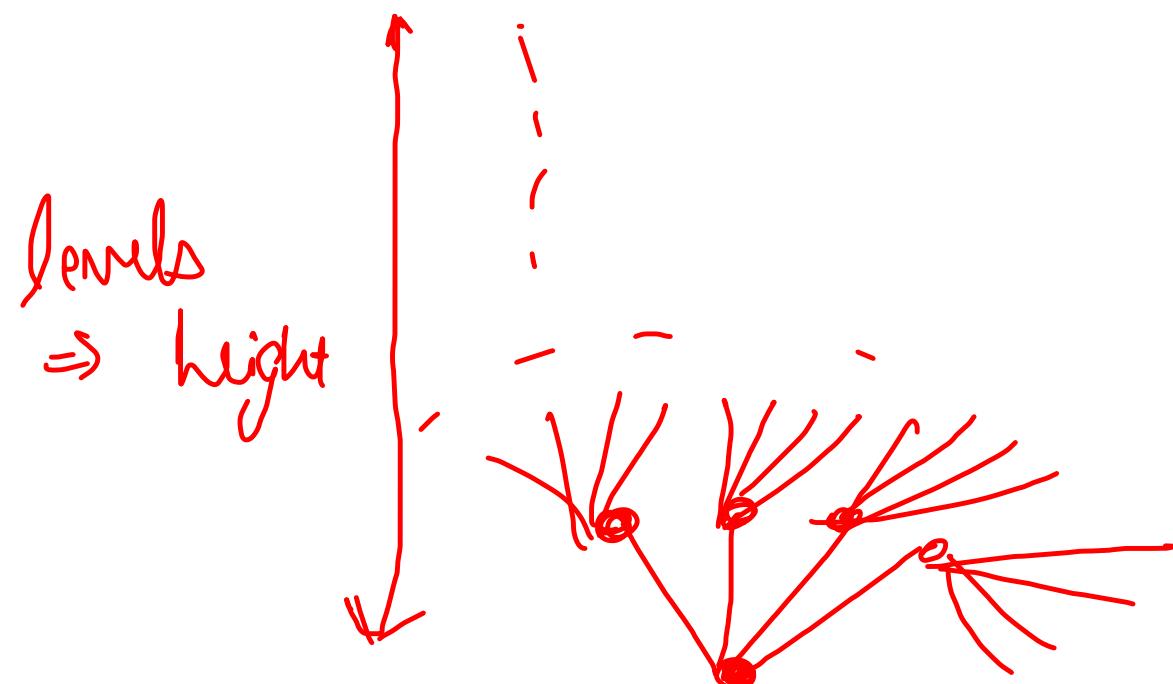


Recursion

- ① Expectation
- ② Faith
- ③ Meeting expectation with faith
- ④ Base Case

~~# Time Complexity ↴~~

(calls)^{height} + preorder * height
+ postorder * height



① Print Increasing

1.1 $\text{fun}(n) : 1 \dots n$

1.2 $\text{fun}(n-1) : 1 \dots (n-1)$

1.3 $M \cdot E \Rightarrow \text{Postorder} : \text{sys}(n);$

$n=0$: return;

$$\left\{ \begin{array}{l} TC \Rightarrow \\ (\underline{y})^n + k * n = \alpha n \end{array} \right.$$

② Print decreasing

1.1 $\text{fun}(n) : n \dots 1$

1.2 $\text{fun}(n-1) : (n-1) \dots 1$

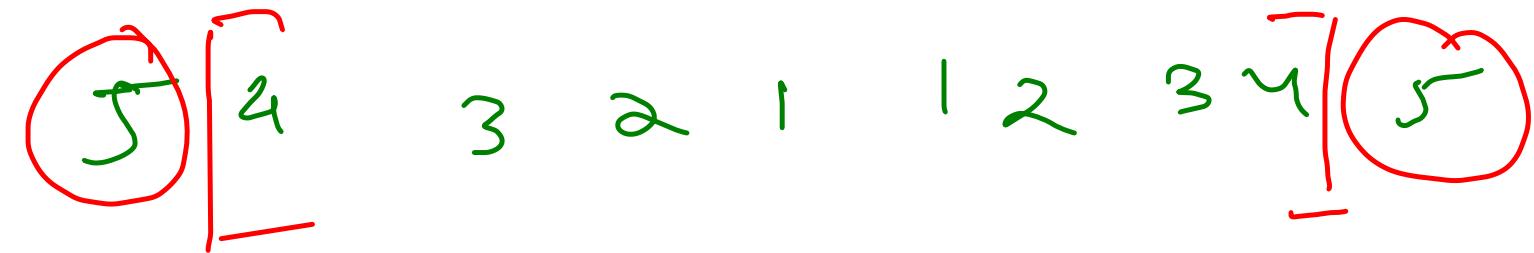
1.3 $M \cdot E \Rightarrow \text{Postorder} : \text{sys}(n);$

$n=0$: return;

C++
 m, n

③ Point Increasing Decreasing

fun(5) :



fun(4) :



M * E : \Rightarrow

Rec :
Sys(n)

fun(n-1)

Post
Sys(n)

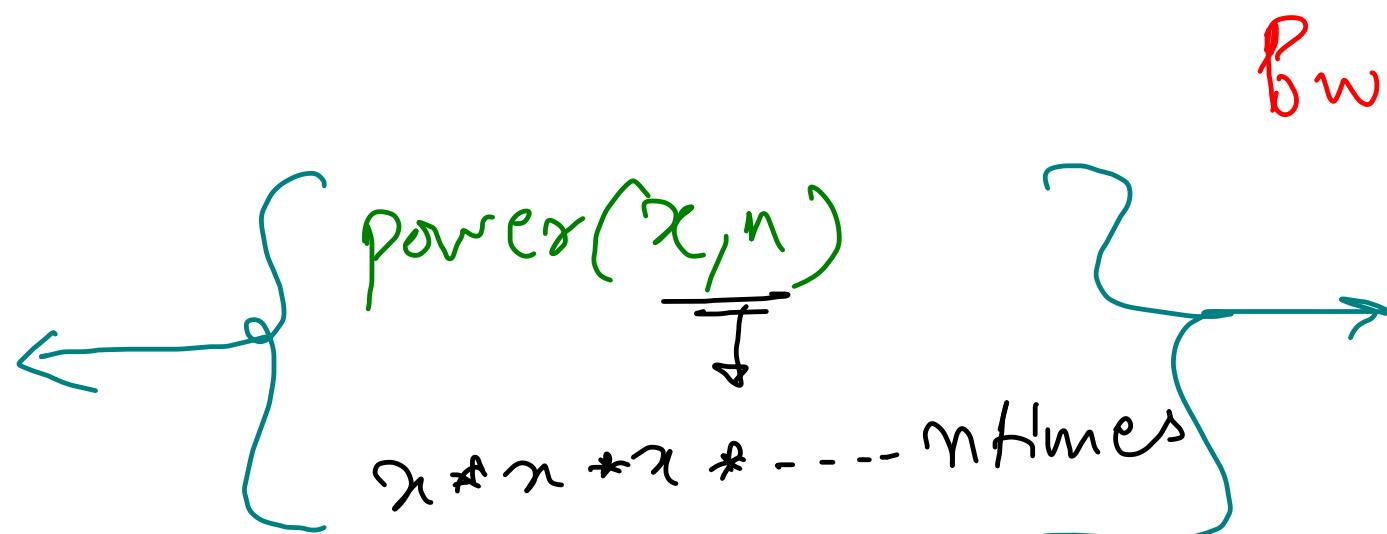
$$TC \Rightarrow \binom{n}{1} + k * n + k \neq n \\ = 2kn + 1 \Rightarrow O(n)$$

Power - linear

power($x, n-1$)

$$x^{n-1} = x * x * \dots \text{ (n-1) times}$$

return $x * x^{n-1}$



Power - logarithmic

$$x^{n/2} = \text{power}(x, n/2)$$

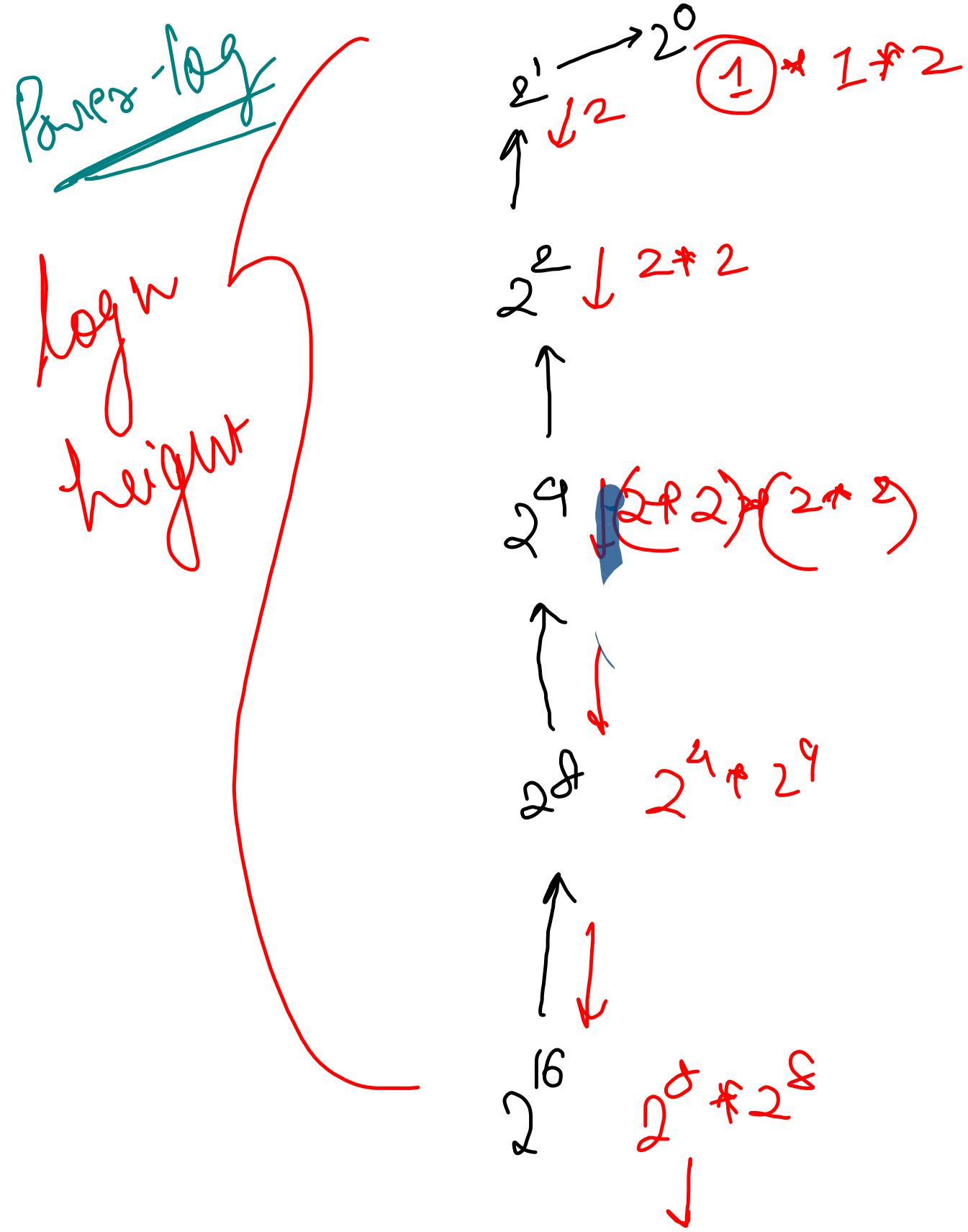
if ($n \neq 0$)
return $x^{n/2} * x^{n/2}$;

if ($n \neq 1$)
return $x^{n/2} * x^{n/2} + x^1$.

Base Case : $n = 0$; return 1;

$$\text{TC} \Rightarrow (1)^n + k * n \Rightarrow O(n)$$

$$\begin{aligned} \text{TC} &\Rightarrow (1)^{\log_2 n} + k * \log_2 n \\ &\Rightarrow O(\log n) \approx O(1) \end{aligned}$$



```

public static int power(int x, int n){
    // Base Case x^0 = 1
    if(n == 0) return 1;

    // 1. Faith : x ^ n/2
    int xpnb2 = power(x, n/2);

    // 2. x^n = x^n/2 * x^n/2
    int xpn = xpnb2 * xpnb2;

    // 3. If n is odd
    if(n % 2 == 1) xpn = xpn * x;

    // 4. Return
    return xpn;
}

```

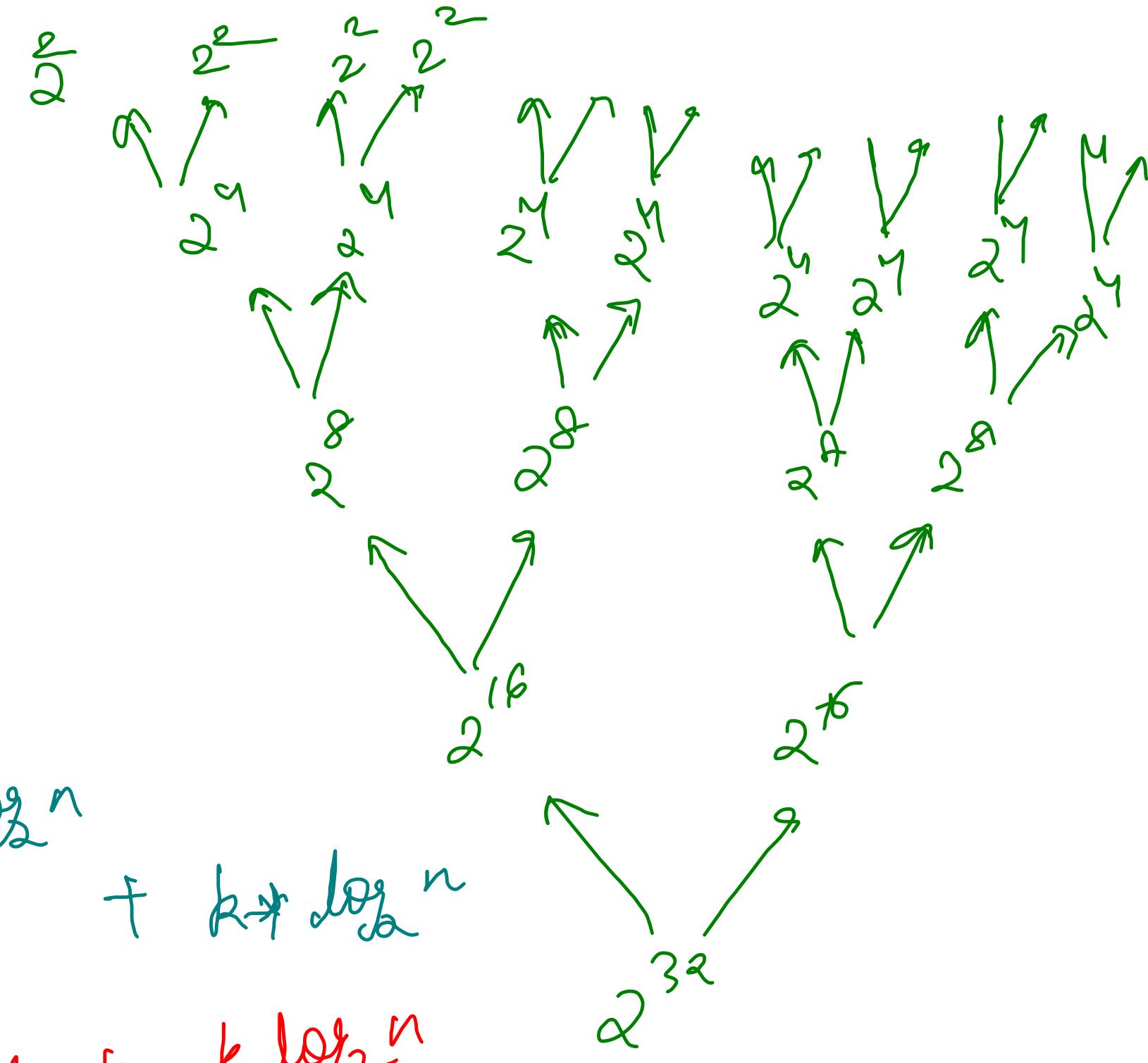
Power - 2

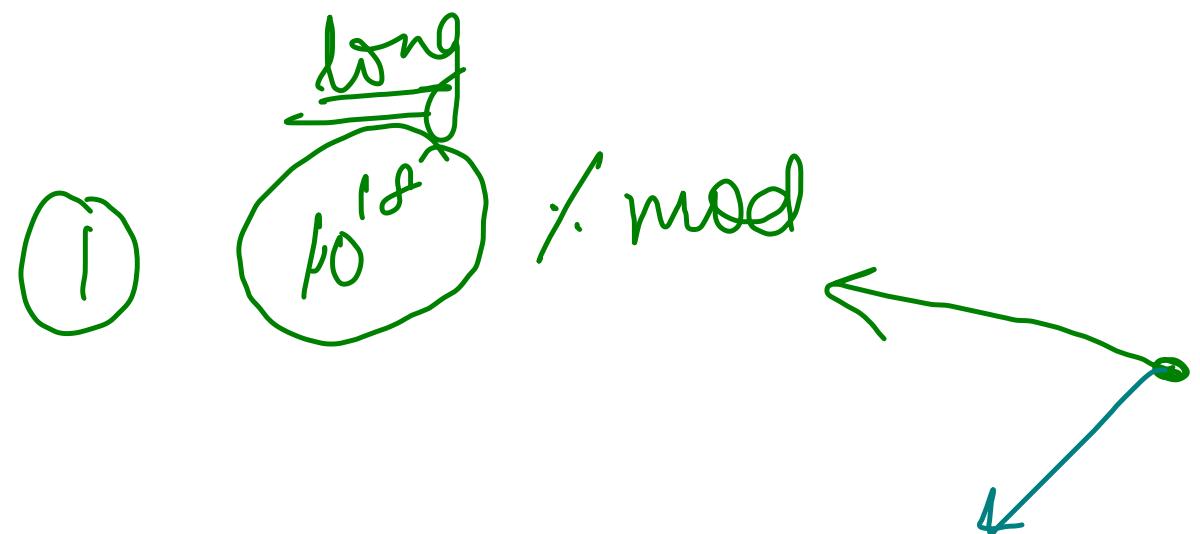
```
if (int x == 0)
    return power(x, n/2) * power(x, n/2)
else
    return pow(x, n/2) + pow(x, n/2) * x;
```

height $\Rightarrow \log_2 n$

calls $\Rightarrow 2$

$$(2)^{\log_2 n} + k \cdot \log_2 n = n + k \log_2 n \Rightarrow \alpha(n)$$





Because a, b
int max
value

$10^9 + 7$

$\text{long} \Rightarrow \approx 10^{18}$

$\text{int} \Rightarrow 2^{31} - 1 \approx 10^9$

$$\begin{aligned} 2 &\rightarrow (a+b)^r \cdot m \\ &[a+b \leq m] \\ &(a+b)^r \cdot m \end{aligned}$$

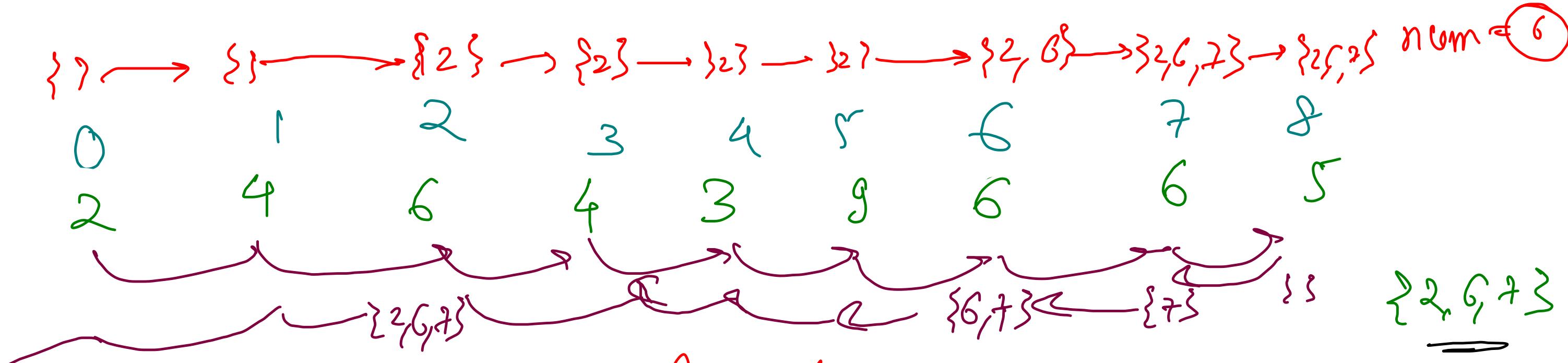
Modulus formulae

① $(a+b) \mod m = (a \mod m + b \mod m) \mod m$

② $(a-b) \mod m = ((a \mod m - b \mod m) \mod m + m) \mod m$

③ $(a * b) \mod m = ((a \mod m) * (b \mod m)) \mod m$

④ Division \Rightarrow Modulo Inverse

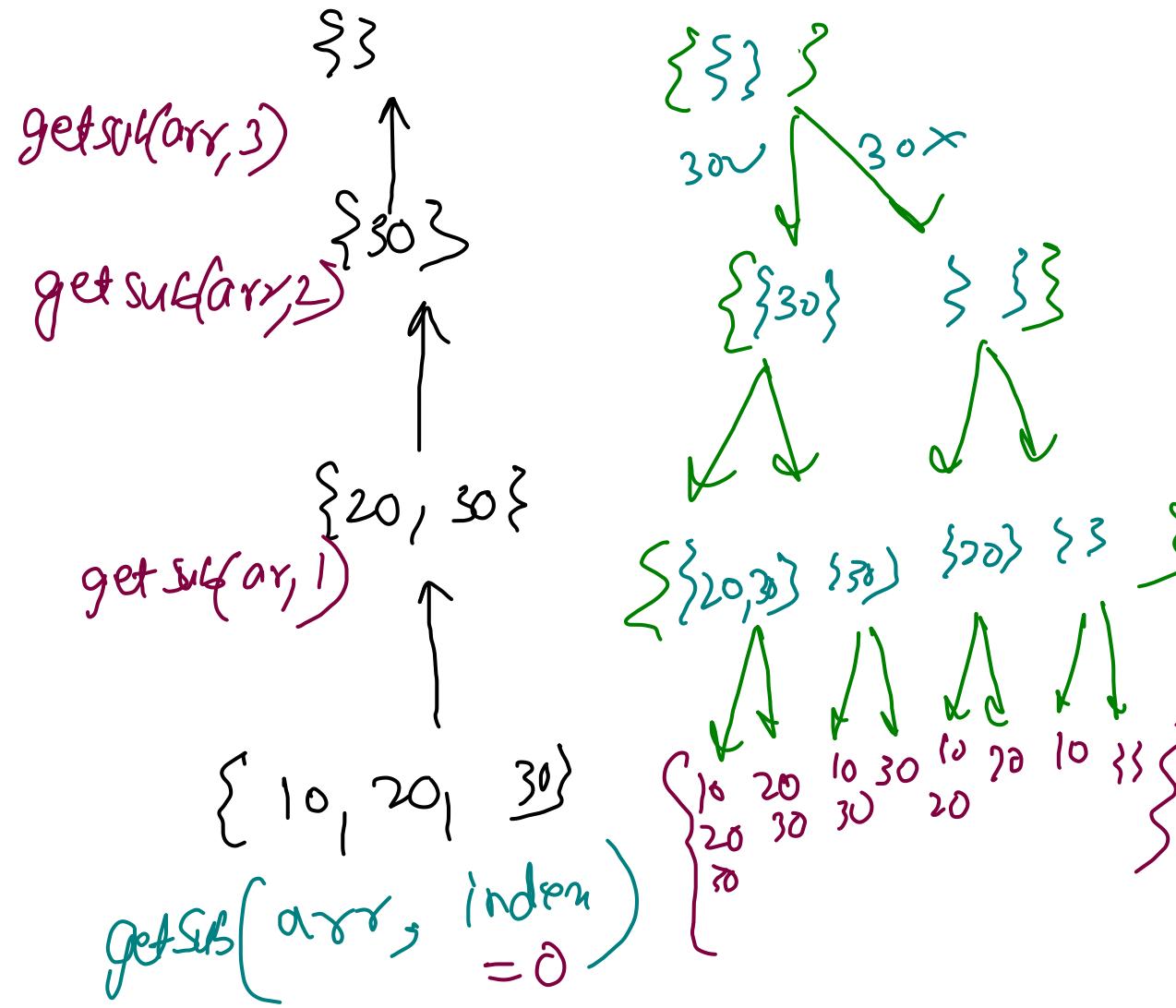


① preorder: append at last

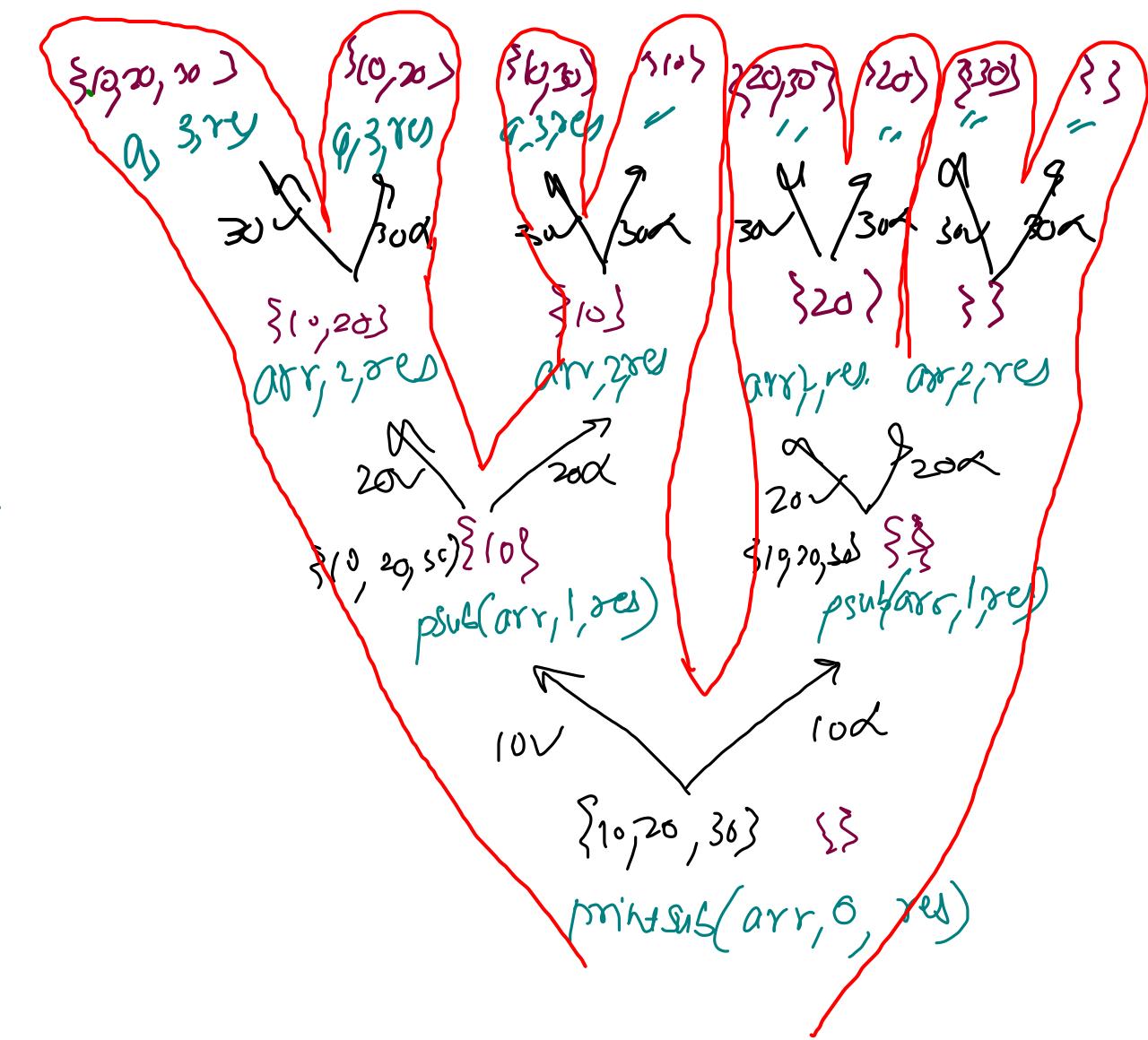
`fun(arr; o; s, n)`

→ ② postorder: append at first

Get Subsequence



Point Subsequence



$$TC = \left(2^n\right) + k \cdot n + k \cdot n \Rightarrow O(2^n)$$

Today's Target

①

Permutations

→ Box on level

→ Item on level

→ Swap method

②

Combinations

→ Box on level

→ Item on level

15-20
↳ follow UP } $\Rightarrow 30 \text{ &}$

③ String Permutations {Unique}

→ Box on level

→ Item on level

④

Palindromic Permutations

Homework

- Factor Combinations

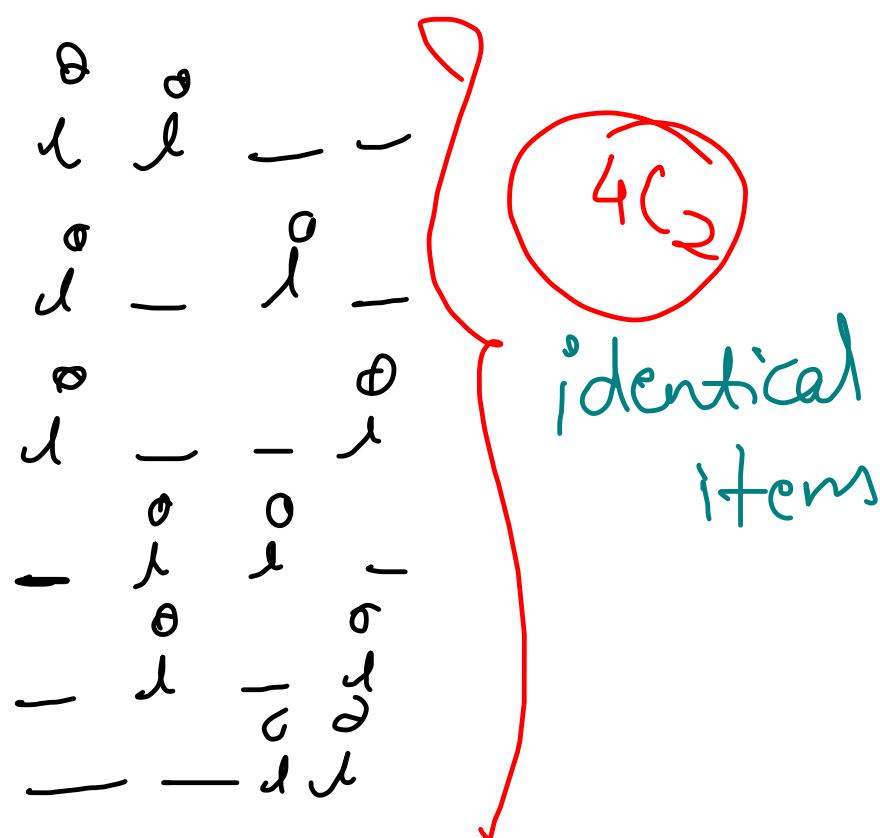
- Dictionary Order

↳ small
↳ large

$$nC_r$$

Combination

$$\frac{n!}{(n-r)! r!}$$



$$nP_r$$

Permutation
→ "select" + Arrangement

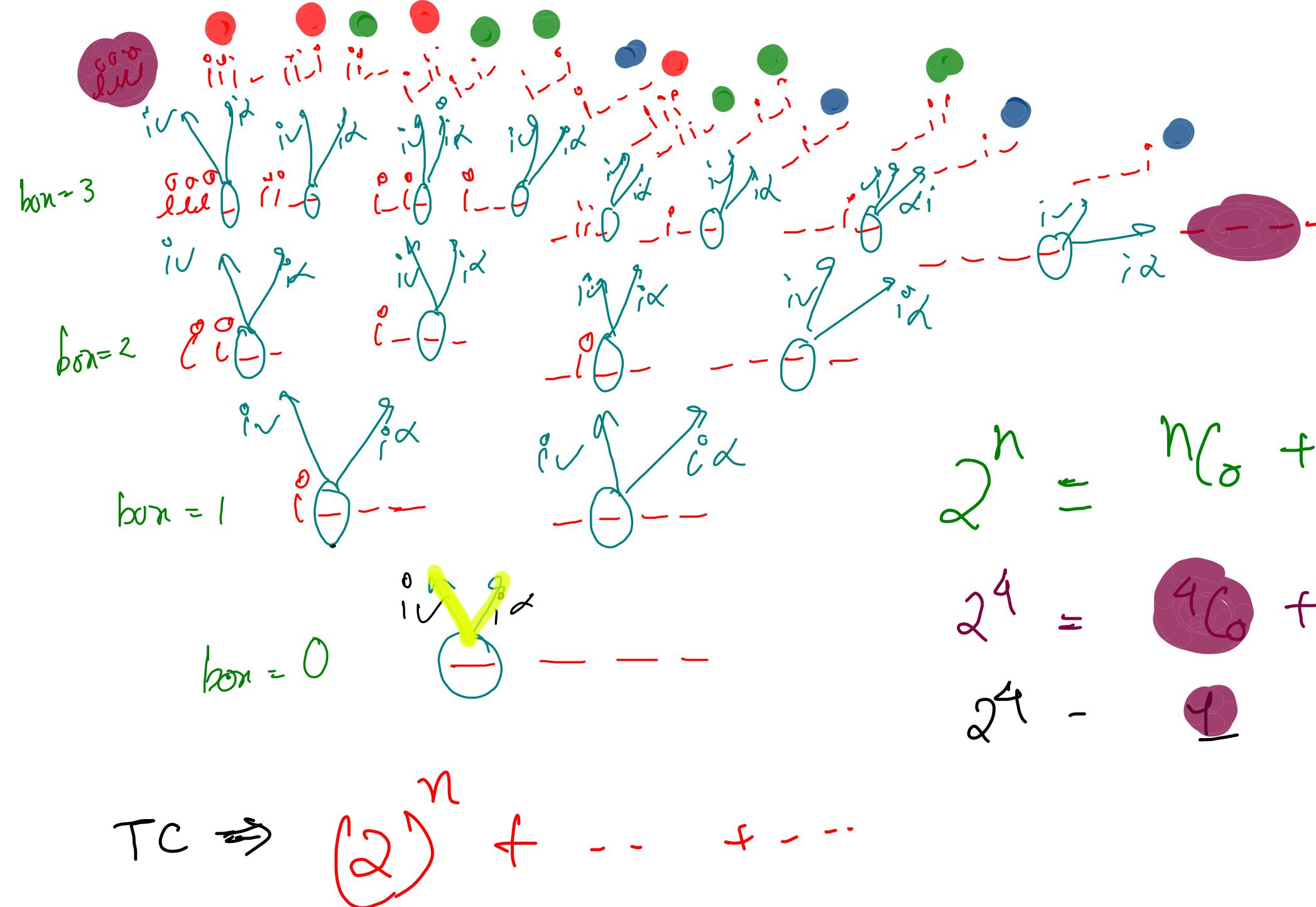
$$\frac{n!}{(n-r)!}$$

non-identical items

$$\begin{array}{cccc}
 12-- & 21-- & \\
 1-2- & 2-1- & \\
 1--2 & 2---1 & \\
 -12- & -21- & \\
 -1-2 & -2- & \\
 --12 & ---2 &
 \end{array}$$

$4P_2 = 4C_2 * 2!$

Combinations \rightarrow Box on level

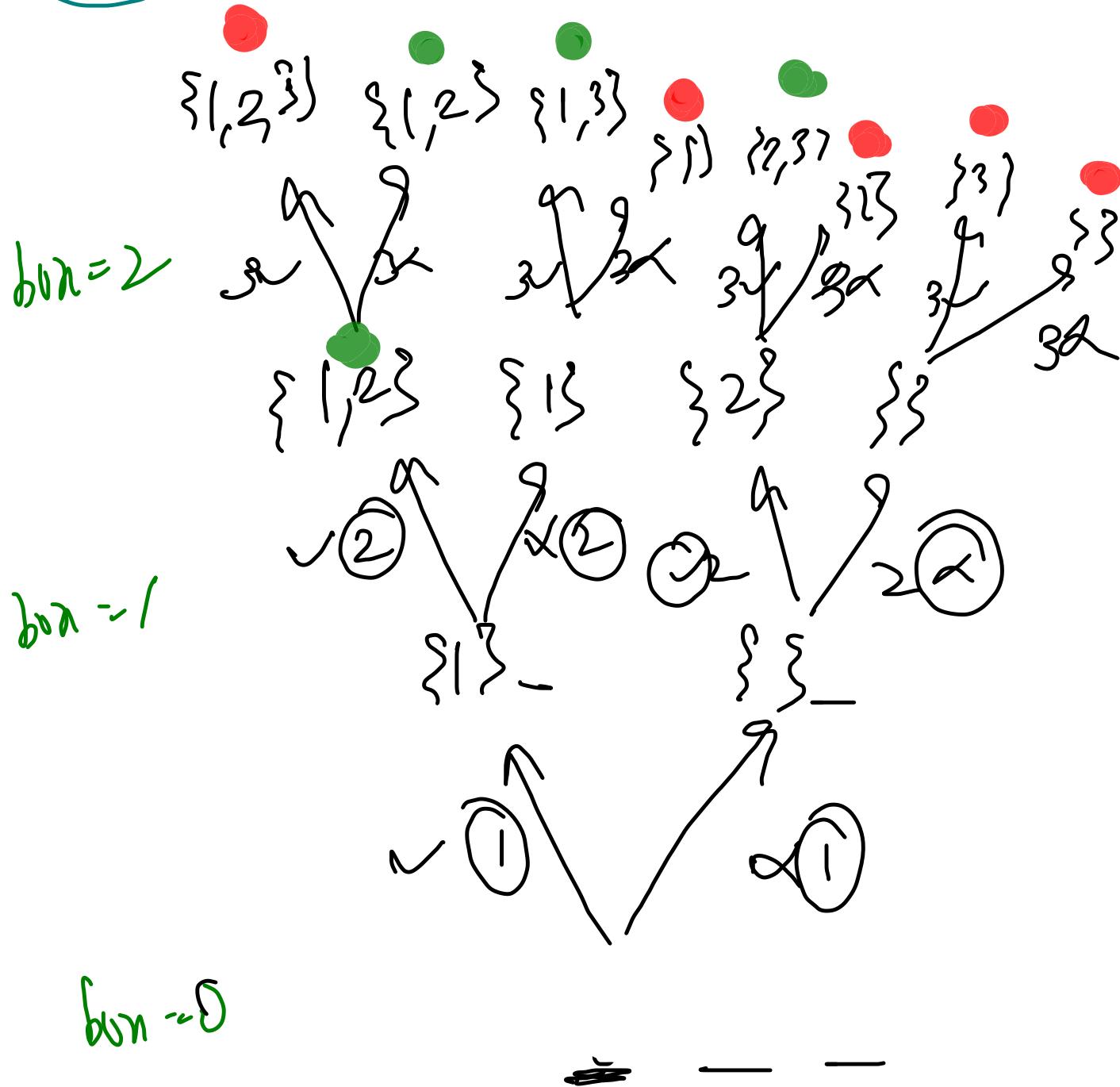


$$2^4 = 4C_0 + 4C_1 + 4C_2 + 4C_3 + 4C_4$$

$$2^4 = 1 + 4 + 6 + 4 + 1$$

TC \Rightarrow $(2)^n + \dots + \dots$

$$BC_2 = 3$$



```

public void combine(List<List<Integer>> combinations,
List<Integer> combination, int currentBox, int n, int k){
    if(currentBox == n){
        if(combination.size() == k){
            // deep copy
            List<Integer> temp = new ArrayList<>(combination);
            combinations.add(temp);
        }
        return;
    }

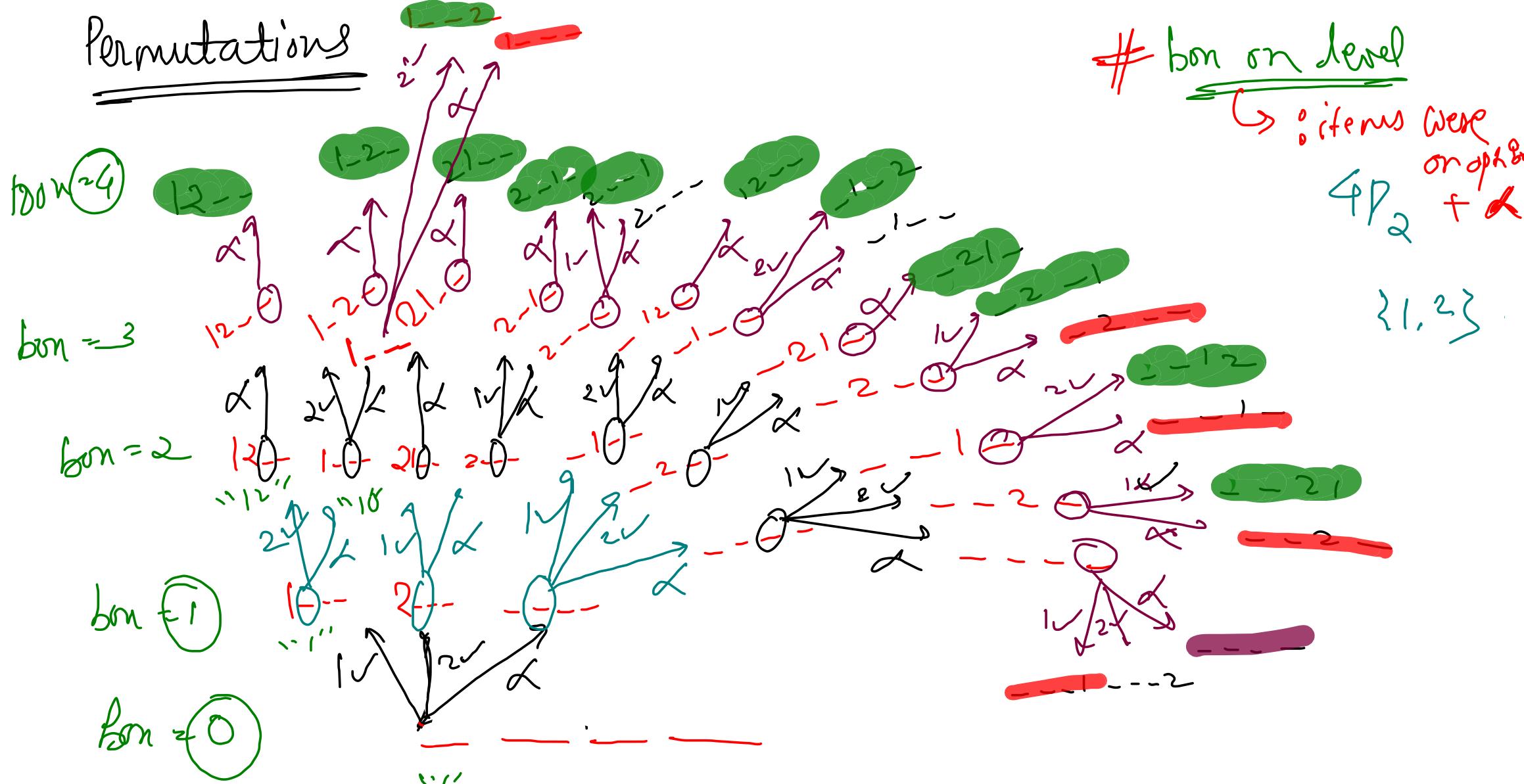
    // options -> current Box -> item should be placed or not
    // yes
    combination.add(currentBox + 1);
    combine(combinations, combination, currentBox + 1, n, k);
    combination.remove(combination.size() - 1);

    // no
    combine(combinations, combination, currentBox + 1, n, k);
}

```

} {1,2}, {1,3}, {2,3}

Permutations



bin on level

↪ % items were on shelves + ↪

$$4P_2$$

$$4P_0 + 4P_1 + 4P_2$$

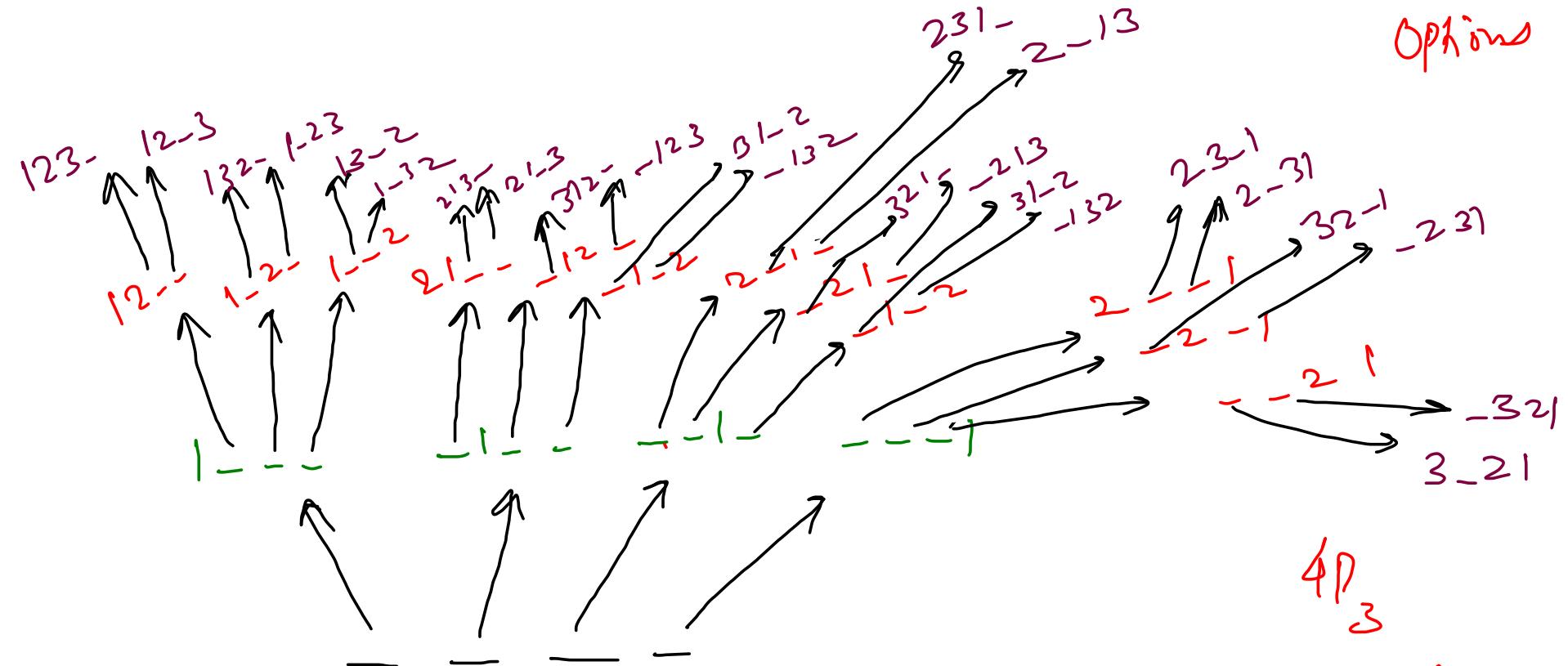
~~# Permutation < 2~~

Items on level \circ boxes
Options

\circ item = 3

\circ item = 2

\circ item = 1



```
public static void permutations(int[] boxes, int ci, int ti){  
    // write your code here  
    if(ci == ti){  
        for(int val: boxes) System.out.print(val);  
        System.out.println();  
    }  
  
    // Item -> Choose box  
    for(int i=0; i<boxes.length; i++){  
        if(boxes[i] == 0){  
            boxes[i] = ci + 1;  
            permutations(boxes, ci + 1, ti);  
            boxes[i] = 0;  
        }  
    }  
}
```

$4P_3$

$n=4$

$A = 3$

Combinations -2

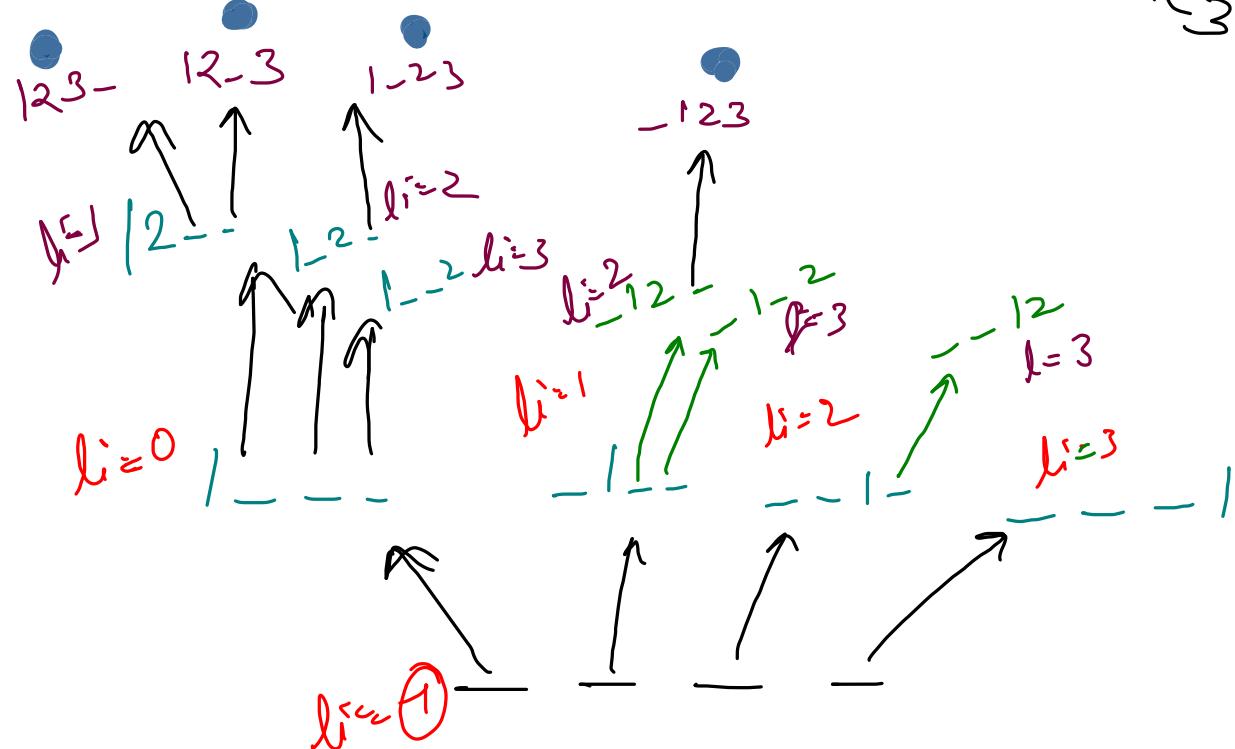
{ Items \rightarrow level ; Option \hookrightarrow box }

item = 3

item = 2

item = 1

$${}^4C_3 = 4$$

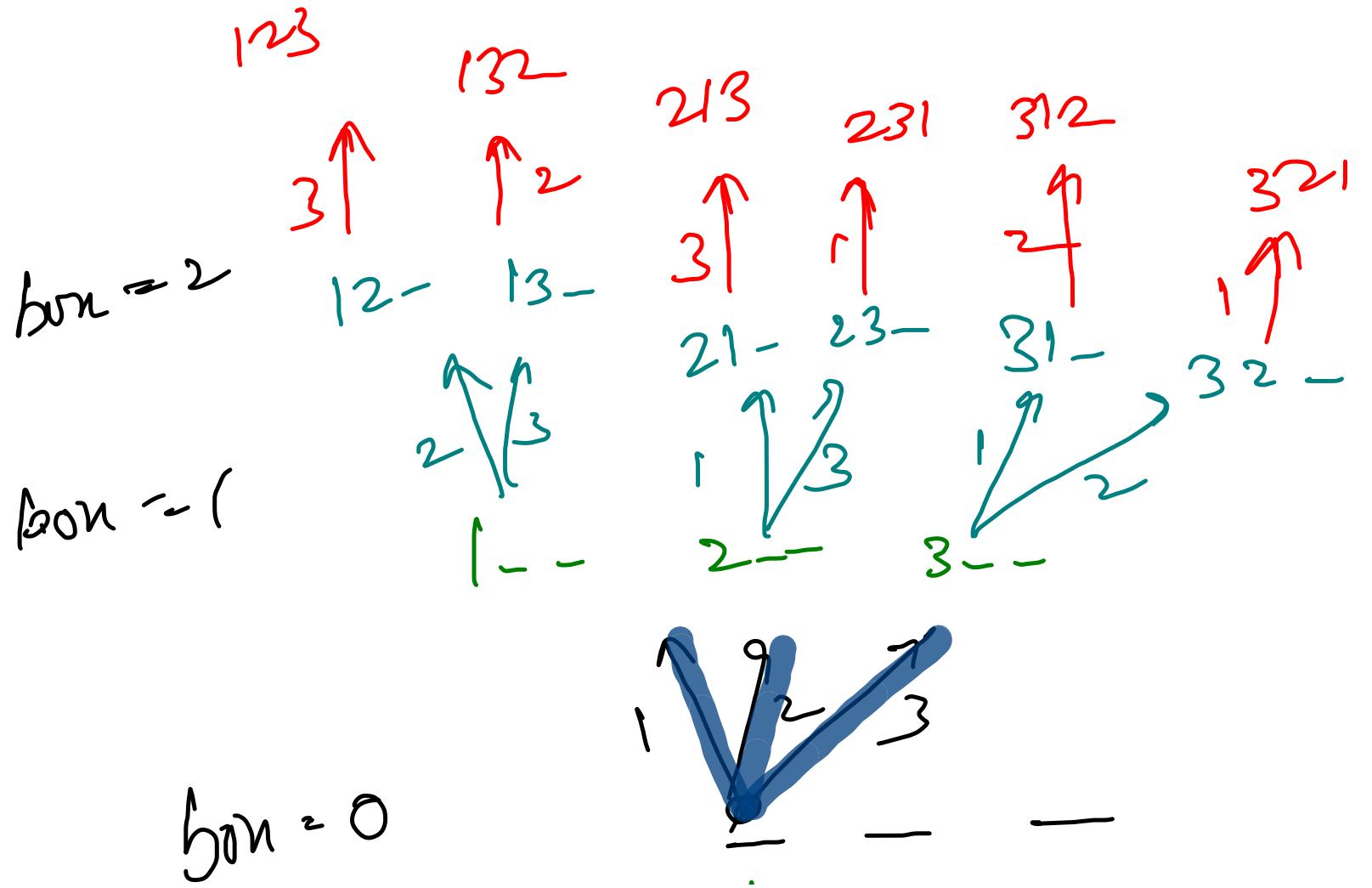


```

public static void combinations(int[] boxes, int ci, int ti, int lastItemIdx){
    // write your code here
    if(ci == ti){
        for(int val: boxes){
            if(val == 0) System.out.print("-");
            else System.out.print("i");
        }
        System.out.println();
    }

    // Item -> Choose box
    for(int i=lastItemIdx + 1; i<boxes.length; i++){
        if(boxes[i] == 0){
            boxes[i] = ci + 1;
            combinations(boxes, ci + 1, ti, i);
            boxes[i] = 0;
        }
    }
}

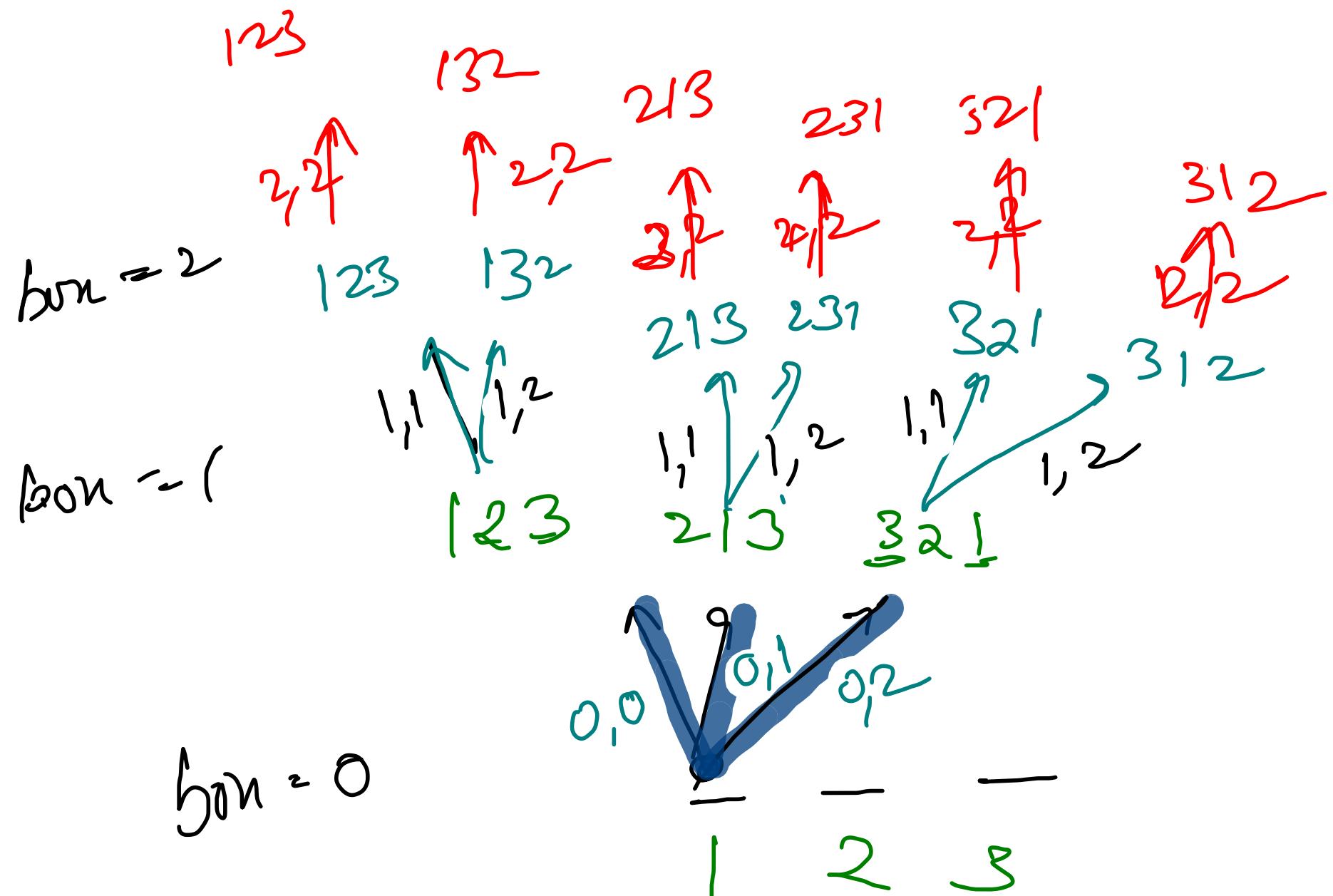
```



~~b_{inv}~~
items

b_{inv} = items

{1, 2, 3}



```

public void permutations(int currentBox, int[] nums){
    if(currentBox == nums.length){
        List<Integer> copy = new ArrayList<>();
        for(int i=0; i<nums.length; i++){
            copy.add(nums[i]);
        }
        res.add(copy);
        return;
    }

    for(int i=currentBox; i<nums.length; i++){
        swap(nums, currentBox, i);
        permutations(currentBox + 1, nums);
        swap(nums, currentBox, i);
    }
}

```

Unique Permutations

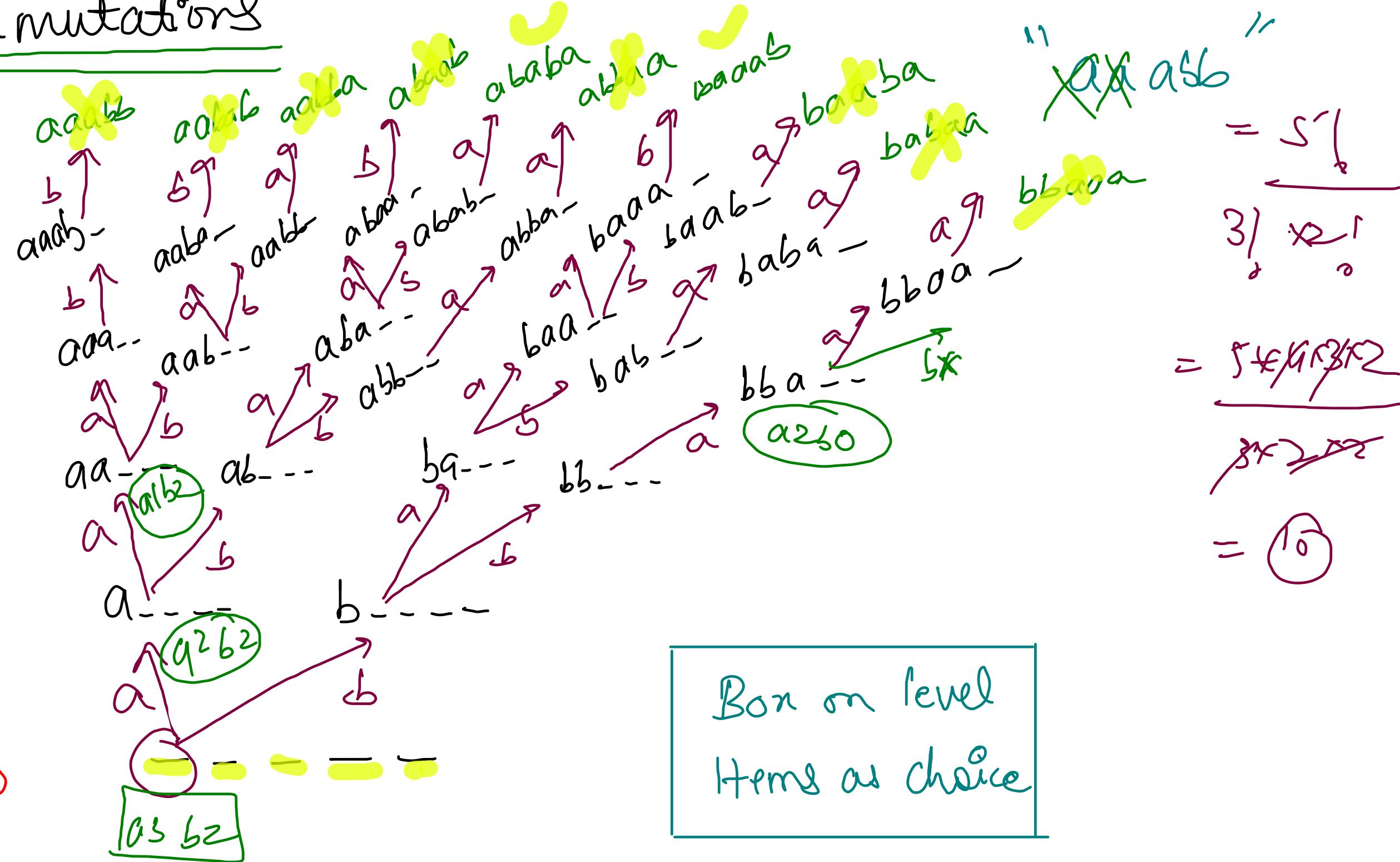
$$b_{on} = 4$$

$$b_{on} = 3$$

$$b_{on} = 2$$

$$b_{on} = 1$$

$$b_{on} = 0$$



helper

```
public void permutations(HashMap<Integer, Integer> freq, int[] nums,
    int currentBox, List<Integer> ans){
    if(currentBox == nums.length){
        List<Integer> copy = new ArrayList<>(ans);
        res.add(copy);
    }

    for(Integer key: freq.keySet()){
        int val = freq.get(key);
        if(val > 0){
            freq.put(key, val - 1);
            ans.add(key);

            permutations(freq, nums, currentBox + 1, ans);

            ans.remove(ans.size() - 1);
            freq.put(key, val);
        }
    }
}
```

main

```
HashMap<Integer, Integer> freq = new HashMap<>();
for(int i=0; i<nums.length; i++){
    if(freq.containsKey(nums[i])){
        int val = freq.get(nums[i]);
        freq.put(nums[i], val + 1);
    } else {
        freq.put(nums[i], 1);
    }
}
res = new ArrayList<>();
permutations1(freq, nums, 0, new ArrayList<>());
```

Hem on level
Box as choice

item = 'b'

item = 'b'

item = 'o'

item = 'a'

$a \xrightarrow{1} b \xrightarrow{3}$
aabb

$a \xrightarrow{1} b \xrightarrow{2}$
aab-

$a \xrightarrow{1} b \xrightarrow{1}$
aa-

abab



aaba



baab



baba



bbbaa



a-

-a-

--a-

---a

$$\frac{4C}{2^r 2^l} = \frac{N^3 \times 2}{2^r 2^l} \\ \therefore = 6$$

helper

```
public void permutations2(HashMap<Integer, Integer> lastIdx, int[] nums,
    int currentItem, List<Integer> ans){
    if(currentItem == nums.length){
        List<Integer> copy = new ArrayList<>(ans);
        res.add(copy);
        return;
    }
    int st = lastIdx.get(nums[currentItem]);
    for(int i=st+1; i<ans.size(); i++){
        if(ans.get(i) == null){
            ans.set(i, nums[currentItem]);
            lastIdx.put(nums[currentItem], i);
            permutations2(lastIdx, nums, currentItem + 1, ans);
            lastIdx.put(nums[currentItem], st);
            ans.set(i, null);
        }
    }
}
```

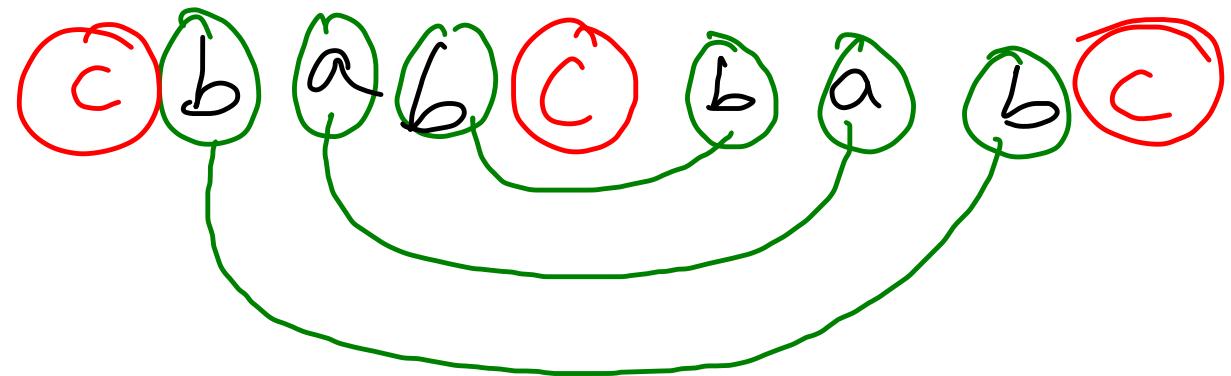
main

```
HashMap<Integer, Integer> lastIdx = new HashMap<>();
for(int i=0; i<nums.length; i++){
    lastIdx.put(nums[i], -1);
}

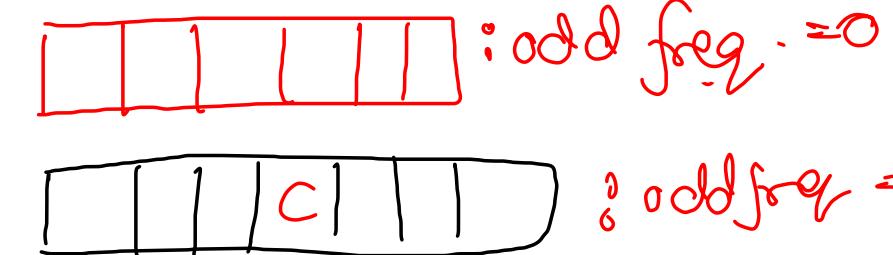
List<Integer> ans = new ArrayList<>();
for(int i=0; i<nums.length; i++){
    ans.add(null);
}

res = new ArrayList<>();
permutations2(lastIdx, nums, 0, ans);
return res;
```

#Palindrome Permutation - I



$a \rightarrow 2$
 $b \rightarrow 4$
 $c \rightarrow 3$



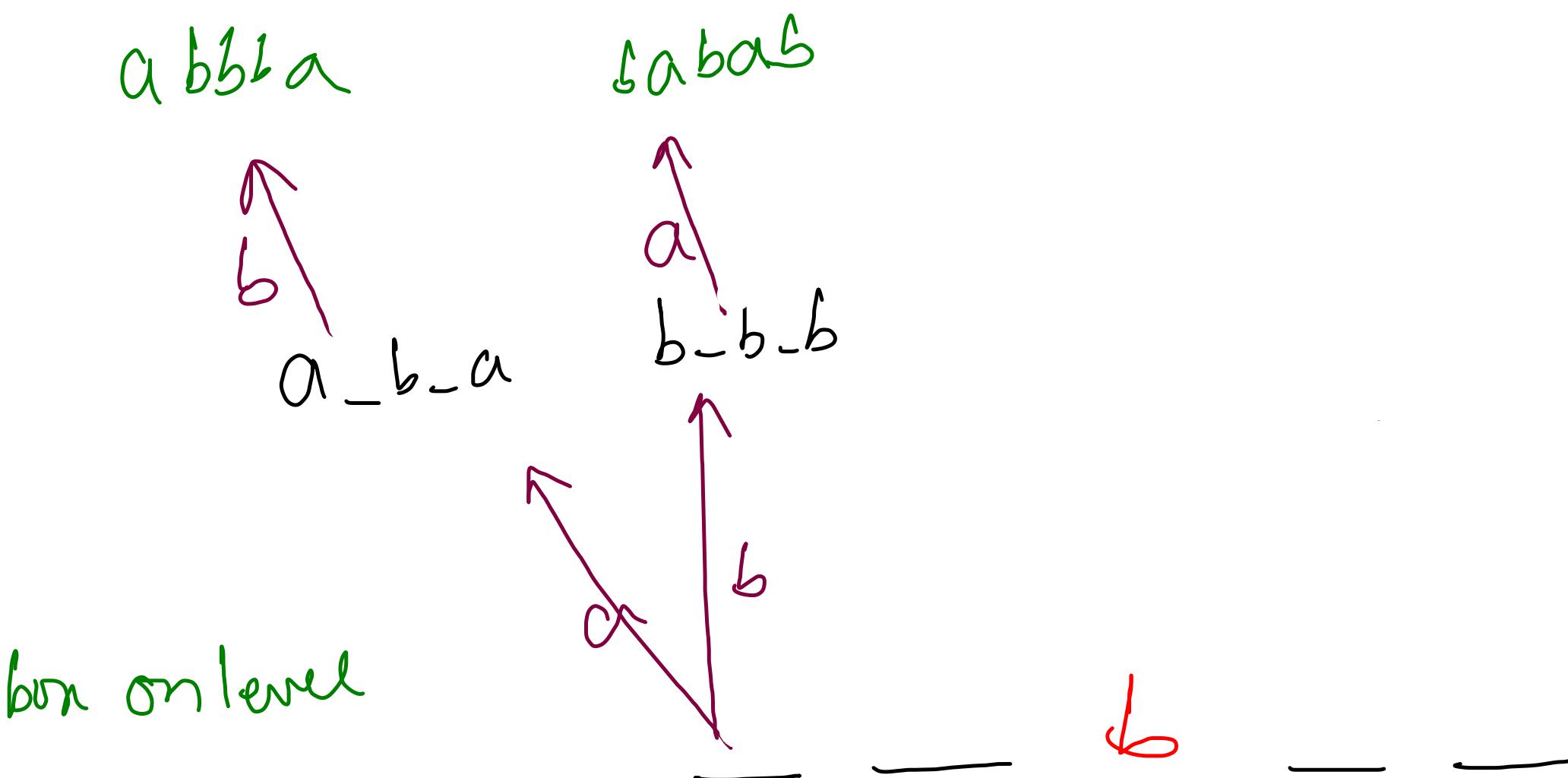
all char
even freq

Atmost 1 char
with
odd freq

```
public boolean canPermutePalindrome(String s) {  
    HashMap<Character, Integer> freq = new HashMap<>();  
    for(int i=0; i<s.length() ; i++){  
        if(freq.containsKey(s.charAt(i)) == false){  
            freq.put(s.charAt(i), 1);  
        } else {  
            freq.put(s.charAt(i), freq.get(s.charAt(i)) + 1);  
        }  
  
        int oddCount = 0;  
        for(Character c: freq.keySet()){  
            if(freq.get(c) % 2 == 1){  
                oddCount++;  
            }  
        }  
  
        if(oddCount > 1) return false;  
        return true;  
    }
```

All Palindromic Permutations $\{ P \in \Pi \}$

aabbba
a²b²



aaaabbc

a⁴b²c

aabcbaa

↑
s

aa-c-aa

abacaba

a↑

ab-c-ba

(a) - - c - - (a)
a₂b₂c₀

baacaaci

a↑

ba-c-ab

a↑

b - - c - - b

a
b
c
n1-cb

box on level

item as
choice

K words

{ Repition not allowed } { Combinations }

a2 b3 c2 d2 e

unique characters : 5 boxes

① Box on level

② Item on level \rightarrow stems
 \rightarrow k characters

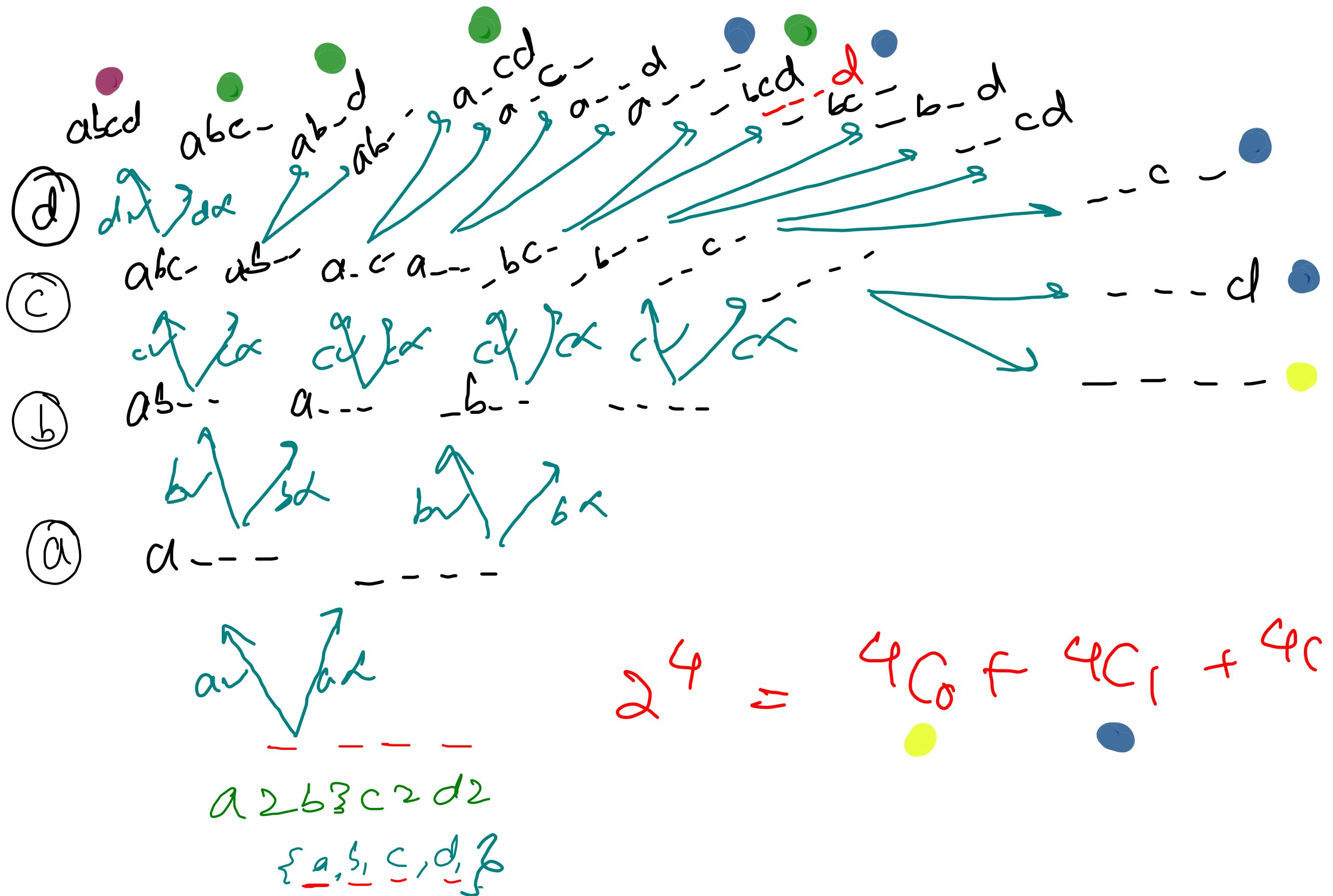
aabbccdde
3

a, b, c, d, e
3 items

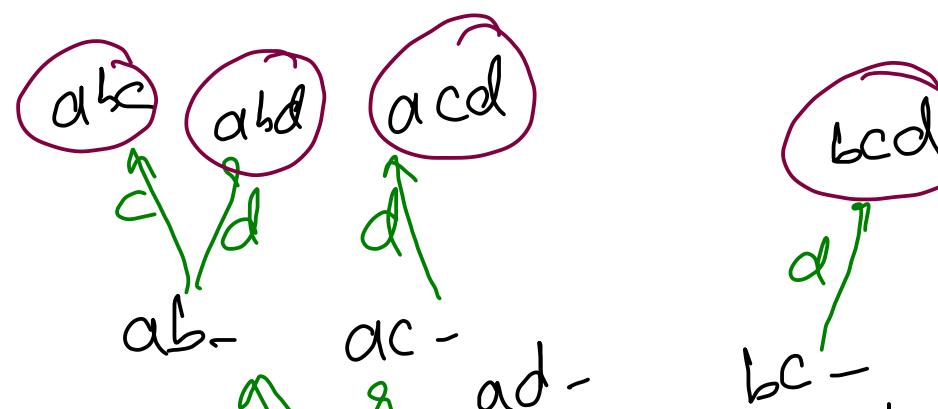
{ a b c d e }

$$5C_3 = \frac{5^4 * 4^2}{3 * 2 * 1} = 10$$

{ }
}

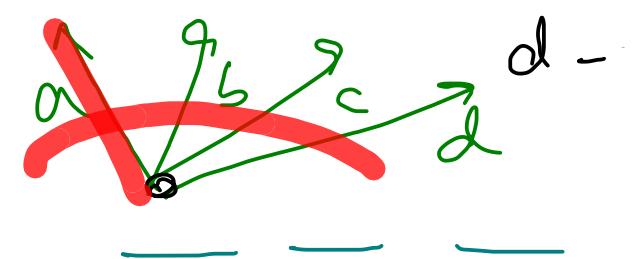


$$2^4 = 4C_0 + 4C_1 + 4C_2 + C_3 + C_4$$



$$4C_3 = 4$$

$O((k))^{unique\ char}$



$a2b3c2d$
 $\{a, b, c, d\}$

```
public static void combination(ArrayList<Character> chs, int lastIdx, String res, int k){
    if(res.length() == k){
        System.out.println(res);
        return;
    }

    for(int i=lastIdx + 1; i<chs.size(); i++){
        combination(chs, i, res + chs.get(i), k);
    }
}
```

Today's Questions

- ① {
 - </> Words - K Selection - 1
 - </> Words - K Selection - 2
- ② {
 - </> Words - K Length Words - 1
 - </> Words - K Length Words - 2
- ③ {
 - </> Words - K Selection - 3
 - </> Words - K Selection - 4
- ④ {
 - </> Words - K Length Words - 3
 - </> Words - K Length Words - 4
- ⑤ {
 - </> Coin Change - Combinations - 1
 - </> Coin Change - Combinations - 2
- ⑥ {
 - </> Coin Change - Permutations - 1
 - </> Coin Change - Permutations - 2

{String Combination → Repitition not allowed}

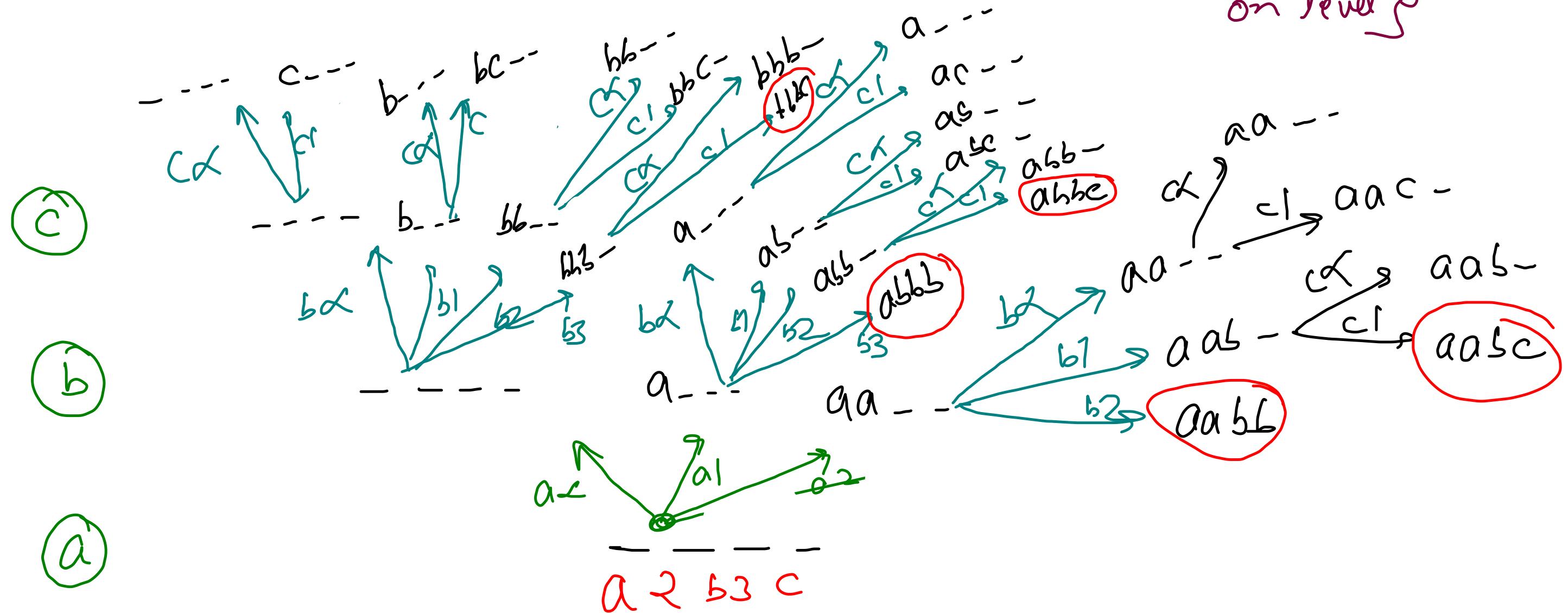
{String Combination → Repitition allowed}

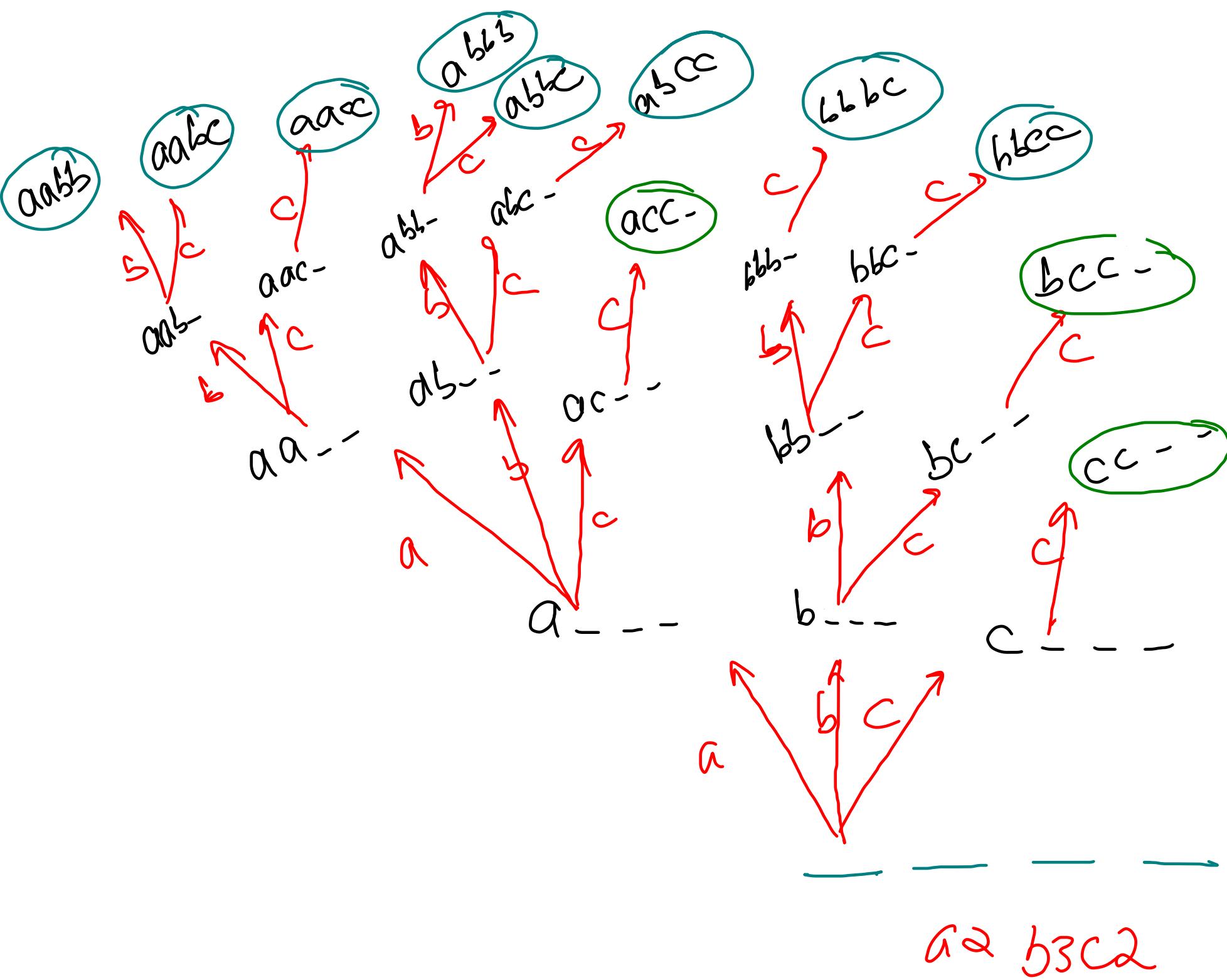
K word select (ii)

Repetition allowed

$$k = 4$$

{ unique character on level }



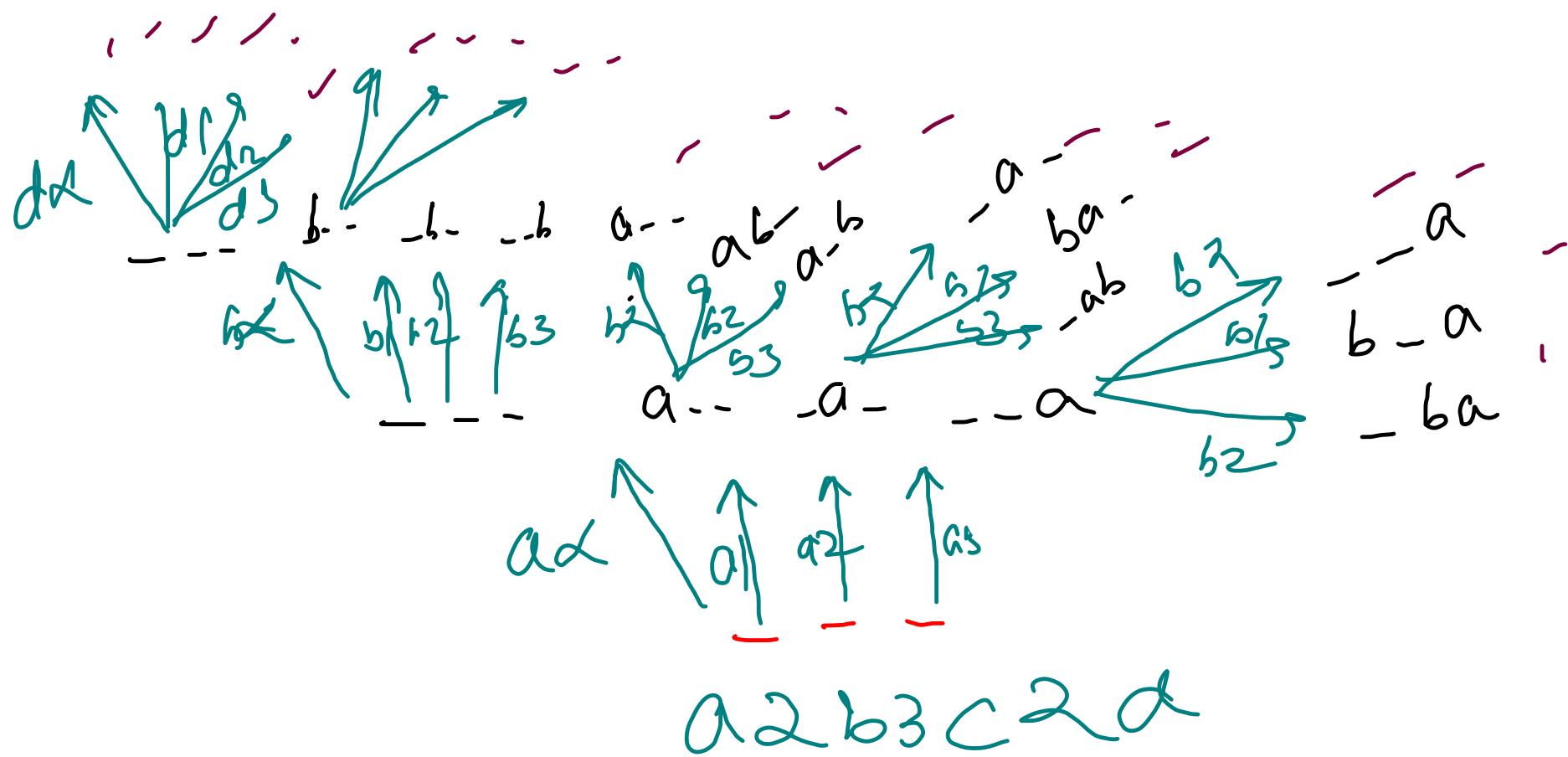


K words
 selector - IV
 combinator
 Repetition allowed

```
public static void generateSelection(String ustr, int cs, int ts, int lc, HashMap<Character, Integer> freq, String asf) {
    if (cs > ts) {
        System.out.println(asf);
        return;
    }
    if(lc == ustr.length()){
        return;
    }

    for (int i = lc; i < ustr.length(); i++) {
        int oldFreq = freq.get(ustr.charAt(i));
        if(oldFreq > 0){
            freq.put(ustr.charAt(i), oldFreq - 1);
            generateSelection(ustr, cs + 1, ts, i, freq, asf + ustr.charAt(i));
            freq.put(ustr.charAt(i), oldFreq);
        }
    }
}
```

K words Permutation



① Repetition not allowed

→ Unique char

→ Slot on level

② Repetition allowed

→ Unique char

→ Slot on level

d

c

b

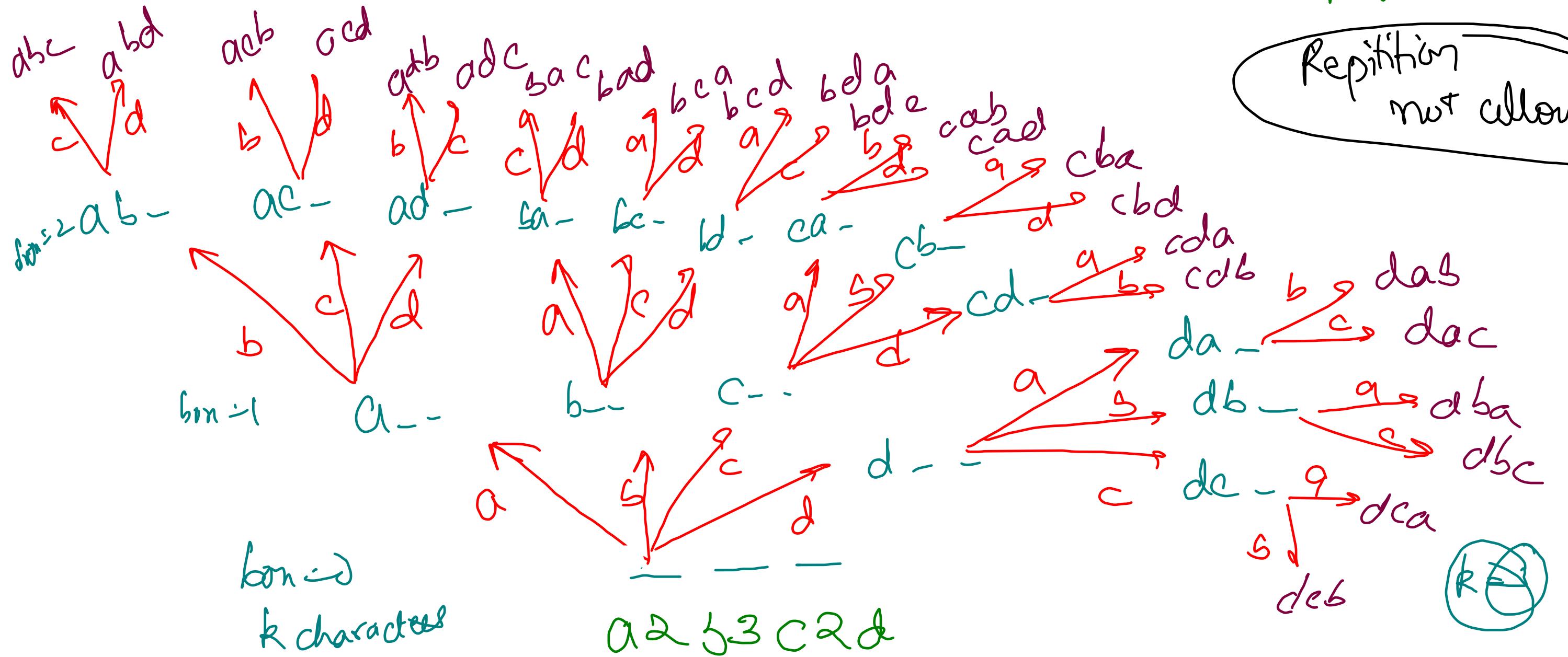
a

$k=3$

Bon on level

k words

Repetition
not allowed

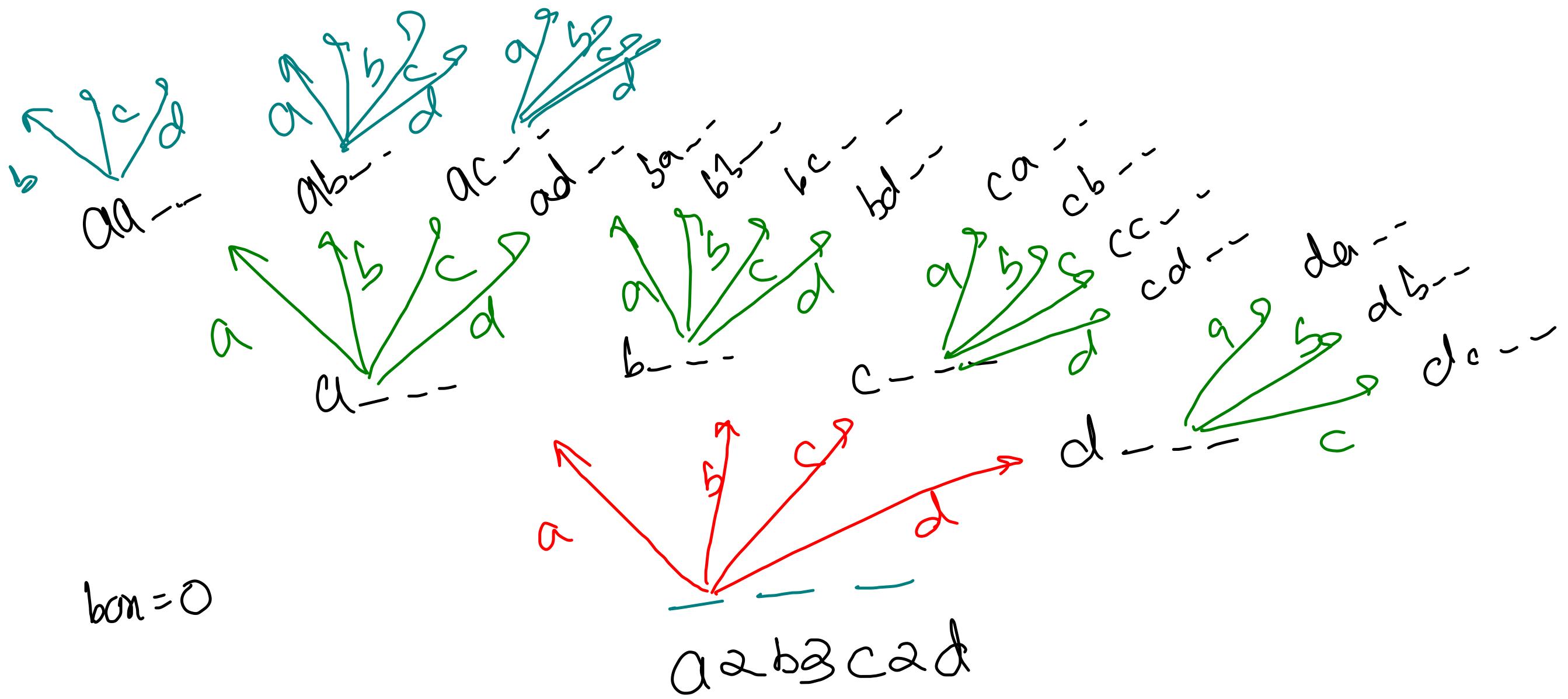


K

```
public static void permutation(String res, String ustr, HashSet<Character> unique, int k){  
    if(res.length() == k){  
        System.out.println(res);  
        return;  
    }  
  
    for(Character ch: ustr.toCharArray()){  
        if(unique.contains(ch) == true){  
            unique.remove(ch);  
            permutation(res + ch, ustr, unique, k);  
            unique.add(ch);  
        }  
    }  
}
```

Press Esc to exit full screen

option
allowed

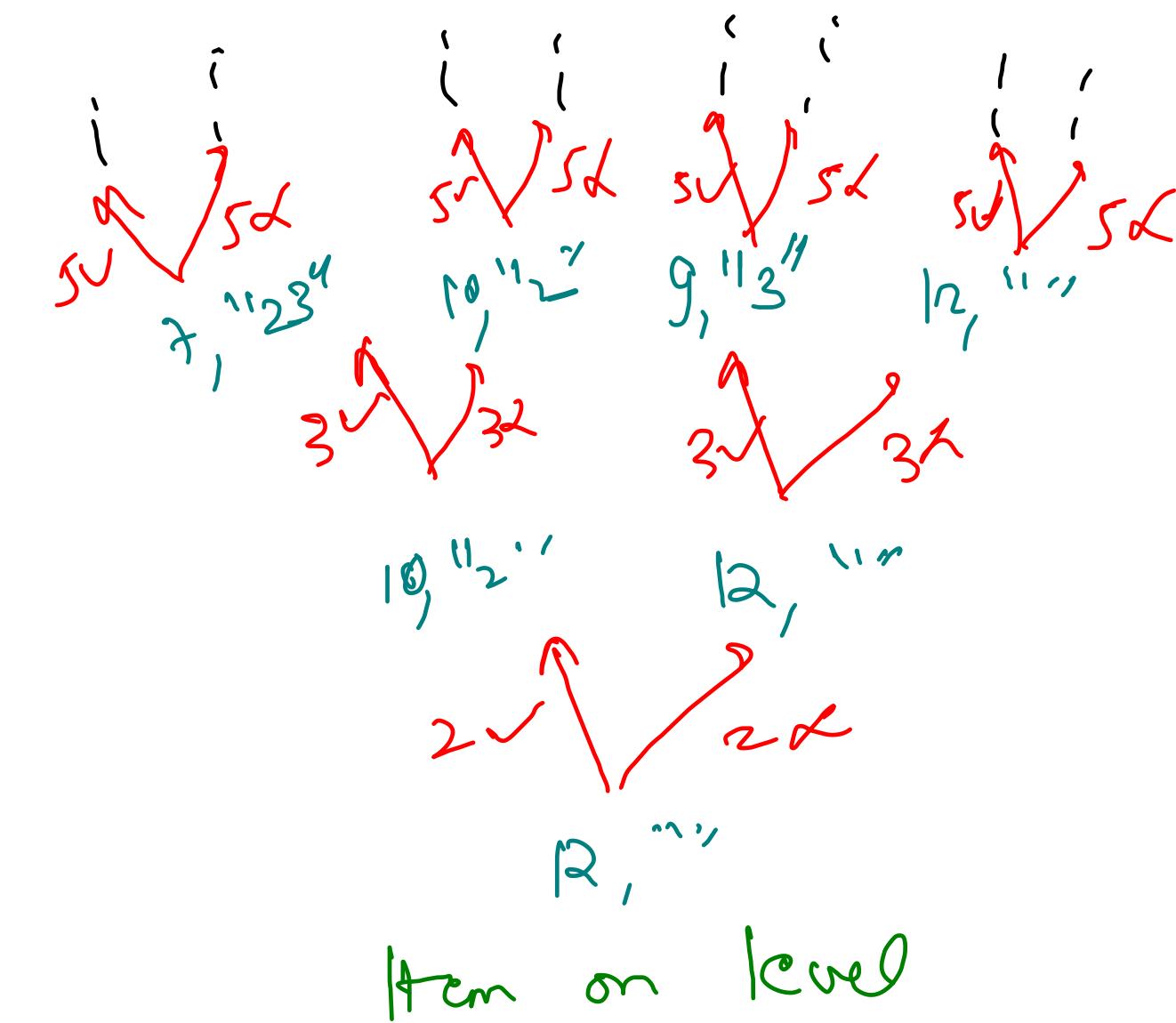
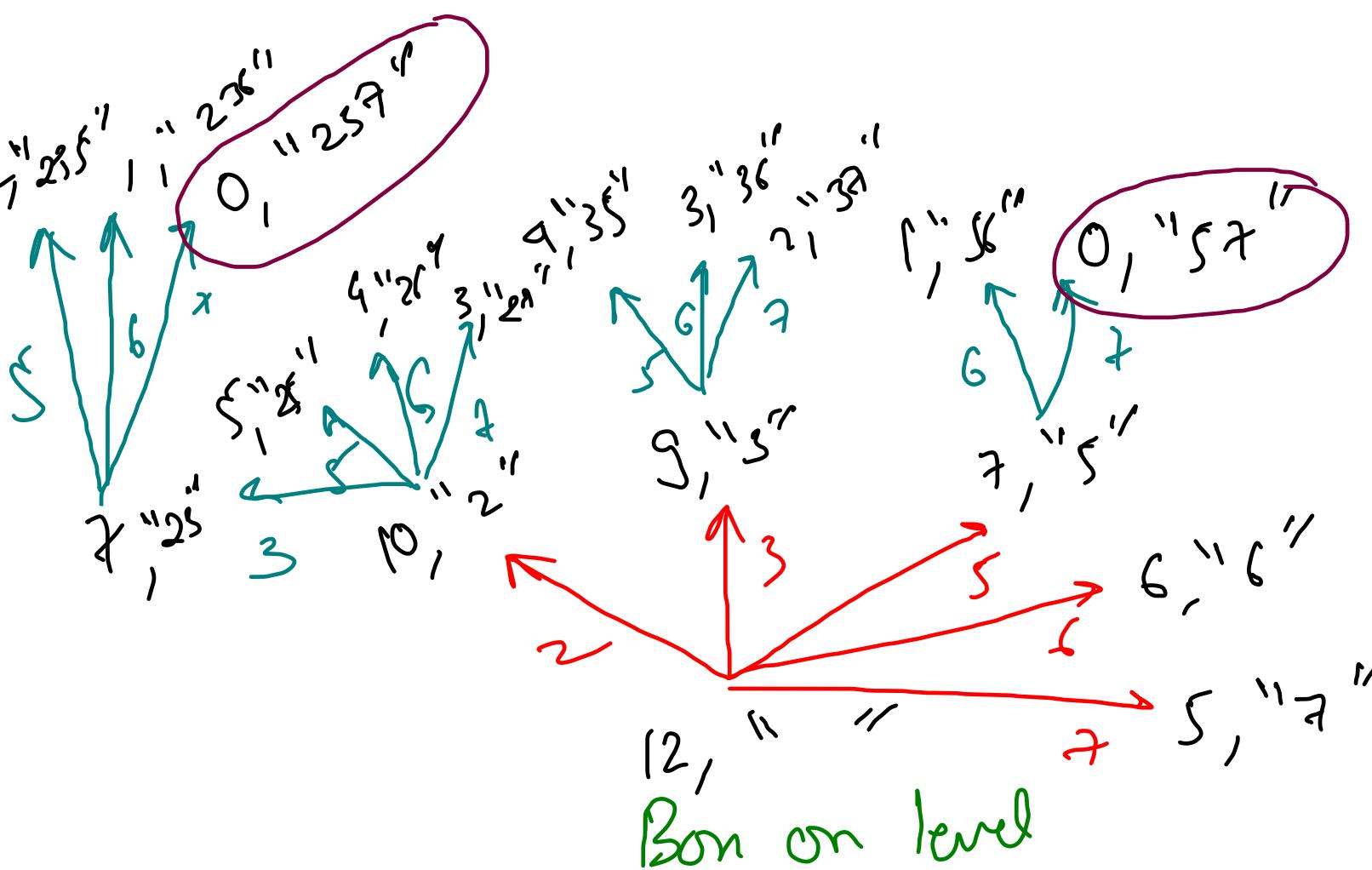


$$b \neq c = 0$$

$a^2 b^3 c^2 d$

```
public static void permutation(String res, String ustr, HashMap<Character, Integer> unique, int k){  
    if(res.length() == k){  
        System.out.println(res);  
        return;  
    }  
  
    for(Character ch: ustr.toCharArray()){  
        int oldFreq = unique.get(ch);  
        if(oldFreq > 0){  
            unique.put(ch, oldFreq - 1);  
            permutation(res + ch, ustr, unique, k);  
            unique.put(ch, oldFreq);  
        }  
    }  
}
```

Coin Change Combination(II)



Finite Supply

```
public void combinations(List<Integer> ans, HashMap<Integer, Integer> freq,
                        ArrayList<Integer> uniqueCoins, int currentCoin, int target){
    if(target == 0){
        res.add(new ArrayList<>(ans));
        return;
    }
    if(currentCoin == uniqueCoins.size()){
        return;
    }

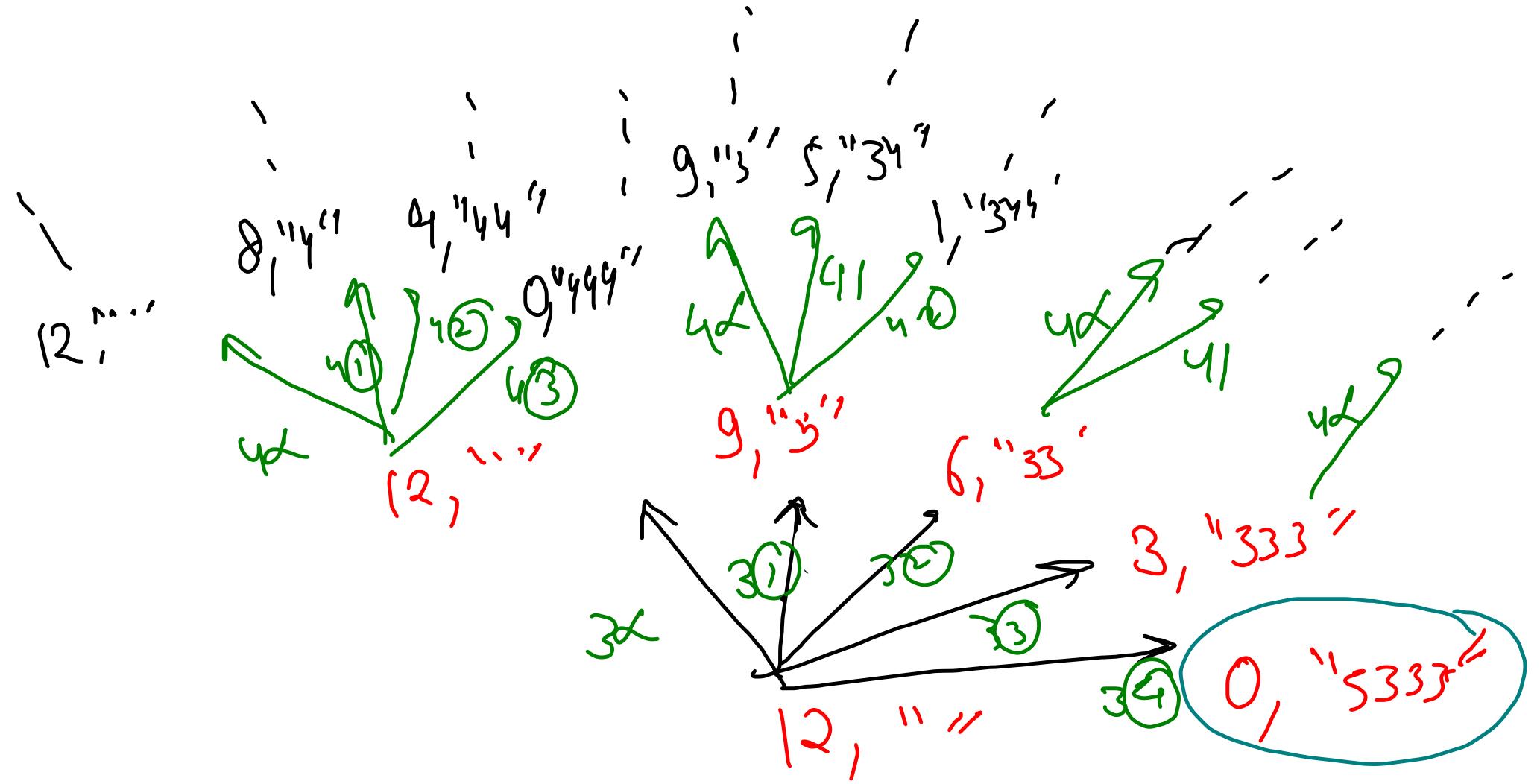
    int val = uniqueCoins.get(currentCoin);
    for(int i=0; i<=freq.get(val); i++) {
        if(target - i * val >= 0){
            for(int j=1; j<=i; j++){
                ans.add(val);
            }

            combinations(ans, freq, uniqueCoins, currentCoin + 1, target - i * val);

            for(int j=1; j<=i; j++){
                ans.remove(ans.size() - 1);
            }
        }
    }
}
```

Infinite Supply

$$\{3, 4, 5\} \quad 12$$



~~max size~~
~~Supply~~

```
public void combinations(List<Integer> ans, int[] uniqueCoins, int currentCoin, int target){
    if(target == 0){
        res.add(new ArrayList<>(ans));
        return;
    }
    if(currentCoin == uniqueCoins.length){
        return;
    }

    int val = uniqueCoins[currentCoin];

    for(int i=0; target - i * val >= 0; i++) {
        for(int j=1; j<=i; j++){
            ans.add(val);
        }

        combinations(ans, uniqueCoins, currentCoin + 1, target - i * val);

        for(int j=1; j<=i; j++){
            ans.remove(ans.size() - 1);
        }
    }
}
```

Today's Questions

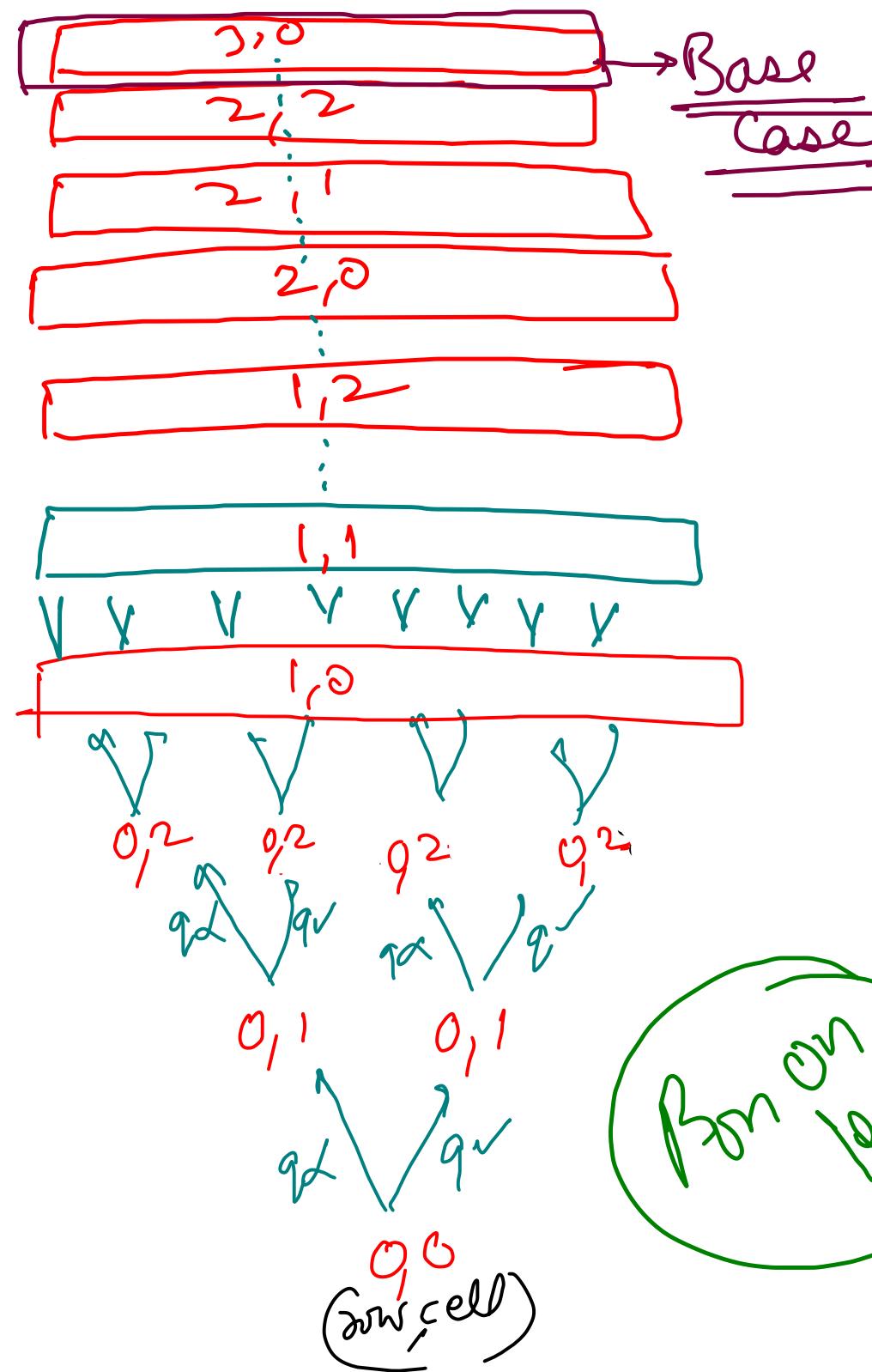
- ① { Coin change - Permutation → I
 Coin change - Permutation → II
- ② { </> Queens Permutations - 2d As 2d - Queen Chooses
</> Queens Permutations - 2d As 2d - Box Chooses
- ③ { </> Queens Combinations - 2d As 2d - Queen Chooses
</> Queens Combinations - 2d As 2d - Box Chooses
- ④ { </> Nqueens Combinations - 2d As 1d - Queen Chooses
</> Nqueens Permutations - 2d As 1d - Queen Chooses
- ⑤ { </> Nknight Combinations - 2d As 1d - Knight Chooses

- ① Recursion Tree & RT
- ② RT ↗ level → parameters
 options → for
- ③ Backtracking
- ④ Recorder ↗ true & -ve base case
- ⑤ Dry run

Dictionary Order - small "cab"

```
void permute(string &input, vector<bool> &vis, string output, string &s){  
    if(output.size() == s.size()){  
        if(output < s){  
            cout << output << endl;  
        }  
        return;  
    }  
    for(int i=0; i<input.size(); i++){  
        if(vis[i] == false){  
            vis[i] = true;  
            permute(input, vis, output + input[i], s);  
            vis[i] = false;  
        }  
    }  
}
```

2D Combination - I



$\{q, q, q\} \quad k=3$

0,0	0,1	0,2
1,0	1,1	1,2
2,0	2,1	3,2

3,0

$k=n$
 $k=r$ n^2
 queen king

$$\text{Total Combinations} = 2^{N \times N} = 2^9$$

$$2^9 = qC_0 + qC_1 + qC_2 + \dots + qC_9$$

Result

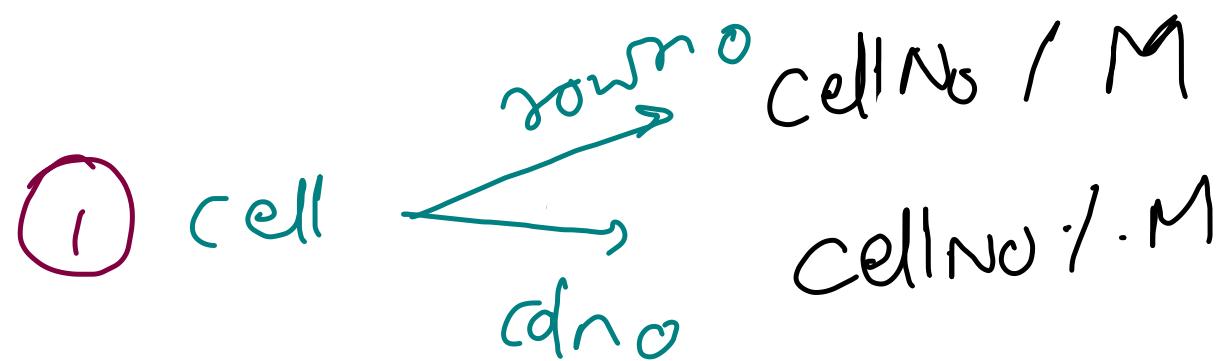
```
// qpsf -> Queens placed so far, tq -> total Queens
public static void queensCombinations(int qpsf, int tq, int row, int col, String mat){
    if(row == tq){
        if(qpsf == tq){
            System.out.println(mat);
        }
        return;
    }

    if(col == tq - 1){
        // Apni Row ka last element, agli row ke pehle element par jana hai
        queensCombinations(qpsf + 1, tq, row + 1, 0, mat + 'q' + '\n'); // yes
        queensCombinations(qpsf, tq, row + 1, 0, mat + '-' + '\n'); // no
    } else {
        queensCombinations(qpsf + 1, tq, row, col + 1, mat + 'q'); // yes
        queensCombinations(qpsf, tq, row, col + 1, mat + '-'); // no
    }
}
```

0	1	2	3	4	5
0	0	1	2	3	4
1	6	7	8	9	10
2	12	13	14	15	16
3	18	19	20	21	22

4 rows $\therefore N = 4$

6 columns $\therefore M = 6$



② $\text{rowCol} \xrightarrow{\text{cellNo}} \text{rowNo} * M + \text{colNo}$

$$2 * 6 + 4 = 16$$

$$3 * 6 + 1 = 19$$

$$16 / 2 = 2$$

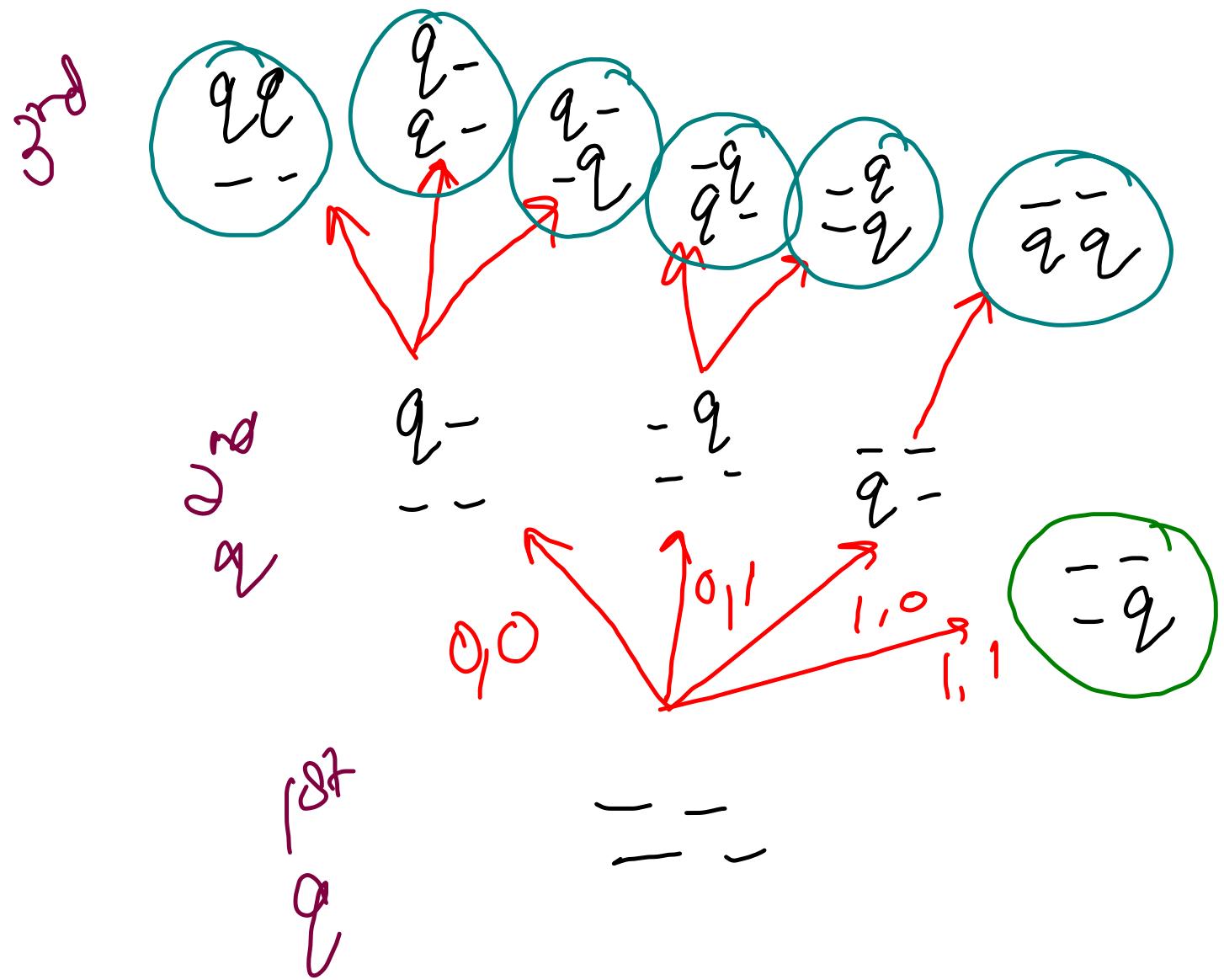
$$16 / .2 = 4$$

$$16 = 6 * 2 + 4$$

```
// qpsf -> Queens placed so far, tq -> total Queens
public static void queensCombinations(int qpsf, int tq, int cellNo, String mat){
    if(cellNo == tq * tq){
        if(qpsf == tq){
            System.out.println(mat);
        }
        return;
    }

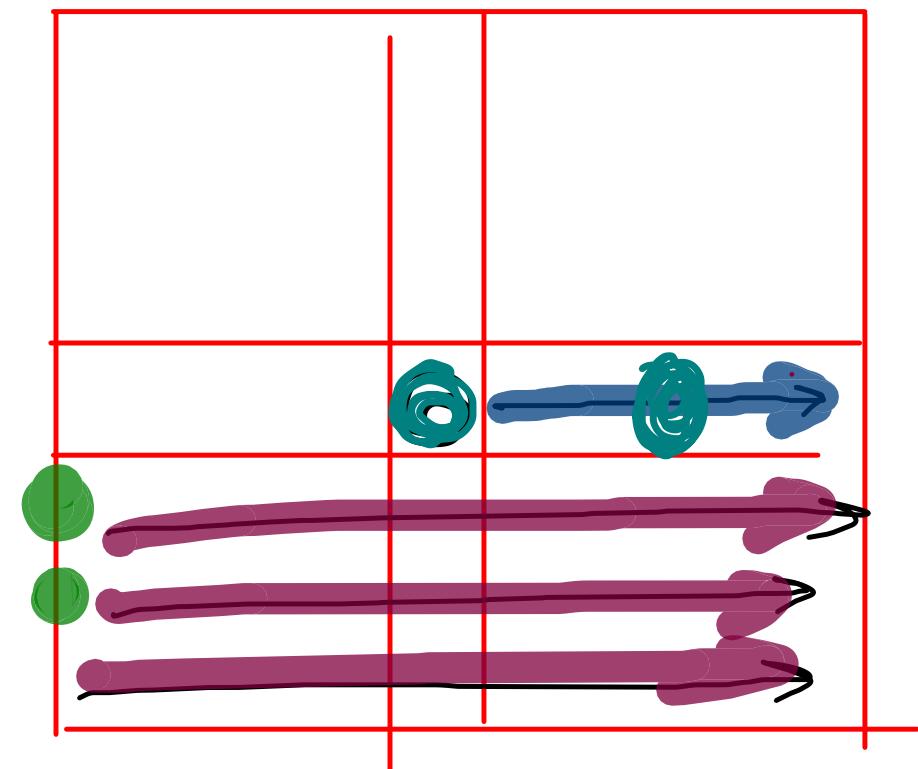
    if(cellNo % tq == tq - 1){
        // Apni Row ka last element, agli row ke pehle element par jana hai
        queensCombinations(qpsf + 1, tq, cellNo + 1, mat + 'q' + '\n'); // yes
        queensCombinations(qpsf, tq, cellNo + 1, mat + '-' + '\n'); // no
    } else {
        queensCombinations(qpsf + 1, tq, cellNo + 1, mat + 'q'); // yes
        queensCombinations(qpsf, tq, cellNo + 1, mat + '-'); // no
    }
}
```

2D Combinatⁿ : Queen on level



$$k=2$$

$$n=2 \Rightarrow \text{boxes} = n^2 = 4$$



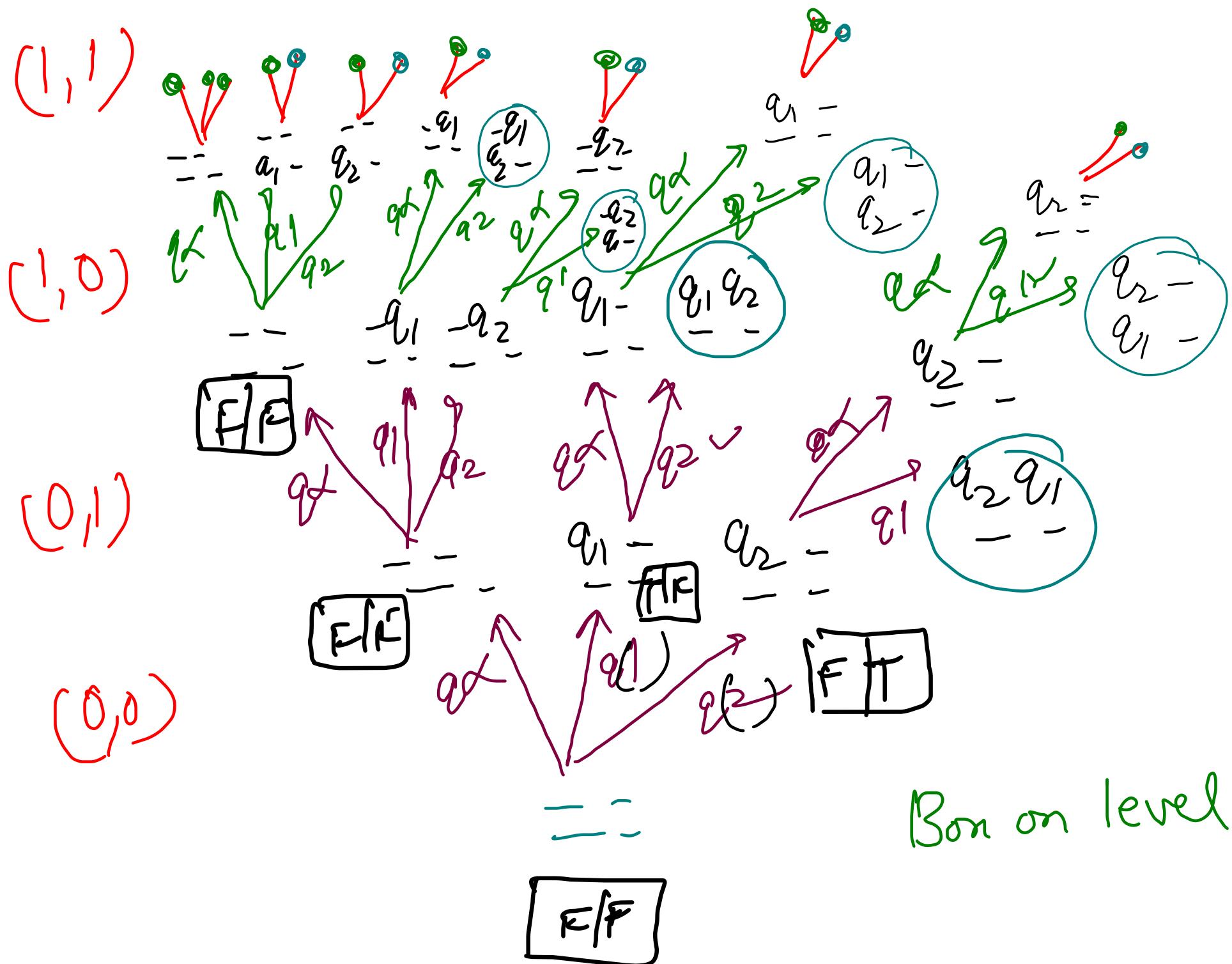
```
public static void queensCombinations(int qpsf, int tq, character[][] chess, int lastQueenRow, int lastQueenCol){  
    if(qpsf == tq){  
        for(int i=0; i<tq; i++) { }  
        System.out.println();  
        return;  
    }  
  
    for(int j=lastQueenCol + 1; j < tq; j++){  
        if(chess[lastQueenRow][j] == '-'){  
            chess[lastQueenRow][j] = 'q';  
            queensCombinations(qpsf + 1, tq, chess, lastQueenRow, j);  
            chess[lastQueenRow][j] = '-';  
        }  
    }  
  
    for(int i=lastQueenRow + 1; i < tq; i++){  
        for(int j=0; j<tq; j++){  
            if(chess[i][j] == '-'){  
                chess[i][j] = 'q';  
                queensCombinations(qpsf + 1, tq, chess, i, j);  
                chess[i][j] = '-';  
            }  
        }  
    }  
}
```

2D queen - Permutation

$$\{q_1, q_2\}$$

$$k=2$$

$$n=2 \Rightarrow n^2=4$$



```

if(cellNo % tq == tq - 1){
    // Apni Row ka last element, agli row ke pehle element par jana hai
    for(int i=0; i<tq; i++){
        if(vis[i] == false){
            vis[i] = true;
            queensPermutations(qpsf + 1, tq, cellNo + 1, mat + "q" + (i + 1) + "\t\n", vis); // yes
            vis[i] = false;
        }
    }
    queensPermutations(qpsf, tq, cellNo + 1, mat + "-\t\n", vis); // no
} else {
    for(int i=0; i<tq; i++){
        if(vis[i] == false){
            vis[i] = true;
            queensPermutations(qpsf + 1, tq, cellNo + 1, mat + "q" + (i + 1) + "\t" , vis); // yes
            vis[i] = false;
        }
    }
    queensPermutations(qpsf, tq, cellNo + 1, mat + "-\t", vis); // no
}

```

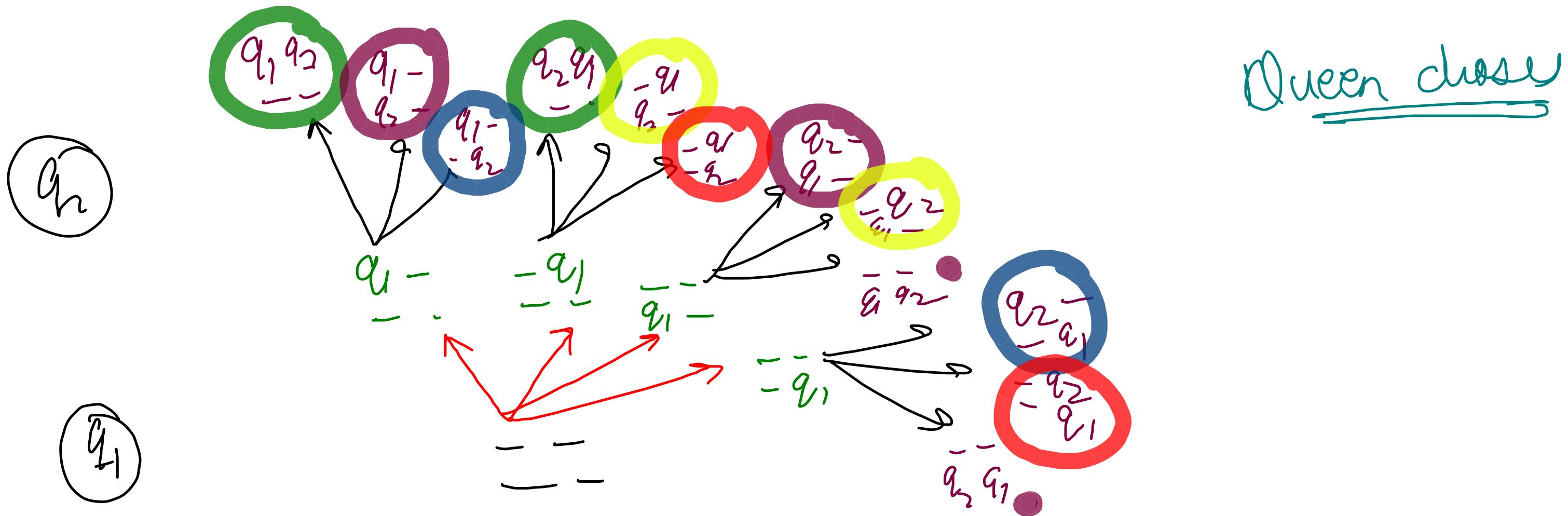
2D queen - Permutation

$$\{q_1, q_2\}$$

$$k=2$$

$$n=2 \Rightarrow n^2 = 4$$

$$4P_2 = 4C_2 * 2!$$



```
public static void queensCombinations(int qpsf, int tq,
Character[][] chess, int lastcellNo){
    if(qpsf == tq){
        for(int i=0; i<tq; i++) {➡}
        System.out.println();
        return;
    }

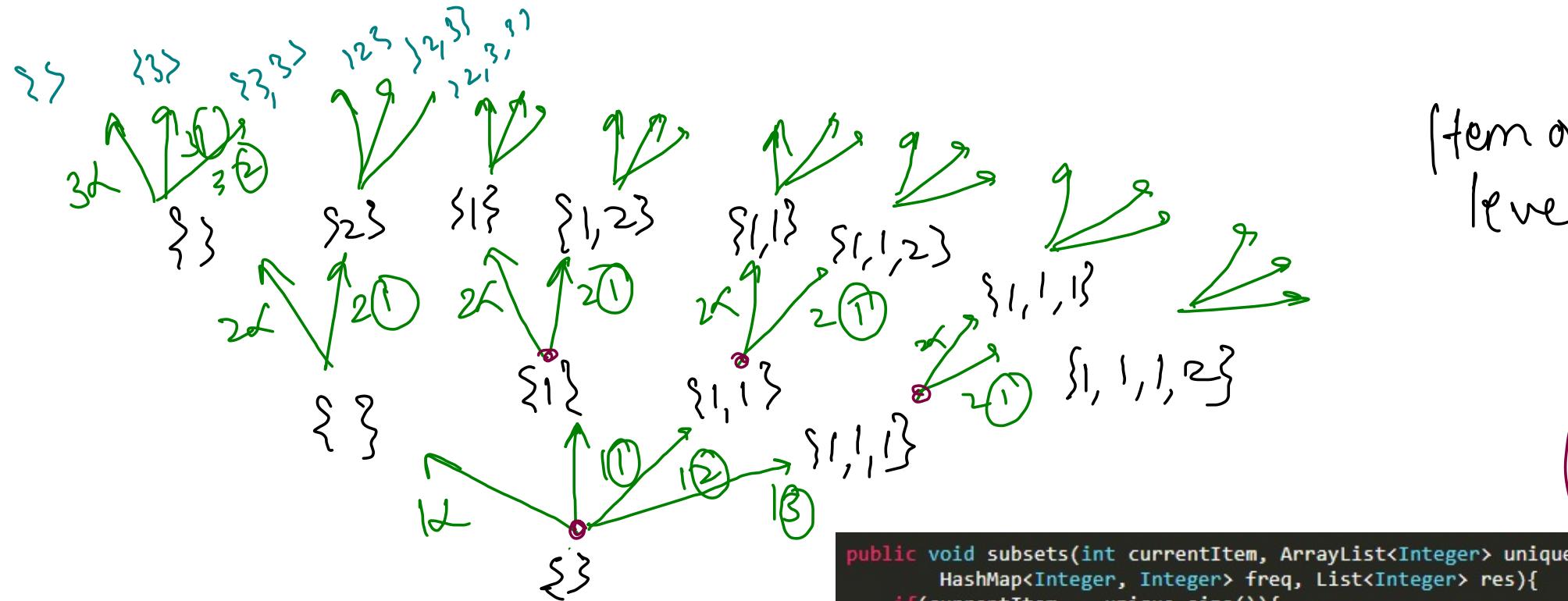
    for(int c=lastcellNo + 1; c < tq * tq; c++){
        int rowNo = c / tq;
        int colNo = c % tq;

        chess[rowNo][colNo] = 'q';
        queensCombinations(qpsf + 1, tq, chess, c);
        chess[rowNo][colNo] = '-';
    }
}
```

Unique Subsets {Subsets - II}

{0, 1, 1, 2, 3, 3}

HashMap of
freq



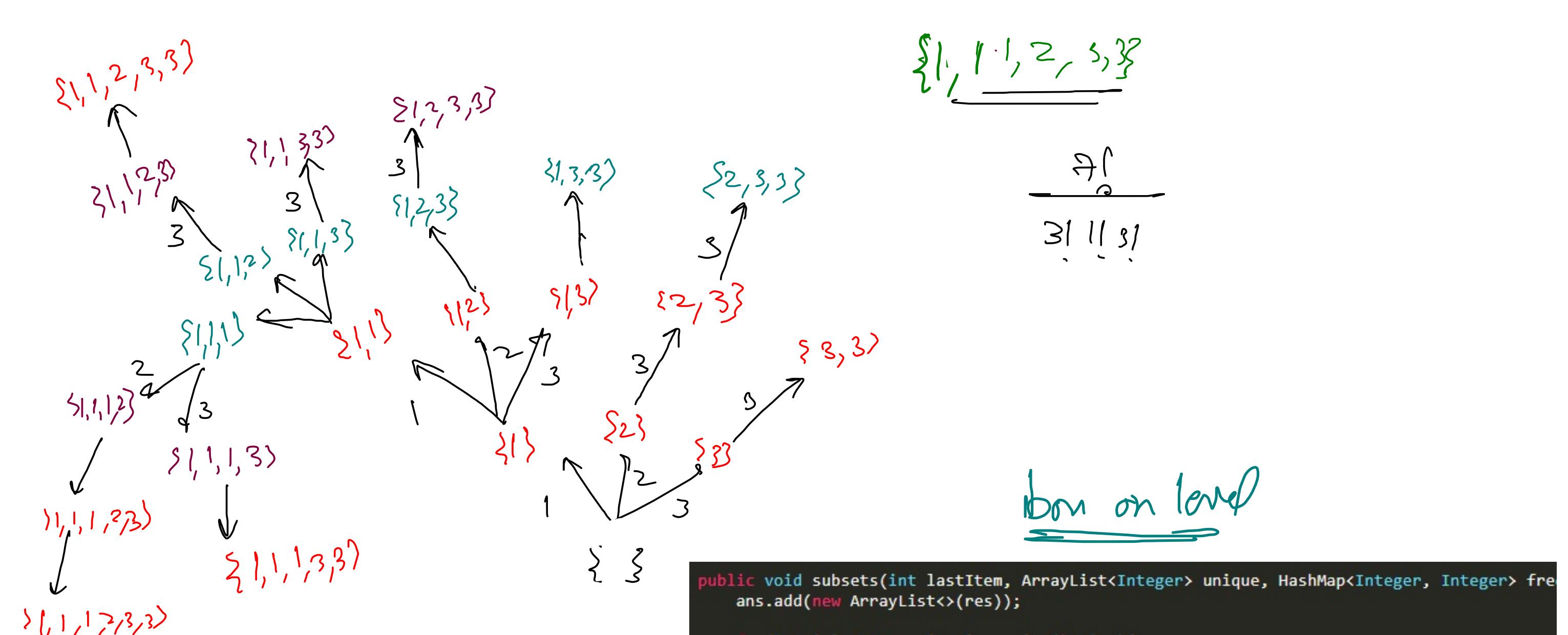
1 → 3
2 → 1
3 → 2

```
public void subsets(int currentItem, ArrayList<Integer> unique,
    HashMap<Integer, Integer> freq, List<Integer> res){
    if(currentItem == unique.size()){
        ans.add(new ArrayList<>(res));
        return;
    }

    int val = unique.get(currentItem);
    subsets(currentItem + 1, unique, freq, res); // no

    for(int f=0; f<freq.get(val); f++){
        res.add(val);
        subsets(currentItem + 1, unique, freq, res);
    }

    for(int f=0; f<freq.get(val); f++){
        res.remove(res.size() - 1);
    }
}
```



```

public void subsets(int lastItem, ArrayList<Integer> unique, HashMap<Integer, Integer> freq,
ans.add(new ArrayList<>(res));

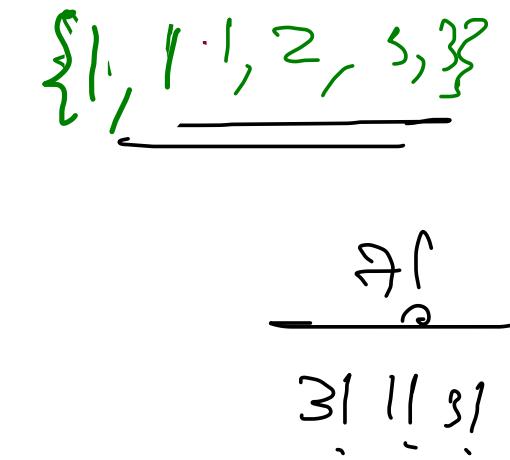
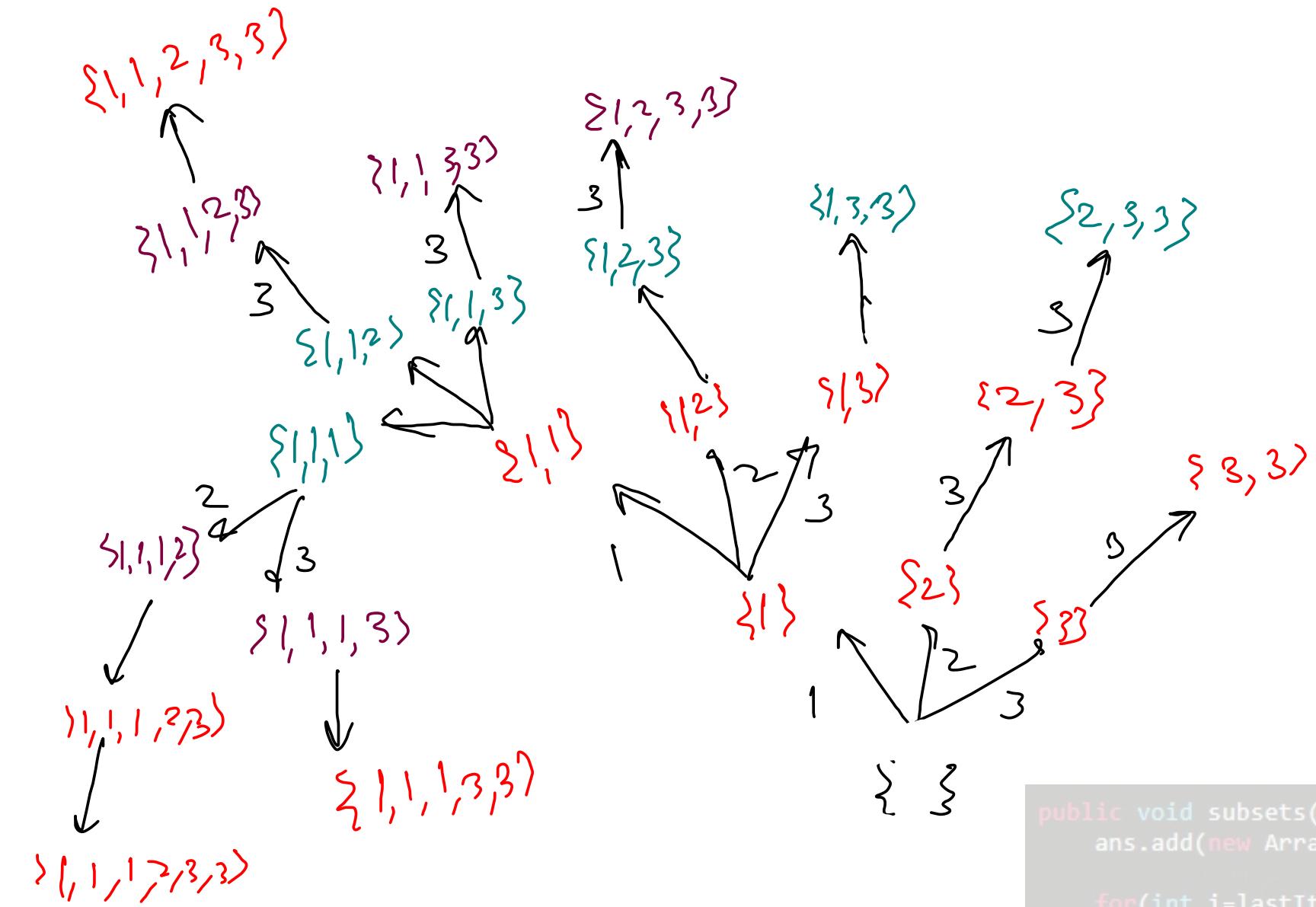
for(int i=lastItem; i<unique.size(); i++){
    int val = unique.get(i);
    int oldFreq = freq.get(val);

    if(oldFreq > 0){
        freq.put(val, oldFreq - 1);
        res.add(val);

        subsets(i, unique, freq, res);

        res.remove(res.size() - 1);
        freq.put(val, oldFreq);
    }
}
}

```



bottom on level

```

public void subsets(int lastItem, ArrayList<Integer> unique, HashMap<Integer, Integer> freq) {
    ArrayList<Integer> res = new ArrayList<Integer>();
    ans.add(res);

    for(int i=lastItem; i<unique.size(); i++){
        int val = unique.get(i);
        int oldFreq = freq.get(val);

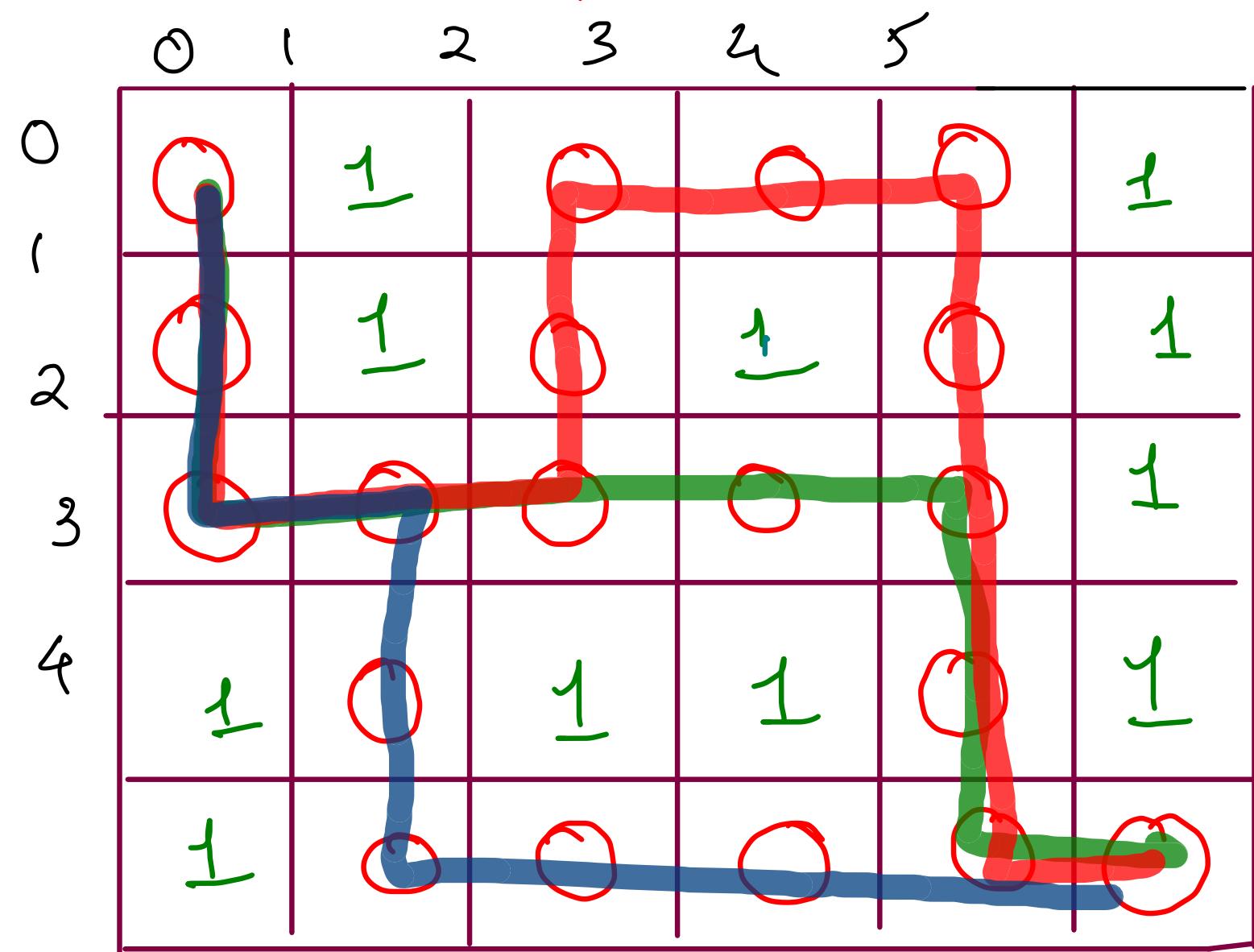
        if(oldFreq > 0){
            freq.put(val, oldFreq - 1);
            res.add(val);

            subsets(i, unique, freq, res);

            res.remove(res.size() - 1);
            freq.put(val, oldFreq);
        }
    }
}

```

Rat in Maze / Flood Fill



① Preorder {Node-Pre}

$\text{vis}[r][c] = \text{true}$

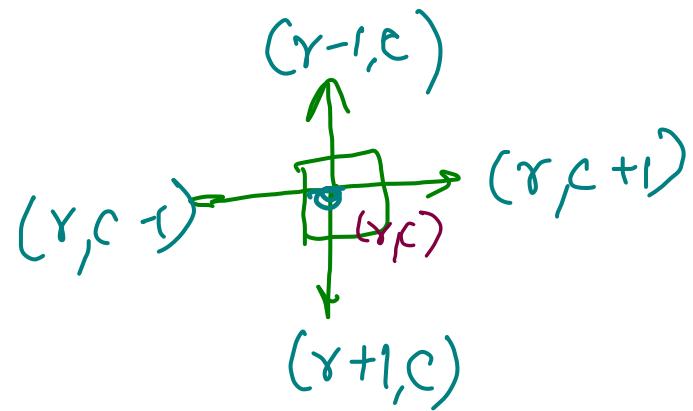
② Calls & TLD R }

③ Postorder {Node-Post}

$\text{vis}[r][c] = \text{false}$

4 way adjacent

{TRDL}



$x \Rightarrow \{-1, 0, +1, 0\}$

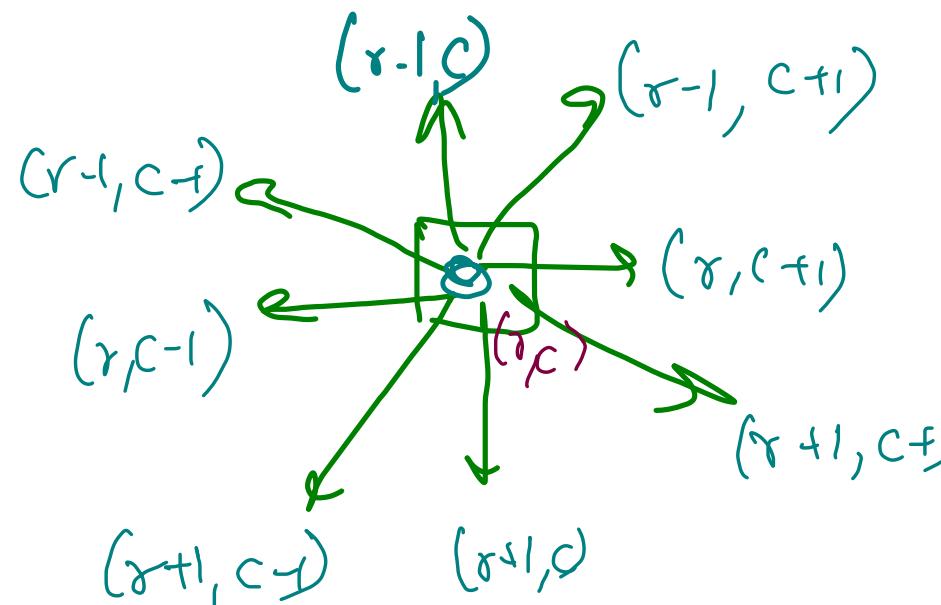
$y \Rightarrow \{0, +1, 0, -1\}$

for ($i=0$; $i < 4$; $i++$)

```
{ ff( $r+x^i$ ,  $c+y^i$ )  
}
```

8 way adjacent

{ TLDR ; $\overline{\text{TL}}$, TR , DL , DR }
edge adjacent corner adjacent



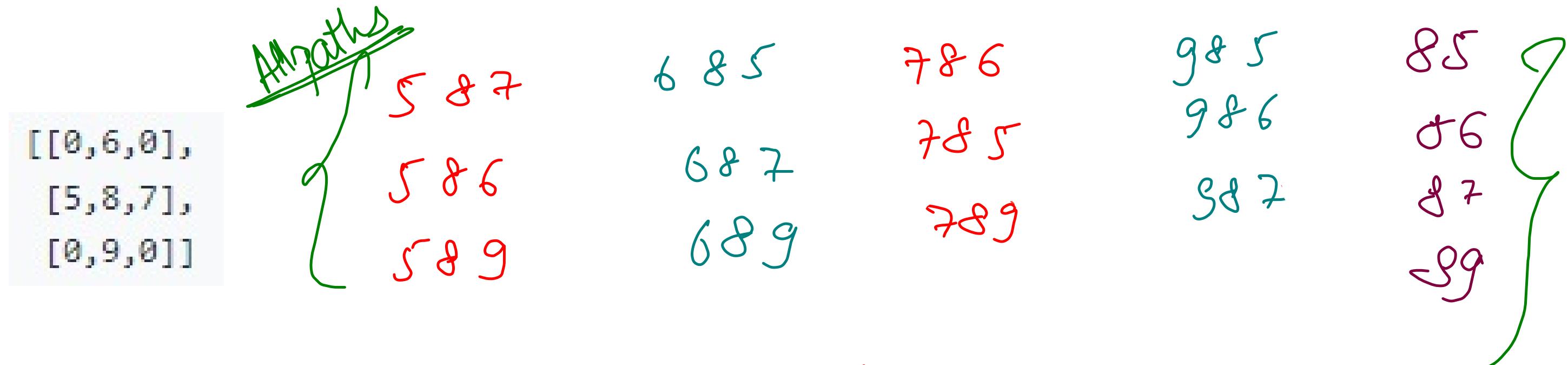
$x: \{-1, -1, 0, 1, 1, 1, 0, -1\}$

$y: \{0, 1, 1, 1, 0, -1, -1, -1\}$

for ($i=0$; $i < 8$; $i++$)

```
{ ff( $r+x^i$ ,  $c+y^i$ )  
}
```

Path with Maximum Gold {Gold Mine}



- ① you can revisit cells \rightarrow visited matrix
- ② Any node can be \rightarrow All nodes can act as a source node \rightarrow All nodes can act as a source node.
- ③ Treat 0 as a blockage.

```

public int DFS(int r, int c, int[][] grid){
    if(r < 0 || c < 0 || r >= grid.length || c >= grid[0].length
    || grid[r][c] == 0 || grid[r][c] == -1){
        return 0;
    }

    int val = grid[r][c];
    grid[r][c] = -1; // visited mark

    int maxGold = 0;
    for(int i=0; i<4; i++){
        maxGold = Math.max(maxGold, DFS(r + x[i], c + y[i], grid));
    }

    grid[r][c] = val;
    return maxGold + grid[r][c];
}

public int getMaximumGold(int[][] grid) {

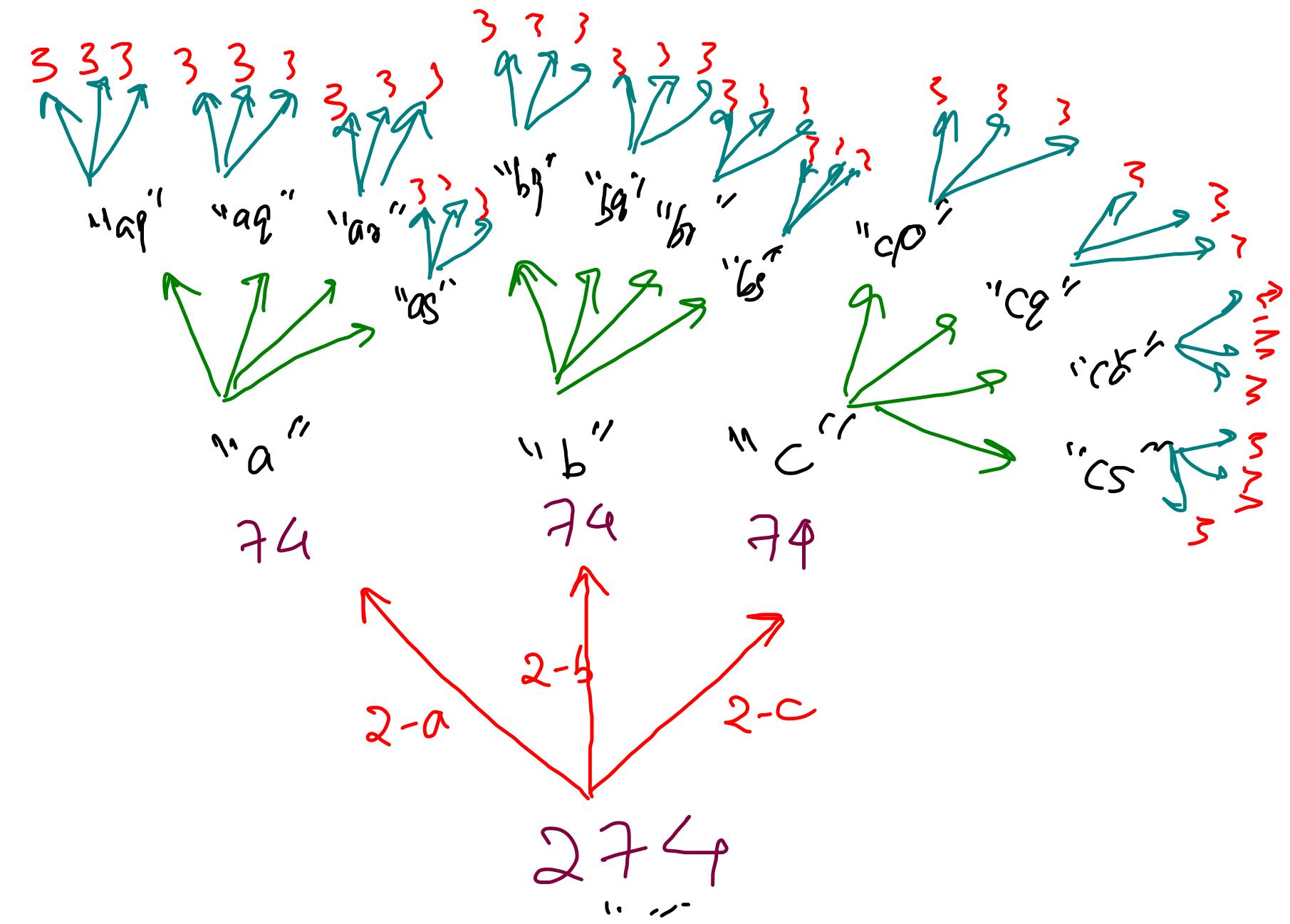
    int maxGold = 0;
    // DFS Starting from every node as source node
    for(int i=0; i<grid.length; i++){
        for(int j=0; j<grid[0].length; j++){
            int currentAns = DFS(i, j, grid);
            maxGold = Math.max(maxGold, currentAns);
        }
    }

    return maxGold;
}

```

[[0, 6, 0]],
 [5, 8, 7],
 [0, 9, 0]]

Reg pad



Bare case
3 * 4 * 3

" =

4

74

274

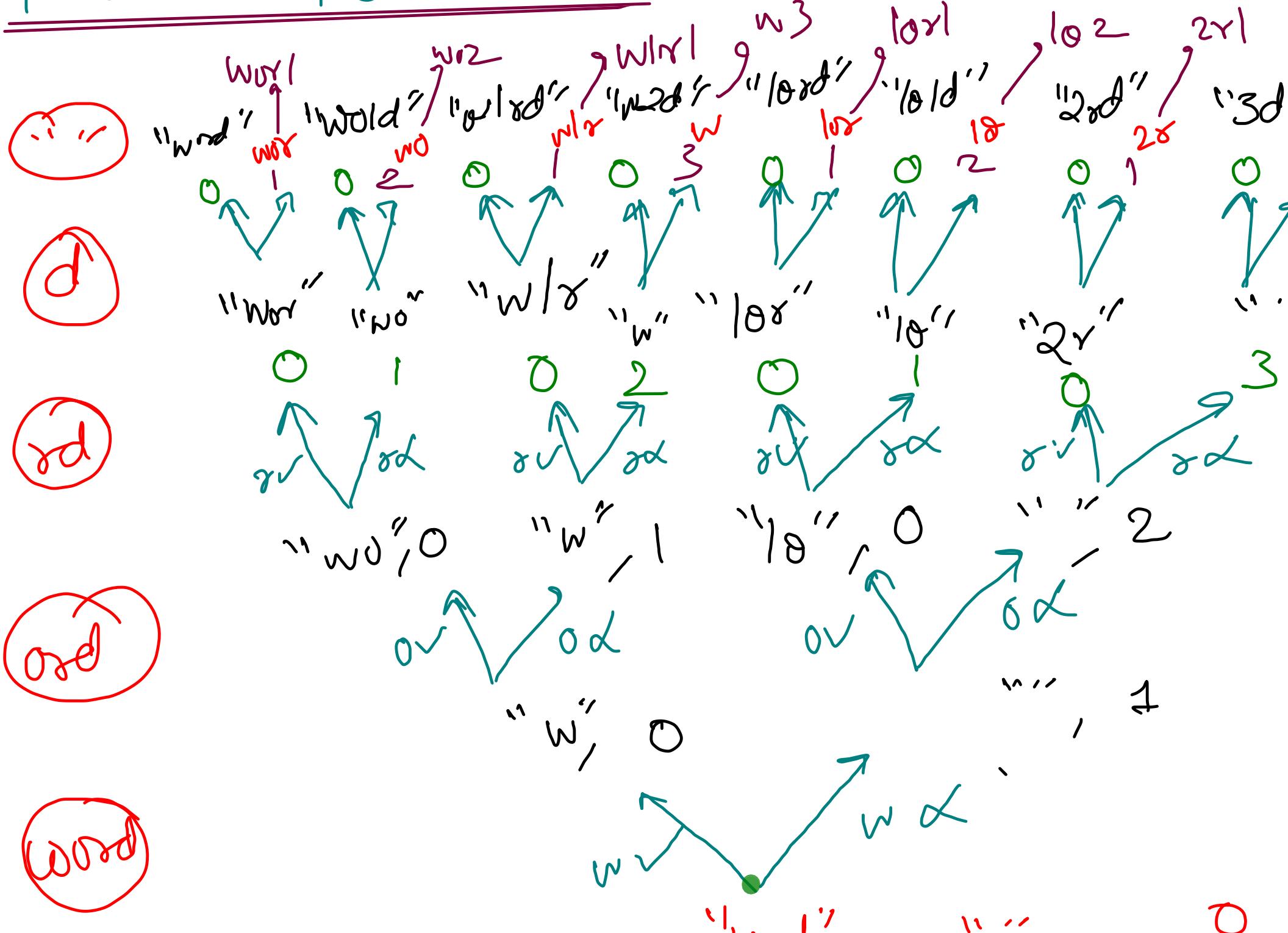
```
List<String> res;

String[] keypad = {"", "", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz"};
public void letterCombinations(int idx, String input, String output){
    if(idx == input.length()){
        if(output.length() > 0){
            res.add(output);
        }
        return;
    }

    char digit = input.charAt(idx);
    for(char letter: keypad[digit - '0'].toCharArray()){
        letterCombinations(idx + 1, input, output + letter);
    }
}

public List<String> letterCombinations(String digits) {
    res = new ArrayList<>();
    letterCombinations(0, digits, "");
    return res;
}
```

Generate Abbreviations



Input, output, count

Base case
if count > 0
append in result

① Yes

→ append count

if > 0
in output

→ append character

→ call count = 0

② No

→ pass output
as it is

→ count + 1

Code

```
class Solution {
    List<String> res;
    public void generateAbb(String input, int idx, String output, int count){
        if(idx == input.length()){
            if(count > 0) res.add(output + count);
            else res.add(output);
            return;
        }

        char ch = input.charAt(idx);
        generateAbb(input, idx + 1, output + ((count == 0) ? "" : count) + ch, 0); // yes
        generateAbb(input, idx + 1, output, count + 1); // no
    }

    public List<String> generateAbbreviations(String word) {
        res = new ArrayList<>();
        generateAbb(word, 0, "", 0);
        return res;
    }
}
```