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/>
2. <https://leetcode.com/problems/maximum-subarray/>
3. <https://leetcode.com/problems/product-of-array-except-self/>
4. <https://leetcode.com/problems/missing-number/>
5. <https://leetcode.com/problems/maximum-average-subarray-i/>
6. <https://leetcode.com/problems/range-sum-query-2d-immutable/>
7. <https://leetcode.com/problems/rotate-image/>

Strings

1. <https://leetcode.com/problems/reverse-words-in-a-string/>
2. <https://leetcode.com/problems/license-key-formatting/>
3. <https://leetcode.com/problems/string-to-integer-atoi/>
4. <https://leetcode.com/problems/integer-to-english-words/>
5. <https://leetcode.com/problems/reverse-words-in-a-string-iii/>

**Answers:**

Arrays:

1. **303. Range Sum Query – Immutable**

class NumArray {

private int[] sums;

public NumArray(int[] nums) {

sums = new int[nums.length];

if (nums.length == 0) {

return;

}

sums[0] = nums[0];

for (int i = 1; i < nums.length; i++) {

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

}

}

public int sumRange(int i, int j) {

if (i == 0) {

return sums[j];

} else {

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

}

}

}

First of all, if you just ask [i, j] And that's too easy. This question is mainly for investigation by calling sumRange many times a way to make sure it doesn't expire.

So if you just sum directly it will certainly not be up to date.

1. **53. Maximum Subarray**

class Solution {

public:

int maxSubArray(vector<int>& nums) {

int max = INT\_MIN;

int sum = 0;

int i = 0;

while(i < nums.size())

{

sum += nums[i];

if(max < sum)

max = sum;

if(sum < 0)

sum = 0;

i++;

}

return max;

}

};

dynamic programming.

1. **238. Product of Array Except Self**

class Solution {

public int[] productExceptSelf(int[] nums) {

int l = nums.length;

int[] res = new int[ l ];

int pro = 1;

res[ 0 ] = 1;

for( int i = 1 ; i < l ; i ++ ){

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

}

for( int i = l - 1 ; i >= 0 ; i -- ){

res[ i ] \*= pro;

pro \*= nums[ i ];

}

return res;

}

}

first find the product of all numbers, then divide by the number corresponding to each subscript. There is a problem if the result of the multiplication is going to be very large, which let me try first, during the writing process I thought it might be 0, but I didn't handle it well. I looked at the discussion. The idea is to write the product from the first item to the current item and then compute the edge update from right to left.

1. **268. Missing Number**

class Solution {

public int missingNumber(int[] nums) {

int n = nums.length;

int sum = ((n+1)\*n)/2;

for(int num : nums)

sum -= num;

return sum;

}

}

Find the sum of 0, 1, 2, ..., n and subtract the elements of the array in A to get the missing elements.

1. **643. Maximum Average Subarray I**

public class Solution {

public double findMaxAverage(int[] nums, int k) {

int max = Integer.MIN\_VALUE;

int i = 0;

while(i + k - 1 < nums.length) {

int curr = 0;

int j = i;

int cnt = k;

while(cnt-- > 0) {

curr += nums[j];

j++;

}

if(curr > max)

max = curr;

i++;

}

return (double)max / k;

}

}

This question requires finding the maximum average of consecutive k digits, which is essentially the maximum sum of consecutive k digits. The most intuitive solution is to start with i = 0, add consecutive k bits and compare with the maximum value of max, then move i back one bit to continue calculating the sum of consecutive k bits until the last bit of consecutive k bits The number of subscribers has exceeded the bounds

1. **304. Range Sum Query 2D – Immutable**

class NumMatrix {

public:

NumMatrix(vector<vector<int> > &matrix) {

if (matrix.empty() || matrix[0].empty()) return;

dp.resize(matrix.size() + 1, vector<int>(matrix[0].size() + 1, 0));

for (int i = 1; i <= matrix.size(); ++i) {

for (int j = 1; j <= matrix[0].size(); ++j) {

dp[i][j] = dp[i][j - 1] + dp[i - 1][j] - dp[i - 1][j - 1] + matrix[i - 1][j - 1];

}

}

}

int sumRegion(int row1, int col1, int row2, int col2) {

return dp[row2 + 1][col2 + 1] - dp[row1][col2 + 1] - dp[row2 + 1][col1] + dp[row1][col1];

}

private:

vector<vector<int> > dp;

};

Maintain an array and then sum it locally, usually as a tree array, or use a string tree.

1. **48. Rotate Image**

public class Solution {

public void rotate(int[][] matrix) {

int n = matrix.length;

for (int i = 0; i < n / 2; i++) {

for (int j = 0; j < Math.ceil(((double) n) / 2.); j++) {

int temp = matrix[i][j];

matrix[i][j] = matrix[n-1-j][i];

matrix[n-1-j][i] = matrix[n-1-i][n-1-j];

matrix[n-1-i][n-1-j] = matrix[j][n-1-i];

matrix[j][n-1-i] = temp;

}

}

}

}

Idea: find the relationship between the coordinates of the matrix and the left and exchange as shown in the picture. I think the easiest way is to swap each row and column and then reorder each row.

String:

1. **151. Reverse Words in a String**

class Solution {

public String reverseWords(String s) {

String result = new String();

int i = 0;

int n = s.length();

while( i < n){

while( i < n && s.charAt(i) == ' ')

i++;

if(i >= n) break;

int j = i + 1;

while( j < n && s.charAt(j) != ' ')

j++;

String sub = s.substring(i, j);

if(result.length() == 0) result = sub;

else result = sub + " " + result;

i = j + 1;

}

return result;

}

};

Reverse method twice. Note that multiple spaces between words leave only one space and remove leading and trailing 0s.

1. **482. License Key Formatting**

class Solution {

public String licenseKeyFormatting(String S, int K) {

StringBuilder sb = new StringBuilder();

for (int i = S.length() - 1; i >= 0; i--){

if (S.charAt(i) != '-'){

sb.append(sb.length() % (K + 1) == K ? '-' : "").append(S.charAt(i));

}

}

return sb.reverse().toString().toUpperCase();

}

}

Give a string to indicate the license plate and K to indicate the length of each paragraph of text after processing. With the exception of the first paragraph, he may not be happy with K, and all subsequent paragraphs must be in the form K.

1. **8. String to Integer (atoi)**

class Solution {

public int myAtoi(String str) {

if (str.isEmpty()) return 0;

int sign = 1, base = 0, i = 0, n = str.length();

while (i < n && str.charAt(i) == ' ') ++i;

if (i < n && (str.charAt(i) == '+' || str.charAt(i) == '-')) {

sign = (str.charAt(i++) == '+') ? 1 : -1;

}

while (i < n && str.charAt(i) >= '0' && str.charAt(i) <= '9') {

if (base > Integer.MAX\_VALUE / 10 || (base == Integer.MAX\_VALUE / 10 && str.charAt(i) - '0' > 7)) {

return (sign == 1) ? Integer.MAX\_VALUE : Integer.MIN\_VALUE;

}

base = 10 \* base + (str.charAt(i++) - '0');

}

return base \* sign;

}

}

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.

1. **273. Integer to English Words**

class Solution {

public String numberToWords(int num) {

if (num == 0) {

return "Zero";

}

ArrayList<String> unit = new ArrayList<String>();

HashMap<Integer, String> singles = new HashMap<Integer, String>();

HashMap<Integer, String> doubles = new HashMap<Integer, String>();

init(unit, singles, doubles);

String res = "";

int index = 0;

while (num != 0) {

int tmp = num % 1000;

if (tmp != 0) {

String part = "";

int hund = tmp / 100;

if (hund != 0) {

part = part + singles.get(hund) + " Hundred";

}

tmp = tmp % 100;

if (tmp >= 10 && tmp <= 19) {

part = part + " " + doubles.get(tmp);

}

else {

int tens = tmp / 10;

if (tens != 0) {

part = part + " " + doubles.get(tens);

}

int ones = tmp % 10;

if (ones != 0) {

part = part + " " + singles.get(ones);

}

}

part = part + " " + unit.get(index);

part = part.trim() + " ";

res = part + res;

}

index ++;

num = num / 1000;

}

return res.trim();

}

private void init(ArrayList<String> unit, HashMap<Integer, String> singles, HashMap<Integer, String> doubles) {

unit.add("");

unit.add("Thousand");

unit.add("Million");

unit.add("Billion");

singles.put(1, "One");

singles.put(2, "Two");

singles.put(3, "Three");

singles.put(4, "Four");

singles.put(5, "Five");

singles.put(6, "Six");

singles.put(7, "Seven");

singles.put(8, "Eight");

singles.put(9, "Nine");

doubles.put(10, "Ten");

doubles.put(11, "Eleven");

doubles.put(12, "Twelve");

doubles.put(13, "Thirteen");

doubles.put(14, "Fourteen");

doubles.put(15, "Fifteen");

doubles.put(16, "Sixteen");

doubles.put(17, "Seventeen");

doubles.put(18, "Eighteen");

doubles.put(19, "Nineteen");

doubles.put(2, "Twenty");

doubles.put(3, "Thirty");

doubles.put(4, "Forty");

doubles.put(5, "Fifty");

doubles.put(6, "Sixty");

doubles.put(7, "Seventy");

doubles.put(8, "Eighty");

doubles.put(9, "Ninety");

}

}

You can see this problem is annoying, there are many branches of affairs, and there are a few special cases. These comparisons are particularly long. The real part of the algorithm is over 30 lines long. The key to this question is that the numbers in English are grouped every three.

For example, this number: 1 234 567. This is 1 million 234 thousand 567.

1. **557. Reverse Words in a String III**

class Solution {

public String reverseWords(String s) {

String[] words = s.split(" ");

StringBuilder res = new StringBuilder();

for(String word : words){

res.append(new StringBuilder(word).reverse().append(" "));

}

return res.toString().trim();

}

}

method

1 、 split (): Split string according to regex.

2 、 reverse (): Reverse a string

3 、 trim () Method: Used to remove leading and trailing whitespace characters from a string.

4 、 toString () Returns this object itself (this is already a string)