Todays Content:

- for each query, check if given query is prefix of any N words
- Given 2D mat[N][M] find number of unique rows
- Given ar[N] elements find pair with max xor
- Given ar[N] elements find subarray with max xor

Q) Given N Input Strings & Q Queries, for each query check if given query is prefix of any Input Strings.

Constraints: $1 \le \text{len(Strings)} \le l$

Note: Substring starting at index = 0, full string is prefix string

Input Strings:	Queries	Yes/No
anaconda	anaco	Yes
dress	fry	No
eaten	roade	Yes
friends	algor	Yes
roades	sour	No
anaco	dress	Yes
algorithms		
sound		

Idea1: For every query, check if it's prefix to given input strings

TC: $Q * \{N * l\}$ SC: $O(1)$

Idea2: Insert all strings in Tries for every query iterate from root, if at we get null, no input string with given prefix

TC: $N * l + Q * l$ SC: $< N * l$

Obs: Tries are also known prefix Trees

↳ Hint: If Question has something related to prefix search Tries is a good idea to start

1Q) Given a binary mat[N][M] find no of distinct rows ?

Constraints: M > 64 (rows / cols)

↳ {For every row consider only 2 occurence}

mat[7][5] ans = 5

Eg: ar[5] = { 6 3 6 9 3 }

↑ ↑ ↑ ↑ ↑

✓ ✓ ✗ ✓ ✗

	0	1	2	3	4	
0	1	0	0	1	0	✓
1	0	1	0	1	0	✓
2	1	1	0	1	1	✓
3	1	1	0	0	1	✓
4	1	1	0	1	1	✗
5	1	0	0	1	0	✗
6	0	0	1	1	0	✓

Idea1: → Convert every row into string & insert into hashset<string>

TC: N(M + M) SC: N*M

Note: In a row ele = M

Note: To Concatenate M ele → Strings: TC: O(M)

Note: Insert a string of len = M in hashset TC: O(M)

Idea2: Insert in Trie

```
class Node {
    Node left
    Node right
    Node() {
        left = null
        right = null
    }
}
```

```
class Node {
    Node c[2]
    Node() {
        c[0] = null
        c[1] = null
    }
}
```

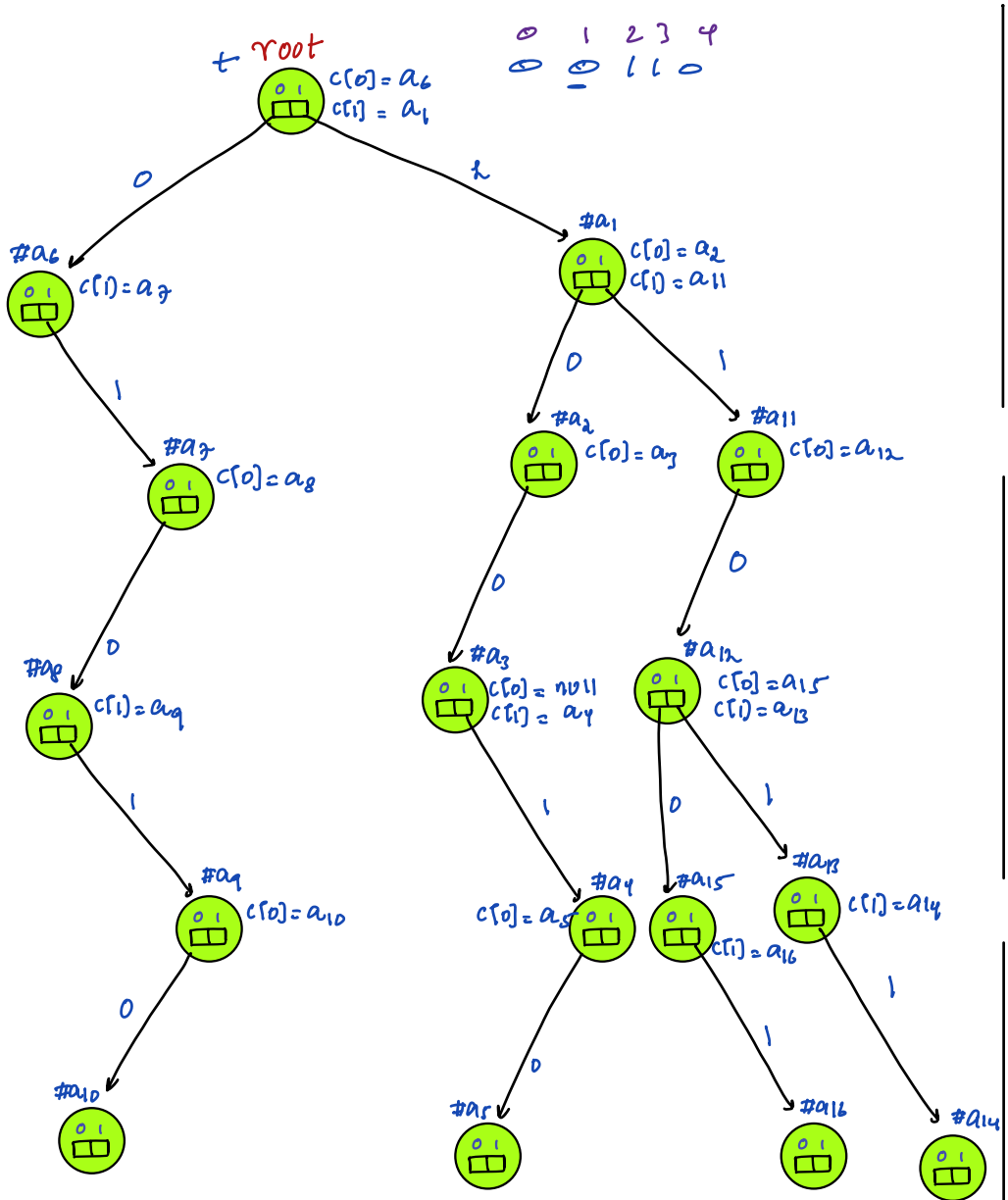
obs: data : 0 → c[0]
data : 1 → c[1]
Code is slightly easy

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

Idea2: Insert all rows in Trie

TC: O(N*M) SC: < O(N*M)

↳ 10:7 → 10:15 pm



obs: While inserting a row in trie,
if a single new node is also not created : row is already present

```
class Node {
    Node c[2];
    Node() {
        c[0] = NULL
        c[1] = NULL
    }
}
```

```
int uniqueRows(int mat[][], int N, int M) {
    Node root = new Node()
    int c = 0
    for(i=0; i<N; i++) {
        // insert ith row : mat[i]   // passing only ith row
        if(insert(mat[i], root) == true) {
            // its new row
            c = c+1
        }
    }
    return c;
}
```

```
bool insert(int ar[], Node root) {
    Node t = root
    int m = ar.length;
    bool flag = false
    for(i=0; i<m; i++) {   TC: O(m)
        // val we need to check = ar[i] at temp node t
        int val = ar[i]
        if(t.c[val] != null) { // child exist
            t = t.c[val]
        }
        else {
            flag = true // row is a new row,
            Node nn = new Node()
            t.c[val] = nn
            t = nn
        }
    }
    return flag
}
```

3Q) Given ar[N] elements, find max xor pair value → bits
↳ calculate max A ^ B

	0	1	2	3	
ar[4] =	4	3	2	7	ans = 7

pairs : 4^3 = 7 3^2 = 1 2^7 = 5
4^2 = 6 3^7 = 4
4^7 = 3

	0	1	2	
ar[3] =	2	3	7	ans = 5

pairs : 2^3 = 1 2^7 = 5 3^7 = 4

Idea: Calculate xor for all pairs & get overall max
↳ TC: O(N^2) SC: O(1)

Ideas:
1) Calculate xor between min & max fails
2) Calculate xor between 1st max & 2nd max fails
3) Sort[], Calculate xor of adjacent pair & get overall max fails

	0	1	2	3	4	5	6	7	8
ar[9] =	22	61	38	27	21	34	42	37	43

	5	4	3	2	1	0
22:	0	1	0	1	1	0
61:	1	1	1	1	0	1
38:	1	0	0	1	1	0
27:	0	1	1	0	1	1
21:	0	1	0	1	0	1
34:	1	0	0	0	1	0
42:	1	0	1	0	1	0
37:	1	0	0	1	0	1
43:	1	0	1	0	1	1

Q) Calculate max $A \wedge B = ?$

Hint: Given A = 22, find B in ar[] such that A^B is max nr value?

	5	4	3	2	1	0
A: 34:	1	0	0	0	1	0
B:	0	1	1	_	_	_
ans:	1	1	_	_	_	_

Idea: Fix A = each individual ar[] element & repeat above process

A

ar[0] → find B, such A^B is max

ar[1] → find B, such A^B is max

ar[2] → find B, such A^B is max

⋮

ar[n-1] → find B, such A^B is max

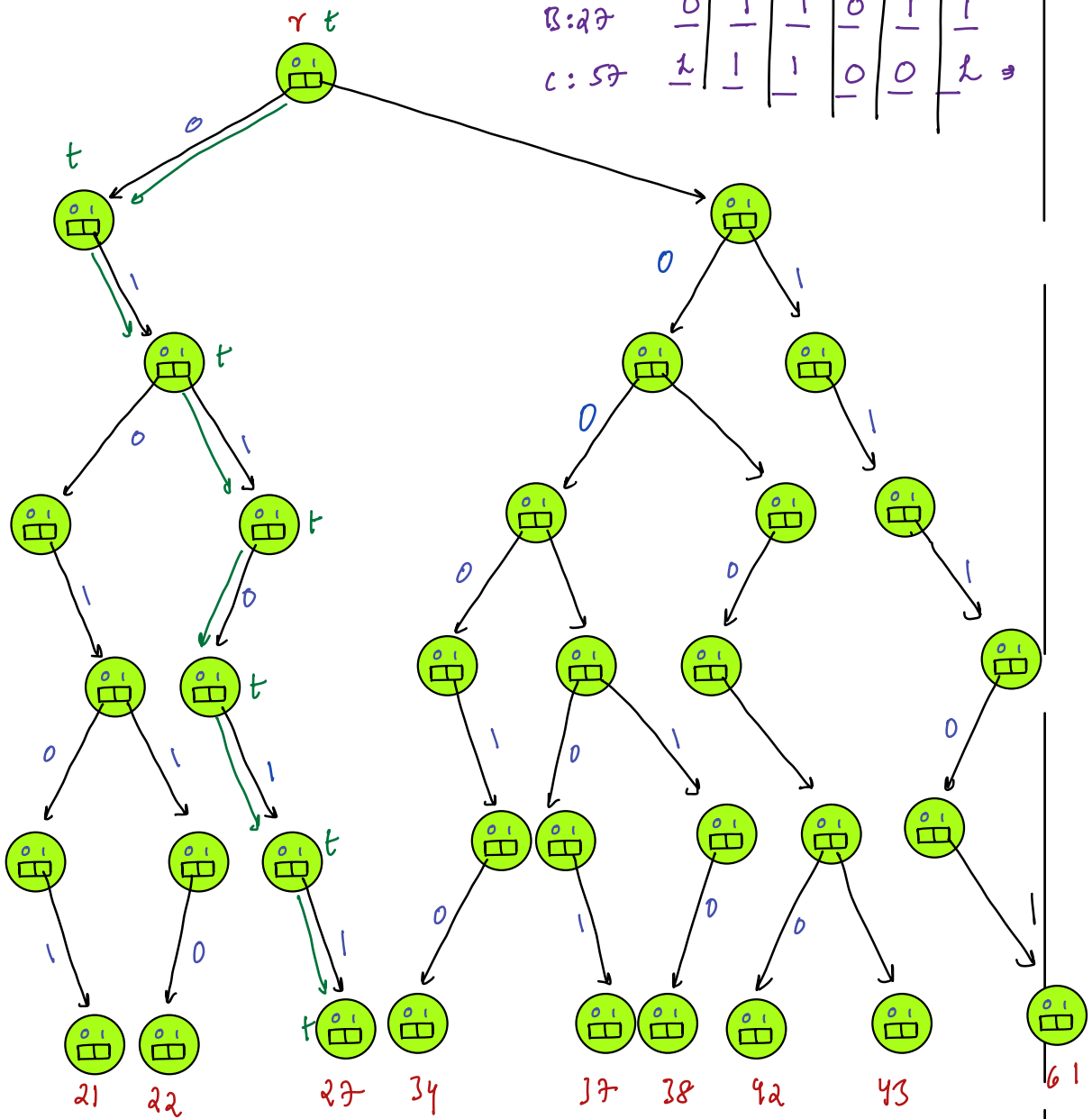
Note: Max of all of them is final ans

TC: $N * \{32 * N\} \Rightarrow 32N^2 \Rightarrow O(N^2)$

Hint: After setting a bit pos, we iterate in all ar[], to check if ar[] ele are starting with req data prefix → Trie

Inserting all elements in Binary Trie p=5

	5	4	3	2	1	0
A: 34 :	1	0	0	0	1	0
B: 27	0	1	1	0	1	1
C: 57	1	1	1	0	0	1



Note1: No: of bits we need to insert for all ele is same

Note2: How many bits?

- p : 32
- : for max of arr(), get msb say it's p
- : for all elements insert from bit p → 0

Pseudocode:

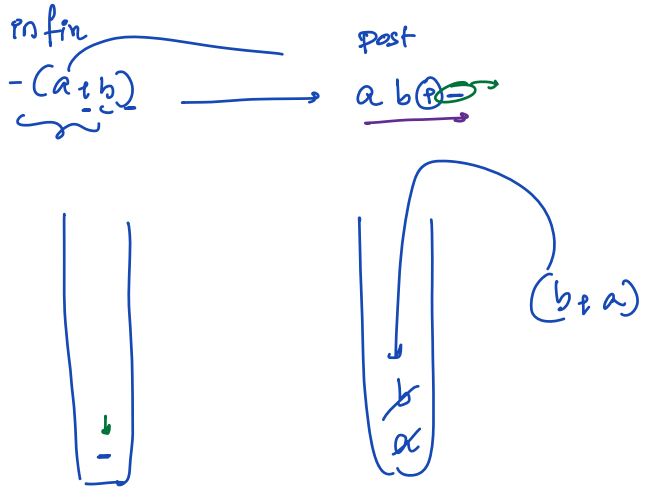
```
Class Node{
    Node c[2];
    Node(){
        c[0] = NULL
        c[1] = NULL
    }
}
```

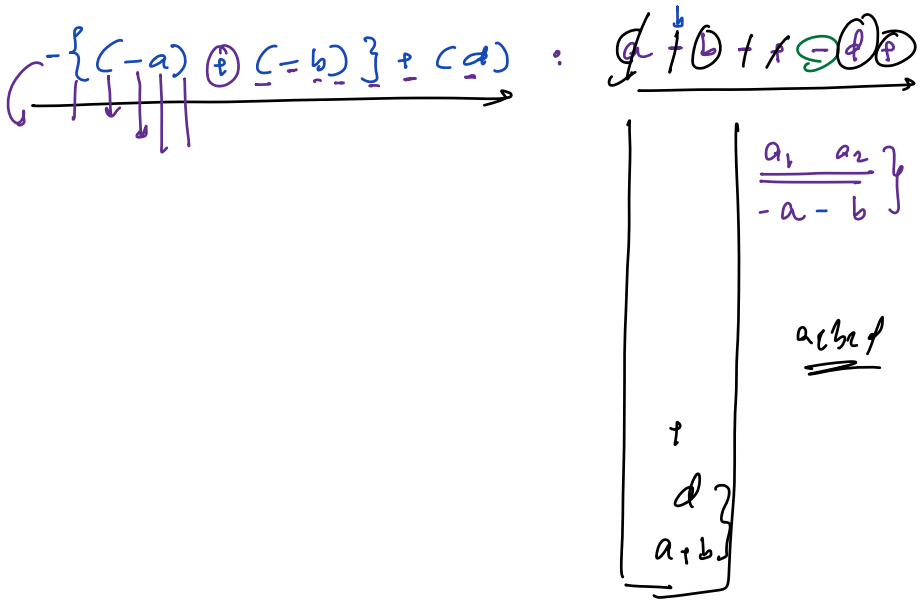
TC: N*32 → O(N) SC: N*32 → O(N)

```
int maxnr(int ar[], int N){
    int Mval = max(ar[]) → N
    int p = 31;
    for(i=31; i>=0; i--){ ⟶ 32
        // for Mval, check if i^th bit set
        if(checkBit(Mval, i)==True){
            p = i; break
        }
    }
    Node root = new Node();
    for(i=0; i<N; i++){ ⟶ N*p → 32N
        // insert ar[i] in trie
        insert(root, ar[i], p)
        // p : indicates bits p→0
    }
    ans = 0
    for(i=0; i<N; i++){ ⟶ N*p → 32N
        | max nr value with ele = A |
        int v = query(root, A, p)
        ans = max(ans, v)
    }
    return ans
}
```

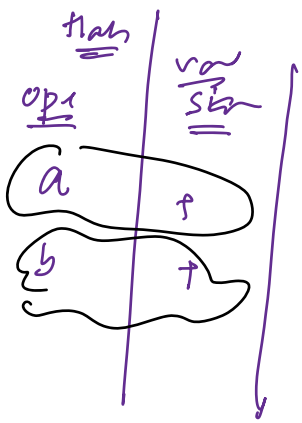
```
void insert(Node root, int ele, int p){
    Node t = root;
    // for ele insert from p→0
    for(i=p; i>=0; i--){
        // check i^th bit for ele
        int val = checkBit(ele, i)
        if(t.c[val] != null){ // child exist
            t = t.c[val]
        }
        else{
            Node nn = new Node()
            t.c[val] = nn
            t = nn
        }
    }
}
```

```
int query(Node root, int A, int p){
    int c = 0 // max xor
    Node t = root;
    for(i=p; i>=0; i--){
        // check i^th bit for ele
        int v = checkBit(ele, i)
        if(t.c[1-v] != null){
            // in nr val i^th bit = 1
            c = c + 2^i (1<<i)
            t = t.c[1-v]
        }
        else{
            // in nr val i^th bit = 0
            t = t.c[v]
        }
    }
    return c;
    // max nr with 1 ele = found
}
```


infix
- (a + b)
post
a b +
-
(b + a)
b
a


- { (- a) + (- b) } + (- d)
a + b + c - d
a1 a2
- a - b
a + b + c - d
+
d
a + b

$-(a-b)$: $-a+b$



$-(a-b)$

$-a+b$

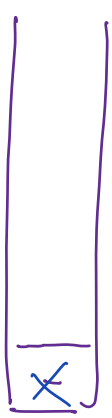
→ operand = one

$-(a+b)-(-c) = -a-b+c$

$=(a-d-(b+c-(d-g)+k)+e)$ hashmap

$-a+d+b+c--$

curr = ~~X~~ → + → − → −



a : −
d : +
b : +
c : +
d : −
g : +
k : +
e : −

→ operand : befr : '(' or ')' : sign : currentsign

→ operand : befr : -, + / check if current sign :

→ '(' : push cur sign in stack
update cur sign

→ ')' : update cur = st.top
st.pop()