

CSE 431/531: Algorithm Analysis and Design (Fall 2024)

# Dynamic Programming

Lecturer: Kelin Luo

*Department of Computer Science and Engineering*  
*University at Buffalo*

# Outline

- 1 Optimum Binary Search Tree
- 2 Summary
- 3 Summary of Studies Until Oct 30

# Optimum Binary Search Tree

**Def.** Binary search tree (BST), also called an ordered or sorted binary tree, is a rooted binary tree data structure with the key of each internal node being greater than all the keys in the respective node's left subtree and less than the ones in its right subtree.

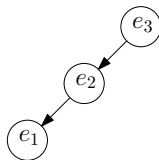
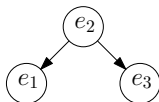
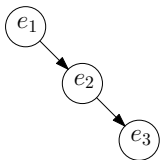
# Optimum Binary Search Tree

- $n$  elements  $e_1 < e_2 < e_3 < \dots < e_n$
- $e_i$  has frequency  $f_i$
- goal: build a binary search tree for  $\{e_1, e_2, \dots, e_n\}$  with the minimum accessing cost:

$$\sum_{i=1}^n f_i \times (\text{depth of } e_i \text{ in the tree})$$

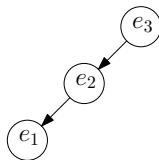
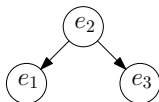
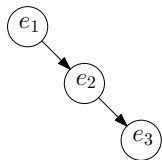
# Optimum Binary Search Tree

- Example:  $f_1 = 10, f_2 = 5, f_3 = 3$



# Optimum Binary Search Tree

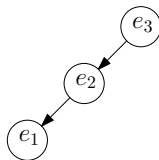
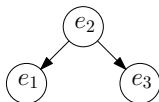
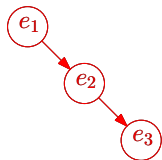
- Example:  $f_1 = 10, f_2 = 5, f_3 = 3$



- $10 \times 1 + 5 \times 2 + 3 \times 3 = 29$
- $10 \times 2 + 5 \times 1 + 3 \times 2 = 31$
- $10 \times 3 + 5 \times 2 + 3 \times 1 = 43$

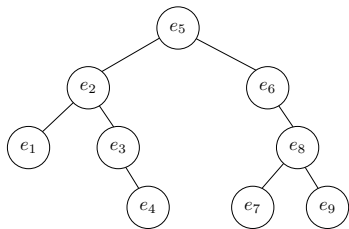
# Optimum Binary Search Tree

- Example:  $f_1 = 10, f_2 = 5, f_3 = 3$



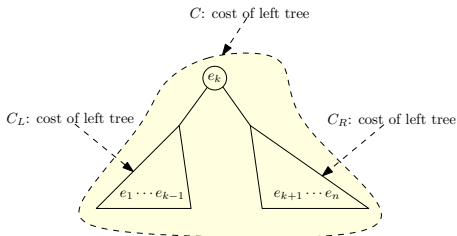
- $10 \times 1 + 5 \times 2 + 3 \times 3 = 29$
- $10 \times 2 + 5 \times 1 + 3 \times 2 = 31$
- $10 \times 3 + 5 \times 2 + 3 \times 1 = 43$

- suppose we decided to let  $e_k$  be the root
- $e_1, e_2, \dots, e_{k-1}$  are on left sub-tree
- $e_{k+1}, e_{k+2}, \dots, e_n$  are on right sub-tree
- $d_j$ : depth of  $e_j$  in our tree
- $C, C_L, C_R$ : cost of tree, left sub-tree and right sub-tree



- $d_1 = 3, d_2 = 2, d_3 = 3, d_4 = 4, d_5 = 1,$
- $d_6 = 2, d_7 = 4, d_8 = 3, d_9 = 4,$
- $C = 3f_1 + 2f_2 + 3f_3 + 4f_4 + f_5 + 2f_6 + 4f_7 + 3f_8 + 4f_9$
- $C_L = 2f_1 + f_2 + 2f_3 + 3f_4$
- $C_R = f_6 + 3f_7 + 2f_8 + 3f_9$
- $C = C_L + C_R + \sum_{j=1}^9 f_j$





$$\begin{aligned}
 C &= \sum_{\ell=1}^n f_{\ell} d_{\ell} = \sum_{\ell=1}^n f_{\ell} (d_{\ell} - 1) + \sum_{\ell=1}^n f_{\ell} \\
 &= \sum_{\ell=1}^{k-1} f_{\ell} (d_{\ell} - 1) + \sum_{\ell=k+1}^n f_{\ell} (d_{\ell} - 1) + \sum_{\ell=1}^n f_{\ell} \\
 &= C_L + C_R + \sum_{\ell=1}^n f_{\ell}
 \end{aligned}$$

$$C = C_L + C_R + \sum_{\ell=1}^n f_{\ell}$$

- In order to minimize  $C$ , need to minimize  $C_L$  and  $C_R$  respectively

$$C = C_L + C_R + \sum_{\ell=1}^n f_{\ell}$$

- In order to minimize  $C$ , need to minimize  $C_L$  and  $C_R$  respectively
- $opt[i, j]$ : the optimum cost for the instance  $(f_i, f_{i+1}, \dots, f_j)$

$$C = C_L + C_R + \sum_{\ell=1}^n f_{\ell}$$

- In order to minimize  $C$ , need to minimize  $C_L$  and  $C_R$  respectively
- $opt[i, j]$ : the optimum cost for the instance  $(f_i, f_{i+1}, \dots, f_j)$

$$opt[1, n] = (opt[1, k-1] + opt[k+1, n]) + \sum_{\ell=1}^n f_{\ell}$$

$$C = C_L + C_R + \sum_{\ell=1}^n f_{\ell}$$

- In order to minimize  $C$ , need to minimize  $C_L$  and  $C_R$  respectively
- $opt[i, j]$ : the optimum cost for the instance  $(f_i, f_{i+1}, \dots, f_j)$

$$opt[1, n] = \min_{k: 1 \leq k \leq n} (opt[1, k-1] + opt[k+1, n]) + \sum_{\ell=1}^n f_{\ell}$$

$$C = C_L + C_R + \sum_{\ell=1}^n f_{\ell}$$

- In order to minimize  $C$ , need to minimize  $C_L$  and  $C_R$  respectively
- $opt[i, j]$ : the optimum cost for the instance  $(f_i, f_{i+1}, \dots, f_j)$

$$opt[1, n] = \min_{k: 1 \leq k \leq n} (opt[1, k-1] + opt[k+1, n]) + \sum_{\ell=1}^n f_{\ell}$$

- In general,  $opt[i, j] =$

$$\begin{cases} 0 & \text{if } i = j + 1 \\ \min_{k: i \leq k \leq j} (opt[i, k-1] + opt[k+1, j]) + \sum_{\ell=i}^j f_{\ell} & \text{if } i \leq j \end{cases}$$

## Optimum Binary Search Tree

```
1:  $fsum[0] \leftarrow 0$ 
2: for  $i \leftarrow 1$  to  $n$  do  $fsum[i] \leftarrow fsum[i - 1] + f_i$ 
    $\triangleright fsum[i] = \sum_{j=1}^i f_j$ 
3: for  $i \leftarrow 0$  to  $n$  do  $opt[i + 1, i] \leftarrow 0$ 
4: for  $\ell \leftarrow 1$  to  $n$  do
5:   for  $i \leftarrow 1$  to  $n - \ell + 1$  do
6:      $j \leftarrow i + \ell - 1, opt[i, j] \leftarrow \infty$ 
7:     for  $k \leftarrow i$  to  $j$  do
8:       if  $opt[i, k - 1] + opt[k + 1, j] < opt[i, j]$  then
9:          $opt[i, j] \leftarrow opt[i, k - 1] + opt[k + 1, j]$ 
10:         $\pi[i, j] \leftarrow k$ 
11:     $opt[i, j] \leftarrow opt[i, j] + fsum[j] - fsum[i - 1]$ 
```

# Printing the Tree

## Print-Tree( $i, j$ )

```
1: if  $i > j$  then  
2:   return  
3: else  
4:   print('(')  
5:   Print-Tree( $i, \pi[i, j] - 1$ )  
6:   print( $\pi[i, j]$ )  
7:   Print-Tree( $\pi[i, j] + 1, j$ )  
8:   print('')
```



# Outline

- 1 Optimum Binary Search Tree
- 2 Summary
- 3 Summary of Studies Until Oct 30

## Dynamic Programming

- Break up a problem into many **overlapping** sub-problems
- Build solutions for larger and larger sub-problems
- Use a **table** to store solutions for sub-problems for reuse

## Comparison with greedy algorithms

- Greedy algorithm: each step is making a small progress towards constructing the solution
- Dynamic programming: the whole solution is constructed in the last step

## Comparison with divide and conquer

- Divide and conquer: an instance is broken into many **independent** sub-instances, which are solved separately.
- Dynamic programming: the sub-instances we constructed are overlapping.

# Definition of Cells for Problems We Learnt

- Weighted interval scheduling:  $opt[i] =$  value of instance defined by jobs  $\{1, 2, \dots, i\}$
- Subset sum, knapsack:  $opt[i, W'] =$  value of instance with items  $\{1, 2, \dots, i\}$  and budget  $W'$
- Longest common subsequence:  $opt[i, j] =$  value of instance defined by  $A[1..i]$  and  $B[1..j]$
- Shortest paths in DAG:  $f[v] =$  length of shortest path from  $s$  to  $v$
- Matrix chain multiplication, optimum binary search tree:  
 $opt[i, j] =$  value of instances defined by matrices  $i$  to  $j$

# Outline

- 1 Optimum Binary Search Tree
- 2 Summary
- 3 Summary of Studies Until Oct 30

# Important notations/algorithms

- Greedy algorithms: safety strategy+self reduce
  - In class Quiz 6

# Important notations/algorithms

- Greedy algorithms: safety strategy+self reduce
  - In class Quiz 6
  - HW2

# Important notations/algorithms

- Divide-and-Conquer algorithms: Divide+Conquer+Combine
  - Sorting problem: merge-sort algorithm, quick-sort algorithm (and In-Place sorting algorithm)



# Important notations/algorithms

- Divide-and-Conquer algorithms: Divide+Conquer+Combine
  - Sorting problem: merge-sort algorithm, quick-sort algorithm (and In-Place sorting algorithm)
  - Counting inversions problem: sort-and-count algorithm

# Important notations/algorithms

- Divide-and-Conquer algorithms: Divide+Conquer+Combine
  - Sorting problem: merge-sort algorithm, quick-sort algorithm (and In-Place sorting algorithm)
  - Counting inversions problem: sort-and-count algorithm
  - Selection problem: selection algorithm based on quicksort

# Important notations/algorithms

- Divide-and-Conquer algorithms: Divide+Conquer+Combine
  - Sorting problem: merge-sort algorithm, quick-sort algorithm (and In-Place sorting algorithm)
  - Counting inversions problem: sort-and-count algorithm
  - Selection problem: selection algorithm based on quicksort
  - Polynomial Multiplication problem: multiply algorithm

# Important notations/algorithms

- Divide-and-Conquer algorithms: Divide+Conquer+Combine
  - Sorting problem: merge-sort algorithm, quick-sort algorithm (and In-Place sorting algorithm)
  - Counting inversions problem: sort-and-count algorithm
  - Selection problem: selection algorithm based on quicksort
  - Polynomial Multiplication problem: multiply algorithm
  - Recurrences: recursive-tree method and Master Theorem

# Important notations/algorithms

- Divide-and-Conquer algorithms: Divide+Conquer+Combine
  - Sorting problem: merge-sort algorithm, quick-sort algorithm (and In-Place sorting algorithm)
  - Counting inversions problem: sort-and-count algorithm
  - Selection problem: selection algorithm based on quicksort
  - Polynomial Multiplication problem: multiply algorithm
  - Recurrences: recursive-tree method and Master Theorem
  - Fibonacci number problem: power algorithm

# Important notations/algorithms

- Divide-and-Conquer algorithms: Divide+Conquer+Combine
  - Sorting problem: merge-sort algorithm, quick-sort algorithm (and In-Place sorting algorithm)
  - Counting inversions problem: sort-and-count algorithm
  - Selection problem: selection algorithm based on quicksort
  - Polynomial Multiplication problem: multiply algorithm
  - Recurrences: recursive-tree method and Master Theorem
  - Fibonacci number problem: power algorithm
  - Quiz 7-8

# Important notations/algorithms

- Dynamic Programming algorithms: subproblem+recurrence relation+calculate from base case
  - Weighted interval scheduling problem

# Important notations/algorithms

- Dynamic Programming algorithms: subproblem+recurrence relation+calculate from base case
  - Weighted interval scheduling problem
  - Subset Sum problem



# Important notations/algorithms

- Dynamic Programming algorithms: subproblem+recurrence relation+calculate from base case
  - Weighted interval scheduling problem
  - Subset Sum problem
  - Knapsack problem

# Important notations/algorithms

- Dynamic Programming algorithms: subproblem+recurrence relation+calculate from base case
  - Weighted interval scheduling problem
  - Subset Sum problem
  - Knapsack problem
  - Longest common subsequence problem (LCS)

# Important notations/algorithms

- Dynamic Programming algorithms: subproblem+recurrence relation+calculate from base case
  - Weighted interval scheduling problem
  - Subset Sum problem
  - Knapsack problem
  - Longest common subsequence problem (LCS)
  - Edit distance with insertions and deletions problem

# Important notations/algorithms

- Dynamic Programming algorithms: subproblem+recurrence relation+calculate from base case
  - Weighted interval scheduling problem
  - Subset Sum problem
  - Knapsack problem
  - Longest common subsequence problem (LCS)
  - Edit distance with insertions and deletions problem
  - Edit distance with insertions, deletions and replacing problem

# Important notations/algorithms

- Dynamic Programming algorithms: subproblem+recurrence relation+calculate from base case
  - Weighted interval scheduling problem
  - Subset Sum problem
  - Knapsack problem
  - Longest common subsequence problem (LCS)
  - Edit distance with insertions and deletions problem
  - Edit distance with insertions, deletions and replacing problem
  - Shortest Path in Directed Acyclic Graph (DAG)

# Important notations/algorithms

- Dynamic Programming algorithms: subproblem+recurrence relation+calculate from base case
  - Weighted interval scheduling problem
  - Subset Sum problem
  - Knapsack problem
  - Longest common subsequence problem (LCS)
  - Edit distance with insertions and deletions problem
  - Edit distance with insertions, deletions and replacing problem
  - Shortest Path in Directed Acyclic Graph (DAG)
  - Matrix Chain Multiplication problem

# Important notations/algorithms

- Dynamic Programming algorithms: subproblem+recurrence relation+calculate from base case
  - Weighted interval scheduling problem
  - Subset Sum problem
  - Knapsack problem
  - Longest common subsequence problem (LCS)
  - Edit distance with insertions and deletions problem
  - Edit distance with insertions, deletions and replacing problem
  - Shortest Path in Directed Acyclic Graph (DAG)
  - Matrix Chain Multiplication problem
  - Optimum Binary Search Tree Problem

# Important notations/algorithms

- Dynamic Programming algorithms: subproblem+recurrence relation+calculate from base case
  - Weighted interval scheduling problem
  - Subset Sum problem
  - Knapsack problem
  - Longest common subsequence problem (LCS)
  - Edit distance with insertions and deletions problem
  - Edit distance with insertions, deletions and replacing problem
  - Shortest Path in Directed Acyclic Graph (DAG)
  - Matrix Chain Multiplication problem
  - Optimum Binary Search Tree Problem
  - Quiz 9