→ Today's 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 N array elements find pair with max xor
- Given N array elements find subarray with max xor
  - TODO → complexity depends on above Question

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

prefix → substring starting at index = 0

Constraints : $1 \le$ len(string) $\le l$

Input Strings(N)	Queries Q	Yes/No
anaconda	anaco →	Yes
dress	fry →	No
eaten	roade →	Yes
friends	algor →	Yes
roades	sour →	No
anaco	dress →	
algorithms		
sound		

Note: Tries are also known as prefix Trees

Idea:

Step1: Insert all words in Trie

Step2: for every query, iterate in trie from root & check query is prefix or not

TC: $N * l * O(1) + Q * l * O(1)$

↳ Even to simply iterate in all queries it will take $Q * l$ time

SC: $N * l$

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

M > 60

mat[7][5] // ans = 5

{For every element consider only 1 occurence}

	0	1	2	3	4	
0	1	0	0	1	0	*
1	1	1	0	1	1	* → "11011"
2	0	1	0	1	0	✓
3	1	1	0	1	1	✓ → "11011"
4	1	1	0	0	1	✓
5	1	0	0	1	0	✓ → "10010"
6	0	0	1	1	0	✓

Idea1: For every row compare with all rows below, if freq = 0 in that case increase cnt

TC:
- [# Total rows comparsi] * [TC to each compare 2 rows]
- → O(N²) * (M) → O(N²M)

SC: O(1) → No Extra Space

Note: In hashset/hashmap to insert a string of M length it will take O(M) time, search → O(M)

Idea2:

Convert each row into String & insert in hashset

TC: N*M + N*M
- Converting each row into string
- To insert all N strings to hashset

SC: O(N*M)

Ideas:

	16	8	4	2	1	decimal
0	1	0	0	1	0	18
1	1	1	0	1	1	27
2	0	1	0	1	0	10
3	1	1	0	1	1	27
4	1	1	0	0	1	25
5	1	0	0	1	0	18
6	0	0	1	1	0	6

ans = 5

hashset ans = hs

Step 1: for every row convert it into decimal & insert in hashset

TC: N * M + N * O(1) → for inserting N elements

SC: O(N)

Issues in Ideas: Every column is a bit

datatype

Mx = 31 : int

Mx = 63 : long

M > 63 : Won't work → { We cannot store extreme number as a int/long }

Note: If Mx = 63, in that can prefer Ideas

range

M = 100 → ~ 2^{100}

M = 200 ~ 2^{200}

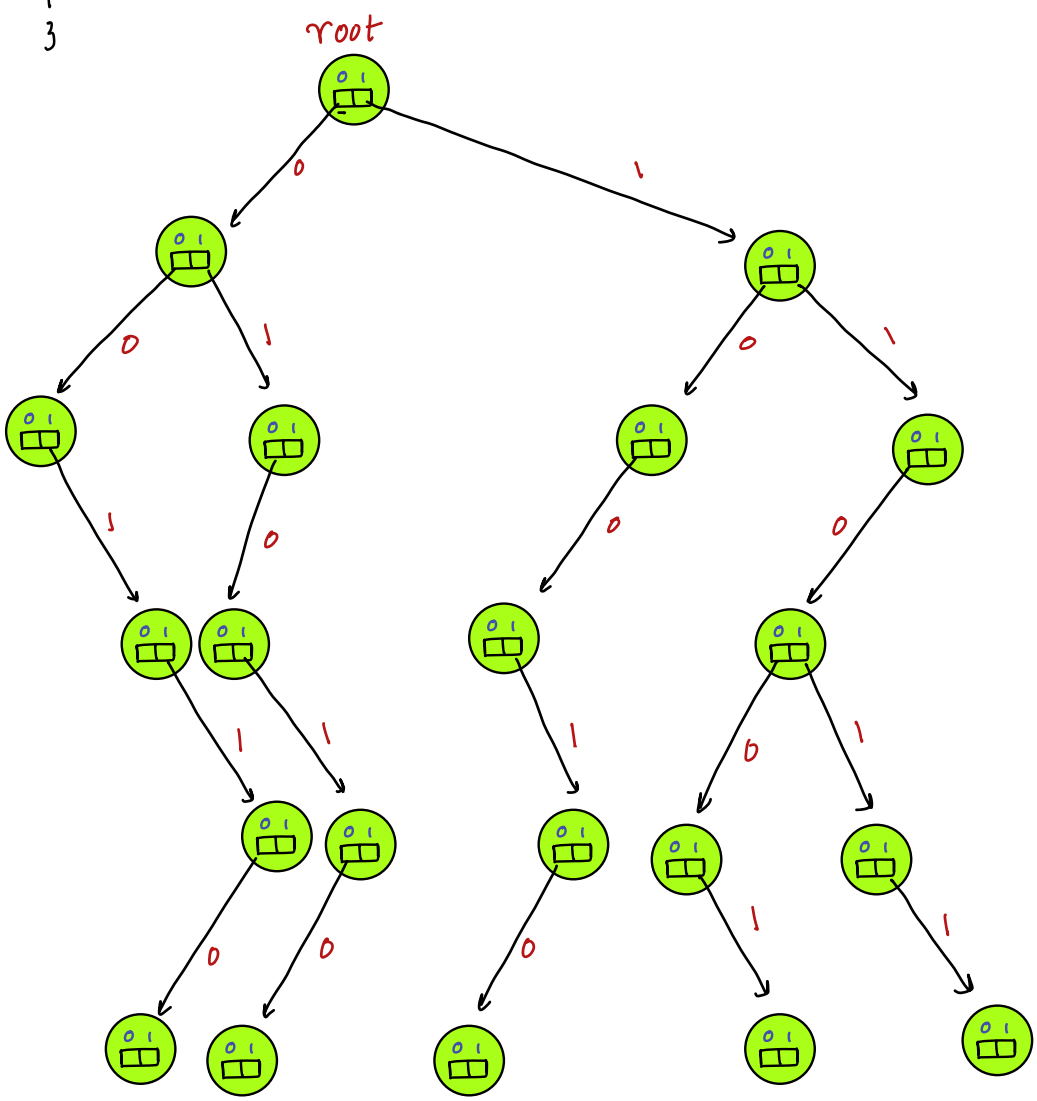
M ~ 10^5

Idea 4) → { When inserting a row in Trie, if we are not creating a single new node, { row is repeating }

Insert every row in a Trie

	16	8	4	2	1
0	1	0	0	1	0
1	1	1	0	1	1
2	0	1	0	1	0
3	1	1	0	1	1
4	1	1	0	0	1
5	1	0	0	1	0
6	0	0	1	1	0

```
class Node{ → // Binary Trie
    Node C[2]; ⇒ { data can only be 0/1 }
    Node C ) {
        C[0] = NULL
        C[1] = NULL
    }
}
```



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

```
int uniqueRows(int mat[][], int N, int M) {
    Node root = new Node();
    cnt = 0
    for(int i=0; i<N; i=i+1) {
        if(insert(root, mat[i], M)) {
            cnt = cnt+1
        }
    }
    return cnt;
}
```

will return if we create even a single new node

```
boolean insert(Node root, int ar[], int M) {
    bool flag = false;
    for i = 0; i<M; i=i+1) {
        // We want to insert ar[i]
        e = ar[i]
        if(root.c[e] == NULL) {
            // we need to create new node
            root.c[e] = new Node(); flag = True
            root = root.c[e]
        }
        else { root = root.c[e] }
    }
    return flag
}
```

even if we created a single new node

TC: N*M

SC: less than N*M

10:35pm → 10:43pm

3rd Given N array elements, find max xor value of a pair(i,j)

i != j i, j such that ar[i] ^ ar[j] is max

N=4 : 0 1 2 3 → ar[0]^ar[1] ar[0]^ar[2] ar[0]^ar[3]

4 3 2 7 → 7 6 3

ans = 7

ar[1]^ar[2] ar[1]^ar[3] ar[2]^ar[3]

1 4 5

2^7

min

Max

Idea1: (Check all pairs) → TC: O(N^2) SC: O(1)

App: Why?

1) Sliding Window - It's not a gurantee element have to continous

2) Sorting,

a) Compare adjacent elements → wont work

Ex: [2 3 7] → 5 ans = 5

1 4

b) xor with Max & Min → wont work

Ex: [4 3 2 7] → ans = 7

Hint: (A) (B) → more

Ex: 1 0 0 0 0 vs 0 1 1 1 1 A: is more

left → right we need to iterate to get bigger value

N=9 : 0 1 2 3 4 5 6 7 8

22 61 38 27 21 34 42 37 43

5 4 3 2 1 0

32 16 8 4 2 1

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

A = 22, get max A^B value, B should be present in given array elements

	5	4	3	2	1	0
A : 22 :	0	1	0	1	1	0
B : _ :	1	0	1	0	0	1
A^B :	1	1	1	1	0	1

If A = 22, max A^B = 61

Idea 2: If we do above approach for every element, it will work.

TC: N * { Every step we are checking if there is a number with required prefix : N^31 }

{ We can use trie to optimize this }

{ need to do it for each bit }

Inserting all elements in Binary Trie

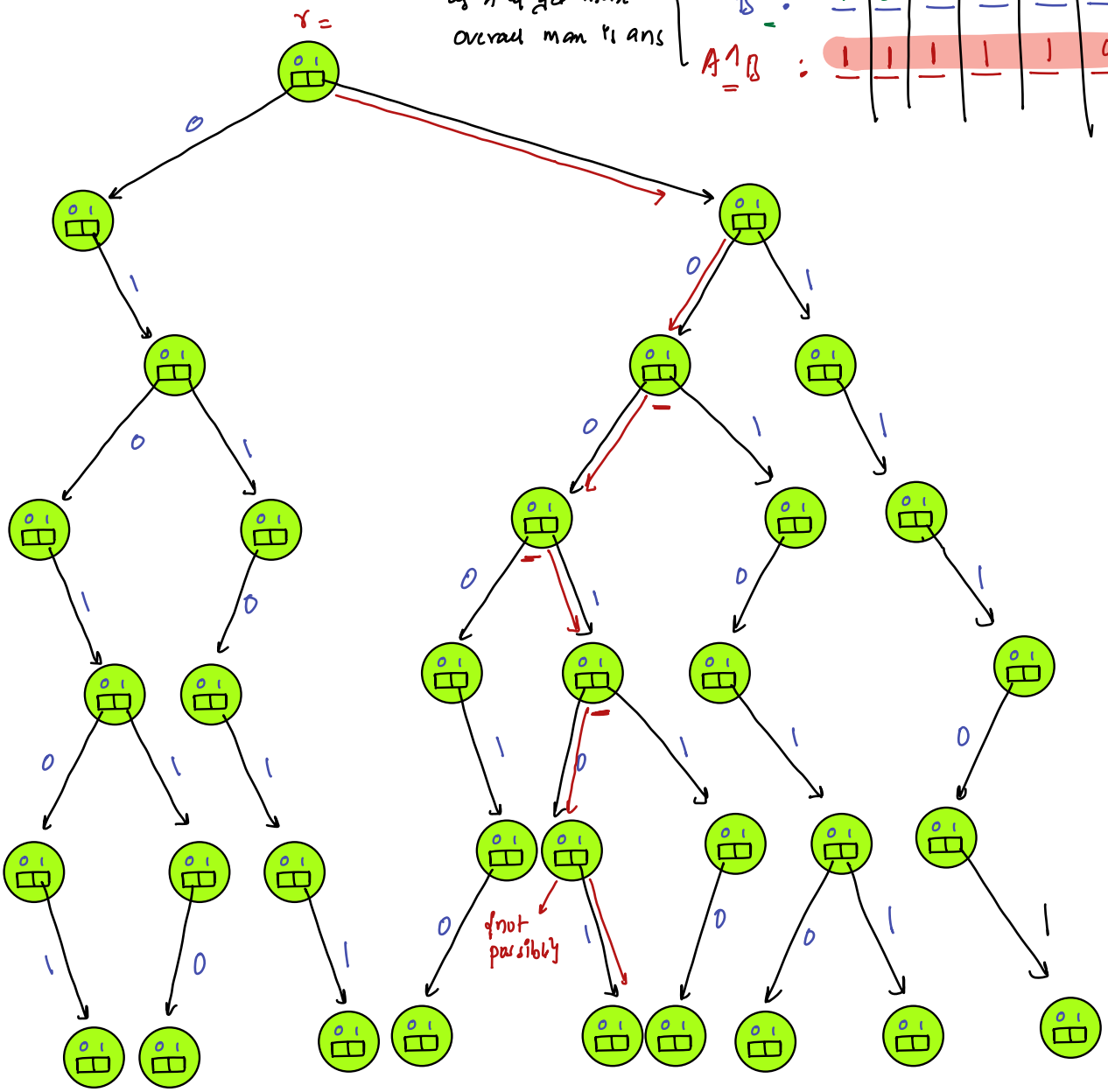
For every element of A & get max overall max pi ans

A : 27 :

B :

A ^ B :

	0	1	2	3	4	5
A	0	1	1	0	1	1
B	1	0	0	1	0	1
A^B	1	1	1	1	1	0



Note1: For all number we need to insert same number of bits

Note2: Get max of array & get max set bit pos

Ex: 10 → 1010 (3 2 1 0) → Max set pos 3

Ex: 17 → 10001 (4 3 2 1 0) → ans 4

Pseudocode:

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

int maxnr(int ar[], int N) {
    int mc = maxar(ar)        // max of ar
    int b = maxSetBit(mc)     // left most set bit pos

    Node root = new root();
    for(i=0; i<N; i=i+1) {
        insert(root, ar[i], b)
    }
    ans = 0
    // for every get max nr
    for(i=0; i<N; i=i+1) {
        ans = max(ans, query(root, ar[i], b))   // A = ar[i], max nor
    }
    return ans;   // max nr pair value in given array
}
```

TC: $(N*b + N*b)$

SC: $(N*b)$

N = no. of array elements

b = max set bit position

```
void insert(Node root, int ele, int b) {
    for(i=b; i>=0; i--) {
        // ith bit in ele
        int e = checkbit(ele, i)
        if(root.c[e] == NULL) {
            root.c[e] = new Node()
            root = root.c[e]
        }
        else {
            root = root.c[e]
        }
    }
}

int query(Node root, int ele, int b) {
    int ans = 0
    for(i=b; i>=0; i--) {
        // ith bit in ele
        int e = checkBit(ele, i);
        // if e -> 1 we need 0
        // if e -> 0 we need 1
        // for e -> search 1-e
        if(root.c[1-e] != NULL) {
            // 1-e is present
            // we can set ith bit = 1 in our ans
            ans = ans + (1<<i)[2^i]
            root = root.c[1-e]
        }
        else { root = root.c[e] }
    }
    return ans;
}
```