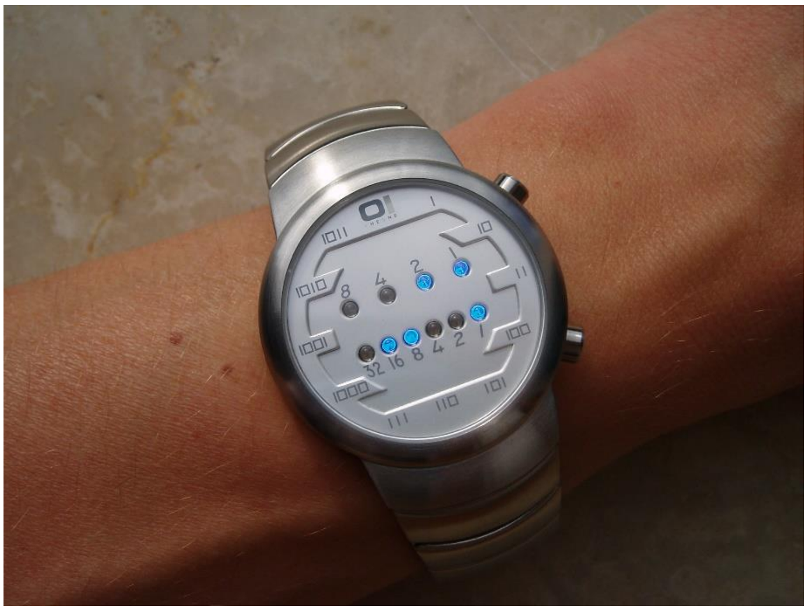
目录

401. Binary Watch

Easy

A binary watch has 4 LEDs on the top which represent the **hours** (**0-11**), and the 6 LEDs on the bottom represent the **minutes** (**0-59**).

Each LED represents a zero or one, with the least significant bit on the right.



For example, the above binary watch reads "3:25".

Given a non-negative integer *n* which represents the number of LEDs that are currently on, return all possible times the watch could represent.

Example:

```
Input: n = 1
Return: ["1:00", "2:00", "4:00", "8:00", "0:01", "0:02", "0:04", "0:08", "0:16", "0:32"]
```

Note:

- The order of output does not matter.
- The hour must not contain a leading zero, for example "01:00" is not valid, it should be "1:00".
- The minute must be consist of two digits and may contain a leading zero, for example "10:2" is not valid, it should be "10:02".

```cpp
class Solution {
public:
   vector<string> readBinaryWatch(int num) {
       vector<string> res;
       for (int h = 0; h < 12; h++)
           for (int m = 0; m < 60; m++) {
               if (bitset<10>(h << 6 | m).count() == num)
                   res.push_back(to_string(h) + (m < 10 ? ":0" : ":")
                                    + to_string(m));
        }
       return res;
   }
};
```

402. Remove K Digits

Medium

Given a non-negative integer *num* represented as a string, remove *k* digits from the number so that the new number is the smallest possible.

Note:

- The length of *num* is less than 10002 and will be ≥ *k*.
- The given *num* does not contain any leading zero.

Example 1:

```
Input: num = "1432219", k = 3

Output: "1219"

Explanation: Remove the three digits 4, 3, and 2 to form the new number 1219 which is the smallest.
```

Example 2:

```
Input: num = "10200", k = 1

Output: "200"

Explanation: Remove the leading 1 and the number is 200. Note that the output must not contain
leading zeroes.
```

Example 3:

```
Input: num = "10", k = 2

Output: "0"

Explanation: Remove all the digits from the number and it is left with nothing which is 0.
```

```
class Solution {
public:
    string removeKdigits(string num, int k) {
        string res = "";
        for (char c : num) {
            while (!res.empty() && res.back() > c && k) {
                res.pop_back();
                k--;
            }
            if (!res.empty() || c != '0') res.push_back(c);
        }
        while (!res.empty() && k--) res.pop_back();
        return res.empty() ? "0" : res;
    }
};
```

403. Frog Jump

Hard

A frog is crossing a river. The river is divided into x units and at each unit there may or may not exist a stone. The frog can jump on a stone, but it must not jump into the water.

Given a list of stones' positions (in units) in sorted ascending order, determine if the frog is able to cross the river by landing on the last stone. Initially, the frog is on the first stone and assume the first jump must be 1 unit.

If the frog's last jump was *k* units, then its next jump must be either *k* - 1, *k*, or *k* + 1 units. Note that the frog can only jump in the forward direction.

Note:

- The number of stones is ≥ 2 and is < 1,100.
- Each stone's position will be a non-negative integer < 2^{31}.
- The first stone's position is always 0.

Example 1:

[0,1,3,5,6,8,12,17]

There are a total of 8 stones.

The first stone at the 0th unit, second stone at the 1st unit,

third stone at the 3rd unit, and so on...

The last stone at the 17th unit.

Return true. The frog can jump to the last stone by jumping

1 unit to the 2nd stone, then 2 units to the 3rd stone, then

2 units to the 4th stone, then 3 units to the 6th stone,

4 units to the 7th stone, and 5 units to the 8th stone.

Example 2:

[0,1,2,3,4,8,9,11]

Return false. There is no way to jump to the last stone as

the gap between the 5th and 6th stone is too large.

```cpp
class Solution {
public:
    bool canCross(vector<int>& stones) {
        int n = stones.size();
        unordered_set<int> s[n];
        s[0].insert(0);
        for (int i = 1; i < n; i++) {
            for (int j = 0; j < i; j++) {
                if (s[j].empty()) continue;
                int d = stones[i] - stones[j];
                if (d > n) continue;
                if (s[j].count(d-1) || s[j].count(d) || s[j].count(d+1) )  {
                    s[i].insert(d);
                }
            }
        }
        return !s[n-1].empty();
    }
};
```

```cpp
class Solution {
public:
    bool canCross(vector<int>& stones) {
        int N = stones.size();
        vector<vector<bool>> dp(N, vector<bool>(N+1, false));
        dp[0][1] = true;
        for(int i = 1; i < N; ++i){
            for(int j = 0; j < i; ++j){
                int d = stones[i] - stones[j];
                if (d < 0 || d > N || !dp[j][d]) continue;
                dp[i][d] = true;
                if (d - 1 >= 0) dp[i][d-1] = true;
                if (d + 1 <= N) dp[i][d+1] = true;
                if (i == N-1) return true;
            }
        }
        return false;
    }
};
```

404. Sum of Left Leaves

Easy

Find the sum of all left leaves in a given binary tree.

Example:

```
    3
   / \
  9  20
    /  \
   15   7
```

There are two left leaves in the binary tree, with values **9** and **15** respectively. Return **24**.

```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    int sumOfLeftLeaves(TreeNode* root) {
        int res = 0;
        dfs(root, res, false);
        return res;
    }
private:
    void dfs(TreeNode* root, int &res, bool type) {
        if (root == nullptr) return;
        if (!root->left && !root->right && type) {
            res += root->val;
        }
        dfs(root->left, res, true);
        dfs(root->right, res, false);
    }
};
```

405. Convert a Number to Hexadecimal

Easy

Given an integer, write an algorithm to convert it to hexadecimal. For negative integer, two’s complement method is used.

Note:

1. All letters in hexadecimal (`a-f`) must be in lowercase.
2. The hexadecimal string must not contain extra leading `0`s. If the number is zero, it is represented by a single zero character `'0'`; otherwise, the first character in the hexadecimal string will not be the zero character.
3. The given number is guaranteed to fit within the range of a 32-bit signed integer.
4. You **must not use *any* method provided by the library** which converts/formats the number to hex directly.

Example 1:

```
Input:

26

Output:

"1a"
```

Example 2:

```
Input:

-1

Output:

"ffffffff"
```

```cpp
class Solution {
public:
    string toHex(int num) {
        const string HEX = "0123456789abcdef";
        if (num == 0) return "0";
        string res;
        unsigned int n = num;
        while (n) {
            res = HEX[(n & 0xf)] + res;
            n >>= 4;
        }
        return res;
    }
};
```

406. Queue Reconstruction by Height

Medium

Suppose you have a random list of people standing in a queue. Each person is described by a pair of integers `(h, k)`, where `h` is the height of the person and `k` is the number of people in front of this person who have a height greater than or equal to `h`. Write an algorithm to reconstruct the queue.

Note:
The number of people is less than 1,100.

Example

```
Input:

[[7,0], [4,4], [7,1], [5,0], [6,1], [5,2]]

Output:

[[5,0], [7,0], [5,2], [6,1], [4,4], [7,1]]
```

```
class Solution {
public:
    vector<vector<int>> reconstructQueue(vector<vector<int>>& people) {
        sort(people.begin(), people.end(), [](const vector<int> &lhs, const
vector<int> &rhs) {
            if (lhs[0] == rhs[0]) return lhs[1] >= rhs[1];
            else return lhs[0] < rhs[0];
        });
        int n = people.size();
        vector<int> indices;
        vector<vector<int>> res(n);

        for (int i = 0; i < n; ++i) {
            indices.push_back(i);
        }

        for (int i = 0; i < n; ++i) {
            int idx = indices[people[i][1]];
            res[indices[people[i][1]]] = people[i];
            indices.erase(indices.begin() + people[i][1]);
        }
        return res;
    }
};
```

```cpp
class Solution {
public:
    vector<vector<int>> reconstructQueue(vector<vector<int>>& people) {
        sort(people.begin(), people.end(), [](const vector<int> &lhs, const
vector<int> &rhs) {
            if (lhs[0] == rhs[0]) return lhs[1] < rhs[1];
            else return lhs[0] > rhs[0];
        });
        vector<vector<int>> res;
        for (const auto &v : people) {
            res.insert(res.begin()+v[1], v);
        }
        return res;
    }
};
```

407. Trapping Rain Water II

Hard

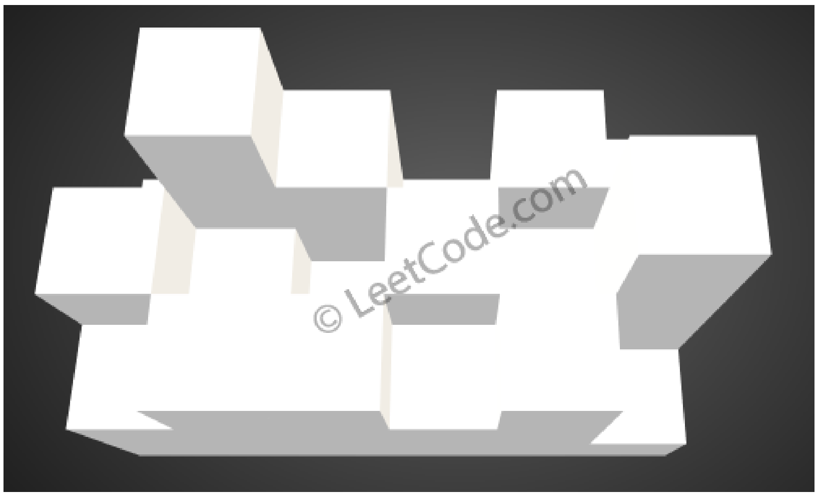
Given an `m x n` matrix of positive integers representing the height of each unit cell in a 2D elevation map, compute the volume of water it is able to trap after raining.

Note:

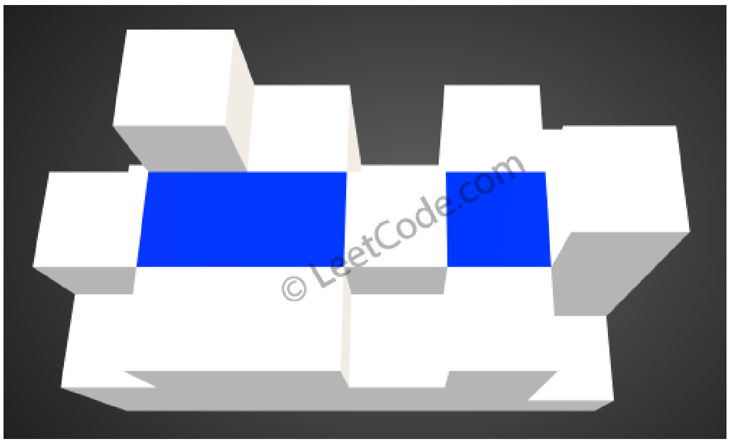
Both *m* and *n* are less than 110. The height of each unit cell is greater than 0 and is less than 20,000.

Example:

```
Given the following 3x6 height map:

[

  [1,4,3,1,3,2],

  [3,2,1,3,2,4],

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

]

Return 4.
```



The above image represents the elevation map `[[1,4,3,1,3,2],[3,2,1,3,2,4],[2,3,3,2,3,1]]` before the rain.



After the rain, water is trapped between the blocks. The total volume of water trapped is 4.

```
class Solution {
public:
    int trapRainWater(vector<vector<int>>& heightMap) {
        if (heightMap.empty()) return 0;
        n = heightMap.size(), m = heightMap[0].size();
        priority_queue<node> pq;
        for (int i = 0; i < n; i++) {
            if (i == 0 || i == n-1) {
                for (int j = 0; j < m; j++)  {
                    pq.push({i, j, heightMap[i][j]});
                    heightMap[i][j] = -1;
                }
            } else {
                pq.push({i, 0, heightMap[i][0]});
                pq.push({i, m-1, heightMap[i][m-1]});
                heightMap[i][0] = heightMap[i][m-1] = -1;
            }
        }
        int res = 0, MIN_height = INT_MIN;
        while (!pq.empty()) {
            node t = pq.top();
            pq.pop();
            MIN_height = max(MIN_height, t.value);
            for (int k = 0; k < 4; k++) {
                int xx = t.x+dx[k], yy = t.y+dy[k];
                if (inside(xx, yy) && heightMap[xx][yy] >= 0) {
                    int &h = heightMap[xx][yy];
                    if (h < MIN_height) res += MIN_height-h;
                    pq.push({xx, yy, h});
                    h = -1;
                }
            }
        }
        return res;
    }

private:
    int n, m;
    const int dx[4] = {0,0,1,-1}, dy[4] = {-1,1,0,0};

    struct node{
        int x, y, value;
        node(int x, int y, int v):x(x), y(y), value(v){}
        bool operator < (const node &rhs) const {
            return value > rhs.value;
        }
    };

    bool inside(int x, int y) {
        return x >= 0 && y >= 0 && x < n && y < m;
    }
};
```

409. Longest Palindrome

Easy

Given a string which consists of lowercase or uppercase letters, find the length of the longest palindromes that can be built with those letters.

This is case sensitive, for example `"Aa"` is not considered a palindrome here.

Note:
Assume the length of given string will not exceed 1,010.

Example:

```
Input:

"abccccdd"

Output:

7

Explanation:

One longest palindrome that can be built is "dccaccd", whose length is 7.
```

```
class Solution {
public:
    int longestPalindrome(string s) {
        unordered_map<char, int> m;
        for (auto c : s) m[c]++;
        int res = 0;
        for (auto pci : m) {
            res += pci.second/2*2;
            if (res % 2 == 0 && pci.second % 2 == 1) res++;
        }
        return res;
    }
};
```

410. Split Array Largest Sum

Hard

Given an array which consists of non-negative integers and an integer *m*, you can split the array into *m* non-empty continuous subarrays. Write an algorithm to minimize the largest sum among these *m* subarrays.

Note:
If *n* is the length of array, assume the following constraints are satisfied:

- $1 \le n \le 1000$
- $1 \le m \le \min(50, n)$

Examples:

Input:

nums = [7,2,5,10,8]

m = 2

Output:

18

Explanation:

There are four ways to split **nums** into two subarrays.

The best way is to split it into **[7,2,5]** and **[10,8]**,

where the largest sum among the two subarrays is only 18.

```cpp
class Solution {
public:
    int splitArray(vector<int>& nums, int m) {
        long long left = 0, right = 0;
        for (auto i : nums) {
            left = max(left, (long long)i);
            right += i;
        }

        while (left < right) {
            auto mid = left + (right - left) / 2;
            if (judge(nums, m - 1, mid)) right = mid;
            else left = mid + 1;
        }
        return left;
    }
private:
    bool judge(const vector<int>& nums, int cnt, long long max) {
        long long sum = 0;
        for (auto i : nums) {
            if (i > max) return false;
            else if (sum + i <= max) sum += i;
            else {
                sum = i;
                if (--cnt < 0) return false;
            }
        }
        return true;
    }
};
```

412. Fizz Buzz

Easy

Write a program that outputs the string representation of numbers from 1 to *n*.

But for multiples of three it should output “Fizz” instead of the number and for the multiples of five output “Buzz”. For numbers which are multiples of both three and five output “FizzBuzz”.

Example:

```
n = 15,

Return:

[

    "1",

    "2",

    "Fizz",

    "4",

    "Buzz",

    "Fizz",

    "7",

    "8",

    "Fizz",

    "Buzz",

    "11",

    "Fizz",

    "13",

    "14",

    "FizzBuzz"

]
```

```
class Solution {
public:
    vector<string> fizzBuzz(int n) {
        vector<string> res;
        for (int i = 1; i <= n; i++) {
            if (i % 3 != 0 && i % 5 != 0) {
                res.push_back(to_string(i));
            } else if (i % 3 != 0) {
                res.push_back("Buzz");
            } else if (i % 5 != 0) {
                res.push_back("Fizz");
            } else {
                res.push_back("FizzBuzz");
            }
        }
        return res;
    }
};
```

413. Arithmetic Slices

Medium

A sequence of number is called arithmetic if it consists of at least three elements and if the difference between any two consecutive elements is the same.

For example, these are arithmetic sequence:

```
1, 3, 5, 7, 9

7, 7, 7, 7

3, -1, -5, -9
```

The following sequence is not arithmetic.

```
1, 1, 2, 5, 7
```

A zero-indexed array A consisting of N numbers is given. A slice of that array is any pair of integers (P, Q) such that 0 <= P < Q < N.

A slice (P, Q) of array A is called arithmetic if the sequence:
A[P], A[p + 1], ..., A[Q - 1], A[Q] is arithmetic. In particular, this means that P + 1 < Q.

The function should return the number of arithmetic slices in the array A.

Example:

```
A = [1, 2, 3, 4]

return: 3, for 3 arithmetic slices in A: [1, 2, 3], [2, 3, 4] and [1, 2, 3, 4] itself.
```

```cpp
class Solution {
public:
    int numberOfArithmeticSlices(vector<int>& A) {
        if (A.size() < 3) return 0;
        int cnt = 0, res = 0;
        for(int i = 2; i < A.size(); i++) {
            if (A[i-1] - A[i-2] == A[i] - A[i-1]) res += ++cnt;
            else cnt = 0;
        }
        return res;
    }
};
```

414. Third Maximum Number

Easy

Given a **non-empty** array of integers, return the **third** maximum number in this array. If it does not exist, return the maximum number. The time complexity must be in O(n).

Example 1:

Input: [3, 2, 1]

Output: 1

Explanation: The third maximum is 1.

Example 2:

Input: [1, 2]

Output: 2

Explanation: The third maximum does not exist, so the maximum (2) is returned instead.

Example 3:

Input: [2, 2, 3, 1]

Output: 1

Explanation: Note that the third maximum here means the third maximum distinct number.

Both numbers with value 2 are both considered as second maximum.

```cpp
class Solution {
public:
    int thirdMax(vector<int>& nums) {
        long long a, b, c;
        a = b = c = LLONG_MIN;
        for (auto num : nums) {
            if (num <= c || num == b || num == a) continue;
            c = num;
            if (c > b) swap(b, c);
            if (b > a) swap(a, b);
        }
        return c == LLONG_MIN ? a : c;
    }
};
```

```cpp
class Solution {
public:
    int thirdMax(vector<int>& nums) {
        set<int> top3;
        for (int num : nums) {
            top3.insert(num);
            if (top3.size() > 3)
                top3.erase(top3.begin());
        }
        return top3.size() == 3 ? *top3.begin() : *top3.rbegin();
    }
};
```

415. Add Strings

Easy

Given two non-negative integers `num1` and `num2` represented as string, return the sum of `num1` and `num2`.

Note:

1. The length of both `num1` and `num2` is < 5100.
2. Both `num1` and `num2` contains only digits `0-9`.
3. Both `num1` and `num2` does not contain any leading zero.
4. You **must not use any built-in BigInteger library** or **convert the inputs to integer** directly.

```
class Solution {
public:
    string addStrings(string num1, string num2) {
        reverse(num1.begin(), num1.end());
        reverse(num2.begin(), num2.end());
        string res;
        int carry = 0, n = num1.length(), m = num2.length();
        int i = 0, j = 0;
        while (i < n || j < m) {
            carry += i < n ? num1[i++]-'0' : 0;
            carry += j < m ? num2[j++]-'0' : 0;
            res += carry % 10 + '0';
            carry /= 10;
        }
        if (carry) res += '1';
        reverse(res.begin(), res.end());
        return res;
    }
};
```

416. Partition Equal Subset Sum

Medium

Given a **non-empty** array containing **only positive integers**, find if the array can be partitioned into two subsets such that the sum of elements in both subsets is equal.

Note:

1. Each of the array element will not exceed 100.
2. The array size will not exceed 200.

Example 1:

```
Input: [1, 5, 11, 5]

Output: true

Explanation: The array can be partitioned as [1, 5, 5] and [11].
```

Example 2:

```
Input: [1, 2, 3, 5]

Output: false

Explanation: The array cannot be partitioned into equal sum subsets.
```

```cpp
class Solution {
public:
    bool canPartition(vector<int>& nums) {
        int sum = accumulate(nums.begin(), nums.end(), 0);
        if (sum % 2 != 0) return false;
        int V = sum/2, n = nums.size();
        vector<bool> dp(V+1, false);
        dp[0] = true;
        for (int i = 0; i < n; i++) {
            for (int j = V; j >= 0; j--) {
                if (j < nums[i])  continue;
                dp[j] = dp[j] || dp[j-nums[i]];
            }
            if (dp[V]) return true;
        }
        return false;
    }
};
```

```cpp
class Solution {
public:
    bool canPartition(vector<int>& nums) {
        bitset<100*200/2+1> bits(1);
        int sum = accumulate(nums.begin(), nums.end(), 0);
        if (sum % 2 != 0) return false;
        for (auto n : nums) bits |= bits << n;
        return bits[sum >> 1];
    }
};
```

417. Pacific Atlantic Water Flow

Medium

Given an `m x n` matrix of non-negative integers representing the height of each unit cell in a continent, the "Pacific ocean" touches the left and top edges of the matrix and the "Atlantic ocean" touches the right and bottom edges.

Water can only flow in four directions (up, down, left, or right) from a cell to another one with height equal or lower.

Find the list of grid coordinates where water can flow to both the Pacific and Atlantic ocean.

Note:

1. The order of returned grid coordinates does not matter.
2. Both *m* and *n* are less than 150.

Example:

```
Given the following 5x5 matrix:

  Pacific ~   ~   ~   ~   ~

       ~  1   2   2   3  (5) *

       ~  3   2   3  (4) (4) *

       ~  2   4  (5)  3   1  *

       ~ (6) (7)  1   4   5  *

       ~ (5)  1   1   2   4  *

          *   *   *   *   * Atlantic

Return:

[[0, 4], [1, 3], [1, 4], [2, 2], [3, 0], [3, 1], [4, 0]] (positions with parentheses in above matrix).
```

```cpp
class Solution {
public:
    vector<vector<int>> pacificAtlantic(vector<vector<int>>&matrix) {
        if (matrix.empty()) return {};
        n = matrix.size(), m = matrix[0].size();
        buffer.resize(n, vector<int> (m, 0));
        visit.resize(n, vector<bool> (m, false));
        for (int j = 0; j < m; ++j) dfs(0, j, matrix);
        for (int i = 1; i < n; ++i) dfs(i, 0, matrix);
        visit = vector<vector<bool>>(n, vector<bool> (m, false));
        for (int j = 0; j < m; ++j) dfs(n-1, j, matrix);
        for (int i = 0; i < n-1; ++i) dfs(i, m-1, matrix);
        return res;
    }
private:
    int n, m;
    vector<vector<int>> res, buffer;
    vector<vector<bool>> visit;
    const vector<int> dx{0,0,1,-1};
    const vector<int> dy{-1,1,0,0};

    void dfs(int i, int j, vector<vector<int>>& matrix) {
        if (visit[i][j]) return;
        visit[i][j] = true;
        if (++buffer[i][j] == 2){
            res.push_back({i, j});
        }
        for (int k = 0; k < 4; ++k) {
            int x = i + dx[k], y = j + dy[k];
            if (x < 0 || y < 0 || x >= n || y >= m) continue;
            if (!visit[x][y] && matrix[x][y] >= matrix[i][j]) {
                dfs(x, y, matrix);
            }
        }
    }
};
```

419. Battleships in a Board

Medium

Given an 2D board, count how many battleships are in it. The battleships are represented with `'X'`s, empty slots are represented with `'.'`s. You may assume the following rules:

- You receive a valid board, made of only battleships or empty slots.
- Battleships can only be placed horizontally or vertically. In other words, they can only be made of the shape `1xN` (1 row, N columns) or `Nx1` (N rows, 1 column), where N can be of any size.
- At least one horizontal or vertical cell separates between two battleships - there are no adjacent battleships.

Example:

```
X..X

...X

...X
```

In the above board there are 2 battleships.

Invalid Example:

```
...X

XXXX

...X
```

This is an invalid board that you will not receive - as battleships will always have a cell separating between them.

Follow up:

Could you do it in **one-pass**, using only **O(1) extra memory** and **without modifying** the value of the board?

```cpp
class Solution {
public:
    int countBattleships(vector<vector<char>>& board) {
        int n = board.size(), m = board[0].size();
        int res = 0;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                if (board[i][j] == 'X') {
                    res++;
                    if (j != 0 && board[i][j-1] == 'X') res--;
                    if (i != 0 && board[i-1][j] == 'X') res--;
                }
            }
        }
        return res;
    }
};
```

420. Strong Password Checker

Hard

A password is considered strong if below conditions are all met:

1. It has at least 6 characters and at most 20 characters.
2. It must contain at least one lowercase letter, at least one uppercase letter, and at least one digit.
3. It must NOT contain three repeating characters in a row ("...aaa..." is weak, but "...aa...a..." is strong, assuming other conditions are met).

Write a function strongPasswordChecker(s), that takes a string s as input, and return the **MINIMUM** change required to make s a strong password. If s is already strong, return 0.

Insertion, deletion or replace of any one character are all considered as one change.

421. Maximum XOR of Two Numbers in an Array

Medium

Given a **non-empty** array of numbers, a_0, a_1, a_2, … , a_{n-1}, where $0 \le a_i < 2^{31}$.

Find the maximum result of a_i XOR a_j, where $0 \le i, j < n$.

Could you do this in O(*n*) runtime?

Example:

Input: [3, 10, 5, 25, 2, 8]

Output: 28

Explanation: The maximum result is **5** ^ **25** = 28.

```
class Solution {
public:
    int findMaximumXOR(vector<int>& nums) {
        TrieNode *Trie = new TrieNode(-1);
        for (auto &i : nums) Trie->insert(i);
        int res = 0;
        for (auto &i : nums) {
            res = max(res, Trie->search(i));
        }
        return res;
     }
    class TrieNode{
    public:
        TrieNode(int v) : left(nullptr), right(nullptr), val(v) {}
        void insert(int i) {
            TrieNode *p = this;
            for (int k = 31; k >= 0; --k) {
                if (i & (1 << k)) {
                    if (!p->left) p->left = new TrieNode(1);
                    p = p->left;
                }
                else {
                    if (!p->right) p->right = new TrieNode(0);
                    p = p->right;
                }
            }
        }

        int search(int i) {
            TrieNode *p = this;
            int sum = 0;
            for(int k = 31; k >= 0; k--){
                int tmp = i & (1 << k);
                if (p->left && p->right){
                    if (!tmp) p = p->left;
                    else p = p->right;
                } else {
                    p = p->left ? p->left : p->right;
                }
                sum += tmp ^ (p->val << k);
            }
            return sum;
        }
    private:
        int val;
        TrieNode *left, *right;
    };
};
```

423. Reconstruct Original Digits from English

Medium

Given a **non-empty** string containing an out-of-order English representation of digits `0-9`, output the digits in ascending order.

Note:

1. Input contains only lowercase English letters.
2. Input is guaranteed to be valid and can be transformed to its original digits. That means invalid inputs such as "abc" or "zerone" are not permitted.
3. Input length is less than 50,000.

Example 1:

```
Input: "owoztneoer"

Output: "012"
```

Example 2:

```
Input: "fviefuro"

Output: "45"
```

```cpp
class Solution {
public:
    string originalDigits(string s) {
        string res;
        chars.resize(26, 0);
        for (auto &i : s) chars[i-'a']++;
        for (auto &pci : vect) {
            f(pci.first, pci.second, res);
        }
        sort(res.begin(), res.end());
        return res;
    }

private:
    vector<int> chars;
    vector<string> nums{"zero", "one", "two", "three", "four",
                        "five", "six", "seven", "eight", "nine"};
    vector<pair<char, int>> vect{{'z', 0}, {'w', 2}, {'u', 4},
                                 {'x', 6}, {'r', 3}, {'f', 5},
                                 {'s', 7}, {'h', 8}, {'o', 1}, {'i', 9}};
    void f(char c, int number, string &res) {
        int cnt = chars[c-'a'];
        for (auto &i : nums[number]) chars[i-'a'] -= cnt;
        res += string(cnt, '0'+number);
    }
};
```

424. Longest Repeating Character Replacement

Medium

Given a string `s` that consists of only uppercase English letters, you can perform at most `k` operations on that string.

In one operation, you can choose **any** character of the string and change it to any other uppercase English character.

Find the length of the longest sub-string containing all repeating letters you can get after performing the above operations.

Note:
Both the string's length and *k* will not exceed 10^4.

Example 1:

Input:

s = "ABAB", k = 2

Output:

4

Explanation:

Replace the two 'A's with two 'B's or vice versa.

Example 2:

Input:

s = "AABABBA", k = 1

Output:

4

Explanation:

Replace the one 'A' in the middle with 'B' and form "AABBBBA".

The substring "BBBB" has the longest repeating letters, which is 4.

```
class Solution {
public:
    int characterReplacement(string s, int k) {
        int n = s.size(), res = 0;
        vector<int> cnt(26, 0);
        int start = 0, end = 0, localMaxFreq = 0;
        for(; end < n; end++) {
            localMaxFreq = max(localMaxFreq, ++cnt[s[end]-'A']);
            if ((end-start+1) - localMaxFreq > k) {
                ret = max(ret, end-start);
                --cnt[s[start++]-'A'];
                localMaxFreq = *max_element(count.begin(), count.end());
            }
        }
        return max(ret, end-start);
    }
};
```

427. Construct Quad Tree

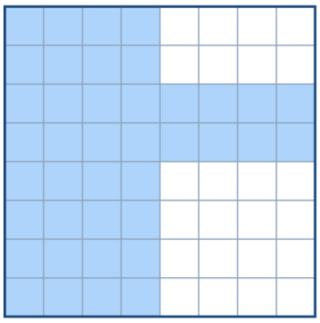
Medium

We want to use quad trees to store an `N x N` boolean grid. Each cell in the grid can only be true or false. The root node represents the whole grid. For each node, it will be subdivided into four children nodes **until the values in the region it represents are all the same**.

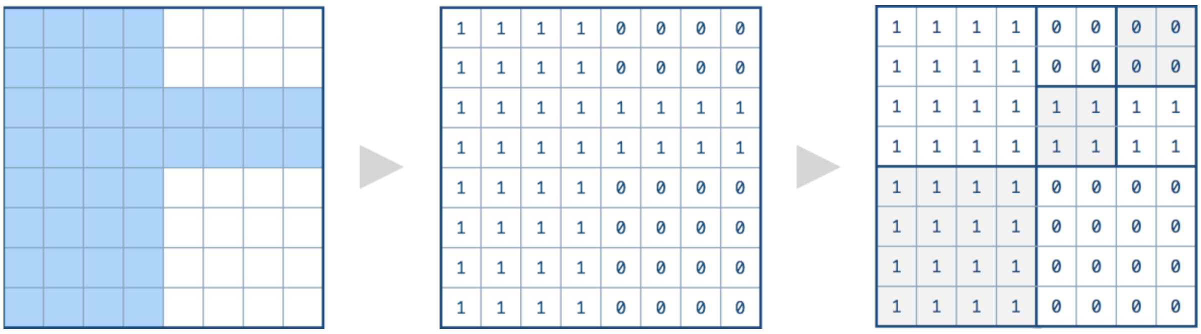
Each node has another two boolean attributes : `isLeaf` and `val`. `isLeaf` is true if and only if the node is a leaf node. The `val` attribute for a leaf node contains the value of the region it represents.

Your task is to use a quad tree to represent a given grid. The following example may help you understand the problem better:

Given the `8 x 8` grid below, we want to construct the corresponding quad tree:

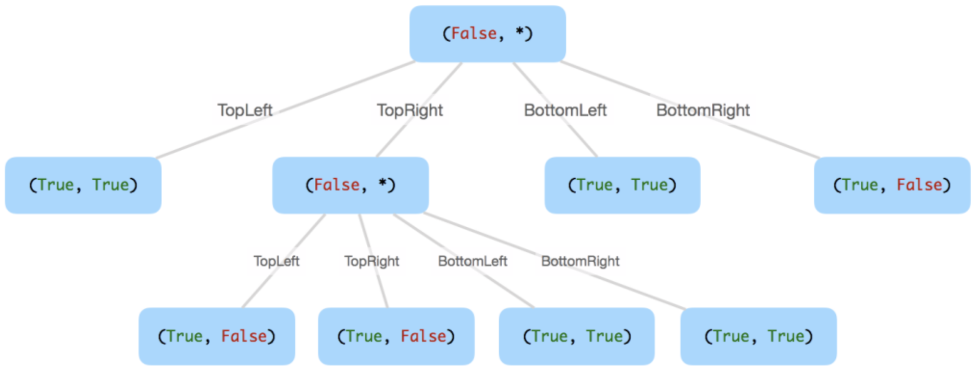


It can be divided according to the definition above:



The corresponding quad tree should be as following, where each node is represented as a `(isLeaf, val)` pair.

For the non-leaf nodes, `val` can be arbitrary, so it is represented as `*`.



Note:

1. `N` is less than `1000` and guaranteened to be a power of 2.
2. If you want to know more about the quad tree, you can refer to its wiki.

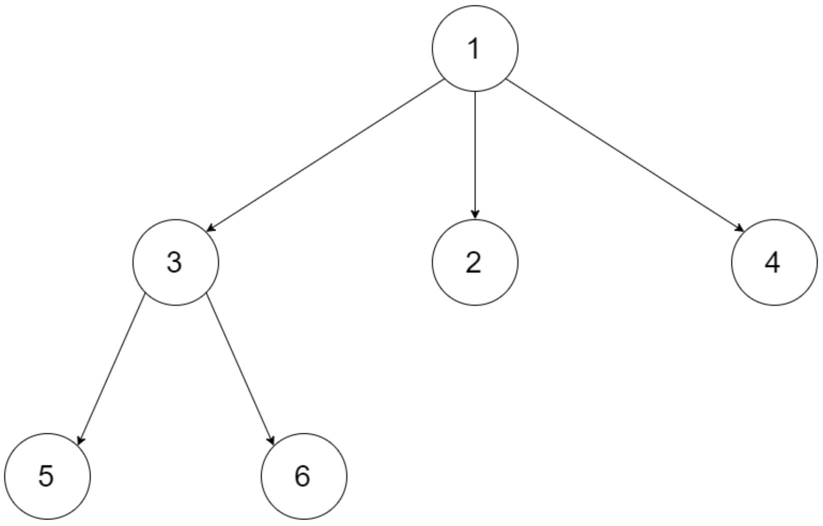
```
class Solution {
public:
    Node* construct(vector<vector<int>>& grid) {
        int n = grid.size();
        return dfs(0, 0, n, n, grid);
    }

private:
    Node* dfs(int x1, int y1, int x2, int y2, vector<vector<int>>&
grid) {
        int x_mid = x1 + (x2-x1)/2;
        int y_mid = y1 + (y2-y1)/2;
        int val = grid[x1][y1];
        for (int r = x1; r < x2; r++) {
            for (int c = y1; c < y2; c++) {
                if (grid[r][c] != val) {
                    return new Node(
                        true, false,
                        dfs(x1, y1, x_mid, y_mid, grid),
                        dfs(x1, y_mid, x_mid, y2, grid),
                        dfs(x_mid, y1, x2, y_mid, grid),
                        dfs(x_mid, y_mid, x2, y2, grid)
                    );
                }
            }
        }
        return new Node(val != 0, true, nullptr, nullptr, nullptr,
nullptr);
    }
};
```

429. N-ary Tree Level Order Traversal

Easy

Given an n-ary tree, return the level order traversal of its nodes' values. (ie, from left to right, level by level).

For example, given a `3-ary` tree:



We should return its level order traversal:

```
[

     [1],

     [3,2,4],

     [5,6]

]
```

Note:

1. The depth of the tree is at most `1000`.
2. The total number of nodes is at most `5000`.

```cpp
/*
// Definition for a Node.
class Node {
public:
    int val;
    vector<Node*> children;

    Node() {}

    Node(int _val, vector<Node*> _children) {
        val = _val;
        children = _children;
    }
};
*/
class Solution {
public:
    vector<vector<int>> levelOrder(Node* root) {
        vector<vector<int>> res;
        if (root == nullptr) return res;
        queue<Node*> q;
        q.push(root);
        while (!q.empty()) {
            int sz = q.size();
            vector<int> vect;
            while (sz--) {
                vect.push_back(q.front()->val);
                auto &child = q.front()->children;
                q.pop();
                for (auto &i : child) q.push(i);
            }
            res.push_back(vect);
        }
        return res;
    }
};
```

430. Flatten a Multilevel Doubly Linked List

Medium

49185FavoriteShare

You are given a doubly linked list which in addition to the next and previous pointers, it could have a child pointer, which may or may not point to a separate doubly linked list. These child lists may have one or more children of their own, and so on, to produce a multilevel data structure, as shown in the example below.

Flatten the list so that all the nodes appear in a single-level, doubly linked list. You are given the head of the first level of the list.

Example:

Input:

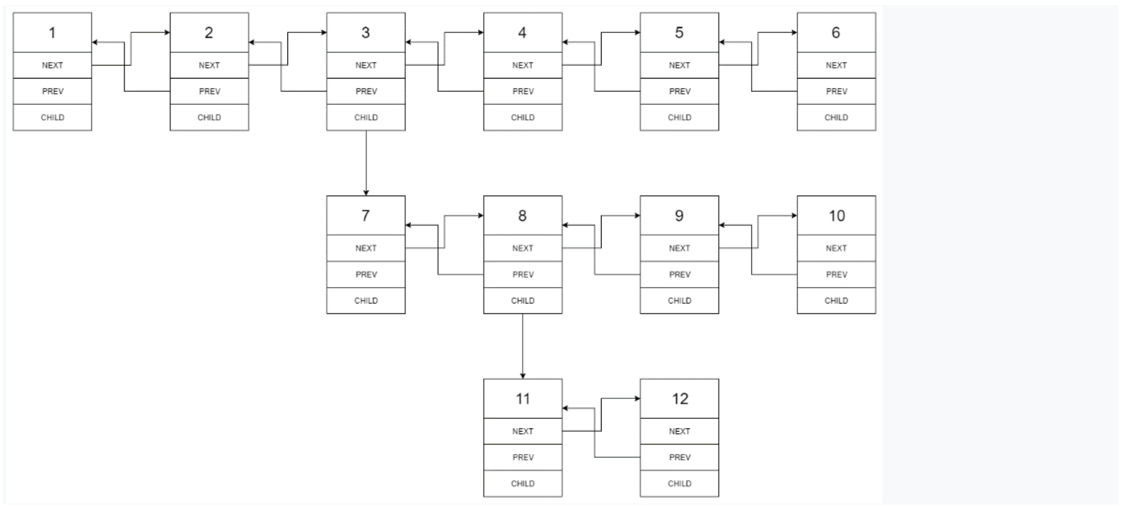
```
 1---2---3---4---5---6--NULL
         |
         7---8---9---10--NULL
             |
             11--12--NULL
```

Output:

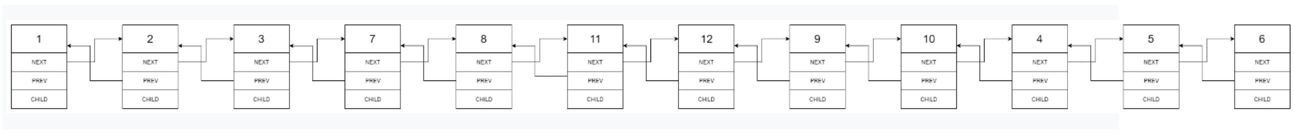
```
1-2-3-7-8-11-12-9-10-4-5-6-NULL
```

Explanation for the above example:

Given the following multilevel doubly linked list:



We should return the following flattened doubly linked list:



```
/*
// Definition for a Node.
class Node {
public:
    int val;
    Node* prev;
    Node* next;
    Node* child;
};
*/
class Solution {
public:
    Node* flatten(Node* head) {
        if (!head) return nullptr;
        return f(head).first;
    }
private:
    pair<Node*, Node*> f(Node* head) {
        if (!head) return {nullptr, nullptr};
        auto p = f(head->next);
        Node *t = head;
        if (head->child) {
            auto q = f(head->child);
            head->child = nullptr;
            Union(head, q.first);
            t = q.second;
        }
        if (!p.first) return {head, t};
        Union(t, p.first);
        return {head, p.second};
    }

    void Union(Node *a, Node *b) {
        a->next = b;
        b->prev = a;
    }
};
```

432. All O`one Data Structure

Hard

Implement a data structure supporting the following operations:

1. Inc(Key) - Inserts a new key with value 1. Or increments an existing key by 1. Key is guaranteed to be a **non-empty** string.
2. Dec(Key) - If Key's value is 1, remove it from the data structure. Otherwise decrements an existing key by 1. If the key does not exist, this function does nothing. Key is guaranteed to be a **non-empty** string.
3. GetMaxKey() - Returns one of the keys with maximal value. If no element exists, return an empty string `""`.
4. GetMinKey() - Returns one of the keys with minimal value. If no element exists, return an empty string `""`.

Challenge: Perform all these in O(1) time complexity.

```
class AllOne {
public:
    /** Initialize your data structure here. */
    AllOne() {}
    /** Inserts a new key <Key> with value 1. Or increments an existing key by
1. */
    void inc(string key) {
        if (!mp.count(key)) {
            mp[key] = List.insert(List.begin(), {0, {key}});
        }
        auto next = mp[key], cur = next++;
        if (next == List.end() || next->value > cur->value + 1)
            next = List.insert(next, {cur->value + 1, {}});
        next->keys.insert(key);
        mp[key] = next;
        cur->keys.erase(key);
        if (cur->keys.empty())
            List.erase(cur);
    }
    /** Decrements an existing key by 1. If Key's value is 1, remove it from
the data structure. */
    void dec(string key) {
        if (!mp.count(key)) return;

        auto prev = mp[key], cur = prev--;
        mp.erase(key);
        if (cur->value > 1) {
            if (cur == List.begin() || prev->value < cur->value - 1)
                prev = List.insert(cur, {cur->value - 1, {}});
            prev->keys.insert(key);
            mp[key] = prev;
        }
        cur->keys.erase(key);
        if (cur->keys.empty())
            List.erase(cur);
    }



    /** Returns one of the keys with maximal value. */
    string getMaxKey() {
        return List.empty() ? "" : *(List.rbegin()->keys.begin());
    }
    /** Returns one of the keys with Minimal value. */
    string getMinKey() {
        return List.empty() ? "" : *(List.begin()->keys.begin());
    }
private:
    struct node {
        int value;
        unordered_set<string> keys;
    };

    list <node> List;
    unordered_map<string, list<node>::iterator> mp;
};

/**
 * Your AllOne object will be instantiated and called as such:
 * AllOne* obj = new AllOne();
 * obj->inc(key);
 * obj->dec(key);
 * string param_3 = obj->getMaxKey();
 * string param_4 = obj->getMinKey();
 */
```

433. Minimum Genetic Mutation

Medium

A gene string can be represented by an 8-character long string, with choices from `"A"`, `"C"`, `"G"`, `"T"`.

Suppose we need to investigate about a mutation (mutation from "start" to "end"), where ONE mutation is defined as ONE single character changed in the gene string.

For example, `"AACCGGTT"` -> `"AACCGGTA"` is 1 mutation.

Also, there is a given gene "bank", which records all the valid gene mutations. A gene must be in the bank to make it a valid gene string.

Now, given 3 things - start, end, bank, your task is to determine what is the minimum number of mutations needed to mutate from "start" to "end". If there is no such a mutation, return -1.

Note:

1. Starting point is assumed to be valid, so it might not be included in the bank.
2. If multiple mutations are needed, all mutations during in the sequence must be valid.
3. You may assume start and end string is not the same.

Example 1:

```
start: "AACCGGTT"

end:      "AACCGGTA"

bank: ["AACCGGTA"]

return: 1
```

Example 2:

```
start: "AACCGGTT"

end:      "AAACGGTA"

bank: ["AACCGGTA", "AACCGCTA", "AAACGGTA"]

return: 2
```

Example 3:

```
start: "AAAAACCC"

end:      "AACCCCCC"

bank: ["AAAACCCC", "AAACCCCC", "AACCCCCC"]

return: 3
```

```cpp
class Solution {
public:
    int minMutation(string start, string end, vector<string> &bank) {
        vector<char> gene{'A', 'T', 'C', 'G'};
        queue<string> q;
        q.push(start);
        int res = 0;
        unordered_set<string> My_set(bank.begin(), bank.end());
        if (!My_set.count(end)) return -1;
        while (!q.empty()) {
            int sz = q.size();
            res++;
            while (sz--) {
                string s = q.front();
                q.pop();
                for (int i = 0; i < 8; i++) {
                    int c = s[i];
                    for (int j = 0; j < 4; j++) {
                        if (gene[j] == c) continue;
                        s[i] = gene[j];
                        auto it = My_set.find(s);
                        if (it != My_set.end()) {
                            if (s == end) return res;
                            My_set.erase(it);
                            q.push(s);
                        }
                    }
                    s[i] = c;
                }
            }
        }
        return -1;
    }
};
```

434. Number of Segments in a String

Easy

Count the number of segments in a string, where a segment is defined to be a contiguous sequence of non-space characters.

Please note that the string does not contain any **non-printable** characters.

Example:

Input: "Hello, my name is John"

Output: 5

```
class Solution {
public:
    int countSegments(string s) {
        int res = 0;
        istringstream ss(s);
        while (ss >> s) res++;
        return res;
    }
};
```

435. Non-overlapping Intervals

Medium

Given a collection of intervals, find the minimum number of intervals you need to remove to make the rest of the intervals non-overlapping.

Example 1:

Input: [[1,2],[2,3],[3,4],[1,3]]

Output: 1

Explanation: [1,3] can be removed and the rest of intervals are non-overlapping.

Example 2:

Input: [[1,2],[1,2],[1,2]]

Output: 2

Explanation: You need to remove two [1,2] to make the rest of intervals non-overlapping.

Example 3:

Input: [[1,2],[2,3]]

Output: 0

Explanation: You don't need to remove any of the intervals since they're already non-overlapping.

Note:

1. You may assume the interval's end point is always bigger than its start point.
2. Intervals like [1,2] and [2,3] have borders "touching" but they don't overlap each other.

```cpp
class Solution {
public:
    int eraseOverlapIntervals(vector<vector<int>>& intervals) {
        auto cmp = [](const vector<int> &lhs, const vector<int> &rhs){
            return lhs[0] < rhs[0];
        };
        sort(intervals.begin(), intervals.end(), cmp);
        stack<vector<int>> stk;
        int i = 0, n = intervals.size();
        while (i < n) {
            auto &v = intervals[i];
            if (stk.empty() || stk.top()[1] <= v[0]) {
                stk.push(v);
                i++;
            }
            else if (stk.top()[1] > v[1]) stk.pop();
            else i++;
        }
        return n - stk.size();
    }
};
```

436. Find Right Interval

Medium

Given a set of intervals, for each of the interval i, check if there exists an interval j whose start point is bigger than or equal to the end point of the interval i, which can be called that j is on the "right" of i.

For any interval i, you need to store the minimum interval j's index, which means that the interval j has the minimum start point to build the "right" relationship for interval i. If the interval j doesn't exist, store -1 for the interval i. Finally, you need output the stored value of each interval as an array.

1. You may assume the interval's end point is always bigger than its start point.
2. You may assume none of these intervals have the same start point.

Example 1:

Input: [[1,2]]

Output: [-1]

Explanation: There is only one interval in the collection, so it outputs -1.

Example 2:

Input: [[3,4], [2,3], [1,2]]

Output: [-1, 0, 1]

Explanation: There is no satisfied "right" interval for [3,4].

For [2,3], the interval [3,4] has minimum-"right" start point;

For [1,2], the interval [2,3] has minimum-"right" start point.

Example 3:

Input: [[1,4], [2,3], [3,4]]

Output: [-1, 2, -1]

Explanation: There is no satisfied "right" interval for [1,4] and [3,4].

For [2,3], the interval [3,4] has minimum-"right" start point.

```cpp
class Solution {
public:
    vector<int> findRightInterval(vector<vector<int>>& intervals) {
        map<int, int> mp;
        vector<int> res;
        int n = intervals.size();
        for (int i = 0; i < n; ++i)
            mp[intervals[i][0]] = i;
        for (auto in : intervals) {
            auto pos = mp.lower_bound(in[1]);
            res.push_back(pos == mp.end() ? -1 : pos->second);
        }
        return res;
    }
};
```

437. Path Sum III

Easy

You are given a binary tree in which each node contains an integer value.

Find the number of paths that sum to a given value.

The path does not need to start or end at the root or a leaf, but it must go downwards (traveling only from parent nodes to child nodes).

The tree has no more than 1,000 nodes and the values are in the range -1,000,000 to 1,000,000.

Example:

```
root = [10,5,-3,3,2,null,11,3,-2,null,1], sum = 8



      10

     /  \

    5    -3

   / \     \

  3   2    11

 / \   \

3  -2   1



Return 3. The paths that sum to 8 are:



1.  5 -> 3

2.  5 -> 2 -> 1

3. -3 -> 11
```

```cpp
class Solution {
public:
    int pathSum(TreeNode* root, int sum) {
        return dfs(root, sum, 0);
    }

private:
    unordered_map<int, int> mp;

    int dfs(TreeNode *root, int sum, int pre) {
        if (!root) return 0;
        int cur = root->val + pre;
        int res = (cur == sum) + (mp.count(cur - sum) ?
                                  mp[cur - sum] : 0);
        mp[cur]++;
        res += dfs(root->left, sum, cur) + dfs(root->right, sum, cur);
        mp[cur]--;
        return res;
    }
};
```

438. Find All Anagrams in a String

Medium

Given a string **s** and a **non-empty** string **p**, find all the start indices of **p**'s anagrams in **s**.

Strings consists of lowercase English letters only and the length of both strings **s** and **p** will not be larger than 20,100.

The order of output does not matter.

Example 1:

Input:

s: "cbaebabacd" p: "abc"

Output:

[0, 6]

Explanation:

The substring with start index = 0 is "cba", which is an anagram of "abc".

The substring with start index = 6 is "bac", which is an anagram of "abc".

Example 2:

Input:

s: "abab" p: "ab"

Output:

[0, 1, 2]

Explanation:

The substring with start index = 0 is "ab", which is an anagram of "ab".

The substring with start index = 1 is "ba", which is an anagram of "ab".

The substring with start index = 2 is "ab", which is an anagram of "ab".

```cpp
class Solution {
public:
    vector<int> findAnagrams(string s, string p) {
        int n = s.length(), m = p.length();
        vector<int> res;
        unordered_map<char, int> m1, m2;
        for (auto &c : p) m1[c]++;
        int cnt = 0, i = 0, j = 0;
        while(i <= j) {
            while (j < n && cnt < m) {
                if (m1.count(s[j]) && m1[s[j]] >= ++m2[s[j]]) {
                    cnt++;
                }
                j++;
            }
            if (cnt == m && j-i == m) res.push_back(i);
            if (m1.count(s[i]) && m1[s[i]] >= m2[s[i]]--) {
                cnt--;
            }
            i++;
        }
        return res;
    }
};
```

440. K-th Smallest in Lexicographical Order

Hard

Given integers `n` and `k`, find the lexicographically k-th smallest integer in the range from `1` to `n`.

Note: $1 \leq k \leq n \leq 10^9$.

Example:

Input:

n: 13 k: 2

Output:

10

Explanation:

The lexicographical order is [1, 10, 11, 12, 13, 2, 3, 4, 5, 6, 7, 8, 9], so the second smallest number is 10.

```cpp
class Solution {
public:
    int findKthNumber(int n, int k) {
        long cur = 1, i = 1;
        while (i != k) {
            long cnt = getCount(cur, n);
            if (i + cnt <= k) {
                i += cnt;
                ++cur;
            }
            else {
                i += 1;
                cur = cur * 10;
            }
        }
        return static_cast<int> (cur);
    }

private:
    //返回以 cur 为前缀的个数
    long getCount(long cur, long n) {
        long cnt = 0, next = cur + 1;
        while (cur <= n) {
            cnt += min(n + 1, next) - cur;
            cur *= 10;
            next *= 10;
        }
        return cnt;
    }
};
```

441. Arranging Coins

Easy

You have a total of *n* coins that you want to form in a staircase shape, where every *k*-th row must have exactly *k* coins.

Given *n*, find the total number of **full** staircase rows that can be formed.

n is a non-negative integer and fits within the range of a 32-bit signed integer.

Example 1:

```
n = 5

The coins can form the following rows:

¤

¤ ¤

¤ ¤

Because the 3rd row is incomplete, we return 2.
```

Example 2:

```
n = 8

The coins can form the following rows:

¤

¤ ¤

¤ ¤ ¤

¤ ¤

Because the 4th row is incomplete, we return 3.
```

```
class Solution {
public:
    int arrangeCoins(int n) {
        return floor(-0.5+sqrt((double)2*n+0.25));
    }
};
```

442. Find All Duplicates in an Array

Medium

Given an array of integers, 1 ≤ a[i] ≤ *n* (*n* = size of array), some elements appear **twice** and others appear **once**.

Find all the elements that appear **twice** in this array.

Could you do it without extra space and in O(*n*) runtime?

Example:

Input:

[4,3,2,7,8,2,3,1]

Output:

[2,3]

```cpp
class Solution {
public:
    vector<int> findDuplicates(vector<int>& nums) {
        int n = nums.size();
        for (int i = 0; i < n; i++) {
            while (i != nums[i]-1 && nums[i] != nums[nums[i]-1]) {
                swap(nums[i], nums[nums[i]-1]);
            }
        }
        vector<int> res;
        for (int i = 0; i < n; i++) {
            if (i != nums[i]-1) {
                res.push_back(nums[i]);
            }
        }
        return res;
    }
};
```

443. String Compression

Easy

Given an array of characters, compress it **in-place**.

The length after compression must always be smaller than or equal to the original array.

Every element of the array should be a **character** (not int) of length 1.

After you are done **modifying the input array in-place**, return the new length of the array.

Follow up:
Could you solve it using only O(1) extra space?

Example 1:

Input:

["a","a","b","b","c","c","c"]

Output:

Return 6, and the first 6 characters of the input array should be: ["a","2","b","2","c","3"]

Explanation:

"aa" is replaced by "a2". "bb" is replaced by "b2". "ccc" is replaced by "c3".

Example 2:

Input:

["a"]

Output:

Return 1, and the first 1 characters of the input array should be: ["a"]

Explanation:

Nothing is replaced.

Example 3:

Input:

["a","b","b","b","b","b","b","b","b","b","b","b","b"]

Output:

Return 4, and the first 4 characters of the input array should be: ["a","b","1","2"].

Explanation:

Since the character "a" does not repeat, it is not compressed. "bbbbbbbbbbbb" is replaced by "b12".

Notice each digit has it's own entry in the array.

Note:

1. All characters have an ASCII value in `[35, 126]`.
2. `1 <= len(chars) <= 1000`.

```cpp
class Solution {
public:
    int compress(vector<char> &chars) {
        chars.push_back('&');
        char pre = chars[0];
        int cnt = 1, res = 0, j = 0, n = chars.size();
        for (int i = 1; i < n; i++){
            if (chars[i] == pre) cnt++;
            else {
                string s = to_string(cnt);
                chars[j++] = pre;
                res += 1 + (cnt == 1 ? 0 : s.length());
                if (cnt != 1) {
                   for (auto &c : s) chars[j++] = c;
                }
                pre = chars[i];
                cnt = 1;
            }
        }
        return res;
    }
};
```

445. Add Two Numbers II

Medium

You are given two **non-empty** linked lists representing two non-negative integers. The most significant digit comes first and each of their nodes contain a single digit. Add the two numbers and return it as a linked list.

You may assume the two numbers do not contain any leading zero, except the number 0 itself.

Follow up:
What if you cannot modify the input lists? In other words, reversing the lists is not allowed.

Example:

```
Input: (7 -> 2 -> 4 -> 3) + (5 -> 6 -> 4)

Output: 7 -> 8 -> 0 -> 7
```

```
class Solution {
public:
    ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
        int n1 = count_len(l1), n2 = count_len(l2), diff = abs(n1 -
n2);
        if (n1 < n2) swap(l1, l2);
        ListNode *dummy = new ListNode(0), *cur = dummy, *right = cur;
        while (diff-- > 0) {
            cur->next = new ListNode(l1->val);
            if (l1->val != 9) right = cur->next;
            cur = cur->next;
            l1 = l1->next;
        }
        while (l1) {
            int val = l1->val + l2->val;
            if (val > 9) {
                val %= 10;
                ++right->val;
                while (right->next) {
                    right->next->val = 0;
                    right = right->next;
                }
                right = cur;
            }
            cur->next = new ListNode(val);
            if (val != 9) right = cur->next;
            cur = cur->next;
            l1 = l1->next;
            l2 = l2->next;
        }
        return (dummy->val == 1) ? dummy : dummy->next;
    }

private:
    int count_len(ListNode *l) {
        int len = 0;
        while (l) {
            ++len;
            l = l->next;
        }
        return len;
    }
};
```

446. Arithmetic Slices II - Subsequence

Hard

A sequence of numbers is called arithmetic if it consists of at least three elements and if the difference between any two consecutive elements is the same.

For example, these are arithmetic sequences:

```
1, 3, 5, 7, 9

7, 7, 7, 7

3, -1, -5, -9
```

The following sequence is not arithmetic.

```
1, 1, 2, 5, 7
```

A zero-indexed array A consisting of N numbers is given. A **subsequence** slice of that array is any sequence of integers (P_0, P_1, ..., P_k) such that $0 \le P_0 < P_1 < ... < P_k < N$.

A **subsequence** slice (P_0, P_1, ..., P_k) of array A is called arithmetic if the sequence A[P_0], A[P_1], ..., A[P_{k-1}], A[P_k] is arithmetic. In particular, this means that $k \ge 2$.

The function should return the number of arithmetic subsequence slices in the array A.

The input contains N integers. Every integer is in the range of -2^{31} and $2^{31}-1$ and $0 \le N \le 1000$. The output is guaranteed to be less than $2^{31}-1$.

Example:

```
Input: [2, 4, 6, 8, 10]

Output: 7

Explanation:

All arithmetic subsequence slices are:

[2,4,6]

[4,6,8]

[6,8,10]

[2,4,6,8]

[4,6,8,10]
```

```
[2,4,6,8,10]

[2,6,10]
```

```
class Solution {
public:
    int numberOfArithmeticSlices(vector<int>& A) {
        if (A.size() < 3) return 0;
        int n = A.size();
        vector<unordered_map<long long, int>> dp(n);
        /*unordered_set<int> s(A.begin(), A.end());*/
        int res = 0;
        for (int i = 1; i < n; ++i) {
            for (int j = 0; j < i; ++j) {
                auto delta = (long long)A[i] - (long long)A[j];
                int tmp = dp[j].count(delta) ? dp[j][delta] : 0;
                res += tmp;
                /*if (s.count(A[i]+delta))*/
                    dp[i][delta] += 1+tmp;
            }
        }
        return res;
    }
};
```

447. Number of Boomerangs

Easy

Given *n* points in the plane that are all pairwise distinct, a "boomerang" is a tuple of points `(i, j, k)` such that the distance between `i` and `j` equals the distance between `i` and `k` (**the order of the tuple matters**).

Find the number of boomerangs. You may assume that *n* will be at most **500** and coordinates of points are all in the range **[-10000, 10000]** (inclusive).

Example:

Input:

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

Output:

2

Explanation:

The two boomerangs are **[[1,0],[0,0],[2,0]]** and **[[1,0],[2,0],[0,0]]**

```cpp
class Solution {
public:
    int numberOfBoomerangs(vector<vector<int>>& points) {
        int n = points.size(),res = 0;
        for (int i = 0; i < n; ++i) {
            unordered_map<int, int> m;
            for (int j = 0; j < n; ++j) {
                int a = points[i][0] - points[j][0];
                int b = points[i][1] - points[j][1];
                ++m[a*a+b*b];
            }
            for (auto &pii : m) {
                res += pii.second * (pii.second - 1);
            }
        }
        return res;
    }
};
```

448. Find All Numbers Disappeared in an Array

Easy

Given an array of integers where 1 ≤ a[i] ≤ *n* (*n* = size of array), some elements appear twice and others appear once.

Find all the elements of [1, *n*] inclusive that do not appear in this array.

Could you do it without extra space and in O(*n*) runtime? You may assume the returned list does not count as extra space.

Example:

Input:

[4,3,2,7,8,2,3,1]

Output:

[5,6]

```cpp
class Solution {
public:
    vector<int> findDisappearedNumbers(vector<int>& nums) {
        int n = nums.size();
        for (int i = 0; i < n; i++) {
            while (i != nums[i]-1 && nums[i] != nums[nums[i]-1]) {
                swap(nums[i], nums[nums[i]-1]);
            }
        }
        vector<int> res;
        for (int i = 0; i < n; i++) {
            if (i != nums[i]-1) {
                res.push_back(i+1);
            }
        }
        return res;
    }
};
```

```cpp
class Solution {
public:
    vector<int> findDisappearedNumbers(vector<int>& nums) {
        int n = nums.size();
        for(int i = 0; i < n; i++) {
            int m = abs(nums[i])-1; // index start from 0
            nums[m] = nums[m] > 0 ? -nums[m] : nums[m];
        }
        vector<int> res;
        for(int i = 0; i < n; i++) {
            if (nums[i] > 0) res.push_back(i+1);
        }
        return res;
    }
};
```

449. Serialize and Deserialize BST

Medium

Serialization is the process of converting a data structure or object into a sequence of bits so that it can be stored in a file or memory buffer, or transmitted across a network connection link to be reconstructed later in the same or another computer environment.

Design an algorithm to serialize and deserialize a **binary search tree**. There is no restriction on how your serialization/deserialization algorithm should work. You just need to ensure that a binary search tree can be serialized to a string and this string can be deserialized to the original tree structure.

The encoded string should be as compact as possible.

Note: Do not use class member/global/static variables to store states. Your serialize and deserialize algorithms should be stateless.

```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Codec {
public:
    string serialize(TreeNode *root) {
        ostringstream out;
        serialize(root, out);
        return out.str();
    }

    TreeNode *deserialize(string data) {
        istringstream in(data);
        return deserialize(in);
    }

private:
    void serialize(TreeNode *root, ostringstream &out) {
        if (!root) {
            out << "# ";
            return;
        }
        out << root->val << ' ';
        serialize(root->left, out);
        serialize(root->right, out);
    }

    TreeNode* deserialize(istringstream &in) {
        string val;
        in >> val;
        if (val == "#") return nullptr;
        TreeNode* root = new TreeNode(stoi(val));
        root->left = deserialize(in);
        root->right = deserialize(in);
        return root;
    }
};
```

450. Delete Node in a BST

Medium

Given a root node reference of a BST and a key, delete the node with the given key in the BST. Return the root node reference (possibly updated) of the BST.

Basically, the deletion can be divided into two stages:

1. Search for a node to remove.
2. If the node is found, delete the node.

Note: Time complexity should be O(height of tree).

Example:

```
root = [5,3,6,2,4,null,7]

key = 3



    5

   / \

  3   6

 / \   \

2   4   7

Given key to delete is 3. So we find the node with value 3 and delete it.

One valid answer is [5,4,6,2,null,null,7], shown in the following BST.

    5

   / \

  4   6

 /     \

2       7
```

```
Another valid answer is [5,2,6,null,4,null,7].

    5

   / \

  2   6

   \   \

    4   7
```

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    TreeNode *deleteNode(TreeNode* root, int key) {
        if (!root) return nullptr;
        if (root->val == key) {
            TreeNode *t = root->right;
            if (t == nullptr) {
                return root->left;
            }
            while (t->left) t = t->left;
            root->val = t->val;
            root->right = deleteNode(root->right, t->val);
        }
        else if (root->val > key && root->left)
           root->left = deleteNode(root->left, key);
        else if (root->val < key && root->right)
            root->right = deleteNode(root->right, key);
        return root;
    }
};
```

451. Sort Characters By Frequency

Medium

Given a string, sort it in decreasing order based on the frequency of characters.

Example 1:

Input:

"tree"

Output:

"eert"

Explanation:

'e' appears twice while 'r' and 't' both appear once.

So 'e' must appear before both 'r' and 't'. Therefore "eetr" is also a valid answer.

Example 2:

Input:

"cccaaa"

Output:

"cccaaa

Explanation:

Both 'c' and 'a' appear three times, so "aaaccc" is also a valid answer.

Note that "cacaca" is incorrect, as the same characters must be together.

Example 3:

Input:

"Aabb"

Output:

"bbAa"

Explanation:

"bbaA" is also a valid answer, but "Aabb" is incorrect.

Note that 'A' and 'a' are treated as two different characters.

```
class Solution {
public:
    string frequencySort(string s) {
        string res;
        unordered_map<char, int> m;
        for (auto &c : s)  m[c]++;
        set<pair<int, char>> st;
        for (auto &i : m)  st.insert({i.second, i.first});
        for (auto it = st.rbegin(); it != st.rend(); it++)
             res += string(it->first, it->second);
        return res;
    }
};
```

452. Minimum Number of Arrows to Burst Balloons

Medium

There are a number of spherical balloons spread in two-dimensional space. For each balloon, provided input is the start and end coordinates of the horizontal diameter. Since it's horizontal, y-coordinates don't matter and hence the x-coordinates of start and end of the diameter suffice. Start is always smaller than end. There will be at most 10^4 balloons.

An arrow can be shot up exactly vertically from different points along the x-axis. A balloon with x_{start} and x_{end} bursts by an arrow shot at x if $x_{start} \leq x \leq x_{end}$. There is no limit to the number of arrows that can be shot. An arrow once shot keeps travelling up infinitely. The problem is to find the minimum number of arrows that must be shot to burst all balloons.

Example:

Input:

[[10,16], [2,8], [1,6], [7,12]]

Output:

2

Explanation:

One way is to shoot one arrow for example at x = 6 (bursting the balloons [2,8] and [1,6]) and another arrow at x = 11 (bursting the other two balloons).

```cpp
class Solution {
public:
    int findMinArrowShots(vector<vector<int>>& points) {
        if (points.empty()) return 0;
        sort(points.begin(), points.end(), [](const vector<int> &lhs,
const vector<int> &rhs) {
            return lhs[0] != rhs[0] ? lhs[0] < rhs[0]
                                    : lhs[1] < rhs[1];
        });
        int res = 1, low = INT_MIN, high = INT_MAX;
        for (auto &pnt : points) {
            if (pnt[0] > high) {
                low = pnt[0], high = pnt[1];
                res++;
            }
            high = min(high, pnt[1]);
        }
        return res;
    }
};
```

453. Minimum Moves to Equal Array Elements

Easy

Given a **non-empty** integer array of size *n*, find the minimum number of moves required to make all array elements equal, where a move is incrementing *n* - 1 elements by 1.

Example:

Input:

[1,2,3]

Output:

3

Explanation:

Only three moves are needed (remember each move increments two elements):

[1,2,3]   =>   [2,3,3]   =>   [3,4,3]   =>   [4,4,4]

```
class Solution {
public:
    int minMoves(vector<int>& nums) {
        long long Min = nums[0], sum = 0;
        for (const auto &i : nums) {
            if (i < Min) Min = i;
            sum += i;
        }
        return sum - Min*nums.size();
    }
};
```

454. 4Sum II

Medium

Given four lists A, B, C, D of integer values, compute how many tuples `(i, j, k, l)` there are such that `A[i] + B[j] + C[k] + D[l]` is zero.

To make problem a bit easier, all A, B, C, D have same length of N where $0 \le N \le 500$. All integers are in the range of -2^{28} to $2^{28} - 1$ and the result is guaranteed to be at most $2^{31} - 1$.

Example:

Input:

A = [1, 2]

B = [-2,-1]

C = [-1, 2]

D = [0, 2]

Output:

2

Explanation:

The two tuples are:

1. (0, 0, 0, 1) -> A[0] + B[0] + C[0] + D[1] = 1 + (-2) + (-1) + 2 = 0

2. (1, 1, 0, 0) -> A[1] + B[1] + C[0] + D[0] = 2 + (-1) + (-1) + 0 = 0

```cpp
class Solution {
public:
    int fourSumCount(vector<int>& A, vector<int>& B, vector<int>& C,
vector<int>& D) {
        unordered_map<int, int> m;
        for(auto a : A) {
            for(auto b : B) {
                ++m[a+b];
            }
        }
        int res = 0;
        for(auto c : C) {
            for(auto d : D) {
                auto it = m.find(-c-d);
                if (it != m.end()) {
                    res += it->second;
                }
            }
        }
        return res;
    }
};
```

455. Assign Cookies

Easy

Assume you are an awesome parent and want to give your children some cookies. But, you should give each child at most one cookie. Each child i has a greed factor g_i, which is the minimum size of a cookie that the child will be content with; and each cookie j has a size s_j. If $s_j >= g_i$, we can assign the cookie j to the child i, and the child i will be content. Your goal is to maximize the number of your content children and output the maximum number.

Note:
You may assume the greed factor is always positive.
You cannot assign more than one cookie to one child.

Example 1:

Input: [1,2,3], [1,1]

Output: 1

Explanation: You have 3 children and 2 cookies. The greed factors of 3 children are 1, 2, 3.

And even though you have 2 cookies, since their size is both 1, you could only make the child whose greed factor is 1 content.

You need to output 1.

Example 2:

Input: [1,2], [1,2,3]

Output: 2

Explanation: You have 2 children and 3 cookies. The greed factors of 2 children are 1, 2.

You have 3 cookies and their sizes are big enough to gratify all of the children,

You need to output 2.

```cpp
class Solution {
public:
    int findContentChildren(vector<int>& g, vector<int>& s) {
        sort(g.begin(), g.end());
        sort(s.begin(), s.end());
        int i = 0, j = 0, n = g.size(), m = s.size();
        int res = 0;
        while (i < n && j < m) {
            if (g[i] <= s[j++]) {
                res++;
                i++;
            }
        }
        return res;
    }
};
```

456. 132 Pattern

Medium

Given a sequence of n integers $a_1, a_2, ..., a_n$, a 132 pattern is a subsequence a_i, a_j, a_k such that **i** < **j** < **k** and $a_i < a_k < a_j$. Design an algorithm that takes a list of n numbers as input and checks whether there is a 132 pattern in the list.

Note: n will be less than 15,000.

Example 1:

Input: [1, 2, 3, 4]

Output: False

Explanation: There is no 132 pattern in the sequence.

Example 2:

Input: [3, 1, 4, 2]

Output: True

Explanation: There is a 132 pattern in the sequence: [1, 4, 2].

Example 3:

Input: [-1, 3, 2, 0]

Output: True

Explanation: There are three 132 patterns in the sequence: [-1, 3, 2], [-1, 3, 0] and [-1, 2, 0].

```cpp
class Solution {
public:
    bool find132pattern(vector<int>& nums) {
        int third = INT_MIN;
        stack<int> stk; //second, all of it is bigger than Third
        for (int i = nums.size() - 1; i >= 0; --i) {
            if (nums[i] < third) return true;
            while (!stk.empty() && nums[i] > stk.top()) {
                third = stk.top();
                stk.pop();
            }
            stk.push(nums[i]);
        }
        return false;
    }
};
```

457. Circular Array Loop

Medium

You are given a **circular** array `nums` of positive and negative integers. If a number *k* at an index is positive, then move forward *k* steps. Conversely, if it's negative (-*k*), move backward *k* steps. Since the array is circular, you may assume that the last element's next element is the first element, and the first element's previous element is the last element.

Determine if there is a loop (or a cycle) in `nums`. A cycle must start and end at the same index and the cycle's length > 1. Furthermore, movements in a cycle must all follow a single direction. In other words, a cycle must not consist of both forward and backward movements.

Example 1:

Input: [2,-1,1,2,2]

Output: true

Explanation: There is a cycle, from index 0 -> 2 -> 3 -> 0. The cycle's length is 3.

Example 2:

Input: [-1,2]

Output: false

Explanation: The movement from index 1 -> 1 -> 1 ... is not a cycle, because the cycle's length is 1. By definition the cycle's length must be greater than 1.

Example 3:

Input: [-2,1,-1,-2,-2]

Output: false

Explanation: The movement from index 1 -> 2 -> 1 -> ... is not a cycle, because movement from index 1 -> 2 is a forward movement, but movement from index 2 -> 1 is a backward movement. All movements in a cycle must follow a single direction.

Note:

1. $-1000 \le \text{nums}[i] \le 1000$
2. $\text{nums}[i] \neq 0$
3. $1 \le \text{nums.length} \le 5000$

458. Poor Pigs

Hard

There are 1000 buckets, one and only one of them is poisonous, while the rest are filled with water. They all look identical. If a pig drinks the poison it will die within 15 minutes. What is the minimum amount of pigs you need to figure out which bucket is poisonous within one hour?

Answer this question, and write an algorithm for the general case.

General case:

If there are `n` buckets and a pig drinking poison will die within `m` minutes, how many pigs (`x`) you need to figure out the **poisonous** bucket within `p` minutes? There is exactly one bucket with poison.

Note:

1. A pig can be allowed to drink simultaneously on as many buckets as one would like, and the feeding takes no time.
2. After a pig has instantly finished drinking buckets, there has to be a **cool down time** of *m* minutes. During this time, only observation is allowed and no feedings at all.
3. Any given bucket can be sampled an infinite number of times (by an unlimited number of pigs).

```
class Solution {
public:
    int poorPigs(int buckets, int minutesToDie, int minutesToTest) {
        int states = minutesToTest / minutesToDie + 1;
        return ceil(log(buckets) / log(states));
    }
};
```

459. Repeated Substring Pattern

Easy

Given a non-empty string check if it can be constructed by taking a substring of it and appending multiple copies of the substring together. You may assume the given string consists of lowercase English letters only and its length will not exceed 10000.

Example 1:

Input: "abab"

Output: True

Explanation: It's the substring "ab" twice.

Example 2:

Input: "aba"

Output: False

Example 3:

Input: "abcabcabcabc"

Output: True

Explanation: It's the substring "abc" four times. (And the substring "abcabc" twice.)

```cpp
class Solution {
public:
    bool repeatedSubstringPattern(string s) {
        int n = s.size();
        vector<int> f(n, 0);
        for(int i = 1, len = 0; i < n;) {
            if (s[i] == s[len]) f[i++] = ++len;
            else if (len) len = f[len-1];
            else f[i++] = 0;
        }
        return f[n-1] && n % (n-f[n-1]) == 0;
    }
};
```

```cpp
class Solution {
public:
    bool repeatedSubstringPattern(string s) {
        int n = s.length();
        for (int i = 1; i <= n/2; i++) {
            if (n % i != 0) continue;
            int cnt = n / i;
            string a, b = s.substr(0, i);
            while (cnt--) a += b;
            if (a == s) return true;
        }
        return false;
    }
};
```

460. LFU Cache

Hard

Design and implement a data structure for Least Frequently Used (LFU) cache. It should support the following operations: `get` and `put`.

`get(key)` - Get the value (will always be positive) of the key if the key exists in the cache, otherwise return -1.
`put(key, value)` - Set or insert the value if the key is not already present. When the cache reaches its capacity, it should invalidate the least frequently used item before inserting a new item. For the purpose of this problem, when there is a tie (i.e., two or more keys that have the same frequency), the least **recently** used key would be evicted.

Note that the number of times an item is used is the number of calls to the `get` and `put` functions for that item since it was inserted. This number is set to zero when the item is removed.

Follow up:
Could you do both operations in **O(1)** time complexity?

Example:

```
LFUCache cache = new LFUCache( 2 /* capacity */ );

cache.put(1, 1);

cache.put(2, 2);

cache.get(1);            // returns 1

cache.put(3, 3);       // evicts key 2

cache.get(2);            // returns -1 (not found)

cache.get(3);            // returns 3.

cache.put(4, 4);       // evicts key 1.

cache.get(1);            // returns -1 (not found)

cache.get(3);            // returns 3

cache.get(4);            // returns 4
```

```
class LFUCache {
private:
    struct LRU {
        LRU(int cnt) : cnt(cnt) {}
        int cnt;
        list<pair<int, int>> myList;
        unordered_map<int, list<pair<int, int>>::iterator> mp;
    };
public:
    LFUCache(int capacity) : capacity(capacity){}

    int get(int key) {
        if (!capacity ||!LRUCache.count(key)) return -1;
        auto lru = LRUCache[key], next_lru = next(lru);
        int value = lru->mp[key]->second, cnt = lru->cnt + 1;
        if (next_lru == LRUList.end() || next_lru->cnt != cnt) {
            next_lru = LRUList.emplace(next_lru, cnt);
        }
        deleteKey(key);
        LRUinsertKey(next_lru, key, value);
        return value;
    }

    void put(int key, int value) {
        if (capacity == 0) return;
        if (LRUCache.count(key)) {
            get(key);
            LRUCache[key]->mp[key]->second = value;
            return;
        }
        if (sz++ == capacity) {
            deleteKey(LRUList.front().myList.back().first);
            --sz;
        }
        if (LRUList.front().cnt != 1) {
            LRUList.emplace_front(1);
        }
        LRUinsertKey(LRUList.begin(), key, value);
    }

    void deleteKey(int key) {
        auto lru = LRUCache[key];
        lru->myList.erase(lru->mp[key]);
        lru->mp.erase(key);
```

```
            if (lru->myList.empty()) {
                LRUList.erase(lru);
            }
            LRUCache.erase(key);
        }

        void LRUinsertKey(list<LRU>::iterator lru, int key, int value) {
             lru->myList.push_front({key, value});
             lru->mp[key] = lru->myList.begin();
             LRUCache[key] = lru;
        }

    private:
        int capacity, sz = 0;
        list<LRU> LRUList;
        unordered_map<int, list<LRU>::iterator> LRUCache;
    };

    /**
     * Your LFUCache object will be instantiated and called as such:
     * LFUCache* obj = new LFUCache(capacity);
     * int param_1 = obj->get(key);
     * obj->put(key,value);
     */
```

461. Hamming Distance

Easy

The Hamming distance between two integers is the number of positions at which the corresponding bits are different.

Given two integers `x` and `y`, calculate the Hamming distance.

Note:
0 ≤ `x`, `y` < 2^{31}.

Example:

```
Input: x = 1, y = 4

Output: 2

Explanation:

1   (0 0 0 1)

4   (0 1 0 0)

       ↑   ↑

The above arrows point to positions where the corresponding bits are different.
```

```
class Solution {
public:
    int hammingDistance(int x, int y) {
        //return bitset<32>(x^y).count();
        int res = 0;
        for (int i = 0; i < 32; i++) {
            if ((x&1) != (y&1)) res++;
            x >>= 1;
            y >>= 1;
        }
        return res;
    }
};
```

462. Minimum Moves to Equal Array Elements II

Medium

Given a **non-empty** integer array, find the minimum number of moves required to make all array elements equal, where a move is incrementing a selected element by 1 or decrementing a selected element by 1.

You may assume the array's length is at most 10,000.

Example:

Input:

[1,2,3]

Output:

2

Explanation:

Only two moves are needed (remember each move increments or decrements one element):

[1,2,3] => [2,2,3] => [2,2,2]

```
class Solution {
public:
    int minMoves2(vector<int>& nums) {
        sort(nums.begin(), nums.end());
        long long med = nums[nums.size()/2], res = 0;
        for (const auto &i : nums) res += abs(i-med);
        return res;
    }
};
```

463. Island Perimeter

Easy

You are given a map in form of a two-dimensional integer grid where 1 represents land and 0 represents water.

Grid cells are connected horizontally/vertically (not diagonally). The grid is completely surrounded by water, and there is exactly one island (i.e., one or more connected land cells).

The island doesn't have "lakes" (water inside that isn't connected to the water around the island). One cell is a square with side length 1. The grid is rectangular, width and height don't exceed 100. Determine the perimeter of the island.

Example:

Input:

[[0,1,0,0],

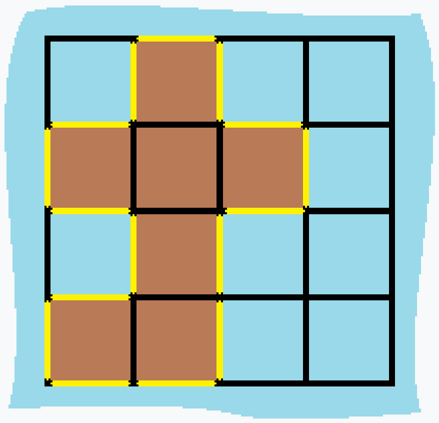
 [1,1,1,0],

 [0,1,0,0],

 [1,1,0,0]]

Output: 16

Explanation: The perimeter is the 16 yellow stripes in the image below:



```
class Solution {
```

```cpp
public:
    int islandPerimeter(vector<vector<int>>& grid) {
        int n = grid.size(), m = grid[0].size();
        int res = 0;
        for (int i = 0;i < n; i++) {
            for (int j = 0; j < m; j++) {
                if (grid[i][j] == 1) {
                    res += 4;
                    if (i && grid[i-1][j] == 1) res -= 2;
                    if (j && grid[i][j-1] == 1) res -= 2;
                }
            }
        }
        return res;
    }
};
```

464. Can I Win

Medium

In the "100 game," two players take turns adding, to a running total, any integer from 1..10. The player who first causes the running total to reach or exceed 100 wins.

What if we change the game so that players cannot re-use integers?

For example, two players might take turns drawing from a common pool of numbers of 1..15 without replacement until they reach a total >= 100.

Given an integer `maxChoosableInteger` and another integer `desiredTotal`, determine if the first player to move can force a win, assuming both players play optimally.

You can always assume that `maxChoosableInteger` will not be larger than 20 and `desiredTotal` will not be larger than 300.

Example

Input:

maxChoosableInteger = 10

desiredTotal = 11

Output:

false

Explanation:

No matter which integer the first player choose, the first player will lose.

The first player can choose an integer from 1 up to 10.

If the first player choose 1, the second player can only choose integers from 2 up to 10.

The second player will win by choosing 10 and get a total = 11, which is >= desiredTotal.

Same with other integers chosen by the first player, the second player will always win.

466. Count The Repetitions

Hard

Define `S = [s,n]` as the string S which consists of n connected strings s. For example, `["abc", 3]` ="abcabcabc".

On the other hand, we define that string s1 can be obtained from string s2 if we can remove some characters from s2 such that it becomes s1. For example, “abc” can be obtained from “abdbec” based on our definition, but it can not be obtained from “acbbe”.

You are given two non-empty strings s1 and s2 (each at most 100 characters long) and two integers $0 \leq n1 \leq 10^6$ and $1 \leq n2 \leq 10^6$. Now consider the strings S1 and S2, where `S1=[s1,n1]` and `S2=[s2,n2]`. Find the maximum integer M such that `[S2,M]` can be obtained from `S1`.

Example:

```
Input:

s1="acb", n1=4

s2="ab", n2=2



Return:

2
```

```cpp
class Solution {
public:
    //same as 418
    int getMaxRepetitions(string s1, int n1, string s2, int n2) {
        int n = s2.length();
        // s2 中 每个字符开头，能在 s1 中 重复多少次
        vector<int> repeatNum(n);
        // s2 中 每个字符开头，尽可能重复，下一次在 s1 中 出现的第一个字符的索引
        vector<int> nextChar(n);

        for (int i = 0; i < n; i++) {
            int cnt = 0, idx = i;
            for (auto c : s1) {
                if (c == s2[idx]) {
                    if (++idx == n) {
                        idx = 0;
                        cnt++;
                    }
                }
            }
            nextChar[i] = idx;
            repeatNum[i] = cnt;
        }

        int res = 0, next = 0;

        while (n1--) {
            res += repeatNum[next];
            next = nextChar[next];
        }
        return res / n2;
    }
};
```

467. Unique Substrings in Wraparound String

Medium

Consider the string `s` to be the infinite wraparound string of "abcdefghijklmnopqrstuvwxyz", so `s` will look like this: "...zabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcd....".

Now we have another string `p`. Your job is to find out how many unique non-empty substrings of `p` are present in `s`. In particular, your input is the string `p` and you need to output the number of different non-empty substrings of `p` in the string `s`.

Note: `p` consists of only lowercase English letters and the size of p might be over 10000.

Example 1:

Input: "a"

Output: 1

Explanation: Only the substring "a" of string "a" is in the string s.

Example 2:

Input: "cac"

Output: 2

Explanation: There are two substrings "a", "c" of string "cac" in the string s.

Example 3:

Input: "zab"

Output: 6

Explanation: There are six substrings "z", "a", "b", "za", "ab", "zab" of string "zab" in the string s.

```cpp
class Solution {
public:
    int findSubstringInWraproundString(string p) {
        vector<int> v(26, 0);
        int len = 0;
        for (int i = 0; i < p.size(); i++) {
            int cur = p[i] - 'a';
            if (i > 0 && p[i - 1] != (cur + 26 - 1) % 26 + 'a') len = 0;
            v[cur] = max(v[cur], ++len);
        }
        return accumulate(v.begin(), v.end(), 0);
    }
};
```

468. Validate IP Address

Medium

Write a function to check whether an input string is a valid IPv4 address or IPv6 address or neither.

IPv4 addresses are canonically represented in dot-decimal notation, which consists of four decimal numbers, each ranging from 0 to 255, separated by dots ("."), e.g.,`172.16.254.1`;

Besides, leading zeros in the IPv4 is invalid. For example, the address `172.16.254.01` is invalid.

IPv6 addresses are represented as eight groups of four hexadecimal digits, each group representing 16 bits. The groups are separated by colons (":"). For example, the address `2001:0db8:85a3:0000:0000:8a2e:0370:7334` is a valid one. Also, we could omit some leading zeros among four hexadecimal digits and some low-case characters in the address to upper-case ones, so `2001:db8:85a3:0:0:8A2E:0370:7334` is also a valid IPv6 address(Omit leading zeros and using upper cases).

However, we don't replace a consecutive group of zero value with a single empty group using two consecutive colons (::) to pursue simplicity. For example, `2001:0db8:85a3::8A2E:0370:7334` is an invalid IPv6 address.

Besides, extra leading zeros in the IPv6 is also invalid. For example, the address `02001:0db8:85a3:0000:0000:8a2e:0370:7334` is invalid.

Note: You may assume there is no extra space or special characters in the input string.

Example 1:

Input: "172.16.254.1"

Output: "IPv4"

Explanation: This is a valid IPv4 address, return "IPv4".

Example 2:

Input: "2001:0db8:85a3:0:0:8A2E:0370:7334"

Output: "IPv6"

Explanation: This is a valid IPv6 address, return "IPv6".

Example 3:

Input: "256.256.256.256"

Output: "Neither"

Explanation: This is neither a IPv4 address nor a IPv6 address.

```cpp
class Solution {
public:
    string validIPAddress(string IP) {
        string s;
        int cnt = 0;
        if (IP.find('.') != string::npos) {
            for (auto &c : IP) {
                if (c == '.' && ++cnt) c = ' ';
                else if (!isdigit(c)) return "Neither";
            }
            if (cnt != 3) return "Neither";
            stringstream ss(IP);
            while(ss >> s) {
                --cnt;
                if (s.empty() || s.length() > 3 || stoi(s) > 255) {
                    return "Neither";
                } else if (s[0] == '0' && s != "0") {
                    return "Neither";
                }
            }
            return cnt == -1 ? "IPv4" : "Neither";
        } else {
            for (auto &c : IP) {
                if (c == ':' && ++cnt)  c = ' ';
                else if (!isalnum(c)) return "Neither";
                else if (isalpha(c)&&(isupper(c)?c > 'F':c > 'f'))
                    return "Neither";
            }
            if (cnt != 7) return "Neither";
            stringstream ss(IP);
            while(ss >> s) {
                --cnt;
               if (s.empty() || s.length() > 4) {
                   return "Neither";
               }
            }
            return cnt == -1 ? "IPv6" : "Neither";
        }
        return "Neither";
    }
};
```

470. Implement Rand10() Using Rand7()

Medium

Given a function `rand7` which generates a uniform random integer in the range 1 to 7, write a function `rand10` which generates a uniform random integer in the range 1 to 10.

Do NOT use system's `Math.random()`.

Example 1:

Input: 1

Output: [7]

Example 2:

Input: 2

Output: [8,4]

Example 3:

Input: 3

Output: [8,1,10]

Note:

1. `rand7` is predefined.
2. Each testcase has one argument: `n`, the number of times that `rand10` is called.

Follow up:

1. What is the expected value for the number of calls to `rand7()` function?
2. Could you minimize the number of calls to `rand7()`?

```
// The rand7() API is already defined for you.
// int rand7();
// @return a random integer in the range 1 to 7

class Solution {
public:
    int rand10() {
        while (1) {
            int num = 7 * (rand7() - 1) + (rand7() - 1);
            int ret = 1 + num/4;
            if (ret < 11) return ret;
        }
    }
};
```

472. Concatenated Words

Hard

Given a list of words (**without duplicates**), please write a program that returns all concatenated words in the given list of words.
A concatenated word is defined as a string that is comprised entirely of at least two shorter words in the given array.

Example:

```
Input: ["cat","cats","catsdogcats","dog","dogcatsdog","hippopotamuses","rat","ratcatdogcat"]

Output: ["catsdogcats","dogcatsdog","ratcatdogcat"]



Explanation: "catsdogcats" can be concatenated by "cats", "dog" and "cats";
  "dogcatsdog" can be concatenated by "dog", "cats" and "dog";
"ratcatdogcat" can be concatenated by "rat", "cat", "dog" and "cat".
```

Note:

1. The number of elements of the given array will not exceed `10,000`
2. The length sum of elements in the given array will not exceed `600,000`.
3. All the input string will only include lower case letters.
4. The returned elements order does not matter.

```cpp
class Trie {
public:
    void insert(const string &s) {
        Trie *p = this;
        for (auto c : s) {
            if (!p->mp.count(c)) {
                p->mp[c] = new Trie();
            }
            p = p->mp[c];
        }
        p->isWord = true;
    }
    bool isWord = false;
    unordered_map<char, Trie*> mp;
};


class Solution {
public:
    vector<string> findAllConcatenatedWordsInADict(vector<string>& words) {
        root = new Trie();
        for (const auto &s : words) root->insert(s);
        vector<string> res;
        for (const auto &s : words) {
            if (isConcatenatedWord(0, s.length(), s, root, 0)) {
                res.push_back(s);
            }
        }
        return res;
    }
private:
    Trie *root;

    bool isConcatenatedWord(int i, int n, const string &s, Trie *p, int cnt) {
        if (i >= n || !p->mp.count(s[i])) return false;
        p = p->mp[s[i]];
        if (p->isWord) {
            if (i == n-1) return cnt > 0;
            if (isConcatenatedWord(i+1, n, s, root, cnt+1)) return true;
        }
        return isConcatenatedWord(i+1, n, s, p, cnt);
    }
};
```

473. Matchsticks to Square

Medium

Remember the story of Little Match Girl? By now, you know exactly what matchsticks the little match girl has, please find out a way you can make one square by using up all those matchsticks. You should not break any stick, but you can link them up, and each matchstick must be used **exactly** one time.

Your input will be several matchsticks the girl has, represented with their stick length. Your output will either be true or false, to represent whether you could make one square using all the matchsticks the little match girl has.

Example 1:

Input: [1,1,2,2,2]

Output: true

Explanation: You can form a square with length 2, one side of the square came two sticks with length 1.

Example 2:

Input: [3,3,3,3,4]

Output: false

Explanation: You cannot find a way to form a square with all the matchsticks.

Note:

1. The length sum of the given matchsticks is in the range of `0` to `10^9`.
2. The length of the given matchstick array will not exceed `15`.

```cpp
class Solution {
public:
    bool makesquare(vector<int>& nums) {
        if (nums.size() < 4) return false;
        int sum = accumulate(nums.begin(), nums.end(), 0);
        if (sum % 4 != 0) return false;
        int n = nums.size(), all = (1 << n) - 1, target = sum / 4;
        vector<int> masks, validHalf(1 << n, false);
        for (int i = 0; i <= all; ++i) {
            int curSum = 0;
            for (int j = 0; j < n; ++j) {
                if ((i >> j) & 1) curSum += nums[j];
            }
            if (curSum == target) {
                for (auto &mask : masks) {
                    if ((mask & i) != 0) continue;
                    int half = mask | i;
                    validHalf[half] = true;
                    if (validHalf[all ^ half]) return true;
                }
                masks.push_back(i);
            }
        }
        return false;
    }
};
```

```
class Solution {
public:
    bool makesquare(vector<int>& nums) {
        if (nums.size() < 4) return false;
        int sum = accumulate(nums.begin(), nums.end(), 0);
        if (sum % 4 != 0) return false;
        sort(nums.begin(), nums.end());
        return dfs(nums, nums.size()-1, vector<int> (4, sum/4));
    }
private:
    bool dfs(vector<int> &nums, int cur, vector<int> lens) {
        if (cur < 0) return true;
        for (int i = 0; i < 4; ++i) {
            if (lens[i] - nums[cur] >= 0) {
                lens[i] -= nums[cur];
                if (dfs(nums, cur-1, lens)) return true;
                lens[i] += nums[cur];
            }
        }
        return false;
    }
};
```

474. Ones and Zeroes

Medium

In the computer world, use restricted resource you have to generate maximum benefit is what we always want to pursue.

For now, suppose you are a dominator of **m** `0s` and **n** `1s` respectively. On the other hand, there is an array with strings consisting of only `0s` and `1s`.

Now your task is to find the maximum number of strings that you can form with given **m** `0s` and **n** `1s`. Each `0` and `1` can be used at most **once**.

Note:

1. The given numbers of `0s` and `1s` will both not exceed `100`
2. The size of given string array won't exceed `600`.

Example 1:

Input: Array = {"10", "0001", "111001", "1", "0"}, m = 5, n = 3

Output: 4

Explanation: This are totally 4 strings can be formed by the using of 5 0s and 3 1s, which are “10,”0001”,”1”,”0”

Example 2:

Input: Array = {"10", "0", "1"}, m = 1, n = 1

Output: 2

Explanation: You could form "10", but then you'd have nothing left. Better form "0" and "1".

```
class Solution {
public:
    int findMaxForm(vector<string>& strs, int m, int n) {
        vector<vector<int>> dp(m+1,vector<int>(n+1, 0));
        for (auto &s: strs) {
            int ones = count(s.begin(), s.end(), '1'), zeros =
s.size()-ones;
            for (int i = m; i >= zeros; i--)
                for (int j = n; j >= ones; j--)
                    dp[i][j] = max(dp[i][j], dp[i-zeros][j-ones] + 1);
        }
        return dp[m][n];
    }
};
```

475. Heaters

Easy

Winter is coming! Your first job during the contest is to design a standard heater with fixed warm radius to warm all the houses.

Now, you are given positions of houses and heaters on a horizontal line, find out minimum radius of heaters so that all houses could be covered by those heaters.

So, your input will be the positions of houses and heaters seperately, and your expected output will be the minimum radius standard of heaters.

Note:

1. Numbers of houses and heaters you are given are non-negative and will not exceed 25000.
2. Positions of houses and heaters you are given are non-negative and will not exceed 10^9.
3. As long as a house is in the heaters' warm radius range, it can be warmed.
4. All the heaters follow your radius standard and the warm radius will the same.

Example 1:

Input: [1,2,3],[2]

Output: 1

Explanation: The only heater was placed in the position 2, and if we use the radius 1 standard, then all the houses can be warmed.

Example 2:

Input: [1,2,3,4],[1,4]

Output: 1

Explanation: The two heater was placed in the position 1 and 4. We need to use radius 1 standard, then all the houses can be warmed.

```
class Solution {
public:
    int findRadius(vector<int>& houses, vector<int>& heaters) {
        sort(heaters.begin(), heaters.end());
        int minRadius = 0, n = houses.size();
        for (int i = 0; i < n; ++i) {
            auto larger = lower_bound(heaters.begin(), heaters.end(),
houses[i]);
            int curRadius = INT_MAX;
            if (larger != heaters.end()) curRadius = *larger -
houses[i];
            if (larger != heaters.begin()) {
                auto smaller = larger - 1;
                curRadius = min(curRadius, houses[i] - *smaller);
            }
            minRadius = max(minRadius, curRadius);
        }
        return minRadius;
    }
};
```

476. Number Complement

Easy

Given a positive integer, output its complement number. The complement strategy is to flip the bits of its binary representation.

Note:

1. The given integer is guaranteed to fit within the range of a 32-bit signed integer.
2. You could assume no leading zero bit in the integer’s binary representation.

Example 1:

Input: 5

Output: 2

Explanation: The binary representation of 5 is 101 (no leading zero bits), and its complement is 010. So you need to output 2.

Example 2:

Input: 1

Output: 0

Explanation: The binary representation of 1 is 1 (no leading zero bits), and its complement is 0. So you need to output 0.

```cpp
class Solution {
public:
    int findComplement(int num) {
        for(unsigned i = 1; i <= num; i <<= 1) num ^= i;
        return num;
    }
};
```

```cpp
class Solution {
public:
    int findComplement(int num) {
        int cnt = 0, n = num;
        while (n) {
            n >>= 1;
            cnt++;
        }
        return  pow(2, cnt) - num - 1;
    }
};
```

477. Total Hamming Distance

Medium

The Hamming distance between two integers is the number of positions at which the corresponding bits are different.

Now your job is to find the total Hamming distance between all pairs of the given numbers.

Example:

Input: 4, 14, 2

Output: 6

Explanation: In binary representation, the 4 is 0100, 14 is 1110, and 2 is 0010 (just

showing the four bits relevant in this case). So the answer will be:

HammingDistance(4, 14) + HammingDistance(4, 2) + HammingDistance(14, 2) = 2 + 2 + 2 = 6.

Note:

1. Elements of the given array are in the range of `0` to `10^9`
2. Length of the array will not exceed `10^4`.

```
class Solution {
public:
    int totalHammingDistance(vector<int>& nums) {
        int res = 0, n = nums.size();
        for (int i = 0; i < 32; ++i) {
            int cnt = 0;
            for (auto &num : nums) {
                if (num & (1 << i)) ++cnt;
            }
            res += cnt * (n - cnt);
        }
        return res;
    }
};
```

478. Generate Random Point in a Circle

Medium

110186FavoriteShare

Given the radius and x-y positions of the center of a circle, write a function `randPoint` which generates a uniform random point in the circle.

Note:

1. input and output values are in floating-point.
2. radius and x-y position of the center of the circle is passed into the class constructor.
3. a point on the circumference of the circle is considered to be in the circle.
4. `randPoint` returns a size 2 array containing x-position and y-position of the random point, in that order.

Example 1:

```
Input:

["Solution","randPoint","randPoint","randPoint"]

[[1,0,0],[],[],[]]

Output: [null,[-0.72939,-0.65505],[-0.78502,-0.28626],[-0.83119,-0.19803]]
```

Example 2:

```
Input:

["Solution","randPoint","randPoint","randPoint"]

[[10,5,-7.5],[],[],[]]

Output: [null,[11.52438,-8.33273],[2.46992,-16.21705],[11.13430,-12.42337]]
```

Explanation of Input Syntax:

The input is two lists: the subroutines called and their arguments. `Solution`'s constructor has three arguments, the radius, x-position of the center, and y-position of the center of the circle. `randPoint` has no arguments. Arguments are always wrapped with a list, even if there aren't any.

```cpp
class Solution {
public:
    Solution(double r, double x, double y) :r(r), x(x), y(y) {}

    vector<double> randPoint() {
        double xx, yy;
        do {
            xx = 2*rand1() - 1;
            yy = 2*rand1() - 1;
        } while (xx*xx + yy*yy > 1);
        return vector<double> {r*xx + x, r*yy + y};
    }

private:
    double r, x, y;
    double rand1() {return (double)rand() / RAND_MAX;}
};
```

```
class Solution {
public:
    Solution(double r, double x, double y) :r(r), x(x), y(y) {}

    vector<double> randPoint() {
        double theta = 2 * PI * rand1();
        double rr = sqrt(rand1());
        return vector<double> {x + rr*r*cos(theta),y +
rr*r*sin(theta)};
    }

private:
    const double PI = 3.14159265358979732384626433832795;
    double r, x, y;

    double rand1() {return (double)rand() / RAND_MAX;}
};
```

479. Largest Palindrome Product

Hard

741247FavoriteShare

Find the largest palindrome made from the product of two n-digit numbers.

Since the result could be very large, you should return the largest palindrome mod 1337.

Example:

Input: 2

Output: 987

Explanation: 99 x 91 = 9009, 9009 % 1337 = 987

Note:

The range of n is [1,8].

480. Sliding Window Median

Hard

Median is the middle value in an ordered integer list. If the size of the list is even, there is no middle value. So the median is the mean of the two middle value.

Examples:

`[2,3,4]` , the median is `3`

`[2,3]`, the median is `(2 + 3) / 2 = 2.5`

Given an array *nums*, there is a sliding window of size *k* which is moving from the very left of the array to the very right. You can only see the *k* numbers in the window. Each time the sliding window moves right by one position. Your job is to output the median array for each window in the original array.

For example,
Given *nums* = `[1,3,-1,-3,5,3,6,7]`, and *k* = 3.

```
Window position                Median

---------------               -----

[1  3  -1] -3  5  3  6  7       1

 1 [3  -1  -3] 5  3  6  7      -1

 1  3 [-1  -3  5] 3  6  7      -1

 1  3  -1 [-3  5  3] 6  7       3

 1  3  -1  -3 [5  3  6] 7       5

 1  3  -1  -3  5 [3  6  7]      6
```

Therefore, return the median sliding window as `[1,-1,-1,3,5,6]`.

Note:
You may assume `k` is always valid, ie: `k` is always smaller than input array's size for non-empty array.

```
class Solution {
public:
    vector<double> medianSlidingWindow(vector<int>& nums, int k) {
        vector<double> medians;
        unordered_map<int, int> hash_table;
        priority_queue<int> lo;                                 // max heap
        priority_queue<int, vector<int>, greater<int> > hi;     // min heap
        int i = 0;     // index of current incoming element being processed

        // initialize the heaps
        while (i < k) lo.push(nums[i++]);
        for (int j = 0; j < k / 2; j++) {
            hi.push(lo.top());
            lo.pop();
        }
        while (1) {
        // get median of current window
            medians.push_back(k & 1 ? lo.top()
                                    : ((double)lo.top() + (double)hi.top()) * 0.5);
            if (i >= nums.size()) break;
            int out_num = nums[i - k], in_num = nums[i++];
            int balance = (out_num <= lo.top() ? -1 : 1);

            ++hash_table[out_num];
            // number `in_num` enters window
            if (!lo.empty() && in_num <= lo.top()) {
                balance++;
                lo.push(in_num);
            }
            else {
                balance--;
                hi.push(in_num);
            }
        // re-balance heaps
            if (balance < 0) {          // `lo` needs more valid elements
                lo.push(hi.top());
                hi.pop();
                balance++;
            }
            if (balance > 0) {          // `hi` needs more valid elements
                hi.push(lo.top());
                lo.pop();
                balance--;
            }
```

```
      // remove invalid numbers that should be discarded from heap tops
            while (hash_table[lo.top()]) {
                hash_table[lo.top()]--;
                lo.pop();
            }
            while (!hi.empty() && hash_table[hi.top()]) {
               hash_table[hi.top()]--;
               hi.pop();
            }
        }
        return medians;
    }
};
```

481. Magical String

Medium

A magical string **S** consists of only '1' and '2' and obeys the following rules:

The string **S** is magical because concatenating the number of contiguous occurrences of characters '1' and '2' generates the string **S** itself.

The first few elements of string **S** is the following: **S** = "1221121221221121122……"

If we group the consecutive '1's and '2's in **S**, it will be:

1 22 11 2 1 22 1 22 11 2 11 22 ......

and the occurrences of '1's or '2's in each group are:

1 2 2 1 1 2 1 2 2 1 2 2 ......

You can see that the occurrence sequence above is the **S** itself.

Given an integer N as input, return the number of '1's in the first N number in the magical string **S**.

Note: N will not exceed 100,000.

Example 1:

Input: 6

Output: 3

Explanation: The first 6 elements of magical string S is "12211" and it contains three 1's, so return 3.

```
class Solution {
public:
    int magicalString(int n) {
        if (n == 1) return 1;
        int i = 0;
        string s = "1";
        char cur = '2';
        while (s.length() < n) {
            s += string(s[i++]-'0', cur);
            cur = (cur == '1' ? '2' : '1');
        }
        int res = 0;
        for (int i = 0; i < n-1; ++i) {
            if (s[i] == '1') ++res;
        }
        return res;
    }
};
```

482. License Key Formatting

Easy

You are given a license key represented as a string S which consists only alphanumeric character and dashes. The string is separated into N+1 groups by N dashes.

Given a number K, we would want to reformat the strings such that each group contains *exactly* K characters, except for the first group which could be shorter than K, but still must contain at least one character. Furthermore, there must be a dash inserted between two groups and all lowercase letters should be converted to uppercase.

Given a non-empty string S and a number K, format the string according to the rules described above.

Example 1:

Input: S = "5F3Z-2e-9-w", K = 4

Output: "5F3Z-2E9W"

Explanation: The string S has been split into two parts, each part has 4 characters.

Note that the two extra dashes are not needed and can be removed.

Example 2:

Input: S = "2-5g-3-J", K = 2

Output: "2-5G-3J"

Explanation: The string S has been split into three parts, each part has 2 characters except the first part as it could be shorter as mentioned above.

Note:

1. The length of string S will not exceed 12,000, and K is a positive integer.
2. String S consists only of alphanumerical characters (a-z and/or A-Z and/or 0-9) and dashes(-).
3. String S is non-empty.

```
class Solution {
public:
    string licenseKeyFormatting(string S, int K) {
        string res;
        int cnt = 0;
        for (auto it = S.rbegin(); it != S.rend(); it++)  if (*it !=
'-') {
            if (cnt++ % K == 0 && cnt != 1) res += '-';
            res += toupper(*it);
        }
        reverse(res.begin(), res.end());
        return res;
    }
};
```

483. Smallest Good Base

Hard

For an integer n, we call k>=2 a ***good base*** of n, if all digits of n base k are 1.

Now given a string representing n, you should return the smallest good base of n in string format.

Example 1:

Input: "13"

Output: "3"

Explanation: 13 base 3 is 111.

Example 2:

Input: "4681"

Output: "8"

Explanation: 4681 base 8 is 11111.

Example 3:

Input: "1000000000000000000"

Output: "999999999999999999"

Explanation: 1000000000000000000 base 999999999999999999 is 11.

Note:

1. The range of n is [3, 10^18].
2. The string representing n is always valid and will not have leading zeros.

```
/*
tip
n = k^(m-1) + k ^(m-2) ... * k + 1
n = (1-k^m) / (1-k)
*/

class Solution {
public:
    string smallestGoodBase(string n) {
        using ll = long long int;
        ll num = stoll(n);
        for (int m = log2(num); m >= 1; m--) {
            ll l = 2, r = powl(num, 1.0 / m) + 1;
            while (l < r) {
               ll mid = l + (r-l)/2, sum = 0;
               for (int j = 0; j <= m; ++j) {
                   sum = sum * mid + 1;
               }
               if (sum == num) return to_string(mid);
               else if (sum < num) l = mid + 1;
               else r = mid;
            }
        }
        return "";
    }
};
```

485. Max Consecutive Ones

Easy

Given a binary array, find the maximum number of consecutive 1s in this array.

Example 1:

Input: [1,1,0,1,1,1]

Output: 3

Explanation: The first two digits or the last three digits are consecutive 1s.

    The maximum number of consecutive 1s is 3.

Note:

- The input array will only contain `0` and `1`.
- The length of input array is a positive integer and will not exceed 10,000

```
class Solution {
public:
    int findMaxConsecutiveOnes(vector<int>& nums) {
        int cnt = 0, res = 0;
        for (const auto &i : nums) {
            if (i)  res = max(res, ++cnt);
            else cnt = 0;
        }
        return res;
    }
};
```

486. Predict the Winner

Medium

Given an array of scores that are non-negative integers. Player 1 picks one of the numbers from either end of the array followed by the player 2 and then player 1 and so on. Each time a player picks a number, that number will not be available for the next player. This continues until all the scores have been chosen. The player with the maximum score wins.

Given an array of scores, predict whether player 1 is the winner. You can assume each player plays to maximize his score.

Example 1:

Input: [1, 5, 2]

Output: False

Explanation: Initially, player 1 can choose between 1 and 2.
If he chooses 2 (or 1), then player 2 can choose from 1 (or 2) and 5. If player 2 chooses 5, then player 1 will be left with 1 (or 2).
So, final score of player 1 is 1 + 2 = 3, and player 2 is 5.
Hence, player 1 will never be the winner and you need to return False.

Example 2:

Input: [1, 5, 233, 7]

Output: True

Explanation: Player 1 first chooses 1. Then player 2 have to choose between 5 and 7. No matter which number player 2 choose, player 1 can choose 233.
Finally, player 1 has more score (234) than player 2 (12), so you need to return True representing player1 can win.

Note:

1. 1 <= length of the array <= 20.
2. Any scores in the given array are non-negative integers and will not exceed 10,000,000.
3. If the scores of both players are equal, then player 1 is still the winner.

```cpp
class Solution {
public:
    bool PredictTheWinner(vector<int>& nums) {
        int n = nums.size(), S = 0;
        if (n % 2 == 0) return true;
        dp.resize(n, vector<int>(n, 0));
        for (auto &i : nums) sum.push_back(S += i);
        return 2*f(0, n-1, nums) >= S;
    }
private:
    vector<vector<int>> dp;
    vector<int> sum;
    int get_sum(int i, int j) {
        return sum[j] - (i ? sum[i-1] : 0);
    }

    int f(int i, int j, vector<int> &nums) {
        if (i == j) return nums[i];
        if (dp[i][j]) return dp[i][j];
        return dp[i][j] = max(get_sum(i+1, j) - f(i+1, j, nums)+nums[i],
                              get_sum(i, j-1) - f(i, j-1, nums)+nums[j]);
    }
};
```

488. Zuma Game

Hard

Think about Zuma Game. You have a row of balls on the table, colored red(R), yellow(Y), blue(B), green(G), and white(W). You also have several balls in your hand.

Each time, you may choose a ball in your hand, and insert it into the row (including the leftmost place and rightmost place). Then, if there is a group of 3 or more balls in the same color touching, remove these balls. Keep doing this until no more balls can be removed.

Find the minimal balls you have to insert to remove all the balls on the table. If you cannot remove all the balls, output -1.

Examples:

Input: "WRRBBW", "RB"

Output: -1

Explanation: WRRBBW -> WRR[R]BBW -> WBBW -> WBB[B]W -> WW

Input: "WWRRBBWW", "WRBRW"

Output: 2

Explanation: WWRRBBWW -> WWRR[R]BBWW -> WWBBWW -> WWBB[B]WW -> WWWW -> empty

Input:"G", "GGGGG"

Output: 2

Explanation: G -> G[G] -> GG[G] -> empty

Input: "RBYYBBRRB", "YRBGB"

Output: 3

Explanation: RBYYBBRRB -> RBYY[Y]BBRRB -> RBBBRRB -> RRRB -> B -> B[B] -> BB[B] -> empty

Note:

1. You may assume that the initial row of balls on the table won’t have any 3 or more consecutive balls with the same color.
2. The number of balls on the table won't exceed 20, and the string represents these balls is called "board" in the input.
3. The number of balls in your hand won't exceed 5, and the string represents these balls is called "hand" in the input.
4. Both input strings will be non-empty and only contain characters 'R','Y','B','G','W'.

491. Increasing Subsequences

Medium

Given an integer array, your task is to find all the different possible increasing subsequences of the given array, and the length of an increasing subsequence should be at least 2.

Example:

Input: [4, 6, 7, 7]

Output: [[4, 6], [4, 7], [4, 6, 7], [4, 6, 7, 7], [6, 7], [6, 7, 7], [7,7], [4,7,7]]

Note:

1. The length of the given array will not exceed 15.
2. The range of integer in the given array is [-100,100].
3. The given array may contain duplicates, and two equal integers should also be considered as a special case of increasing sequence.

```cpp
class Solution {
public:
    vector<vector<int>> findSubsequences(vector<int>& nums) {
        vector<vector<int>> res;
        vector<int> path;
        dfs(res, path, nums, 0);
        return res;
    }

    void dfs(vector<vector<int>> &res, vector<int> &path,
vector<int>& nums, int cur) {
        if (path.size() > 1) res.push_back(path);
        unordered_set<int> hash;
        for (int i = cur; i < nums.size(); ++i) {
            if ((path.empty() || nums[i] >= path.back())
&& !hash.count(nums[i])) {
                path.push_back(nums[i]);
                dfs(res, path, nums, i + 1);
                path.pop_back();
                hash.insert(nums[i]);
            }
        }
    }
};
```

492. Construct the Rectangle

Easy

For a web developer, it is very important to know how to design a web page's size. So, given a specific rectangular web page’s area, your job by now is to design a rectangular web page, whose length L and width W satisfy the following requirements:

```
1. The area of the rectangular web page you designed must equal to the given target area.


2. The width W should not be larger than the length L, which means L >= W.


3. The difference between length L and width W should be as small as possible.
```

You need to output the length L and the width W of the web page you designed in sequence.
Example:

Input: 4

Output: [2, 2]

Explanation: The target area is 4, and all the possible ways to construct it are [1,4], [2,2], [4,1].

But according to requirement 2, [1,4] is illegal; according to requirement 3, [4,1] is not optimal compared to [2,2]. So the length L is 2, and the width W is 2.

Note:

1. The given area won't exceed 10,000,000 and is a positive integer
2. The web page's width and length you designed must be positive integers.

```cpp
class Solution {
public:
    vector<int> constructRectangle(int area) {
        int W = sqrt(area);
        while (1) {
            if (area % W == 0) return {area/W, W};
            else W--;
        }
        return {area, 1};
    }
};
```

493. Reverse Pairs

Hard

Given an array `nums`, we call `(i, j)` an ***important reverse pair*** if `i < j` and `nums[i] > 2*nums[j]`.

You need to return the number of important reverse pairs in the given array.

Example1:

Input: [1,3,2,3,1]

Output: 2

Example2:

Input: [2,4,3,5,1]

Output: 3

Note:

1. The length of the given array will not exceed `50,000`.
2. All the numbers in the input array are in the range of 32-bit integer.

494. Target Sum

Medium

You are given a list of non-negative integers, a1, a2, ..., an, and a target, S. Now you have 2 symbols `+` and `-`. For each integer, you should choose one from `+` and `-` as its new symbol.

Find out how many ways to assign symbols to make sum of integers equal to target S.

Example 1:

```
Input: nums is [1, 1, 1, 1, 1], S is 3.

Output: 5

Explanation:



-1+1+1+1+1 = 3

+1-1+1+1+1 = 3

+1+1-1+1+1 = 3

+1+1+1-1+1 = 3

+1+1+1+1-1 = 3



There are 5 ways to assign symbols to make the sum of nums be target 3.
```

Note:

1. The length of the given array is positive and will not exceed 20.
2. The sum of elements in the given array will not exceed 1000.
3. Your output answer is guaranteed to be fitted in a 32-bit integer.

```cpp
class Solution {
public:
    int findTargetSumWays(vector<int>& nums, int S) {
        int sum = 0, n = nums.size();
        for (auto &i : nums) sum += i;
        if ((sum-S) % 2 != 0 || S > sum) return 0;
        int newS = (sum + S) / 2;
        vector<int> dp(newS + 1, 0);
        dp[0] = 1;
        for (int i = 0; i < n; ++i) {
            for (int j = newS; j >= nums[i]; --j) {
                dp[j] += dp[j - nums[i]];
            }
        }
        return dp[newS];
    }
};
```

```
class Solution {
public:
    int findTargetSumWays(vector<int>& nums, int S) {
        return dfs(nums, 0, S);
    }
private:
    int dfs(vector<int>& nums, int cur, long S) {
        if (cur == nums.size()) return S == 0 ? 1 : 0;
        return dfs(nums, cur+1, S+nums[cur])
              + dfs(nums, cur+1, S-nums[cur]);
    }
};
```

495. Teemo Attacking

Medium

In LOL world, there is a hero called Teemo and his attacking can make his enemy Ashe be in poisoned condition. Now, given the Teemo's attacking **ascending** time series towards Ashe and the poisoning time duration per Teemo's attacking, you need to output the total time that Ashe is in poisoned condition.

You may assume that Teemo attacks at the very beginning of a specific time point, and makes Ashe be in poisoned condition immediately.

Example 1:

Input: [1,4], 2

Output: 4

Explanation: At time point 1, Teemo starts attacking Ashe and makes Ashe be poisoned immediately.

This poisoned status will last 2 seconds until the end of time point 2.

And at time point 4, Teemo attacks Ashe again, and causes Ashe to be in poisoned status for another 2 seconds.

So you finally need to output 4.

Example 2:

Input: [1,2], 2

Output: 3

Explanation: At time point 1, Teemo starts attacking Ashe and makes Ashe be poisoned.

This poisoned status will last 2 seconds until the end of time point 2.

However, at the beginning of time point 2, Teemo attacks Ashe again who is already in poisoned status.

Since the poisoned status won't add up together, though the second poisoning attack will still work at time point 2, it will stop at the end of time point 3.

So you finally need to output 3.

Note:

1. You may assume the length of given time series array won't exceed 10000.
2. You may assume the numbers in the Teemo's attacking time series and his poisoning time duration per attacking are non-negative integers, which won't exceed 10,000,000.

```
class Solution {
public:
    int findPoisonedDuration(vector<int>& timeSeries, int duration) {
        int endTime = -1, res = 0;
        for (auto &i : timeSeries) {
            if (i > endTime) {
                res += duration;
                endTime = i + duration;
            }
            else if (i+duration > endTime){
                res += i + duration - endTime;
                endTime = i + duration;
            }
        }
        return res;
    }
};
```

496. Next Greater Element I

Easy

You are given two arrays **(without duplicates)** `nums1` and `nums2` where `nums1`'s elements are subset of `nums2`. Find all the next greater numbers for `nums1`'s elements in the corresponding places of `nums2`.

The Next Greater Number of a number **x** in `nums1` is the first greater number to its right in `nums2`. If it does not exist, output -1 for this number.

Example 1:

Input: nums1 = [4,1,2], **nums2** = [1,3,4,2].

Output: [-1,3,-1]

Explanation:

For number 4 in the first array, you cannot find the next greater number for it in the second array, so output -1.

For number 1 in the first array, the next greater number for it in the second array is 3.

For number 2 in the first array, there is no next greater number for it in the second array, so output -1.

Example 2:

Input: nums1 = [2,4], **nums2** = [1,2,3,4].

Output: [3,-1]

Explanation:

For number 2 in the first array, the next greater number for it in the second array is 3.

For number 4 in the first array, there is no next greater number for it in the second array, so output -1.

Note:

1. All elements in `nums1` and `nums2` are unique.
2. The length of both `nums1` and `nums2` would not exceed 1000.

```cpp
class Solution {
public:
    vector<int> nextGreaterElement(vector<int>& nums1, vector<int>&
nums2) {
        unordered_map<int, int> m;
        stack<int> stk;
        for (auto &i : nums2) {
            while (!stk.empty() && stk.top() < i) {
                m[stk.top()] = i;
                stk.pop();
            }
            stk.push(i);
        }
        for (auto &i : nums1) {
            if (m.count(i)) i = m[i];
            else i = -1;
        }
        return nums1;
    }
};
```

497. Random Point in Non-overlapping Rectangles

Medium

Given a list of **non-overlapping** axis-aligned rectangles `rects`, write a function `pick` which randomly and uniformily picks an **integer point** in the space covered by the rectangles.

Note:

1. An **integer point** is a point that has integer coordinates.
2. A point on the perimeter of a rectangle is **included** in the space covered by the rectangles.
3. `i`th rectangle = `rects[i]` = `[x1,y1,x2,y2]`, where `[x1, y1]` are the integer coordinates of the bottom-left corner, and `[x2, y2]` are the integer coordinates of the top-right corner.
4. length and width of each rectangle does not exceed `2000`.
5. `1 <= rects.length <= 100`
6. `pick` return a point as an array of integer coordinates `[p_x, p_y]`
7. `pick` is called at most `10000` times.

Example 1:

```
Input:

["Solution","pick","pick","pick"]

[[[[1,1,5,5]]],[],[],[]]

Output:

[null,[4,1],[4,1],[3,3]]
```

Example 2:

```
Input:

["Solution","pick","pick","pick","pick","pick"]

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

Output:

[null,[-1,-2],[2,0],[-2,-1],[3,0],[-2,-2]]
```

Explanation of Input Syntax:

The input is two lists: the subroutines called and their arguments. `Solution`'s constructor has one argument, the array of rectangles `rects`. `pick` has no arguments. Arguments are always wrapped with a list, even if there aren't any.

```
class Solution {
    vector<vector<int>> &rects;
    mt19937 rng;
    discrete_distribution<int> dist;

public:
    Solution(vector<vector<int>>& r) : rects(r), rng(random_device()()) {
        vector<int> weights;
        for (const auto &r : rects) {
            weights.push_back((r[2]-r[0]+1)*(r[3]-r[1]+1));
        }
        dist = discrete_distribution<int>(weights.begin(), weights.end());
    }

    vector<int> pick() {
        auto &r = rects[dist(rng)];
        int x = uniform_int_distribution<int>(r[0], r[2])(rng);
        int y = uniform_int_distribution<int>(r[1], r[3])(rng);
        return {x, y};
    }
};
```

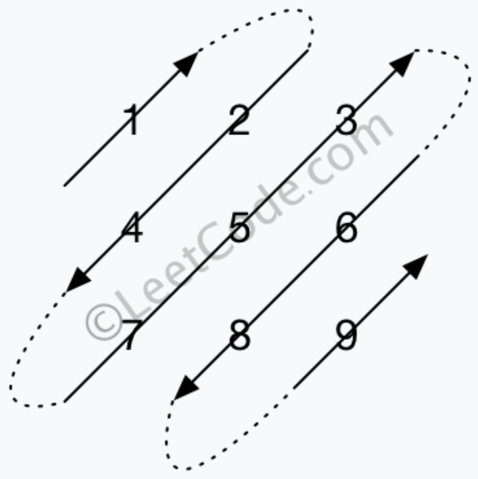
498. Diagonal Traverse

Medium

Given a matrix of M x N elements (M rows, N columns), return all elements of the matrix in diagonal order as shown in the below image.

Example:

```
Input:

[

  [ 1, 2, 3 ],

  [ 4, 5, 6 ],

  [ 7, 8, 9 ]

]

Output:  [1,2,4,7,5,3,6,8,9]

Explanation:
```



Note:

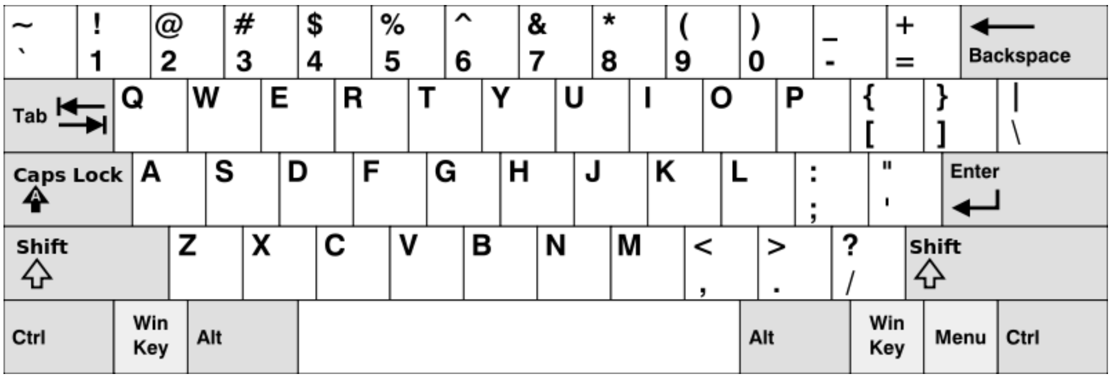
The total number of elements of the given matrix will not exceed 10,000.

```cpp
class Solution {
public:
    vector<int> findDiagonalOrder(vector<vector<int>>& matrix) {
        vector<int> res;
        if (matrix.empty()) return res;

        int m = matrix.size(), n = matrix[0].size(), flag = 0;
        for (int k = 0; k < m+n-1; k++) {
            int st = max(0, k+1-n), ed = min(m-1, k);
            if (flag) for (int i = st; i <= ed; i++) {
                res.push_back(matrix[i][k-i]);
            }
            else for (int i = ed; i >= st; i--) {
                res.push_back(matrix[i][k-i]);
            }
            flag ^= 1;
        }
        return res;
    }
};
```

500. Keyboard Row

Easy

Given a List of words, return the words that can be typed using letters of **alphabet** on only one row's of American keyboard like the image below.



Example:

Input: ["Hello", "Alaska", "Dad", "Peace"]

Output: ["Alaska", "Dad"]

Note:

1. You may use one character in the keyboard more than once.
2. You may assume the input string will only contain letters of alphabet.

```cpp
class Solution {
public:
    vector<string> findWords(vector<string>& words) {
        vector<int> vect{1,2,2,1,0,1,1,1,0,1,1,1,2,2,
                         0,0,0,0,1,0,0,2,0,2,0,2};
        vector<string> res;
        for (auto &s : words) {
            bool ok = true;
            int r = vect[tolower(s[0])-'a'];
            for (auto c : s) {
                if (vect[tolower(c)-'a'] != r) {
                    ok = false;
                    break;
                }
            }
            if (ok) res.push_back(s);
        }
        return res;
    }
};
```