Stack:
- LIFO
- Insert at top and delete at top

Insert at top	push()
Remove from top	pop()
Return number of elements in stack	size()
Check empty	empty()
Returns a reference to the newest element in the stack	top()
STL	#include <stack>

Queue:
- FIFO

Insert at back	push()
Remove from front	pop()
Return the first element	front()
Return the last element	back()
Return number of elements in queue	size()
Check empty	empty()
Exchange two values in queue with same data types	swap()
STL	#include <queue>

String:
- the last character is '\0'

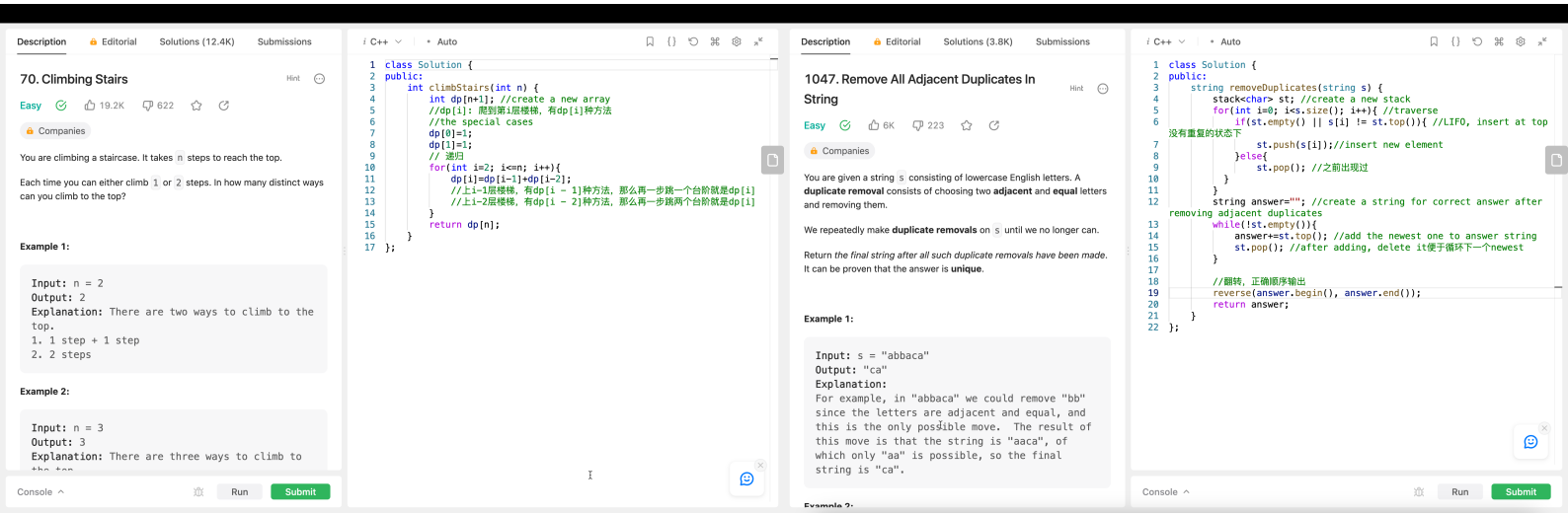
Remove the last element from a vector	pop_back()
Append each character to a string object	push_back()
Make the vector empty	clear()
Check whether it is empty	empty()
Erase character from string	erase()
Swap string values	swap()
Find the content in string	find()
Return the max size of string	max_size()
Get the character with a given position in string	at()
Assign content to string	assign()
STL	#include <string>

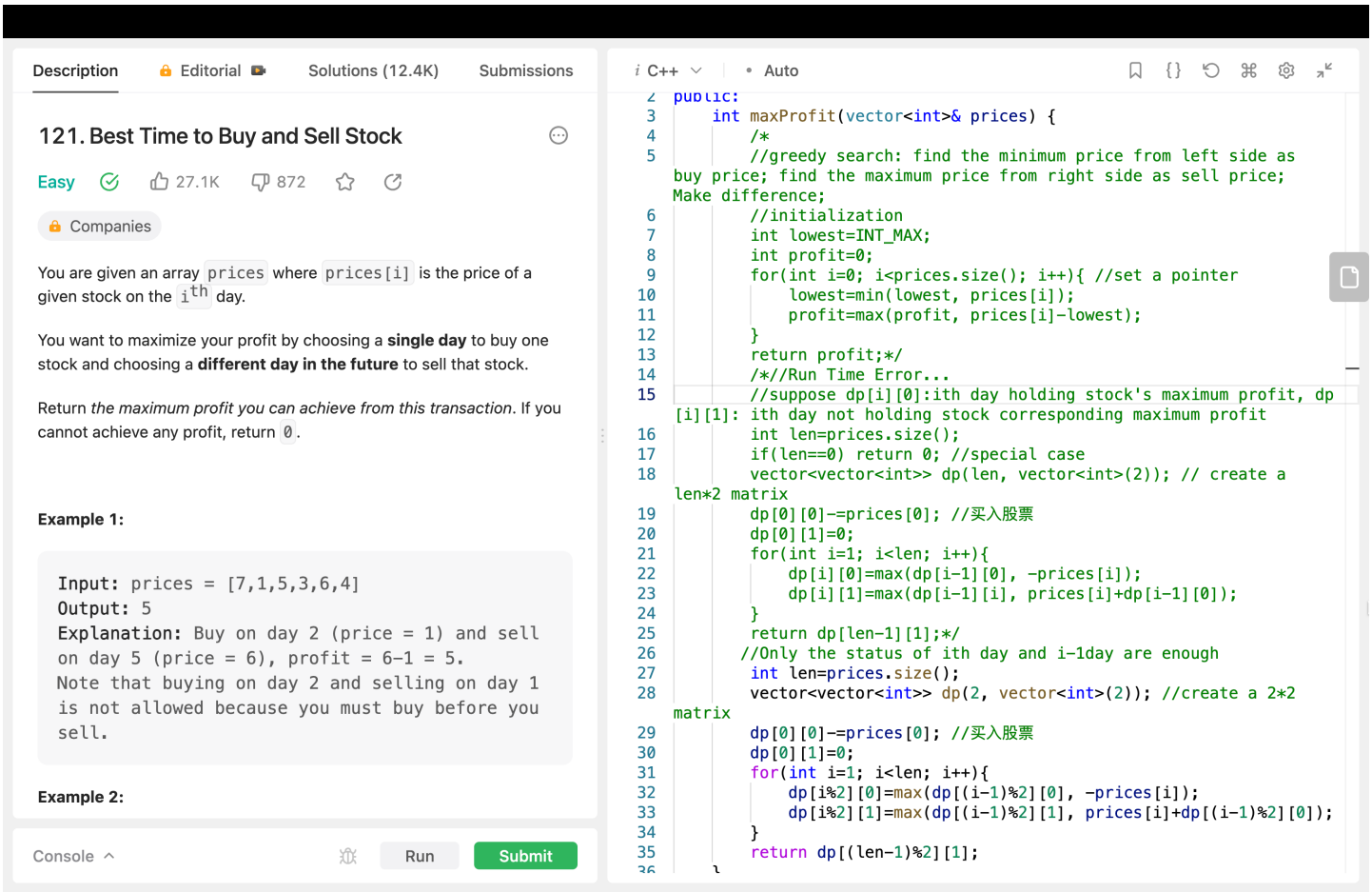
Dynamic Programming:

- create a new array/table
- Initialized by filling with 0, with a specific size n, m
- Store the calculated value, recalculated is not needed anymore, extract at the corresponding index is the only need

Array:

- can be referenced by index, starting from 0
- Initialized with a given size is a must, but a initialized value is not an obligation(it may randomly filled with garbage value)
- Notation: arrayName[element]
- sizeof()





3. Document Chunking

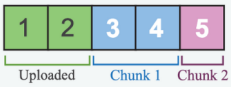
A financial services company is uploading documents to a compliance system for analysis. They use a chunking mechanism as below.

- Each document is divided into equal sized packets.
- Documents are then divided and uploaded in "chunks" of packets. A chunk is defined as a contiguous collection of 2^n packets, where n is any integer ≥ 0.
- After the document is divided into chunks, randomly selected chunks are uploaded until the entire document is completely uploaded.

There is one document that is partially uploaded, described in *uploadedChunks*. Determine the minimum number of chunks that are yet to be uploaded.

Example
totalPackets = 5
n = 2 number of chunks already uploaded
uploadedChunks = [[1, 2]]

- The document has *5* packets and 1 chunk of $2^1 = 2$ *packets [1, 2]* is already uploaded.
- The remaining *3* packets are uploaded in *2* chunks. The length of chunk1 is $2^1 = 2$, packets [3, 4], and the length of chunk2 is $2^0 = 1$ *packet [5].*



Function Description
Complete the *minimumChunksRequired* function in the editor below.

minimumChunksRequired has the following parameter(s):
long int totalPackets: the number of packets in the document.
long int uploadedChunks[n][2]: each *uploadedChunks[i]* describes the start and end packet numbers of the uploaded chunks.
Returns
int: an integer that denotes the minimum number of chunks that need to be uploaded.

Constraints

- $1 \le totalPackets < 10^{18}$
- $0 \le uploaded < 10^5$
- The uploaded chunks do not overlap
- $1 \le$ starting packet $\le$ ending packet $\le totalPackets$

2. K Smallest Substring

There is a string *input_str* consisting of characters '0' and '1' only and an integer *k*. Find a substring of string *input_str* such that:

- The number of '1's is equal to *k*
- It has the smallest length
- It is lexicographically smallest

Note: It is guaranteed that answer always exists.

Example
input_str = "0101101"
k = 3

Some of the possible substrings following the first condition:

- *"01011"*
- *"1101"*
- *"1011"*

The substring that is smallest in length and lexicographically smallest is *"1011"*.

It can be proven that there is no other substring that is smaller than *"1011"* in length and lexicographic order. Hence the answer is *"1011"*.

Function Description
Complete the function *getSubstring* in the editor below.

getSubstring has the following parameters:
string input_str: a string that consists of '0' and '1'
int k: the number of '1's in the answer

Returns

1. Match Outcomes

Given the initial setup of a match between two players, evaluate the match's outcome.

There are two players, and there is a number sequence of size *n*. Players alternate turns for *n* rounds. Each round, a player removes the first number from the sequence and adds its value to their score. After that, if the 'removed' number is even, the remaining sequence is reversed. Determine the difference in scores between the two players after the game.

More precisely, suppose *first_score* and *second_score* are *the final scores of the first and second player, respectively.* The goal is to calculate the value of *first_score - second_score.*

Example

The number of elements is *n = 5* and *numSeq = [3, 6, 2, 3, 5].*

- 1^{st} round: The first player picks 3, *first_score* = 3. The remaining sequence: [6, 2, 3, 5].
- 2^{nd} round: The second player picks 6, *second_score* = 6. Since 6 is even, the remaining sequence is reversed: [5, 3, 2].
- 3^{rd} round: The first player picks 5, *first_score = 3 + 5 = 8.* The remaining sequence: [3, 2].
- 4^{th} round: Second player picks 3, *second_score = 6 + 3 = 9.* The remaining sequence: [2].
- 5^{th} round (final): First player picks 2, *second_score = 8 + 2 = 10.* The remaining sequence: [].

The total difference between players' scores is *first_score - second_score = 10 - 9 = 1.*

Function Description
Complete the function *getScoreDifference* in the editor.

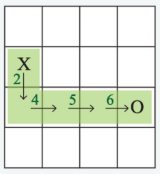
getScoreDifference has the following parameter:
numSeq: the given array of integers

Returns
int: first_score - second_score

1. Laser Grid

A player stands on a cell within a grid. The player can move to one of four adjacent cells, but the motion is constrained by lasers. To move from one position to another involves a cost: the cost to move from row *i* to row $i \pm 1$ is *costRows[i]* and the cost to move from column *j* to column $j \pm 1$ is *costCols[j].* Find the minimum cost to move from a starting point to an ending point within the grid.

Example
rows = 4
cols = 4
initR = 1
initC = 0
finalR = 2
finalC = 3
costRows = [1, 2, 3]
costCols = [4, 5, 6]



The player starts at position *(row, col) = (1, 0)* (marked with an X) and must reach position *(2, 3)* (marked with an O).

- To move from *row 1* from *row 2*, the cost is *costRows[1] = 2.*
- To move from column *0* to column *3*, the player must move to column *1*, then *2*, then *3* at a cost of *costCols[0] + costCols[1] + costCols[2] = 4 + 5 + 6 = 15.*
- The total cost is *2 + 15 = 17.*

2. Alternating Prime Sequence

A student needs to construct an *alternating prime sequence* from an array of integers for a coding challenge. If an integer cannot be used, its value is added to a penalty accumulator.

An alternating prime sequence is a sequence where adjacent elements alternate between prime and non-prime. For example, [1, 2, 4, 3, 6, 5] is a valid alternating prime sequence while [1, 2, 3, 4, 5, 6] is not as 2 and 3 are both primes and they are adjacent. Find the minimum possible penalty that the student can have after completing the task.

Example

Consider, for example, a sequence *arr* = [3, 7, 1, 4, 6, 6] of length *n = 6*. The student can construct a sequence [4, 3, 6, 7, 6] which has a penalty of 1. Note that there are other possible sequences as well, but 1 is the minimum penalty.

Function Description

Complete the function *minPenalty* in the editor below.

minPenalty has the following parameter(s):
int arr[n]: an array of integers

Returns

int: the minimum penalty value

Constraints

- $1 \leq n \leq 10^4$
- $1 \leq arr[i] \leq 10^4$

Problem | My Submissions | All Submissions | Discussion

Maximum MEX :

Author
Shikhar Mehrotra

Difficulty Level : Medium
Marks :20
Submissions : 68
: 3 | : 0
Asked In : Atlassian

Given an array containing N non negative integers and an element X , In one operation, X can be added to or subtracted from any element of the array.MEX of an array is defined as the **smallest non negative integer which is not present in the array**.

for example, the MEX of [0,1,1,3] is 2, and the MEX of [1,2,4] is 0.

Find the maximum possible MEX of the array that can be achieved by doing the above operation any number of times.

Input

The first line contains a single integer $1 \leq N \leq 10^5$ denoting size of the array.

The second line contains N space separated integer denoting elements of the array $0 \leqslant a[i] \leqslant 10^9$ where $(0 \leqslant i < n)$.

Third line contains a single integer X where $(1 \leqslant X \leqslant 10^5)$

Output

Output a single integer denoting the maximum MEX possible by applying operations any number of times.

Examples

input
3 1 3 4 2
output
2

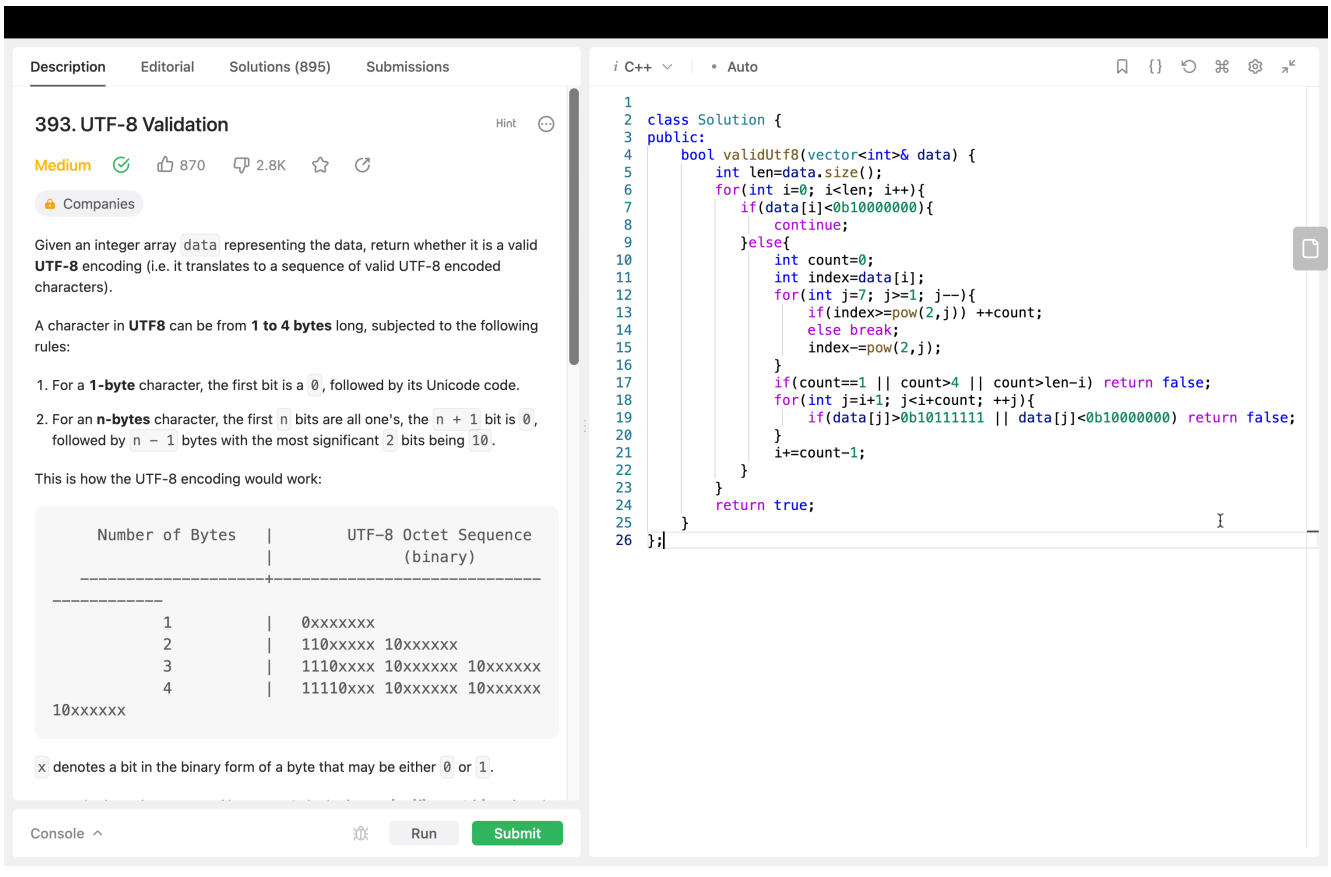
input
5 0 1 2 1 3

3. Statistic Indicators

Given an array, find two statistic indicators for the array and report the difference between the two stats. The indicators are defined as follows:

- *Indicator 1*: This is defined by the number of instances where a number *k* appears for **exactly** *k* consecutive times in the array. For example, in the array [1, 2, 2, 2, 2, 3, 3, 3, 1, 1, 2, 2], we have 1(1), 2(4), 3(3), 1(2), 2(2) where the value in the braces represents the number of times the integer appeared consecutively. The value of indicator 1 for the given array would be 3 corresponding to 1(1), 3(3) and 2(2).
- *Indicator 2*: This is defined by the number of instances where a number *k* appears for **exactly** *k* consecutive times in the array, starting from index *k* assuming 1-based indexing. For example, in the array [2, 2, 2, 4, 4, 4, 4, 4, 4], if we start at index 2 we have exactly 2 consecutive 2s coming up. While when we start at index 4, we have 6 consecutive 4s coming up. Hence the value of indicator 2 for the array would be 1 corresponding to the index 2.

Your task is to find the absolute difference between the two indicators.



Leetcode 1319:
DFS:
Search in a specific direction continuously, if it cannot search any more, then change another direction.
While, BFS is search all connected nodes of current node for each node.
Application:

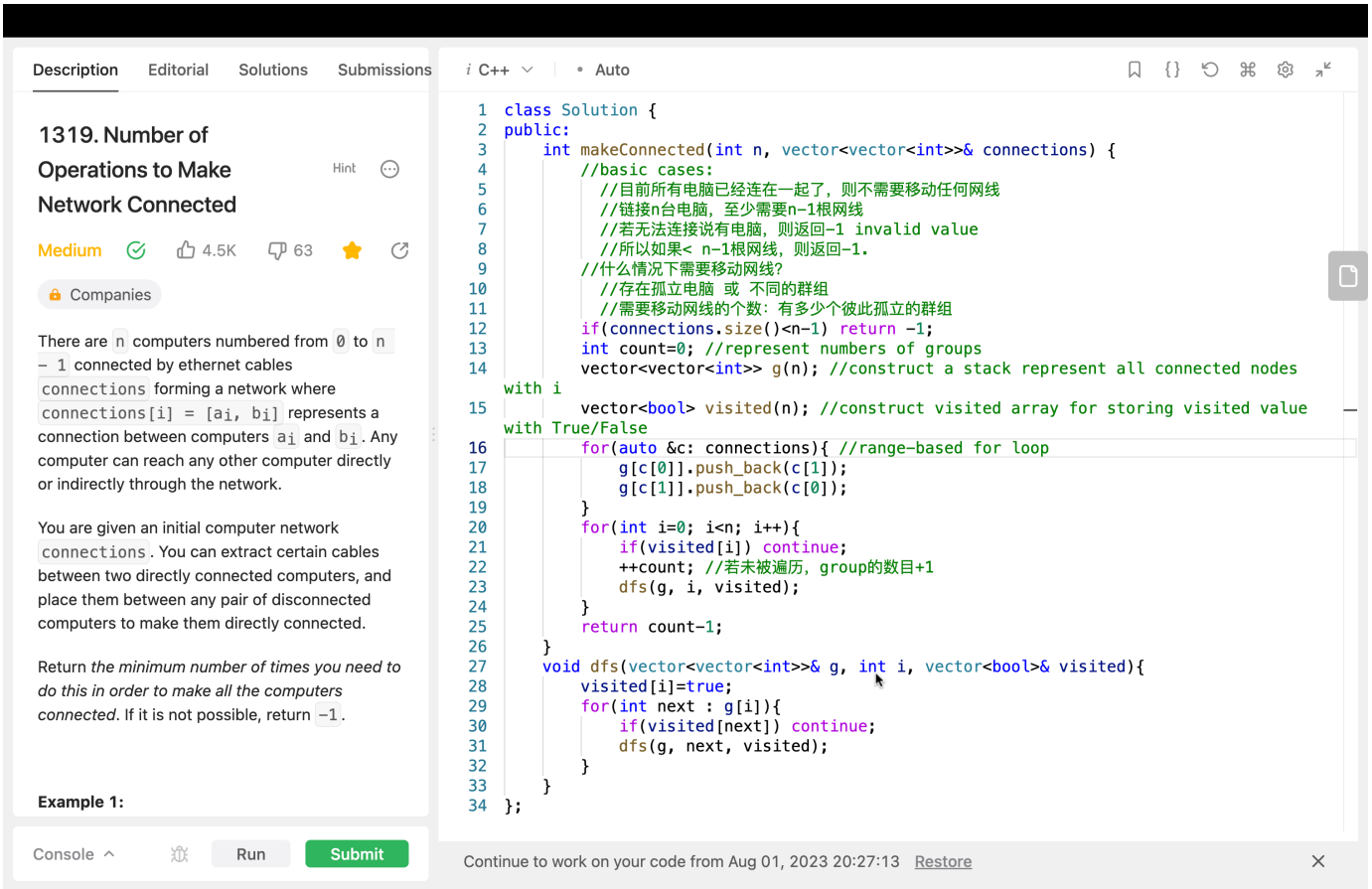
1. Detect cycles in a graph
2. Find the strongly connected components of a graph
3. Find a path

Solution:

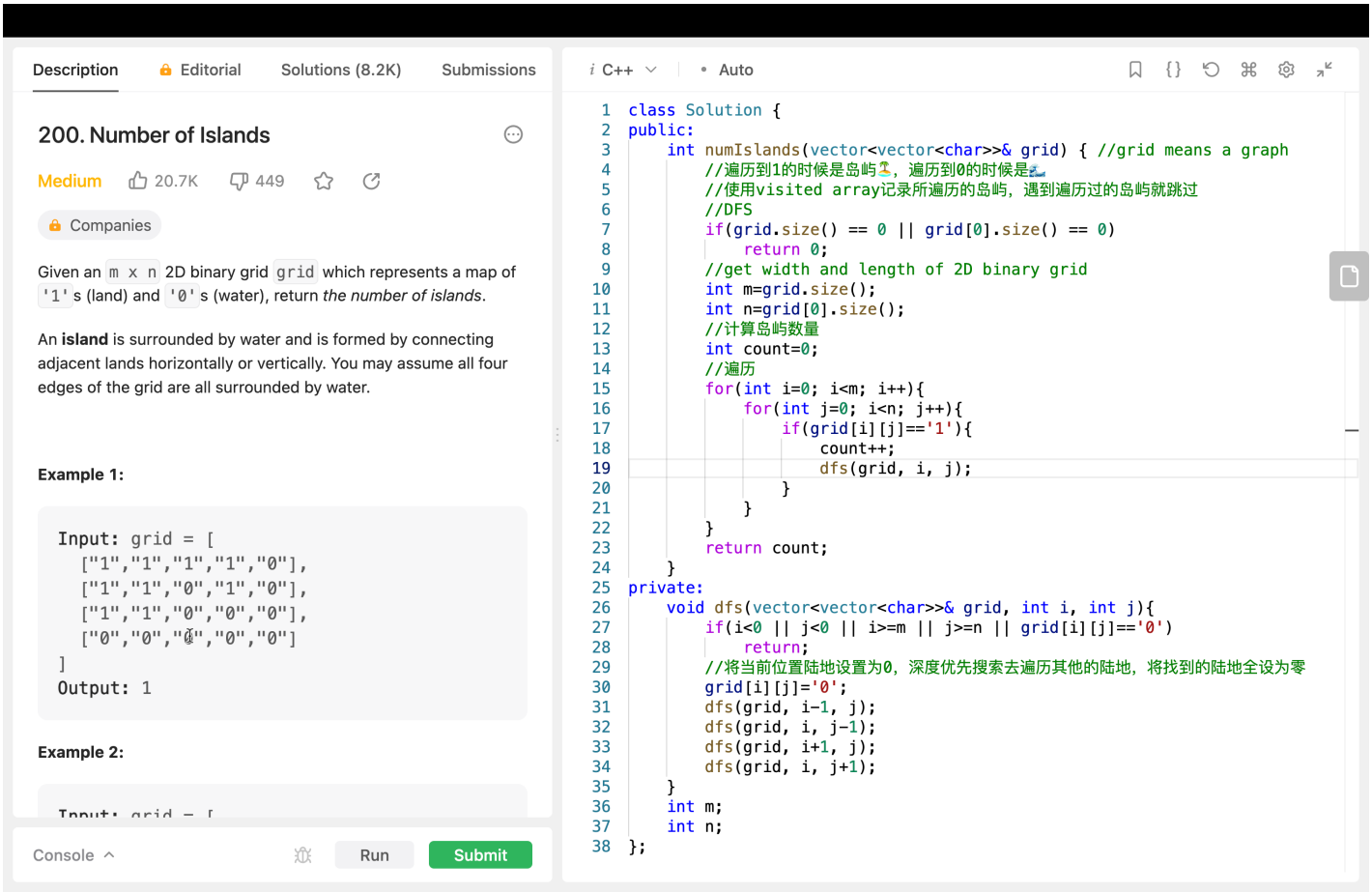
1. Construct a visited array(filled with T/F value) and stack
2. Store the adjacent vertex in stack first
3. Store root node in visited array first
4. Once the element is actually called, set the value in array for True, remove the value from stack to visited array; iterated from first left unvisited vertex

- Notes:

• Range-based for loop: for (range_declaration : range_expression)
  • range_declaration: a named variable
  • range_expression: a suitable sequence or a braced-init-list
  • Reference: https://www.geeksforgeeks.org/range-based-loop-c/
• The difference between ++count; and count++:
  • ++count: pre-increment. Increase the value before executing the statement; value is first incremented and then used inside the expression.
  • count++: post-increment. Increase the value after executing this statement; value is first used in an expression and then incremented.



Similar Problem: LeetCode200



Description | Editorial | Solutions (8.2K) | Submissions

200. Number of Islands

Medium 20.7K 449

Companies

Given an `m x n` 2D binary grid `grid` which represents a map of `'1'`s (land) and `'0'`s (water), return *the number of islands*.

An **island** is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

Example 1:

```
Input: grid = [
  ["1","1","1","1","0"],
  ["1","1","0","1","0"],
  ["1","1","0","0","0"],
  ["0","0","0","0","0"]
]
Output: 1
```

Example 2:

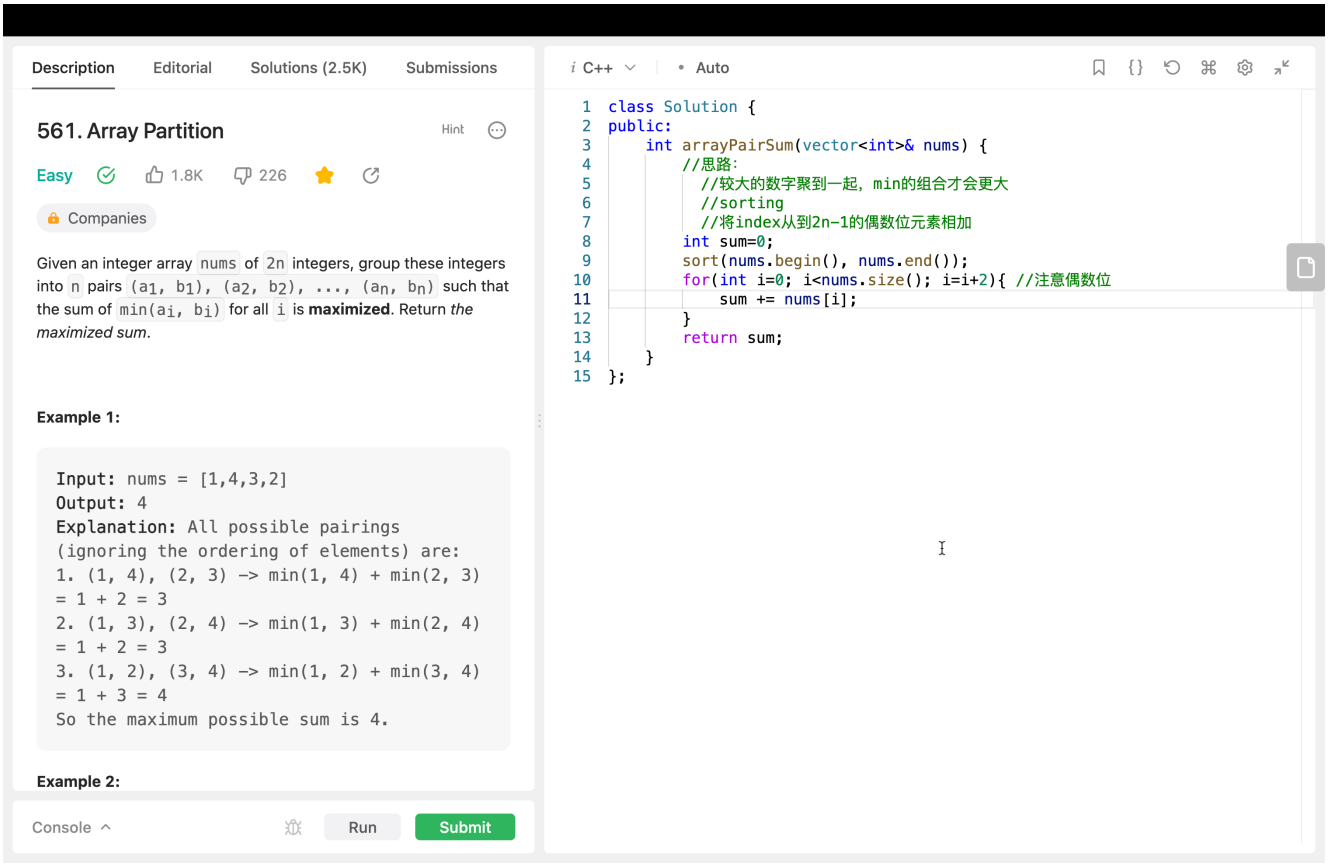
```
Input: grid = [
```

Console | Run | Submit

C++ | Auto

```cpp
class Solution {
public:
    int numIslands(vector<vector<char>>& grid) { //grid means a graph
        //遍历到1的时候是岛屿🏝️，遍历到0的时候是🌊
        //使用visited array记录所遍历的岛屿，遇到遍历过的岛屿就跳过
        //DFS
        if(grid.size() == 0 || grid[0].size() == 0)
            return 0;
        //get width and length of 2D binary grid
        int m=grid.size();
        int n=grid[0].size();
        //计算岛屿数量
        int count=0;
        //遍历
        for(int i=0; i<m; i++){
            for(int j=0; i<n; j++){
                if(grid[i][j]=='1'){
                    count++;
                    dfs(grid, i, j);
                }
            }
        }
        return count;
    }
private:
    void dfs(vector<vector<char>>& grid, int i, int j){
        if(i<0 || j<0 || i>=m || j>=n || grid[i][j]=='0')
            return;
        //将当前位置陆地设置为0，深度优先搜索去遍历其他的陆地，将找到的陆地全设为零
        grid[i][j]='0';
        dfs(grid, i-1, j);
        dfs(grid, i, j-1);
        dfs(grid, i+1, j);
        dfs(grid, i, j+1);
    }
    int m;
    int n;
};
```

LeetCode561: Array Partition



Description | Editorial | Solutions (2.5K) | Submissions

561. Array Partition

Hint

Easy 1.8K 226

Companies

Given an integer array `nums` of `2n` integers, group these integers into `n` pairs `(a1, b1), (a2, b2), ..., (an, bn)` such that the sum of `min(ai, bi)` for all `i` is **maximized**. Return *the maximized sum*.

Example 1:

```
Input: nums = [1,4,3,2]
Output: 4
Explanation: All possible pairings
(ignoring the ordering of elements) are:
1. (1, 4), (2, 3) -> min(1, 4) + min(2, 3)
= 1 + 2 = 3
2. (1, 3), (2, 4) -> min(1, 3) + min(2, 4)
= 1 + 2 = 3
3. (1, 2), (3, 4) -> min(1, 2) + min(3, 4)
= 1 + 3 = 4
So the maximum possible sum is 4.
```

Example 2:

Console | Run | Submit

C++ | Auto

```cpp
class Solution {
public:
    int arrayPairSum(vector<int>& nums) {
        //思路:
          //较大的数字聚到一起，min的组合才会更大
          //sorting
          //将index从到2n-1的偶数位元素相加
        int sum=0;
        sort(nums.begin(), nums.end());
        for(int i=0; i<nums.size(); i=i+2){ //注意偶数位
            sum += nums[i];
        }
        return sum;
    }
};
```

LeetCode621: Task Scheduler

Solution: Greedy Search:
- local optima—>whole optima
- 拆分成子问题
- 找出贪心策略
- 找局部最优
- 堆叠局部最优，得到全局最优

Hash Map:
- key-value pair
- Delete, get, put, DeleteHashTable
- STL: #include <unordered_map>
- Notations: unordered_map<keyDataType, valueDataType> mapName;

Binary Search Tree:
- each node's children <=2
- Value of all nodes of left tree<value of root node
- Value of all nodes of right tree>value of root node

Priority Queue:
- with a priority value, higher priority, the corresponding element served first;
- If elements have same priority number, they are served to their order in queue.
- Not FIFO, but removed based on priority. Highest priority elements removed first.
- Can be implemented by array, binary search tree, linked listheap. While, heap provides most efficient implement.
- When inserting, deleting elements of PQ, we must obey the value property of BST.
- Min heap and Max heap:
  - If it is a min heap, the top of tree is minimum value. We only need to get the peak of the tree is enough. https://www.programiz.com/dsa/heap-sort#heap
  - If the element of the root is not larger/smaller than its children, use .swap() function to realize.
  - Convert the vector into max_heap using make_heap(iter_first, iter_last) in STL under #include<algorithm>

LeetCode703
- Template of priority queue: priority_queue<Type, Container, Functional>
  - Functional: way of element comparison
  - Container: to store data
- .emplace(value): The parameter is added to the priority queue at the top position.
  - The difference between emplace and push: emplace saves unnecessary copy of object

703. Kth Largest Element in a Stream

Easy 4.9K 3K

Companies

Design a class to find the k^{th} largest element in a stream. Note that it is the k^{th} largest element in the sorted order, not the k^{th} distinct element.

Implement `KthLargest` class:

- `KthLargest(int k, int[] nums)` Initializes the object with the integer `k` and the stream of integers `nums`.
- `int add(int val)` Appends the integer `val` to the stream and returns the element representing the k^{th} largest element in the stream.

```cpp
class KthLargest {//goal: 求最大K个元素中的最小值
private:
    priority_queue<int, vector<int>, greater<int>> pq;//min heap
    int k;
public:
    KthLargest(int k, vector<int>& nums) {
        this->k=k;
        for(auto& num: nums){
            pq.emplace(num);
            if(pq.size()>k) pq.pop();
        }
    }

    int add(int val) {
        pq.emplace(val); // add new element
        if(pq.size()>k) pq.pop(); //check the length of pq
        return pq.top(); //get the Kth largest value
    }
};

/**
 * Your KthLargest object will be instantiated and called as such:
 * KthLargest* obj = new KthLargest(k, nums);
 * int param_1 = obj->add(val);
 */
```

SQL:
SELECT
FROM
WHERE
GROUP BY
HAVING
ORDER BY
LIMIT

window functions:
- RANK, DENSE_RANK, ROW_NUMBER
- LAG, LEAD
- CASE
  WHEN…THEN…
  WHEN…THEN…
  ELSE…
  END

join & union
- (LEFT/RIGHT/SELF/INNER/FULL) JOIN…ON

OR
AND
IN
COUNT DISTINCT
LOWER/UPPER
IS NULL
NULLIF
IFNULL
ROUND
AVG
SUM
MIN
MAX

Key Points:
- General linear model VS Generalized linear model
- Frequency VS bayesian
- 异方差
- GARCH, how to calculate volatility by GARCHs
- Local volatility & Implied volatility
- Hash function and hash map
- Variance swap; Swaption 套利策略, volatility swap
- Volatility surface计算,以及参数
- 负利率，bond with negative interest
- Hull-White Model
- Future VS Forwards
- Barrier Option pricing
- Decision tree VS Random forest, Entropy?

- Greeks and option pricing
- When Vega comes to maximum value
- Point process—>passion distribution
- GBM
- Martingale
- Ito Formula
- 多重共线性怎么解决
- SQL： CTE
- ETL Warehouses Pipeline, multi-threadings
- QuickSort
- Linear regression assumption
- Pandas dataFrame, sort, lambda, join