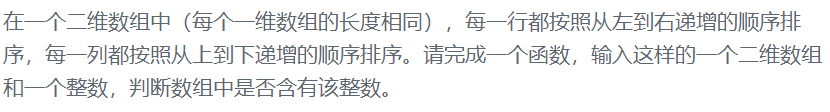
**2月28日**

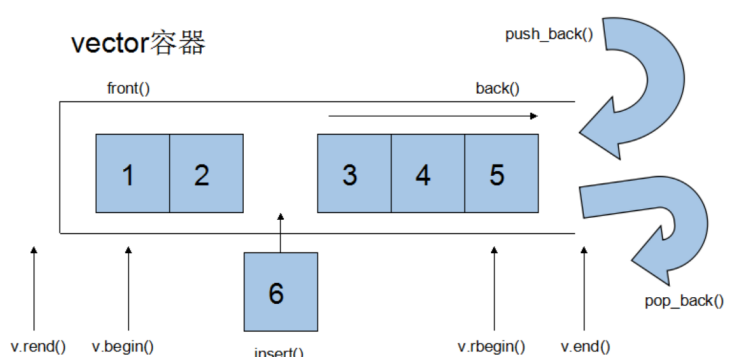
* 第一题
* 题目描述



* 相关知识补充
* Vector容器

1. 基本概念

Vector容器与array非常相似，区别就在于空间使用的灵活性，Array是静态控件，一旦配置过后就不能更改。Vector是动态空间，随着元素的增加，他的内部机制会自动的扩充内容以容纳新的元素。



1. vector数据结构

线性连续空间，他以两个迭代器\_Myfirst和\_Mylast分别指向配置得来的连续空间中目前被使用的范围，并以迭代器\_Myend指向整块连续空间的尾端

注意：一个vector的容量永远大于或等于其大小.

1. vector常用的API操作
2. 大小操作

size();//返回容器中元素的个数

empty();//判断容器是否为空

resize(int num);//重新指定容器的长度为num，若容器变长，则以默认值填充新位置。如果容器变短，则末尾超出容器长度的元素被删除。

resize(int num, elem);//重新指定容器的长度为num，若容器变长，则以elem值填充新位置。如果容器变短，则末尾超出容器长>度的元素被删除。

capacity();//容器的容量

reserve(int len);//容器预留len个元素长度，预留位置不初始化，元素不可访问。

void printVector(vector <int>&v)

{

for(vector<int>::iterator it = v.begin();it != v.end();it++)

{

cout<<\*it<<"";

}

cout<<endl;

}

1. 赋值的使用—assign(beg,end)
2. Vector数据存取操作

at(int idx); //返回索引idx所指的数据，如果idx越界，抛出out\_of\_range异常。

operator[];//返回索引idx所指的数据，越界时，运行直接报错

front();//返回容器中第一个数据元素

back();//返回容器中最后一个数据元素代码片

1. Vector插入删除操作

insert(const\_iterator pos, int count,ele);//迭代器指向位置pos插入count个元素ele.

push\_back(ele); //尾部插入元素ele

pop\_back();//删除最后一个元素

erase(const\_iterator start, const\_iterator end);//删除迭代器从start到end之间的元素

erase(const\_iterator pos);//删除迭代器指向的元素

clear();//删除容器中所有元素

* 题目分析

利用二维数组由上到下，由左到右递增的规律，选取右上角的元素arr[row][col]与target进行比较。

当target大于arr[row][col]的时候，row++

当target小于arr[row][col]的时候，col—

易错点：

1. 计算数组的大小，应该调用array.size(),而不是array.length
2. 如果使用左下角元素的话，int row = array[0].size() - 1;就不正确，会存在数组越界的情况

* 代码求解

class Solution {

public:

bool Find(int target, vector<vector<int> > array) {

if(array.size() != 0)

{

int col = array[0].size() - 1;

int row = 0;

while(col >= 0 && row < array.size())

{

if(array[row][col] == target)

{

return true;

}

else if(array[row][col] > target)

col--;

else

row++;

}

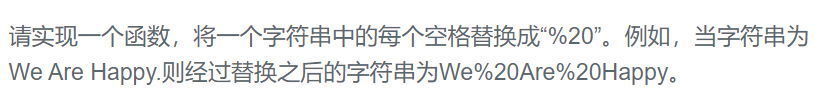
return false;

}

}

};

* 第二题
* 题目描述



* 相关知识补充
* 题目分析

首先，按照我们常规的思考方式，字符串的替换一对一的替换就是遍历字符串找到要替换的位置，直接替换就行。但是这道题的替换是一对多的替换

方法一：开辟一个新的字符串来加以替换，但是这样浪费空间

方法二：在当前字符串替换，但是就要思考怎么替换才是最有效的呢

1. 从前往后替换，因为后面的字符要不断往后移动，要多次移动，所以效率低下
2. 从后往前替换，先计算需要多少空间，然后从后往前移动，则每个字符只会移动一次，这样的效率更高

注意：1.遍历字符串的时候，循环的条件I < length,因为是从0下标开始的

2.在字符串替换的时候，一定记住count--

* 代码求解

class Solution {

public:

void replaceSpace(char \*str,int length) {

int count = 0;

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

{

if(str[i] == ' ')

count++;

}

for(int j = length-1;j >= 0;j--)

{

if(str[j] != ' ')

str[j + 2\*count] = str[j];

else

{

count--;

str[j + 2\*count] = '%';

str[j + 2\*count+1] = '2';

str[j + 2\*count+2] = '0';

}

}

}

};

* 第三题
* 题目描述



* 相关知识补充
* 题目分析

第一种思路：利用栈先入后出的特性完成。依次遍历链表，将链表中的数据存入栈中。然后再出栈

注意：1、链表入栈的条件的p!=NULL

2、栈出栈再重新入vector容器时，使用的是value.push\_back，从尾部插入一个元素

第二种思路：数组翻转，用c++自带的函数

第三种思路：递归，不提倡。

* 代码求解

思路一：

class Solution {

public:

vector<int> printListFromTailToHead(ListNode\* head) {

vector<int> value;

stack<int> stk;

ListNode\* p = NULL;

p = head;

while(p != NULL)

{

stk.push(p->val);

p = p->next;

}

while(!stk.empty())

{

value.push\_back(stk.top());

stk.pop();

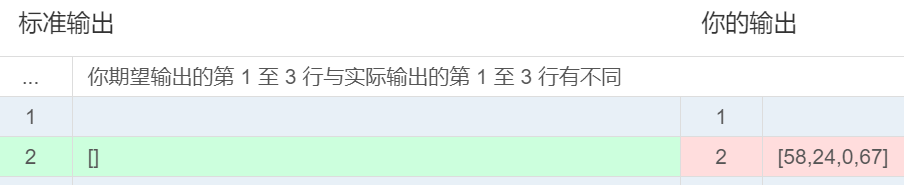
}

return value;

}

};

思路二：错误的，当链表是空的时候输出错误



class Solution {

public:

vector<int> printListFromTailToHead(ListNode\* head) {

vector<int> value;

stack<int> stk;

ListNode\* p = NULL;

p = head;

while(p != NULL)

{

stk.push(p->val);

p = p->next;

}

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

return value;

}

};