ← i – 1
    else
        N ← N ∪ {x_i}
        i ← i – 1
        b ← b – x_i
return N
```

Problem 2
Optimal Binary Search Tree

Reminder: Binary Search Tree (BST)

All keys in the left subtree less than 8

All keys in the right subtree greater than 8



This property holds for all nodes.

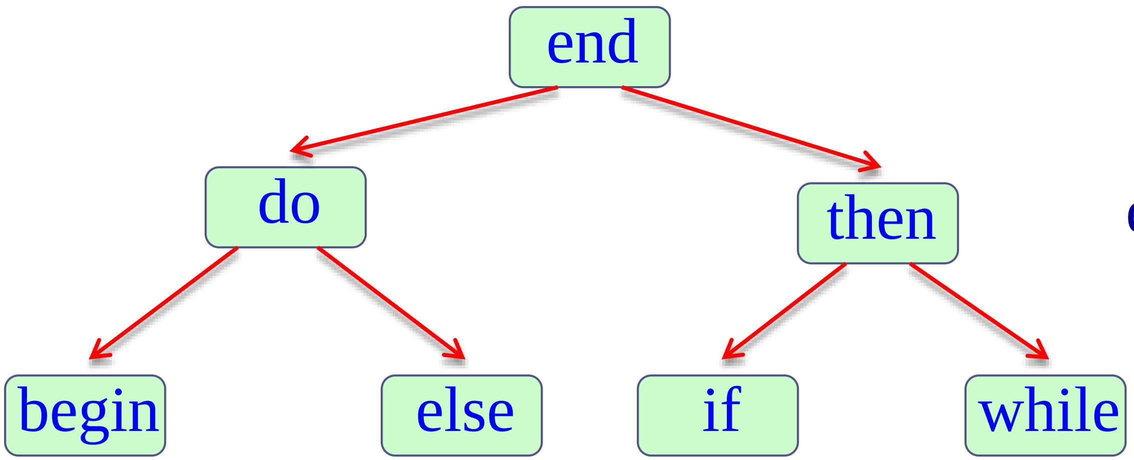
Image from Wikimedia

Binary Search Tree Example

Example: English-to-French translation

Organize (English, French) word pairs in a BST

- Keyword: English word
- Satellite data: French word

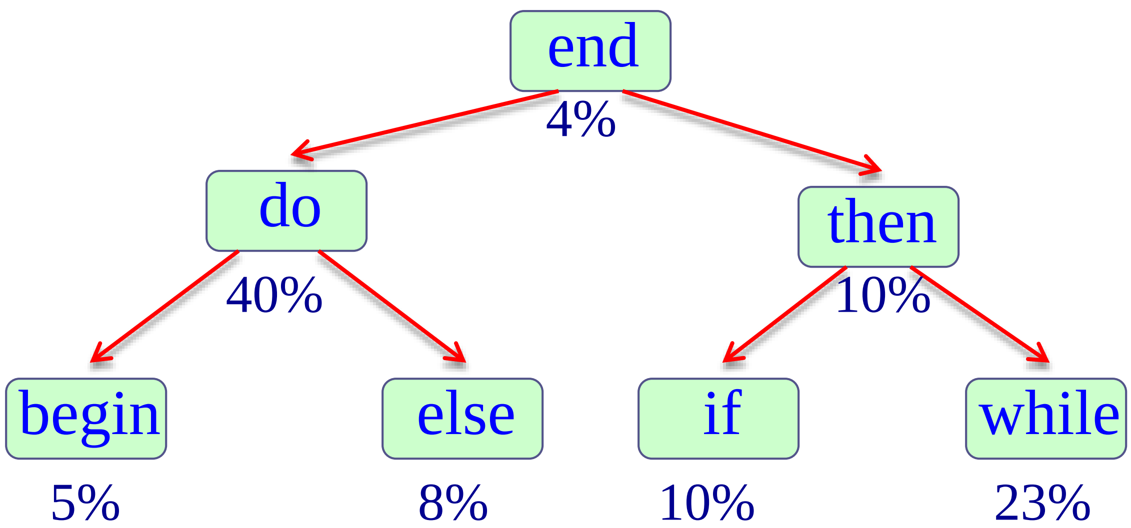


We can search for an English word (node key) efficiently, and return the corresponding French word (satellite data).

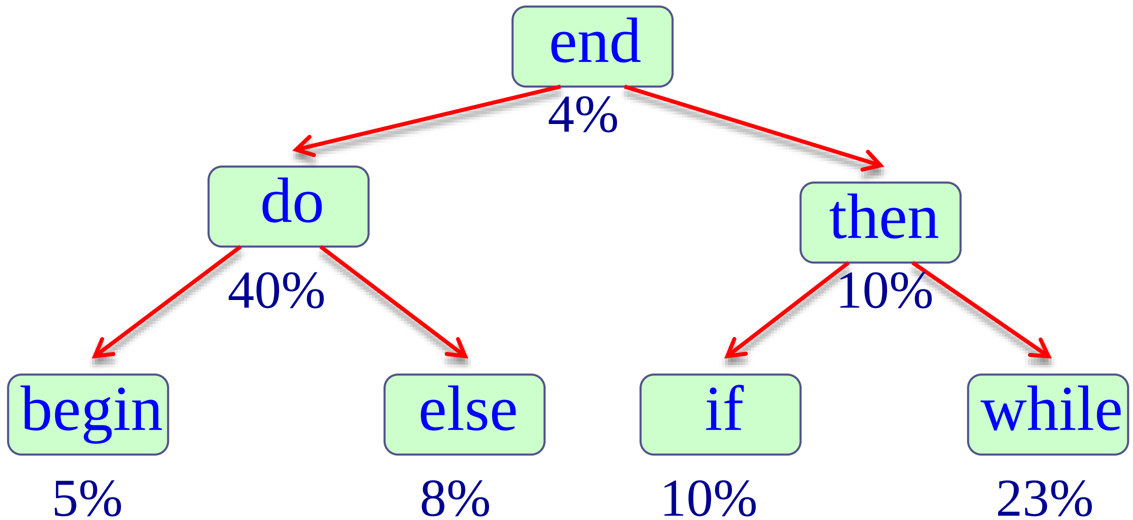
Binary Search Tree Example

Suppose we know the frequency of each keyword in texts:

begin	do	else	end	if	then	while
5%	40%	8%	4%	10%	10%	23%



Cost of a Binary Search Tree

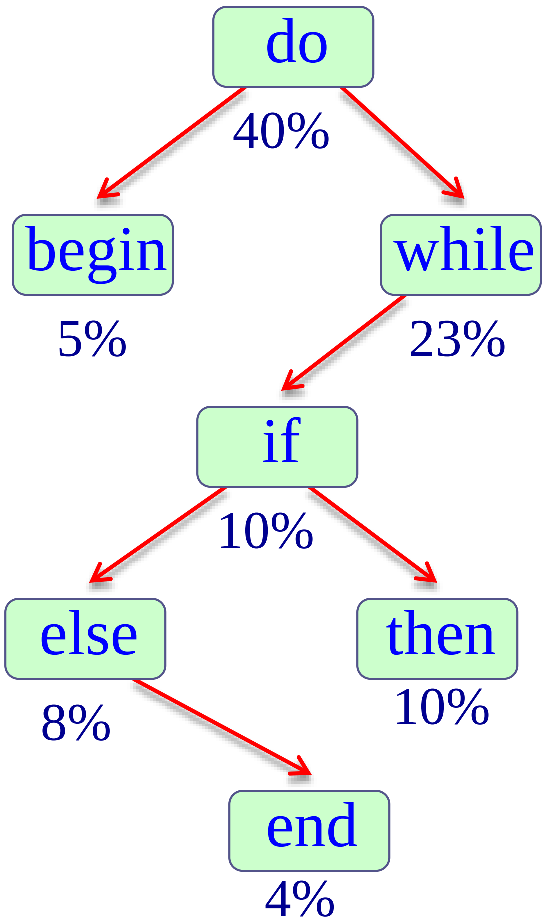


Example: If we search for keyword “while”, we need to access 3 nodes. So, 23% of the queries will have cost of 3.

$$\text{Total cost} = \sum_{i} (depth(i)+1) \times freq(i)$$

= 1x0.04 + 2x0.4 + 2x0.1 + 3x0.05 + 3x0.08 + 3x0.1 + 3x0.23
= 2.42

Cost of a Binary Search Tree



A different binary search tree (BST) leads to a different total cost:

$$\text{Total cost} = 1\text{x}0.4 + 2\text{x}0.05 + 2\text{x}0.23 + 3\text{x}0.1 + 4\text{x}0.08 + 4\text{x}0.1 + 5\text{x}0.04 = 2.18$$

This is in fact an optimal BST.

Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Optimal Binary Search Tree Problem

Given:

A collection of n keys $K_1 < K_2 < \ldots K_n$ to be stored in a BST.

The corresponding p_i values for $1 \leq i \leq n$

p_i: probability of searching for key K_i

Find:

An optimal BST with minimum total cost:

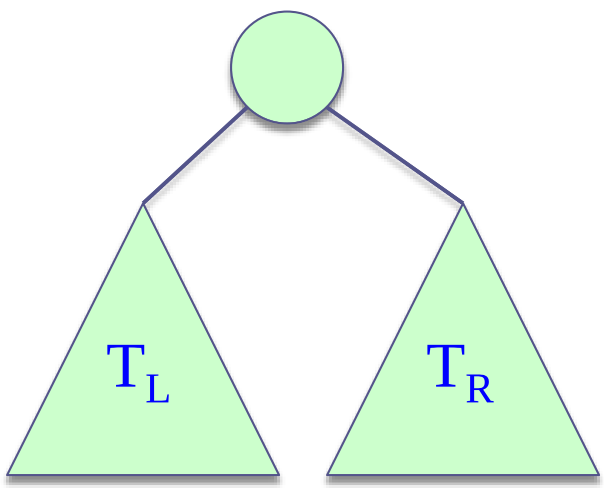
$$\text{Total cost} = \sum_i (depth(i) + 1) \times freq(i)$$

Note: The BST will be static. Only search operations will be performed. No insert, no delete, etc.

Cost of a Binary Search Tree

Lemma 1: Let T_{ij} be a BST containing keys $K_i < K_{i+1} < \ldots < K_j$.
Let T_L and T_R be the left and right subtrees of T. Then we have:

$$\text{cost}(T_{ij}) = \text{cost}(T_L) + \text{cost}(T_R) + \sum_{h=i}^{j} p_h$$



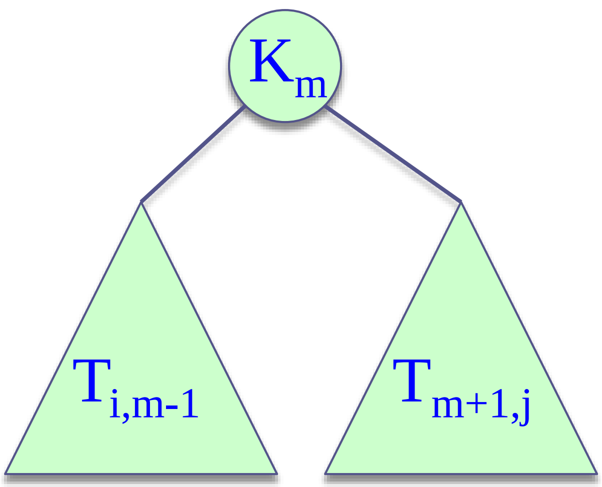
Intuition: When we add the root node, the depth of each node in T_L and T_R increases by 1. So, the cost of node h increases by p_h. In addition, the cost of root node r is p_r. That's why, we have the last term at the end of the formula above.

Optimal Substructure Property

Lemma 2: Optimal substructure property

Consider an optimal BST T_{ij} for keys $K_i < K_{i+1} < \ldots < K_j$

Let K_m be the key at the root of T_{ij}



Then:

$T_{i,m-1}$ is an optimal BST for subproblem containing keys: $K_i < \ldots < K_{m-1}$

$T_{m+1,j}$ is an optimal BST for subproblem containing keys: $K_{m+1} < \ldots < K_j$

$$\boxed{\text{cost}(T_{ij}) = \text{cost}(T_{i,m-1}) + \text{cost}(T_{m+1,j}) + \sum_{h=i}^{j} p_h}$$

Recursive Formulation

Note: We don't know which root vertex leads to the minimum total cost. So, we need to try each vertex m, and choose the one with minimum total cost.

c[i, j]: cost of an optimal BST T_{ij} for the subproblem $K_i < \ldots < K_j$

$$c[i,j] = \begin{cases} 0 & \textit{if } i > j \\ \min\limits_{i \le r \le j} \{c[i, r-1] + c[r+1, j] + P_{ij}\} & \textit{otherwise} \end{cases}$$

where $P_{ij} = \sum_{h=i}^{j} p_h$

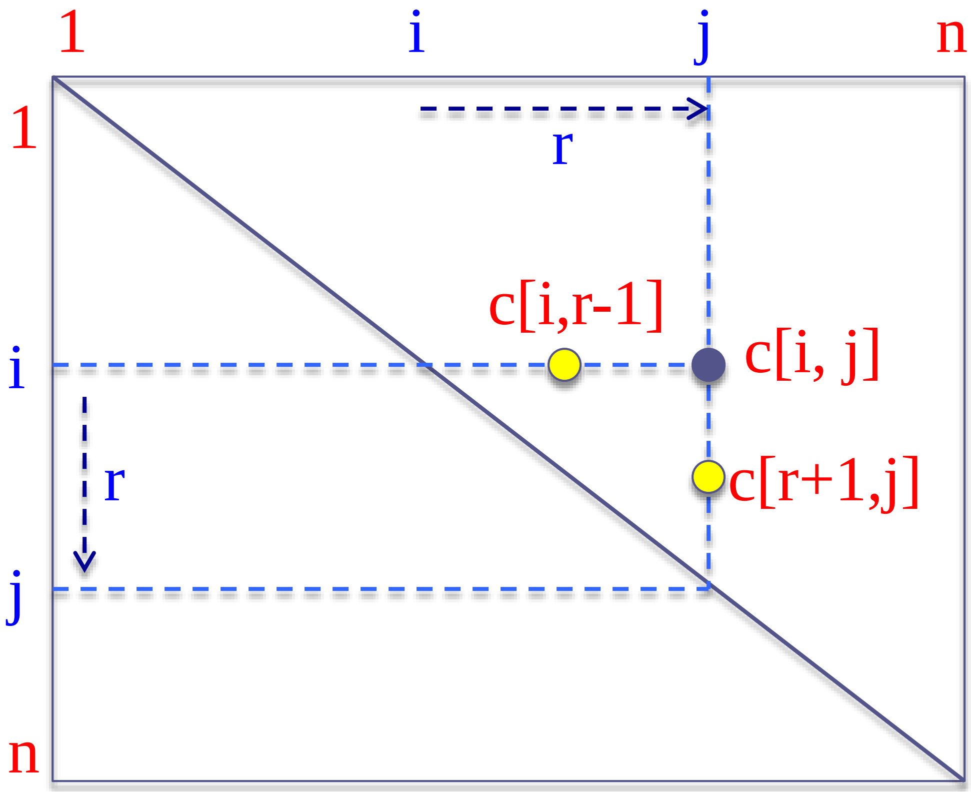
Bottom-up computation

$$c[i,j] = \begin{cases} 0 & \textit{if } \mathrm{i} > \mathrm{j} \\ \min\limits_{i \le r \le j} \{ c[i, r-1] + c[r+1, j] + P_{ij} \} & \textit{otherwise} \end{cases}$$

How to choose the order in which we process c[i, j] values?

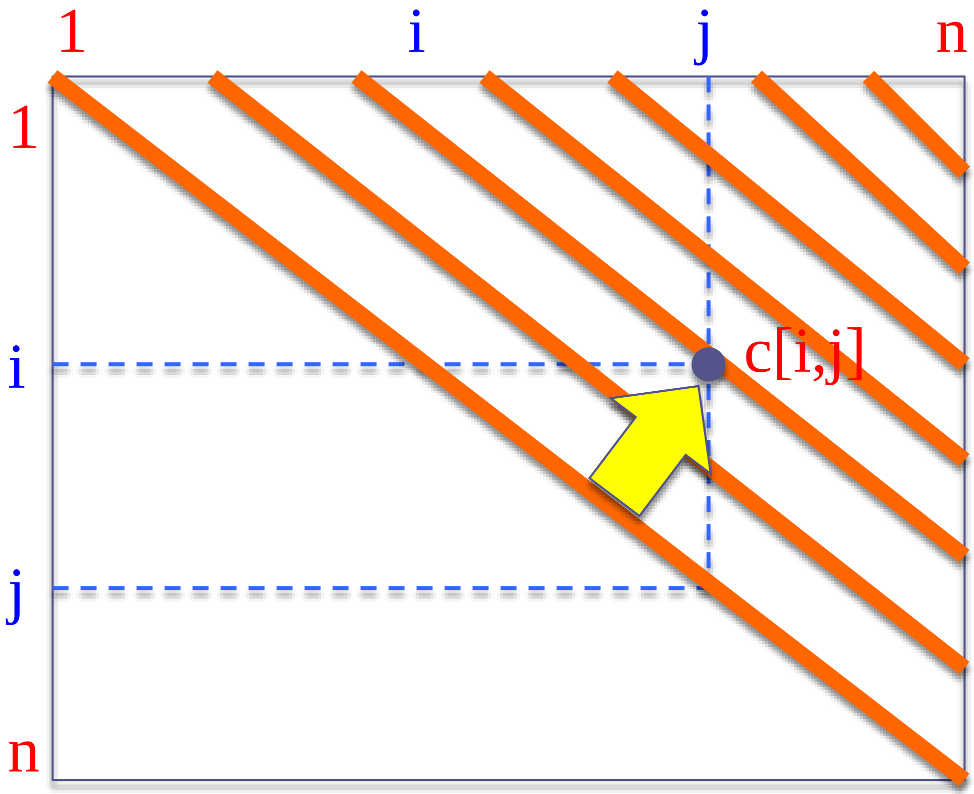
Before computing c[i, j], we have to make sure that the values for c[i, r-1] and c[r+1,j] have been computed for all r.

$$c[i,j] = \begin{cases} 0 & \text{if } i > j \\ \min\limits_{i \le r \le j} \{c[i,r-1] + c[r+1,j] + P_{ij}\} & \text{otherwise} \end{cases}$$



c[i,j] must be processed after c[i,r-1] and c[r+1,j]

$$c[i,j] = \begin{cases} 0 & \text{if } i > j \\ \min_{i \le r \le j} \{c[i, r-1] + c[r+1, j] + P_{ij}\} & \text{otherwise} \end{cases}$$



If the entries c[i,j] are computed in the shown order, then c[i,r-1] and c[r+1,j] values are guaranteed to be computed before c[i,j].

Computing the Optimal BST Cost

COMPUTE-OPTIMAL-BST-COST (p, n)

```
for i ← 1 to n+1 do
    c[i, i-1] ← 0
PS[1] ← p[1]    // PS[i]: prefix_sum(i): Sum of all p[j] values for j ≤ i
for i ← 2 to n do
    PS[i] ← p[i] + PS[i-1]   // compute the prefix sum
for d ← 0 to n−1 do
    for  i ← 1 to n – d do
        j ← i + d
        c[i, j] ← ∞
        for r ← i to j do
            c[i, j] ← min{c[i, j], c[i,r-1] + c[r+1, j] + PS[j] – PS[i-1]}
return c[1, n]
```

Note on Prefix Sum

- We need P_{ij} values for each i, j ($1 \le i \le n$ and $1 \le j \le n$),

  where: $$P_{ij} = \sum_{h=i}^{j} p_h$$

- If we compute the summation directly for every (i, j) pair, the total runtime would be $\Theta(n^3)$.

- Instead, we spend $O(n)$ time in preprocessing to compute the prefix sum array PS. Then we can compute each P_{ij} in $O(1)$ time using PS.

Note on Prefix Sum

In preprocessing, compute for each i:

PS[i]: the sum of p[j] values for $1 \leq j \leq i$

Then, we can compute P_{ij} in $O(1)$ time as follows:

$$P_{ij} = PS[i] - PS[j-1]$$

Example:

	1	2	3	4	5	6	7	8
p:	0.05	0.02	0.06	0.07	0.20	0.05	0.08	0.02
PS:	0.05	0.07	0.13	0.20	0.40	0.45	0.53	0.55

$P_{27} = PS[7] - PS[1] = 0.53 - 0.05 = 0.48$

$P_{36} = PS[6] - PS[2] = 0.45 - 0.07 = 0.38$