Dynamic programming

Chain matrix multiplication

Given a sequence of matrices, determine the order of multiplication that minimize the number of operations.

Matrix multiplication is an associative but not a commutative operation. There are several choices:

\[ M_1 = [10 \times 20] \]
\[ M_2 = [20 \times 50] \]
\[ M_3 = [50 \times 1] \]
\[ M_4 = [1 \times 100] \]

Matrix multiplication is an associative but not a commutative operation. There are several choices:

\[ M_1 \cdot (M_2 \cdot (M_3 \cdot M_4)) \]
\[ (M_1 \cdot (M_2 \cdot M_3)) \cdot M_4 \]

Brute Force Approach

1) Do all possible multiplicative orders
2) Choose the optimal

What is the complexity of this approach?
What is the number of full binary trees with \( n \) leaves?

**Chain matrix multiplication**

\[ B(n) = \text{# of full binary trees with } n \text{ leaves} \]

\[ B(n) = B(1)B(n-1) + B(2)B(n-2) + \ldots + B(n-1)B(1) \]

\[ B(1) = 1 \]

\[ B(n) = C(n-1) \]

**Brute Force Approach**

This approach takes an exponential time...

\[ C_n = \frac{1}{n+1}\binom{2n}{n}, \quad n=0,1,... \]

\[ n! \approx n^n \]

\[ \binom{2n}{n} = \frac{(2n)!}{(n!)^2} \approx \frac{(2n)^{2n}}{n^{2n}} = 4^n \]

**Greedy Approach**

Repeatedly select the product that uses the fewest operations.

\[ M_1 = [10 \times 20] \]
\[ M_2 = [20 \times 50] \]
\[ M_3 = [50 \times 1] \]
\[ M_4 = [1 \times 100] \]

There are several choices:

\[ M_1^* (M_2^* (M_3^* M_4)) \]

\[ (M_1^* (M_2^* M_3))^* M_4 \]
**Dynamic Programming**

The main question in DP is, what are the subproblems?

**Matrix Multiplication**

\[ M_1 \times M_2 \times \ldots \times M_n \]

How do we define subproblems?

\[ m(i, j) = \min \text{ cost of } M_i \times M_{i+1} \times \ldots \times M_j \]

\[ m(i, i) = 0 \]

**Combining step**

These two pieces will eventually produce two matrices

\[(M_i \times M_{i+1} \times \ldots \times M_k) \times (M_{k+1} \times \ldots \times M_j)\]

It takes \(r_{i-1} \times r_k \times r_j\) multiplications to multiply two matrices.

**Chain matrix multiplication**

How would you fill out the table?

**Filling up the table**

\[ m(i, j) = \min \text{ cost of } M_i \times M_{i+1} \times \ldots \times M_j \]

\[ m(i, i) = 0, \quad i = 1, 2, \ldots, n \]

\[ m(i, i+1) = r_{i-1} \times r_i \times r_{i+1}, \quad i = 1, 2, \ldots, n-1 \]

\[ m(i, i+2) = \ldots, \quad i = 1, 2, \ldots, n-2 \]
Filling up the table

Set \( m(i,i) = 0 \) for all \( i \).

```plaintext
for(s = 1; s < n; s++)
    for(i = 1; i <= n-s, i++)
        j = i + s;
        m(i,j) = \min_k (m(i,k) + m(k+1,j) + \text{comb\_step})
```

return \( m(1,n) \);

Filling up the table

Runtime complexity

\[ m(i,j) = \min_k (m(i,k) + m(k+1,j) + \text{comb\_step}) \]

What is the complexity of this algorithm?

Table size – \( O(n^2) \)
Cost per entry – \( O(n) \)
Total – \( O(n^3) \)

Chain matrix multiplication

\[ M_1 \ast M_2 \ast M_3 \ast M_4 \]

How would you recover the optimal set of parentheses?

We have to memorize the split marker indicating the best split: this is the value \( k \).

Basic Steps of DP

1. Define subproblems.
2. Write the recurrence relation.
3. Prove that an algorithm is correct.
4. Compute its runtime complexity.
Optimal Binary Search Trees

- Given sequence \( k_1 < k_2 < \ldots < k_n \) of \( n \) sorted keys, with a search probability \( p_i \) for each key \( k_i \).
- Want to build a binary search tree (BST) with minimum expected search cost.
- For key \( k_i \), search cost = \( \text{depth}(k_i) \), where depth of the root is 1.
- Actual cost = # of items examined.

Expected Cost = \( \sum_{i=1}^{n} p_i \cdot \text{depth}(k_i) \)

Note the difference between this problem and Huffman trees.

Example

Consider 5 keys with these search probabilities:
\[ p_1 = 0.25, \ p_2 = 0.2, \ p_3 = 0.05, \ p_4 = 0.2, \ p_5 = 0.3. \]

\[
\begin{array}{c|c|c}
  i & \text{depth} & \text{depth}(k_i) \cdot p_i \\
  \hline
  1 & 2 & 0.5 \\
  2 & 1 & 0.2 \\
  3 & 4 & 0.2 \\
  4 & 3 & 0.6 \\
  5 & 2 & 0.6 \\
\end{array}
\]

Therefore, \( E[\text{search cost}] = 2.15. \)

Example

\[
\begin{array}{c|c|c}
  i & \text{depth} & \text{depth}(k_i) \cdot p_i \\
  \hline
  1 & 2 & 0.5 \\
  2 & 1 & 0.2 \\
  3 & 3 & 0.15 \\
  4 & 2 & 0.4 \\
  5 & 3 & 0.9 \\
\end{array}
\]

Therefore, \( E[\text{search cost}] = 2.1. \)

Example

Observations:
- Optimal BST may not have the smallest height.
- Optimal BST may not have highest-probability key at the root.

Naïve algorithm: build by exhaustive checking
- Construct each \( n \)-node BST.
- For each assign keys and compute expected cost.

How many trees? Described by Catalan numbers
\( \# \text{ trees} = O(4^n) \)

Step 1: Optimal Substructure

To find an optimal solution for \( k_1, \ldots, k_n \),
we must be able to find an optimal solution for \( k_i, \ldots, k_j \).

One of the keys in \( k_i, \ldots, k_j \) must be the root
Left subtree of \( k_r \) contains \( k_i, \ldots, k_{r-1} \).
Right subtree of \( k_r \) contains \( k_r+1, \ldots, k_j \).

Step 2: Recurrence relation

Let \( C_{i,j} \) be the optimal cost for \( k_i, \ldots, k_j \)

\[
C_{i,j} = \min_{k_r \in [i,j]} (C_{i,k_r-1} + C_{k_r+1,j}) + w_{i,j}
\]

\[
w_{i,j} = p_i + \ldots + p_j
\]

\[
C_{i,j} = p_i
\]
Step 3: Correctness

Let \( T \) be an optimal subtree with \( k_r \) be the root.

\[
C_{i,j} = \min_{k \in S_{i,j}} (C_{r-1,i,j} + C_{r+1,i,j}) + w_{i,j}
\]

where

\[
w_{i,j} = p_i + \ldots + p_j
\]

To prove the above formula, we compute the tree cost directly

\[
\text{Cost}(T) = 1 \times p_r + \sum_{m=1}^{r-1} p_{m \cdot \text{depth}_r(k_m)} + \sum_{m=r+1}^{j} p_{m \cdot \text{depth}_r(k_m)}
\]

Conclude the proof by changing

\[
\text{depth}_T \rightarrow 1 + \text{depth}_{T_L} \text{ and } \text{depth}_T \rightarrow 1 + \text{depth}_{T_R}
\]

Step 3: Correctness

Finally, we need to prove that

\[
C_{i,j} = \text{OPT}_{i,j}
\]

Case 1). \( \text{OPT}_{i,j} \leq C_{i,j} \). Trivial, just return a tree with \( k_r \) being the root.

Case 2). \( C_{i,j} \leq \text{OPT}_{i,j} \). Proof by induction

We computed in the previous slide that

\[
C_{i,j} = w_{i,j} + C_{r-1,i,j} + C_{r+1,i,j} \leq w_{i,j} + \text{OPT}_{r-1,i,j} + \text{OPT}_{r+1,i,j}
\]

\[
= \text{OPT}_{i,j}
\]

Step 4: Runtime Complexity

\[
C_{i,j} = \min_{k \in S_{i,j}} (C_{r-1,i,j} + C_{r+1,i,j}) + w_{i,j}
\]

where

\[
w_{i,j} = p_i + \ldots + p_j
\]

with initial conditions

\[
C_{i,j} = p_i \quad \text{and} \quad C_{i,j} = 0, \text{ if } j < i
\]

Table size - \( O(n^2) \)  

Total - \( O(n^3) \)

Cost per entry - \( O(n) \)