Min. No. of Software developers :

Date: 21st Jan 2022.

1. Minimum Number Of Software Developers
2. Number Of Valid Words
3. All Repeating Except One
4. All Repeating Except Two
5. One Repeating And One Missing
6. All Repeating Three Times Except One

Skills → a, b, c, d, e

0 dev. → a, c

1 dev. → b, d

2 dev. → a, b, c, d

3 dev → e

Min no. of dev. such that all skill set can be grasp from them.

Ex → we can grasp all skills.

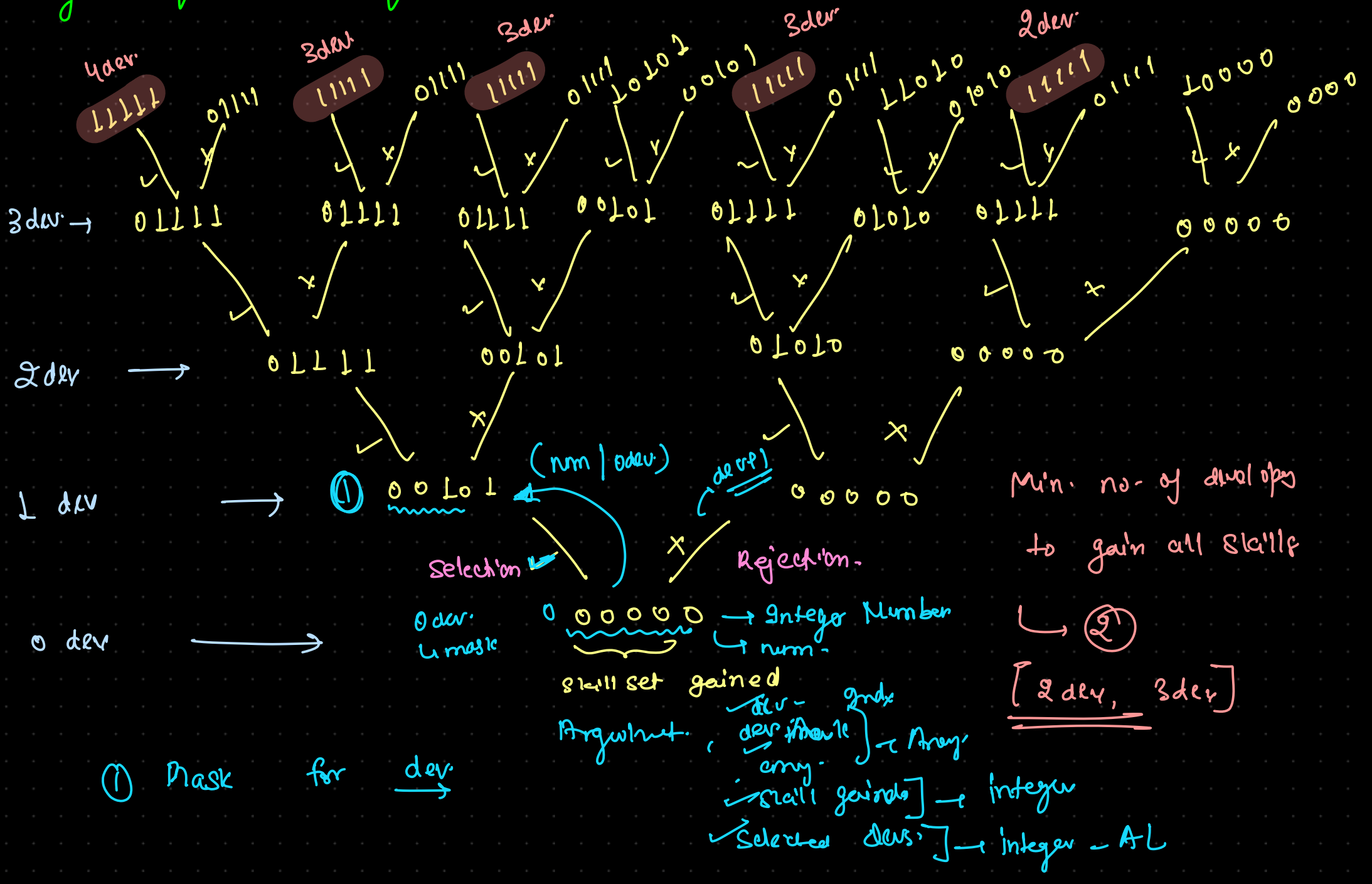
① Map skills with bit index.

e	d	c	b	a	← Skills
4	3	2	1	0	← bit index

② Mask for dev. with their skills

		e	d	c	b	a	
0 dev. →	a, c	0	0	1	0	1	these are bit mask
1 dev. →	b, d	0	1	0	1	0	
2 dev. →	a, b, c, d	0	1	1	1	1	
3 dev →	e	1	0	0	0	0	

Using Subsequence technique.



Min. no. of dual ops to gain all skills

↳ (2)

[2dev, 3dev]

(1) Mask for dev.

```
req_skills =
["algorithms","math","java","reactjs","csharp","aws"], people =
[["algorithms","math","java"],
["algorithms","math","reactjs"],
["java","csharp","aws"],
["reactjs","csharp"],["csharp","math"],
["aws","java"]]
```

dev.Mask → array Nums

Skill to bit map → Map

- algorithms → 0
- math → 1
- java → 2
- reactjs → 3
- csharp → 4
- aws → 5

dev	5	4	3	2	1	0	Integer Nos
0	0	0	0	1	1	1	=
1	0	0	1	0	1	1	
2	1	1	0	1	0	0	
3	0	1	1	0	0	0	
4	0	1	0	0	1	0	
5	1	0	0	1	0	0	

4 − 1 → $(11)_2$ → 3

100 → 4

111 → 7

1000 → 8

Ans = [ ]

0 0 1 1 1 1 1 ←

Right to left

First K bits ON (k = 5)

① Iterate and ON bit.

② (1 << 6) − 1 → O(1)

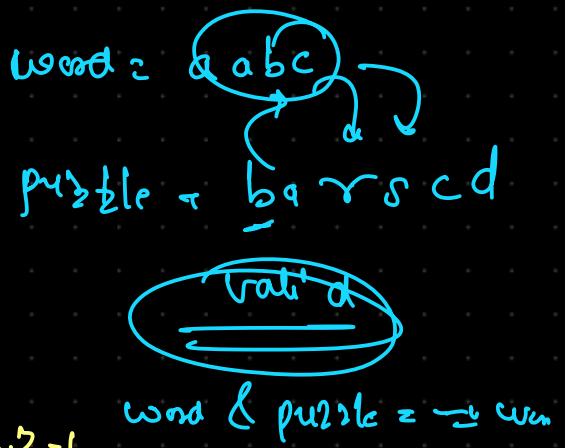
0 1 0 0 0 0 0 ← n+1

0 0 0 0 0 1

0 1 1 1 1 1 → n

No. of valid words :

7 → words
aaaa asas able ability actt actor access
6 → Puzzle
aboveyz abrodyz abslute absoryz actresz gaswxyz



For a given puzzle word is valid if

① word contains first letter of puzzle
② each letter of word should present in puzzle.

Puzzle ↓ / words →	aaa	asas	able	ability	actt	actor	access.
aboveyz → ①	✓	✗	✗	✗	✗	✗	✗
abrodyz → ①	✓	✗	✗	✗	✗	✗	✗
abslute → ③	✓	✓	✓	✗	✗	✗	✗
absoryz → ②	✓	✓	✗	✗	✗	✗	✗
actresz → ④	✓	✓	✗	✗	✓	✗	✓
gaswxyz → ①	✗	✗	✗	✗	✗	✗	✗

Suppose → a b c d e f g h i j

Puzzle

d g c j b

b g j i

a f i e

c g h f

HashMap< char, Something > → AL<Integer>

a → hiba, jbaf
b → bcgi, hiba, jbaf
c → bcgi, cegh
d → dihj
e → cegh
f → jbaf
g → bcgi, cegh
h → dihj, hiba, cegh
i → bcgi, dihj, hiba
j → dihj, jbaf

How to store this:

bits

	h	g	e	d	c	b	a
4							
2							
1	0	1	0	1	0	1	0

words

bcgi
dihj
hiba
cegh
jbaf

Mab change to mask

a → hiba, jbaf
b → bcgi, hiba, jbaf
c → bcgi, cegh
d → dihj
e → cegh
f → jbaf
g → bcgi, cegh
h → dihj, hiba, cegh
i → bcgi, dihj, hiba
j → dihj, jbaf

change to num

d g c j b → num1 → num1 → dihj

b g j i → num2 → bcgi, hiba, jbaf

a f i e → num3 → hiba, jbaf

c g h f → m4 → bcgi, cegh

puzzle

num[i] → bit mask having bit one ON on character index

h g f e d c b a

cghf → 0 1 1 1 0 0 1 0 0

h g f e d c b a

0 1 1 0 1 0 1 0 0

How to check if x have same ON bit as y. ??

→ may be 011010100 x have more ON bit

AND

→ ((y&x) == y) → same bit one ON

→ all character of y are present in x

① Map < Character , AL < Integer >>

char — bit mask of word

```
HashMap<Character, ArrayList<Integer>> map = new
for(int i = 0; i < 26; i++) {
    char ch = (char)(i + 'a');
    map.put(ch, new ArrayList<>());
}

for(String word : words) {
    for(int i = 0; i < word.length(); i++) {
        char ch = word.charAt(i);
        map.put(ch, word);
    }
}
```

a a a a

a → aaaa , aaaa , aaaa

↳ to avoid this problem

↳ HashSet<Char>

a → [ ]

Output for this section

Map change to mask

a → hiba , jbaf

b → bcgi , hiba , jbaf

c → bcgi , cegh

d → dihj

e → cegh

f → jbaf

g → bcgi , cegh

h → dihj , hiba , cegh

i → bcgi , dihj , hiba

j → dihj , jbaf

word are present in terms of mask

```java
// map of char vs bit mask of word
HashMap<Character, ArrayList<Integer>> map = new HashMap<>();
for(int i = 0; i < 26; i++) {
    char ch = (char)(i + 'a');
    map.put(ch, new ArrayList<>());
}

for(String word : words) {
    HashSet<Character> set = new HashSet<>();
    // prepare mask for word
    int mask = 0;
    for(int i = 0; i < word.length(); i++) {
        char ch = word.charAt(i);
        int k = (int)(ch - 'a');
        mask = (mask | (1 << k));
    }

    for(int i = 0; i < word.length(); i++) {
        char ch = word.charAt(i);
        if(set.contains(ch)) continue;
        set.add(ch);
        map.get(ch).add(mask);
    }
}
```

a b c d e f g

Assumption :- suppose Alphabets are

puzzle

word → aaa aab deb fgb gac

bcfg m1 → 3, 26, 98

cbea m2

deaa m3

fcab m4

fged m5

gacb m6

& m1 == 3

count ++

Add in map corresponding to letter

a → [1] [3] [6 9]

b → [3] [26] [98]

c → [6 9]

d → [26]

e → [26]

f → [98]

g → [98] [69]

Return counts.

aaa → (0 0 0 0 0 1)₂ = (1)₁₀

aab → (0 0 0 1 1)₂ = (3)

fgb = (1 1 0 0 0 1 0)₂ = 98

```java
// prepare count
ArrayList<Integer> counts = new ArrayList<>();
for(String puzzle : puzzles) {
    // prepare mask of puzzle
    int mask = 0;
    for(int i = 0; i < puzzle.length(); i++) {
        char ch = puzzle.charAt(i);
        int k = (int)(ch - 'a');
        mask = (mask | (1 << k));
    }
    // check in valid word and find count
    char firstChar = puzzle.charAt(0);
    int count = 0;
    for(int wordMask : map.get(firstChar)) {
        if((wordMask & mask) == wordMask) {
            count++;
        }
    }
    counts.add(count);
}
return counts;
```

gac 64 + 4 + 1 = 69

1 0 0 0 1 0 1

deb → g f e d c b a

(0 1 1 0 1 0)₂ = (26)₁₀

16 + 8 + 2

mask 22 64 + 2 = 98