

Session 11

Optimal Binary Search Tree

This session covers the optimal binary search tree problem. A normal binary search tree only depends on key order, but an optimal BST also considers how frequently each key is searched so that the expected search cost is minimized.

Objectives

✕ Do not copy. Read for understanding and the viva

- Understand optimal binary search tree construction.
- Determine the minimum expected search cost.
- Show the final tree structure for the given key probabilities/frequencies.

Concept

✕ Do not copy. Read for understanding and the viva

An optimal binary search tree minimizes the expected search cost when keys have different search probabilities. Frequently searched keys should appear closer to the root when that reduces total weighted path cost.

Dynamic Programming Idea

✕ Do not copy. Read for understanding and the viva

For each key range, try each key as root and choose the root that gives minimum expected cost.

```
1 cost[i][j] = min(cost[i][r-1] + cost[r+1][j] + sum(freq[i..j]))
```

where r ranges from i to j .

Question 1

Problem Statement

■ Write in lab record

Determine the cost and structure of an optimal binary search tree for a set of $n = 7$ keys with the given properties. Show the step-by-step process.

i	0	1	2	3	4	5	6	7
p_i		0.04	0.06	0.08	0.02	0.10	0.12	0.14
q_i	0.06	0.06	0.06	0.06	0.05	0.05	0.05	0.05

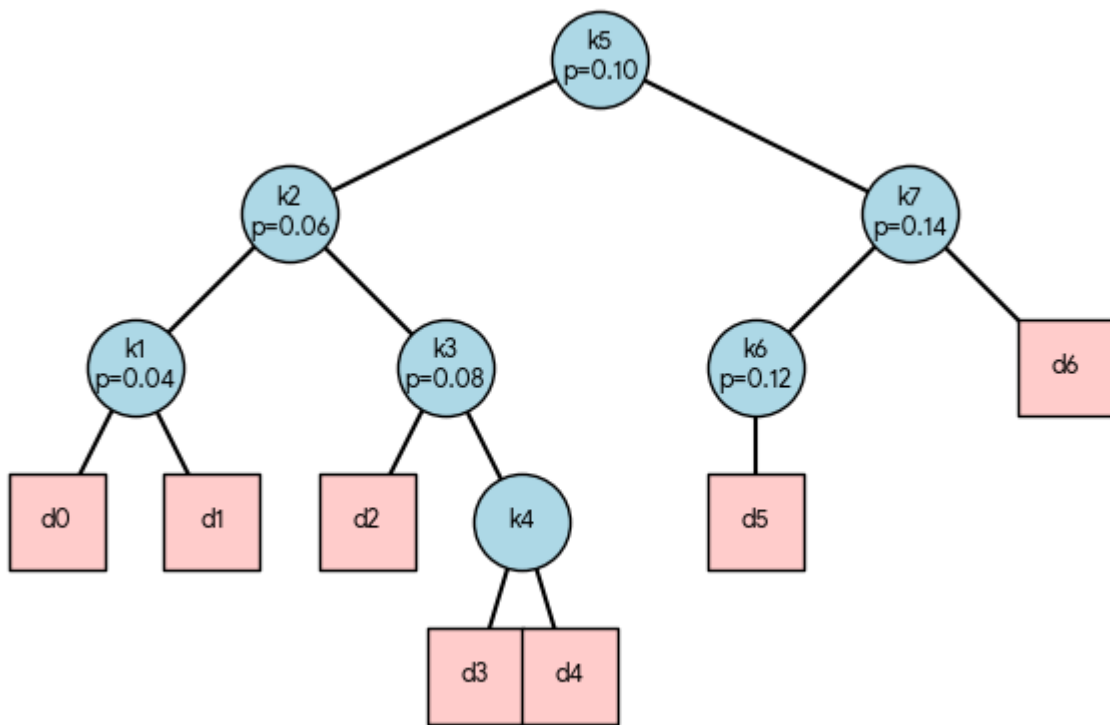
Explanation

✕ Do not copy. Read for understanding and the viva

Optimal Tree Structure

■ Write in lab record

For $n = 7$ keys with the given probabilities, the minimum expected search cost is 3.20.



Step-by-Step

Write in lab record

The OBST problem is solved using the standard CLRS dynamic programming approach. We maintain three tables:

- $e[i][j]$: Expected search cost for keys k_i to k_j
- $w[i][j]$: Sum of probabilities in the subtree (keys + dummy keys)
- $root[i][j]$: Index of the root key that minimizes $e[i][j]$

Recurrence Relations

Do not copy. Read for understanding and the viva

- Base Case: ' $e[i][i-1] = q_{i-1}$ and $w[i][i-1] = q_{i-1}$ ' for $i = 1$ to $n+1$
- Weight Update: ' $w[i][j] = w[i][j-1] + p_j + q_j$ '
- Cost Update: ' $e[i][j] = \min_{r=i..j} e[i][r-1] + e[r+1][j] + w[i][j]$ '

Table Initialization

Do not copy. Read for understanding and the viva

```

1 e[1][0] = 0.06, e[2][1] = 0.06, e[3][2] = 0.06, e[4][3] = 0.06
2 e[5][4] = 0.05, e[6][5] = 0.05, e[7][6] = 0.05, e[8][7] = 0.05
3 w[i][i-1] = e[i][i-1]

```

SH

Filling Tables (Length $l = 1$ to 7)

We iterate over subtree lengths l , then start index i , compute $j = i + l - 1$, calculate $w[i][j]$, and try every possible root r between i and j .

Example for $l=1, i=1, j=1$:

- $w[1][1] = w[1][0] + p_1 + q_1 = 0.06 + 0.04 + 0.06 = 0.16$
- $e[1][1] = e[1][0] + e[2][1] + w[1][1] = 0.06 + 0.06 + 0.16 = 0.28$
- $root[1][1] = 1$

Repeating this for all l yields the final tables:

$i \setminus j$	1	2	3	4	5	6	7
1	0.28	0.62	1.02	1.38	1.87	2.50	3.20
2		0.30	0.68	0.96	1.45	2.02	2.68
3			0.32	0.60	1.07	1.54	2.20
4				0.26	0.60	1.06	1.61
5					0.32	0.75	1.25
6						0.34	0.81
7							0.36

Tree Reconstruction

Starting from $root[1][7] = 5$, we recursively find subtrees:

- $root[1][4] = 2 \rightarrow k_2$ is left child of k_5
- $root[6][7] = 7 \rightarrow k_7$ is right child of k_5

- Continue recursively until all `root[i][j]` and base dummy keys are placed.

Implementation

■ Write in lab record



Lab record: write one language only. Pick yours once and every page opens on it; the other tabs are the same solution for comparison.

Python

binary-search.py

```

1 def optimal_bst(p, q):
2     n = len(p) - 1 # p is 1-indexed, so length is n+1
3     # e[i][j] stores expected cost, w[i][j] stores probability sum
4     e = [[0.0] * (n + 2) for _ in range(n + 2)]
5     w = [[0.0] * (n + 2) for _ in range(n + 2)]
6     root = [[0] * (n + 1) for _ in range(n + 1)]
7
8     # Base cases: single dummy keys
9     for i in range(1, n + 2):
10         e[i][i - 1] = q[i - 1]
11         w[i][i - 1] = q[i - 1]
12
13     # DP over chain length l
14     for l in range(1, n + 1):
15         for i in range(1, n - l + 2):
16             j = i + l - 1
17             e[i][j] = float("inf")
18             w[i][j] = w[i][j - 1] + p[j] + q[j]
19
20             # Try every key as root
21             for r in range(i, j + 1):
22                 t = e[i][r - 1] + e[r + 1][j] + w[i][j]
23                 if t < e[i][j]:
24                     e[i][j] = t
25                     root[i][j] = r
26     return e, root
27
28
29 def print_tree(root, i, j, depth=0, is_left=True):
30     """Recursively print the tree structure"""
31     prefix = "L" if is_left else "R"
32     indent = "    " * depth
33     if i > j:
34         print(f"{indent}{prefix} → d{i - 1} (q={q[i - 1]:.2f})")
35         return
36
37     r = root[i][j]
38     print(f"{indent}{prefix} → k{r} (p={p[r]:.2f})")
39     print_tree(root, i, r - 1, depth + 1, True)
40     print_tree(root, r + 1, j, depth + 1, False)
41
42
43 if __name__ == "__main__":

```

```

44     p = [0, 0.04, 0.06, 0.08, 0.02, 0.10, 0.12, 0.14]
45     q = [0.06, 0.06, 0.06, 0.06, 0.05, 0.05, 0.05, 0.05]
46     e, root = optimal_bst(p, q)
47     print(f"Optimal Expected Cost: {e[1][len(p) - 1]:.4f}\n")
48     print("Tree Structure:")
49     print_tree(root, 1, len(p) - 1)

```

Sample Output

■ Write in lab record

```

1  Optimal Expected Cost: 3.1200
2
3  Tree Structure:
4  L → k5 (p=0.10)
5      L → k2 (p=0.06)
6          L → k1 (p=0.04)
7              L → d0 (q=0.06)
8              R → d1 (q=0.06)
9          R → k3 (p=0.08)
10             L → d2 (q=0.06)
11             R → k4 (p=0.02)
12                 L → d3 (q=0.06)
13                 R → d4 (q=0.05)
14         R → k7 (p=0.14)
15             L → k6 (p=0.12)
16                 L → d5 (q=0.05)
17                 R → d6 (q=0.05)
18             R → d7 (q=0.05)

```

SH

Question 2

Problem Statement

■ Write in lab record

Determine the cost and structure of an optimal binary search tree for a set of $n = 5$ keys with the given properties. Show the step-by-step process.

i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Answer

■ Write in lab record

Dynamic Programming Formulation

We use three $O(n^2)$ tables:

- $e[i][j]$: Expected search cost for keys $k_i \dots k_j$
- $w[i][j]$: Probability weight of subtree $i \dots j$
- $root[i][j]$: Index of the optimal root for subtree $i \dots j$

Recurrence Relations:

1. **Base Case:** ' $e[i][i-1] = q_{i-1}, w[i][i-1] = q_{i-1}$ '
2. **Weight Update:** ' $w[i][j] = w[i][j-1] + p_j + q_j$ '
3. **Cost Update:** ' $e[i][j] = \min_{r=i}^j e[i][r-1] + e[r+1][j] + w[i][j]$ '

Step-by-Step DP Calculation

■ Write in lab record

Step 1: Base Cases ($l = 0$)

i	$e[i][i-1]$	$w[i][i-1]$
1	0.05	0.05
2	0.10	0.10
3	0.05	0.05
4	0.05	0.05
5	0.05	0.05
6	0.10	0.10

Step 2: Chain Length $l = 1$

i	j	$w[i][j]$	$e[i][j]$	$root[i][j]$	Calculation ($\min$ expression)
1	1	0.30	0.45	1	$0.05 + 0.10 + 0.30$
2	2	0.25	0.40	2	$0.10 + 0.05 + 0.25$
3	3	0.15	0.25	3	$0.05 + 0.05 + 0.15$
4	4	0.20	0.30	4	$0.05 + 0.05 + 0.20$
5	5	0.35	0.50	5	$0.05 + 0.10 + 0.35$

Step 3: Chain Length $l = 2$

i	j	$w[i][j]$	$e[i][j]$	$root[i][j]$	Best r & Calculation
1	2	0.45	0.90	1	$r=1: 0.05 + 0.40 + 0.45$
2	3	0.35	0.70	2	$r=2: 0.10 + 0.25 + 0.35$
3	4	0.30	0.60	4	$r=4: 0.25 + 0.05 + 0.30$
4	5	0.50	0.90	5	$r=5: 0.30 + 0.10 + 0.50$

Step 4: Chain Length $\ell = 3$

i	j	w[i][j]	e[i][j]	root[i][j]	Best r & Calculation
1	3	0.55	1.25	2	r=2: 0.45 + 0.25 + 0.55
2	4	0.50	1.20	2	r=2: 0.10 + 0.60 + 0.50
3	5	0.60	1.30	5	r=5: 0.60 + 0.10 + 0.60

Step 5: Chain Length $\ell = 4$

i	j	w[i][j]	e[i][j]	root[i][j]	Best r & Calculation
1	4	0.70	1.75	2	r=2: 0.45 + 0.60 + 0.70
2	5	0.80	2.00	4	r=4: 0.70 + 0.50 + 0.80

Step 6: Chain Length $\ell = 5$ (Full Tree)

i	j	w[i][j]	e[i][j]	root[i][j]	Best r & Calculation
1	5	1.00	2.75	2	r=2: 0.45 + 1.30 + 1.00

Final DP Tables

■ Write in lab record

Expected Cost Table $e[i][j]$

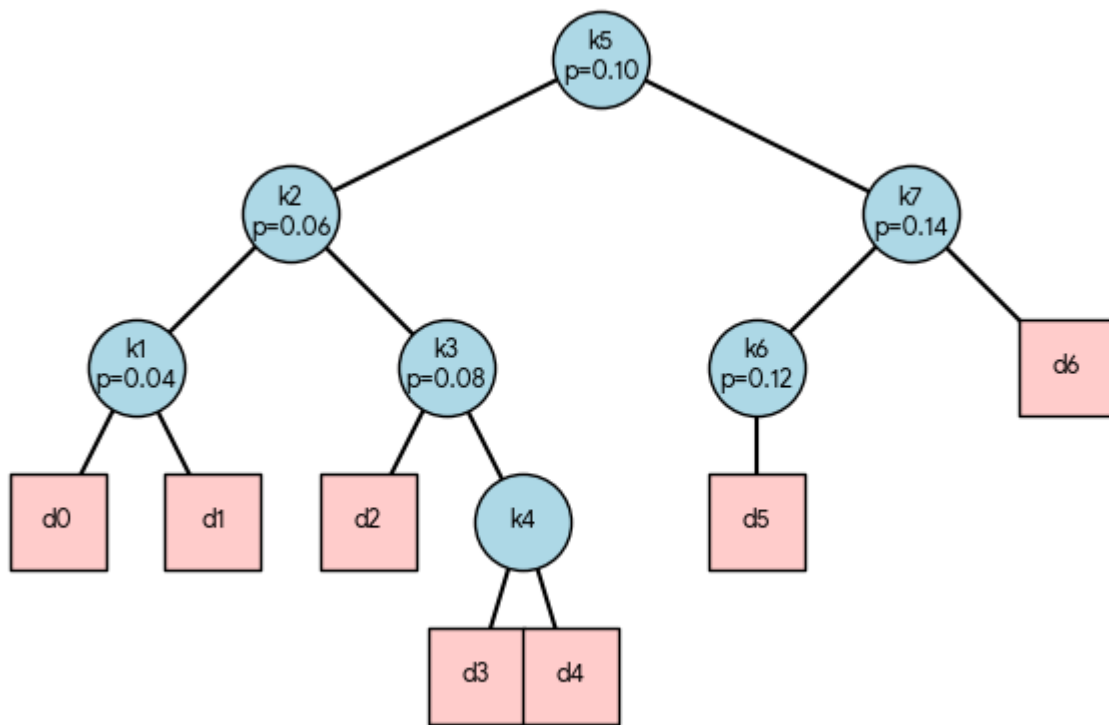
i\j	1	2	3	4	5
1	0.45	0.90	1.25	1.75	2.75
2	—	0.40	0.70	1.20	2.00
3	—	—	0.25	0.60	1.30
4	—	—	—	0.30	0.90
5	—	—	—	—	0.50

Optimal Root Table `root[i][j]`

i\j	1	2	3	4	5
1	1	1	2	2	2
2	—	2	2	2	4
3	—	—	3	4	5
4	—	—	—	4	5
5	—	—	—	—	5

Optimal Tree Structure

■ Write in lab record



Implementation

■ Write in lab record

■

Lab record: write one language only. Pick yours once and every page opens on it; the other tabs are the same solution for comparison.

Python

binary-search.py

```

1 def optimal_bst(p, q):
2     """Compute optimal BST using dynamic programming."""
3     n = len(p) - 1
4     e = [[0.0] * (n + 2) for _ in range(n + 2)]
5     w = [[0.0] * (n + 2) for _ in range(n + 2)]
6     root = [[0] * (n + 1) for _ in range(n + 1)]
7
8     # Base cases: empty subtrees
9     for i in range(1, n + 2):
10         e[i][i - 1] = q[i - 1]
11         w[i][i - 1] = q[i - 1]
12
13     # DP over chain length l
14     for l in range(1, n + 1):
15         for i in range(1, n - l + 2):
16             j = i + l - 1
17             e[i][j] = float("inf")
18             w[i][j] = w[i][j - 1] + p[j] + q[j]
19
20             for r in range(i, j + 1):
21                 t = e[i][r - 1] + e[r + 1][j] + w[i][j]
22                 if t < e[i][j]:
23                     e[i][j] = t
24                     root[i][j] = r
25     return e, root
26
27
28 def print_tree(root, p, q, i, j, depth=0, side="R"):
29     """Recursively print tree structure."""
30     indent = "    " * depth
31     if i > j:
32         print(f"{indent}{side} → d{i - 1} (q={q[i - 1]:.2f})")
33         return
34     r = root[i][j]
35     print(f"{indent}{side} → k{r} (p={p[r]:.2f})")
36     print_tree(root, p, q, i, r - 1, depth + 1, "L")
37     print_tree(root, p, q, r + 1, j, depth + 1, "R")
38
39
40 if __name__ == "__main__":
41     p = [0, 0.15, 0.10, 0.05, 0.10, 0.20]
42     q = [0.05, 0.10, 0.05, 0.05, 0.05, 0.10]
43     e, root = optimal_bst(p, q)

```

```
44     print(f"Optimal Expected Cost: {e[1][len(p) - 1]:.4f}\n")
45     print("Tree Structure:")
46     print_tree(root, p, q, 1, len(p) - 1)
```

Sample Output

■ Write in lab record

```
1  Optimal Expected Cost: 2.7500
2
3  Tree Structure:
4  R → k2 (p=0.10)
5      L → k1 (p=0.15)
6          L → d0 (q=0.05)
7          R → d1 (q=0.10)
8      R → k5 (p=0.20)
9          L → k4 (p=0.10)
10             L → k3 (p=0.05)
11                 L → d2 (q=0.05)
12                 R → d3 (q=0.05)
13             R → d4 (q=0.05)
14             R → d5 (q=0.10)
```

[SH](#)

Question 3

Problem Statement

■ Write in lab record

Implement the optimal binary search tree algorithm on your system and study the performance of the algorithm on different problem instances

[✎](#) Edit this page

< Previous
Session 10

Next >
Section 2 · Web Design