Todays Content:

- → Matrix Chain Multiplication
- → Palindrome partition
- → 2nd Version of Knapsack

Matrix Multiplication:

→ A * B = R
r_1 $c_1 == r_2$ c_2 r_1 c_2

→ A[3, 4] * B[4 2] = R[3 2]

A
	0	1	2	3
0	1	2	0	1
1	3	2	1	4
2	-1	0	1	2

B
	0	1
0	2	1
1	1	0
2	-1	1
3	2	-1

* res
	0	1
0	6	0
1	15	0
2	1	-2

- res[1, 0] = 1^{st} row in A * 0^{th} col in B
- res[2, 1] = 2^{nd} row in A * 1^{st} col in B
- res[2, 0] = 2^{nd} row in A * 0^{th} col in B

→ A[2 5] * B[5 3] = R[2 3]

→ A[4 3] * B[3 2] = R[4 2]

// A B = R
r_1 $c_1 == r_2$ c_2 r_1 c_2

→ # How many elements in Resultant = $r_1 * c_2$

→ R[i, j] = i^{th} in A * j^{th} col in B = c_1 iterations
($c_1 == r_2$)

i^{th} A: 0 0 0 0 0

j^{th} B:
0
0
0
0
0

→ # operations required to get all elements res[]
= $r_1 * c_2 * c_1$ or r_1 c_2 r_2

obs: A (l*m) B (m*n) → R (l*n) TC: O(l*m*n)

→ 3 matrices

M_1 (3*5) * M_2 (5*7) * M_3 (7*4) = Res (3*4)

Case I: $[M_1 * M_2]$ (3*7) * $[M_3]$ (7*4) = Res (3*4)

#cost = 105 + 84 = 189 → prefer way 1

Case II: $[M_1$ (3*5) * $[M_2 * M_3]$ (5*4)$]$ = Res (3*4)

#cost = 60 + 140 = 200

// Given N matrix find min cost iteration to apply all matrices

N=4 → [5 value]

N=5 → [6 value]

N → [N+1 value]

A B C D

a_1 $b_1 == a_2$ $b_2 == a_3$ $b_3 == a_4$ b_4

N = 5 → [6 values]

	0	1	2	3	4	5
v[6] =	2	3	4	1	7	3

	r	c
1st mat =	v[0]	v[1]
2nd mat =	v[1]	v[2]
3rd mat =	v[2]	v[3]

	r	c	
ith mat	v[i-1]	v[i]	# obs1

	0	1	2	3	4	5
v[6] =	2	3	4	1	7	3

→ We multiply all mat 1st → 3rd = v[0] * v[3] (res)

1st 2nd 3rd

v[0] v[1] v[1] v[2] v[2] v[3]

→ We multiply all mat 2nd → 4th = v[1] * v[4] (res)

→ We multiply all mat 2nd → 5th = v[1] * v[5] (res)

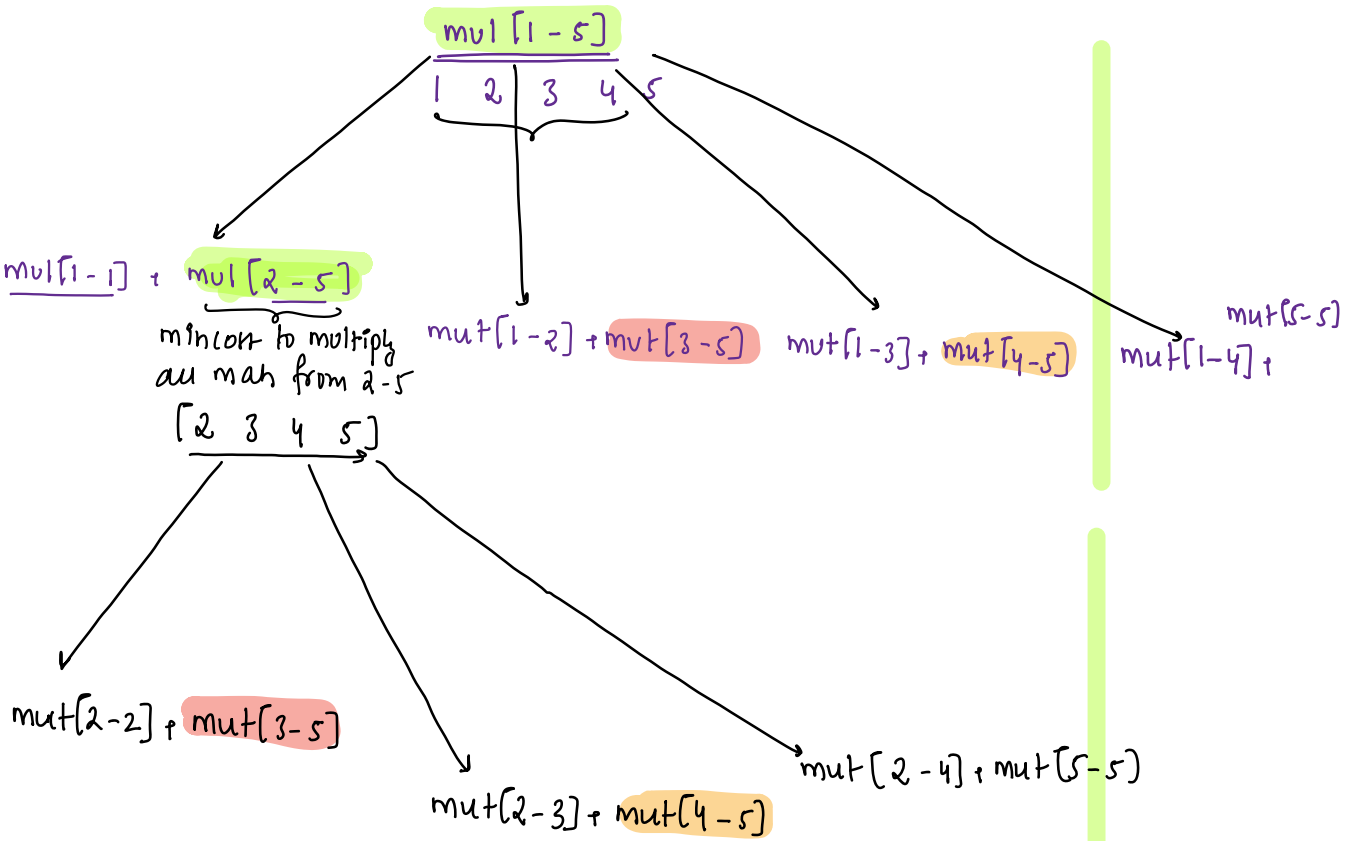
v[1] * v[2] .. v[4] * v[5]

→ We multiply all mat ith → jth = v[i-1] * v[j] # obs2

→ dimensions of resultant matrix

N=5 : → overlapping, optimal substructure ⇒ {dp}

minimum cost to multiply all matrix from [1-5]



mat(i, j) = { min cost to multiply all matrix from (i-j) }

mat(i,j) = { mat[i, i] + mat[i+1, j] + c_1 ; mat[i, i+1] + mat[i+2, j] + c_2 ; mat[i, i+2] + mat[i+3, j] + c_3 ; ⋮ ; mat[i, j-1] + mat[j, j] + $c_{..}$ }

k k+1 cost to resultat

→ = min for(int k = i; k < j; k++){

mat[i, k] + mat[k+1, j] + v[i-1]*v[k]*v[j]

min cost to mul all mat[i, k] — resultat → res_1 = v[i-1]*v[k]

min cost to mult all mat [k+1, j] → res_2 = v[k]*v[j] cost to multiply ralter matrix

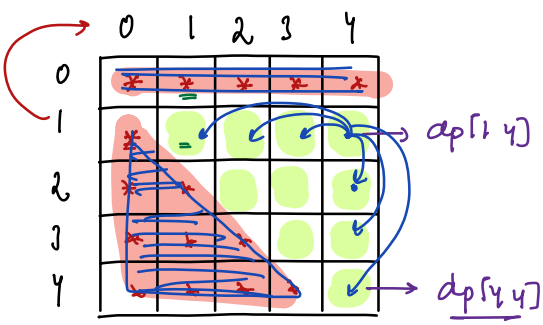
// final dp expression

$$mat[i,j] = \min\left(\begin{array}{l} k<j \\ \forall \\ k=i \end{array} mat[i\ k] + mat[k+1\ j] + v[i-1]*v[k]*v[j]\right)$$

$k=i;\ k<j;\ k++$] if $i==j$, it wont even enter loop

// dp Table : // given N matrix, we need to get min cost to multiply all matrices from [1- N] = mat[1][N]

- mat[N+1][N+1]
- Eg: N=4 : mat[5][5]



Base conditions :

if $i==j$ expression valid

mat[i,j] = 0

```
int mcm(int i, int j, int v[], int mat[][]){
    if (i==j) { return 0;}
    if(mat[i,j]==-1){
        int ans= INT_MAX
        for(int k=i; k<j; k++){
            int l= mcm( i, k, v, mat) // min cost to mul [i-k]
            int r= mcm (k+1, j, v, mat) // min cost to mul [k+1-j]
            int b = v[i-1]*v[k]*v[j]
        }   ans = min (ans, l+r+b)
        mat[i,j] = ans
    }
    return mat[i,j]
}
```

TC: # dp States * # TC for each State

$O(N^2) * N$

TC: $O(N^3)$ SC: $O(N^2)$

↳ dimensions of matrix

```
int minCost(int N, int V[]) {
    int mat[N+1][N+1] = {-1};
    return mcm(1, N, V, mat)
}
```

min cost to multiply all matrix from 1-N

2(Q) Given a String s[N] construct a bool mat[N][N]

mat[i][j] = True if substring [i-j] is a palindrome $i <= j$

	0	1	2	3	4	5
String s =	b	b	d	a	d	b

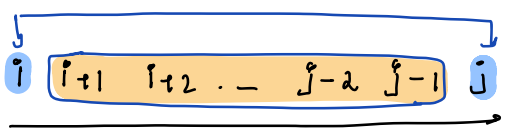
bool mat[6][6] =

	0	1	2	3	4	5
0	T	T	F	F	F	F
1	–	T	F	F	F	T
2	–	–	T	F	T	F
3	–	–	–	T	F	F
4	–	–	–	–	T	F
5	–	–	–	–	–	T

Idea1: For every substring (i, j)
check palindrome or Not
TC: $O(N^2) * O(N) = O(N^3)$

Idea2:

mat[i, j] = { Check if Substring is palindrome or Not }



mat[i, j] = (s[i] == s[j] && mat[i+1, j-1])

Fill matrix TODO

TC: $O(N^2) * 1$

Eg: N = 5:

1) optimal substructure
2) overlapping *

s[0 5]
↓
s[1 4]
↓
s[2 3]

28) Palindrome Partitioning - II

// Given a String s, minimum number partitions required to be done on s, such that all partitions are palindromes

Eg1: a n a | c | o n o | a a | = 4

Eg2: 0 1 2 3 4 5 6 7 8 9 10 11
a | b c b | a b a a a b | a | a = 4

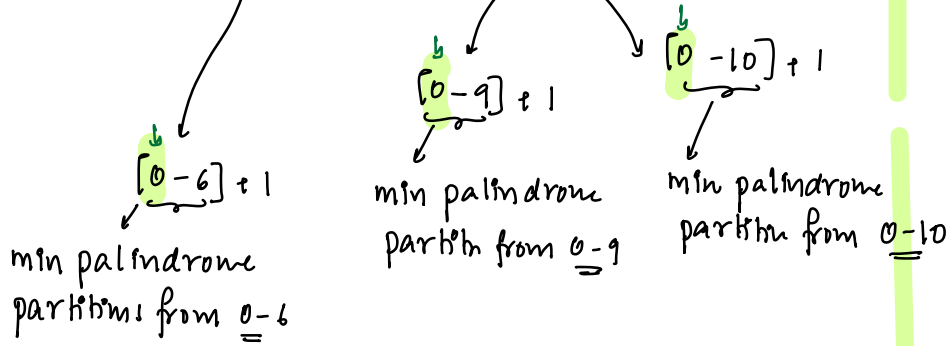
a b c b a | b a a a b | a a | = 3

Eg3: a | b | c | d | e = 5

Eg4: a b c b a = 1

min palindrome partitions for sub [0-11]

0 1 2 3 4 5 6 7 8 9 10 11
a b c b a b a a a b a a



dp[i] = { min palindrome partition from [0 → i] }

0 1 2 3 4 .. i-3 i-2 i-1 i

dp[i] = min {
- if(substr[j, i] is palindrome) dp[0, i-1] + 1
- if(substr[i-1, i] is palindrome) dp[0, i-2] + 1
- if(substr[i-2, i] is palindrome) dp[0, i-3] + 1
- if(substr[i-3, i] is palindrome) dp[0, i-4] + 1
- if(substr[1, i] is palindrome) dp[0, 0] + 1
- if(substr[0, i] is palindrome) dp[0, -1] + 1 ≥ → Base Condition:

// dp table:
- → min partitions for entire string [0, N-1] ⇒ dp[N-1]
- → dp[N]

// Base Conditions:
- → if 0 - i is already palindrome return 1;

```
int minpa(int i, int dp[], char s[], bool pal[][]){
    if(pal[0][i]) { return 1}        // substring [0 i] is palindrome

    if(dp[i] == -1){
        int ans = N //ans cannot exceed N
        for(int j= i-1; j>=0; j--){
            if( pal[j+1][i] ){ Substring from [j+1, i] should be palindrome)
                ans = min(ans, minpa(j, dp, s)+1)
            }                                  // min partitions [0 j]
        }
        dp[i] = ans;
    }
    return dp[i]
}
```

Check substring palindrome or Not N times

TC: N * [N^2] ⇒ O(N^3)

dp states

Time taking to check if substring is palindrome or not

TC using pal[][]

→ # dp states : N * N → O(N^2)

Total TC: O(N^2 + N^2)

SC: O(N + N^2) → to construct pal[][]

```
int partitions(char s[]){
    int n = s.length
    int dp[N]= -1;
    bool pal[N][N] = F;
        → for every subarray i-j check palindrom or not
    minpa(N-1, dp, s, pal)
}
```