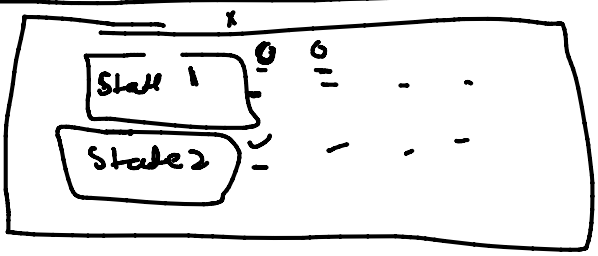
CLASS 86

16 May 2023 21:09

Knapsack Based DP, linear DP, Include Exclude DP



Grap DP

DP using Grap Strategy

longest Pallindrumic substring

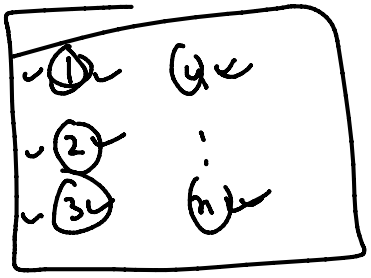
1 s = " b a b a d " → b a b a d

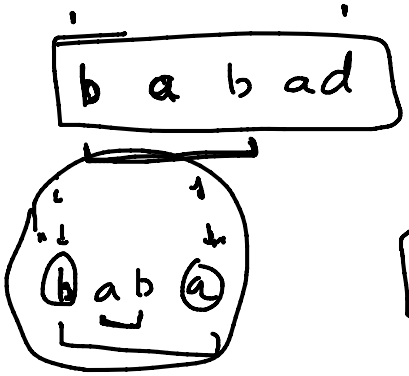
~~Naive Approach~~

Brute Force →

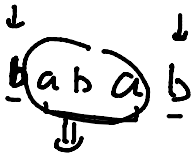
n^2 → substrings generate + O(n) → n^3

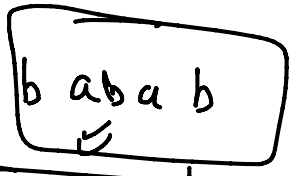
optimized approach





s[i] == s[j]





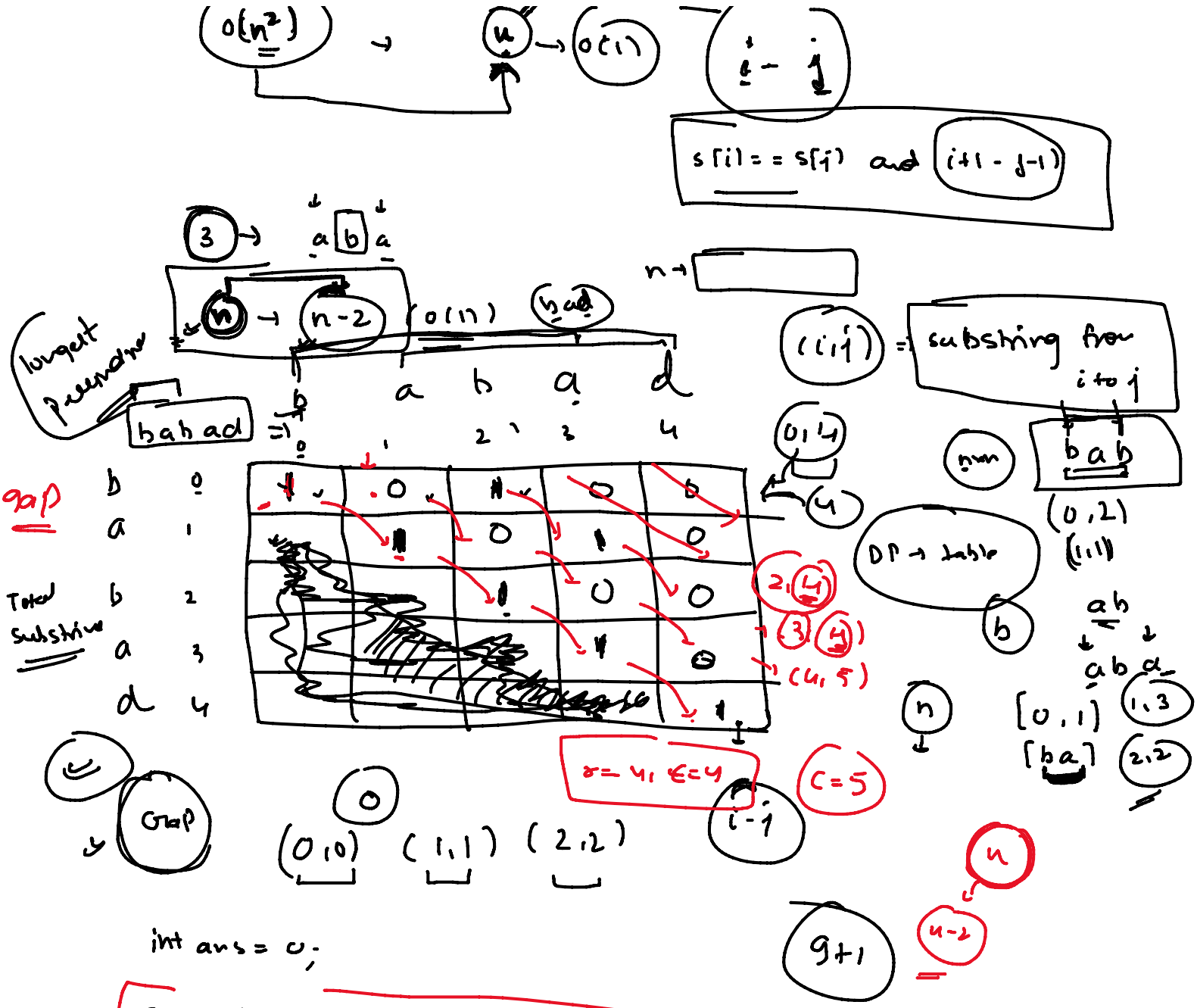
Pallindrom

⇒ s[i] == s[j] and str(i+1, j-1) → O(1) ⇒ 6

i - j

$O(n^2)$ → u → O(1)

i - j


O(n²)
O(1)
i - j
s[i] == s[j] and (i+1 - j-1)
3
a b a
n
n-2
O(1)
babad
longest palindrome
babad
a b a d
0 1 2 3 4
((i,j)) = substring from i to j
bab
(0,2)
((1,1))
gap
b 0
a 1
Total substring
b 2
a 3
d 4
DP → table
(0,4)
4
2,4
3,4
4,5
ab
aba
[0,1]
[ba]
1,3
2,2
r=4, c=4
c=5
i-1
(0,0) (1,1) (2,2)
g+1
u
u-2
int ans = 0;
For (int g=0 ; g < n; g++)
int r = 0;
int c = g;
r=0, c=0
while (c < n)
if (c == r) [// string of length 1]
dp[r][c] = 1;
ab
elseif (c == r+1) [// string of length 2]

```
{   if (s[r] == s[c])
        dp[r][c] = 1;
    else
        dp[r][c] = 0;
}
else  [ // string of length > 2 )
{   if (s[r] == s[c] and dp[r+1][c-1] == 1))
    {   dp[r][c] = 1;
    }
    else
        dp[r][c] = 0;
}
if (dp[r][c])
{   ans = max(ans, c-r+1);
}
r++;
c++;
}
return ans;
```

r - c

c - r + 1

Count total Palindromic substrings

DP Matrix

```
int i,j,k,n;
        n=s.size();

        vector<vector<int>>dp(n,vector<int>(n,0));

        for(int g=1;g<n;g++){
            int r=0;
            int c=g;
            while(c<n){
                if(c==r+1){
                    if(s[r]==s[c]){
                        dp[r][c]=1;
                    }
                }
                if(dp[r+1][c-1] and s[r]==s[c]){
                    dp[r][c]=1;
                }
                r++;
                c++;

            }
        }
        int ans=0;
        for(i=0;i<n;i++){
            for(j=0;j<n;j++){
                if(dp[i][j]){
                    ans++;
                }
            }
        }
        return ans;
```

Count No of 1 in Matrix.

Binary search

Linked

Tree

Graph

Recursion

Backtracking

Greedy

DP

Complexity

Exponential

$n <= 16 <= 20$

n^3

$n = 100$

$n = 1000$

2^{100}

n^2

Grid Based DP

(i,j) → num

(i+1,j)

(i,j+1)

Total Node

Total Paths ?

(i-1,j) → (0,0)

(i,j-1)

Path in Maze

(0,0)

Total Path = (i,j+1) + (i+1,j)

(i,j) → (n-1, n-1)

(0,0) → (0,0)

Memoization

(i,j) → (n-1, m-1) (0,0) → (0,0)

```
int total_Paths ( int i, int j, int n, int m)
{  if ( i == n-1 and j == m-1)
   {  return 1;
   }
   if ( dp[i][j] != -1 ) return dp[i][j]
   int op1 = total_Paths ( i+1, j, n, m);
   int op2 = total_Paths ( i, j+1, n, m);
   dp[i][j] =
   return op1 + op2;
}
```

0 - n

(n - 0)

total_Paths (0, 0, n, m);

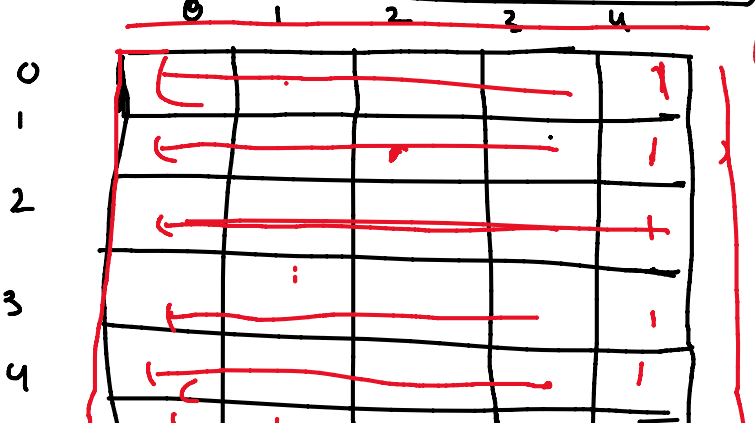
Tabulation

```
int total_Paths ( int i, j )
{  if ( i == 0 and j == 0)
      return 1;
   int op1 = totalPaths (i-1, j);
   int op2 = " " (i, j-1);
   return op1 + op2;
}
```

2D DP

DP on Grid

total_Paths (i, j) =



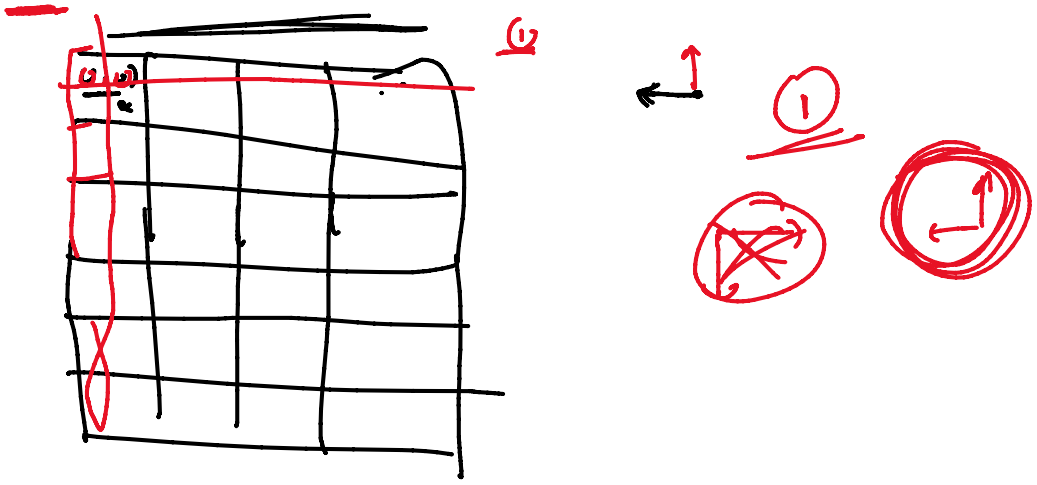
(i,j) → (n,m)

(i,j)

dp[i][j] = dp[i][j+1]

dp[i-1][j]

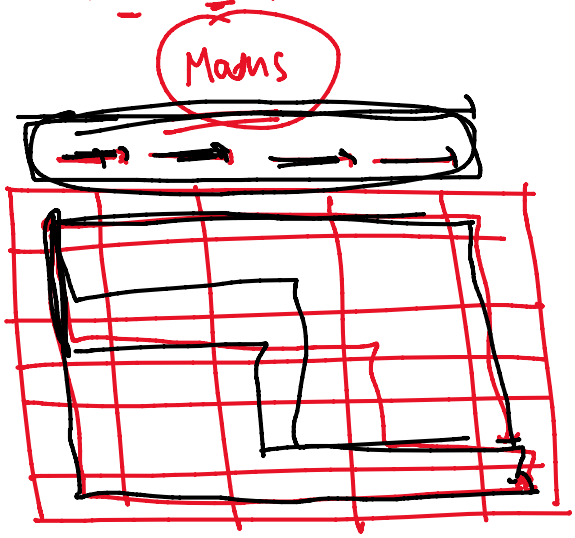
(n,m)

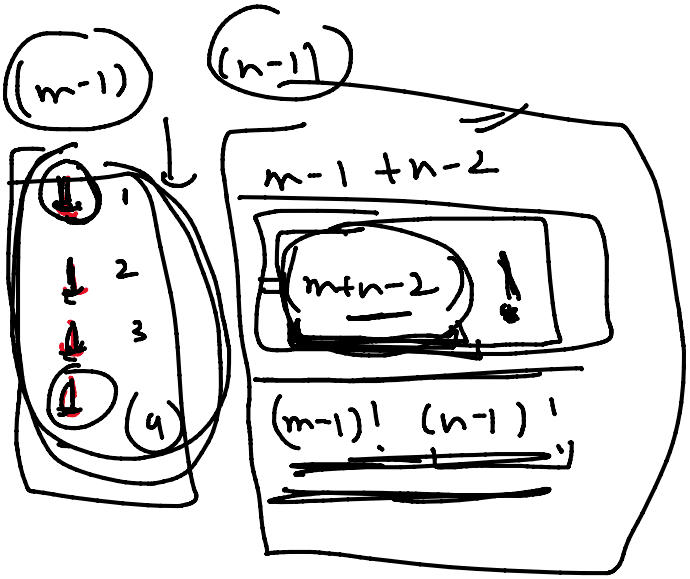


```
Vector < vector<int> > dp(m+1, vector<int>(n+1, 0));

for (i = 0; i<m; i++)
{ dp[0][i] = 1;
}

for (i=0; i<n; i++)
{ dp[i][0] = 1;
}

for (i=1; i<m; i++)
{ for (j=1; j<n; j++)
  { dp[i][j] = dp[i-1][j] + dp[i][j-1];
  }
}
```

base case

return dp[n][m];





(n) element

(n)

11 22

11 2 2

- (1) (2) (3) (4)
- (4) (2) (3) (1)

Total ways

$5! \Rightarrow \frac{5!}{3! \times 2!} \Rightarrow \frac{5 \times 4 \times 3!}{3! \times 2!} = 10$