# leetcode个人刷题笔记

|  |  |
| --- | --- |
| 撰写人 | 刘恒辉 |
| 版本 |  |
| 时间 | 2019.7.5 |

1. 原题目(8)

请你来实现一个 atoi 函数，使其能将字符串转换成整数。

首先，该函数会根据需要丢弃无用的开头空格字符，直到寻找到第一个非空格的字符为止。

当我们寻找到的第一个非空字符为正或者负号时，则将该符号与之后面尽可能多的连续数字组合起来，作为该整数的正负号；假如第一个非空字符是数字，则直接将其与之后连续的数字字符组合起来，形成整数。

该字符串除了有效的整数部分之后也可能会存在多余的字符，这些字符可以被忽略，它们对于函数不应该造成影响。

注意：假如该字符串中的第一个非空格字符不是一个有效整数字符、字符串为空或字符串仅包含空白字符时，则你的函数不需要进行转换。

在任何情况下，若函数不能进行有效的转换时，请返回 0。

说明：

假设我们的环境只能存储 32 位大小的有符号整数，那么其数值范围为 [−231,  231 − 1]。如果数值超过这个范围，请返回  INT\_MAX (231 − 1) 或 INT\_MIN (−231) 。

1. 题目大意

按照题目要求将字符串转化为数字。

1. 思路分析

本人的思路：

将正负号和数字字符收集起来转换成整型数据，并且控制在32位整型之中。

1. 具体代码

本人的代码：

bool isNumericChar(char x)//判断字符是否为数字

{

return (x >= '0' && x <= '9') ? true : false;

}

int myAtoi(char \* str)

{

if (str == NULL)

return 0;

long res = 0; //初始化结果为零

int sign = 1; // 初始标记为一

int i = 0; //初始化索引

for (; str[i] != '\0'; i++)

{

if (str[i] != ' ')

{

break;

}

}

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

{

sign = -1;//结果为负数

i++;//开始识别之后的数字字符

if (str[i] == '+')

{

return 0;

}

}

if (str[i] == '+')

{

sign = 1;//结果为负数

i++;//开始识别之后的数字字符

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

{

return 0;

}

}

for (; str[i] != '\0'; i++)//循环到结束符

{

if (isNumericChar(str[i]) == false)

break;

res = res \* 10 + str[i] - '0';//不断更新字符转化后的数据

if (res >=2147483648)

{

res = sign>0?2147483647: 2147483648;

break;

}

}

return sign \* res;

}

1. 来源

题目连接：<https://leetcode-cn.com/problems/string-to-integer-atoi/>

1. 原题目(10)

给你一个字符串 s 和一个字符规律 p，请你来实现一个支持 '.' 和 '\*' 的正则表达式匹配。

所谓匹配，是要涵盖 整个 字符串 s的，而不是部分字符串。

说明:

s 可能为空，且只包含从 a-z 的小写字母。

p 可能为空，且只包含从 a-z 的小写字母，以及字符 . 和 \*。

1. 题目大意

对两个字符串根据规则判断。

1. 思路分析

根据题目要求，设置多个嵌套判断语句。

1. 具体代码

本人的代码：

bool isMatch(char \* s , char \* p)

{

assert(s && p); //如果其中一个为空，则结束程序

if (\*p == '\0')

{

return \*s == '\0';//\*s=='\0'返回true，\*s=='\0'返回false

}

if (\*(p + 1) != '\*') //字符串p第二个字符不为‘\*’情况

{

assert(\*p != '\*');//字符串p第一个字符为‘\*’结束程序

if ((\*p == \*s) || (\*p == '.' && \*s != '\0'))//两个字符相匹配或者字符串p第一个字符为‘.’且字符串s不为空

{

return isMatch(s + 1 , p + 1);//递归到两个字符串的下一个字符

}

else

{

return 0;

}

}

else//字符串p第二个字符为‘\*’时

{

while ((\*p == \*s) || (\*p == '.' && \*s != '\0'))

{

if (isMatch(s , p + 2))

{

return 1;

}

s++;

}

return isMatch(s , p + 2);

}

}

1. 来源

题目连接：<https://leetcode-cn.com/problems/regular-expression-matching/>

1. 原题目(221)

在一个由 0 和 1 组成的二维矩阵内，找到只包含 1 的最大正方形，并返回其面积。

1. 题目大意

找到矩阵中完整且最大的正方形。

1. 思路分析

本人的思路：

1.我们用一个变量去来记录迄今为止发现的最大正方形的边长，以及用一个变量记录当前正方形的大小，两个变量都初始化为 0；

2.从矩阵的左上角开始搜索 1，找到 0 不需要做任何操作，只要找到 1 我们就试图找到由 1 组成的最大正方形；

3.为此我们向右和向下移动，临时增加列索引和行索引，然后用标志标记该行列是否全都为 1；

4.如果全都为 1，则继续检索行列，如果找到 0，便停止移动，更新最大正方形的边长。然后从最初发现 1 的元素旁边遍历矩阵，直到矩阵的所有元素都被遍历。

1. 具体代码

本人的代码：

int maximalSquare(char \*\* matrix , int matrixSize , int \* matrixColSize)

{

if (matrixSize == 0 || matrixColSize == 0)//数组大小为0时结束

return 0;

int \*\* result = (int \*\*)malloc(matrixSize \* sizeof(int \*));//动态开辟一个二维数组的列

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

{

result[i] = (int \*)malloc(\*matrixColSize \* sizeof(int));//动态开辟数组的行

}

int s\_max = matrix[0][0] - '0';

for (int i = 0; i < matrixSize; i++)//找到第一行和第一列的最大值

{

result[i][0] = matrix[i][0] - '0';

if (result[i][0] == 1)

s\_max = 1;

}

for (int i = 1; i < \*matrixColSize; i++)

{

result[0][i] = matrix[0][i] - '0';

if (result[0][i] == 1)

s\_max = 1;

}

for (int row = 1; row < matrixSize; row++)

{

for (int col = 1; col < \*matrixColSize; col++)

{

if (matrix[row][col] == '1' && result[row - 1][col - 1] > 0)

{

int check\_num = sqrt(result[row - 1][col - 1]);

int i = 0;

for (; i < check\_num; i++)

{

if (result[row][col - 1 - i] == 0 || result[row - 1 - i][col] == 0)

{

break;

}

}

result[row][col] = (i + 1) \* (i + 1);

if (s\_max < result[row][col])

s\_max = result[row][col];

}

else

{

result[row][col] = matrix[row][col] - '0';

}

}

}

return s\_max;//返回最大正方形

}

1. 来源

题目连接：<https://leetcode-cn.com/problems/maximal-square/>

1. 原题目(367)

给定一个正整数 num，编写一个函数，如果 num 是一个完全平方数，则返回 True，否则返回 False。

说明：不要使用任何内置的库函数，如  sqrt。

1. 题目大意

给定一个正整数，判定是否为完全平方数。

1. 思路分析

本人的思路：

计算从1开始的平方，直到平方与该正整数相等或大于该数

四、具体代码

本人的代码：

bool isPerfectSquare(int num)//通过计算1到n的平方，过程中比较是否与目标数相等

{

int i = 1;

long n = i \* i;

while (n <= num)

{

if (n == num)

return true;

else

{

i++;

n = pow(i , 2);

}

}

return false;

}

五、来源

题目连接：<https://leetcode-cn.com/problems/valid-perfect-square/>

1. 原题目(368)

给出一个由无重复的正整数组成的集合，找出其中最大的整除子集，子集中任意一对 (Si，Sj) 都要满足：Si % Sj = 0 或 Sj % Si = 0。

如果有多个目标子集，返回其中任何一个均可。

1. 题目大意

使得给定数据中筛选出能相互整除的数据。

1. 思路分析

本人的思路：

1. 若a % b == 0 && c % a == 0; 那么c % b == 0；

2. 所以当我们 已求得已a为最大数字的最长串后，则 a对于c来说：a串中所有的数都是c的约数.

3. 从上面我们可以利用动态规划的思想，从小到大来求数的最长约数串。然后后面的数字可以使用比其小的数的结果。

1. 具体代码

本人的代码：

int partition(int \* nums , int low , int high)

{

int pivotkey = nums[low];

while (low < high)

{

while (low < high && nums[high] >= pivotkey) --high;

nums[low] = nums[high];

while (low < high && nums[low] <= pivotkey) ++low;

nums[high] = nums[low];

}

nums[low] = pivotkey;

return low;

}

void quickSort(int \* nums , int from , int to)//快速排序

{

if (from < to)

{

int pivot = partition(nums , from , to);

quickSort(nums , from , pivot - 1);

quickSort(nums , pivot + 1 , to);

}

}

int \* largestDivisibleSubset(int \* nums , int numsSize , int \* returnSize)

{

if (numsSize == 0)

{

\*returnSize = 0;

return NULL;

}

// 先排序

quickSort(nums , 0 , numsSize - 1);

// 用于记录最长串的长度

int counts[10000];

counts[0] = 1;

// 用来记住position处的上一个最长约束的position

int last\_max\_inds[10000];

// -1 代表上面没有

last\_max\_inds[0] = -1;

// 用于记录全局的最大长度和其inds

int max = 1;

int inds = 0;

// 用于记录单个数的最长和它的上一个数的inds

int tmp\_max , tmp\_inds;

int i , j;

for (i = 1; i < numsSize; ++i)

{

// 前面的数都不是其nums[i]的约数

tmp\_max = 1;

tmp\_inds = -1;

for (j = 0; j < i; ++j)

{

if (nums[i] % nums[j] == 0)

{

if (counts[j] + 1 > tmp\_max)

{

tmp\_max = counts[j] + 1;

tmp\_inds = j;

}

}

}

counts[i] = tmp\_max;

last\_max\_inds[i] = tmp\_inds;

// 更新全局的

if (max < tmp\_max)

{

max = tmp\_max;

inds = i;

}

}

\*returnSize = max;

int \* res = (int \*)malloc(sizeof(int) \* max);

i = 0;

res[i] = nums[inds];

int last\_inds = last\_max\_inds[inds];

while (last\_inds != -1)

{

res[++i] = nums[last\_inds];

last\_inds = last\_max\_inds[last\_inds];

}

quickSort(res , 0 , max - 1);

return res;

}

1. 来源

题目连接：<https://leetcode-cn.com/problems/largest-divisible-subset/>