Recursion - When a in turns call another function a process of chaining occurs. Recursion is a special case of this proces where a function call itself.

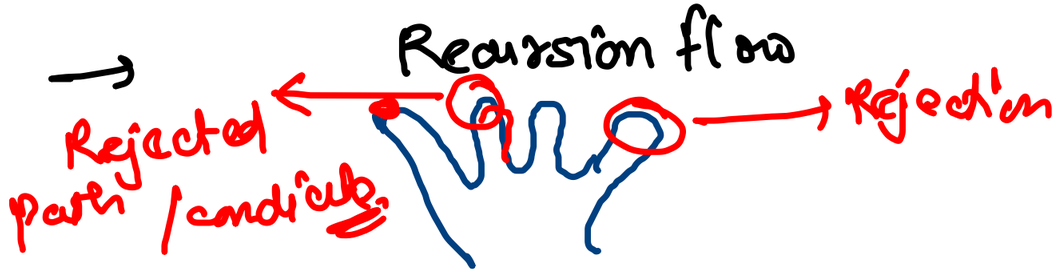
Backtracking → Searching Every possible possibility of combination In order to solve a computational problem & abandonant a path/ condidate (as soon as possible OR it can determines) that the condidate possibly be completely.

Mapping of Recursion with Backtracking →

Agenda →

1. ~~Introduction~~
2. ~~Mapping in Recursion - Backtrack~~
3. Flood Fill → Rat In a maze
4. Target Sum subsets.
5. Box Queen.
7. N-Queens.
8. knights Tour.
9. CRYPTO ARITHMATIC

Floated Question → [print Encoding. / Get permutation.



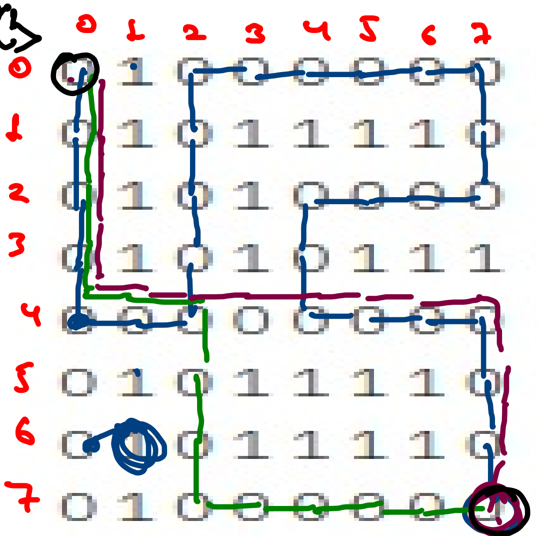
Flood Fill →

src Row = 0
src col = 0

dst row = n-1
dst col = m-1

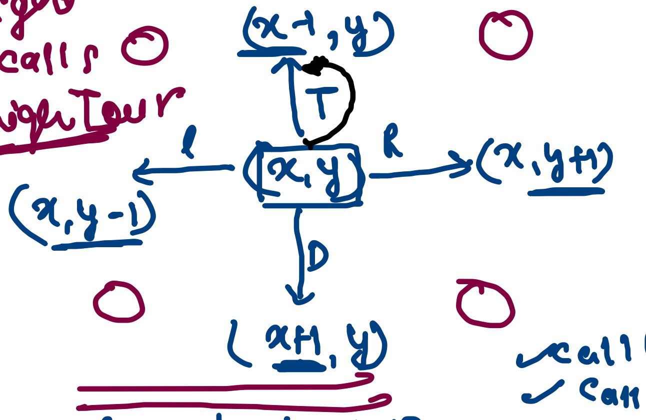
n,m → matrix

print all possible path to reach at dst.



Eight calls ⓪

knight Tour



name = ['T', 'L', 'D', 'R']

global { xdir = [-1, 0, 1, 0]
ydir = [0, -1, 0, 1].

call()1
call()2
call()3
call()4

Modification → ①
→ print any single possible path to reach the destination from source.

dddddrrttttrrrrrddllldddrrrddd
ddddrrdddrrrrr
ddddrrrrrrrrddd

→ jumps allowed → top, left, down, right.

for(int dir = 0; dir < 4; dir++){
next { int x = sr + xdir[dir]; = x-1
int c = sc + ydir [dir]; = y
call ( ) → next level
}

Primitive Data types

① Byte → 0
② Short → 0
③ Int → 0
④ char → '\0'
⑤ Float → 0.0
⑥ Double → 0
⑦ Long → 0
⑧ Boolean → false

Stack

= new datatype[ ] auto initialise → null

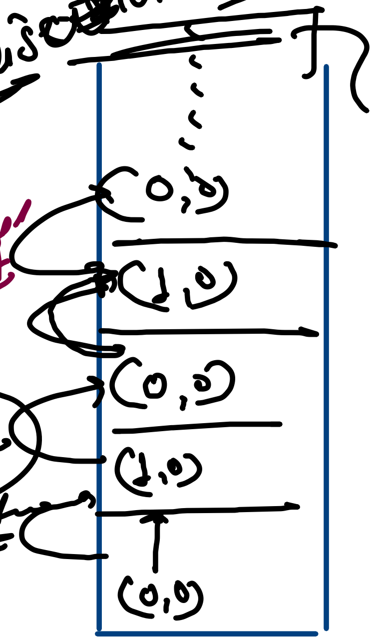
Except Primitive data type

non-primitive

Declaration: Integer, Character

Required new = memory

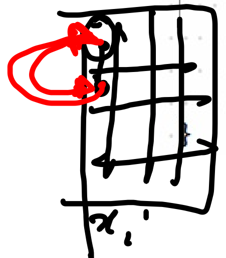
Heap



Stack overflow

marks visited

path → single time.

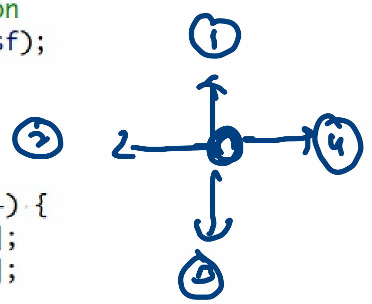


```
public static void floodfill1(int[][] maze, int row, int col, String psf){
    System.out.println(row + " " + col);

    // stupid base case
    if(row == maze.length - 1 && col == maze[0].length - 1) {
        // this is destination
        System.out.println(psf);
        return;
    }

    // make smart calls
    for(int d = 0; d < 4; d++) {
        int x = row + xdir[d];
        int y = col + ydir[d];

        if(x >= 0 && x < maze.length && y >= 0 && y < maze[0].length && maze[x][y] != 1)
            floodfill1(maze, x, y, psf + dname[d]);
    }
}
```



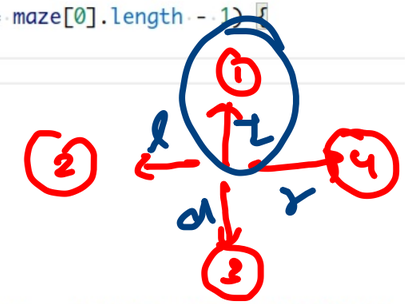
x = row + 1 =
y = col = 0

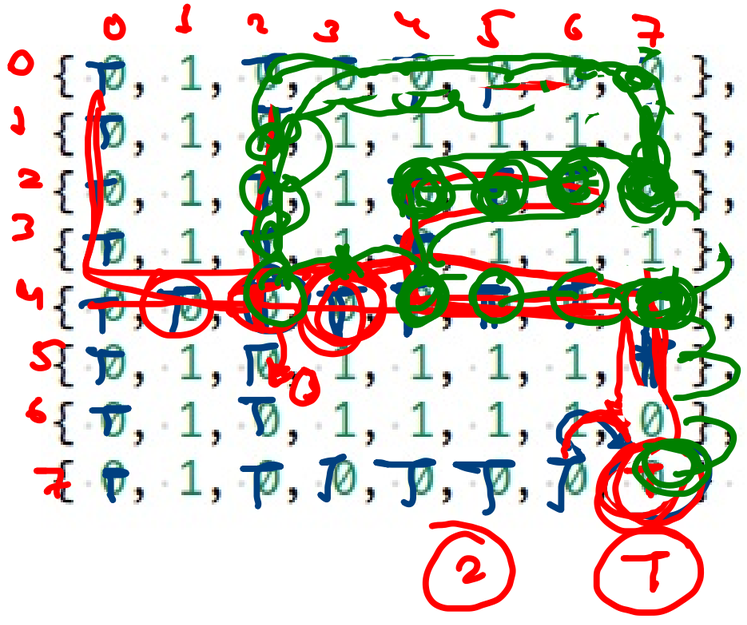
x = row - 1
y = col

```java
public static int[] xdir = {-1, 0, 1, 0};
public static int[] ydir = {0, -1, 0, 1};
public static char[] dname = {'t', 'l', 'd', 'r'};

public static void floodfill1(int[][] maze, int row, int col, String psf
    // stupid base case
    if(row == maze.length - 1 && col == maze[0].length - 1) {
        // this is destination
        System.out.println(psf);
        return;
    }
    visited[row][col] = true;
    // make smart calls
    for(int d = 0; d < 4; d++) {
        int x = row + xdir[d];
        int y = col + ydir[d];

        if(x >= 0 && x < maze.length && y >= 0 && y < maze[0].length &&
                maze[x][y] != 1 && visited[x][y] == false) {
            floodfill1(maze, x, y, psf + dname[d], visited);
        }
    }
}
```





Target sum subset →

arr = 10 20 30 40 50

target = 60

0 1 2

[10, 20, 30]

(10, 20) (30)

subset → [10, 20, 30] target (30)

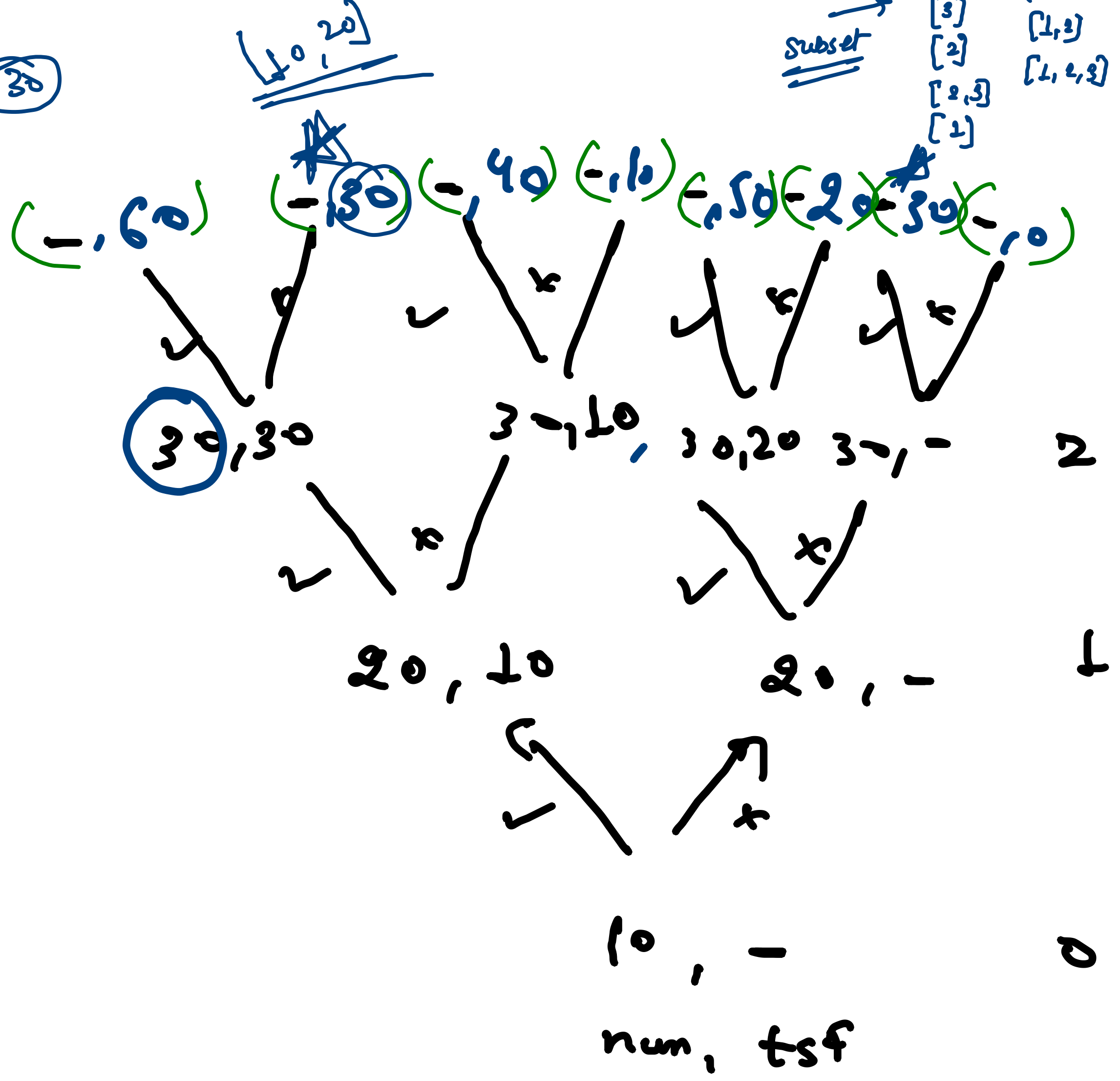
[20, 40]

[10, 50]

level → Index

option → Include that Index

→ Exclude that Index



subset / subsequence | subarray / substring.

① string → subsequence
array → subset

② no. of subset / subseq → ?

"abc" → no = 2^n

$2 \times 2 \times 2 \cdots \to n$ lth

– – –
– – c
– b –
– bc
a – –
a – c
a b –
a bc

[1, 2, 3] → [ ] [3] [2] [2, 3] [1] [1, 3] [1, 2] [1, 2, 3]

subset

① string → substring.
array → subarray.

② no. of subarray / substring →

"abc" → [1] a, [1, 2] ab, [1, 2, 3] abc] n

[1, 2, 3]

[2] b, [2, 3] bc] n-1

[3] c] n-2

1+2+3+ .... +n

no → $\frac{n(n+1)}{2}$

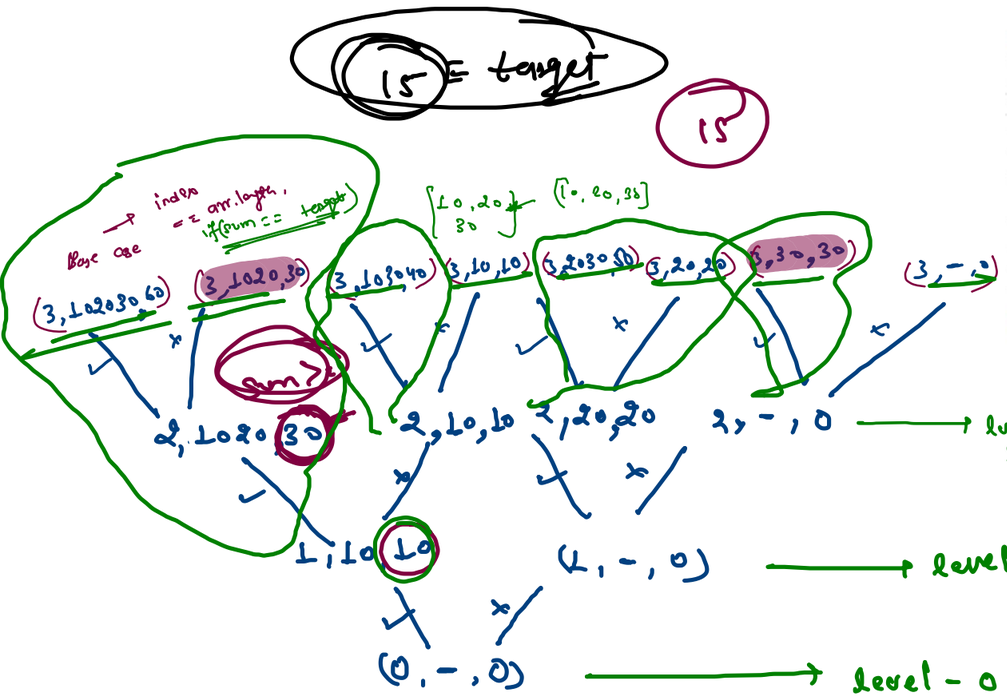
3

$= \frac{3 \times 4}{2} = 6$

[10, 20]

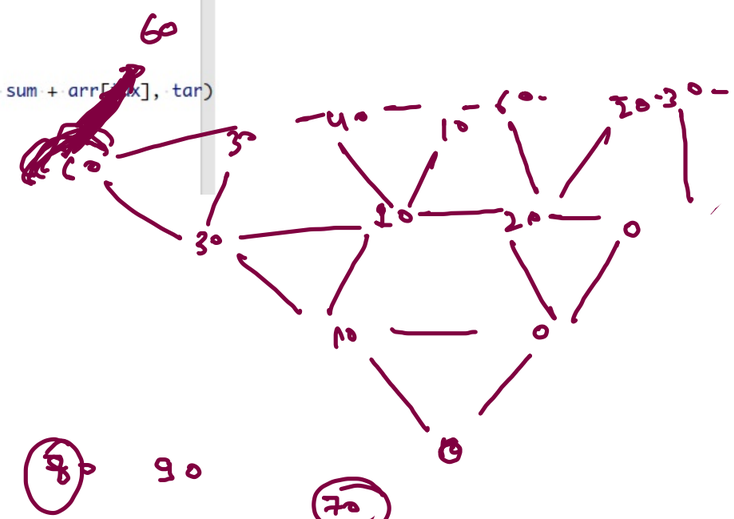
[30]

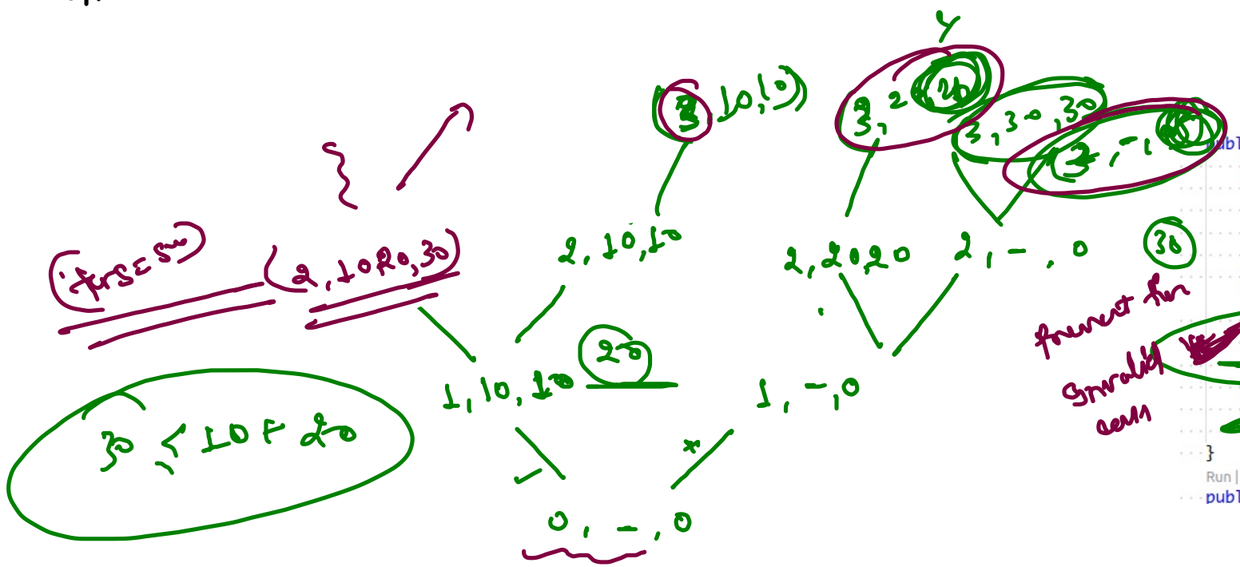
print subseq.




```
public static void printTargetSumSubsets(int[] arr, int idx, String set, int sum, i
    if(idx == arr.length) {
        if(tar == sum) {
            System.out.println(set + ".");
        }
        return;
    }

    // yes call
    printTargetSumSubsets(arr, idx + 1, set + arr[idx] + ", ", sum + arr[idx], tar)
    // no call
    printTargetSumSubsets(arr, idx + 1, set, sum, tar);
}
Run | Debug
```






```
public static void printTargetSumSubsets(int[] arr, int idx, String set, int sum, i
    if(idx == arr.length) {
        if(tar == sum) {
            System.out.println(set + ".");
        }
        return;
    }

    // yes call
    if(sum + arr[idx] <= tar)
        printTargetSumSubsets(arr, idx + 1, set + arr[idx] + ", ", sum + arr[idx],

    // no call
    printTargetSumSubsets(arr, idx + 1, set, sum, tar);
}
Run | Debug
public static void main(String[] args) {
```

Print Encoding: Problematic Situat.

str.charAt(0) == '0'

a → 1
b → 2
c → 3
d → 4
e → 5
f → 6
g → 7
h → 8
i → 9
j → 10
k → 11
l → 12
m → 13
n → 14
o → 15
p → 16
q → 17
r → 18
s → 19
t → 20
u → 21
v → 22
w → 23
x → 24
y → 25
z → 26

'a' + 5 → g

97 + 5

(12)

5 5 1 9 6

5 1 9 6

6 5 5 1 9 6

65

10 < 65 < 26

26