(1) String to Integer (atoi):

Ex: s = "___-042"

→ "-42"

- Ignore the whitespace part.
- Consider the -ve sign.
- leading '0' should be neglected.
- "-42" is read as the answer.

Ex: s = "1337c0d3".

→ "1337"

Steps: (1) Character read for leading whitespace.

(2) character read for (-ve or +ve sign)

(3) Read the number present.

(2) Pow (x, n):

2^4 , 2^{-4}

2^4 $\quad$ $\frac{1}{2^{|4|}}$

(x^n)

$n = +ve \Rightarrow x^n$

$n = -ve \Rightarrow x^{-n} = (1/x)^n = \frac{1}{(x^{|n|})}$

If $(n \% 2 == 0) \rightarrow$

$x^n = (x^{n/2})^2 \Rightarrow$

If $(n \% 2 != 0) \rightarrow$

$x^n = x \cdot x^{n-1}$

Pseudo:

```
long long N = n;
if (N<0){ x = 1/x;
          N = -N; }

double result = 1;
while (N>0){
    if (N%2==1){
        result *= x; }
```

```
        else if (N%2 == 0) {
            x *= x;
            N /= 2;
        }
    }
    result result;
```

```
func. (double x, int n) {
    long long N = n;
    if (N<0) {
        x = 1/x;
        N = -N;
    }
    return power(x, N);
}
power (double x, long long n) {
    if (n == 0) return 1;
    double half = power(x, n/2);
    if (n%2 == 0) {
        return half * half;
    }
```

x^n

else { return x * half * half; }

---

(3) Count good numbers:

→ Even index places have even element at it.

→ odd index places have prime element at it.

Given n,
return total number of good digit strings of length n.

→ Ex: n = 1

Ans: 0, 2, 4, 6, 8

→ Ex: n = 4

Ans: 400.

5 4 5

$\Rightarrow$ 5 — — — — — $(n=5)$

$(0,2,4,6,8)$ $(2,3,5,7)$

$5 \times 4 \times 5 \times 4 \times 5 = 25 \times 80$

$= 20 \times 100 = 2000$

$n=1 \Rightarrow 5$

$n=2 \Rightarrow 5 \times 4$

$n=3 \Rightarrow 5 \times 4 \times 5 \rightarrow (n/2) \times 4 * (\frac{n}{2}+1) * 5$

$n=4 \Rightarrow 5 \times 4 \times 5 \times 4$

$n=5 \Rightarrow 5 \times 4 \times 5 \times 4 \times 5$

$(\frac{5}{2})$

$(\frac{n}{2})^4 * (\frac{n}{2}) 5.$

$(\frac{n}{2}) 4 * (\frac{n}{2}+1) * 5$

For $(n = odd)$ {

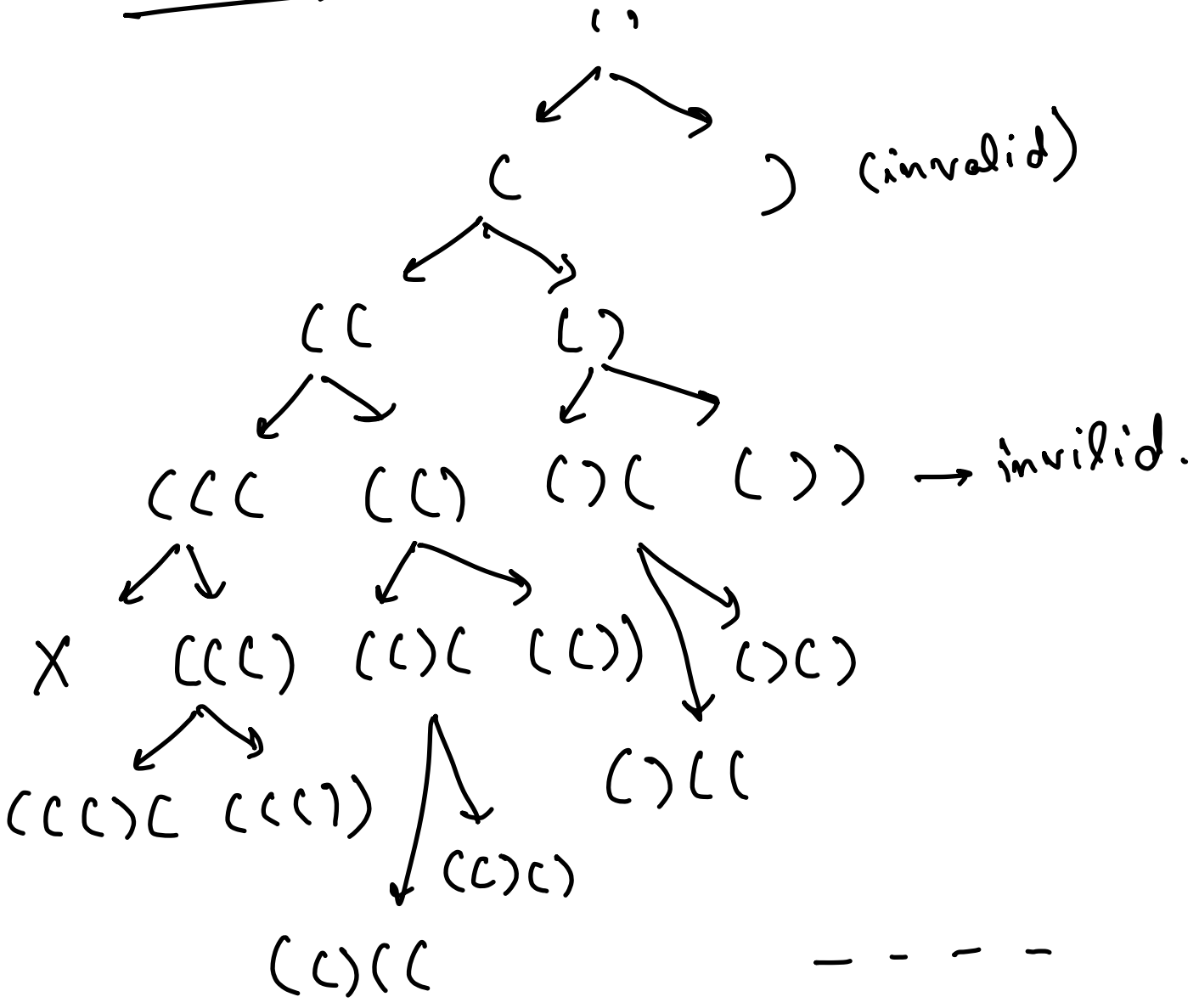
$(\frac{n}{2} * 4) * (\frac{n}{2}+1) * 5$ }

For (n = even) {

$$\left(\frac{n}{2} * 4\right) * \left(\frac{n}{2} * 5\right)$$

---

④ Generate Parentheses :



Code:

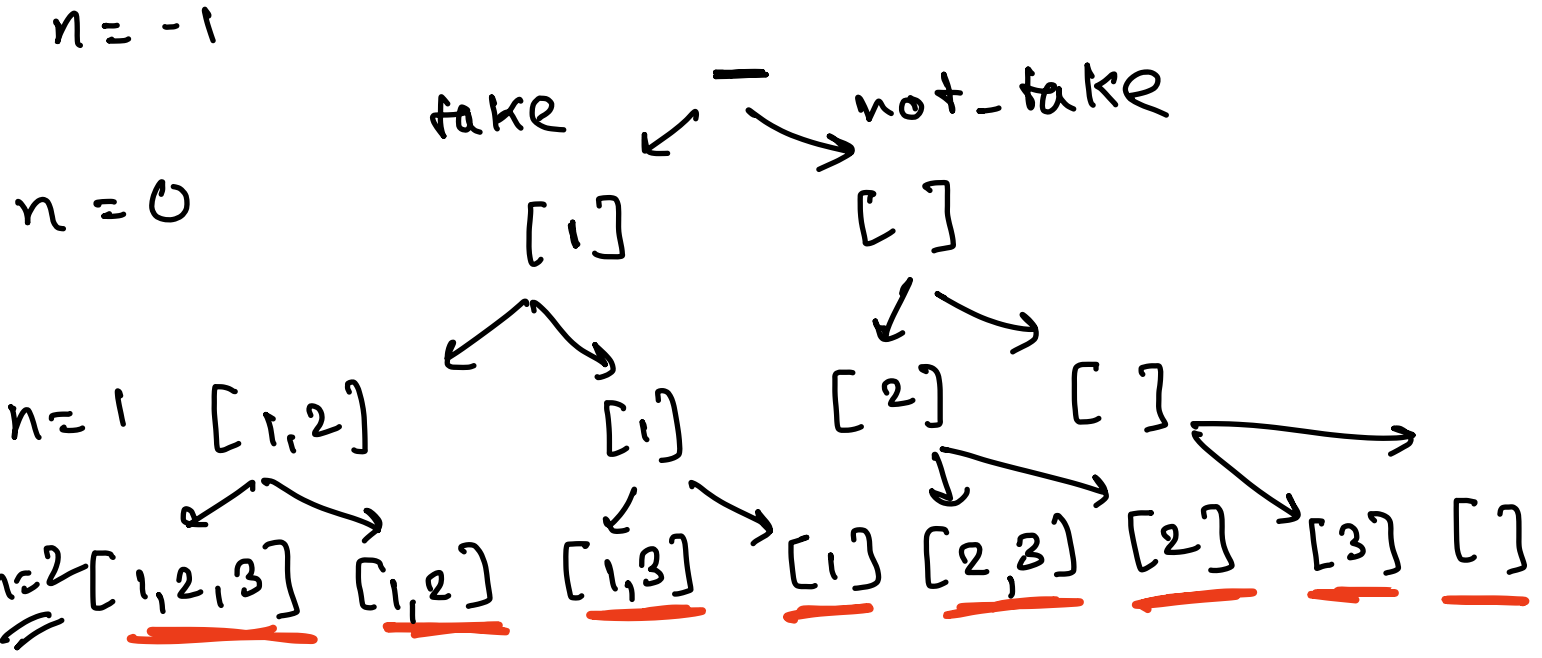
```
generateParenthesis(int open, int close,
                    vector<string>& result,
                    string current) {

    if (open == 0 && close == 0) {
        result.push_back(current);
        return result;}

    if (open > 0) {
        generateParenthesis(open-1, close,
                            result, current + '(');
                            }

    if (close > open) {
        gener.Parenthesis(open, close-1,
                          result, current - ')');
                          }.
```

---

(8) Subsets:

Ex: nums = [1, 2, 3].

⇒ Output: [], [1], [2], [3], [1,2], [1,2,3], [2,3], [1,3].



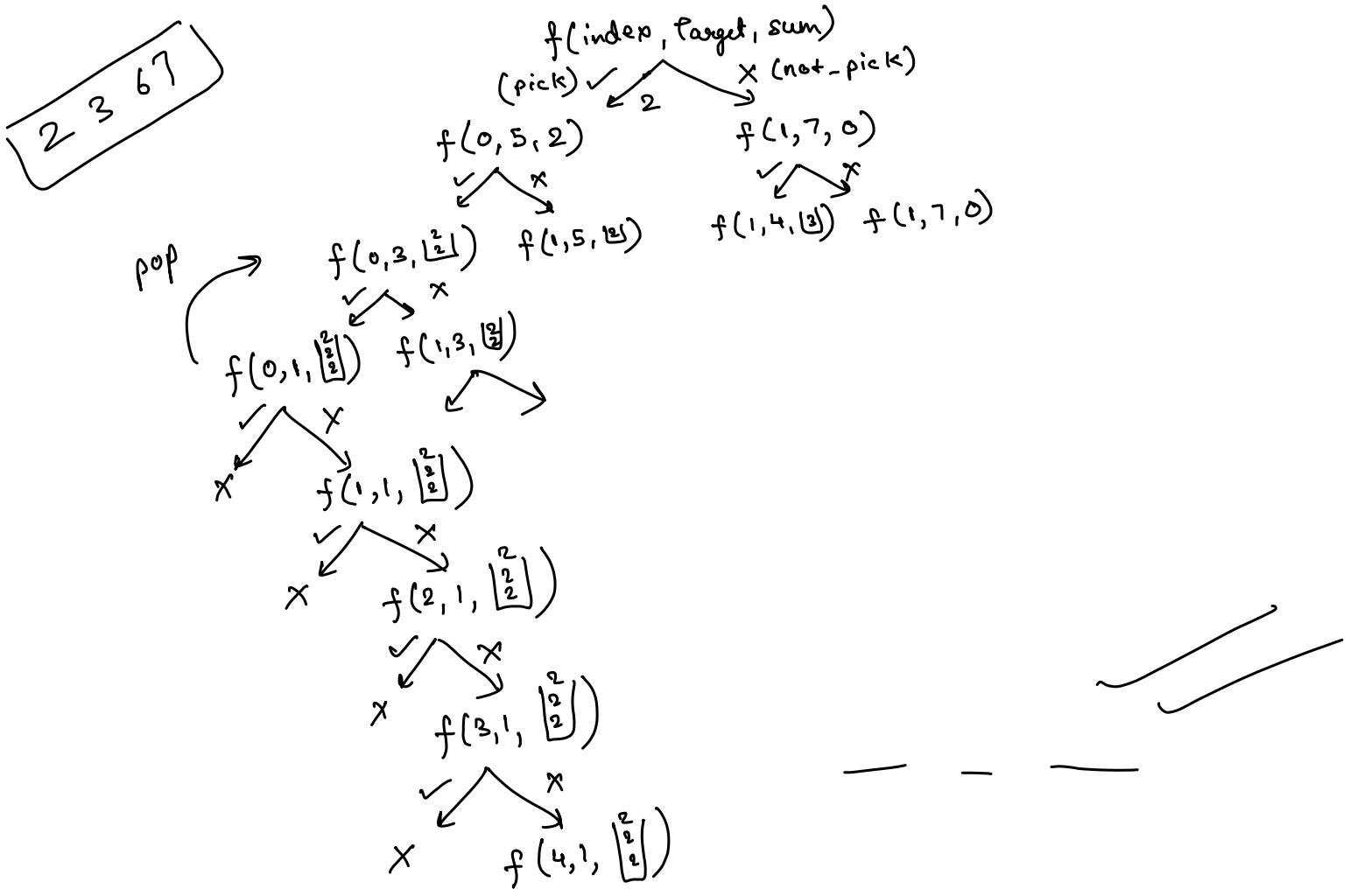

```
helper (vector<int>& nums, vector<int>
        result, vector<int> index) {

if (index == nums.size()) {
    result.push_back(current); }

vector<__> take =
    helper (nums, result, index+1))

notake =
    helper (nums, result, index+1) }
```

(6) Combination Sum:

Ex: candidates = [2, 3, 6, 7].

target = 7.

→ Output: [2+2+3], [7].



```
helper(index, arr, vector<vector<int>> ans,
       vector<int> ds) {
    if (index == arr.size()) {
```

```
if( target == 0) {
    ans.push_back (ds);
}
return; }.

if (arr[index] ≤ target) {
    ds.pushback (arr[index]);
    helper (index, target - arr[index], ans,
            arr, ds);
    ds.pop_back();
}
helper (index+1, target ---- );
                                ?-
```

---

(2) Combination Sum-2 :

Ex: candidates = [10, 1, 2, 7, 6, 1, 5]

target = 8.

```
for(int i = ind; i < nums.size(); i++) {
  if(i > ind && arr[i] == arr[i-1])
        continue;
  if(arr[i] > target) break;
  push_back(arr[i]);
  helper(i+1, target - arr[i], arr, ans,
                                ds);
  ds.pop_back();
}.
```

---

(8) subset sum.

Ex: {5, 2, 1}.

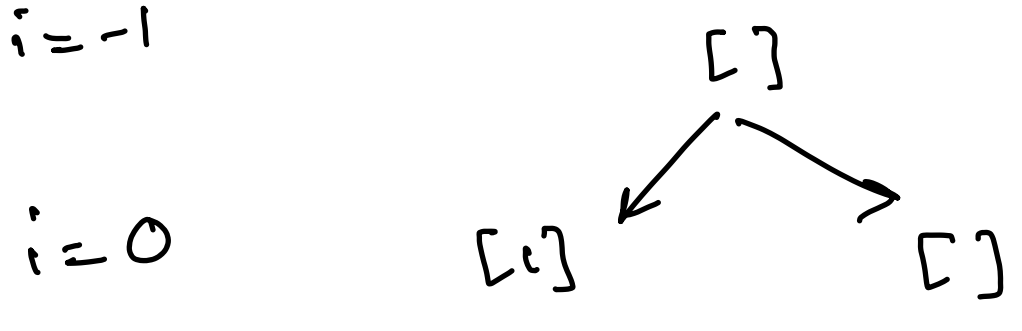
Output: 0, 1, 2, 5, 3, 8, 7, 6

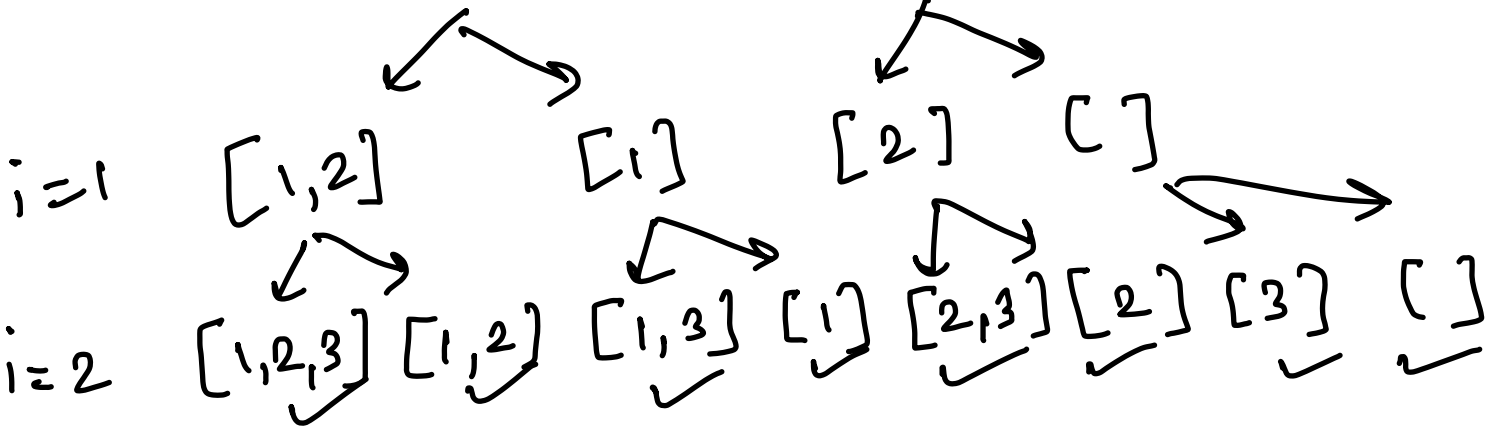
= 0 1 2 3 5 6 7 8.

Ex: {1, 2, 3}

Output: 0, 1, 2, 3, 3, 4, 5, 6.

i = -1






```
void helper (index, vector arr, vector ans,
              sum) {

    if (index == n) {
        ans.push_back (sum);
        return; }

helper (index+1, arr, ans, sum + arr[ind]);
helper (index+1, arr, ans, sum) ;.
```

---

(9) Subset - II :

Ex: nums = [1,2,2].

output = [], [1], [2], [2], [1,2], [1,2],
[2,2], [1,2,2].

```
void helper (vector<int> & nums, int
        index, vector < vector < int >> & ans,
vector < int > & ds ) {
    ans. push_back (ds) ;
    for (int i = ind ; i < nums.size() ; i++){
     if [i != index && nums [i] ==
                        nums [i-1]) continue;

      ds. push_back ( nums [i]) ;
      helper (i+1, nums, ds, ans) ;
       ds. pop_back() ;
    }
}
```

---

(10) combination Sum III :

Ex: k = 3 , n = 7 .

output : [[1,2,4]].

Ex: k = 3, n = 9

Output : [[1,2,6], [1,3,5], [2,3,4]].

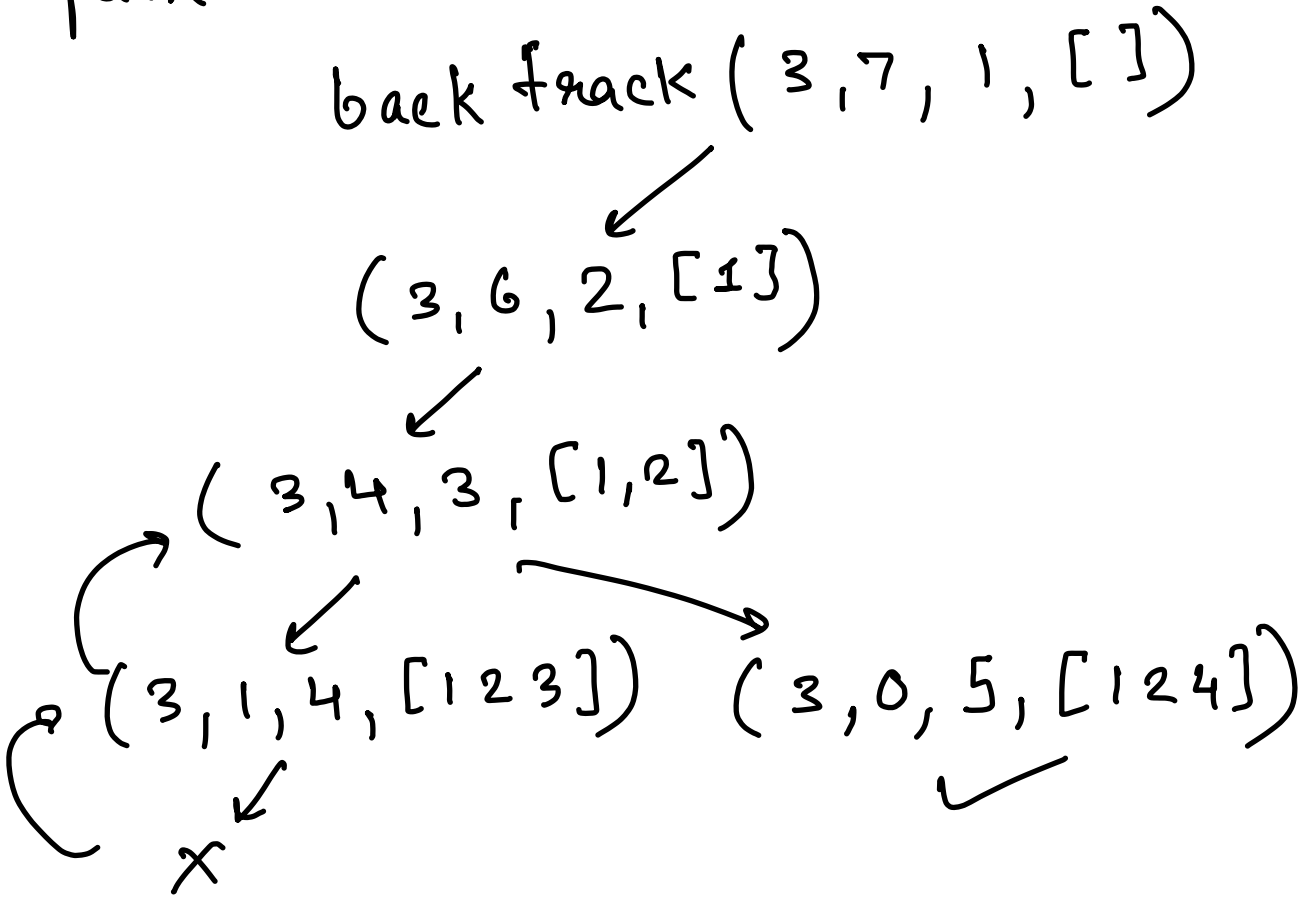
Base Cases: (1) If the combination has total K numbers.

(2) The sum equals n.

K = 3 and n = 7

start = 1

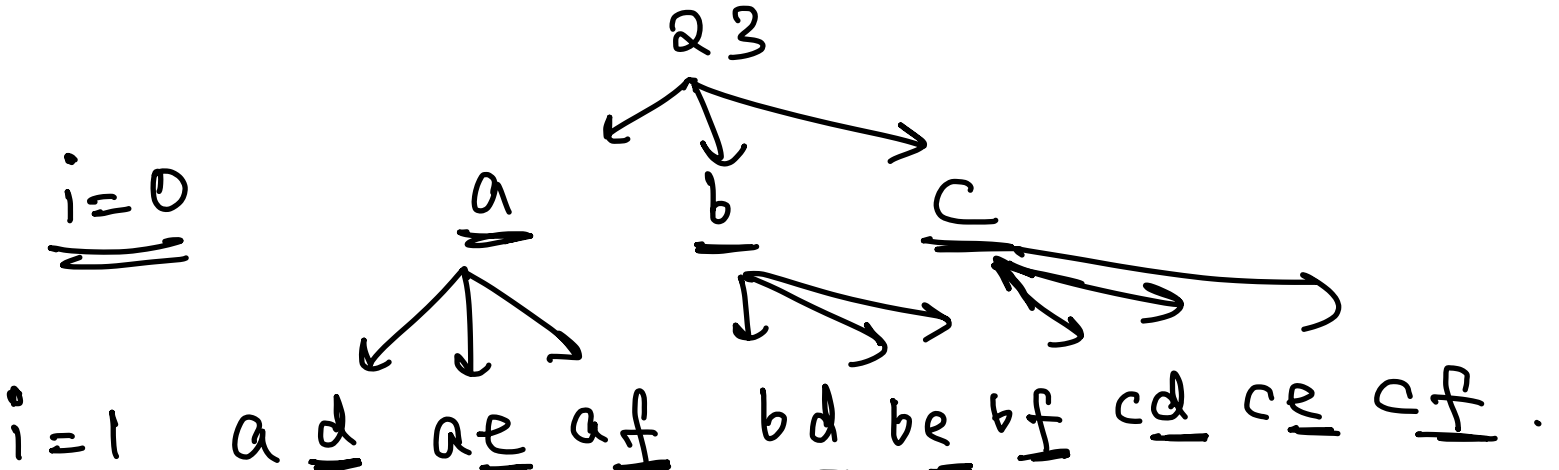
path = [].



(11) Letter combinations of a Phone Number:

Ex: digits = "23"

Output: ["ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"].

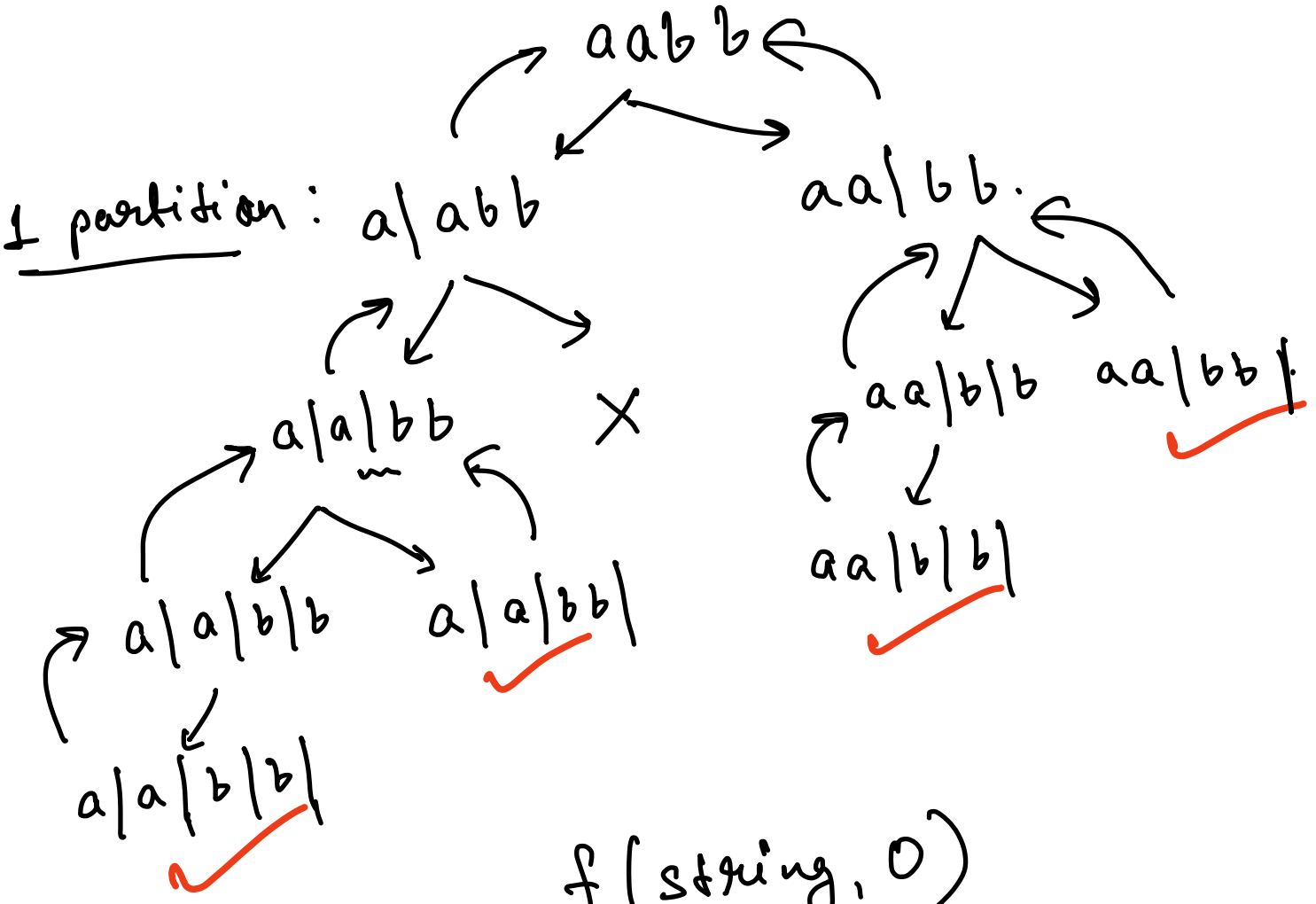


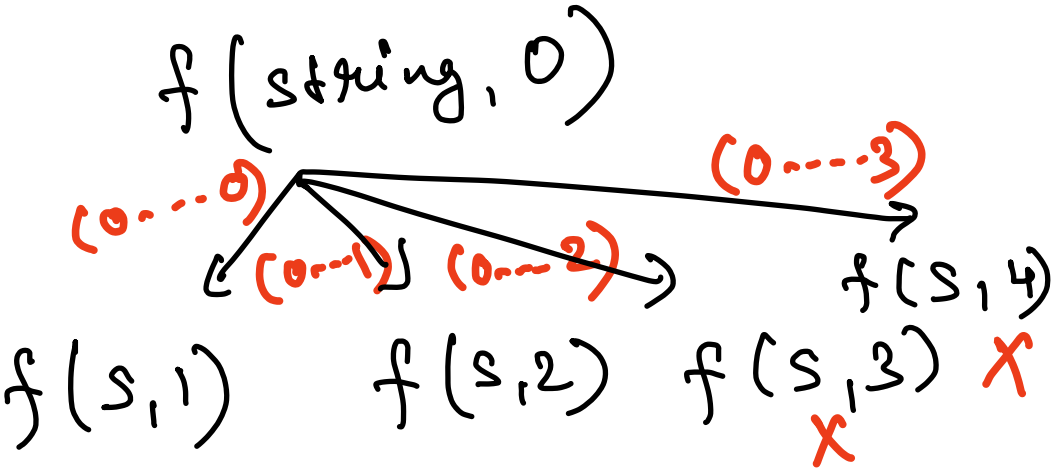

```
vector<string> result;
unordered_map<char, string> mp = {
    {'2', "abc"},
    {'3', "def"},
    :
    :
    {'9', "wxyz"}.
```

(12) Palindrome Partitioning : (Imp question)

Ex: s = "aab"

Output : [["a", "a", "b"], ["aa", "b"]]



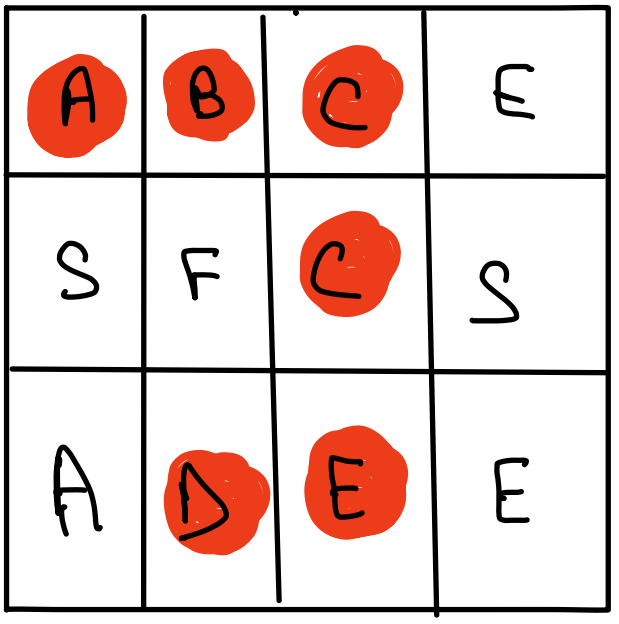


```
if(index == s.size()) {
    result.push_back(path);
        return; }.

for(int i= index; i < s.size(); i++){
    if(isPalindrome(---)) {
        path.push_back(s.substring(
                                index, i+1));
    function(i+1, s, path, result);
    path.pop_back();.
        } } }
```

```
boolean isPalindrome(s, start, end){
    while(start <= end){
        if(s[start++] != s[end--])
            return false; }
        return true; } }
```

(13) Word Search:

Ex:



⇒ iterate in the box:

base condition: (1) $i \geq 0$ && $j \leq n-1$.

⇒ if(i<0 || i≥n || j<0 || j≥m
|| board[i][j] != word[w])
return false;

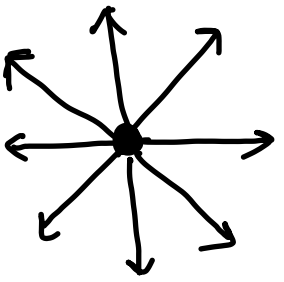
Next:

```
if(dfs(i+1,j,board,w+1) ||
   dfs(i-1,j,board,w+1) ||
   dfs(i,j+1,board,w+1) ||
   dfs(i,j-1,board,w+1))
```

return true;

---

(14) N-Queen: Queen movement:

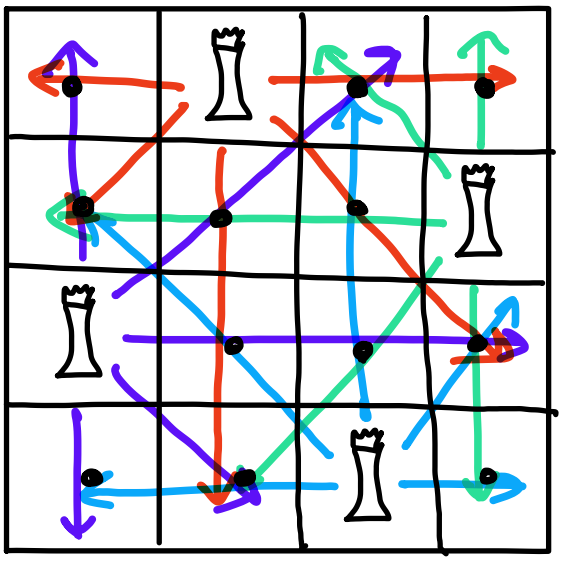


→ can move in all directions.

→ up-down, left-right, diagonal.

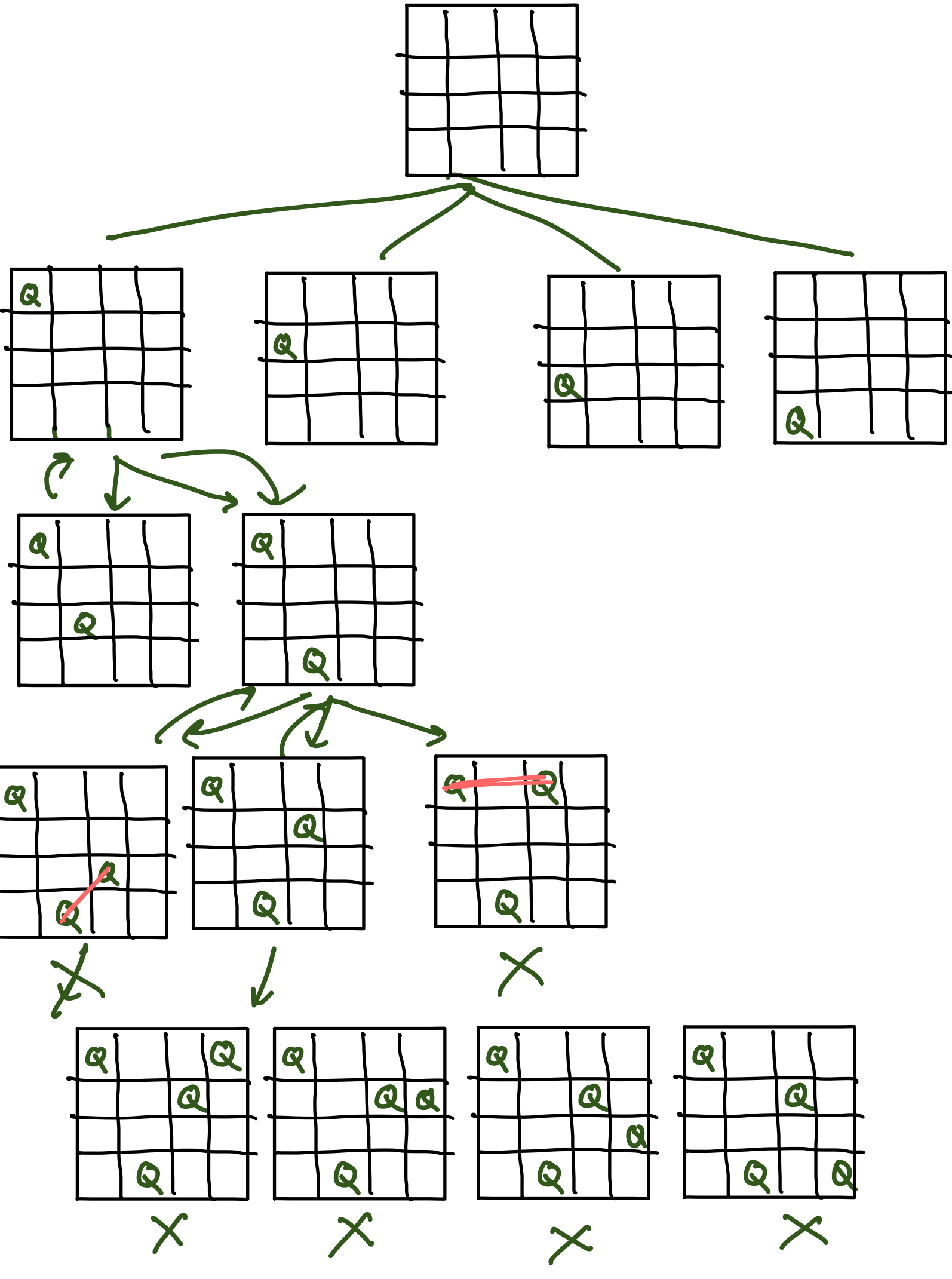
→ So for optimal solution:

The Queens can't watch each other i.e.



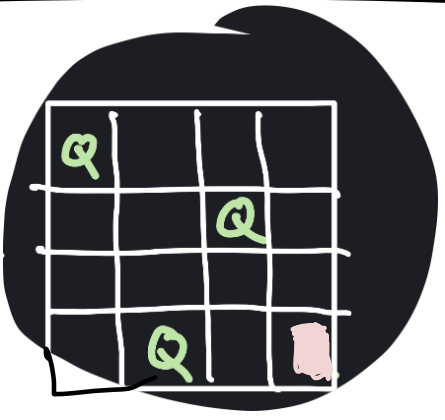
wrong approach:

```
for( int i=0; i<n; i++) {
    for (int j=0; j<m; j++) {
        if ( board [i][col ] && board [row][j]
            && board [i][j] == Q) {
        return false; }
```

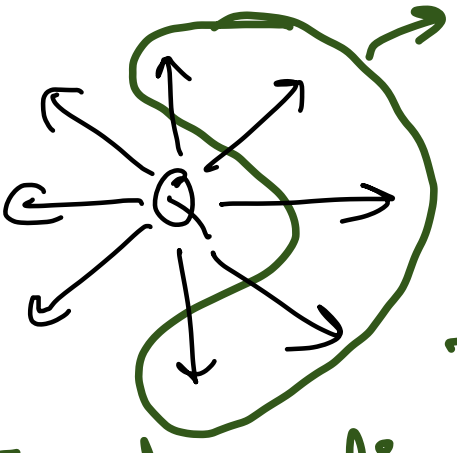


```
func. (column) {
    for ( i=0 → n-1) {{row}
        if ( fill → possible) {
            new col = Q;}
        f (column + 1) {
            column. empty i }
```

Pseudo Code:



If we are placing a queen on the chess board, then we need to check its preceeding elements.

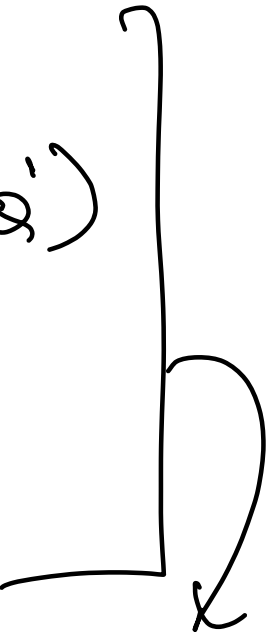


these elements are not filled yet.

So, just check left elements.

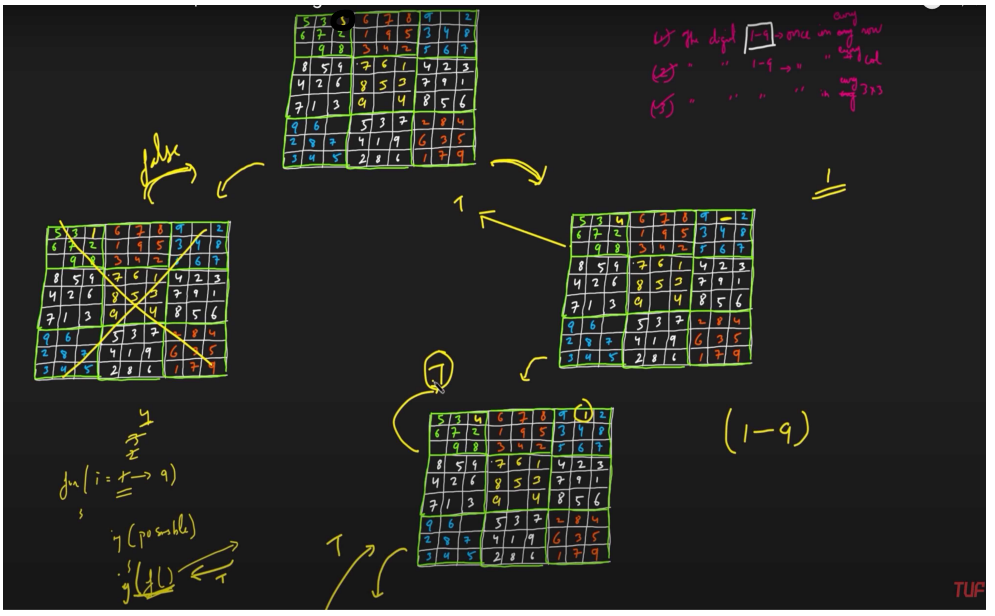
For left upper diagonal:

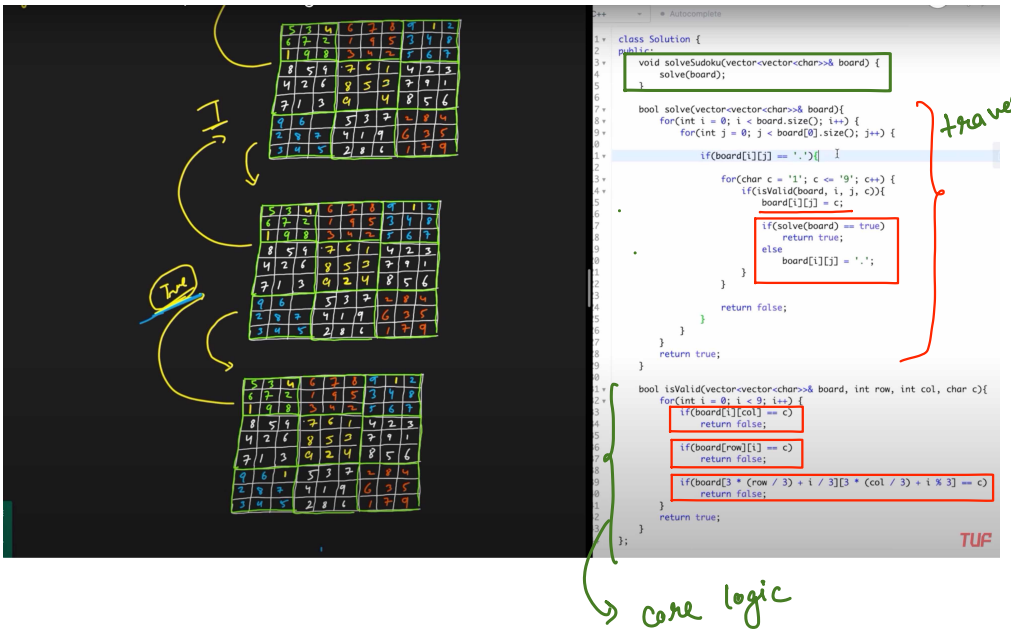
```
int duprow = row;
int dupcol = col;
while( row >= 0 && col >= 0) {
    if(board[row][col] == 'Q')
        return false;

    row--;
    col--; }

col = dupcol;
row = duprow;
while (col >=0) {
    if (board[row][col] == 'Q')
        return false;
    col--;}
```



left (←)

15 Sudoku Solver:





16 Expression Add Operators:

Ex: nums = "123", target = 6.

Output: ["1*2*3", "1+2+3"].

Ex: nums = "232", target = 8.

Output : ["2*3+2", "2+3*2"]