CS170– Spring 2022— Homework 6

CurMack

August 13, 2022

Chain Matrix Multiplication

Multiplying an $m \times n$ matrix by an $n \times p$ matrix takes mnp multiplications. How do we determine the optimal order, if we want to compute $A_1 \times A_2 \times \cdots \times A_n$, where the A_i's are matrices with dimensions $m_0 \times m_1, m_1 \times m_2, \ldots, m_{n-1} \times m_n$, respectively?
Function: For $1 \leq i \leq j \leq n$, define:

$$C(i,j) = \text{minimum cost of multiplying } A_i \times A_{i+1} \times \cdots \times A_j$$

Base Case: when $i = j$, $C(i,i) = 0$
Recurrence:

$$C(i,j) = \min_{i \leq k < j} \{C(i,k) + C(k+1,j) + m_{i-1} \cdot m_k \cdot m_j\}$$

Main Idea:

```
for i = 1 : n do
    C(i,i) ← 0
for s = 1 : n − 1 do
    for i = 1 : n − s do
        j ← i + s
        C(i,j) ← min{C(i,k) + C(k+1,j) + m_{i−1} · m_k · m_j : i ≤ k < j}
return C(1,n)
```

Runtime: $\mathcal{O}(n^3)$

2 Egg Drop Revisited

(a)

$$M(d,k) = M(d-1,k-1) + M(d-1,k) + 1$$

The highest floor we can drop the first egg from is $M(d-1,k-1)+1$, if the egg breaks, we can still solve the problem with the remaining $d-1$ drops and $k-1$ eggs. If the egg doesn't break, now we have $d-1$ drops and k eggs, we can at most solve $M(d-1,k)$ floors. So the the maximum number of floors for which we can always find l in at most d drops using k eggs is $M(d-1,k-1) + M(d-1,k) + 1$.

(b) For base cases, we take $M(0,k) = 0$ for any k and $M(d,0) = 0$ for any d. Starting with $d = 1$, we compute $M(d,x)$ for all, $1 \leq x \leq k$, and do so again for increasing values of d, up until we compute $M(d,x)$ for all $1 \leq x \leq k$. We return $M(d,k)$.
Runtime: $\mathcal{O}(dk)$ (we compute dk subproblems, each of which takes $\mathcal{O}(1)$ time)

(c) Similarly, Starting with $d = 1$, we compute $M(d, x)$ for all, $1 \le x \le k$, and do so again for increasing values of d, up until we firstly compute $M(d, k) \ge n$, then we return the value d as $f(n, k)$.

(d) Notices that d will always be at most n, since each floor will have at most 1 drop for the optimal solution. Since the original runtime is $\mathcal{O}(dk)$ and $d \le n$:
Runtime: $\mathcal{O}(nk)$

(e) we only need to store $M(d-1, x)$ and $M(d, x)$ for all x, i.e. we only ever need to store $\mathcal{O}(k)$ values. In particular, after computing $M(d, x)$ for all x, we can delete our stored values of $M(d-1, x)$.

3 Knightmare

(a) Use M-bit string to represent a valid configuration of knights on a single row, there are 2^M representations. We will solve the $N \times M$ chessboard from the subproblem of size $N - 1 \times M$, since the nth row configuration depends on the $n-1$th row and $n-2$th row.
Function:

$$\begin{gathered} K(n, u, v) = \text{the number of ways in an } n\text{-row board,} \\ u \text{ be the specific configuration of the } n\text{th row,} \\ v \text{ be the specific configuration of the } (n-1)\text{th row.} \end{gathered}$$

(b) **Base Case**: For all configurations u and v (no matter valid or not) :

$$K(2, u, v) = \begin{cases} 1 & \text{if valid} \\ 0 & \text{otherwise} \end{cases}$$

Recurrence:

$$K(n, v, w) = \sum_{\text{all valid } u,v,w} K(n-1, u, v)$$

return $\sum_{\text{all valid } u,v} K(n, u, v)$

(c) **Correctness**: for base case, we will brute force $n = 2$ rows, which s correct. If we have valid configuration $K(n-1, u, v)$, then for the nth row, we check last 3 rows u, v, w to see if they are valid and add all configuration to the n^{th} row solution to solve $K(n, v, w)$, which is correct.

(d) **Runtime**: $\mathcal{O}(2^{3M} \cdot N \cdot M)$, we have $\mathcal{O}(N)$ rows, $\mathcal{O}(2^{3M})$ subproblems, each has $\mathcal{O}(M)$ to check.
Space: $\mathcal{O}(N \cdot 2^{2M})$, we have $\mathcal{O}(N)$rows $\cdot$ $\mathcal{O}(2^{2M})$ subproblems per row.
We only need to store the last two row, so the space we need is: $\mathcal{O}(2^{2M})$.

4 Balloon Popping Problem

(a) Like matrix chain multiplication:
Function:

$$C(i, j) = \text{maximum amount of noise produced by popping balloons } i, i+1, \ldots, j$$

(b) **Base Case**: $s_0 = 1$ and $s_{n+1} = 1$, assuming the input is from 1 to n.
For $i = 1$ to n, $C(i,i) = s_{i-1} \times s_i \times s_{i+1}$
If $i > j$, $C(i,j) = 0$.

(c) **Recurrence**:

$$C(i,j) = \max_{i \le k \le j} \{C(i,k) + C(k+1,j) + s_{i-1} \cdot s_k \cdot s_{j+1}\}$$

k is the last balloon to pop in subset $i, i+1, \ldots, j$.
Finally return $C(1,n)$.
Runtime: $\mathcal{O}(n^3)$

5 Paper Cutting

(a) **Function**:

$$\begin{gathered} B(i_1, j_1, i_2, j_2) = \text{the minimum number of cuts} \\ \text{needed to separate the matrix } A[i_1 \ldots i_2, j_1 \ldots j_2] \end{gathered}$$

(b) **Recurrence**:

$$B(i_1,j_1,i_2,j_2) = \min \begin{cases} 0 & \text{if all entries in} A[i_1 \ldots i_2, j_1 \ldots j_2] \text{ are equal} \\ 1 + B(i_1,j_1,i_1+k,j_2) + B(i_1+k+1,j_1,i_2,j_2) & \text{for any } k \in \{1,\ldots,i_2-i_1\} \\ 1 + B(i_1,j_1,i_2,j_1+k) + B(i_1,j_1+k+1,i_2,j_2) & \text{for any } k \in \{1,\ldots,j_2-j_1\} \end{cases}$$

Base Case: set all single-square pieces to be 0.

(c) **Runtime**: $\mathcal{O}((m+n)m^2n^2)$
We have $\mathcal{O}(m^2n^2)$ total subproblems. For each subproblem, we examine up to m possible choices for horizontal splits, and n possible choices for vertical splits, which takes $\mathcal{O}(n+m)$ time. We can precompute the purities of every single possible subrectangle and store it in a table. So to solve our recurrence relation, if we can determine purity/impurity in $\mathcal{O}(1)$ time, then we can reach an overall time of $\mathcal{O}((m+n)m^2n^2)$.