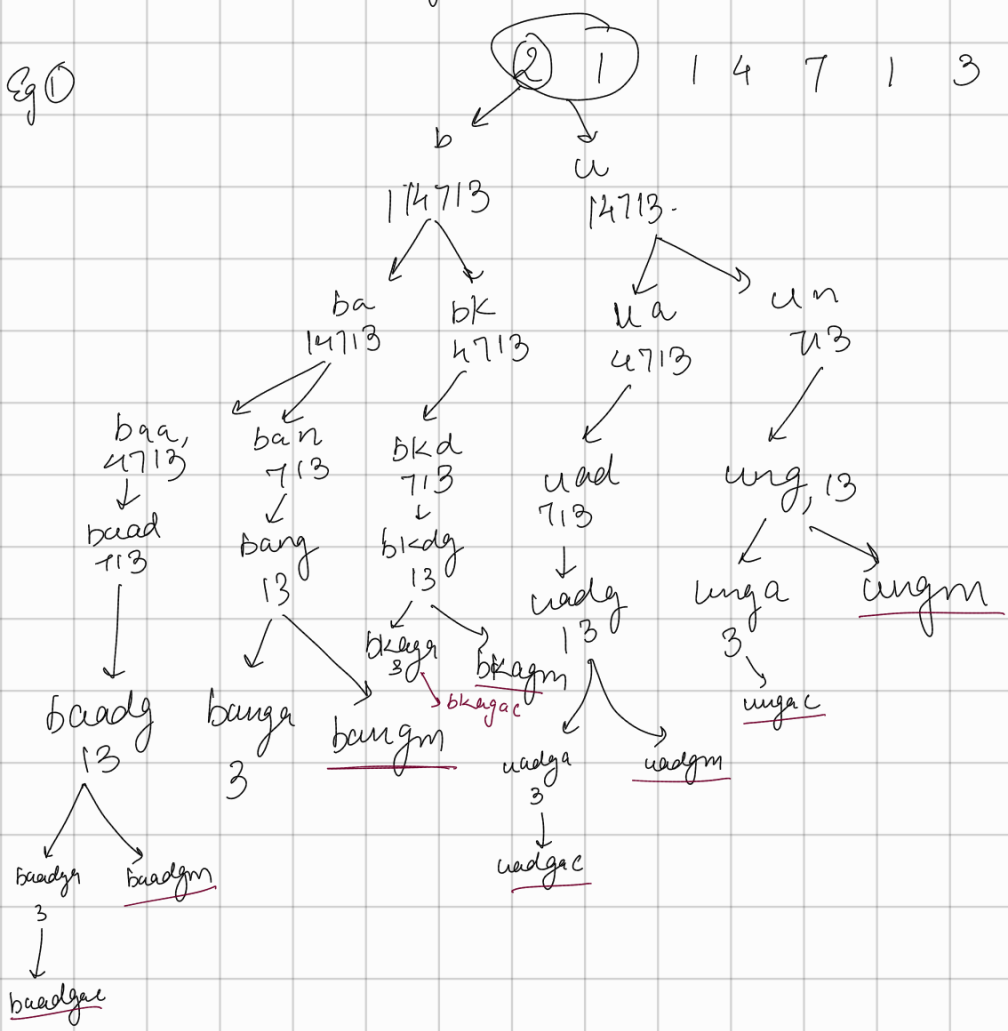
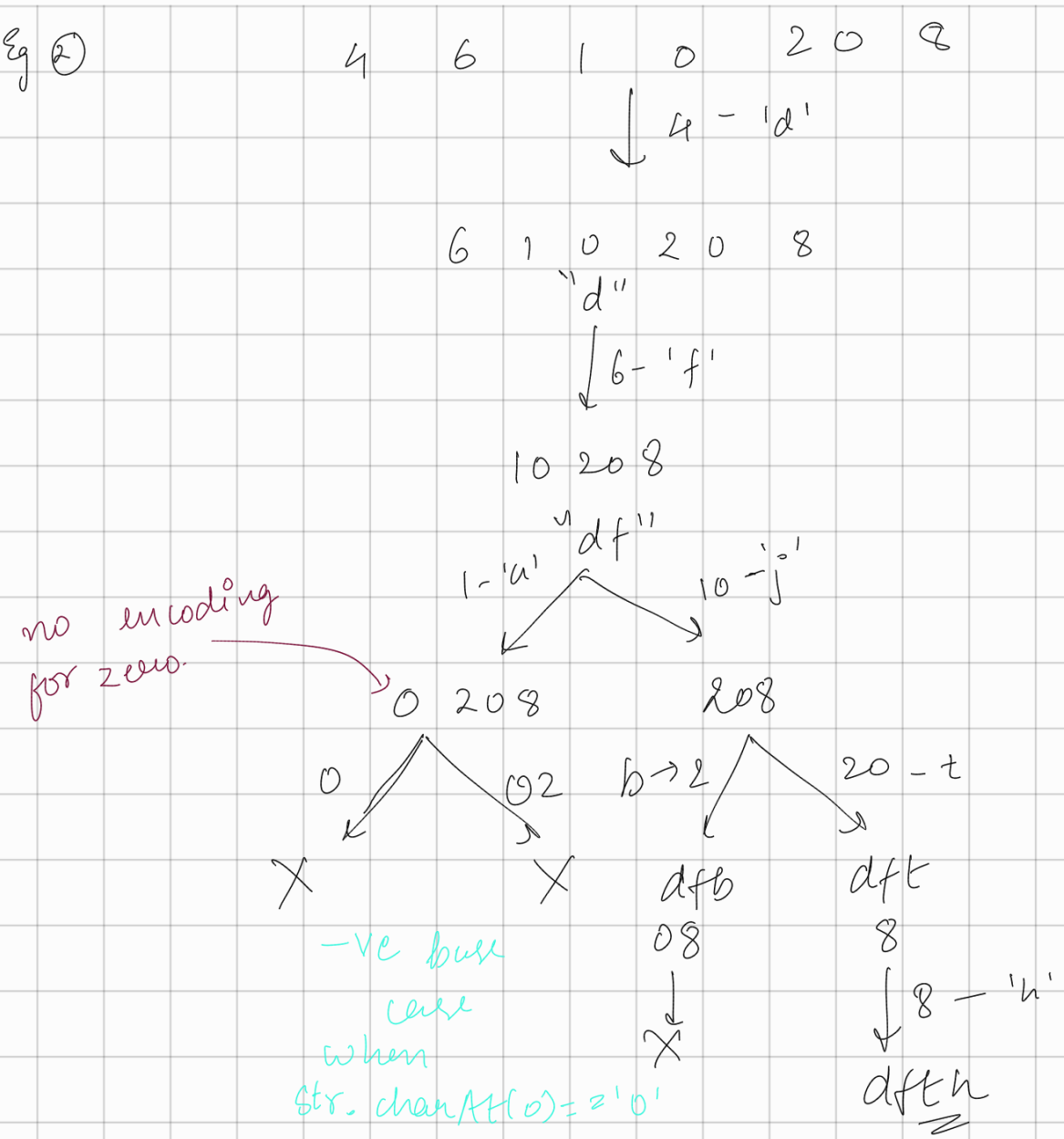
Point Encodings

Eg ①

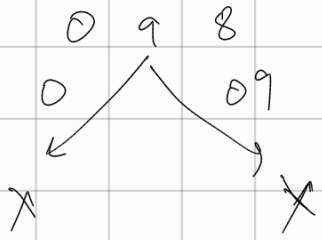


1	a
2	b
3	c
4	d
5	e
6	f
7	g
8	h
9	i
10	j
11	k
12	l
13	m
14	n
15	o
16	p
17	q
18	r
19	s
20	t
21	u
22	v
23	w
24	x
25	y
26	z

Eg ②



Eg ③



∴ CODE:

```
printEncodings(int idx, String input, String output){
    if(idx == input.length()){
        System.out.println(output);
        return;
    }
```

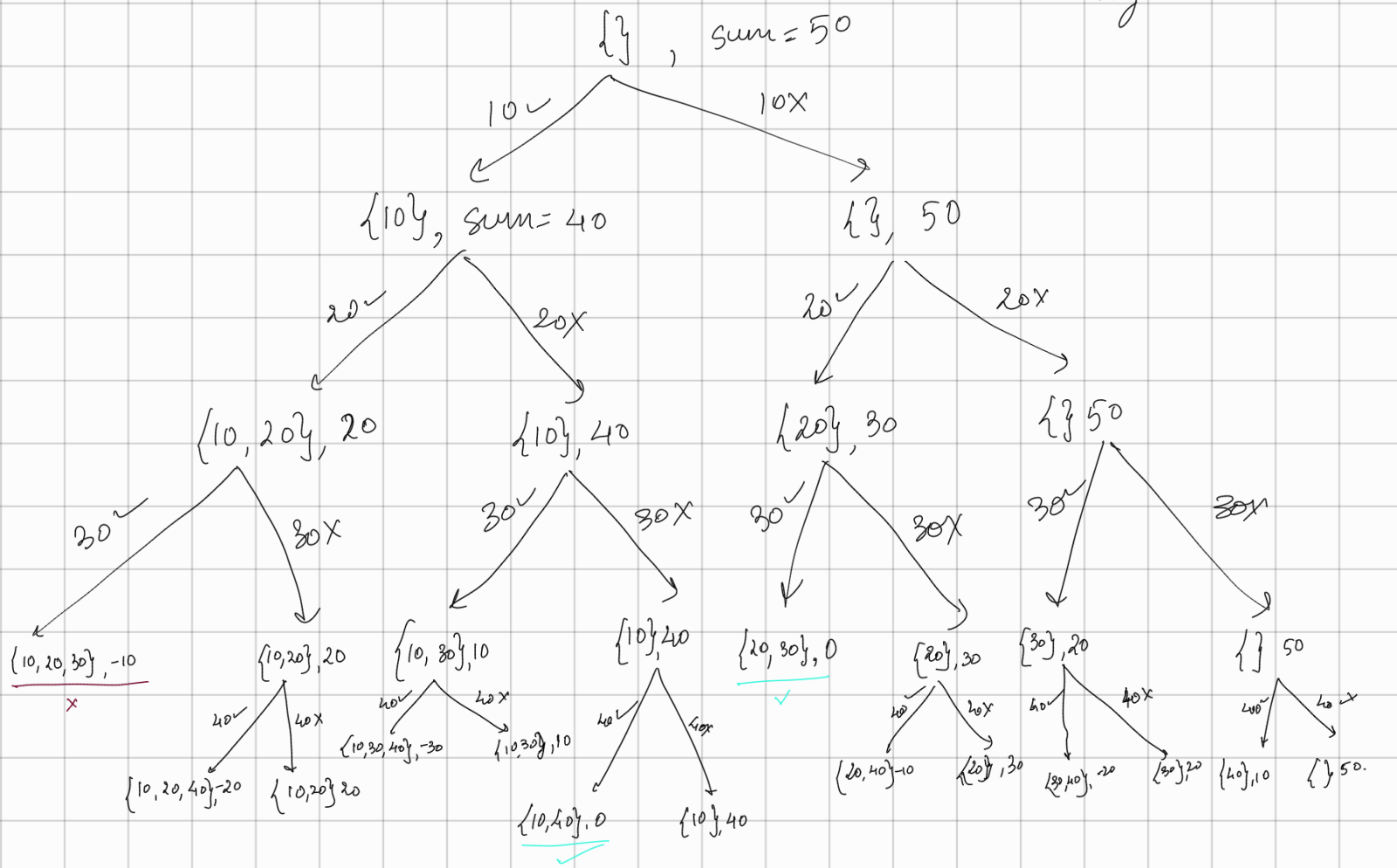
```
int ch1 = input.charAt(idx) - '0';
if (ch1 >= 1 && ch1 <= 9)
    printEncodings(idx+1, input, output + (char)('a' + ch1 - 1));

if (idx+1 < input.length()){
    int ch2 = (input.charAt(idx) - '0') * 10 + (input.charAt(idx+1) - '0');
    if (ch2 >= 10 && ch2 <= 26)
        printEncodings(idx+2, input, output + (char)('a' + ch2 - 1));
}
}
```

② Target Sum Subset.

Eg: target = 50 , {10, 20, 30, 40}

Elements are distinct & Positive

☆ Pruning

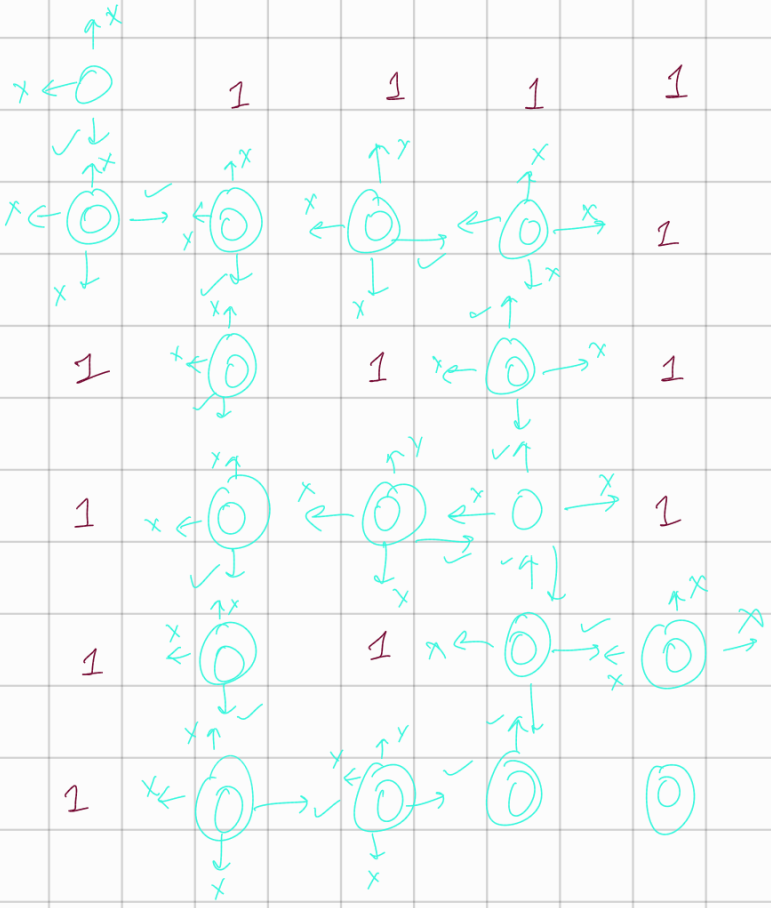


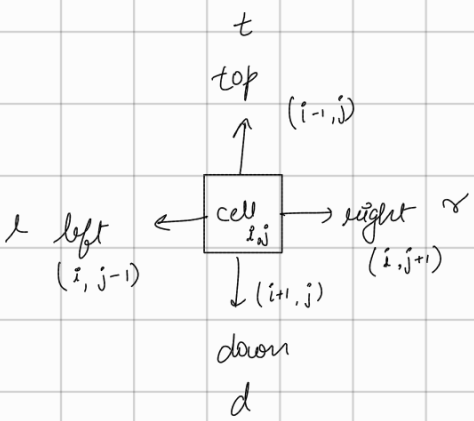
∴ CODE:

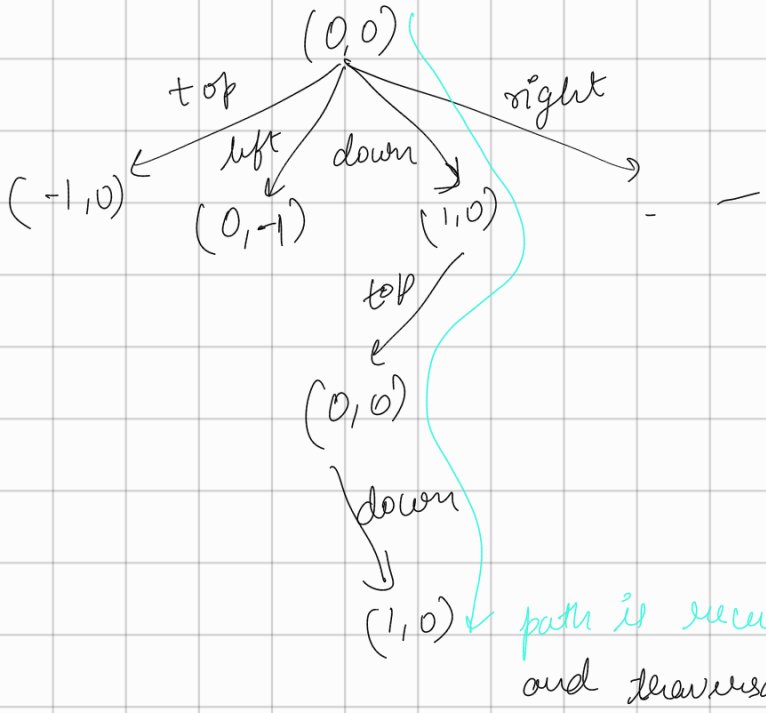
```
printTargetSumSubsets(int[] arr, int idx, String set, int remTarget) {
    if (idx == length) {
        if (remTarget == 0)
            System.out.println(set + ".");
        return;
    }

    if (remTarget < 0)        // Pruning
        return

    printTargetSumSubsets(arr, idx+1, set + arr[idx] + ", ", remTarget - arr[idx]);
    printTargetSumSubsets(arr, idx+1, set, remTarget);
}
```

only stop after reaching the end to account for zero elements

③ Flood fill / Rat in a Maze







So, to prevent such recurring issues. Use visited array concept.
When arriving at a node, mark it as visited

And for each visited node, after all 4 directions are explored by DFS, then mark it as unvisited also.

We can print only 1 path, with recursion but without backtracking
So, in order to explore all paths, we need to backtrack after we have explored DFS for all directions from a cell

∴ CODE:

```
floodfill (int[][] maze, int sr, int sc, String asf, boolean[][] visited){
    int dr = maze.length-1, dc = maze[0].length-1;
    if(sr >dr || sc >dc || sr <0 || sc <0 || maze[sr][sc]==1 ||
                                                visited[sr][sc] == true)
        return;  // negative base case
    if( sr == dr && sc == dc){
        System.out.println(psf);
        return;
    }
}
```

```
visited[sr][sc] = true;
floodfill(maze, sr-1, sc, psf + "t", vis);    // top
floodfill(maze, sr, sc-1, psf + "l", vis);    // left
floodfill(maze, sr+1, sc, psf + "d", vis);    // down
floodfill(maze, sr, sc+1, psf + "r", vis);    // right
visited[sr][sc] = false;  // backtracking
}
```