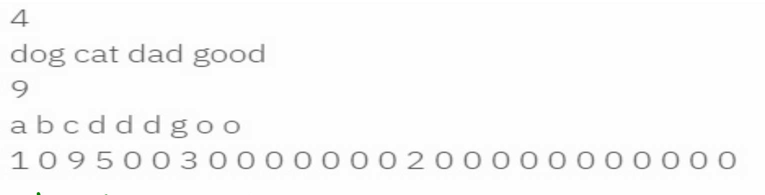
Max Score



```
4
dog cat dad good
9
a b c d d d g o o
1 0 9 5 0 0 3 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0
```

a b c d e f g h i j k l m n o p q r s t u v w x y z

arr: dog cat dad good

S1 : dog — dad —

score :

a – 1	1	
d – 3	15	21
o – 1	2	
g – 1	3	

S2 : — — dad good

d - 3	15	
o - 2	4	23
a - 1	1	
g - 1	3	

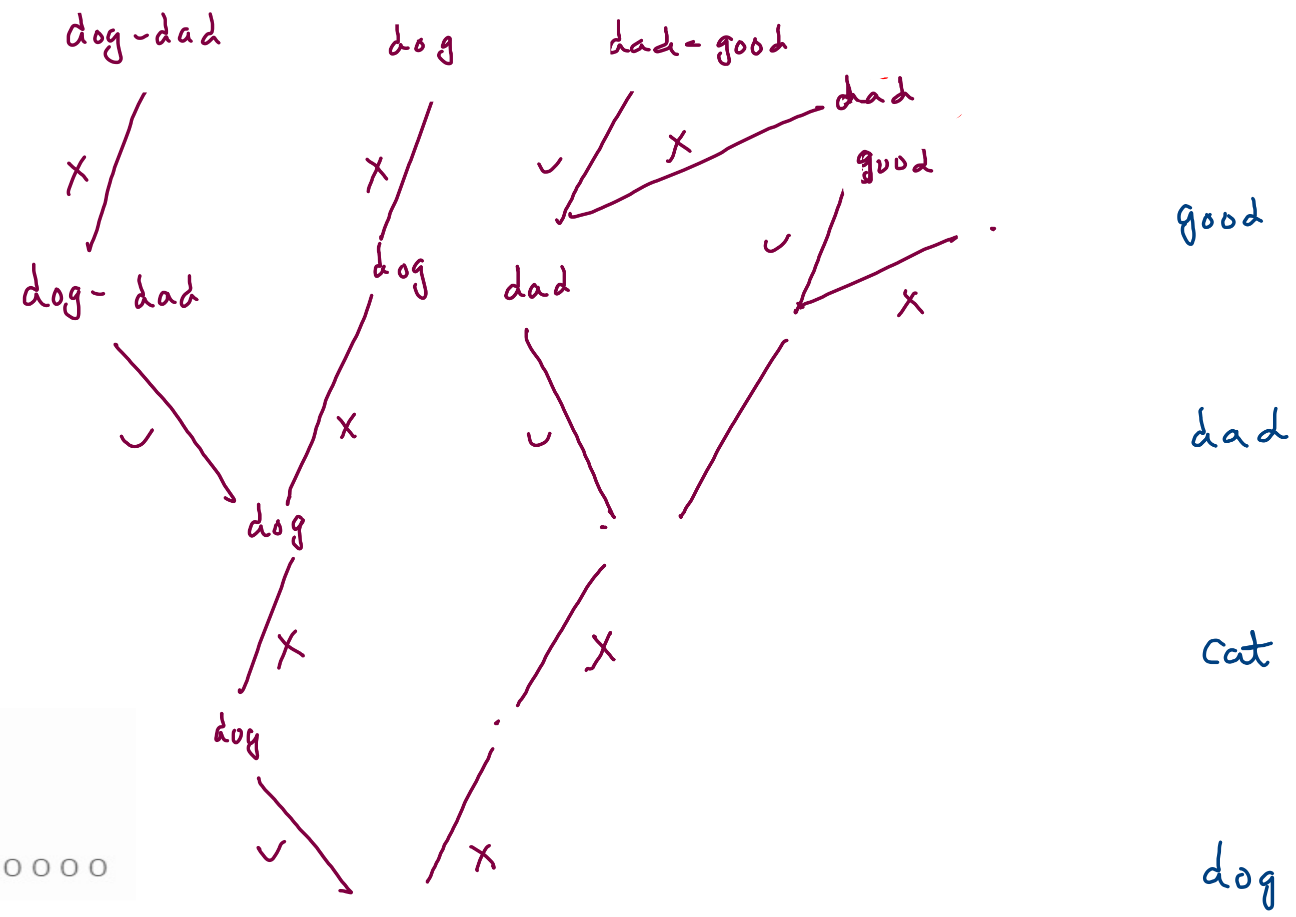
All subsets

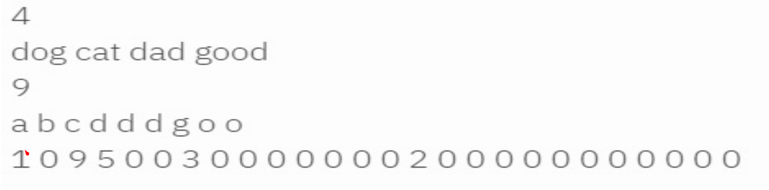
↓

count wise valid

↓

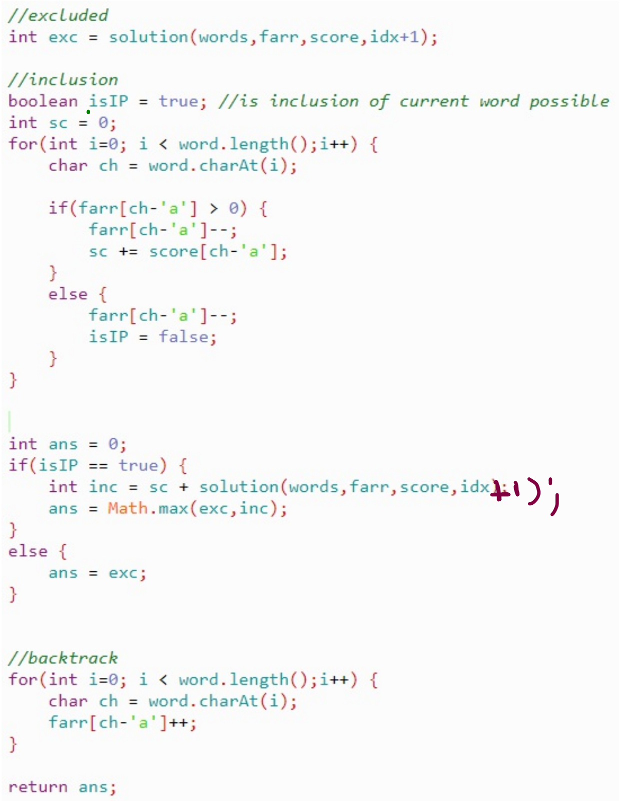
max score



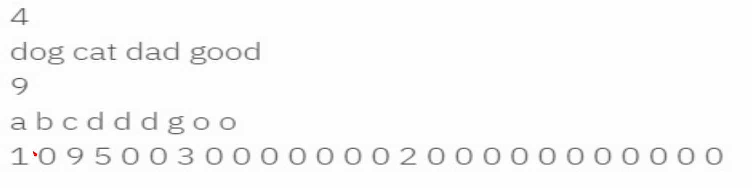


```
4
dog cat dad good
9
a b c d d d g o o
1 0 9 5 0 0 3 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0
```

a b c d e f g h i j k l m n o p q r s t u v w x y z



```
//excluded
int exc = solution(words,farr,score,idx+1);

//inclusion
boolean isIP = true; //is inclusion of current word possible
int sc = 0;
for(int i=0; i < word.length();i++) {
    char ch = word.charAt(i);

    if(farr[ch-'a'] > 0) {
        farr[ch-'a']--;
        sc += score[ch-'a'];
    }
    else {
        farr[ch-'a']--;
        isIP = false;
    }
}


int ans = 0;
if(isIP == true) {
    int inc = sc + solution(words,farr,score,idx+1);
    ans = Math.max(exc,inc);
}
else {
    ans = exc;
}


//backtrack
for(int i=0; i < word.length();i++) {
    char ch = word.charAt(i);
    farr[ch-'a']++;
}

return ans;
```



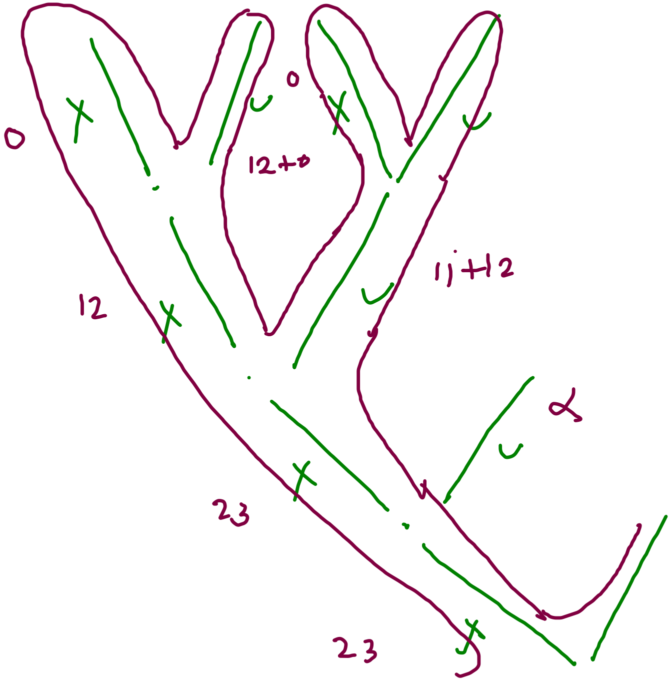
```
4
dog cat dad good
9
a b c d d d g o o
1 0 9 5 0 0 3 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0
```

a b c d e f g h i j k l m n o p q r s t u v w x y z

a - 1
b - 1
c - 1
d - 3
g - 1
o - 2
t - 0

isIP = ~~T~~ F

sc = 0 + 9 + 1

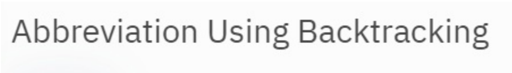


good

dad

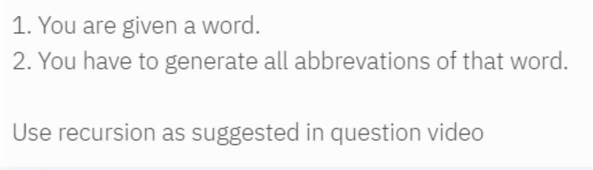
cat

dog



Abbreviation Using Backtracking

str : pep



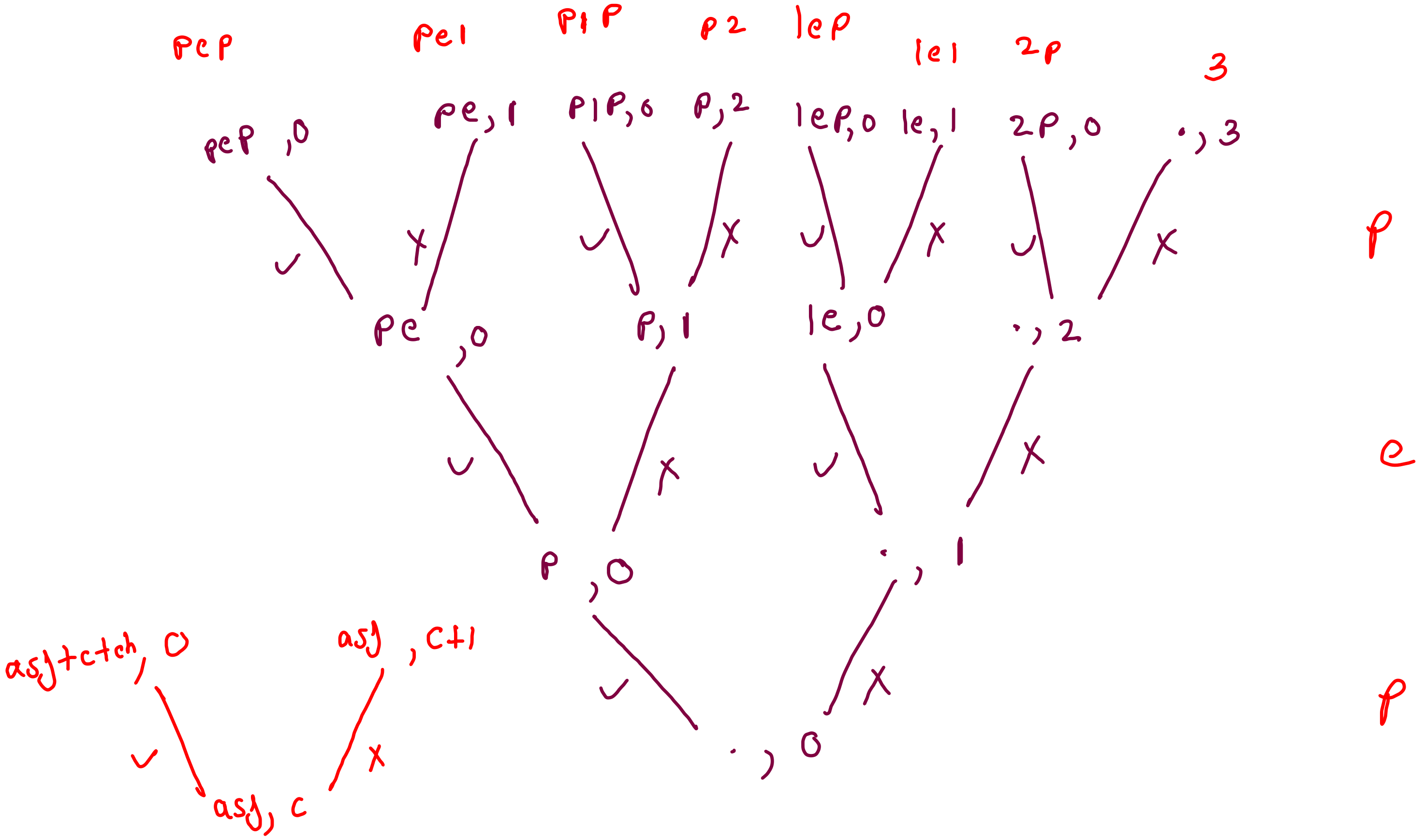
1. You are given a word.
2. You have to generate all abbrevations of that word.

Use recursion as suggested in question video

0 0 0	p e p
0 0 1	p e 1
0 1 0	p 1 p
0 1 1	p 2
1 0 0	1 e p
1 0 1	1 e 1
1 1 0	2 p
1 1 1	3

0 -> no replacement

1 -> replacement with no.



count

→ continous no.

of no calls

Lexicographical Numbers

1
10
100
1000
101
102
103
104
105
106
107
108
109

11
110
111
112
113
114
⋮
119

12
120
121
122
⋮
129

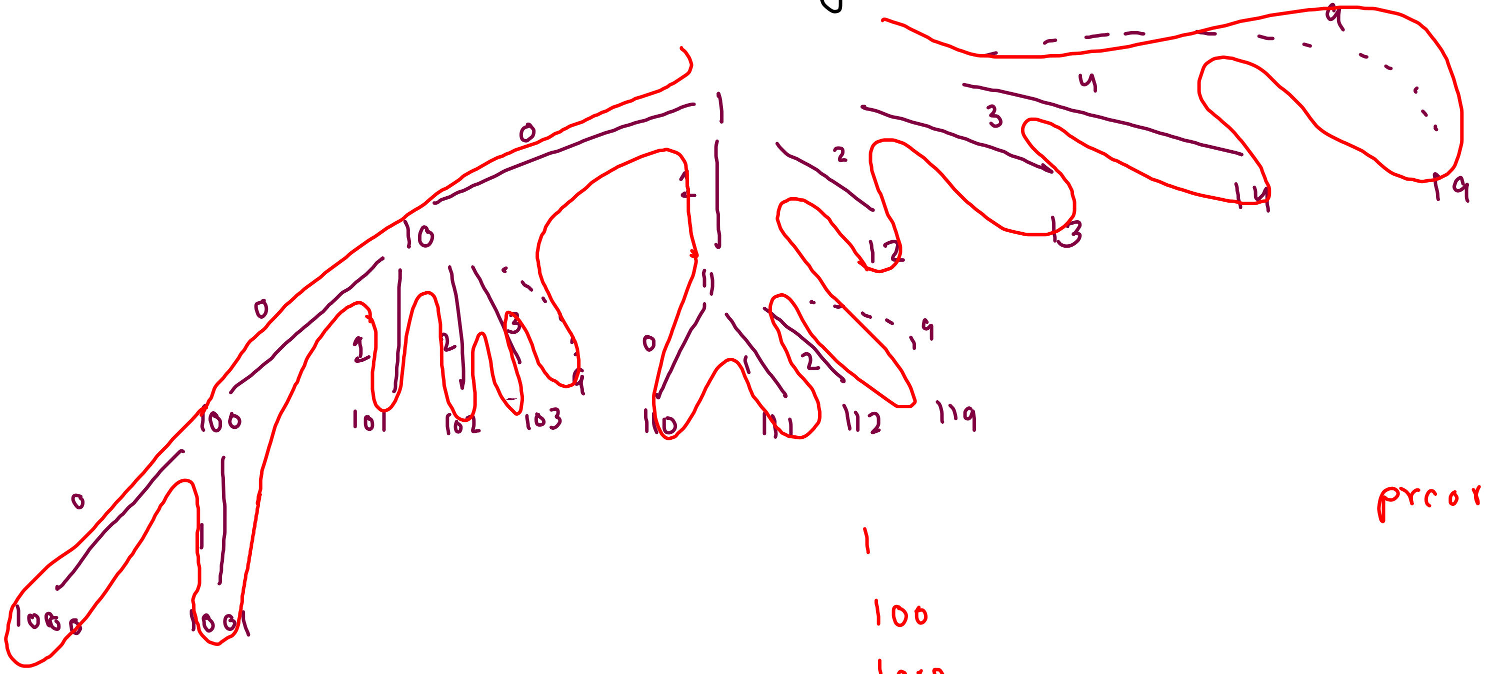
13
130 190
131
132
⋮
139

19
190
191
192
⋮
199

2
20
200
201
202
203

n = 1000

1' s Family



n = 1000

preorder

1

100

1000

101 to 109

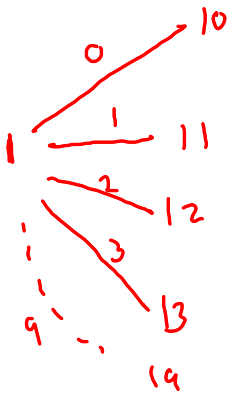
11

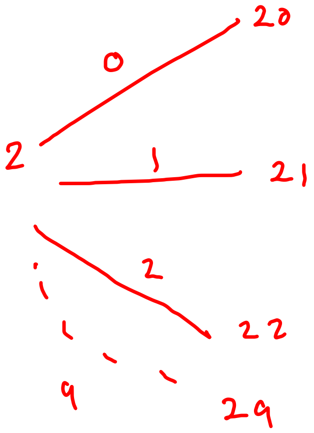
110 to 119

12

12 to 129

1's family   2's family .... 9's family

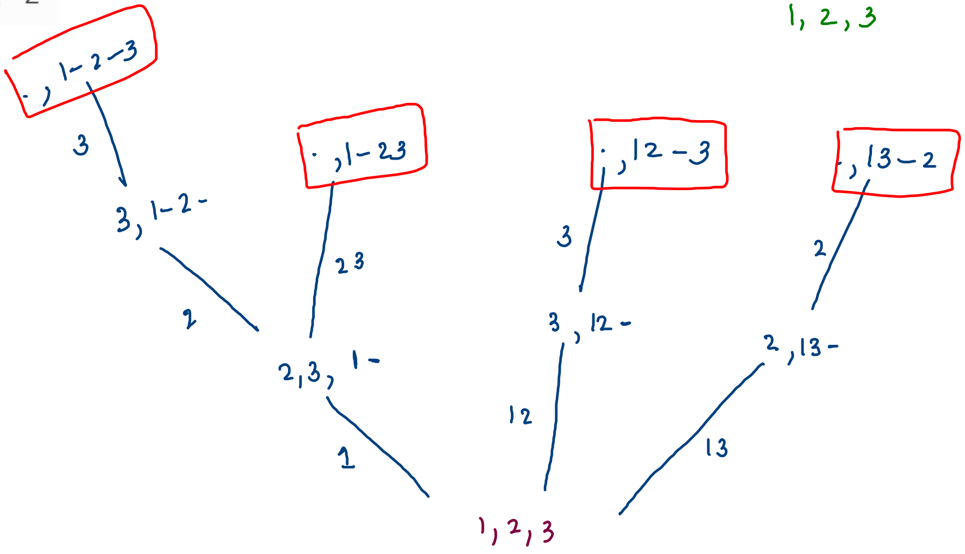




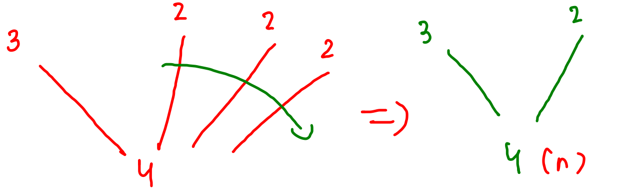
Friends Pairing - 2

n = 3

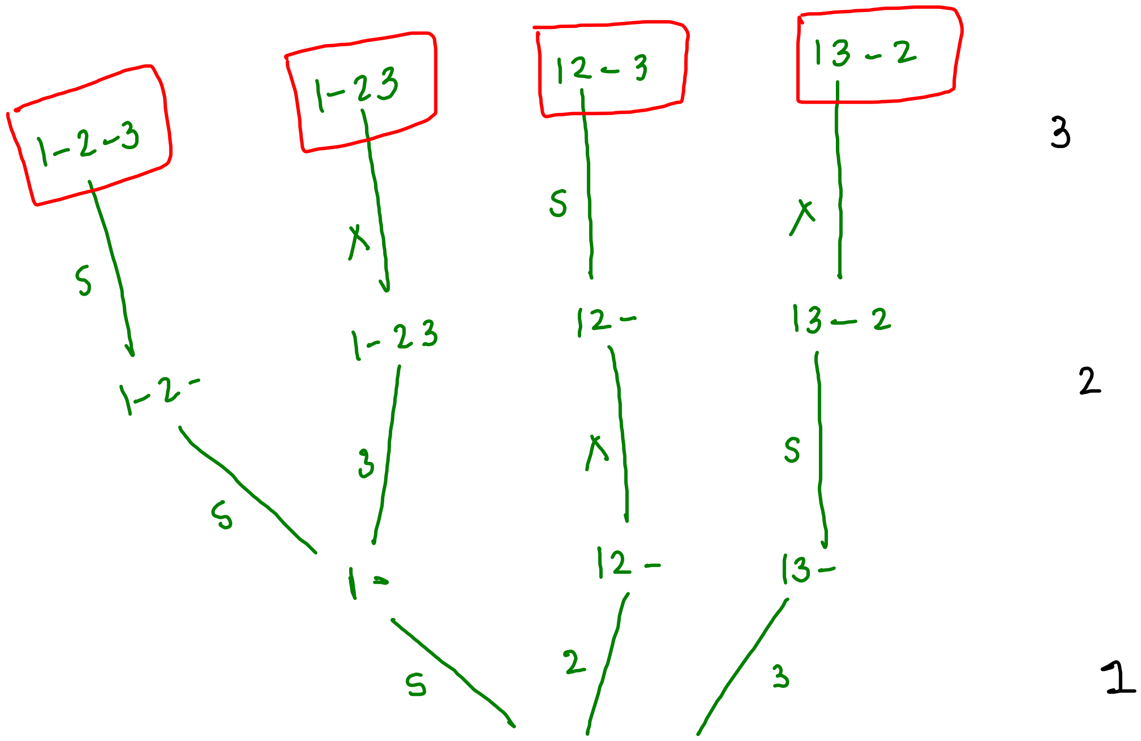
1, 2, 3



1.(1) (2) (3)
2.(1) (2,3)
3.(1,2) (3)
4.(1,3) (2)



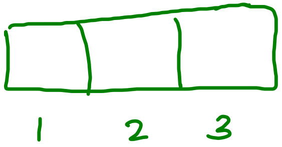
$$\boxed{dp[n-1] + (n-1)*dp[n-2]}$$

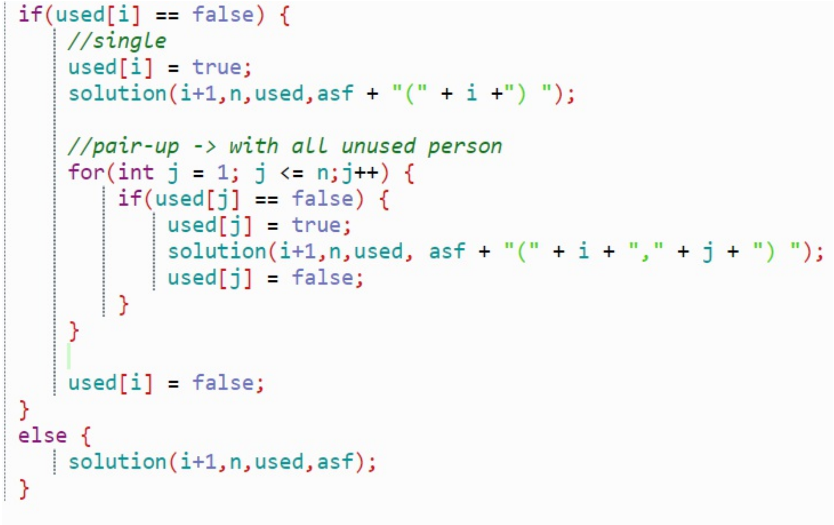


→ single

→ unused → pair

→ nothing



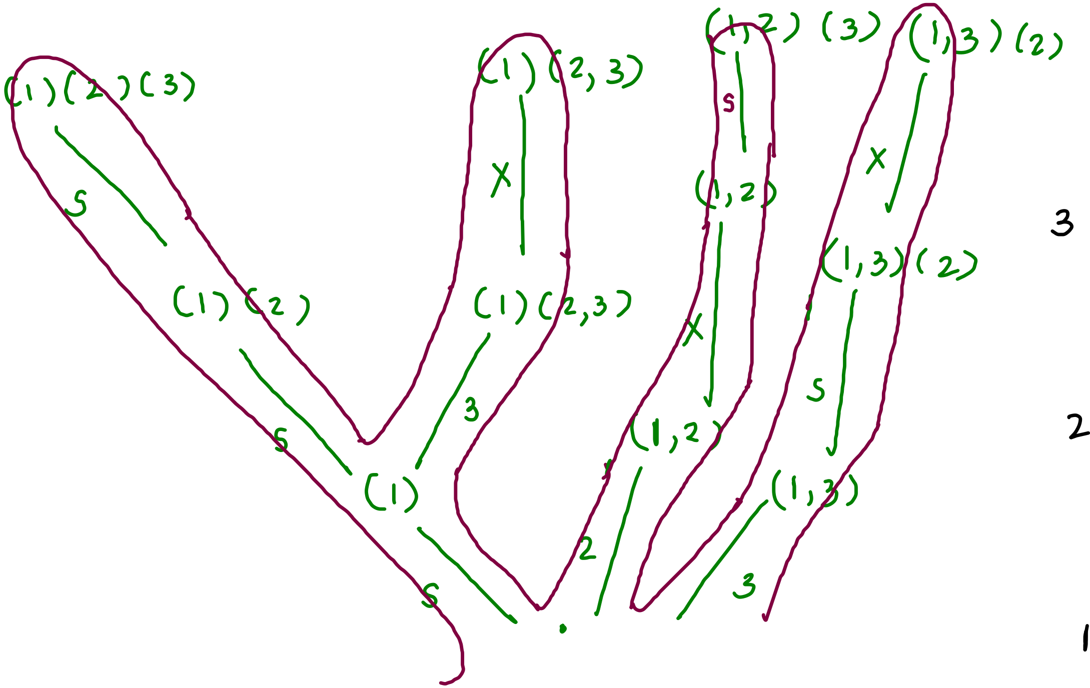


```
if(used[i] == false) {
    //single
    used[i] = true;
    solution(i+1,n,used,asf + "(" + i +") ");

    //pair-up -> with all unused person
    for(int j = 1; j <= n;j++) {
        if(used[j] == false) {
            used[j] = true;
            solution(i+1,n,used, asf + "(" + i + "," + j + ") ");
            used[j] = false;
        }
    }

    used[i] = false;
}
else {
    solution(i+1,n,used,asf);
}
```

(1) (2)(3)

(1) (2,3)

(1,2) (3)



K-partitions

n = 4 , k = 3

ad b c

a bd c

a b cd

ab c d

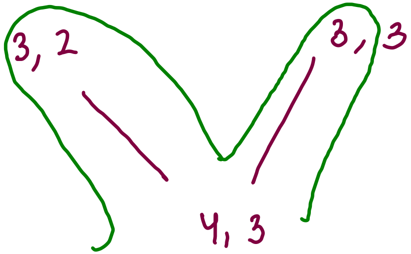
a cb d

ac b d

a b cd

a bc d

a bd c



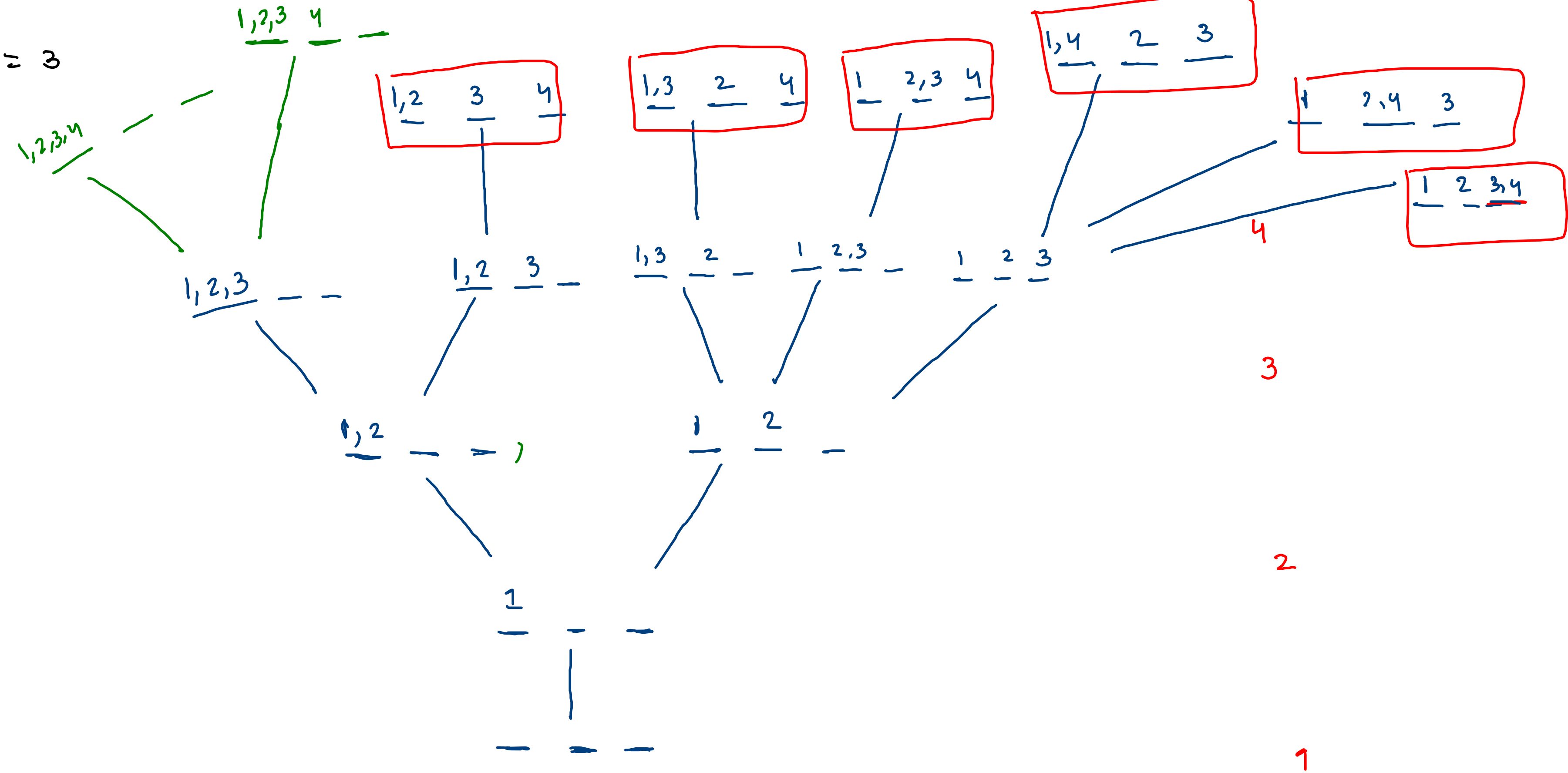
ab c d

b ac d

b c ad

$dp[n][k] = dp[n-1][k-1] + k * dp[n-1][k]$

4, K = 3



1,2,3 4 _
1,2,3,4
1,2 3 4
1,3 2 4
1 2,3 4
1,4 2 3
1 2,4 3
1 2 3,4
4
1,2,3 _ _
1,2 3 _
1,3 2 _
1 2,3 _
1 2 3
3
1,2 _ _
1 2 _
2
1 _ _
_ _ _
1