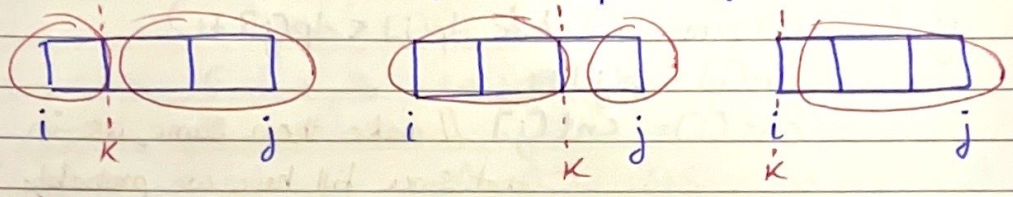
Lecture 37 : Partition DP

① Matrix Chain Multiplication.

Partion DP is used solve question of a similiar pattern, Somewhere where we have to use partition to ~~sto~~ solve part (1) & then part (2)

Eg:- We are given an arr[], starting pt → i
ending pt → j

we can make a partition (K) anywhere then solve the two parts and compute as per requirement. as shown



for MCM → Here we have to tell the minimum cost to Multiple N Matrix... we would be given an (N+1) sized [], which would lead (N) size matrix Eg.

Arr[] → | 10 | 20 | 30 | 40 |

This means Mat A → (10×20)
Mat B → (20×30)
Mat C → (30×40)

Now if we cal (AB)C → Then the cost would

(10×20×30) + (20×30×40)
⇒ 6000 + 24000
⇒ 30000 operations

And if we calc A(BC) ⇒ Then.

$(AB) \times C$ ⇒ $(10 \times 20 \times 30) + (10 \times 30 \times 40)$

$[]_{10\times30}$ $[]_{30\times40}$ ⇒ 6000 + 12000

⇒ 18000

for $A \times (BC)$ ⇒ $(20 \times 30 \times 40) + (10 \times 20 \times 40)$

$[]_{10\times20}$ $[]_{20\times40}$ ⇒ 24000 + 8000

⇒ 32000

Thus Multiplying (AB)C is much more optimal.

Rules

① Start with entire block/Array

* f(i, j)
  start ↘ end

② Try all partitions

* Run a loop to try all partitions.

③ Return the best possible 2 partitions.

PTO

Think in regard with array.

[10, 20, 30, 40, 50]
A B C D
i j

we can access a matrix dimension by (i) & (i-1)
Start → i , end → j

$f(i,j) = f(1,4)$ → return the minimum multiplication to multiply mat 1 → mat 4
Index

Base Case

Try imagining we will hit the base case at $(i == j)$; at that point there's calculation possible thus return 0;

Try possible Partition / Calculations

We ~~took~~ loop K → i − (j − 1)
0 1 2 3 4
i j

[10, 20, 30, 40, 50]
A B C D

K = 1 → f(i, K) , f(K+1, j)

K = 2 → f(1, 2) , f(3, 4)

K = 3 → f(1, 3) , f(4, 4)

Understand for f(1,1) , f(2, 4)

A (10×20) B C D (20×30) × (30×40) × (40×50)

(20×40) × (40×50)

(20×50)

And this same (BC)D == B(CD), look

(20×30) × (30×40) × (40×50)

(20×50) × (30×50)

(20×50)

Same

No. of operations for (A) × (BCD) will be
(10×20) (20×50)

→ (10×20×50) + the rest calculation from (BCD) done by recursion.

A[i-1] * A[k] * A[j]

This is the pattern because (i-1) → 0th index
(k) → 1st index
(j) → 4th index

For Memoisation → [2D] array..

```
int f(int i, int j, vector<int>& arr, vector<vector<int>>
                                        & dp) {

    if (i==j)
        return 0;
    if (dp[i][j] != -1)
        return dp[i][j];

    int mini = 1e9;
    for(int k=i ; k<j ; k++) {

        int cost = arr[i-1] * arr[k] * arr[j]
                 + f(i, k, arr, dp) + f(k+1, j, arr, dp);

        mini = min(cost, mini);
    }
    return dp[i][j] = mini;
}
```

Time Complexity → $O(N*N)*N \sim O(N^3)$

Space Complexity → $O(N^2) + O(N)$
↳ ASS.

Tabulation

The Same Rules As always

1. Copy Base Case
2. Write Down Changing parameters * i & j traverse in opp. order as in Memoisation.

* $i \Rightarrow n-1 \longrightarrow 1$

* $j \Rightarrow i+1 \longrightarrow 1$

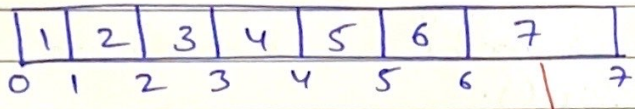
Try to understand that 'j' can never be to the left of i, it's always gonna be towards right.

```
int f(vector<int>& arr) {
    int n = arr.size();
    vector<vector<int>> dp(n, vector<int>(n, 0));

    for(int i = 1; i < N; i++)
        dp[i][i] = 0; // Base Case

    for(int i = n-1; i >= 1; i--) {
        for(int j = i+1; j < N; j++) {
            int mini = 1e9;
            for(int k = i; k < j; k++) {
                int cost = arr[i-1] * arr[k] * arr[j]
                           + dp[i][k] + dp[k+1][j];

                mini = min(cost, mini);
            }
            dp[i][j] = mini;
        }
    }
    return dp[1][n-1];
}
```

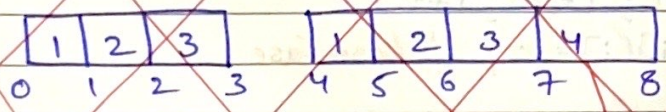
Lecture 38: Minimum cost to Cut the Stick

ATQ, we will be given a cuts[], cuts[i] represend where we want to make a cut, and the to make that cut it would equal to the remainging length Eg.

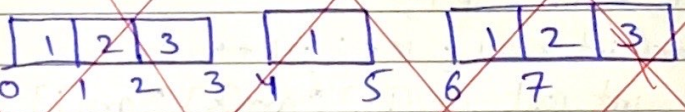
cuts[] = 3, 5, 1, 4.



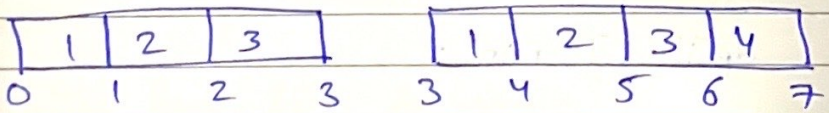
first cut at 3 → cost = 7.



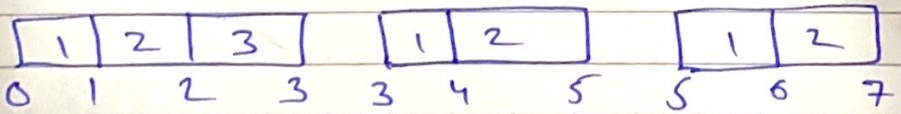
Then at 5 ⇒ cost = 7 + 4.



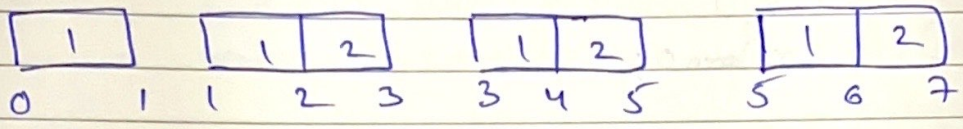
cut at 3 ⇒ cost = 7.



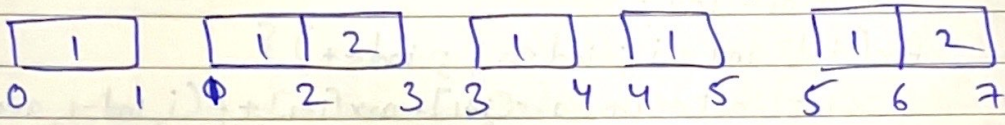
cut at 5 ⇒ cost = 7 + 4.



cut at 1 ⇒ cost ⇒ 7+4+3.



at 4 ⇒ cost ⇒ 7+4+3+2 ⇒ 16



length of Stick.

we will add '0' and 'n' to calculate the cost because we the length of the curr cut, to find that we need ~~ten~~ length

Add '0' & 'n' and sort the cuts[]

Using Partition DP to calc

↳ cuts[j+1] - cuts[i-1] + f(i, ind-1) + f(ind+1, j)

Clearly Base Case is at i > j → can't do anything return 0.

Use 2D[][] for Memoization.

(1). m = size of cuts[].

```
int f(int i, int j, vector<int>& arr, vector<vector<int>>& dp){

    if (i>j)
        return 0;

    if (dp[i][j] != -1)
        return dp[i][j];

    int mini = 1e9;
    for(int ind = i; ind<=j; ind++){
        int cost = abs(arr[j+1] - arr[i-1] + f(i, ind-1, arr, dp)
                   + f(ind+1, j, arr, dp);
        mini = min(mini, cost);
    }
    return dp[i][j] = mini;
}.
```

TC → $O(M^2 \times M) \approx O(M^3)$; SC → $O(M^2) + O(M)$ (ASS)

Tabulation

* Copy base Case
* Traverse the changing parameters in reverse fashion
  - i ⇒ m → 1
  - j ⇒ 1 → m
* Copy Recurrence

TC → $O(M^3)$
SC → $O(M^2)$

```
int f (vector<int>& cuts) {
    int m = cuts.size();
    cuts.push_back(0);
    cuts.push_back(m);
    vector<vector<int>> dp(m+2, vector<int>(m+2,0));

    for(int i= m ; i >= 1 ; i--) {
        for(int j=1 ; j<=m ; j++) {
            if(i>j)
                continue;
            for(int ind=i ; ind<=j ; ind++) {
                cost = abs(cuts[j+1] - cuts[i-1] +
                        dp[i][ind-1] + dp[ind+1][j];
                cost=
                mini = min(mini, cost);
            }
            dp[i][j]=mini;
        }
    }
    return dp[1][m];
}
```