Todays Content:

- # Party Pairs
- # Min Squares to get Sum
- # Max Subsequence Sum
  - # Max Subsequence sum, without adjacent Elements
  - # Similar question in assignments

Party Pairs: Given N persons, how many ways we can pair all people

Note: A person either wants to stay alone or get paired up → {Gold Man Sachs}

people need to present party

N=1: → 1 way

N=2: → 2 ways

→ way1

→ way2

N=3: → 4 ways

→ way1

→ way2

→ way3

→ way4

N=4:

→ Single

→ Pair

4 ways

2 people : 2 ways

2 ways

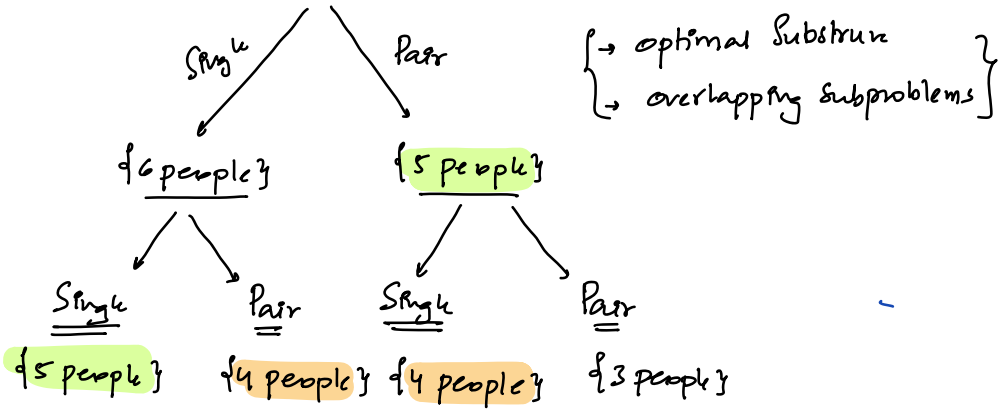
2 people : 2 ways

2 ways

2 people : 2 ways

2 ways

Qn: { P_1 P_2 P_3 P_4 P_5 P_6 P_7 } : # people:



{ → optimal Substruc
→ overlapping subproblems }

Qn: P_1 P_2 P_3 P_4 P_5

party(5) = party(4) + 4 party(3)

P_5 → Single ⇒ { P_1 P_2 P_3 P_4 }
party(4)

P_5 → Mingled :

P_5 P_4 ⇒ { P_1 P_2 P_3 }
party(3)

P_5 P_3 ⇒ { P_1 P_2 P_4 }
party(3)

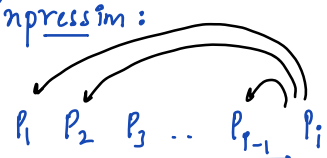
P_5 P_2 ⇒ { P_1 P_3 P_4 }
party(3)

P_5 P_1 ⇒ { P_2 P_3 P_4 }
party(3)

Dp steps:

dp[i] = # ways i people can party

Expression:



# ways	people	# count
P_i	i-1	dp[i-1]
P_i P_1	i-2	dp[i-2]
P_i P_2	i-2	dp[i-2]
P_i P_3	i-2	dp[i-2]
:		
P_i P_{i-1}	i-2	dp[i-2]

Total ways we can pairs i ways = dp[i]

dp State : dp[i] = # number of ways to party i people

dp Expression : dp[i] = dp[i-1] + (i-1) dp[i-2]

Base Condi : Above expression for i=0 / i=1

dp[0] = 0
dp[1] = 1
dp[2] = dp[1] + 1 * dp[0]
= 1
↳ If we go with above base conditions, dp[2] going wrong

dp[0] = 1
dp[1] = 1
dp[2] = dp[1] + 1 * dp[0]
= 2
↳ With above base conditions code is working

Dp Table : dp[N+1]

Pseudocode :

```
int Party(int n){ → Iterative DP → Tabulation
    int dp[n+1]
    dp[0]=1, dp[1]=1        change →  dp[1]=1, dp[2]=2
    i=2; i<=n; i++){        →         i=3;
        dp[i] = dp[i-1] + (i-1)*dp[i-2]
    }
    return dp[n]
}
```

TC: → # States * # TC for each State
N 1

TC: O(N) SC: O(N)

SC: At any given point we are at max using 3 states
↳ O(1) → TODO

2qs) Min no: of perfect Squares to be added to get Target sum

a) optimal Structure

b) Overlapping Sub Problem

$N=6 \rightarrow 1^2+1^2+2^2 = 6 : 3$

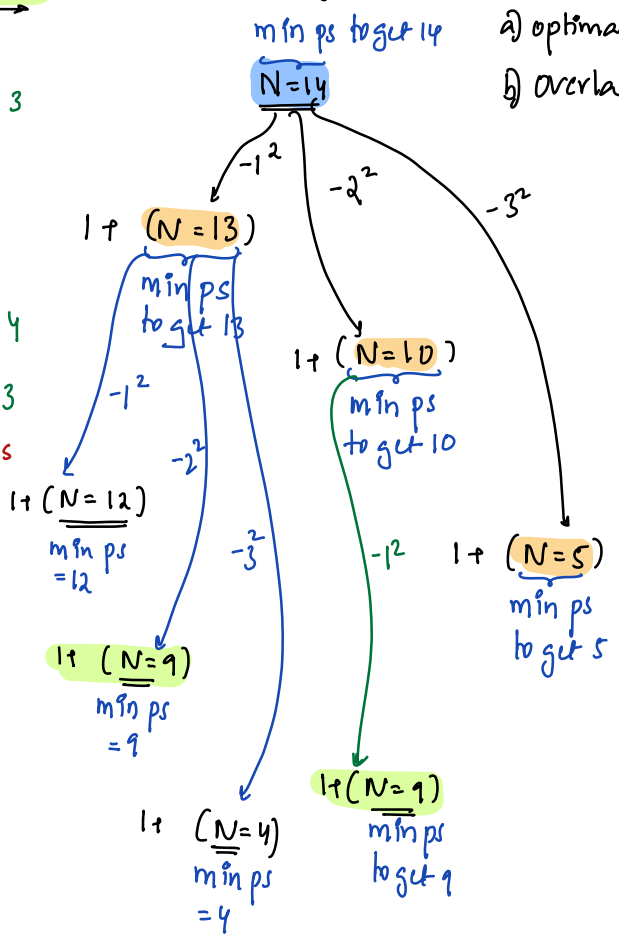
$N=10 \rightarrow 1^2+3^2 = 10 : 2$

$N=9 \rightarrow 3^2 = 9 : 1$

$N=12 \rightarrow 3^2+1^2+1^2+1^2 = 12 : 4$

$\rightarrow 2^2+2^2+2^2 = 12 : 3$

$\rightarrow$ 3, greedy fails



dpState: dp[i] = min psquares required to get i sum

dpexpression: $dp[i] = \min \begin{cases} dp[i-1]+1 \\ dp[i-2^2]+1 \\ dp[i-3^2]+1 \\ \vdots \\ dp[i-j^2]+1 \end{cases}$ $j^2 \le i$

$$dp[i] = \min_{\substack{j^2 \le i \\ j=1 \\ \forall}} \left[dp[i-j^2]+1 \right]$$

dp base : dp[0] = 1 *

$$dp[i] = \min_{j=1}^{} dp[i - j^2] + 1$$

$$= dp[0] + 1$$

$$= 2$$

dp[0] = 0 ✓

$$dp[i] = \min_{j=1}^{} dp[i - j^2] + 1$$

$$= dp[0] + 1$$

$$= 0 + 1 = 1$$

Pseudocode :

```
int minPerfectSquare(int n) {
    int dp[n+1]
    dp[0] = 0
    i = 1; i <= N; i++) {
        ans = i
        j = 1; j*j <= i; j++) {
            ans = min(ans, dp[i - j*j] + 1)
        }
        dp[i] = ans;
    }
    return dp[N]
}
```

TC : #States * TC for each state ⇒ O(N√N)

O(N) O(√N)

SC : O(N)

↳ Space Optimization is not Possible? TODO

10:45pm → 10:55pm

3Q) Given N elements find max Subsequence Sum.

↳ ordered based on index

Ex1: 2 -4 5 3 -8 1 → 11

Ex2: 2 6 -1 4 3 -5 → 15

Ex3: -4 -3 -8 -2

Ex4: 3 2 4 8 → 17

Ideal:

Find sum of +ve elements

Edge: If all are -ve pick max ele

4Q) Given arr[N], find max subsequence sum

Note: In a subsequence, 2 adjacent elements cannot be picked

Max Subsc Sum, Empty Subsequen not possible

Ex1: 9 14 3 → 14

Ex2: 9 4 13 24 → 33

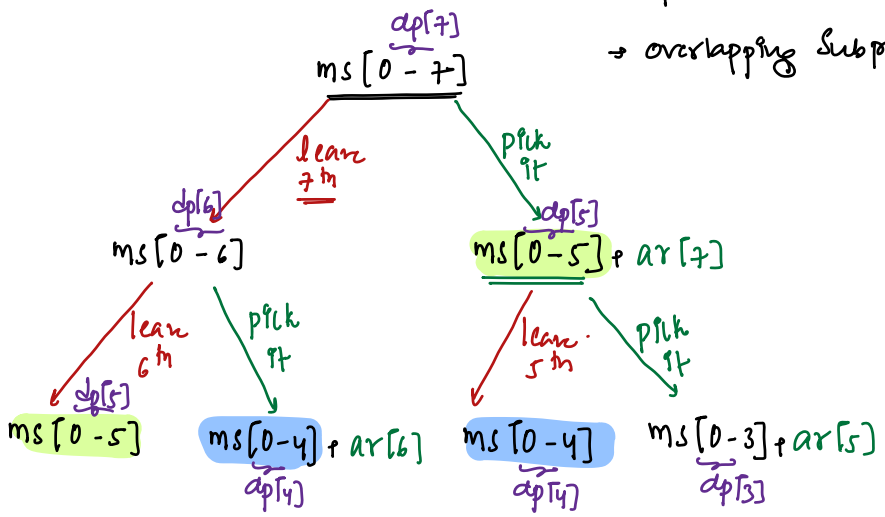
Ex3: 13 14 2 → 15

idea: Max {sum of even inde, sum of odd inde} *

	0	1	2	3	4	5	6	7
ar[8] =	2	-1	-4	5	3	-1	4	2

→ optimal Substructure

→ overlapping Subproblems



dp State: dp[i] = Max subsequence sum from [0-i]

dp Expr: $dp[i] = \max \begin{cases} ar[i] + dp[i-2] & = \text{Pick } i^{th} \text{ index} \\ dp[i-1] & = \text{leave } i^{th} \text{ index} \end{cases}$ (if -ve we avoid it)

dp base:

i=0 : dp[0] = ar[0] Max subsequence sum from [0-0]

i=1 ; dp[1] = max(ar[0], ar[1]) Max subsequence sum from [0-1]

dp table: dp[N]

Pseudocode:

```
int maxSubSum(int ar[]){
    int n = ar.length
    int dp[n];
    dp[0] = ar[0], dp[1] = max(ar[0], ar[1])      // Edge case
    for(i=2; i<n; i++){
        dp[i] = max(dp[i-1], dp[i-2] + ar[i], ar[i])
    }
    return dp[n-1]
}
```

TC: #States * #TC for each State

N * 1

TC: O(N)

SC: O(N)

↳ At max we are using 3 state, we SC: ⇒ O(1) can optimize Space ⇒ O(1) TODO

ar[] = -2 -4 1

dp[3] = -2 -2 -1

dp[2] → { dp[1] = -2 ; dp[0] + ar[2] = -2 + 1 }

ar[] = -3 -6 -8 -2 (0 1 2 3)

dp[4] = -3 -3 -3 -2

dp[2] = { ar[2] = -8 ; ar[2] + dp[0] = -8 + -3 ; dp[1] = -3 }

dp[3] = { ar[3] → -2 ; ar[3] + dp[1] → -2 + -3 ; dp[2] → -3 }