Optimum Binary Search Tree

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

$$\sum_{i=1}^{n} f_i \times \text{(depth of } e_i \text{ in the tree)}$$
**Example:** \( f_1 = 10, f_2 = 5, f_3 = 3 \)
Optimum Binary Search Tree

Example: \( f_1 = 10, f_2 = 5, f_3 = 3 \)

\[
\begin{align*}
10 \times 1 + 5 \times 2 + 3 \times 3 &= 29 \\
10 \times 2 + 5 \times 1 + 3 \times 2 &= 31 \\
10 \times 3 + 5 \times 2 + 3 \times 1 &= 43
\end{align*}
\]
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
- $e_1, e_2, \cdots, e_{k-1}$ are on left sub-tree
- $e_{k+1}, e_{k+2}, \cdots, e_n$ 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

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

$= \sum_{\ell=1}^{k-1} f_{\ell}(d_{\ell} - 1) + \sum_{\ell=k+1}^{n} f_{\ell}(d_{\ell} - 1) + \sum_{\ell=1}^{n} f_{\ell}$

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

- 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 \((f_i, f_{i+1}, \cdots, f_j)\)
\[ C = C_L + C_R + \sum_{\ell=1}^{n} f_{\ell} \]

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

\[
\text{opt}[1, n] = \quad (\text{opt}[1, k - 1] + \text{opt}[k + 1, n]) + \sum_{\ell=1}^{n} f_{\ell}
\]
\[ C = C_L + C_R + \sum_{\ell=1}^{n} f_{\ell} \]

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

\[
\begin{align*}
    opt[1, n] &= \min_{k:1 \leq k \leq n} \left( opt[1, k - 1] + opt[k + 1, n] \right) + \sum_{\ell=1}^{n} f_{\ell}
\end{align*}
\]
\[ C = C_L + C_R + \sum_{\ell=1}^{n} f_{\ell} \]

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

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

- 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: \begin{align*}
& \textbf{for } i \leftarrow 1 \text{ to } n \textbf{ do } fsum[i] \leftarrow fsum[i - 1] + f_i \\
& \quad \triangleright fsum[i] = \sum_{j=1}^{i} f_j
\end{align*}

3: \begin{align*}
& \textbf{for } i \leftarrow 0 \text{ to } n \textbf{ do } opt[i+1, i] \leftarrow 0
\end{align*}

4: \begin{align*}
& \textbf{for } \ell \leftarrow 1 \text{ to } n \textbf{ do }
\end{align*}

5: \begin{align*}
& \quad \textbf{for } i \leftarrow 1 \text{ to } n - \ell + 1 \textbf{ do }
\end{align*}

6: \begin{align*}
& \quad \quad j \leftarrow i + \ell - 1, \text{ opt}[i, j] \leftarrow \infty
\end{align*}

7: \begin{align*}
& \quad \textbf{for } k \leftarrow i \text{ to } j \textbf{ do }
\end{align*}

8: \begin{align*}
& \quad \quad \textbf{if } \text{ opt}[i, k - 1] + \text{ opt}[k + 1, j] < \text{ opt}[i, j] \textbf{ then }
\end{align*}

9: \begin{align*}
& \quad \quad \quad \text{ opt}[i, j] \leftarrow \text{ opt}[i, k - 1] + \text{ opt}[k + 1, j]
\end{align*}

10: \quad \pi[i, j] \leftarrow k

11: \quad \text{ opt}[i, j] \leftarrow \text{ 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. Weighted Interval Scheduling
2. Subset Sum Problem
3. Knapsack Problem
4. Longest Common Subsequence
   - Longest Common Subsequence in Linear Space
5. Shortest Paths in Directed Acyclic Graphs
6. Matrix Chain Multiplication
7. Optimum Binary Search Tree
8. Summary
9. Summary of Studies Until Nov 1st
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: \( \text{opt}[i] = \text{value of instance defined by jobs } \{1, 2, \cdots, i\} \)

- Subset sum, knapsack: \( \text{opt}[i, W'] = \text{value of instance with items } \{1, 2, \cdots, i\} \text{ and budget } W' \)

- Longest common subsequence: \( \text{opt}[i, j] = \text{value of instance defined by } A[1..i] \text{ and } B[1..j] \)

- Shortest paths in DAG: \( f[v] = \text{length of shortest path from } s \text{ to } v \)

- Matrix chain multiplication, optimum binary search tree: \( \text{opt}[i, j] = \text{value of instances defined by matrices } i \text{ to } j \)
Outline

1. Weighted Interval Scheduling
2. Subset Sum Problem
3. Knapsack Problem
4. Longest Common Subsequence
   - Longest Common Subsequence in Linear Space
5. Shortest Paths in Directed Acyclic Graphs
6. Matrix Chain Multiplication
7. Optimum Binary Search Tree
8. Summary
9. Summary of Studies Until Nov 1st
Important notations/algorithms

**Introduction:**
- Asymptotic analysis: $O$, $\Omega$, $\Theta$, compare the orders
Important notations/algorithms

- **Introduction:**
  - Asymptotic analysis: $O$, $\Omega$, $\Theta$, compare the orders
  - Polynomial time (efficient algorithm), exponential time
Important notations/algorithms

- Introduction:
  - Asymptotic analysis: $O$, $\Omega$, $\Theta$, compare the orders
  - Polynomial time (efficient algorithm), exponential time

- Graph Basics:
Important notations/algorithms

- **Introduction:**
  - Asymptotic analysis: $O$, $Ω$, $Θ$, compare the orders
  - Polynomial time (efficient algorithm), exponential time
- **Graph Basics:**
  - Undirected graph, directed graph
Important notations/algorithms

- **Introduction:**
  - Asymptotic analysis: $O$, $\Omega$, $\Theta$, compare the orders
  - Polynomial time (efficient algorithm), exponential time

- **Graph Basics:**
  - Undirected graph, directed graph
  - Two representations: adjacency matrix, linked lists
Important notations/algorithms

- **Introduction:**
  - Asymptotic analysis: $O$, $\Omega$, $\Theta$, compare the orders
  - Polynomial time (efficient algorithm), exponential time

- **Graph Basics:**
  - Undirected graph, directed graph
  - Two representations: adjacency matrix, linked lists
  - Path, cycle, tree, directed acyclic graph, bipartite graph
Introduction:
- Asymptotic analysis: $O$, $\Omega$, $\Theta$, compare the orders
- Polynomial time (efficient algorithm), exponential time

Graph Basics:
- Undirected graph, directed graph
- Two representations: adjacency matrix, linked lists
- Path, cycle, tree, directed acyclic graph, bipartite graph
- Connectivity problem: BFS and DFS algorithm
Important notations/algorithms

- **Introduction:**
  - Asymptotic analysis: $O$, $\Omega$, $\Theta$, compare the orders
  - Polynomial time (efficient algorithm), exponential time

- **Graph Basics:**
  - Undirected graph, directed graph
  - Two representations: adjacency matrix, linked lists
  - Path, cycle, tree, directed acyclic graph, bipartite graph
  - Connectivity problem: BFS and DFS algorithm
  - Testing Bipartiteness problem: test-bipartiteness-BFS or test-bipartiteness-DFS algorithm
Important notations/algorithms

Introduction:
- Asymptotic analysis: $O$, $\Omega$, $\Theta$, compare the orders
- Polynomial time (efficient algorithm), exponential time

Graph Basics:
- Undirected graph, directed graph
- Two representations: adjacency matrix, linked lists
- Path, cycle, tree, directed acyclic graph, bipartite graph
- Connectivity problem: BFS and DFS algorithm
- Testing Bipartiteness problem: test-bipartiteness-BFS or test-bipartiteness-DFS algorithm
- Topological Ordering problem: topological-sort algorithm
Important notations/algorithms

- Greedy algorithms: safety strategy + self reduce
- Box Packing problem: greedy algorithm
Important notations/algorithms

- Greedy algorithms: safety strategy + self reduce
  - Box Packing problem: greedy algorithm
  - Interval Scheduling problem: schedule algorithm
Important notations/algorithms

- Greedy algorithms: safety strategy + self reduce
  - Box Packing problem: greedy algorithm
  - Interval Scheduling problem: schedule algorithm
  - Interval Partitioning problem: partition algorithm
Greedy algorithms: safety strategy + self reduce
- Box Packing problem: greedy algorithm
- Interval Scheduling problem: schedule algorithm
- Interval Partitioning problem: partition algorithm
- Offline Caching problem: FIF algorithm
Important notations/algorithms

- Greedy algorithms: safety strategy + self reduce
  - Box Packing problem: greedy algorithm
  - Interval Scheduling problem: schedule algorithm
  - Interval Partitioning problem: partition algorithm
  - Offline Caching problem: FIF algorithm
  - Priority Queue: heap
Important notations/algorithms

- Greedy algorithms: safety strategy + self reduce
  - Box Packing problem: greedy algorithm
  - Interval Scheduling problem: schedule algorithm
  - Interval Partitioning problem: partition algorithm
  - Offline Caching problem: FIF algorithm
  - Priority Queue: heap
  - Huffman Code problem: prefix code notation, Huffman algorithm
Greedy algorithms: safety strategy+self reduce

- Box Packing problem: greedy algorithm
- Interval Scheduling problem: schedule algorithm
- Interval Partitioning problem: partition algorithm
- Offline Caching problem: FIF algorithm
- Priority Queue: heap
- Huffman Code problem: prefix code notation, Huffman algorithm
- Exercise problems: Fractional knapsack problem, scheduling problem (min weighted sum of completion times)
Important notations/algorithms

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

Exercise problems: Closest Pair, Convex Hull, Two Matrix Multiplication
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

Exercise problems: Closest Pair, Convex Hull, Two Matrix Multiplication
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
  - Exercise problems: Closest Pair, Convex Hull, Two Matrix Multiplication
Important notations/algorithms

- Dynamic Programming algorithms: subproblem + recurrence relation + calculate from base case
- Weighted interval scheduling problem: DP algorithm + Recovering optimal schedule
Important notations/algorithms

- Dynamic Programming algorithms: subproblem + recurrence relation + calculate from base case
- Weighted interval scheduling problem: DP algorithm + Recovering optimal schedule
- Subset Sum problem: DP algorithm + Recovering optimal schedule
- Knapsack problem: DP algorithm + Recovering optimal schedule
- Longest common subsequence problem (LCS): DP algorithm + Recovering optimal schedule
- Edit distance with insertions and deletions problem: apply algorithm for LCS problem
- Edit distance with insertions, deletions and replacing problem
- Shortest Path in Directed Acyclic Graph (DAG): Shortest Paths in DAG algorithm + print-path algorithm
- Matrix Chain Multiplication problem: matrix-chain-multiplication algorithm + print-optimal-order alg
- Optimum Binary Search Tree Problem: Optimum Binary Search Tree alg + Print Tree alg
Important notations/algorithms

- Dynamic Programming algorithms: subproblem + recurrence relation + calculate from base case
- Weighted interval scheduling problem: DP algorithm + Recovering optimal schedule
- Subset Sum problem: DP algorithm + Recovering optimal schedule
- Knapsack problem: DP algorithm + Recovering optimal schedule
- Longest common subsequence problem (LCS): DP algorithm + Recovering optimal schedule
- Edit distance with insertions and deletions problem: apply algorithm for LCS problem
- Edit distance with insertions, deletions and replacing problem
- Shortest Path in Directed Acyclic Graph (DAG): Shortest Paths in DAG algorithm + print-path algorithm
- Matrix Chain Multiplication problem: matrix-chain-multiplication algorithm + print-optimal-order alg
- Optimum Binary Search Tree Problem: Optimum Binary Search Tree alg + Print Tree alg
**Important notations/algorithms**

- Dynamic Programming algorithms: subproblem + recurrence relation + calculate from base case
- Weighted interval scheduling problem: DP algorithm + Recovering optimal schedule
- Subset Sum problem: DP algorithm + Recovering optimal schedule
- Knapsack problem: DP algorithm + Recovering optimal schedule
- Longest common subsequence problem (LCS): DP algorithm + Recovering optimal schedule
- Edit distance with insertions and deletions problem: apply algorithm for LCS problem
- Edit distance with insertions, deletions and replacing problem
- Shortest Path in Directed Acyclic Graph (DAG): Shortest Paths in DAG algorithm + print-path algorithm
- Matrix Chain Multiplication problem: matrix-chain-multiplication algorithm + print-optimal-order alg
- Optimum Binary Search Tree Problem: Optimum Binary Search Tree alg + Print Tree alg
Important notations/algorithms

- Dynamic Programming algorithms: subproblem + recurrence relation + calculate from base case
- Weighted interval scheduling problem: DP algorithm + Recovering optimal schedule
- Subset Sum problem: DP algorithm + Recovering optimal schedule
- Knapsack problem: DP algorithm + Recovering optimal schedule
- Longest common subsequence problem (LCS): DP algorithm + Recovering optimal schedule
- Edit distance with insertions and deletions problem: apply algorithm for LCS problem
- Shortest Path in Directed Acyclic Graph (DAG): Shortest Paths in DAG algorithm + print-path algorithm
- Matrix Chain Multiplication problem: matrix-chain-multiplication algorithm + print-optimal-order alg
- Optimum Binary Search Tree Problem: Optimum Binary Search Tree alg + Print Tree alg
Important notations/algorithms

- Dynamic Programming algorithms: subproblem + recurrence relation + calculate from base case
- Weighted interval scheduling problem: DP algorithm + Recovering optimal schedule
- Subset Sum problem: DP algorithm + Recovering optimal schedule
- Knapsack problem: DP algorithm + Recovering optimal schedule
- Longest common subsequence problem (LCS): DP algorithm + Recovering optimal schedule
- Edit distance with insertions and deletions problem: apply algorithm for LCS problem
- Edit distance with insertions, deletions and replacing problem
Dynamic Programming algorithms: subproblem + recurrence relation + calculate from base case
Weighted interval scheduling problem: DP algorithm + Recovering optimal schedule
Subset Sum problem: DP algorithm + Recovering optimal schedule
Knapsack problem: DP algorithm + Recovering optimal schedule
Longest common subsequence problem (LCS): DP algorithm + Recovering optimal schedule
Edit distance with insertions and deletions problem: apply algorithm for LCS problem
Edit distance with insertions, deletions and replacing problem
Shortest Path in Directed Acyclic Graph (DAG): Shortest Paths in DAG algorithm + print-path algorithm
Important notations/algorithms

- Dynamic Programming algorithms: subproblem + recurrence relation + calculate from base case
- Weighted interval scheduling problem: DP algorithm + Recovering optimal schedule
- Subset Sum problem: DP algorithm + Recovering optimal schedule
- Knapsack problem: DP algorithm + Recovering optimal schedule
- Longest common subsequence problem (LCS): DP algorithm + Recovering optimal schedule
- Edit distance with insertions and deletions problem: apply algorithm for LCS problem
- Edit distance with insertions, deletions and replacing problem
- Shortest Path in Directed Acyclic Graph (DAG): Shortest Paths in DAG algorithm + print-path algorithm
- Matrix Chain Multiplication problem: matrix-chain-multiplication algorithm + print-optimal-order alg
Dynamic Programming algorithms: subproblem + recurrence relation + calculate from base case
- Weighted interval scheduling problem: DP algorithm + Recovering optimal schedule
- Subset Sum problem: DP algorithm + Recovering optimal schedule
- Knapsack problem: DP algorithm + Recovering optimal schedule
- Longest common subsequence problem (LCS): DP algorithm + Recovering optimal schedule
- Edit distance with insertions and deletions problem: apply algorithm for LCS problem
- Edit distance with insertions, deletions and replacing problem
- Shortest Path in Directed Acyclic Graph (DAG): Shortest Paths in DAG algorithm + print-path algorithm
- Matrix Chain Multiplication problem: matrix-chain-multiplication algorithm + print-optimal-order alg
- Optimum Binary Search Tree Problem: Optimum Binary Search Tree alg + Print Tree alg
Quiz 6 about Dynamic Programming algorithms

- Fours problems about Dynamic programming algorithms