Polls

- Rate the difficulty level of contest

  Easy

  Tricky

  Pissed off

Painter's Partition Problem

Flag Question

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

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

example

A = 2 B = 1 C =

1	2	3	4	5

ans = 9

least time to complete painting walls = max(C) * B

Max time to complete painting = sum(C) * B

Brute force

```
for (i = least ; i <= max ; i++) {
    // check if it is possible to paint all
    // boards in i time, using A painters
}
```

Observation

(Binary search on Ans)

```
// check if it is possible to paint all
// boards in mid time, using A painters
```

B = 1 c = | 1 | 2 | 3 | 4 | 5 | A = 2

least = 5 * 1 = 5 Max = 15 * 1 = 15

least	Max	mid	required Painter
5	15	10 (PA)	2
5	9	7	3
8	9	8	3
9	9	9 (PA)	2

ans = 9

B = 1 C = | 1 | 2 | 3 | 4 | 5 | A = 2

pseudo code

```
lo = max(C) * B        hi = sum(C) * B
while (lo <= hi) {
    mid = (lo + hi) / 2;
    // external fn
    painters Req = getReqPainters (C, B, mid)
    if (painters Req <= A)
        ans = mid % Mod;
        high = mid - 1;
    else
        low = mid + 1;
}
```

```
Int getRequiredPainter (C, B, mid) {
    total = 0;  reqP = 1;
    for (i = 0; i < C.length(); i++) {
        total += C[i] * B
        if (total > mid)
            total = C[i] * B;
            reqP += 1;
    }
    return reqP;
}
```

T.C = N * log(hi - lo)

Pangram Check

Flag Question

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

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

```
A = [ "My", "name", "is", "Dhiraj" ]
freq = [0] * 26
for (i=0; i< A.length; i++) {
    word = A[i];
    for (ch in word) {
        ind = ch - 'a';
        freq[ind] += 1;
    }
}

for ( i=0; i<26; i++)
    if (freq[i] < 1)  return 0;
return 1;
```

Minimum Cost with Non-Skippable Staircase

Flag Question

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.

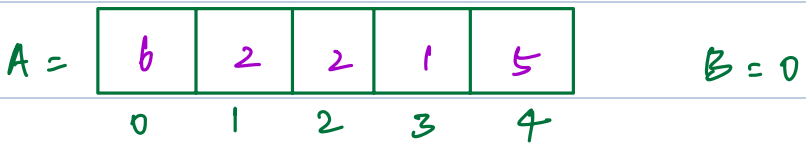
Problem Constraints

2 <= **N** <= 1000

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

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

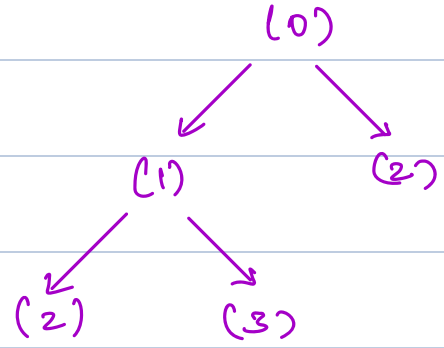
example



Observation

DP state :

DP [i] ⟶ i^{th} stair



B^{th} step

0^{th} index 1^{st} index middle last

DP states in a 1D array

DP [i]

⟶ min cost to reach i^{th} step starting from 0 or 1

DP [0] = A [0]

if (B == 0) DP [1] = A [0] + A [1]

else DP [1] = A [1]

```
for ( i=2; i<n; i++) {
    if (i-1) == B  // cannot skip i-1
        DP[i] = DP[i-1] + A[i];
    else
        DP[i] = A[i] + min (DP[i-1]
                , DP[i-2])
}
return DP[n-1];
```

Distance of nearest cell

Flag Question

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

A:-

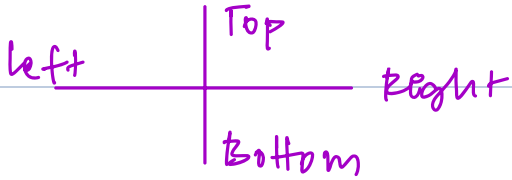
0	0	0	1
0	1	0	0
0	0	0	0
0	0	0	1

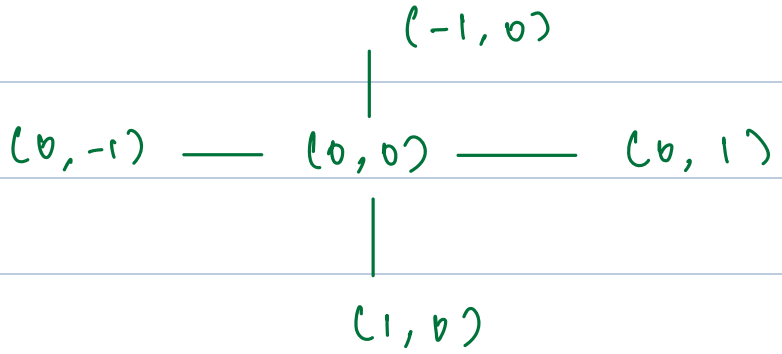
			0
	0		
			0

Observation

C Implementation of multisource BFS]

neighbours





// all neighbours is 1 distance away

Dry Run

	0	1	2	3
0	2	1	1	0
1	1	0	1	1
2	2	1		1
3			1	0

~~(0,3)~~ ~~(1,1)~~ ~~(3,3)~~

~~(0,2)~~ ~~(1,3)~~ ~~(0,1)~~ ~~(1,0)~~

(1,2) (2,1) (3,2) (2,3)

(0,0) (2,0)

First: add all sources to my Queue

Second: Add all its unvisited neighbours to the queue & perform BFS

pseudo code

```
dx = [0, 1, 0, -1]
dy = [1, 0, -1, 0]

for (i=0; i<n; i++) {
    for (j=0; j<m; j++) {
        if (A[i][j] == 1)  // source
            q.append ([i, j]);
        ans[i][j] = 0;
    }
}

while (!q.isEmpty()) {
    x, y = q.dequeue();
    for (i=0; i<4; i++) {
        nx = x + dx[i];
        ny = y + dy[i];
        if (nx >= 0 && nx < n &&
            ny >= 0 && ny < m &&
            ans[nx][ny] > ans[x][y]+1)
            ans[nx][ny] = ans[x][y]+1
            q.append (nx, ny)
    }
}
```

Sort by Color

Flag Question

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.

example

{ 0, 1, 2, 0, 1, 2, 0, 0, 2, 2 }

hm[0] = 4 hm[2] = 4

hm[1] = 2

ans = { 0, 0, 0, 0, 1, 1, 2, 2, 2, 2 }

#pseudo code

```
arr [3] = [0, 0, 0]
for (i=0; i < A.length; i++) {
    arr [A[i]] += 1;
}
```

```
for ( i=0; i<3; i++) {
    for (j=0; j< arr[i]; j++) {
        ans.append (i);
    }
}
```

T.C = O(n)