Lecture :- Backtracking 2

Agenda

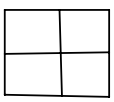
- N Queens
- Sudoko
- Word break.

Backtracking :- 1. Constraints are very less.
2. Recursion.

Qu1 **N Queens**

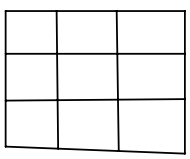
Given n * n board, place n queens such that no 2 queens attack each other.

n = 2



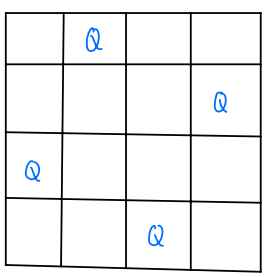
false

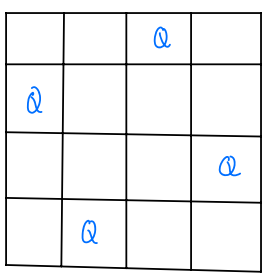
n = 3



false

n = 4



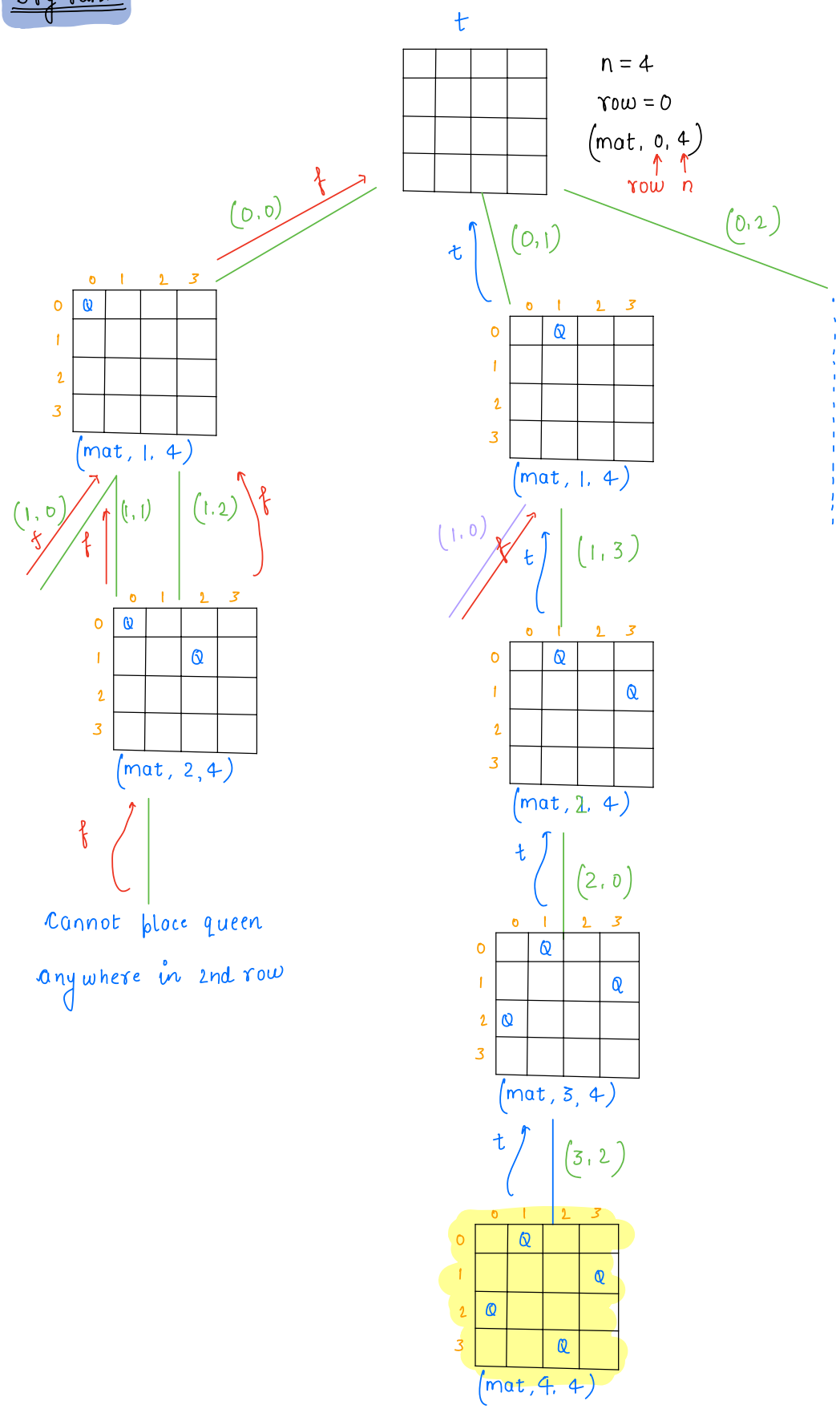


true

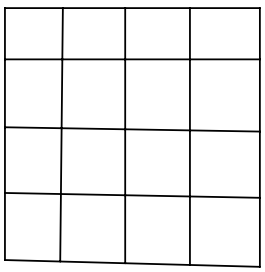
Observation Can't have more than 1 queen in a single row or col.

Plan placing your queens either row by row or col by col

Dry run:


t
n = 4
row = 0
(mat, 0, 4)
row n
(0,0)
(0,1)
(0,2)
(mat, 1, 4)
(mat, 1, 4)
(1,0)
(1,1)
(1,2)
(1,3)
(mat, 2, 4)
(mat, 2, 4)
(2,0)
(mat, 3, 4)
(3,2)
(mat, 4, 4)
Cannot place queen anywhere in 2nd row

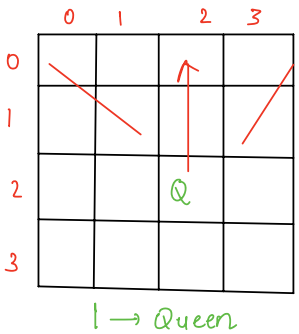
Code:



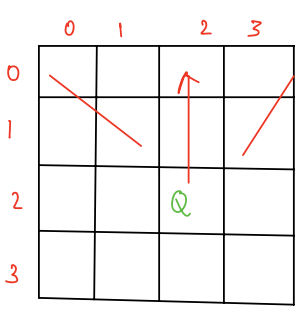

```
void nQueens(mat[][], row, n) {
                              ↑ 0 initially
    if(row == n) {
        print(mat);
        return;
    }
    for(col=0; col<n; col++) {
        boolean safe = issafe(mat, row, col);
        if(safe == true) {
            mat[row][col] = 1;
            nQueens(mat, row+1, n);
            mat[row][col] = 0;
        }
    }
}
```

Implementation of isSafe (mat, row, col)

```
boolean isSafe(mat[][], r, c) {          — O(n)

    // Check for cols
    for(i=0; i<row; i++) {
        if( mat[i][col] == 1 ) {
            return false;
        }
    }

    // Check for diagonal
        i = r-1;
        j = c-1;
        while( i>=0 && j>=0) {
            if( mat[i][j] == 1 ) {
                return false;
            }
            i--;
            j--;
        }

    // Check for diagonal
        i = r-1;
        j = c+1;
        while( i>=0 && j<n) {
            if( mat[i][j] == 1 ) {
                return false;
            }
            i--;
            j++;
        }
    return true;
}
```

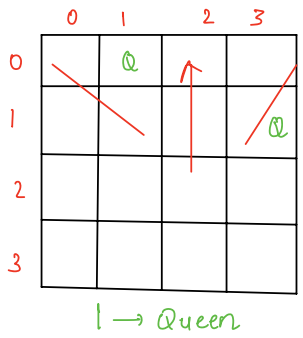


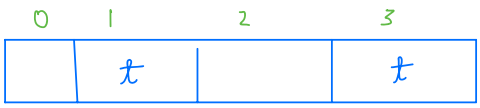
1 → Queen
- ROW (don't check)
- col
- diagonals



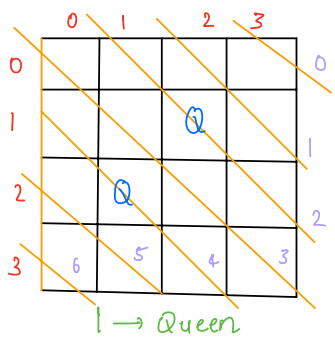
1 → Queen

Qu Can we optimise the isSafe() func^n to O(1) time?





col[4]



n-1

Diag 0 :- r − c = −3 + 3 = 0

0 3

Diag 1 :- r − c = −2 + 3 = 1

Diag 2 : r − c = −1 + 3 = 2

Diag 3 : r − c = 0 + 3 = 3

Diag 4 : r − c = 1 + 3 = 4

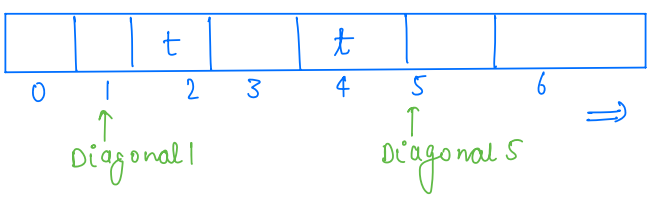
Diag 5 : r − c = 2 + 3 = 5

Diag 6 : r − c = 3 + 3 = 6

Queen (2, 1)

↑ diag

(2 − 1) + 3 = 4

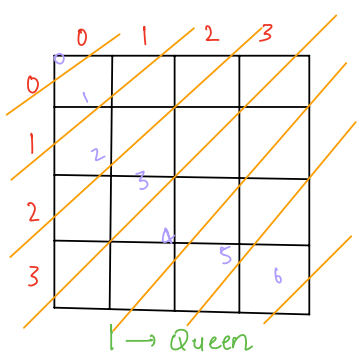


⟹ [2n − 1]

Queen (1, 2)

↑ diag

(1 − 2) + 3 = 2



D0 : r + c = 0

D1 : r + c = 1

D2 : r + c = 2

D3 : r + c = 3

D4 : r + c = 4

⋮

D6 : r + c = 6

Code:

```
boolean[] col = new boolean[n];
              diag1 =          [2n-1];
              diag2 =          [2n-1];

void nQueens ( mat[][], row, n ) {
                        ↑ 0 initially
    if ( row == n ) {
        print (mat);
        return;
    }
    for ( c=0; c<n; c++) {
        boolean safe = col[c]==f && diag1[row-c+n-1]==f
                       && diag2[row+c] ==f;
        if ( safe == true) {
            col[c] = true;
            diag1[row-c+n-1]= true;
            diag2[row+c] =true;
            mat[row][col] =1;
            nQueens ( mat, row+1, n);
            mat[row][col] =0;
            col[c] = false;
            diag1[row-c+n-1]= false;
            diag2[row+c] = false;
        }
    }
}
```

Break:
8:18 - 8:30 AM.

TC: $O(n^n)$
SC: $O(n)$ + isSafe Manipulation space

Qu: Fill sudoko

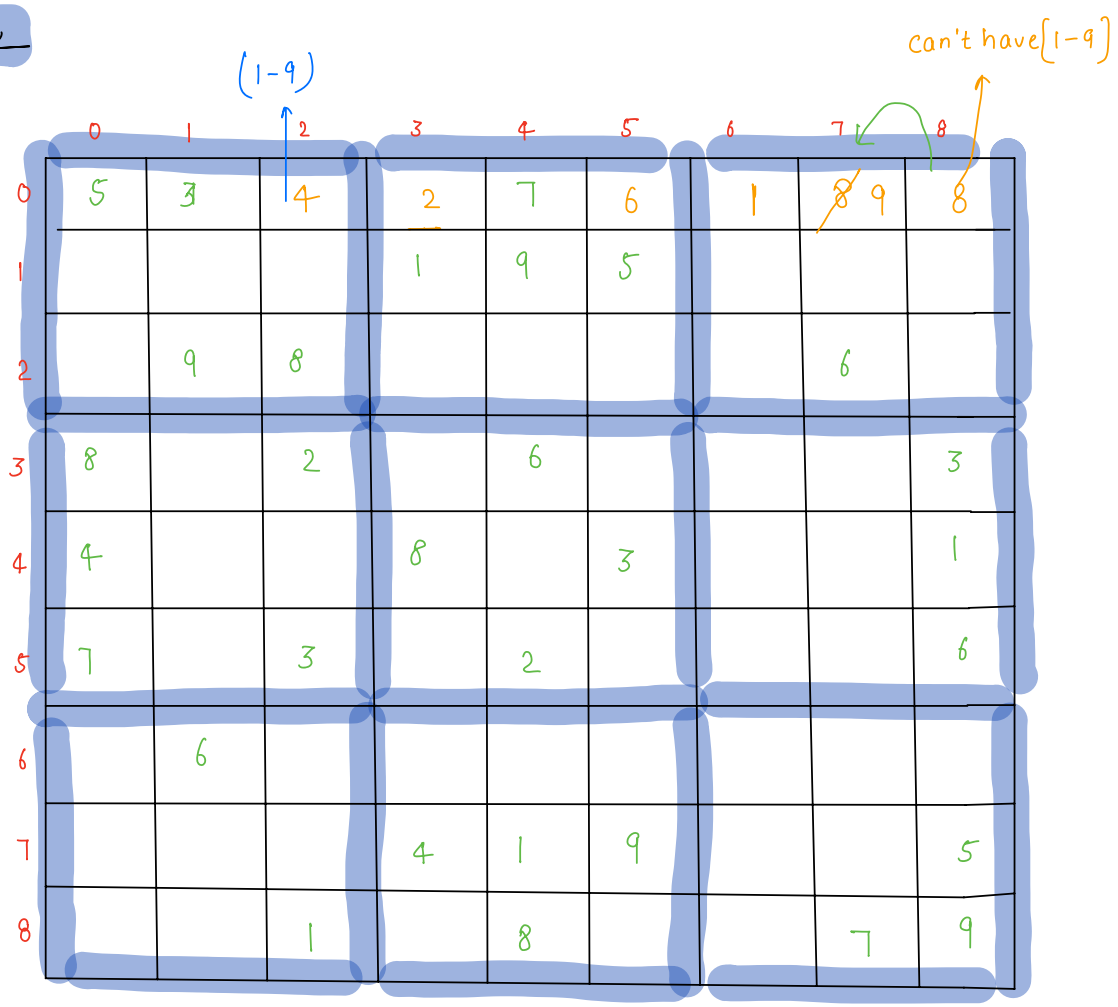
- Given 9*9 matrix. Every cell 1-9.

Rules

- In row, data can't repeat.
- In col, data can't repeat.
- In 3*3 matrix, data can't repeat

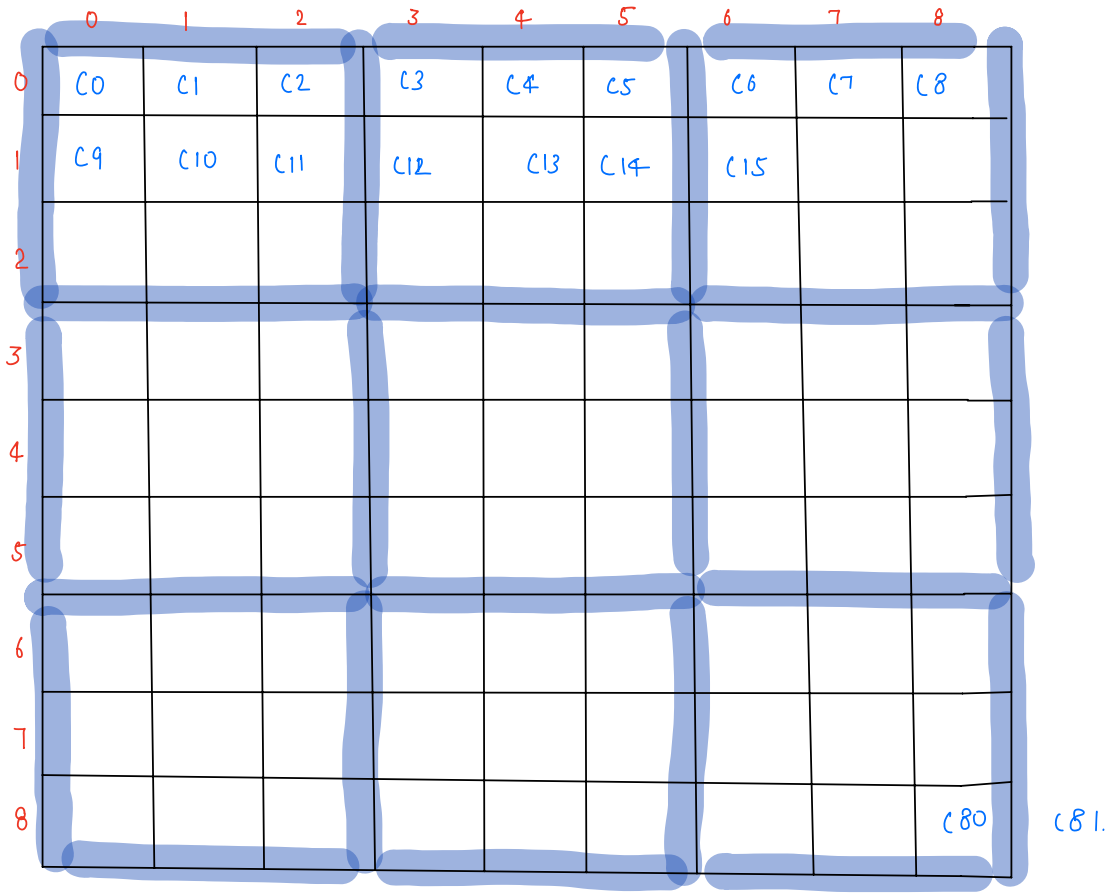
mat[i][j] = 0 means not filled.

Approach


(1-9)
can't have (1-9)
0 1 2 3 4 5 6 7 8
0 5 3 4 2 7 6 1 8 9 8
1 1 9 5
2 9 8 6
3 8 2 6 3
4 4 8 3 1
5 7 3 2 6
6 6
7 4 1 9 5
8 1 8 7 9

Qu How to start filling the sudoko?

fill it cell by cell.




```
if( I reach C81 ){
    return true;
}
```

Flow

Code:

```
boolean sudoko(mat[][], cell) {
    if(cell == 81) {
                ↑
               n*n
        return true;
    }
    i = cell/9;
    j = cell%9;
    if(mat[i][j] != 0) {
        return sudoko(mat, cell+1);
    }
    for(k=1; k<=9; k++) {
        boolean safe = issafe(mat, i, j, k);
        if(safe) {
            mat[i][j] = k;
            sa = sudoko(mat, cell+1);
            if(sa == true) {
                return true;
            }
            mat[i][j] = 0;
        }
    }
    return false;
}
```

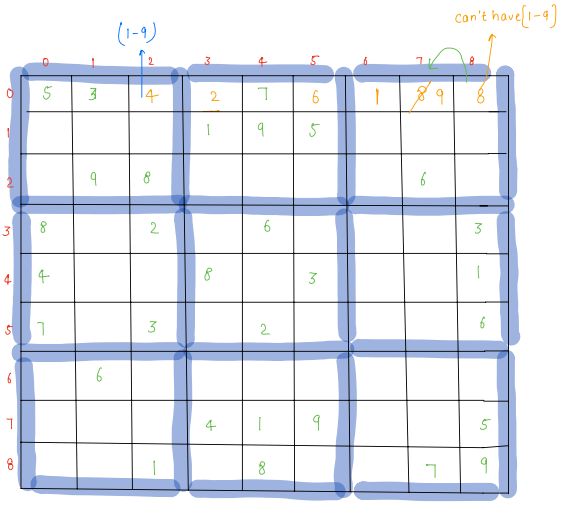
x = no of unfilled cells

TC: 9^x

SC: $O(n)$

* isSafe (mat[][], r, c, k) func^{n} ?

```
boolean isSafe( mat[][] , r, c, K) {
        // Cols
    for(i=0; i<n; i++) {
        if ( mat[i][c] == K) {
            return false;
        }
    }

    // Rows
    for(i=0; i<n; i++){
        if( mat[r][i] == K){
            return false;
        }
    }

    // Box
    start_row = r - r%3;
    start_col = c - c%3;
    for(i = start_row; i< start_row+3; i++){
        for(j= start_col; j< start_col+3; j++){
            if ( mat[i][j] == K) {
                return false;
            }
        }
    }

    return true;
}
```



// Col

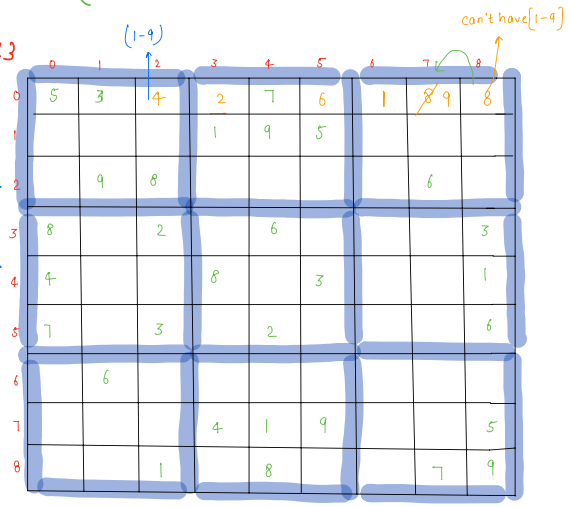
// Row

// Box

Eg: (4, 5) - box (3,3)

4 - 4%3 → 3

5 - 5%3 → 3



TC:

SC:

Qu Word break.

dict[] = { i , like , sam , sung , mango , samsung , mobile , ice , cream }

sentence = ilikemango → true

Ans:

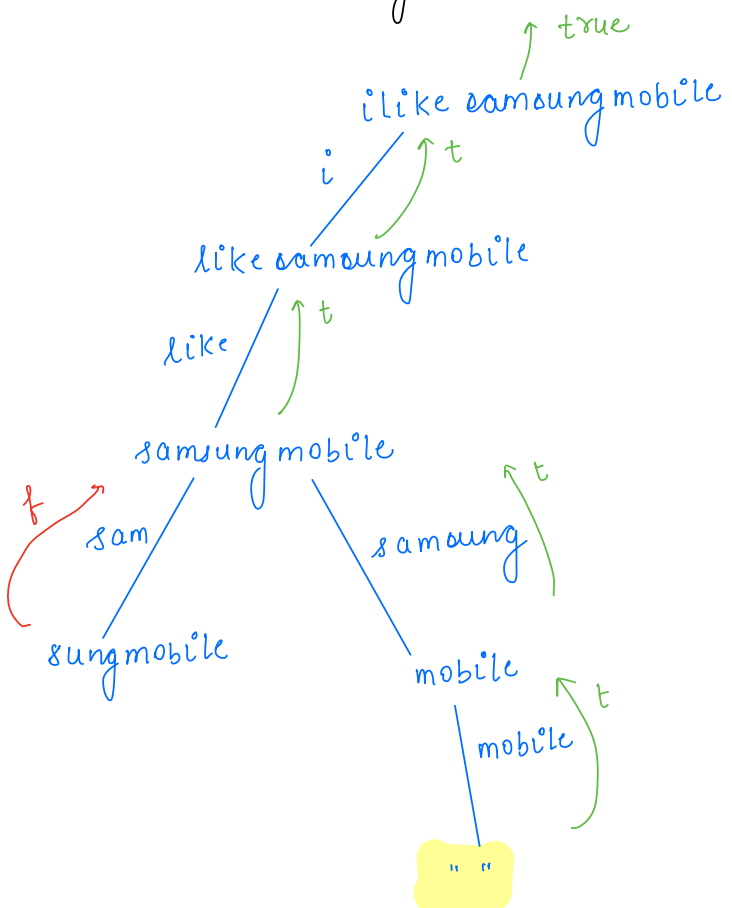
sentence = ilikemangoicecream

sentence = ilovemango = false

Approach:

dict[] = { i , like , sam , go , mango , samsung , mobile , ice , cream . } man

sentence = ilike samsung mobile.



Explanation:

str = 0 1 2 3 4 5 6 7 8 9 10 11
samsung mobile

prefix = str.substring(0,3) = sam.

↑ is in dict

Rec = sung mobile
(str.substring(3))

Code:

```
boolean wordBreak(Hashset(string) dict, String word) {
    if(word.length()==0) {
        return true;
    }
    for(i=1; i<=word.length(); i++){
        prefix= word.substring(0,i);
        if(dict.contains(prefix)){
            sa = wordBreak(word.substring(i));
            if(sa==true){
                return true
            }
        }
    }
    return false;
}
```

TC: O(n!)

SC: O(n)

Thankyou

ilikecamaung mobile.

n
*
n-1
*
n-2
*
n-3
*
n-4
.
.
1