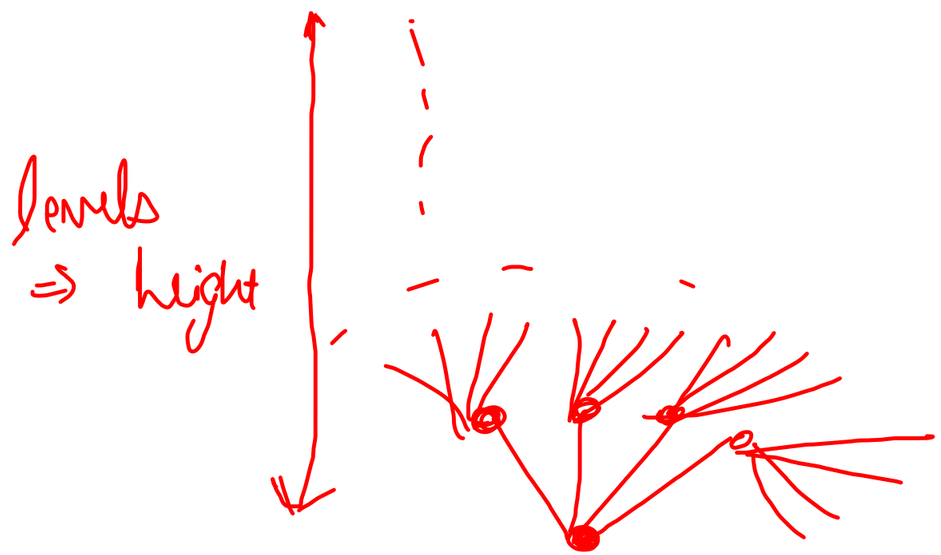
Recursion

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 fun(n) : 1 --- n

1.2 fun(n-1) : 1 --- (n-1)

1.3 M.E ⇒ Postorder : Sys0(n);

n=0 : return;

② Print decreasing

1.1 fun(n) : n --- 1

1.2 fun(n-1) : (n-1) -- 1

1.3 M.E ⇒ Preorder : Sys0(n);

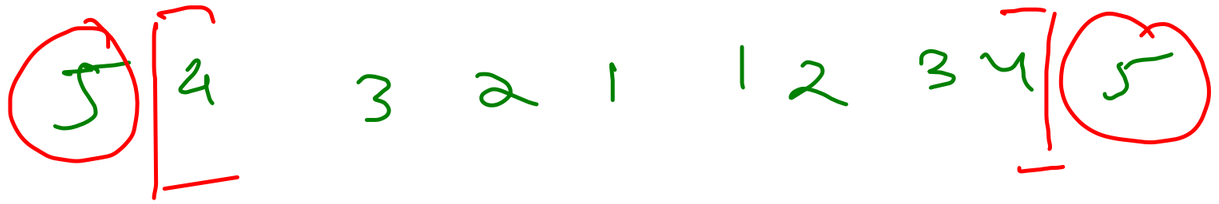
n=0 : return;

{ TC ⇒ $(1)^n$ + k*n = O(n) }

C++ Main

③ Print Increasing Decreasing

fun(5) :



fun(4):

4 3 2 1 1 2 3 4

M.E :→

Pre :
Syso(n)

fun(n-1)

Post
Syso(n)

TC ⇒ $\left(\frac{1}{}\right)^{n}$ + k * n + k * n

= 2 * n + 1 ⇒ O(n)

Power - linear

power(x, n)

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

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

power(x, n)

$x * x * x * \dots$ n times

Power - logarithmic

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

if (n%2 == 0)
return $x^{n/2} * x^{n/2}$;

if (n%2 == 1)
return $x^{n/2} * x^{n/2} * x$;

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

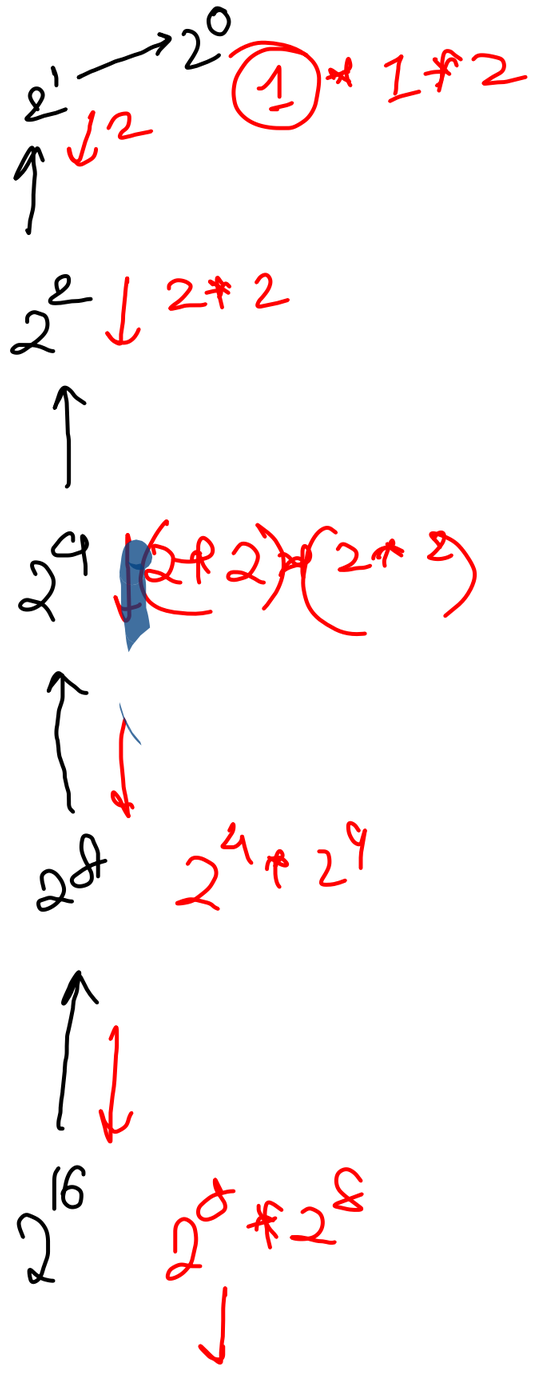
TC ⇒ $(1)^n + k * n \Rightarrow O(n)$

TC ⇒ $(1)^{\log_2 n} + k * \log_2 n$

⇒ $O(\log n) \simeq O(1)$

Power - log

log n height



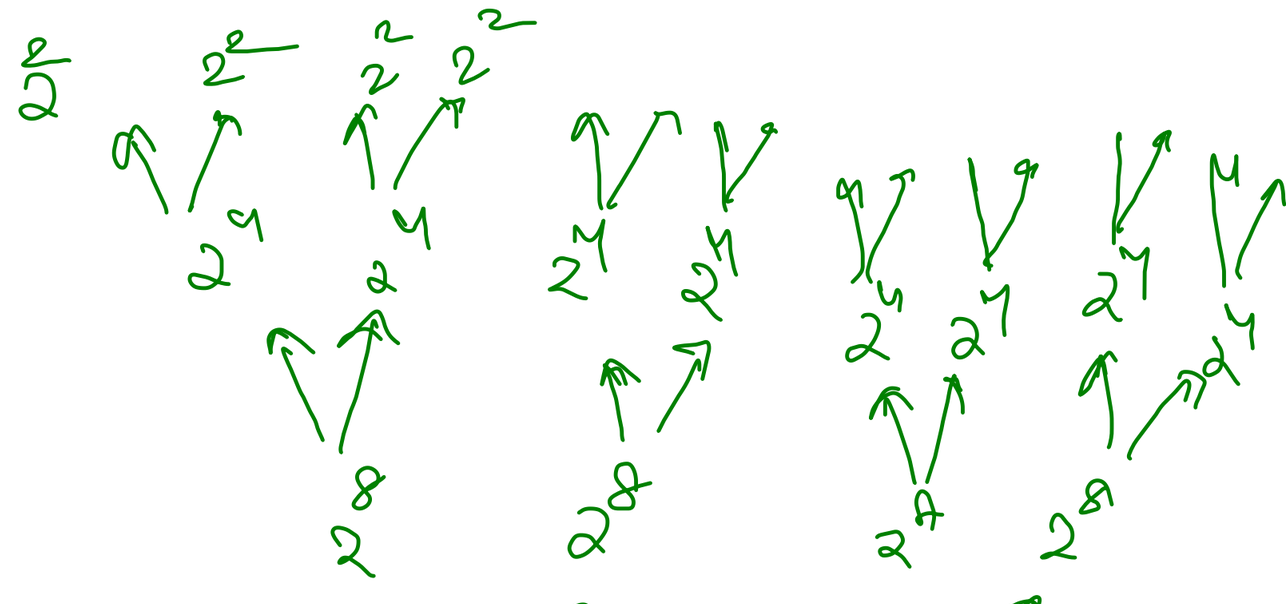

```java
public static int power(int x, int n){
    // Base Case x^0 = 1
    if(n == 0) return 1;

    // 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(n%2==0)
  return power(x, n/2) * power(x, n/2)
else
  retn pow(x, n/2) * pow(x, n/2) * x;
```

height $\Rightarrow \log_2 n$

calls $\Rightarrow 2$



$$(2)^{\log_2 n} + k * \log_2 n$$

$$= n + k \log_2 n$$

$$\Rightarrow O(n)$$

long

① 10^{18} % mod

② → (a+b) % m

Because AB

int max value

$10^9 + 7$

long ⇒ $\approx 10^{18}$

int ⇒ $2^{31} - 1 \approx 10^9$

(a+b) % m

(a*b) % m

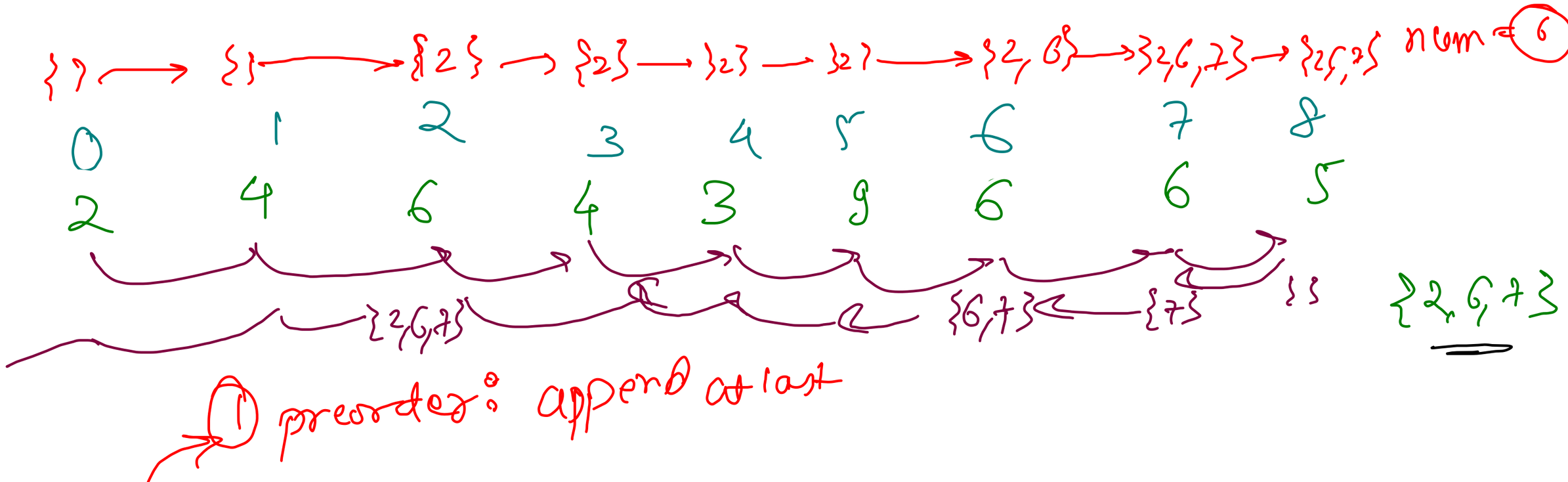
Modulus formulas

1. $(a+b) \% m = (a \% m + b \% m) \% m$

2. $(a-b) \% m = ((a \% m - b \% m) \% m + m) \% m$

3. $(a * b) \% m = (a \% m) * (b \% m) \% m$

4. Division $\Rightarrow$ **Modulo Inverse**

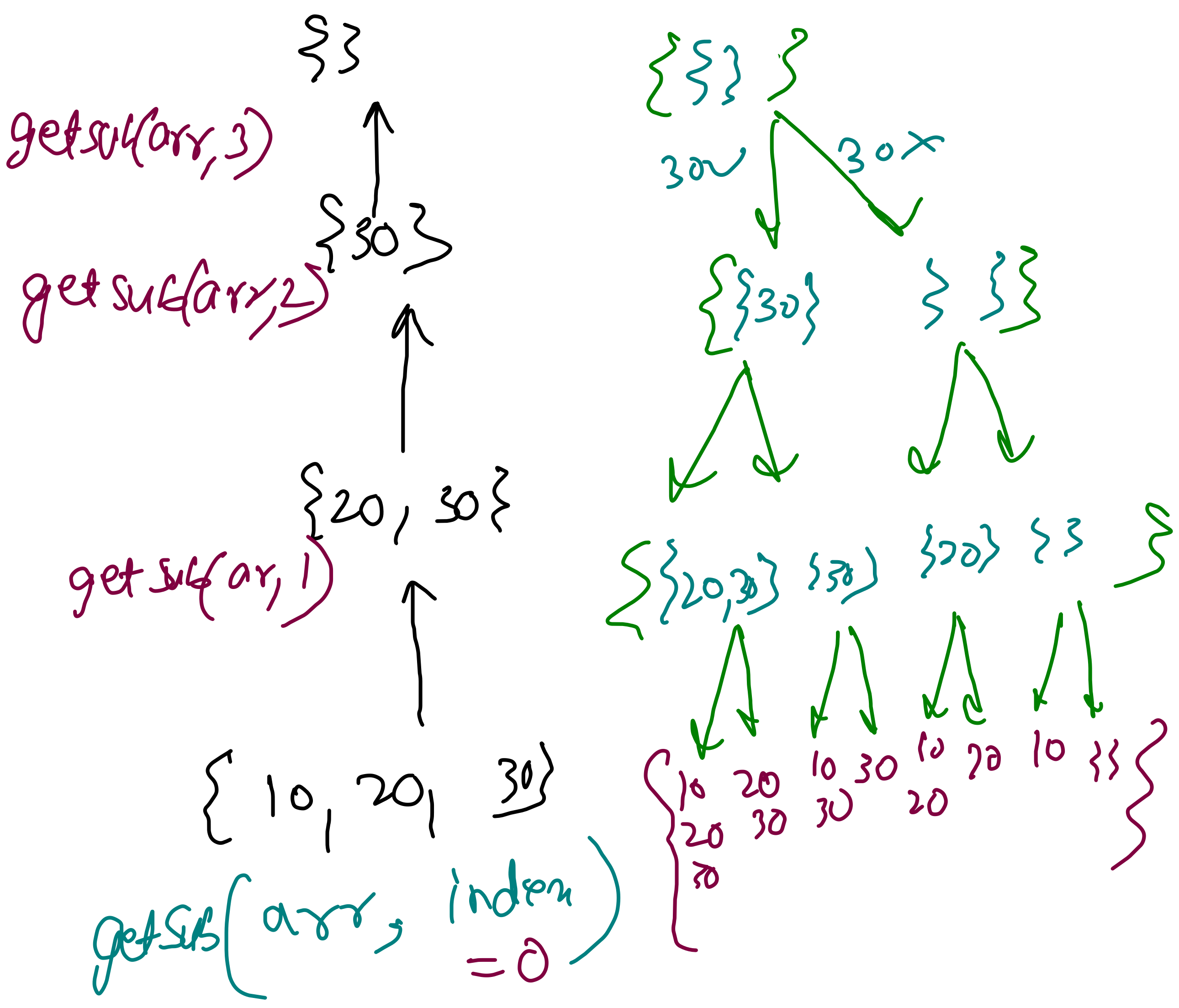


① preorder: append at last

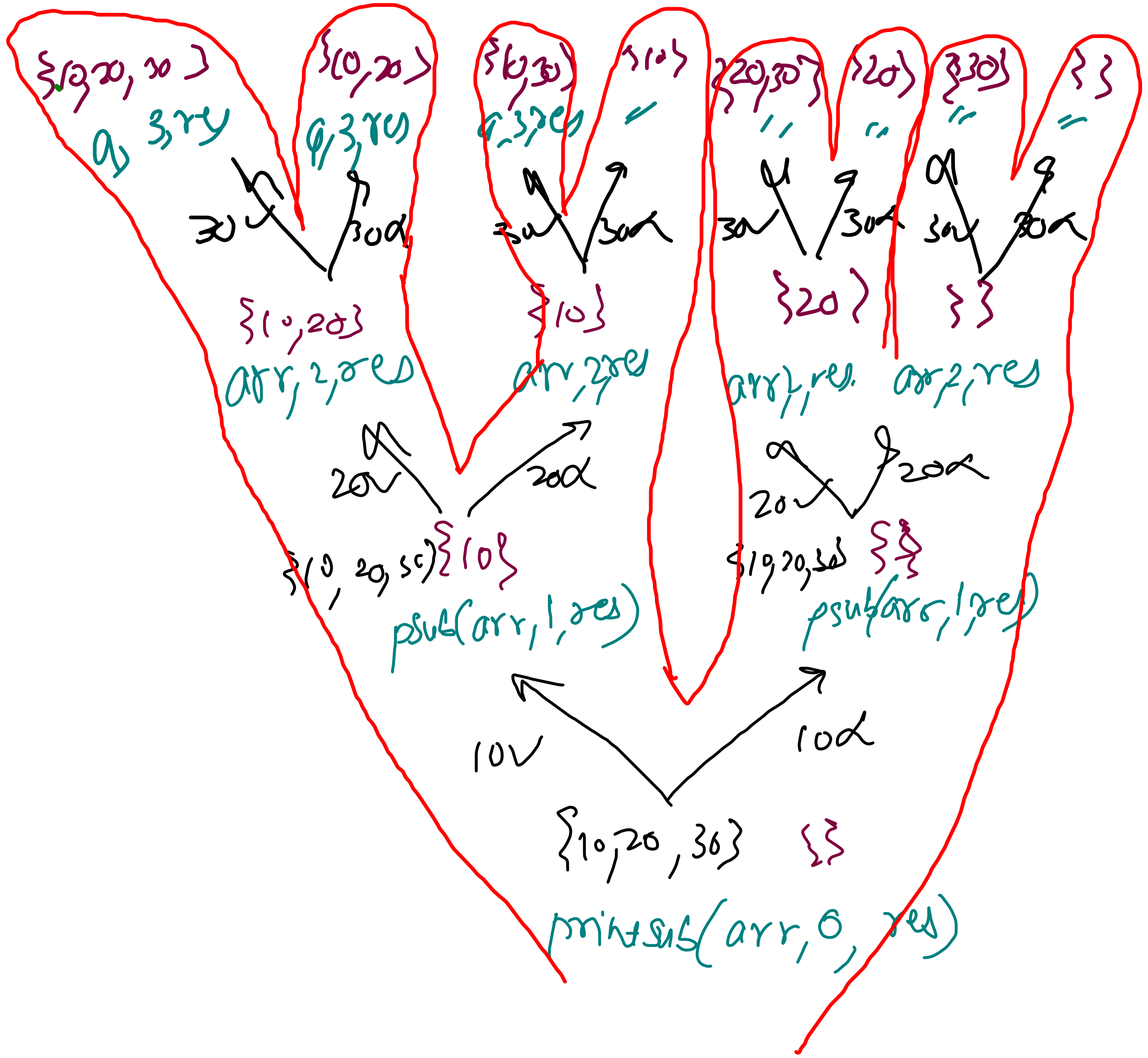
fun(arr, 0, res, num)

② postorder: append at first

Get Subsequence



Print Subsequence



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

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 } → 30 Q

3. String Permutations {Unique}
   - → Box on level
   - → Item on level
4. Palindromic Permutations

Homework :-

- Factor combinations
- Dictionary Order
  - → small
  - → large

Combinatn

^nC_r

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

i i — —

4C2

identical items

Permutation

Selectn + Arrangement

^nP_r

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

non-identical items

12 — — 21 — —

1 — 2 — 2 — 1 —

1 — — 2 2 — — 1

— 12 — — 21 —

— 1 — 2 — 2 —

— — 12 — — 21

$^4P_2 = {}^4C_2 * 2!$

Combinations → Box on level

box = 3

box = 2

box = 1

box = 0

{1, 2}

n = 4

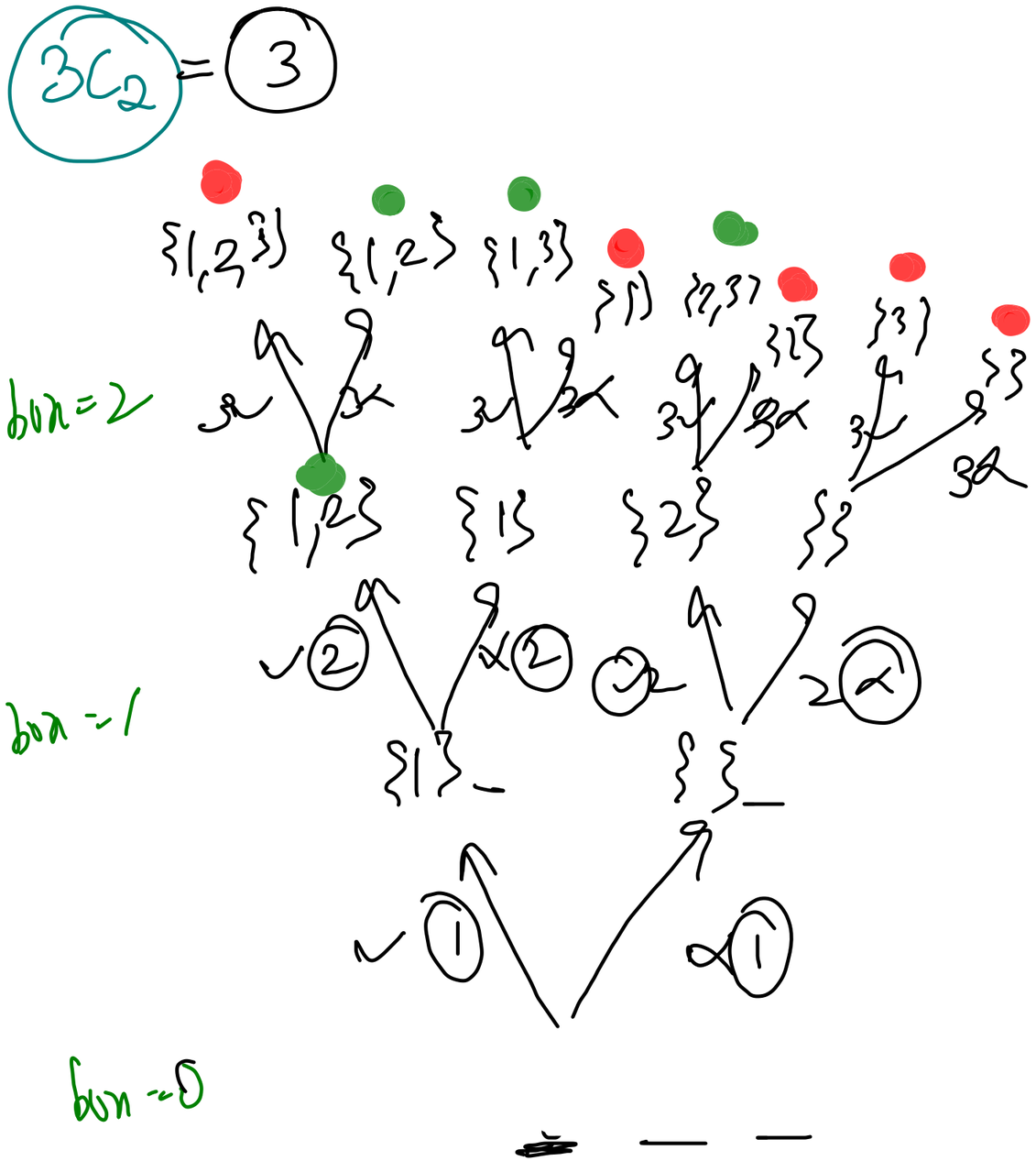
r = 2

$$2^n = {}^nC_0 + {}^nC_1 + {}^nC_2 + \cdots {}^nC_n$$

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

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

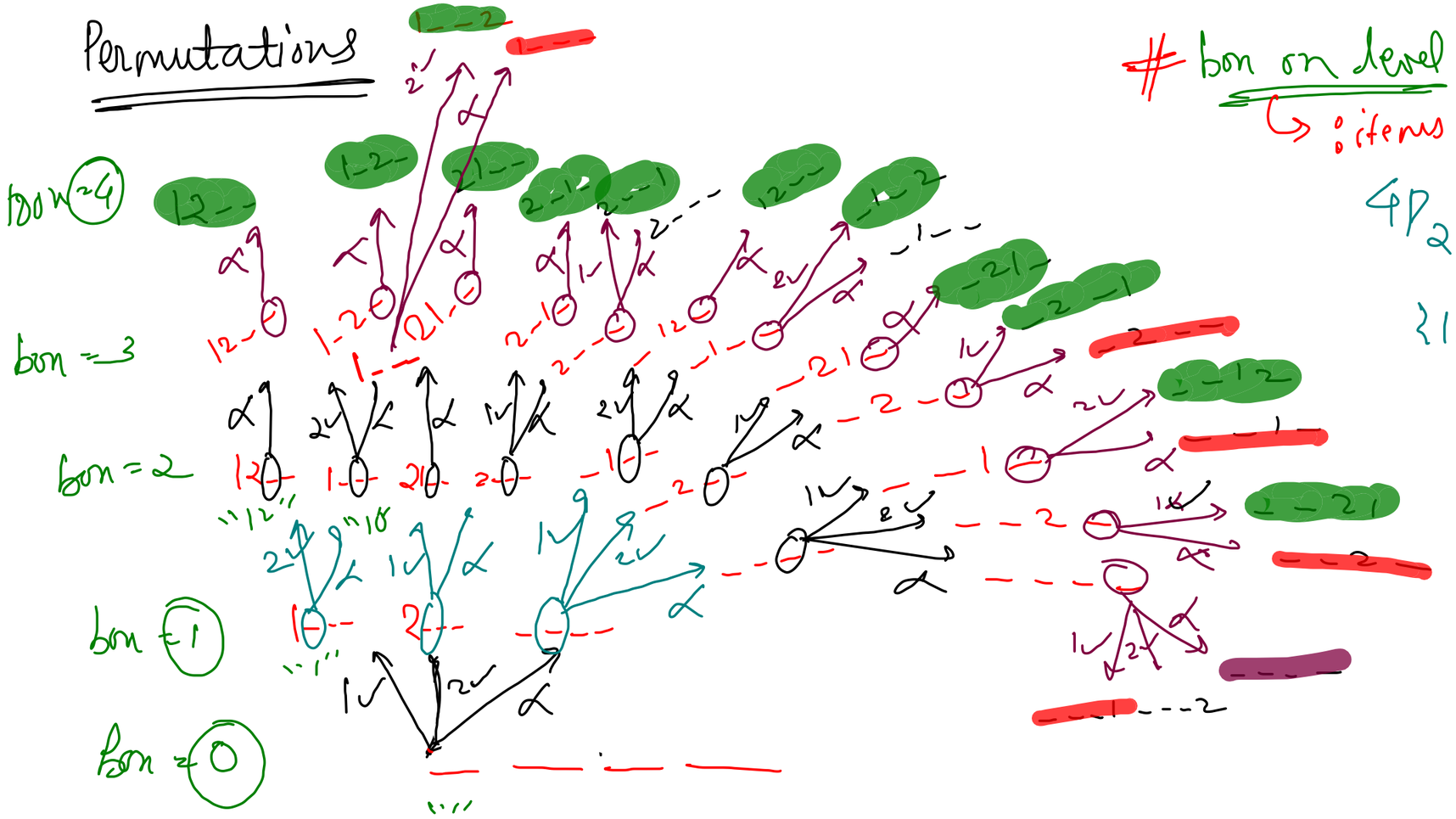
TC ⇒ $(2)^n$ + .. + ...




```java
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



bon on level

↳ : items were on ph&by

$4P_2$ + ✗

{1, 2}

$4P_0 + 4P_1 + 4P_2$

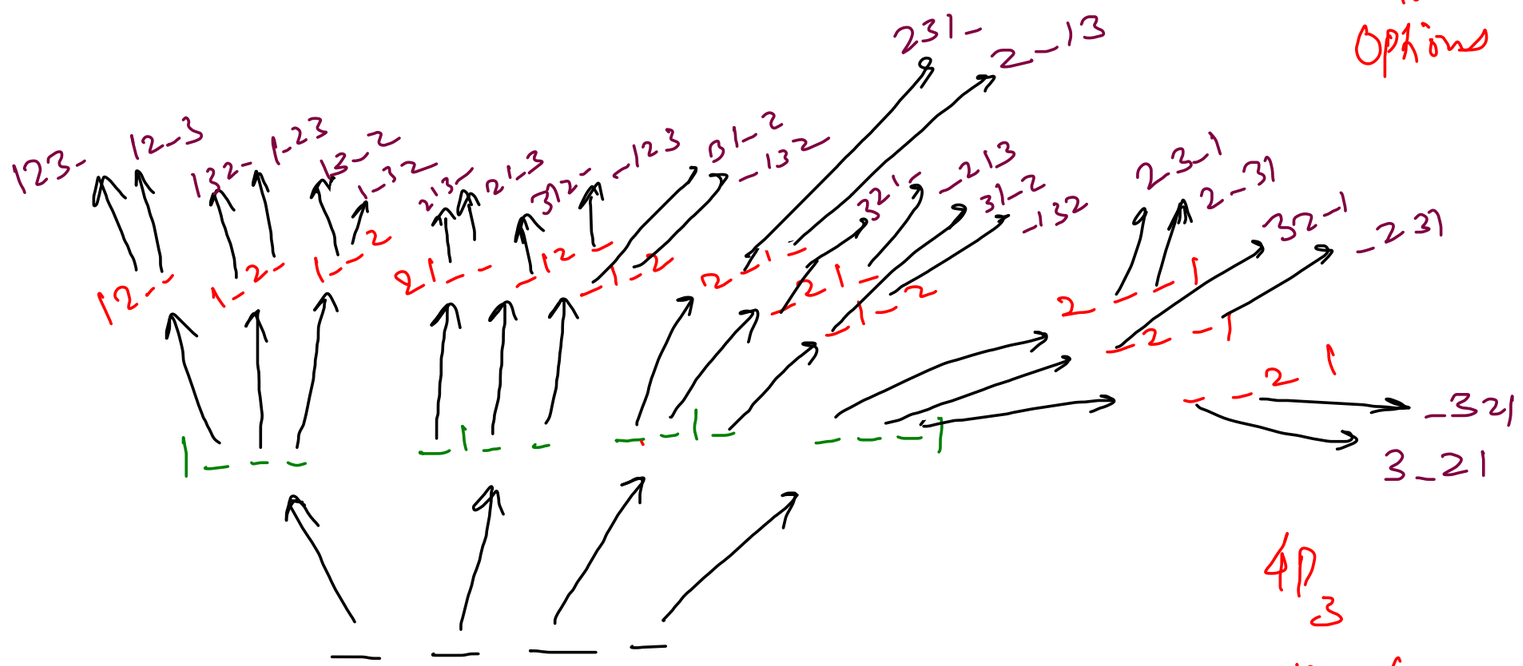
Permutation - 2

items on level: boxes are Options

item = 3

item = 2

item = 1



$^4P_3$

n = 4

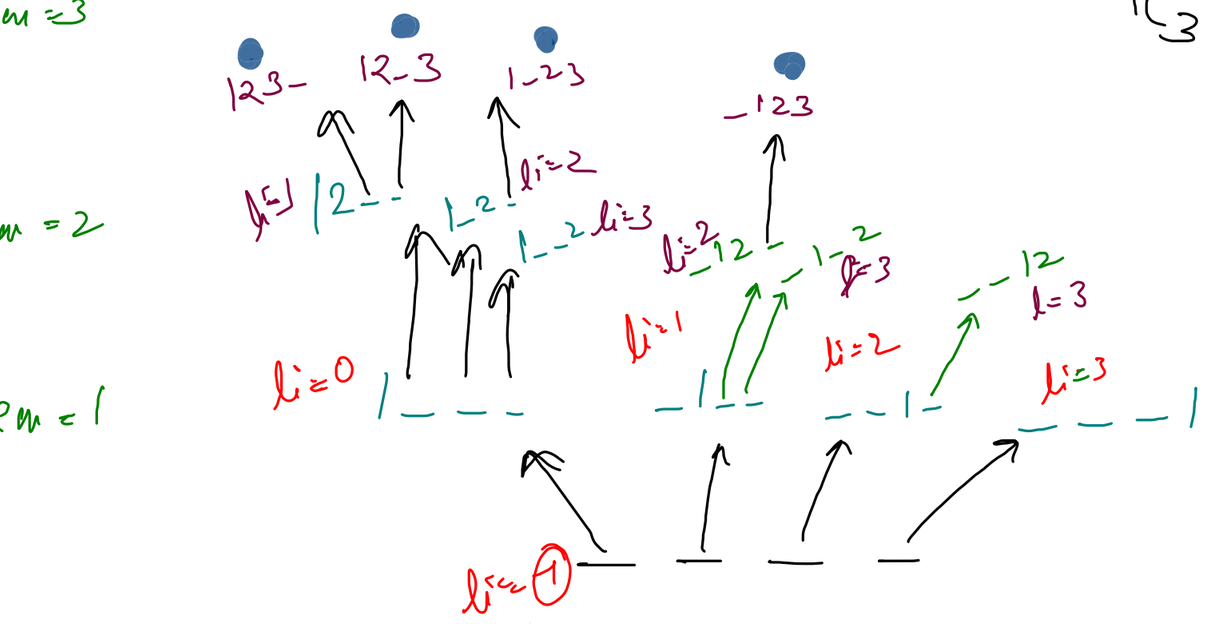
b = 3

```java
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 -> Chose 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;
      }
  }
}
```

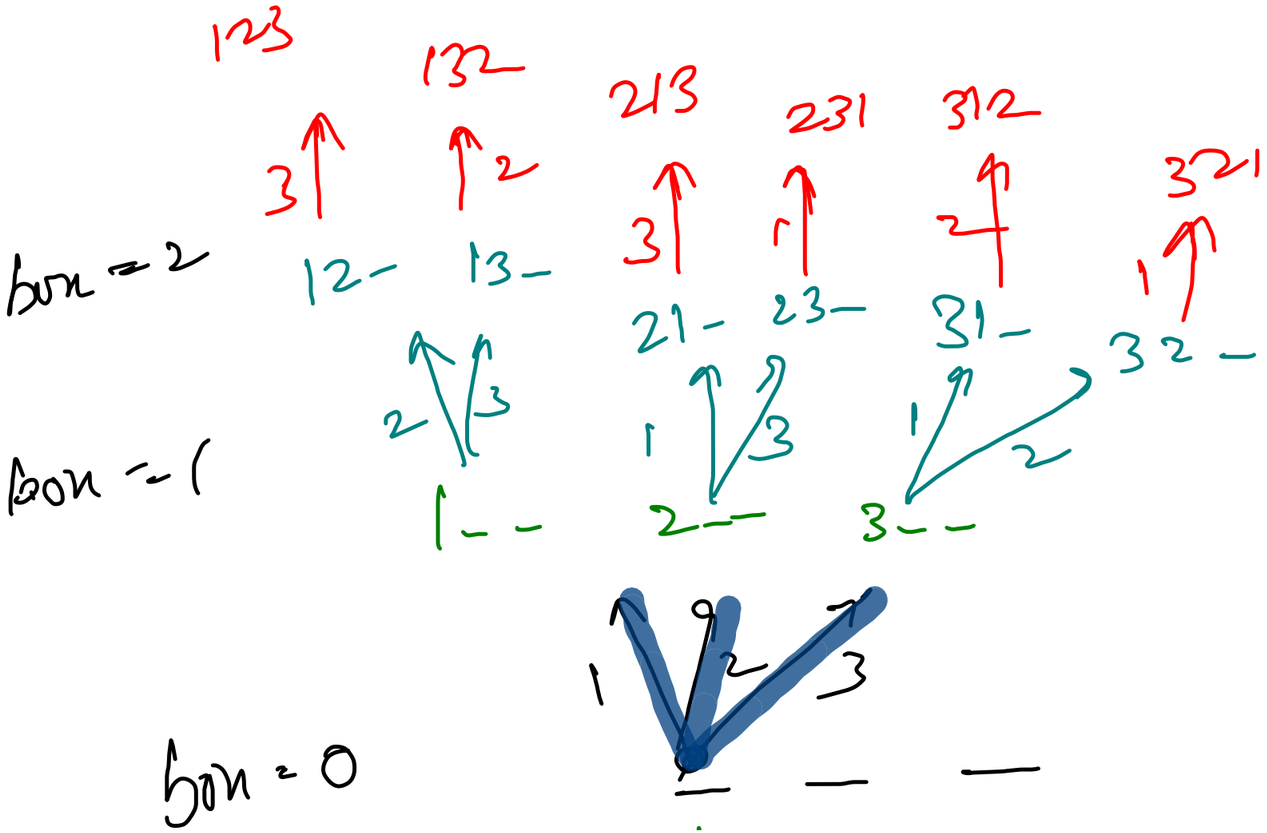
Combinations -2

{Items ⇒ level ; Option ⇒ box}

$^4C_3 = 4$




```java
public static void combunations(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 -> Chose box
  for(int i=lastItemIdx + 1; i<boxes.length; i++){
      if(boxes[i] == 0){
          boxes[i] = ci + 1;
          combunations(boxes, ci + 1, ti, i);
          boxes[i] = 0;
      }
  }
}
```

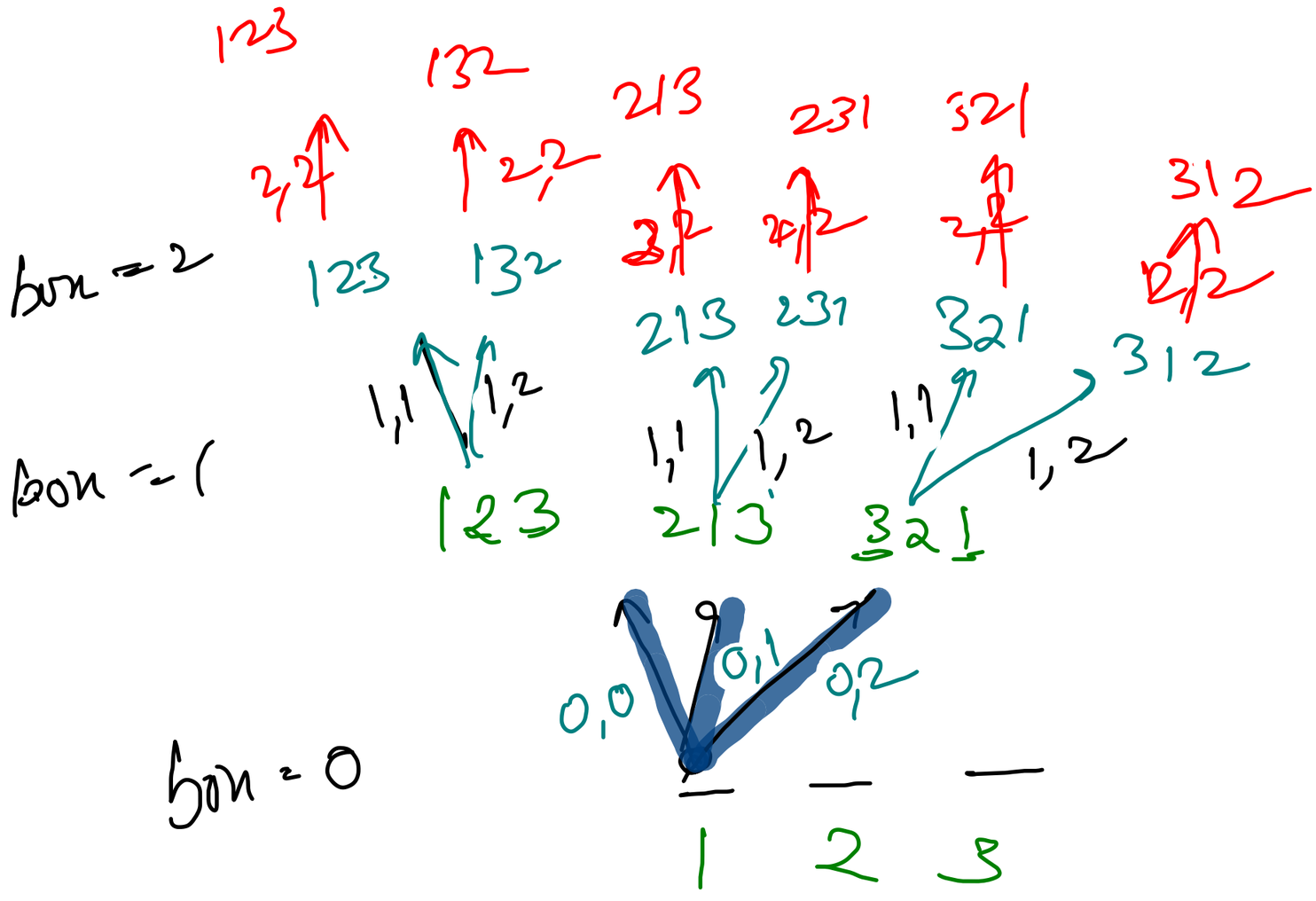


box = items

items

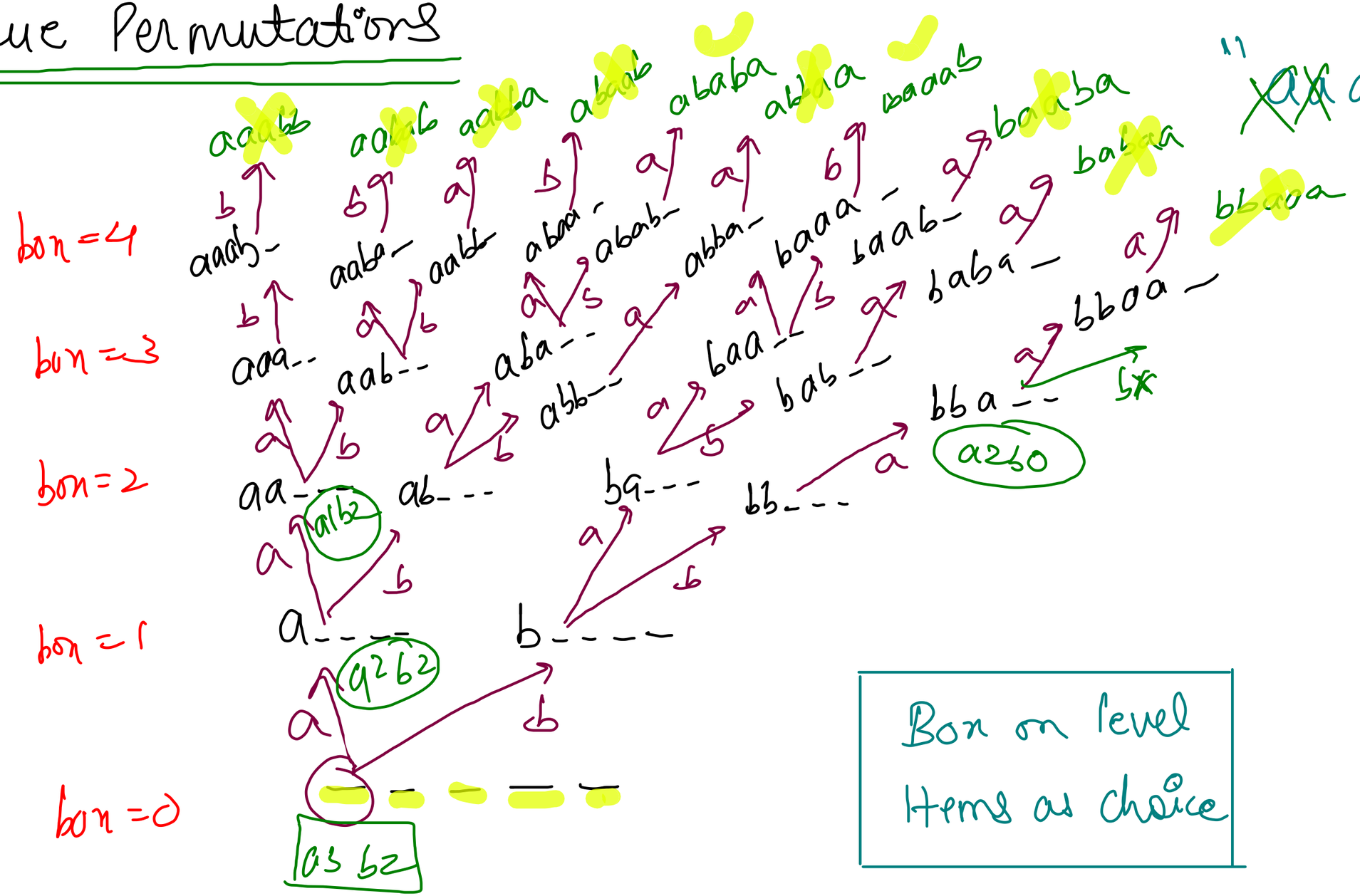
boxes = items

{1, 2, 3}




```java
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



"aaab"

= 5!

3! x 2! x 1!

= 5 x 4 x 3 x 2

= 10

Box on level
Items as choice

helper

```java
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);
        }
    }
}
```

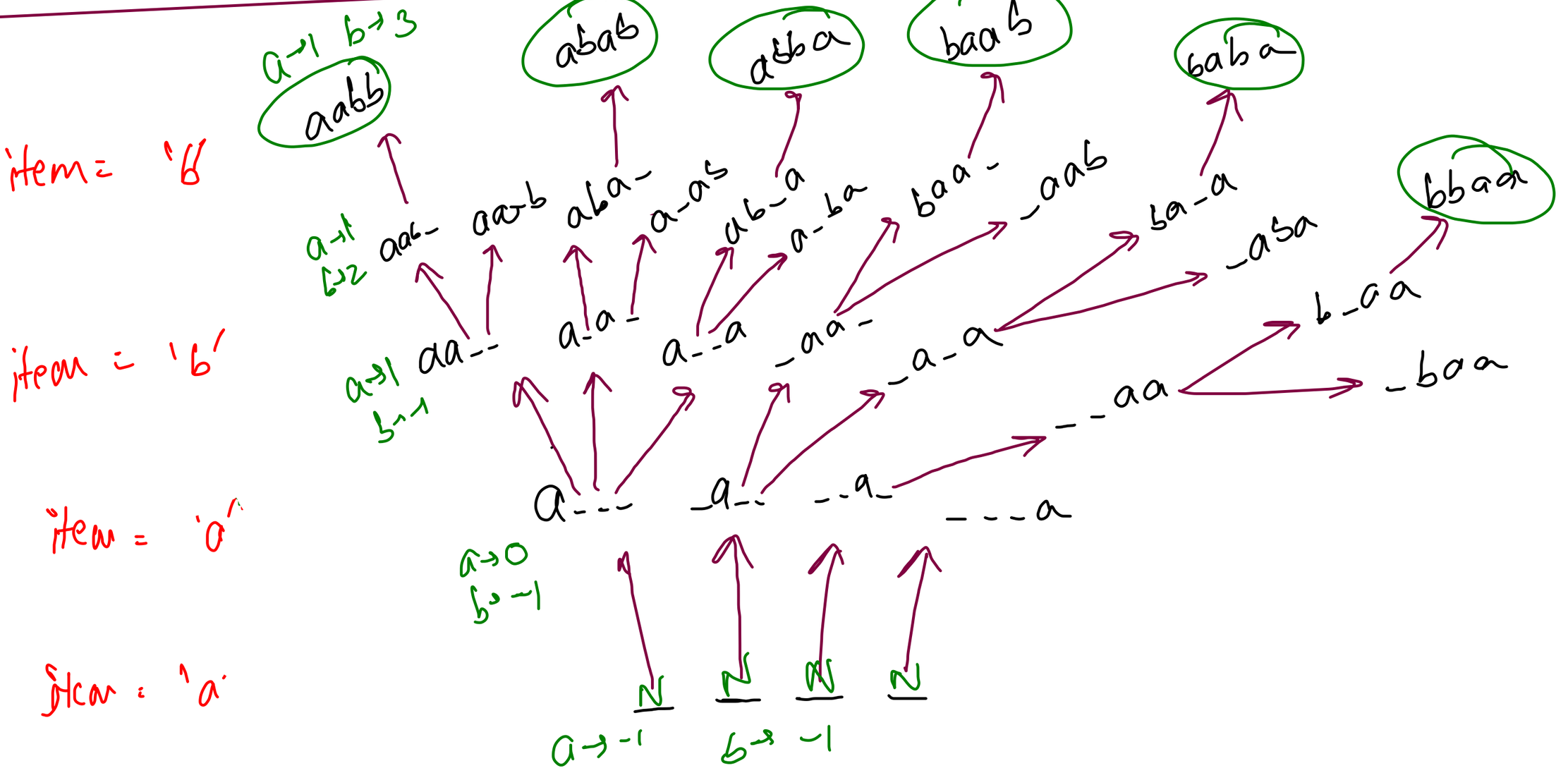
main

```java
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<>());
```

Item on level
Box as choice

'aabb'

$$\frac{4C}{2!2!} = \frac{4\times3\times2}{2\times2} = 6$$



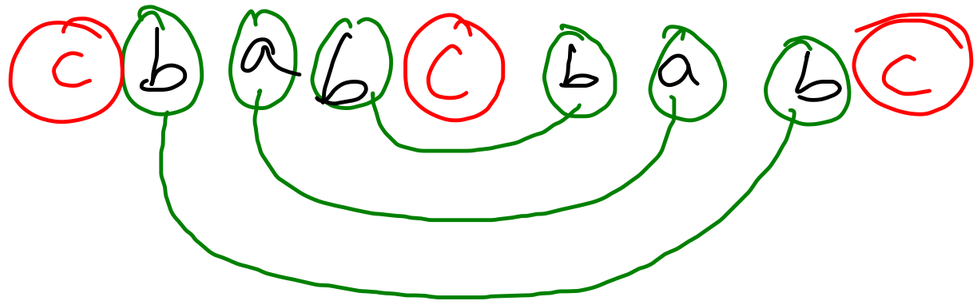
helper

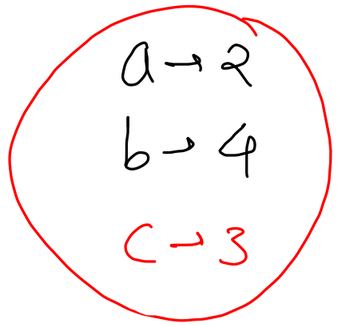
```
public void permutations2(HashMap<Integer, Integer> lastIdx, int[] nums,
        int currentItem, List<Integer> ans){
    if(currentItem == nums.length){
        List<Integer> copy = new ArrayList<>(ans);
        res.add(copy);
        return;
    }

    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);
        }
    }
}
```

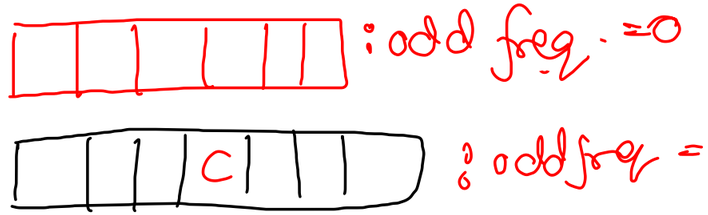
main

```
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 - 1







all char ✓ even freq

atmost 1 char with odd freq

```java
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;
}
```

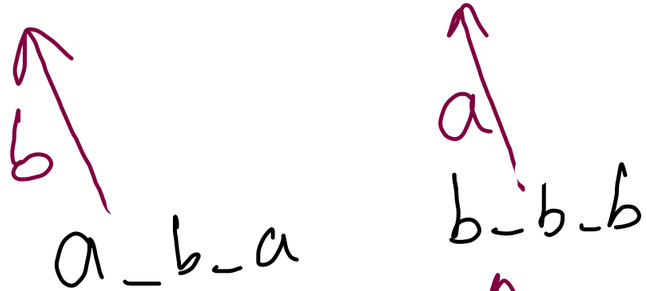
All Palindromic Permutations {Pg-11}

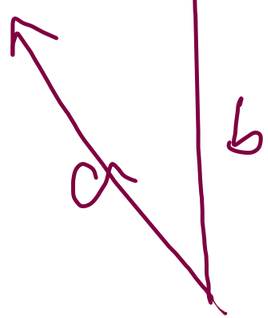
aabbb

a2b2

abbba

baba b





box on level

b

aaaabbc

a4b2c

aabcbaa

5

aa_c_aa

a

b

a _ c _ a

a2b2c0

abababa

a

ab_c_ba

baacaaca

a

ba_c_ab

a

b _ c _ b

a

b

C

— — — — — —

CB

1a4b2c0

n-1-CB

box on level item as choice

K words { Repitition not allowed } {Combinations}

a2 b3 c2 d2 e

```
aabbbccdde
3
```

unique characters : 5 bones

1. Bon on level
2. Hem on level :→ 5 items
   → k characters

a, b, c, d, e

3 items

{ a b c d e }

_ _ _ _ _

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

{ }

abcd abc- ab-d ab-- a-cd a-c- a--d a--- -bcd -bc- -b-d --cd

(d) d✓ d✗

abc- ab-- a-c- a--- -bc- -b-- --c- ---d

(c) c✓ c✗ c✓ c✗ c✓ c✗ ----

ab-- a--- -b-- ----

(b) b✓ b✗ b✓ b✗

a---

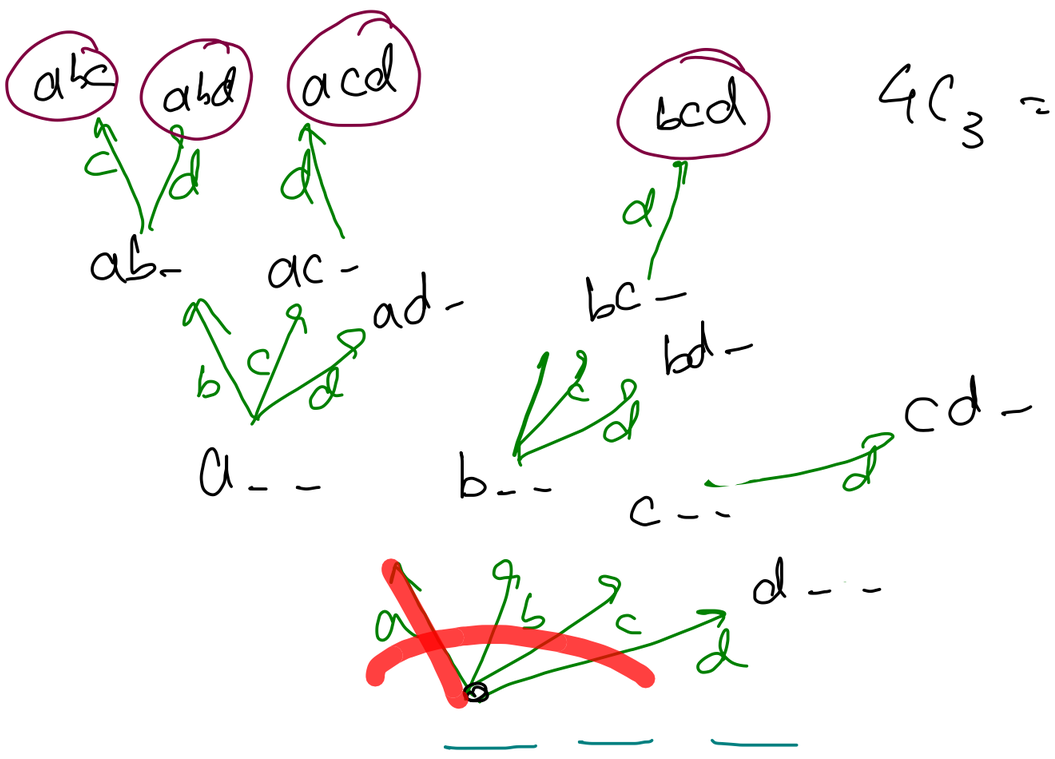
(a) ----

a✓ a✗

a 2b 3c 2d

{a, b, c, d}

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



$4C_3 = 4$

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

```
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);
  }
}
```

a253c2d

{a,5,c,d3

Today's Questions

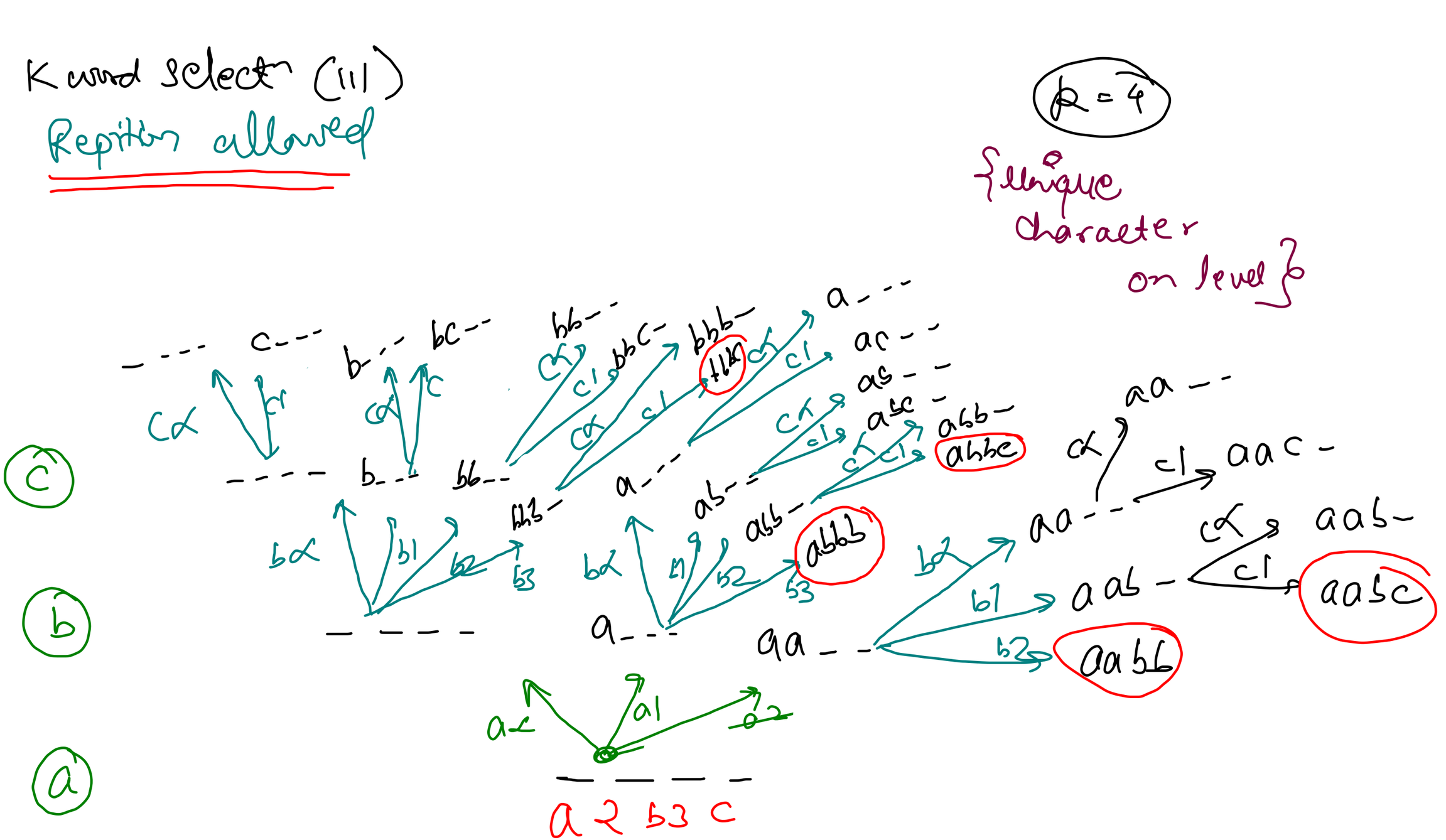
1. Words - K Selection - 1
   Words - K Selection - 2
   } String Combination → Repitition not allowed
2. Words - K Length Words - 1
   Words - K Length Words - 2
3. Words - K Selection - 3
   Words - K Selection - 4
   } String Combinaon → Repitition allowed
4. Words - K Length Words - 3
   Words - K Length Words - 4
5. Coin Change - Combinations - 1
   Coin Change - Combinations - 2
6. Coin Change - Permutations - 1
   Coin Change - Permutations - 2

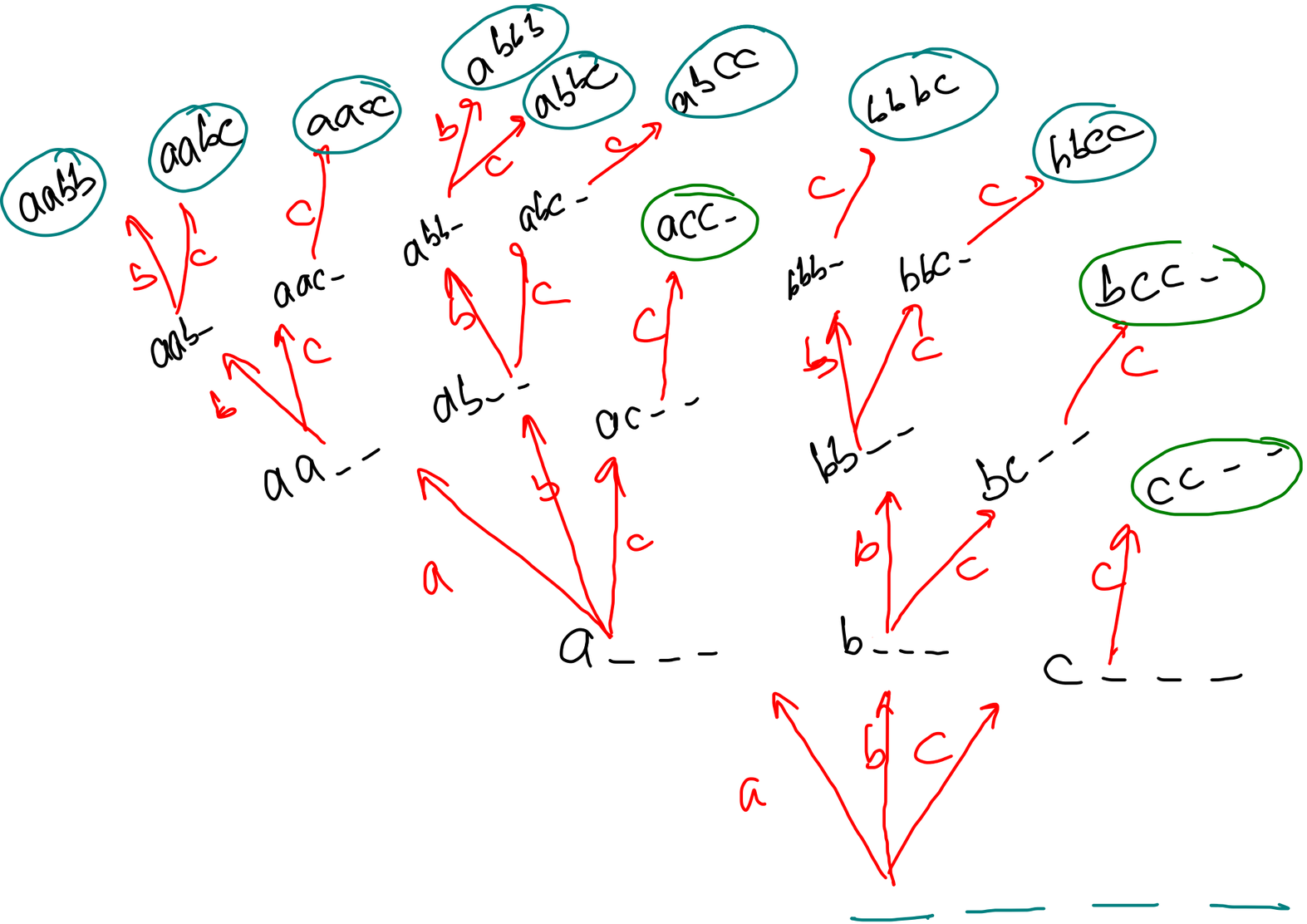
Kword select (111)

Repition allowed

$k = 4$

{Unique character on level}





Kwords
Select -IV
Combinar
Repitition
allowed

a2 b3c2

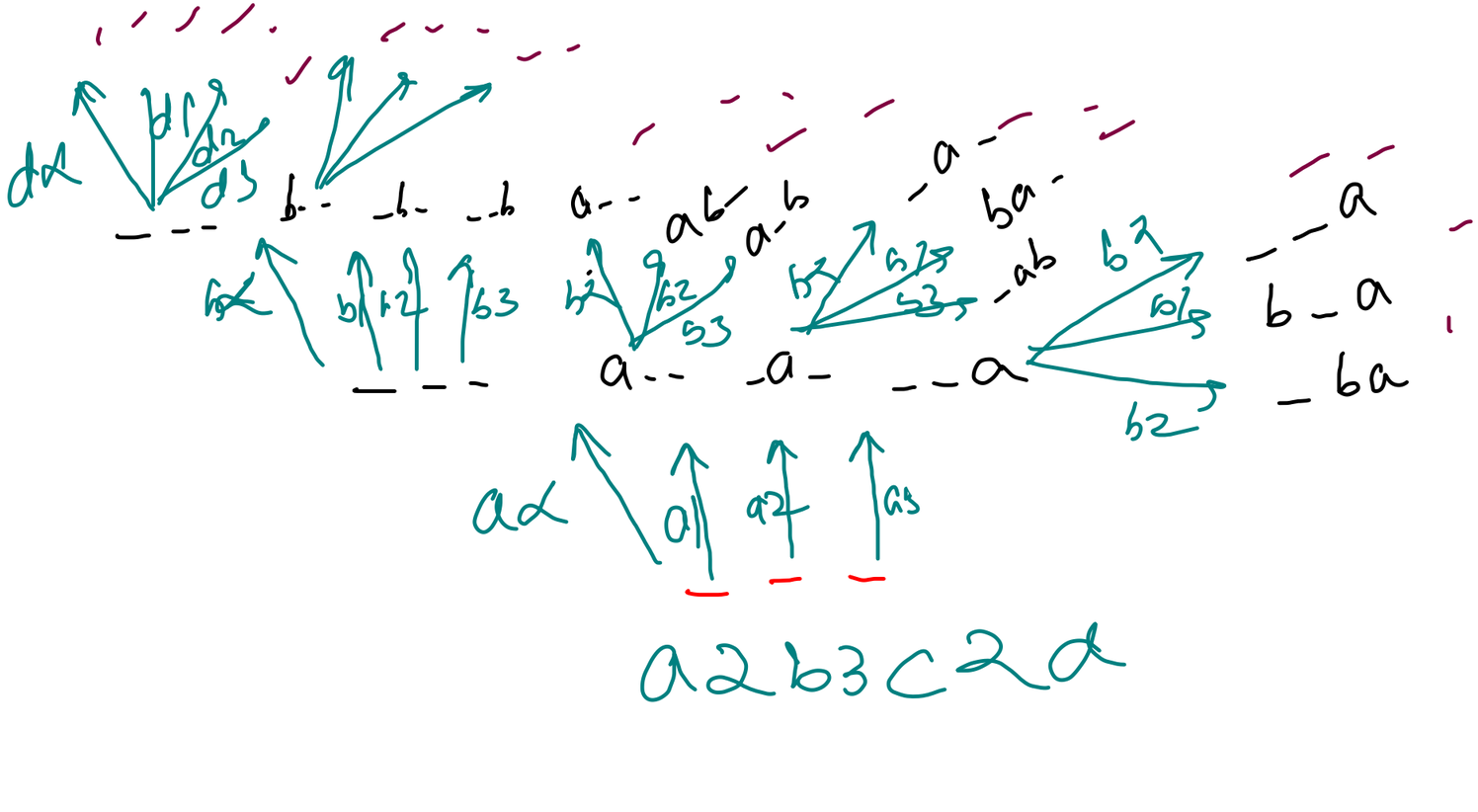
```java
public static void generateSelection(String ustr, int cs, int ts, int lc, HashMap<Character, Integer> freq, String asf) {
  if (cs > ts) {
    System.out.println(asf);
    return;
  }
  if(lc == ustr.length()){
      return;
  }

  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

Unique character on level

① Repitition not allowed
→ Unique char
→ Slot on level

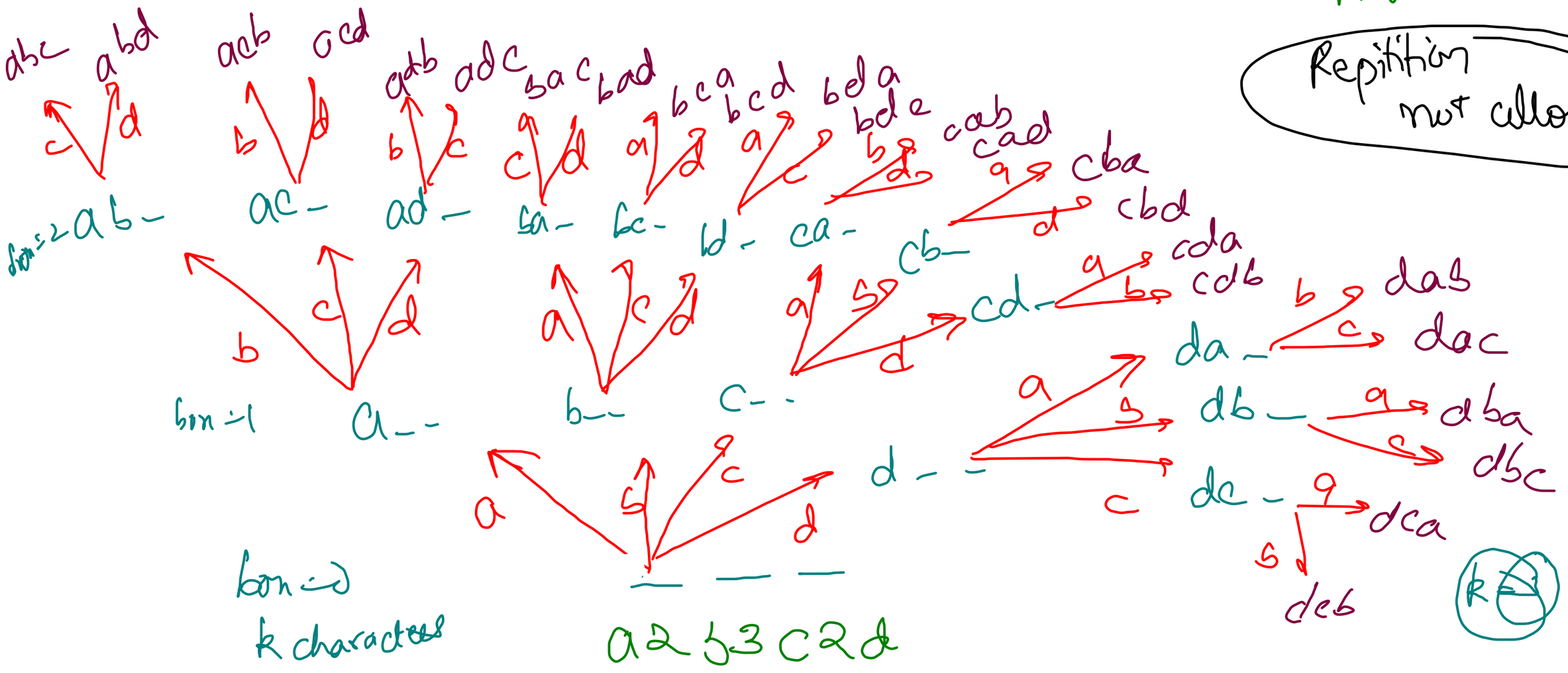
② Repitition allowed
→ Unique char
→ Slot on level



Bon on level

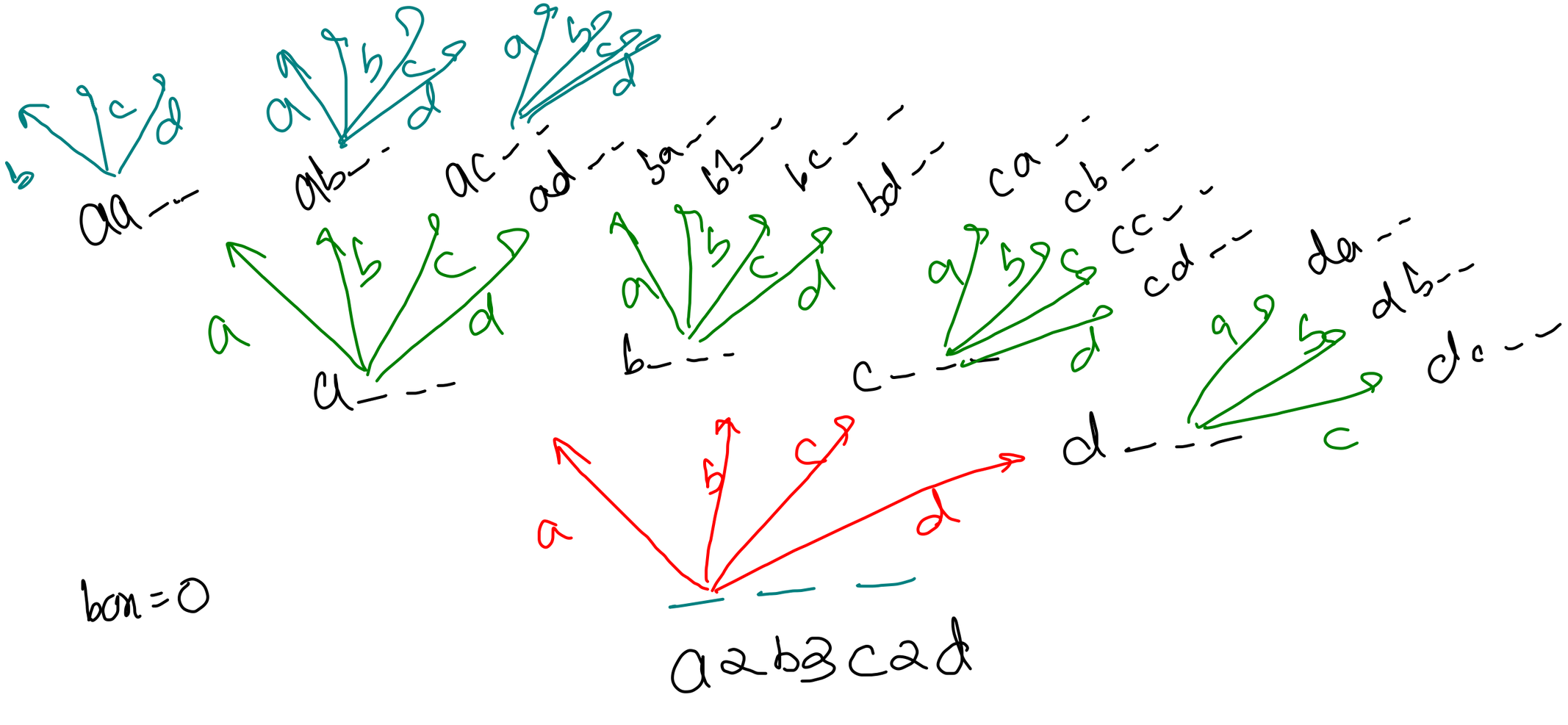
k words

Repition not allowed



```java
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);
        }
    }
}
```

Repitition allowed



box=0

a2b3c2d

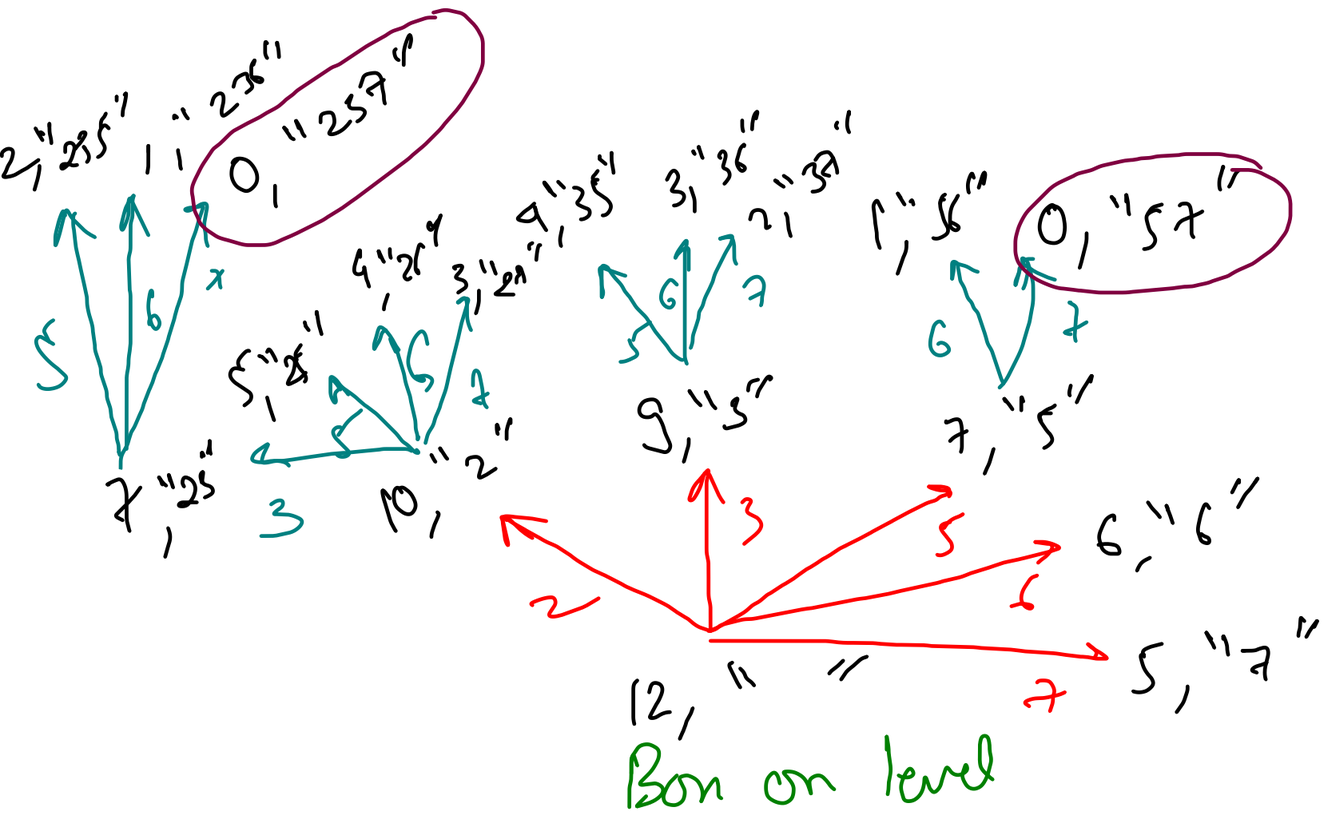
```java
public static void permutation(String res, String ustr, HashMap<Character, Integer> unique, int k){
    if(res.length() == k){
        System.out.println(res);
        return;
    }

    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);
        }
    }
}
```

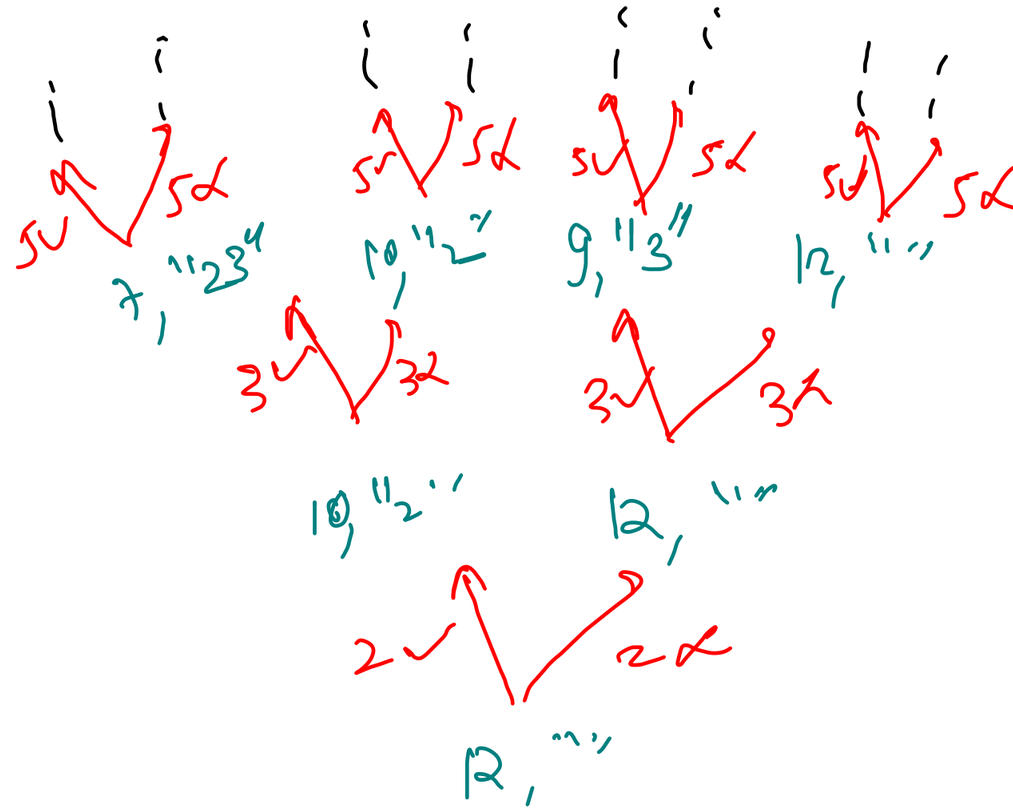
Coin Change Combination(II)

{2, 3, 5, 6, 7}

12

Finite Supply



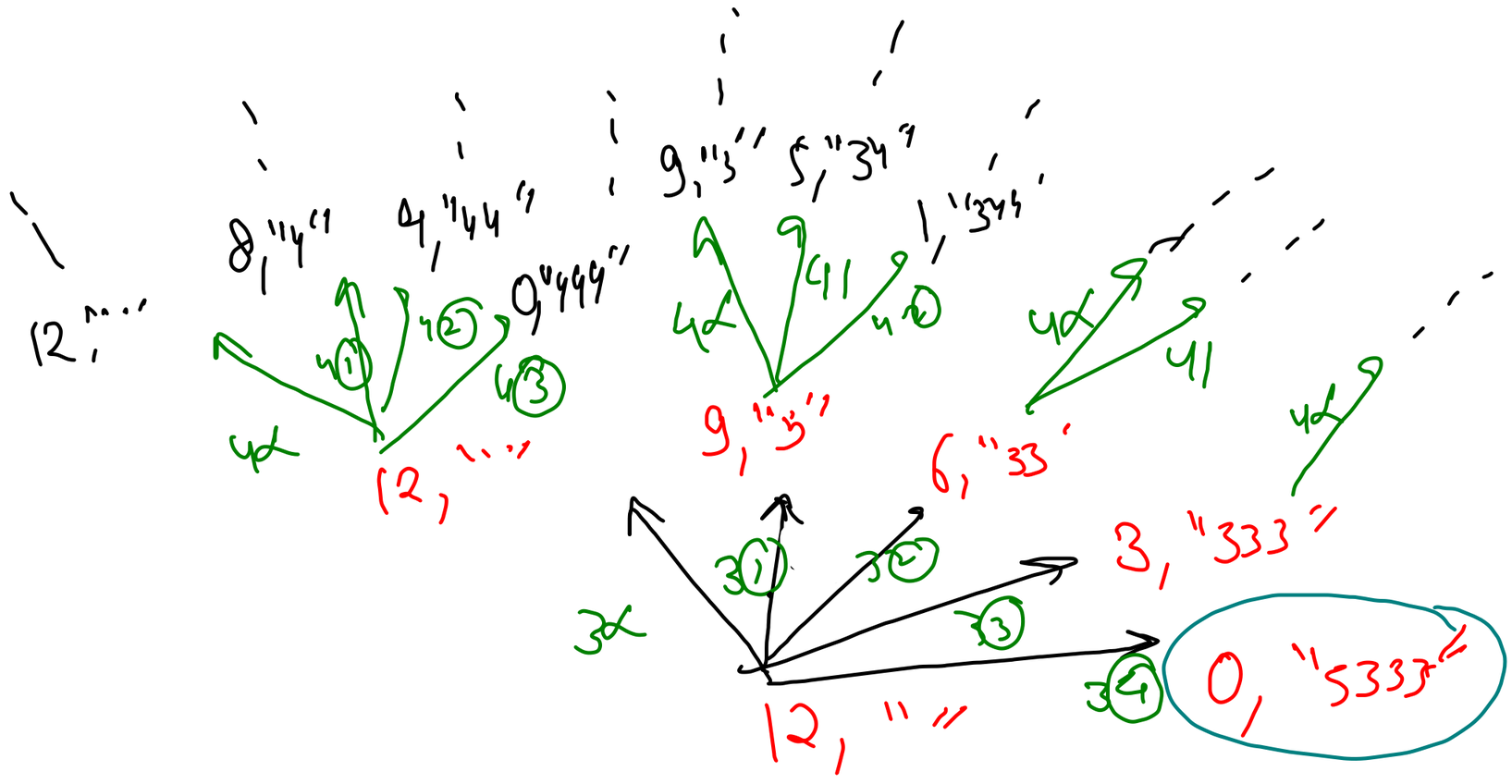


Finite Supply

```java
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} (12)




```java
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 - Permutatn → I
   Coin change - Permutatn → 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
   Queens Combinations - 2d As 1d - Queen Chooses
4. Nqueens Combinations - 2d As 1d - Queen Chooses
   Nqueens Permutations - 2d As 1d - Queen Chooses
5. Nknights Combinations - 2d As 1d - Knight Chooses

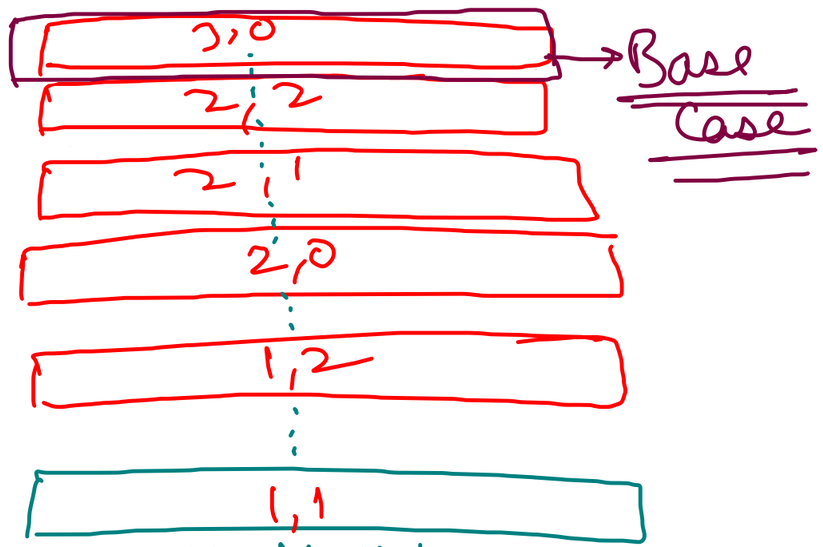
1. Recursion Tree {RT}
2. RT → level → parameter
   option → for
3. Backtracking
4. Rcs → order
   +ve & -ve base case
5. Dry run

Dictionary Order- Small "cab"

```cpp
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 } k = 3

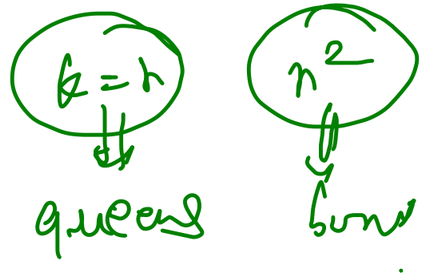


Bottom level

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

3,0

k = n



Total combinations = $2^{N*N} = 2^9$

$2^9 = {}^9C_0 + {}^9C_1 + {}^9C_2 + {}^9C_3 + \dots {}^9C_9$

Result

```java
// 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	5
1	6	7	8	9	10	11
2	12	13	14	15	16	17
3	18	19	20	21	22	23

4 rows :- N = 4

6 columns :- M = 6

(1) cell → rowNo → cellNo / M

→ colNo → cellNo % M

(2) row, col → cellNo → rowNo * M + colNo

2 * 6 + 4 = 16

3 * 6 + 1 = 19

16 / 2 = 2

16 / 2 = 4

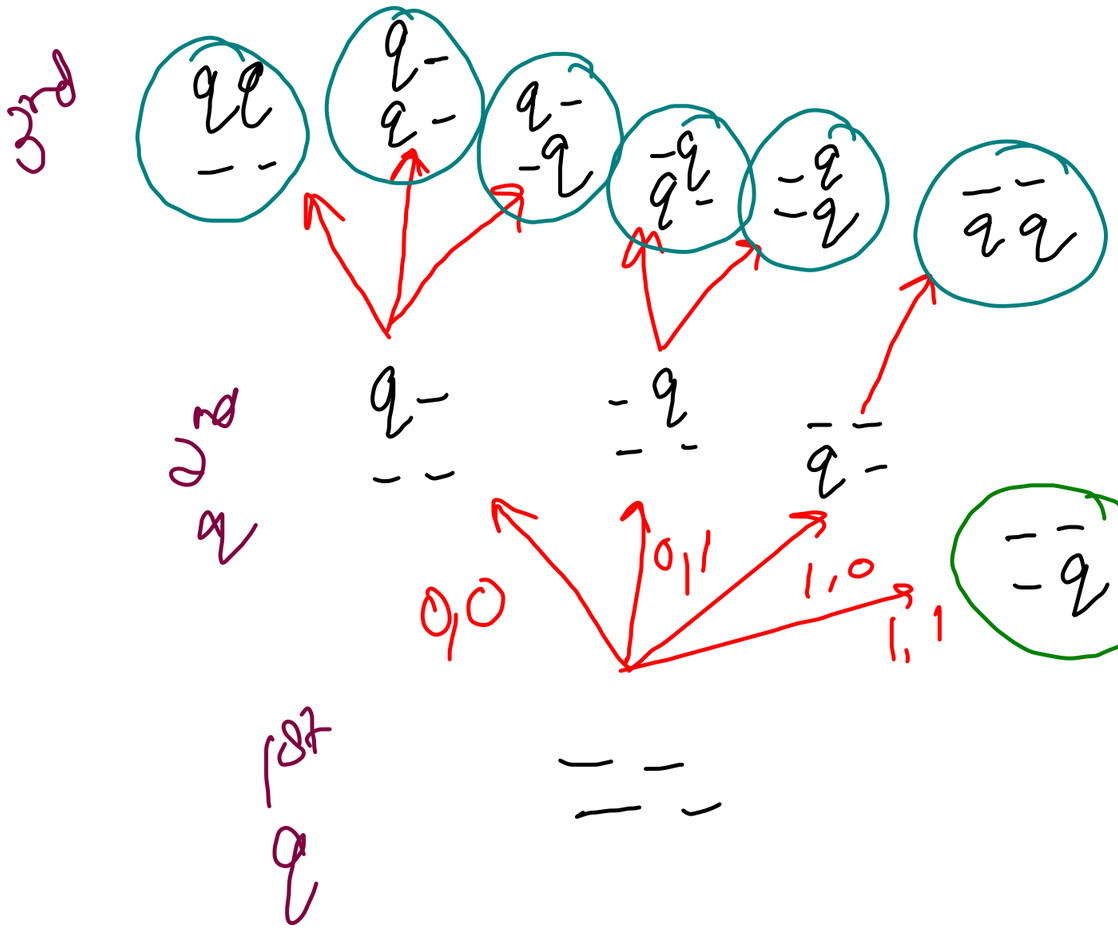
16 = 6 * 2 + 4

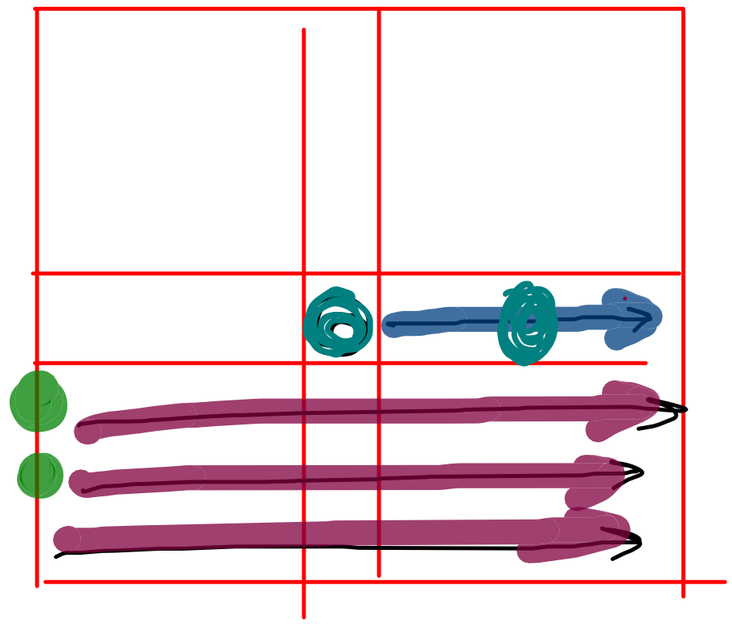
```java
// qpsf -> Queens placed so far, tq -> total Queens
public static void queensCombinations(int qpsf, int tq, int cellNo, String mat){
    if(cellNo == tq * tq){
        if(qpsf == tq){
            System.out.println(mat);
        }
        return;
    }

    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

$k = 2$

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



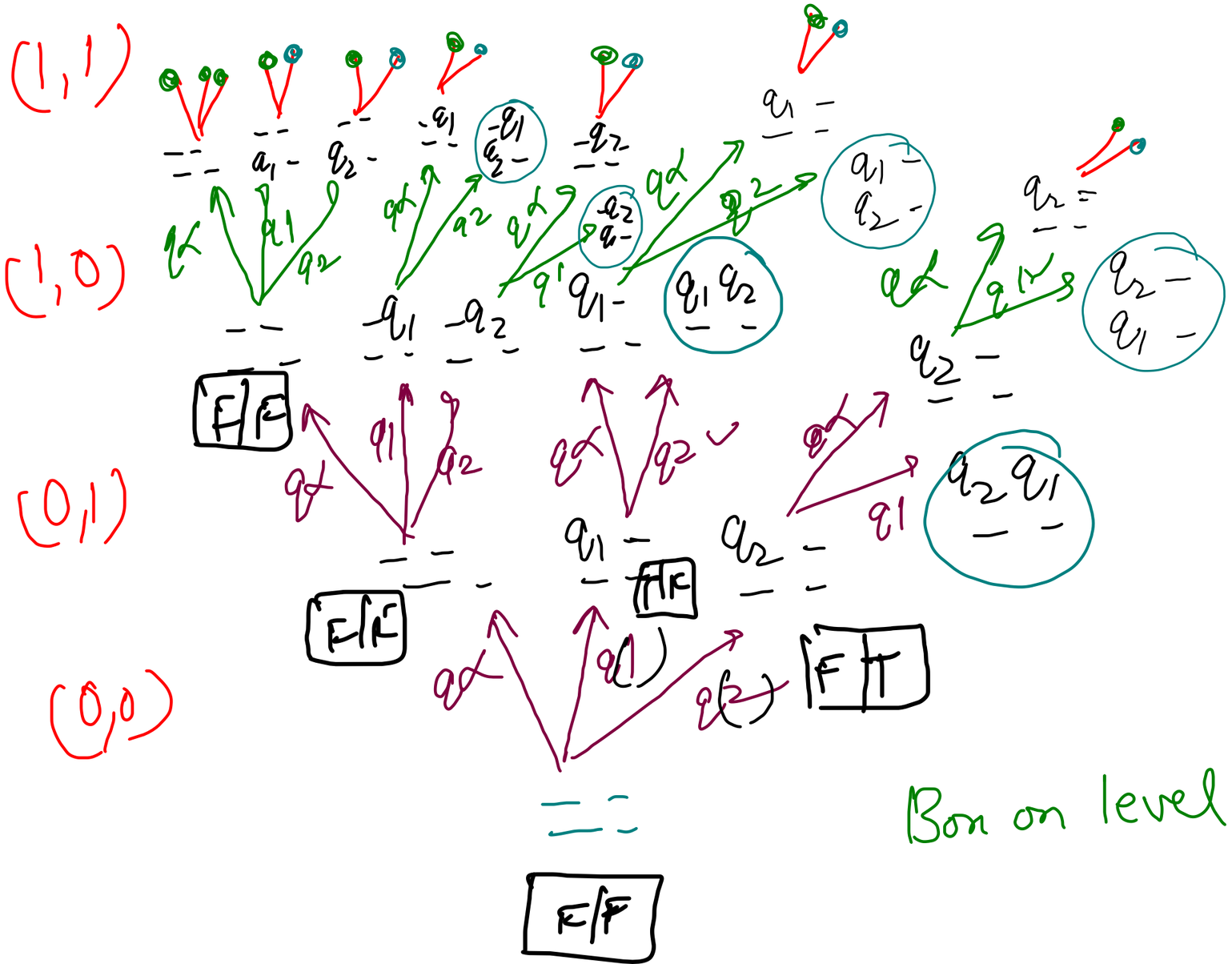


```java
public static void queensCombinations(int qpsf, int tq, Character[][] chess, int lastQueenRow, int lastQueenCol){
    if(qpsf == tq){
        for(int i=0; i<tq; i++) {⟺}
        System.out.println();
        return;
    }

    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

$\{q_1, q_2\}$ $k=2$

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



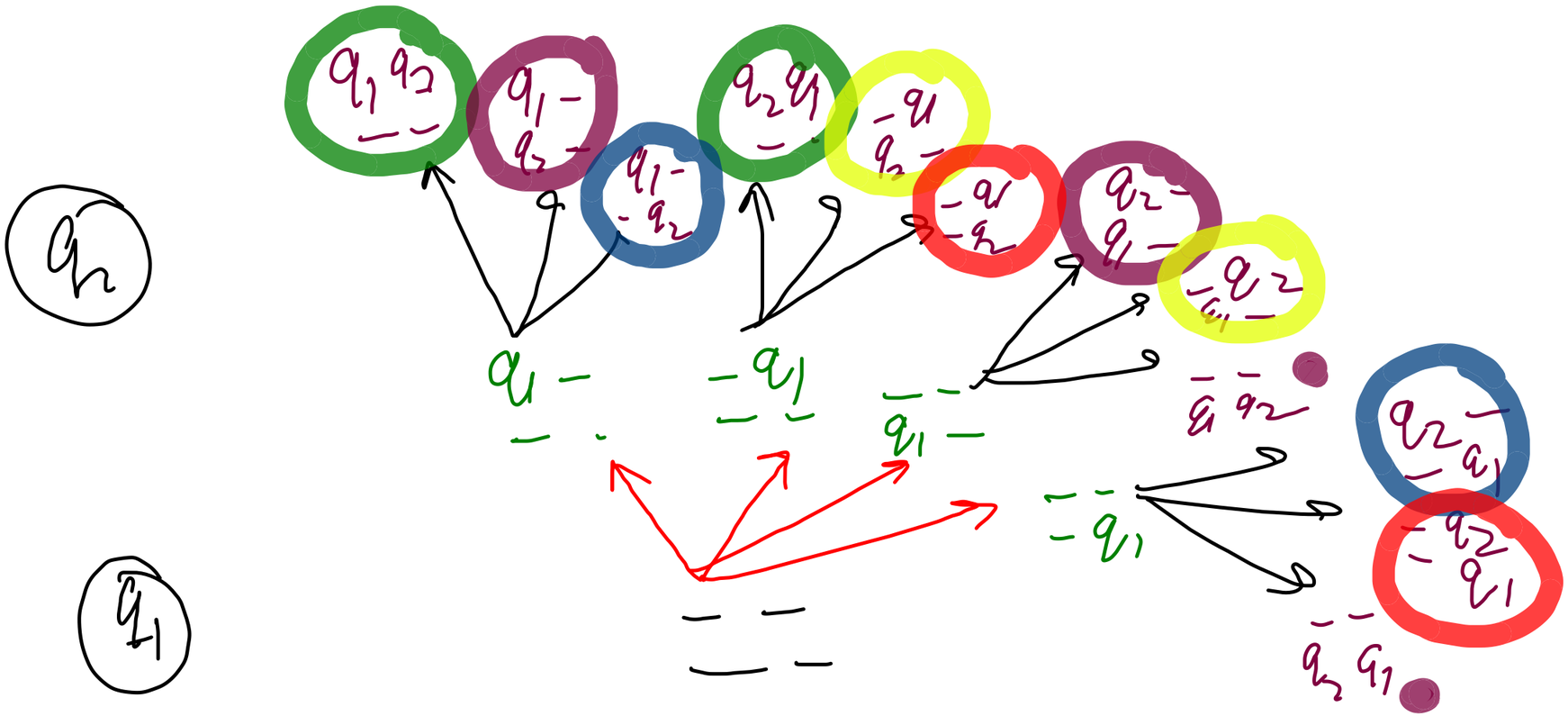
```java
if(cellNo % tq == tq - 1){
    // Apni Row ka last element, agli row ke pehle element par jana hai
    for(int i=0; i<tq; i++){
        if(vis[i] == false){
            vis[i] = true;
            queensPermutations(qpsf + 1, tq, cellNo + 1, mat + "q" + (i + 1) + "\t\n", vis); // yes
            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

$\{q_1, q_2\}$ $k = 2$

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

$^4P_2 = {}^4C_2 * 2!$



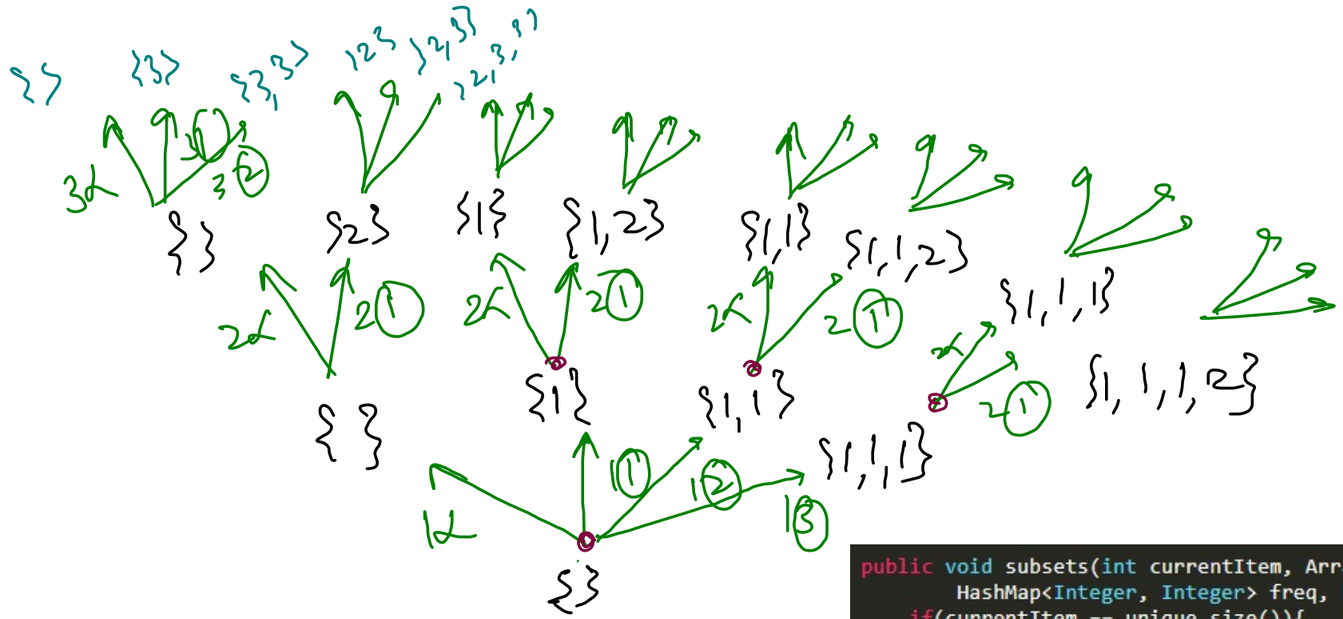
Queen class

```java
    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}

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

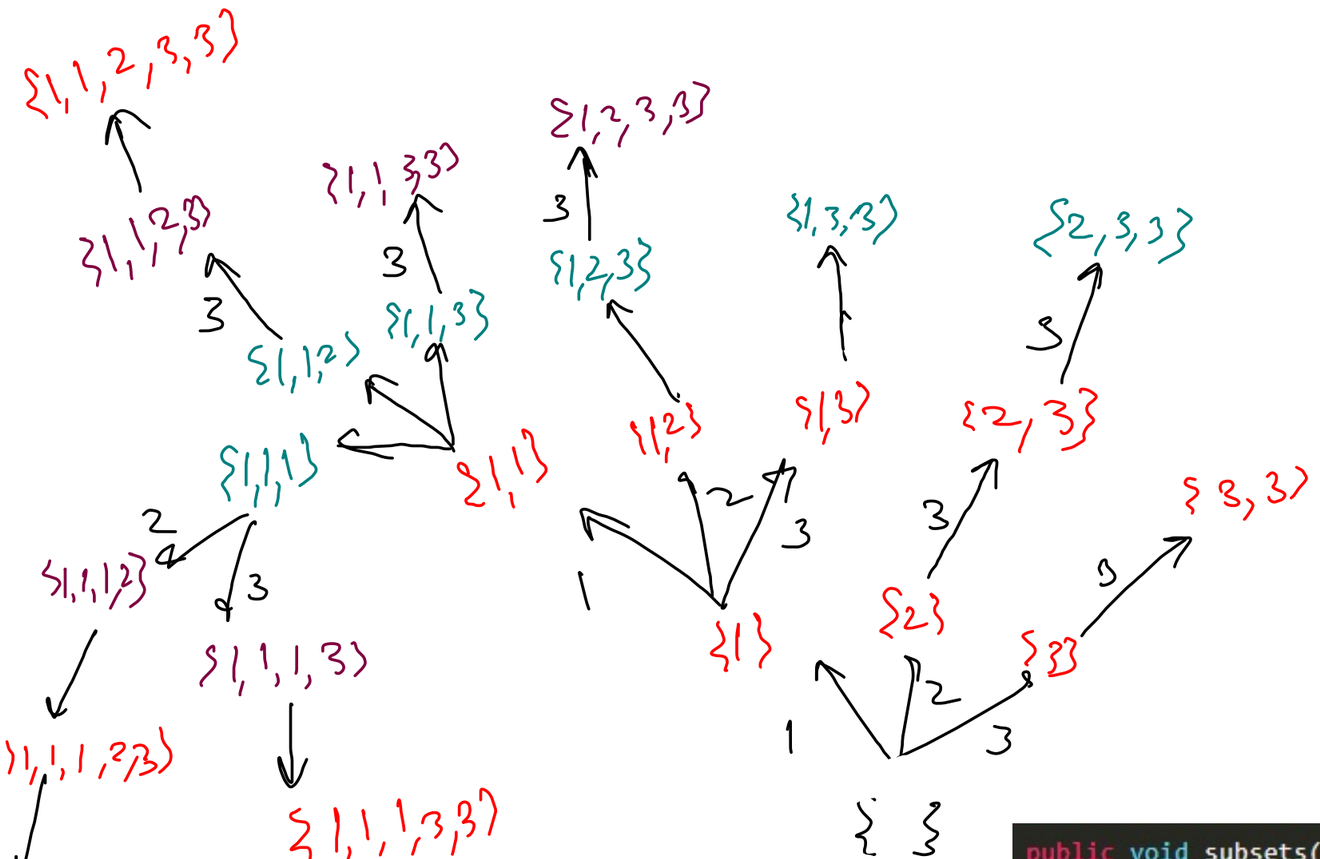
Hashmap of
↓
Array



Item on level

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);
    }
}
```



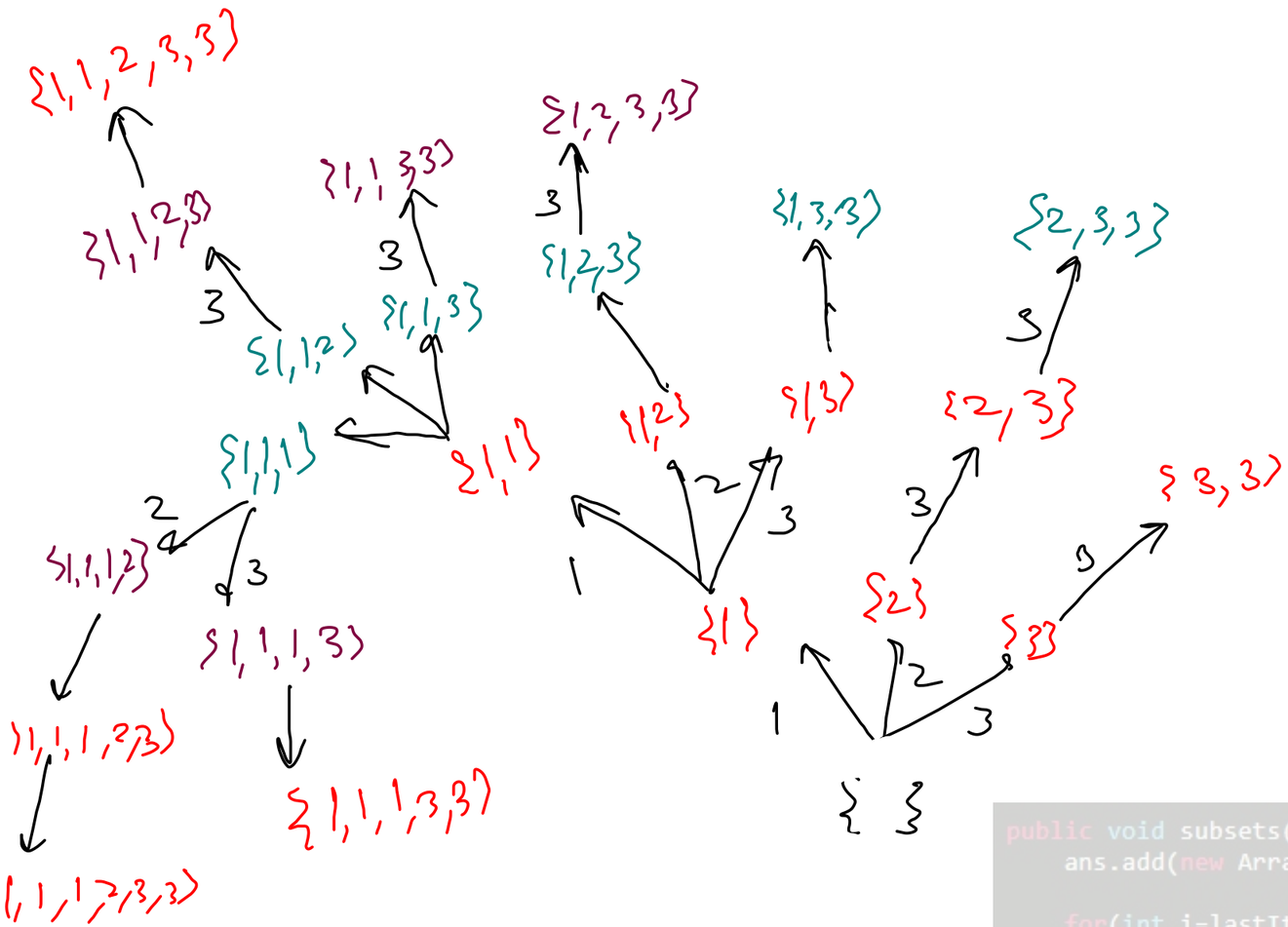
{1, 1, 1, 2, 3, 3}

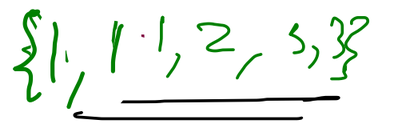
7!

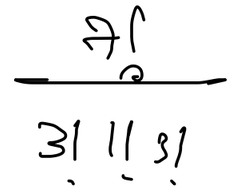
3! 1! 3!

Box on level

```
public void subsets(int lastItem, ArrayList<Integer> unique, HashMap<Integer, Integer> fre
    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);
        }
    }
}
```

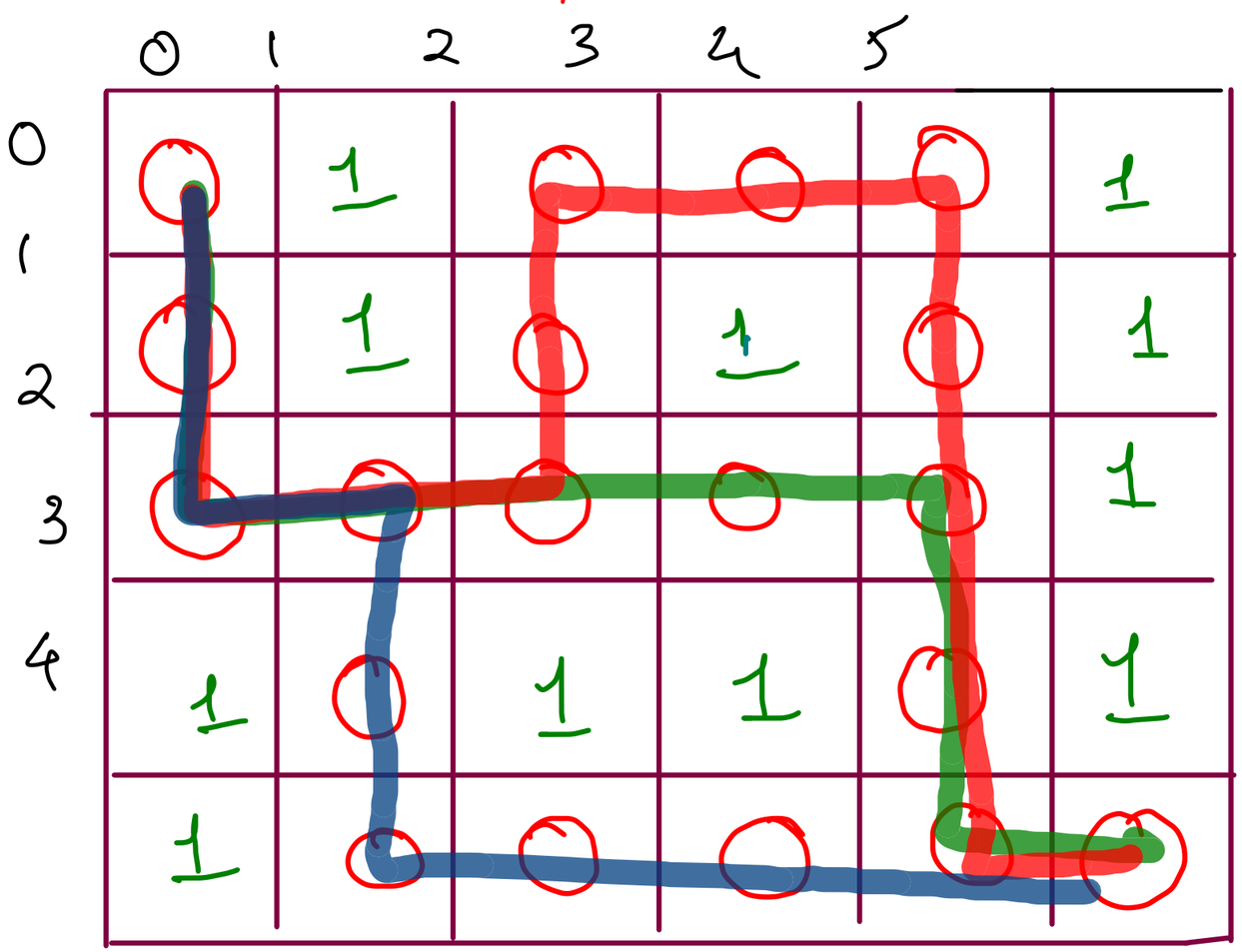






```
public void subsets(int lastItem, ArrayList<Integer> unique, HashMap<Integer, Integer> fre
    ans.add(new ArrayList<>(res));

    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);
        }
    }
}
```

Rat In Maze / Flood Fill



① Preorder {Node-Pre}

vis[r][c] = true

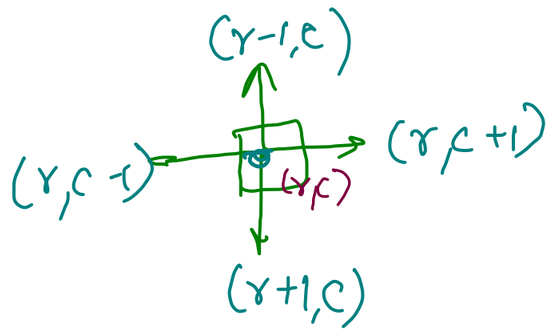
② calls & TLDR}

③ Postorder {Node-Post}

vis[r][c] = false

4 way adjacent

{TRDL}



x :→ {-1, 0, +1, 0}

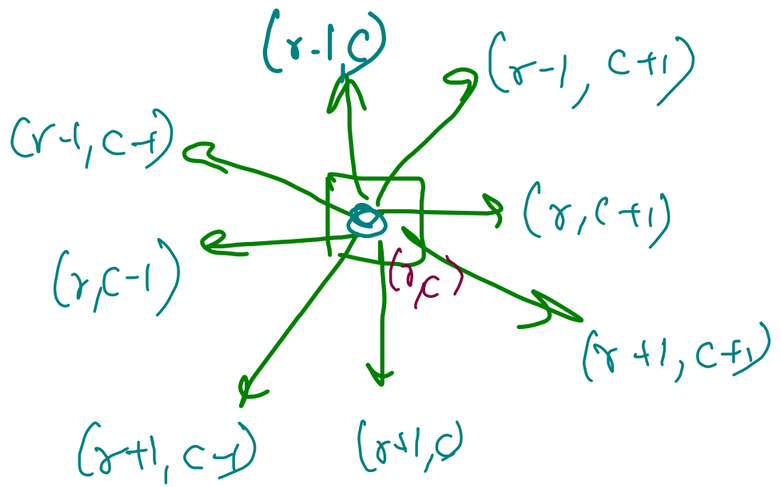
y :→ {0, +1, 0, -1}

```
for(i=0; i<4 ; i++)
  {   ff( r+x[i], c+y[i])
  }
```

8 way adjacent

{ TLDR ; 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}

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

All paths

5 8 7
5 8 6
5 8 9

6 8 5
6 8 7
6 8 9

7 8 6
7 8 5
7 8 9

9 8 5
9 8 6
9 8 7

8 5
8 6
8 7
8 9

(1) you can revisit cell ⟹ visited matrix

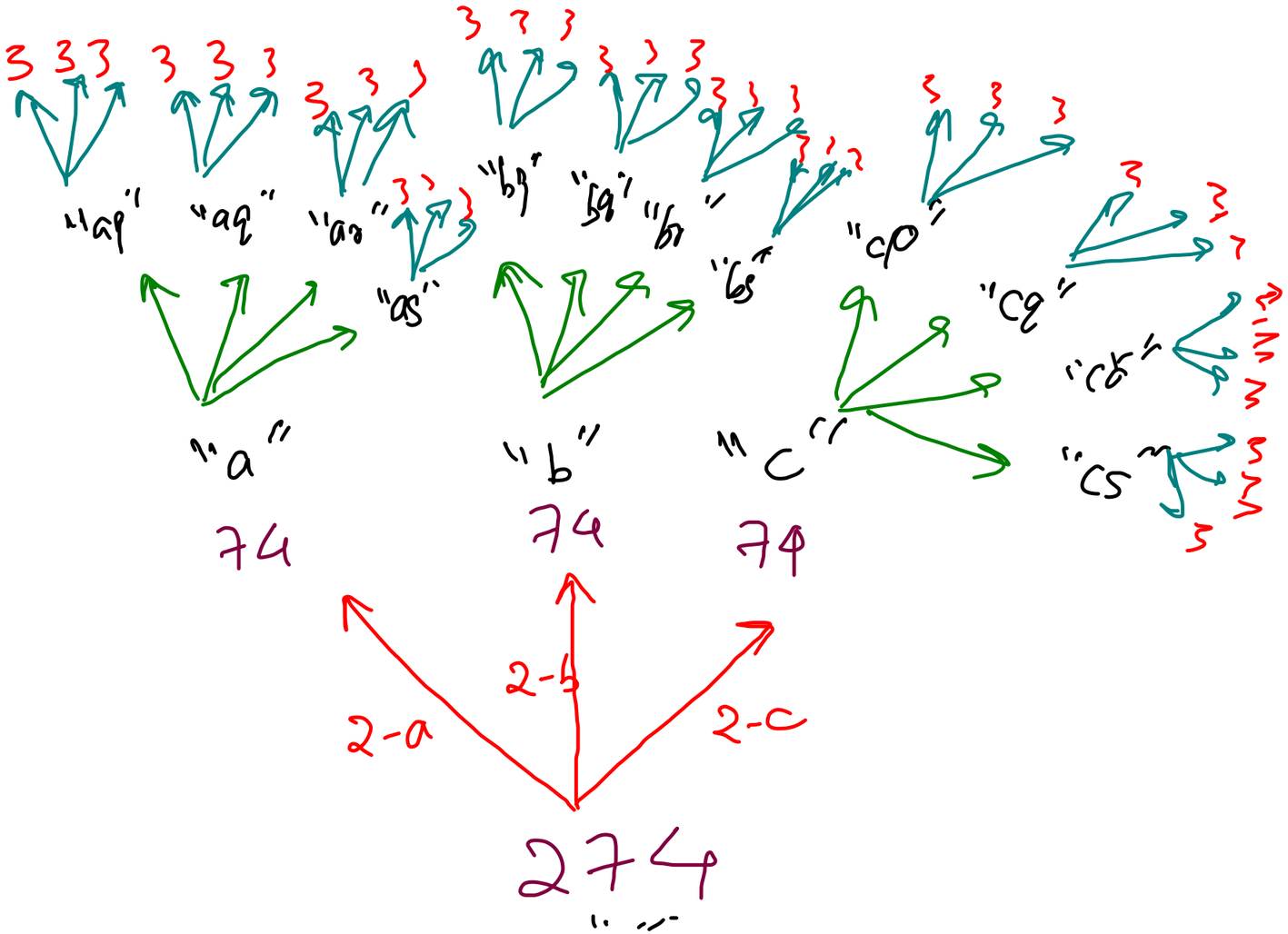
(2) Any node can be source node ⟹ All nodes can act as a source node.

(3) Treat 0 as a blockage.

```java
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]]
```

Keypad

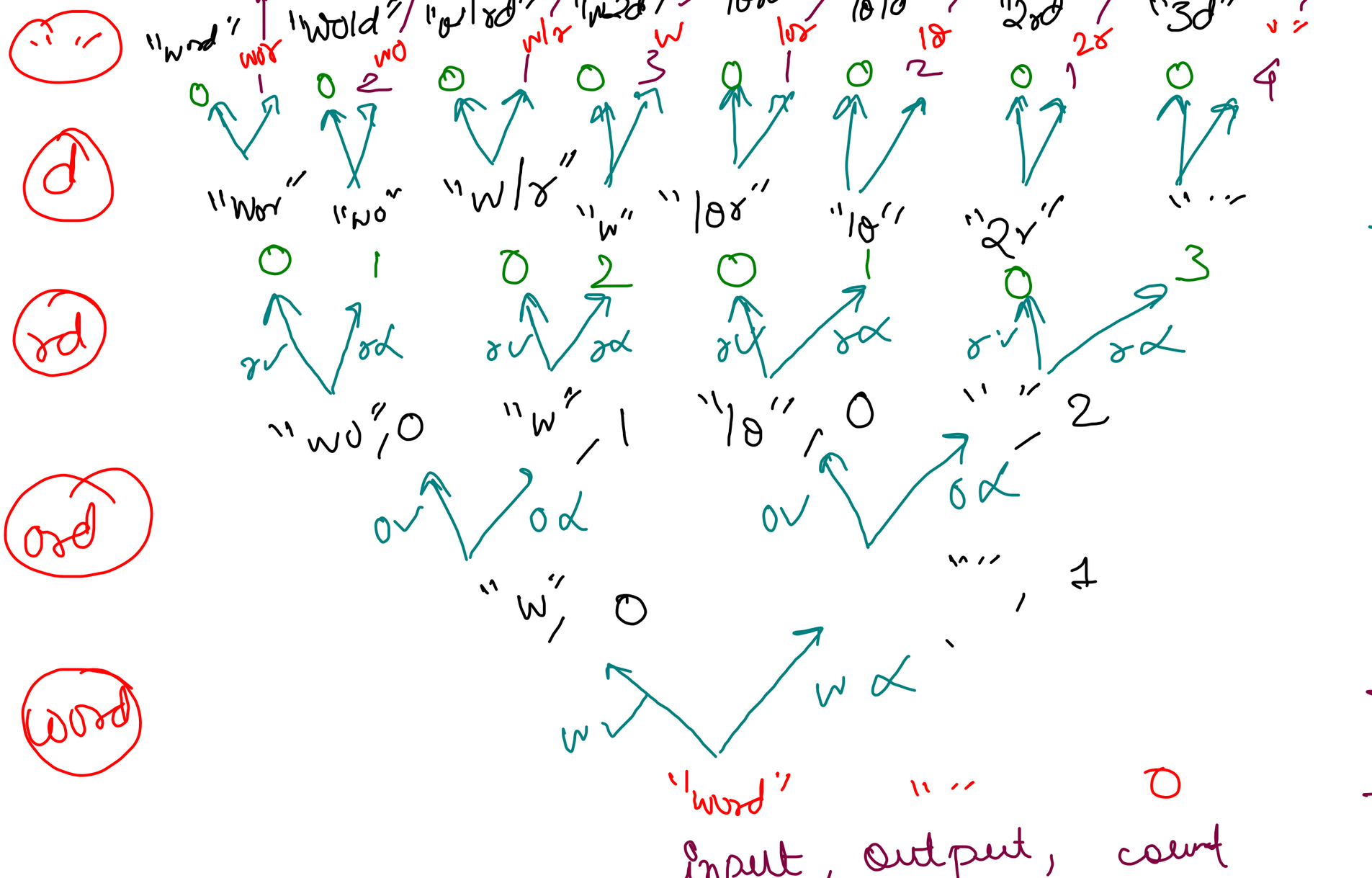


Base case

3 * 4 * 3

```java
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



① Yes

→ append count if > 0 in output

→ append characters

→ call count = 0

② No

→ pass output as it is

→ count + 1