Laboratory work #1. Arrays and String.

Solve all of problems for Arrays and for Strings.

Make a report on each tasks you have solved.

Deadline: 15th September 2020.

Arrays

1. <https://leetcode.com/problems/range-sum-query-immutable/>

Given an integer array *nums*, find the sum of the elements between indices *i* and *j* (*i* ≤ *j*), inclusive.

Given nums = [-2, 0, 3, -5, 2, -1]

sumRange(0, 2) -> 1

sumRange(2, 5) -> -1

sumRange(0, 5) -> -3

**Code:**

class NumArray {

int n;

int ans[2000000];

public:

NumArray(vector<int>& nums) {

n = nums.size();

if(n == 0)

return;

ans[0] = nums[0];

for (int i = 1; i < n; i++)

ans[i] = ans[i - 1] + nums[i];

}

int sumRange(int i, int j) {

if (i == 0)

return ans[j];

else

return ans[j] - ans[i - 1];

}

};

**Report:**

NumArray object will be instantiated and called as such:

NumArray\* obj = new NumArray(nums);

int param\_1 = obj->sumRange(i,j);

1. <https://leetcode.com/problems/product-of-array-except-self/>

Given an array nums of *n* integers where *n* > 1,  return an array output such that output[i] is equal to the product of all the elements of nums except nums[i].

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

**Output:** [24,12,8,6]

**Code:**

class Solution {

public:

vector<int> productExceptSelf(vector<int>& nums) {

int n = nums.size();

vector <int> ans(n);

if (n == 0)

return ans;

int a[n + 100], b[n + 100];

a[0] = nums[0];

b[n - 1] = nums[n - 1];

for (int i = 1; i < n; i++)

a[i] = a[i - 1] \* nums[i];

for (int i = n - 2; i >= 0; i--)

b[i] = b[i + 1] \* nums[i];

ans[0] = b[1];

ans[n - 1] = a[n - 2];

for (int i = 1; i < n - 1; i++)

ans[i] = a[i - 1] \* b[i + 1];

return ans;

}

};

**Report:**

In this task for solving arrays I use the vectors . I declare array an array by array size. In this task we initialized vector answer. First vector<int> productExceptSelf(vector<int>& nums) store product only with elements like [2,3,4,5] will produce [1, 2, 6, 24] as 2 have no left elements its result will be 1. Now store product with elements previously built [1, 2, 6, 24] will become [60,40,30,24] as last element have no right elements its result remain same

1. <https://leetcode.com/problems/missing-number/>

Given an array containing *n* distinct numbers taken from 0, 1, 2, ..., n, find the one that is missing from the array.

**Input:** [3,0,1]

**Output:** 2

**Code:**

class Solution {

public:

int missingNumber(vector<int>& nums) {

int n = nums.size();

int was[n + 100];

for (int i = 0; i <= n; i++)

was[i] = 0;

for (int i = 0; i < n; i++)

was[nums[i]] = 1;

for (int i = 0; i <= n; i++){

if (was[i] == 0)

return i;

}

return -1;

}

};

**Report:**

In this task for sollving the arrays i use the vector and sets. Number is equal to number of size. Resize the array to size + 100 and inserts 0 on the last position Then use condition operations.

1. <https://leetcode.com/problems/maximum-average-subarray-i/>

Given an array consisting of n integers, find the contiguous subarray of given length k that has the maximum average value. And you need to output the maximum average value.

**Input:** [1,12,-5,-6,50,3], k = 4

**Output:** 12.75

**Code:**

class Solution {

public:

double findMaxAverage(vector<int>& nums, int k) {

double mx = 0;

int n = nums.size();

double a[n + 100];

a[0] = nums[0];

for (int i = 1; i < n; i++)

a[i] = a[i - 1] + nums[i];

mx = a[k - 1] / k;

for (int i = k; i < n; i++)

mx = max(mx, (a[i] - a[i - k]) / k);

return mx;

}

};

**Report:**

Inn this task I use the vectors on the array and double max is equal to zero and number size is equal to n, in this case double get n + 100, then I use for index started by one and less than n, index of a equal to index of a minus one to plus index of number. In ths case maximum eual to index of k minus 1. Then also to run by for and give maximum and return to us.

1. <https://leetcode.com/problems/range-sum-query-2d-immutable/>

Given a 2D matrix *matrix*, find the sum of the elements inside the rectangle defined by its upper left corner (*row*1, *col*1) and lower right corner (*row*2, *col*2).

  
The above rectangle (with the red border) is defined by (row1, col1) = **(2, 1)** and (row2, col2) = **(4, 3)**, which contains sum = **8**.

**Code:**

class Solution {

public:

vector<int> productExceptSelf(vector<int>& nums) {

int n = nums.size();

vector <int> ans(n);

if (n == 0)

return ans;

int a[n + 100], b[n + 100];

a[0] = nums[0];

b[n - 1] = nums[n - 1];

for (int i = 1; i < n; i++)

a[i] = a[i - 1] \* nums[i];

for (int i = n - 2; i >= 0; i--)

b[i] = b[i + 1] \* nums[i];

ans[0] = b[1];

ans[n - 1] = a[n - 2];

for (int i = 1; i < n - 1; i++)

ans[i] = a[i - 1] \* b[i + 1];

return ans;

}

};

**Report:** In this task for solving arrays I use the vectors . I declare array an array by array size. In this task we initialized vector answer

1. <https://leetcode.com/problems/rotate-image/>strings

You are given an *n* x *n* 2D matrix representing an image, rotate the image by 90degrees (clockwise).

You have to rotate the image [**in-place**](https://en.wikipedia.org/wiki/In-place_algorithm), which means you have to modify the input 2D matrix directly. **DO NOT** allocate another 2D matrix and do the rotation.



Code:

class Solution {

public:

void rotate(vector<vector<int>>& matrix) {

int n = matrix.size();

for(int i = 0; i < n; ++i){

for(int j = i; j < n; ++j){

swap(matrix[i][j], matrix[j][i]);

}

}

for(int i = 0; i < n; ++i){

reverse(matrix[i].begin(), matrix[i].end());

}

}

};

**Report:**

In this task first we take transpose of the matrix with loop:

for(int i = 0; i < n; ++i){

for(int j = i; j < n; ++j){

swap(matrix[i][j], matrix[j][i]);

As well as then we take mirror image about the middle vertical line and make the reverse

for(int i = 0; i < n; ++i){

reverse(matrix[i].begin(), matrix[i].end());

Strings

1. <https://leetcode.com/problems/reverse-words-in-a-string/>

Given an input string, reverse the string word by word.

**Input:** "the sky is blue"

**Output:**"blue is sky the"

**Code:**

class Solution {

public:

string reverseWords(string s) {

reverse(s.begin(), s.end());

string h = "", ans = "";

int pos = -1;

for (int i = 0; i < s.size(); i++)

if (s[i] != ' ')

pos = i;

if (pos == -1)

return h;

for (int i = 0; i <= pos; i++){

if (s[i] == ' '){

reverse(h.begin(), h.end());

ans += h;

if (h.size() > 0)

ans += " ";

h = "";

}

else

h += s[i];

}

reverse(h.begin(), h.end());

ans += h;

return ans;

}

};

**Report:**

In this task I use function string reverseWords(string s) { and reverse by begin to end : reverse(s.begin(), s.end()); string h and answer is empty in this case position is -1 and advance to use loops if (s[i] is empty our position equal i. if is = -1 pos return to us string h. if in first for we started at zero and by size in next for we run by position. If s[i] equal to empty, reverse and write down to answer. And reversing is done, return to as reversing word.

1. <https://leetcode.com/problems/license-key-formatting/>

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.

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

**Output:** "5F3Z-2E9W"

**Code:**

class Solution {

public:

string licenseKeyFormatting(string S, int K) {

int n = S.size();

string ans = "";

deque <char> q;

int cnt = 0;

for (int i = 0; i < n; i++){

if (S[i] == '-')

continue;

if (S[i] >= 'a' && S[i] <= 'z')

S[i] = S[i] - 'a' + 'A';

q.push\_back(S[i]);

}

while (q.size() % K != 0){

ans += q.front();

q.pop\_front();

}

if (ans.size() > 0 && q.empty() == false)

ans += '-';

while (q.empty() == false){

ans += q.front();

q.pop\_front();

if (q.size() % K == 0 && q.empty() == false)

ans += '-';

}

return ans;

}

};

**Report:**

In this task start from the end of S and count the number of chars. q size which might be not equal and if answer size is more zero and empty, add '-' for every K chars added except at the beginning. Rest of the parts must consist of K characters

1. <https://leetcode.com/problems/string-to-integer-atoi/>

Implement atoi which converts a string to an integer. The function first discards as many whitespace characters as necessary until the first non-whitespace character is found. Then, starting from this character, takes an optional initial plus or minus sign followed by as many numerical digits as possible, and interprets them as a numerical value.

The string can contain additional characters after those that form the integral number, which are ignored and have no effect on the behavior of this function.

If the first sequence of non-whitespace characters in str is not a valid integral number, or if no such sequence exists because either str is empty or it contains only whitespace characters, no conversion is performed. If no valid conversion could be performed, a zero value is returned.

**Input:** "42"

**Output:** 42

**Code:**

class Solution {

public:

int myAtoi(string str) {

string s = str;

long long num = 0, neg = 0, plus = 0, a = 0, b = 1;

long long mx = INT\_MAX;

long long mn = INT\_MIN;

for (int i = 0; i < s.size(); i++){

if (s[i] == '+'){

if (plus > 0 || neg > 0 || a > 0)

return num \* b;

plus = 1;

}

else if (s[i] == '-'){

if (plus > 0 || neg > 0 || a > 0)

return num \* b;

b = b \* -1;

neg = 1;

}

else if (s[i] >= '0' && s[i] <= '9'){

a = 1;

if (num \* 10 >= mx){

if (num == mx / 10 && (s[i] - '0') <= mx % 10)

num = num \* 10 + (s[i] - '0');

else{

if (b == -1)

return mn;

else

return mx;

}

}

else

num = num \* 10 + (s[i] - '0');

}

else if (s[i] == ' '){

if (plus || neg)

return num \* b;

if (a > 0)

return num \* b;

}

else

return num \* b;

}

long long k = num \* b;

if (k < mn)

return mn;

else if (k > mx)

return mx;

else

return k;

}

};

**Report:**

In this task I use condition operations and loops. '+' or '-' can only appear once, after space (if any) and before numberic characters.

1. <https://leetcode.com/problems/integer-to-english-words/>

Convert a non-negative integer to its english words representation. Given input is guaranteed to be less than 231 - 1.

**Input:** 123

**Output:** "One Hundred Twenty Three"

**Code:**

class Solution {

public:

string change(int number){

if (number == 0)

return "";

if (number == 1)

return "One";

if (number == 2)

return "Two";

if (number == 3)

return "Three";

if (number == 4)

return "Four";

if (number == 5)

return "Five";

if (number == 6)

return "Six";

if (number == 7)

return "Seven";

if (number == 8)

return "Eight";

if (number == 9)

return "Nine";

if (number == 10)

return "Ten";

if (number == 11)

return "Eleven";

if (number == 12)

return "Twelve";

if (number == 13)

return "Thirteen";

if (number == 14)

return "Fourteen";

if (number == 15)

return "Fifteen";

if (number == 16)

return "Sixteen";

if (number == 17)

return "Seventeen";

if (number == 18)

return "Eighteen";

if (number == 19)

return "Nineteen";

if (number <= 29)

return "Twenty";

if (number <= 39)

return "Thirty";

if (number <= 49)

return "Forty";

if (number <= 59)

return "Fifty";

if (number <= 69)

return "Sixty";

if (number <= 79)

return "Seventy";

if (number <= 89)

return "Eighty";

if (number <= 99)

return "Ninety";

return "";

}

string numberToWords(int num){

if (num == 0)

return "Zero";

string ans = "";

if (num / 1000000000){

int x = num / 1000000000;

ans = numberToWords(x) + " Billion ";

num = num - x \* 1000000000;

}

if(num / 1000000){

int x = num / 1000000;

ans = ans + numberToWords(x) + " Million ";

num = num - x \* 1000000;

}

if(num / 1000){

int x = num / 1000;

ans = ans + numberToWords(x) +" Thousand ";

num = num - x \* 1000;

}

if(num / 100){

int x = num / 100;

ans = ans + change(x) + " Hundred ";

num = num - x \* 100;

}

if(num >= 20){

int x = num / 10;

ans = ans + change(num);

if(num % 10 != 0){

num = num - x \* 10;

ans = ans + ' ' + change(num);

}

}

else

ans = ans + change(num);

if(ans[ans.size() - 1] == ' ')

ans.erase(ans.size() - 1, 1);

return ans;

}

};

**Report:**

In this task I use ‘IF’ condition operations.

string change(int number){ }function return number < 100 to string. After we check if the input is zero, string zero return to us zero. And we check the billion then return the function itself to return < 1000000000; This operation we do for million, thousand and last is hundred. For hundred, we use the above function to return < 999 value. And we delete the space of the string

1. <https://leetcode.com/problems/reverse-words-in-a-string-iii/>

Given a string, you need to reverse the order of characters in each word within a sentence while still preserving whitespace and initial word order.

**Input:** "Let's take LeetCode contest"

**Output:** "s'teL ekat edoCteeL tsetnoc"

**Code:**

class Solution {

public:

string reverseWords(string s) {

string ans = "", h = "";

for (int i = 0; i < s.size(); i++){

if (s[i] == ' '){

if (h.size() > 0){

reverse(h.begin(), h.end());

ans += h;

}

h = "";

ans += ' ';

}

else

h += s[i];

}

if (h.size() > 0){

reverse(h.begin(), h.end());

ans += h;

}

return ans;

}

};

**Report:**

Save length at first to save time from repeatedly re-calculating it and Add space-buffer to end of input to help find last word. Loop through every char in input to find all the words and a word exists if the current char is a non-space and the next char is more zero. Reverse the current word from indices begin to h. Return the output with in-place reversed words