We’ve done

- Dynamic Programming
  - Matrix Chain Multiplication
  - Longest Common Subsequence

Now

- Dynamic Programming
  - Assembly-line scheduling
  - Optimal Binary Search Trees

Next

- Shortest paths algorithms
Assembly Line Scheduling (ALS)

- A factory has two assembly lines with $n$ stations each
  - Line 1: $S_{1,1}, S_{1,2}, \ldots, S_{1,n}$
  - Line 2: $S_{2,1}, S_{2,2}, \ldots, S_{2,n}$

- Automobile chassis enter one of the lines, have parts added at $n$ stations, and exit at the end of a line

- Enter time for line $i$ is $e_i$

- Exit time for line $i$ is $x_i$

- $S_{1,j}$ and $S_{2,j}$ perform the same function (thus, a chassis goes through exactly one of $S_{1,j}$ or $S_{2,j}$)

- Time required at station $S_{i,j}$ is $a_{i,j}$

- Time required to move from $S_{i,j}$ to the other line is $t_{i,j}$

- Time required to move from $S_{i,j}$ to the next station on the same line is 0.

Find a fastest schedule to complete one auto.
The recurrence

- \( f^* \): the optimal time
- \( f_{i,j} \): the fastest time to get through \( S_{i,j} \)

Then,

\[
f^* = \min\{f[1,n] + x_1, f[2,n] + x_2\}.
\]

\[
f_{1,j} = \begin{cases} 
  e_1 + a_{1,1} & \text{if } j = 1 \\
  \min\{f_{1,j-1} + a_{1,j}, f_{2,j-1} + t_{2,j-1} + a_{1,j}\} & \text{if } j > 1
\end{cases}
\]

\[
f_{2,j} = \begin{cases} 
  e_2 + a_{2,1} & \text{if } j = 1 \\
  \min\{f_{2,j-1} + a_{2,j}, f_{1,j-1} + t_{1,j-1} + a_{2,j}\} & \text{if } j > 1
\end{cases}
\]

The rest (pseudo code and stuff)

- Straightforward, see text for details
- Running time is linear
- Space used is also linear
Binary search trees

Keys $k_1, k_2, \ldots, k_n$, and dummy keys $d_0, d_1, \ldots, d_n$.

Given an ordering:

$$d_0 < k_1 < d_1 < k_2 < d_2 < \cdots < k_n < d_n.$$ 

A BST on these keys is a tree satisfying

- For every node $v$, keys on the left are less than $v$
- For every node $v$, keys on the right are greater than $v$
- Dummy keys are leaf nodes (representing NOT FOUND)
Optimal binary search tree problem

Inputs

- \{k_1, \ldots, k_n\}
- \{d_0, \ldots, d_n\}
- d_0 < k_1 < d_1 < k_2 < d_2 < \cdots < k_n < d_n
- p_i = \text{Prob}[k_i \text{ is queried}]
- q_i = \text{Prob}[d_i \text{ is queried}]
  (i.e. the probability that a query is in between \( k_i \) and \( k_{i+1} \))

Clearly, it is necessary that

\[
\sum_{i=1}^{n} p_i + \sum_{i=0}^{n} q_i = 1.
\]

The COST of a query is the number of nodes visited.

Expected query cost

\[
= \sum_{i=1}^{n} \left( \text{DEPTH}_T(k_i) + 1 \right) \cdot p_i + \sum_{i=0}^{n} \left( \text{DEPTH}_T(d_i) + 1 \right) \cdot q_i
\]

Construct a binary search tree which minimizes expected query cost.
Step 1: Identify subproblems

• Structure of a BST:
  – A root containing $k_r$, for some $r \in \{1, \ldots, n\}$.
  – Left subtree consists of keys $k_1, \ldots, k_{r-1}$, and dummy keys $d_0, \ldots, d_{r-1}$
  – Right subtree consists of keys $k_{r+1}, \ldots, k_n$, and dummy keys $d_r, \ldots, d_n$
  – Subtle point: if $r = 1$, then the left has only $d_0$; if $r = n$, then the right has only $d_n$.

• Optimal substructure:
  – For a BST to be optimal, it is necessary that the left subtree and the right subtree are optimal (why?)

• Sub problems:
  – Given $k_i, \ldots, k_j$ ($1 \leq i \leq j + 1 \leq n + 1$) and $d_{i-1}, \ldots, d_j$ (no key $k_x$ if $i = j + 1$)
  – Construct a BST $T_{ij}$ on these keys that minimizes

$$\sum_{x=i}^{j} (\text{DEPTH}_{T_{ij}}(k_x)+1) \cdot p_x + \sum_{x=i-1}^{j} (\text{DEPTH}_{T_{ij}}(d_x)+1) \cdot q_x$$
Step 2: The recurrence relation

- Given $1 \leq i \leq j + 1 \leq n + 1$.
- Let $e[i, j]$ denote the expected query cost of an optimal BST on keys $k_i, \ldots, k_j$ and dummy keys $d_{i-1}, \ldots, d_j$.
- Define

$$w(i, j) := \sum_{x=i}^{j} p_x + \sum_{x=i-1}^{j} q_x.$$  

Noting that, if $k_r$ was the root of $T_{ij}$, then

$$e[i, j] = p_r + (e[i, r - 1] + w(i, r - 1))$$
$$+ (e[r + 1, j] + w(r + 1, j))$$
$$= e[i, r - 1] + e[r + 1, j] + w(i, j)$$

Hence,

$$e[i, j] = \begin{cases} 
q_{j-1} & \text{if } j = i - 1 \\
\min_{i \leq r \leq j} \{e[i, r - 1] + e[r + 1, j] + w(i, j)\} & \text{if } i \leq j 
\end{cases}$$
Step 3: How to fill out the table?

- This is very similar to Matrix Chain Multiplication

- Entries $e[i, j]$ are dependent on other $e[x, y]$ with $y - x < j - i$.

Let $root[i, j]$ denote the $r$ for which $k_r$ is at the root of $T_{ij}$.

We can record $root[i, j]$ while updating $e[i, j]$

The rest is similar to MCM
Step 4: Pseudo code

**OPTIMAL-BST** \((p, q, n)\)

1. for \(i = 1\) to \(n + 1\) do
2. \(e[i, i - 1] \leftarrow q_{i-1}\) // base cases
3. end for

4. for \(l = 1\) to \(n\) do
5. for \(i \leftarrow 1\) to \(n - l + 1\) do
   6. \(j \leftarrow i + l - 1\); // not really needed, just to be clearer
   7. \(e[i, j] \leftarrow \infty;\)
   8. \(w[i, j] \leftarrow w[i, j - 1] + p_j + q_j;\) // save some time
   9. for \(r \leftarrow i\) to \(j\) do
      10. \(t \leftarrow e[i, r - 1] + e[r + 1, j] + w[i, j];\)
      11. if \(e[i, j] > t\) then
          12. \(e[i, j] \leftarrow t;\)
          13. \(\text{root}[i, j] \leftarrow r;\)
      14. end if
   15. end for
  16. end for
17. end for
18. return \(e[1, n];\)
Step 5: Analysis of time and space

- Time: $\Theta(n^3)$
- Space: $\Theta(n^2)$

Constructing the tree from the table $root$ is easy.