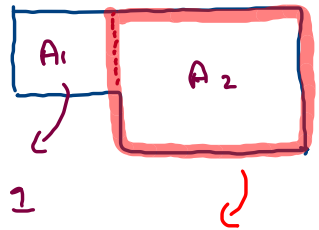
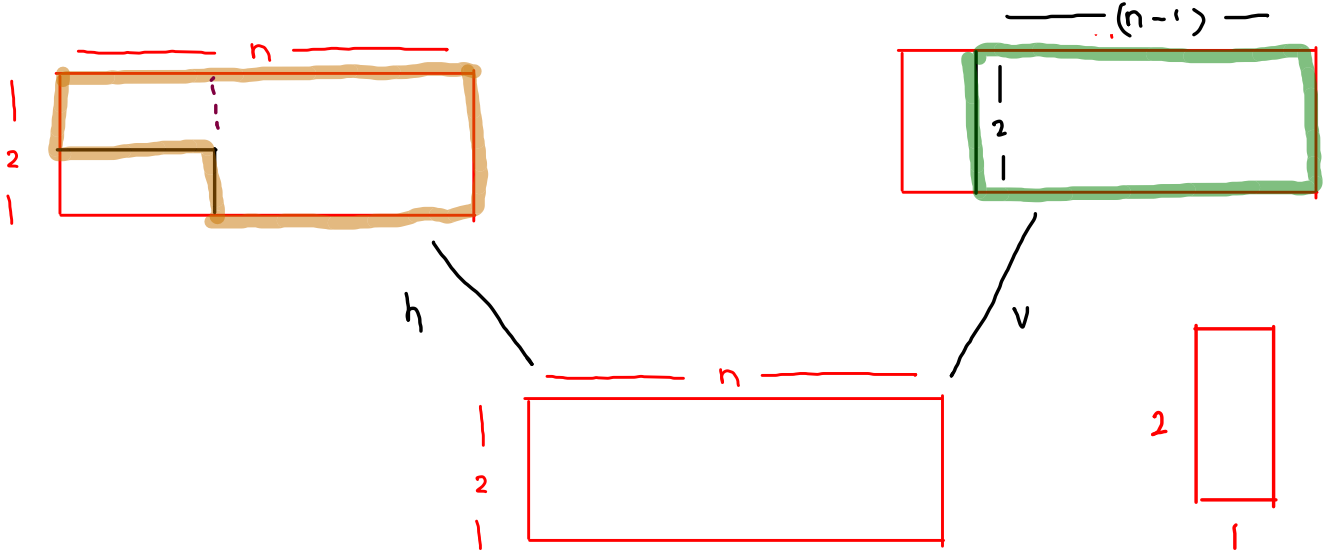
Tiling With 2 * 1 Tiles

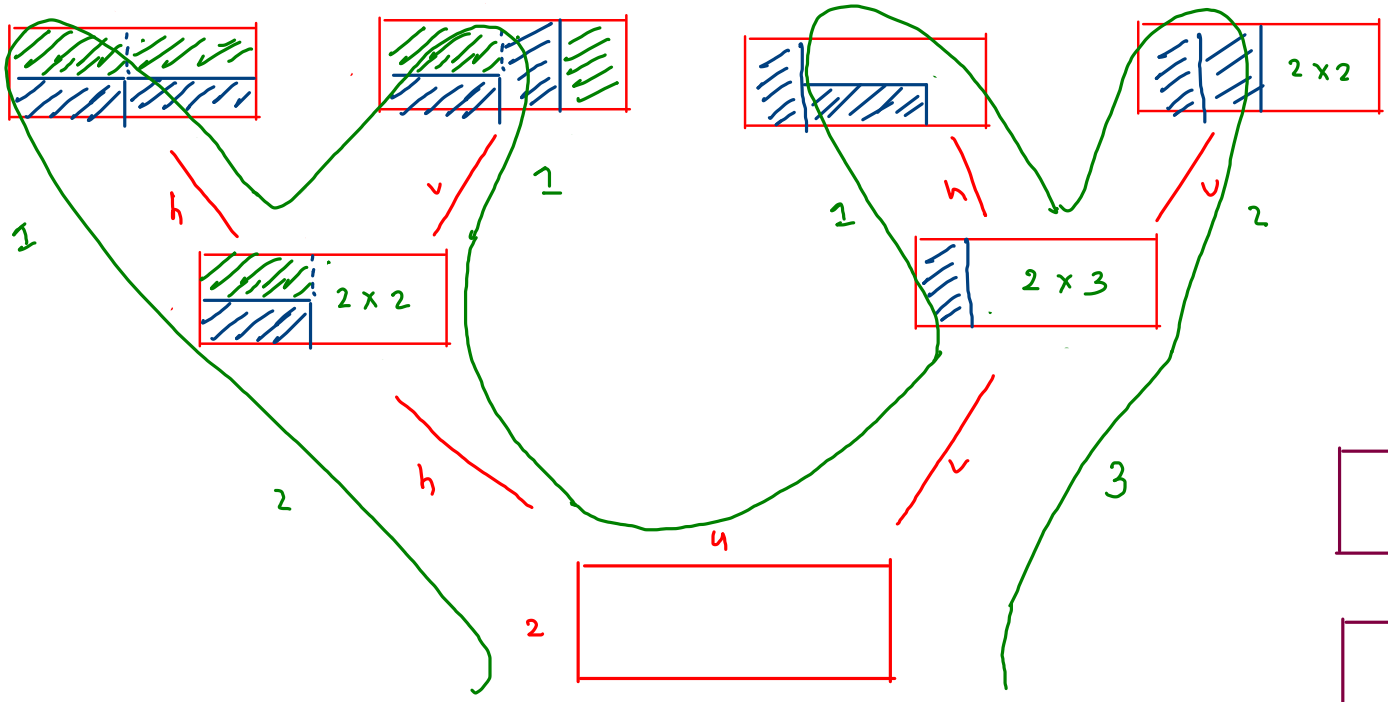


2 × n → floor

tiles → 2×1



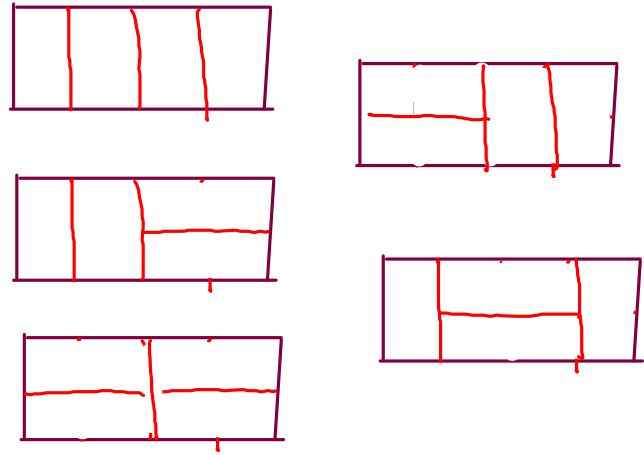
$$f(2,n) = f(2,n-1) + f(2,n-2)$$

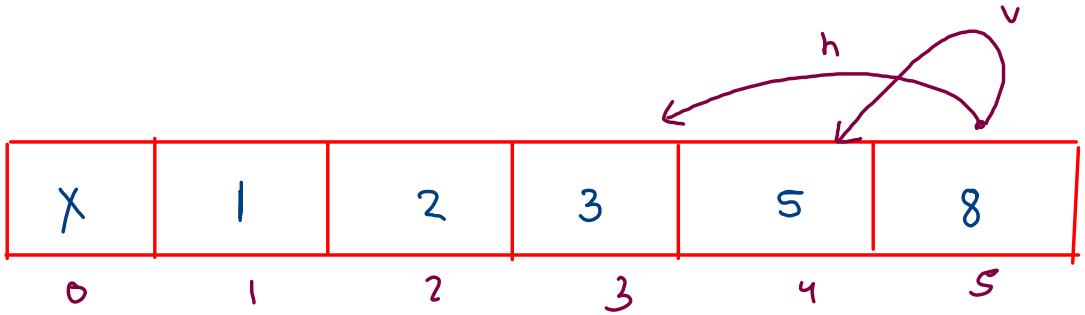


$f(n) = f(n-1) + f(n-2)$

n = 4

5 ways





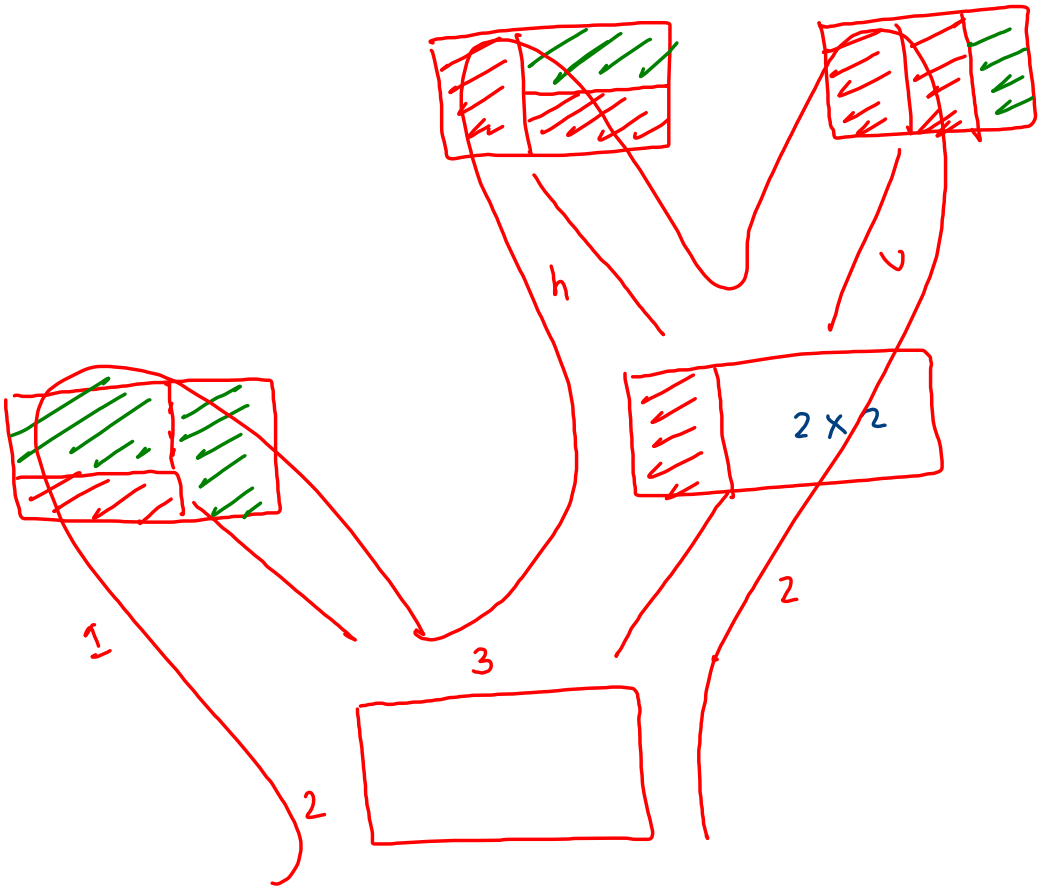
n = 5

$$f(n) = f(n-1) + f(n-2)$$

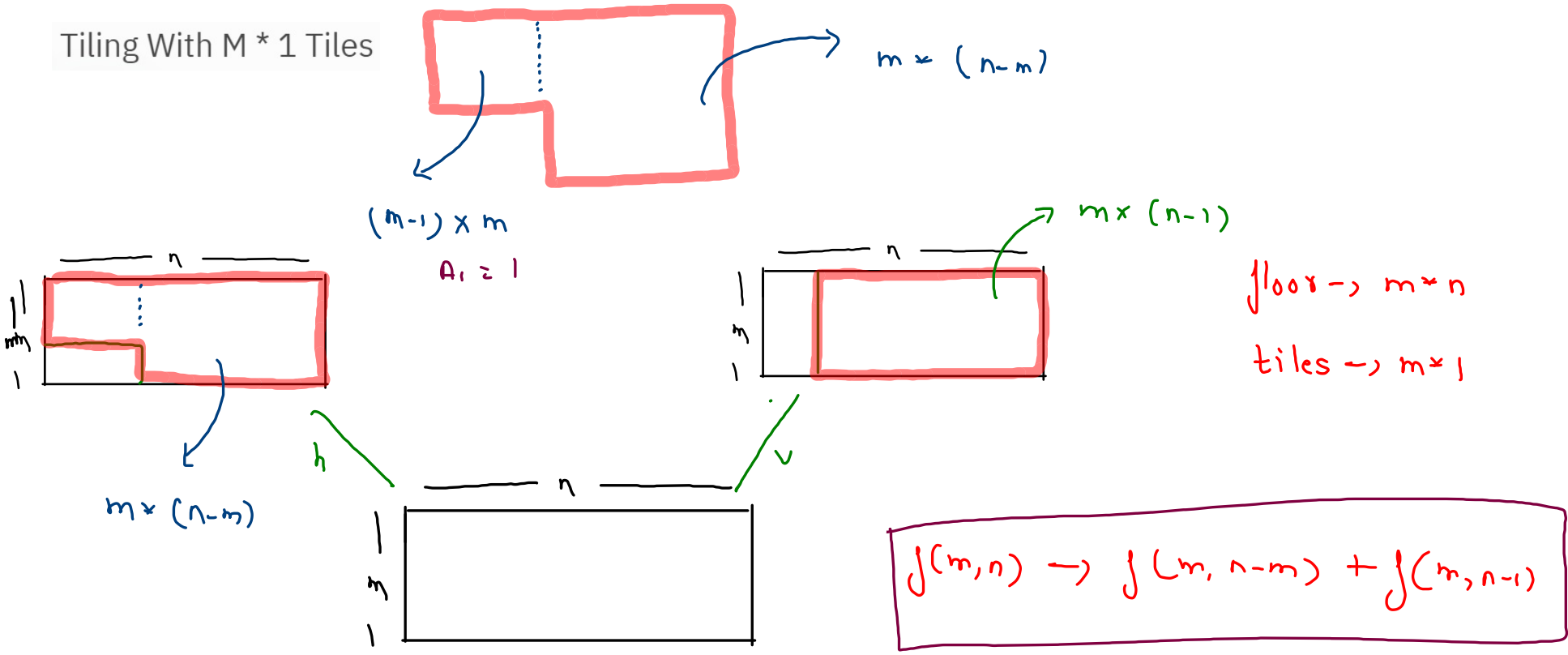
dp[i] —> ways to

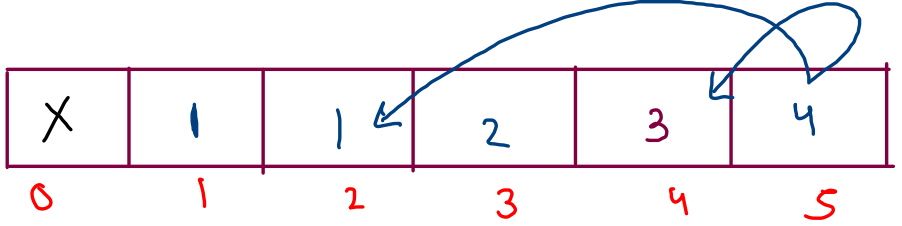
tile - up

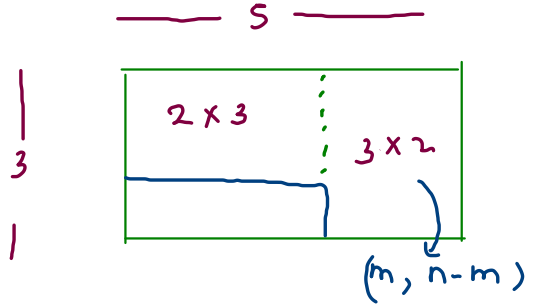
2 * i area

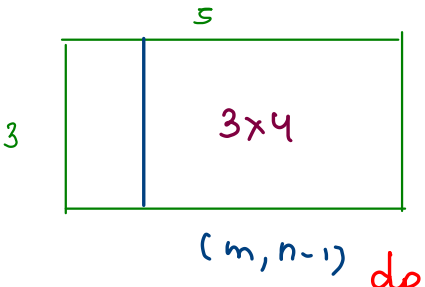

h
v
2 X 2
1
3
2
2

Tiling With M * 1 Tiles





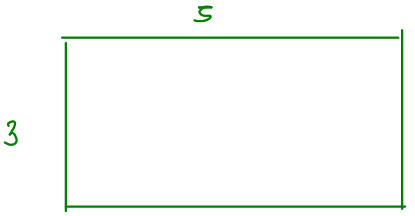




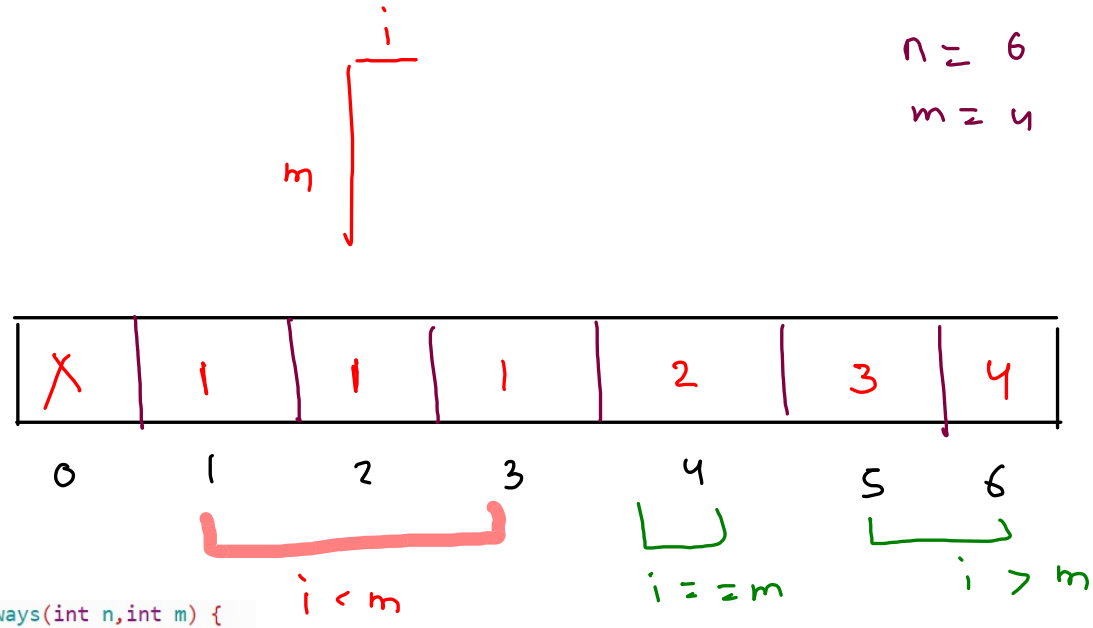
m = 3

n = 5

dp[i] =



dp[i] -> m x i area ways.



dp[i] -> m*i area ways

```
public static int tiling_ways(int n,int m) {
    int[]dp = new int[n+1];

    //dp[i] -> m*i area ways

    for(int i=1; i <= n;i++) {
        if(i < m) {
            dp[i] = 1;
        }
        else if(i == m) {
            dp[i] = 2;
        }
        else {
            dp[i] = dp[i-1] + dp[i-m];
        }
    }

    return dp[n];
}
```

$$dp[i] = \begin{cases} dp[i-m] + dp[i-1] & (i > m) \end{cases}$$

Friends Pairing

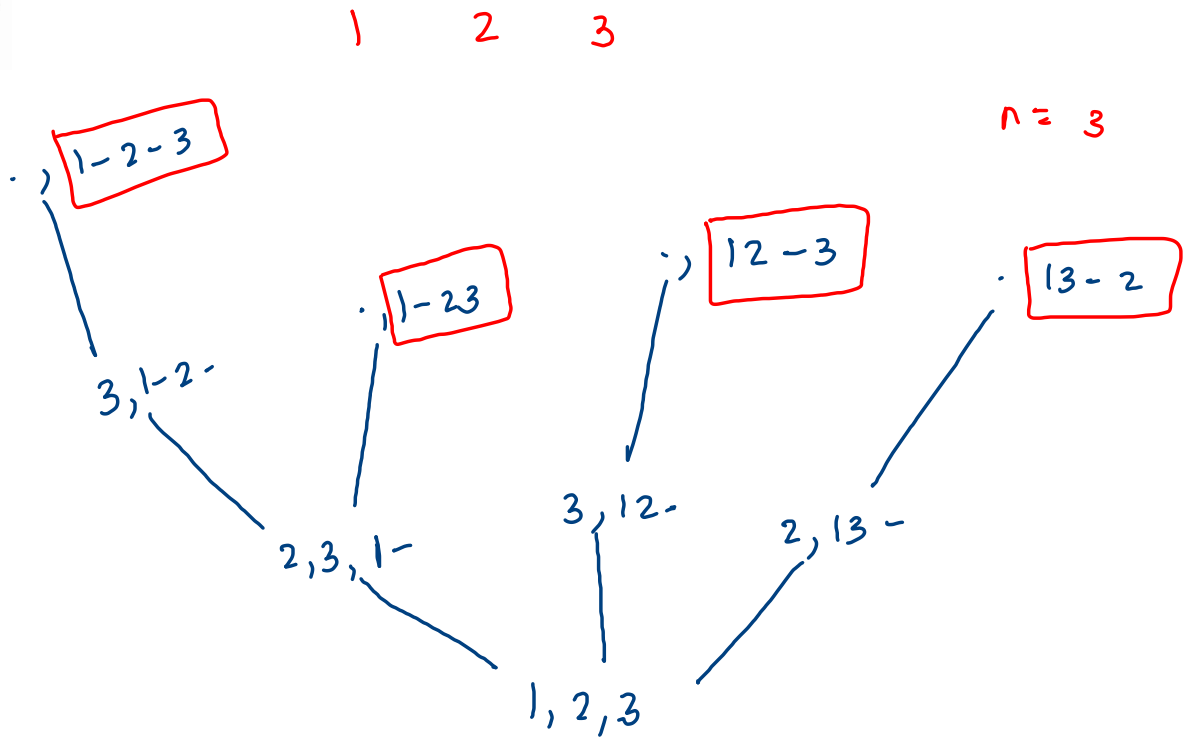
3 people (123) can stay singles or pair up in 4 ways. 123 => 1-2-3, 12-3, 13-2, 23-1.

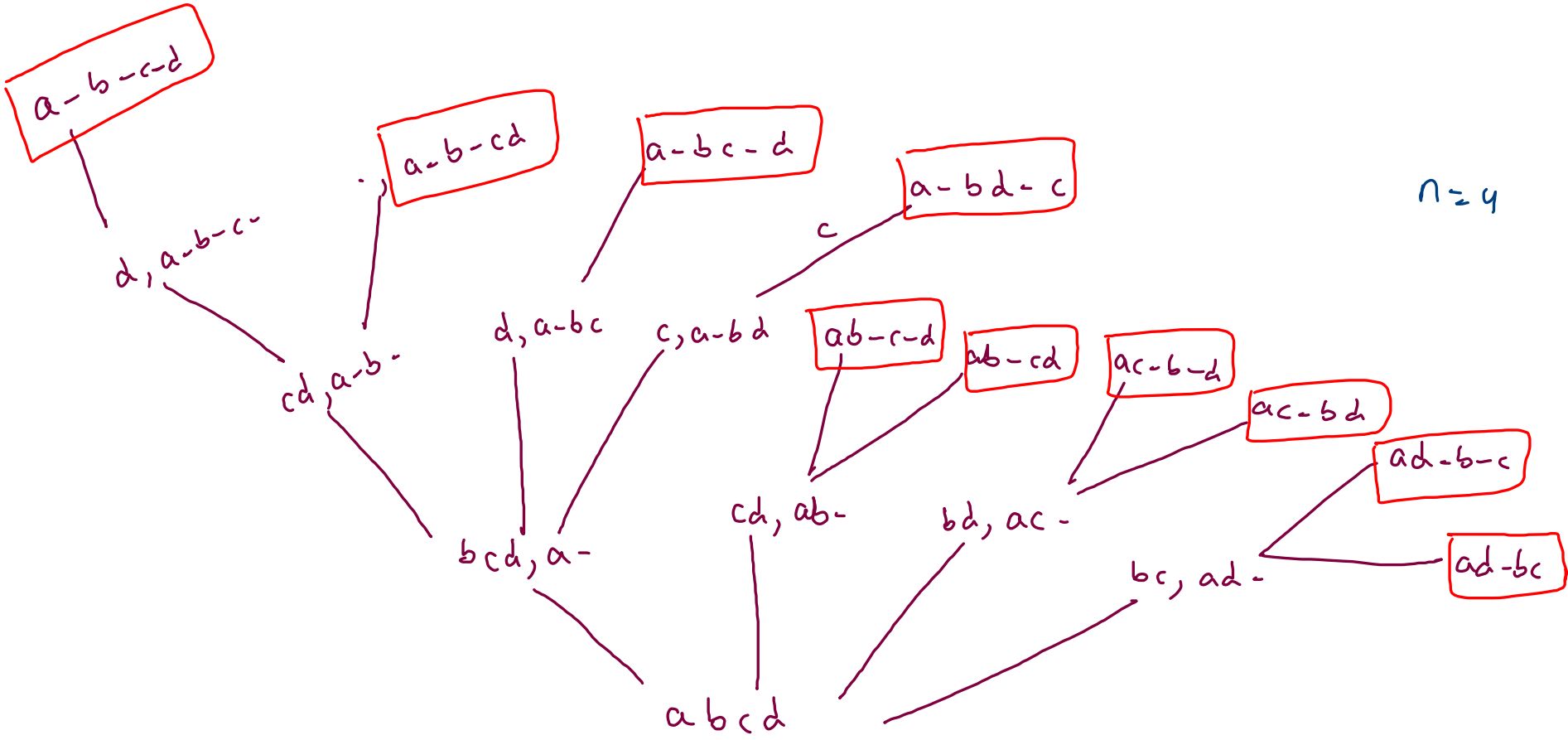
1 - 2 - 3

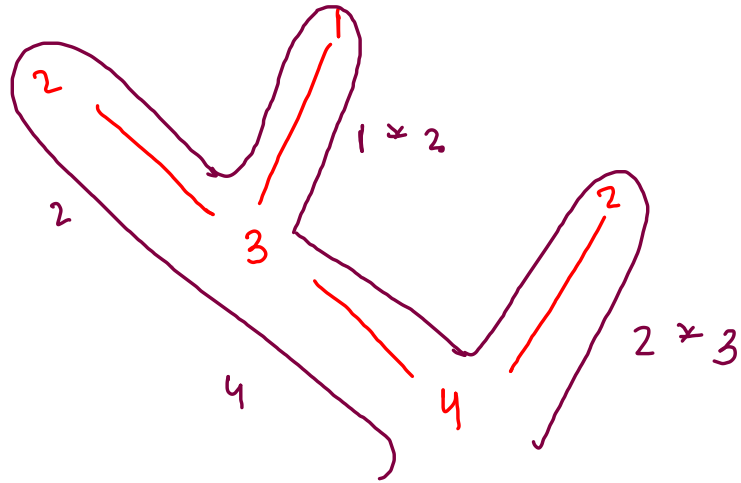
1 2 - 3

1 3 - 2

2 3 - 1



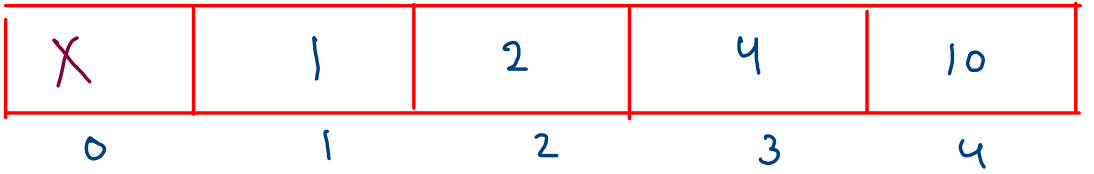

a-b-c-d
d, a-b-c-
a-b-cd
cd, a-b-
a-bc-d
d, a-bc
a-bd-c
c
c, a-bd
ab-c-d
ab-cd
ac-b-d
ac-bd
ad-b-c
ad-bc
cd, ab-
bd, ac-
bc, ad-
bcd, a-
abcd
n=4



$$dp[n] = dp[n-1] + (n-1) * dp[n-2]$$

single pairing

n = 4



dp[i] = dp[i-1] + (i-1)*dp[i-2];

a-b-c-d

a-bc-d

a-bd-c

a-cd-b

2 [ab-c-d
    ab-cd

2 [ac-b-d
    ac-bd

2 [ad-b-c
    ad-bc

Partition Into Subsets

$n = 4$

$k = 3$

ad b c

a bd c

a b cd

ac b d

a bc d

ab c d

$n = 4$

$k = 2$

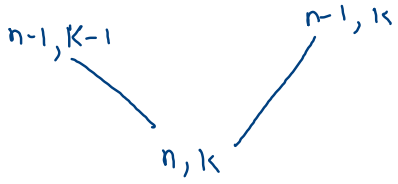
abc d

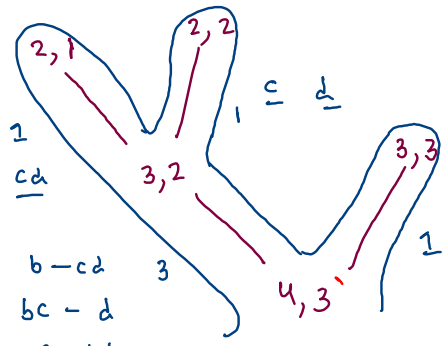
ab cd

a bcd

⋮

2 d n
k





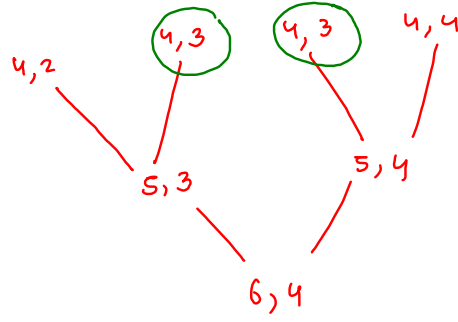
b - cd
bc - d
c - bd

a - b - cd
a - bc - d
a - c - bd

b - c - d

ab - c - d
b - ac - d
b - c - ad

n = 4
k = 3



dp

$$f(n,k) \rightarrow f(n-1,k-1) + k * f(n-1,k)$$

$$dp[i][j] = dp[i-1][j-1] + j * dp[i-1][j]$$

i elements

j sets

```
for(int i=0; i < dp.length;i++) {
    for(int j=0; j < dp[0].length;j++) {
        if(i == 0 || j == 0 || j > i) {
            dp[i][j] = 0;
        }
        else if(j == 1 || i == j) {
            dp[i][j] = 1;
        }
        else {
            dp[i][j] = dp[i-1][j-1] + j * dp[i-1][j];
        }
    }
}

return dp[n][k];
```

n \ k	0	1	2	3
0	0	0	0	0
1	0	1	0	0
2	0	1	1	0
3	0	1	3	1
4	0	1	7	6
5	0	1	15	25

n = 5

k = 3

i -> n

j -> k

(j == 1 || i == j)

dp[i][j] = 1;

(j > i)

dp[i][j] = 0;