Starting at T+10.

Today's Agenda:-

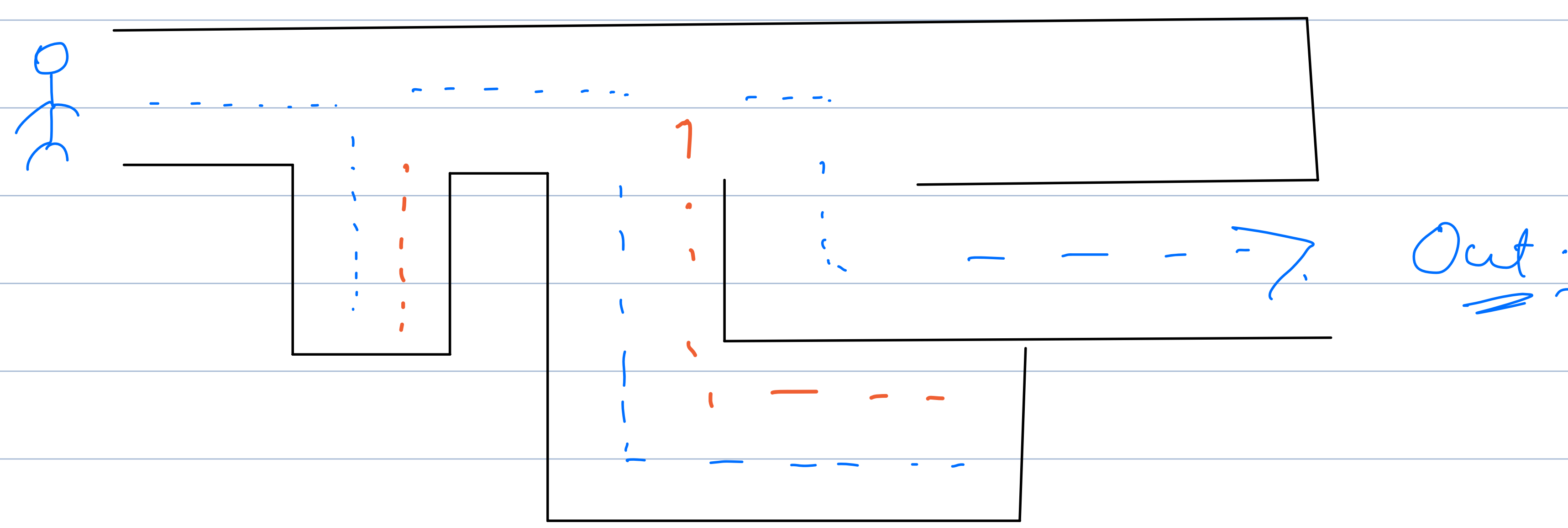
1. Introduction To Backtracking.
2. 3 Questions
   a) Rat in a Maze
   b) Permutations 1
   c) Permutations 2

Backtracking.

Exploring all possibilities using recursion..

An algorithmic technique..

Maze



RAT IN A MAZE

Check if it is possible to go from top-left to bottom right cell in a maze with blocked cell.

Note :- You cannot visit a cell more than once

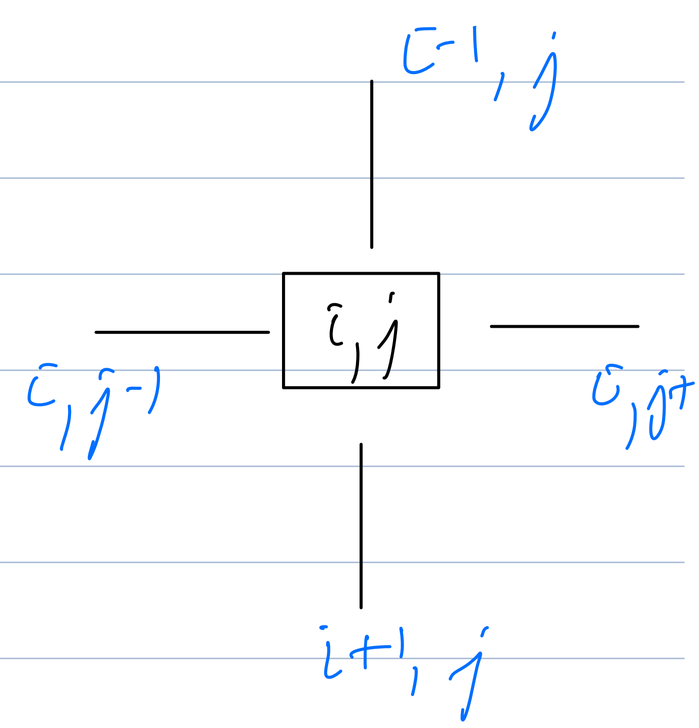
start

	0	1	2	3	4	5	6
0	0	0	0	1	0	0	0
1	0	1	0	1	0	1	0
2	0	1	0	0	1	0	0
3	0	0	1	0	1	0	1
4	1	0	1	0	0	0	0
5	0	0	0	1	0	1	0

end

arr[i][j] = '0' (empty)

arr[i][j] = '1' (blocked)



$N \times M$.

$(N-1), (M-1)$

0 0

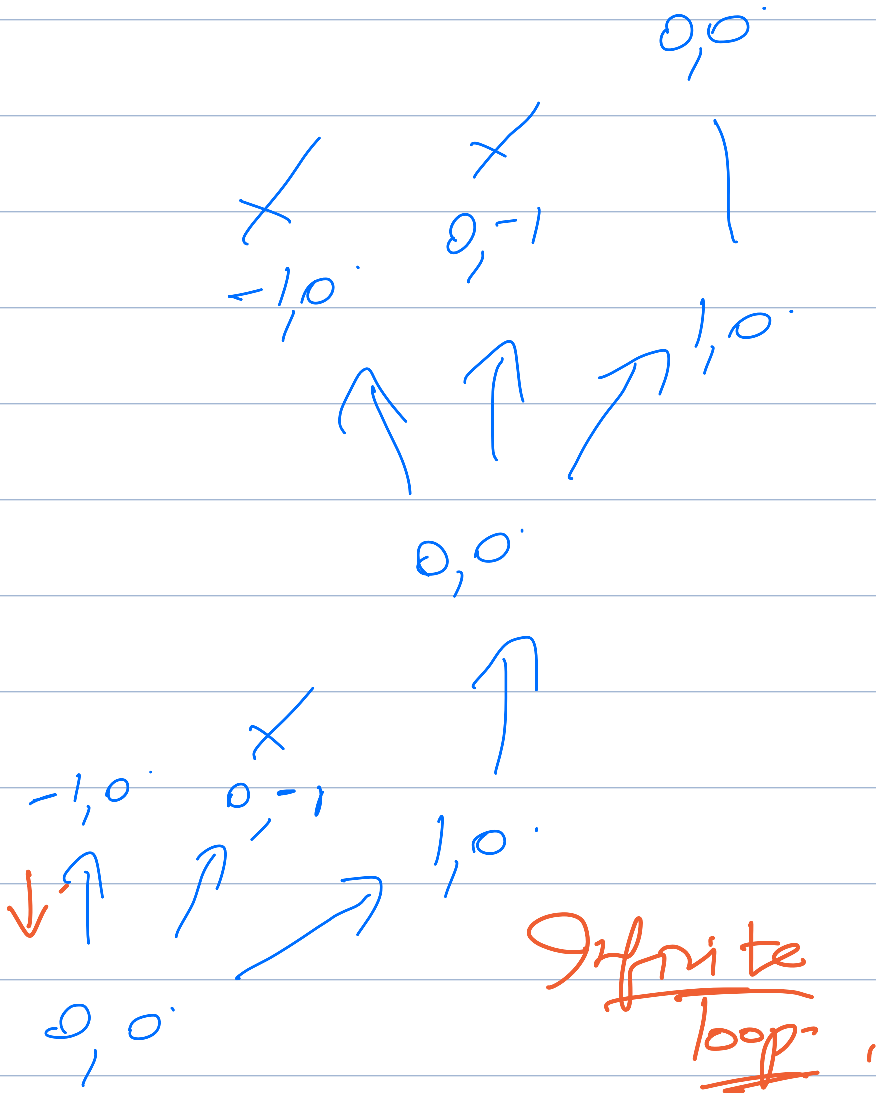
```
boolean check (arr[][], i, j) {
    if (i == N-1 && j == m-1) {
        return true; }

if ( i<0 || i >= N || j<0 || j >= M || arr[i][j] == 2
   || arr[i][j] == 1) { return false; }
    arr[i][j] = 2 ;

return   check ( arr[][], i-1, j) ||
         check ( arr[][], i, j-1) ||
         check ( arr[][], i+1, j) ||
         check ( arr[][], i, j+1) ;
};
```

start

	0	1	2	3	4	5	6
0	0	0	0	1	0	0	0
1	0	1	0	1	0	1	0
2	0	1	0	0	1	0	0
3	0	0	1	0	1	0	1
4	1	0	1	0	0	0	0
5	0	0	0	1	0	1	0



Infinite loop.

cell [i][j] → 0 (Empty cell)
→ 1 (Blocked cell)
→ 2 (Visited cell)

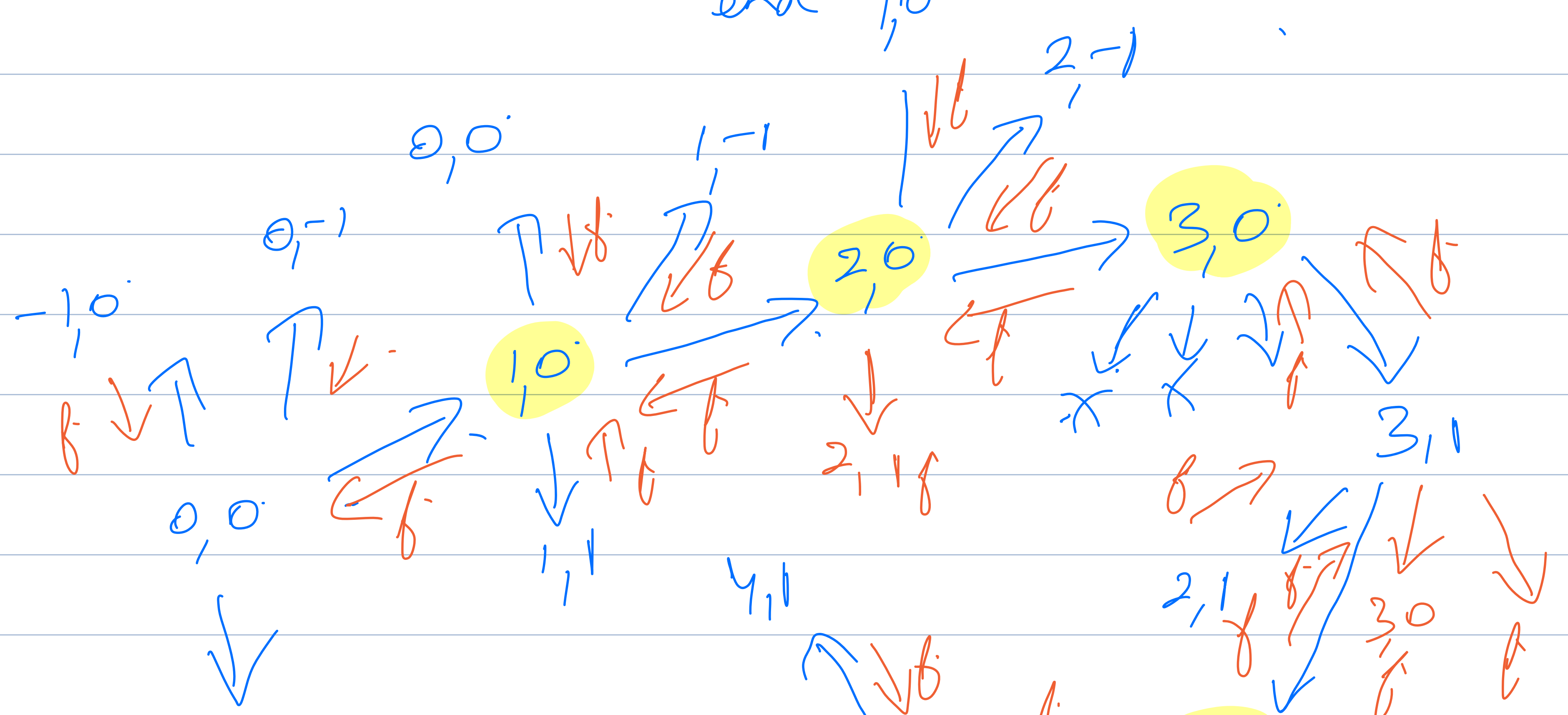
m+n

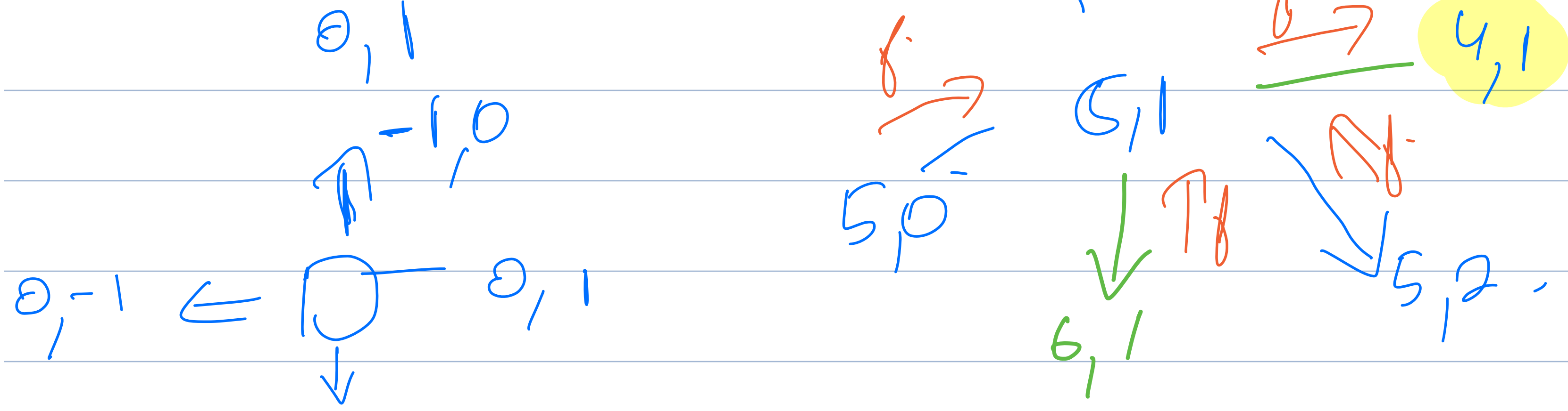
start

	0	1	2	3	4	5	
0	2	0	0	1	0	0	0
1	2	1	0	1	0	1	0
2	2	1	0	0	1	0	0
3	2	2	1	0	1	0	1
4	1	2	1	0	0	0	0
5	2	2	2	1	0	1	0

at x = a || b || c.

end





T.C → O(m × N).

S.C. → O(m × N)

```
boolean check(arr[][], i, j) {
    if (i == N-1 && j == m-1) {
        return true; }

    arr[i][j] = 2;

    dx = {-1, 0, 0, 1}
    dy = {0, 1, -1, 0}
    for (k=0; k<4; k++) {
        ni = i + dx[k];
        nj = j + dy[k];
        if (ni >= 0 && ni < N && nj >= 0 &&
            nj < m && arr[ni][nj] == 0)
        { boolean ans = check(arr, ni, nj); }
          if (ans == true)
            { return true; }
        }
}
    return false;
```

8:40.

PERMUTATIONS

Given a string of characters. Print all N unique possible permutations of that string.

S = "abc"

N!

a b c
a c b
b a c
b c a
c a b
c b a

_ _ _ idx=0

a b c

a _ _ b _ _ c _ _ idx=1

Keep track of used characters.

b c

ab_ ac_ ba_ bc_ ca_ cb_ idx=2

abc acb bac bca cab cba.

{a, b, c}. 0 { }

void perm1 (char[] arr, int idx, char[] ans, visited[]) {

{f, f, f}.

if (idx == N) { print (ans); return; }

for (i=0; i<N; i++) {

if (visited[i] == false) {

visited[i] = true;

ans[idx] = arr[i];

perm1 (arr, idx+1, ans, visited).

visited[i] = false;

}

}

}

idx = 0;

i

0 1 2.

char [a, b, c].

visited {f, f, f}

ans - {a c b}

0 1 2

(arr, 0, ans)

i=0 i=1

(arr, 1, ans)

abc.

acb.

$i=0$ $\quad$ $i=1$ $\quad$ $i=2$

(arr, 2, ans) $\qquad$ (arr, 2, ans)

$i=0$ $\quad$ $i=1$ $\quad$ $i=2$

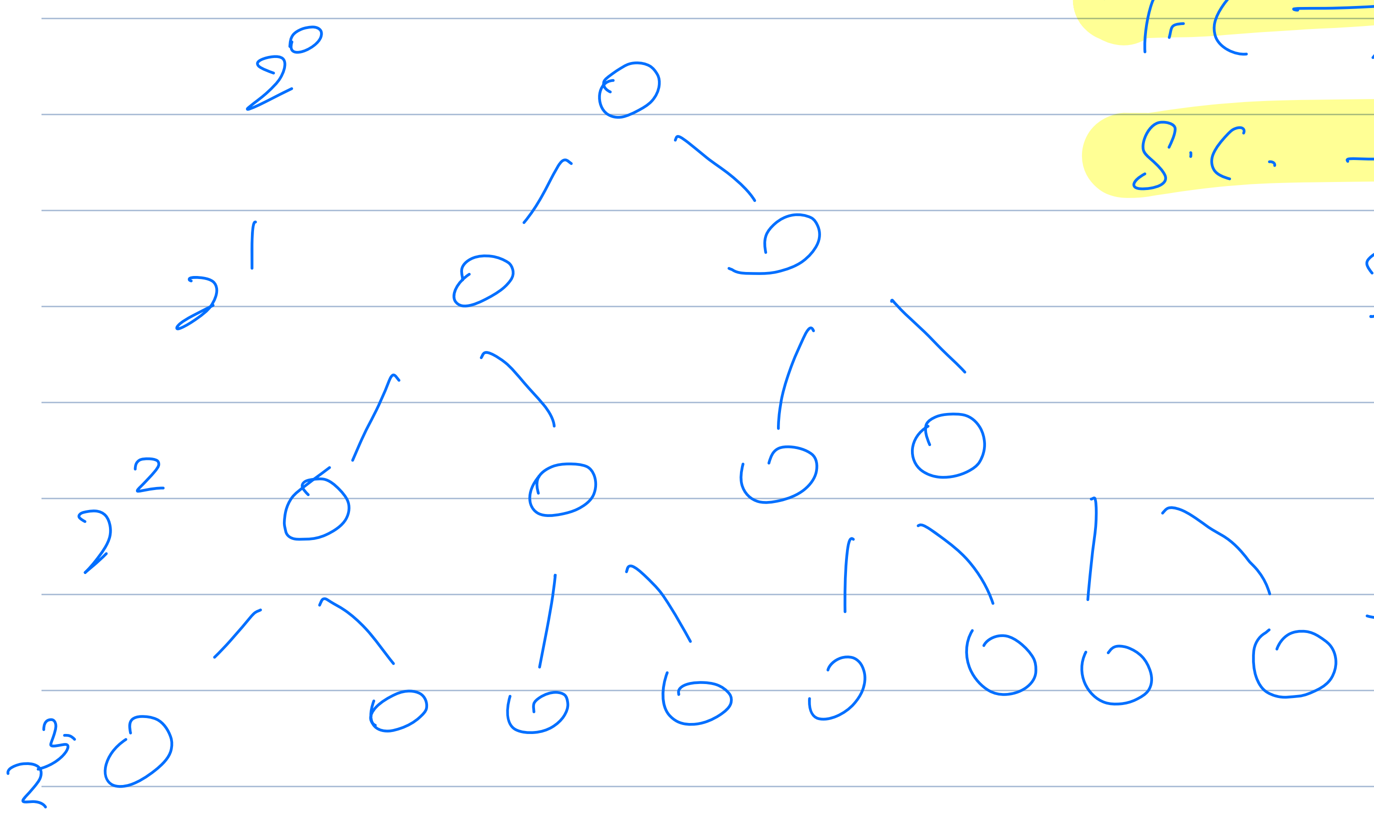
(arr, 3, ans) $\qquad$ perm(arr, 3, ans)

T.C $\rightarrow O(N!)$

S.C. $- O(N + N)$

$\simeq O(N)$.

$= 2^N$



2^N

$2^{N-1} = 2^N/2$.

2^{N-1} leaf nodes

$2^N - 2^{N-1}$

$= 2^{N-1}$.

PERMUTATIONS 2.

Print all unique permutations of the given character array

str → [a b a].

{2, 1, 0, 0, 0 . . . . . 0}
0 1 2 3 — — — 25

aab.
aba
baa.

a-2
b-1

_ _ _

idx = 0

Amortized O(1)

a-1
b-1

a-2.

a _ _   b _ _

idx = 1

b-1   a-1   a-1

aa_   ab_   ba_

aab   aba   baa

"" 0

```
void perms2 (freq[26], N, ans, idx) {
    if (idx == N) { print(ans);
                    return; }.
```

```
for (i = 0; i < 26; i++) {
    if (farr[i] > 0) {
        farr[i]--;
        ans[idx] = (char)(i + 'a');
        perms2(farr, N, ans, idx+1);
        farr[i]++;
    }
}
}
```

T.C → $O(N!)$

S.C → $O(N+C)$

$\approx O(N)$

'a' − 'a' = 0 ; = 0 + 'a'

'b' − 'a' = 1 ; = 1 + 'b'

'c' − 'a' = 2