Pangram Check

Problem Description

Given a sentence represented as an array **A** of strings that contains all lowercase alphabets.
Chech if it is a **pangram** or not.
A pangram is a unique sentence in which every letter of the lowercase alphabet is used at least once.

Problem Constraints

1 <= |A| <= 10^5
1 <= $|A_i|$<= 5

Example Input

Input 1:

```
A = ["the", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog"]
```

Input 2:

```
A = ["bit", "scale"]
```

Example Output

Output 1:

```
1
```

Output 2:

```
0
```

```
vst[26];
∀i , vst[i] = false
for i → 0 to (N-1) {
    l = A[i].length()
    for j → 0 to (l-1) {
        ch = A[i][j]        // A[i].charAt(j)
        vst[ch - 'a'] = true
    }
```

}

for i → 0 to 25 {   // a → z

if (!vst[i]) return 0

}

return 1

TC = $\underline{O(N)}$ SC = O(26) = $\underline{O(1)}$

Sort by Color

Problem Description

Given an array with **N** objects colored **red, white, or blue**, sort them so that objects of the same color are adjacent, with the colors in the **order red, white, and blue.**

We will represent the colors as,

```
red -> 0
white -> 1
blue -> 2
```

Note: Using the library sort function is not allowed.

Problem Constraints

1 <= N <= 1000000
0 <= A[i] <= 2

A = [0 1 2 1 0 0 2]

↳ 0 0 0 1 1 2 2 (Ans)

<u>count Sort</u>

cnt[3] = {0, 0, 0}

for i → 0 to (N-1) {

cnt[A[i]]++

}

```
k = 0
for i → 0 to 2 {
    for j → 1 to cnt[i] {
        A[k] = i
        k++
    }
} return A
```

TC = O(N)   SC = O(1)

Minimum Cost with Non-Skippable Staircase

Problem Description

You are given an integer array **A** of length **N**, where **A[i]** represents the cost of the **i-th** stair on a staircase.

Once you pay the cost, you can either climb one or two steps. You have the option to start from either the **0th** index stair or the **1st** index stair.

However, there is a twist: there is an additional integer **B**, representing a specific stair that you cannot skip while climbing. Your task is to find the minimum cost to reach the top of the staircase while ensuring that you cannot skip the **B-th** stair.

Note : Top of the floor means reaching the Nth stair.

Problem Constraints

2 <= **N** <= 1000

0 <= **B** < **N**

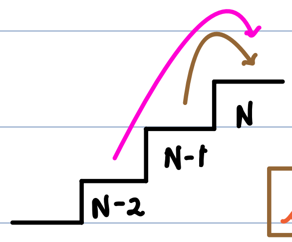
0 <= **A[i]** <= 999

$$\begin{array}{cccccc} & 0 & 1 & 2 & 3 & 4 \\ A = [& 1 & 3 & 2 & 4 & 5 \ ] \\ C \rightarrow & 1 & 3 & 3 & 7 & 8 \end{array}$$

$B = 2$ $\quad$ $Ans = 1 + 2 + 4 = \underline{7}$

$$\begin{array}{cccccc} & 0 & 1 & 2 & 3 & 4 \\ A = [& 1 & 3 & 2 & 4 & 5 \ ] \end{array}$$

$B = 3$ $\quad$ $Ans = 3 + 4 = \underline{7}$



#ways (N) = # ways (N-1) + # ways (N-2)

$$\boxed{cost[i] = A[i] + \min(cost[i-1], cost[i-2])}$$

cost [0] = A[0]

cost [1] = A[1]

$$\overbrace{\{0, 1\} \longrightarrow \{B\}}^{1} \overbrace{\longrightarrow \{N\}}^{2}$$

```
cost [0] = A [0]            cost [1] = A [1]

for i → 2 to B {
|     cost [i] = A[i] + min (cost [i-1], cost [i-2])
}

if (B == N-1 || B = N-2) return cost [B]

cost [B+1] = A [B+1] + cost [B]

for i → (B+2) to (N-1) {
|     cost [i] = A[i] + min (cost [i-1], cost [i-2])
} return min (cost [N-1], cost [N-2])
```

index out of bound for $\underline{B=1}$.

$TC = \underline{O(N)}$ $\qquad$ $SC = \underline{O(N)}$

Painter's Partition Problem

Problem Description

Given 2 integers **A** and **B** and an array of integers **C** of size **N**. Element **C[i]** represents the length of i^{th} board.
You have to paint all **N** boards **[C_0, C_1, C_2, C_3 … C_{N-1}]**. There are **A** painters available and each of them takes **B** units of time to paint **1 unit** of the board.

Calculate and return the minimum time required to paint all boards under the constraints that **any painter will only paint contiguous sections of the board.**
NOTE:
1. 2 painters cannot share a board to paint. That is to say, a board cannot be painted partially by one painter, and partially by another.
2. A painter will only paint contiguous boards. This means a configuration where painter 1 paints boards 1 and 3 but not 2 is invalid.

Return the **ans % 10000003**.

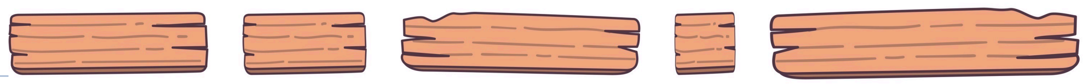
Problem Constraints

1 <= A <= 1000
1 <= B <= 10^6
1 <= N <= 10^5
1 <= C[i] <= 10^6

$\rightarrow 10^6 * 10^6 = \underline{10}^{12}$ (long)

P_1 P_2

A = 2



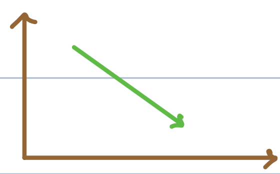
B = 2

	5	3	6	1	9 ✓
C[i] * B →	10	6	12	2	18

Ans = max (10 + 6 + 12 , 2 + 18) = max (28, 20) = $\underline{28}$

# painters (A)	min time to paint all boards
1	10 + 6 + 12 + 2 + 18 = 48
2	max (10 + 6 + 12, 2 + 18) = $\underline{28}$
3	max (10 + 6, 12 + 2, 18) = $\underline{18}$

A ∝ 1/minTime



→ BS on answer

$\forall i$, C[i] = C[i] * B ✗

int

overflow

```
int minPainters (C[], L) {
    cnt = 1      l = L
    for i -> 0 to (N-1) {
        if (C[i] > L)  return Int_Max
        if (C[i] <= l)  l -= C[i]
        else {  cnt++
                l = L - C[i] }
    } return cnt
}
```

$l = \max\limits_{\forall i}(C[i]) \qquad r = \sum\limits_{\forall i} C[i]$

```
while (l <= r) {
    m = l + (r-l)/2    // length
    cnt = minPainters (C, m)
    if ( cnt <= A  &&  minPainters (C, m-1) > A ) {
            return  (1L * m * B) % (10^7+3)
    }
    if (cnt > A)    l = m+1
    else      r = m-1
}
```

TC = $O(N \log(\Sigma C[i]))$ SC = $O(1)$

Distance of nearest cell

Problem Description

Given a matrix of integers **A** of size **N x M** consisting of **0** or **1**.

For each cell of the matrix find the distance of nearest 1 in the matrix.

Distance between two cells **(x1, y1)** and **(x2, y2)** is defined as **|x1 - x2|** + **|y1 - y2|**.

Find and return a matrix **B** of size **N x M** which defines for each cell in A distance of nearest **1** in the matrix A.

NOTE: There is atleast one 1 is present in the matrix.

Problem Constraints

1 <= N, M <= 1000

0 <= A[i][j] <= 1

Example Input

```
Input 1:

  A = [
        [0, 0, 0, 1]
        [0, 0, 1, 1]
        [0, 1, 1, 0]
      ]

Input 2:

  A = [
        [1, 0, 0]
        [0, 0, 0]
        [0, 0, 0]
      ]
```

Example Output

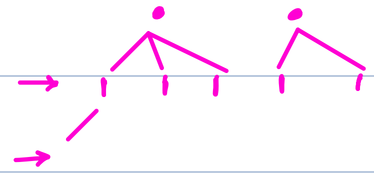
```
Output 1:

  [
    [3, 2, 1, 0]
    [2, 1, 0, 0]
    [1, 0, 0, 1]
  ]

Output 2:

  [
    [0, 1, 2]
    [1, 2, 3]
    [2, 3, 4]
  ]
```

Multisource BFS

for i → 0 to (N-1) {
  for j → 0 to (M-1) {
    d[i][j] = -1
    if (A[i][j] == 1) { q.enqueue ({i,j})
        d[i][j] = 0 }
  }

}

```
dx = {-1  0  0  1}
dy = {0  -1  1  0}
while (!q.isEmpty()) {
    x, y = q.dequeue()
    for i → 0 to 3 {
        u = x + dx[i]
        v = y + dy[i]
        if (0 <= u && u < N &&
            0 <= v && v < M && d[u][v] == -1) {
                q.enqueue({u, v})
                d[u][v] = d[x][y] + 1
        }
    }
} return d
```



$TC = \underline{O(N*M)}$ $SC = \underline{O(N)}$