# 151. Reverse Words in a String

|  |
| --- |
| Given an input string, reverse the string word by word.  Try to solve it in-place in O(1) space. |
| For example,  Given s = "the sky is blue",  return "blue is sky the". |

**corner cases:**

word的定义：假设为由空格分开的子串

空格是否可以有多个：假设可以

空格是否还要保持原来的个数：假设间隔缩短到1个

开始／结束位置的空格怎么处理：假设把它们都删去

**Sol: global+local reverse**

(1) rectify输入，删掉所有多余的空格。

(2) 定位字符串中所有的单词，每个单词字串reverse。

(3) 把整个单词reverse。

|  |
| --- |
| **void reverseWords(string &s)** {  compressStr(s);  reversePerWord(s);  reverse(s.begin(), s.end());  }  inline void compressStr(string &s) {  size\_t iw = 0;  char last\_char = ' ';  for (size\_t ir = 0; ir < s.size(); ++ir) {  if (s[ir] == ' ') {  if (last\_char != ' ') s[iw++] = ' ';  }  else {  s[iw++] = s[ir];  }  last\_char = s[ir];  }  if (iw > 0 && s[iw-1] == ' ') iw--;  s.resize(iw);  }  inline void reversePerWord(string &s) {  for (int k = 0; k < s.size();) {  int j;  for (j = k+1; j < s.size() && s[j] != ' '; ++j);  reverse(s.begin()+k, s.begin()+j);  k = j+1;  }  } |

# 152. Maximum Product Subarray

|  |
| --- |
| Find the contiguous subarray within an array (containing at least one number) which has the largest product.  For example, given the array [2,3,-2,4],  the contiguous subarray [2,3] has the largest product = 6. |

**Sol: DP**

遍历所有以nums[k]为末元素的子数组，求对应的最大乘积和最小乘积。更新nums[k]对应的curmax,curmin时，可被挑选的乘积是nums[k], curmax\*nums[k], curmin\*nums[k]

|  |
| --- |
| **int maxProduct(vector<int>& nums)** {  if (nums.size() == 0) return 0;  long long curmax = nums[0], curmin = nums[0], res = nums[0];  for(int i=1; i<nums.size(); ++i){  long long premax = curmax;  curmax = max(curmax\*nums[i], max(curmin\*nums[i], (long long)nums[i]));  curmin = min(premax\*nums[i], min(curmin\*nums[i], (long long)nums[i]));  res = max(res, curmax);  }  return int(res);  } |

# 

# 

# 153. Find Minimum in Rotated Sorted Array

|  |
| --- |
| Suppose a sorted array is rotated at some pivot unknown to you beforehand.  (i.e., 0 1 2 4 5 6 7 might become 4 5 6 7 0 1 2).  Find the minimum element.  You may assume no duplicate exists in the array. |

**Sol: binary search**

如果某个子段首元素小于末元素，则该子段里没有pivot

|  |
| --- |
| **int findMin(vector<int>& nums)** {  int first = 0, last = nums.size()-1;  while (first < last && nums[first] > nums[last]) {  int mid = (first + last)/2;  if (nums[mid] < nums[last]) last = mid;  else first = mid+1;  }  return nums[first];  } |

# 

# 154. Find Minimum in Rotated Sorted Array II

|  |
| --- |
| Follow up for "Find Minimum in Rotated Sorted Array":  What if duplicates are allowed?  Would this affect the run-time complexity? How and why?  Suppose a sorted array is rotated at some pivot unknown to you beforehand.  (i.e., 0 1 2 4 5 6 7 might become 4 5 6 7 0 1 2).  Find the minimum element.  The array may contain duplicates. |

**Sol: binary search**

子数组两端的值相等时难以折半，因此只能数组右端向左缩减一。因此算法不能再是O(logn)的，最坏情况是O(n)的。

|  |
| --- |
| **int findMin(vector<int>& nums)** {  int first = 0, last = nums.size()-1;  while (first < last && nums[first] >= nums[last]) {  if (nums[first] == nums[last]) { --last; continue; }  int mid = (first + last)/2;  if (nums[mid] <= nums[last]) last = mid;  else first = mid+1;  }  return nums[first];  } |

# 155. Min Stack

|  |
| --- |
| Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.  push(x) -- Push element x onto stack.  pop() -- Removes the element on top of the stack.  top() -- Get the top element.  getMin() -- Retrieve the minimum element in the stack. |
| Example:  MinStack minStack = new MinStack();  minStack.push(-2);  minStack.push(0);  minStack.push(-3);  minStack.getMin(); --> Returns -3.  minStack.pop();  minStack.top(); --> Returns 0.  minStack.getMin(); --> Returns -2. |

**Sol1: two stacks**

在stack中存储当前元素，以及当前最小元素

|  |
| --- |
| **class MinStack** {  public:  **MinStack()** {  stk.push(make\_pair(0, INT\_MAX));  }    **void push(int x)** {  stk.push(make\_pair(x, min(x, stk.top().second)));  }    **void pop()** {  stk.pop();  }    **int top()** {  return stk.top().first;  }    **int getMin()** {  return stk.top().second;  }    private:  stack<pair<int, int>> stk;  **};** |

**Sol2: stack + global diff**

维护一个全局的minv，用一个stack diff存储当前值和压入当前值之前的min的差值

因此：

(0) 初始化：在栈底压入INT\_MAX以简化push时的情形

(1) push(x): 压入x-minv, 更新minv

(2) pop(): 弹出diff.top()，如果diff < 0，说明对应的x值压入把minv减少了diff，因此相应地更新minv

(3) top(): 如果diff.top()>0，则当前值是minv+diff.top()，否则当前值即为当前最小值

(4) getMin(): 返回minv即可

|  |
| --- |
| **class MinStack {**  **public:**  **MinStack()** {  diff.push(0);  minv = INT\_MAX;  }    **void push(int x)** {  diff.push(x - minv);  if (x < minv) minv = x;  }    **void pop()** {  if (diff.size()==1) throw "empty stack";  long diff\_top = diff.top();  diff.pop();  if (diff\_top < 0) minv -= diff\_top;  }    **int top()** {  if (diff.size()==1) throw "empty stack";  long diff\_top = diff.top();  return minv + (diff\_top>0?diff\_top:0);  }    **int getMin()** {  return minv;  }    private:  stack<long> diff;  long minv;  **};** |

# 

# 

# 156. Binary Tree Upside Down [Locked]

|  |
| --- |
| Given a binary tree where all the right nodes are either leaf nodes with a sibling  (a left node that shares the same parent node) or empty, flip it upside down and  turn it into a tree where the original right nodes turned into left leaf nodes.  Return the new root. |
| For example:  Given a binary tree {1,2,3,4,5},  1  / \  2 3  / \  4 5  return the root of the binary tree [4,5,2,#,#,3,1].  4  / \  5 2  / \  3 1 |

**Sol: recursion**

递归地构造node->left的upside down view，并返回该树的右下角指针tail。把当前指针的右指针改为tail的左指针。把当前指针的children改为NULL，并append为tail的右指针。

|  |
| --- |
| **TreeNode \*upsideDownBinaryTree(TreeNode \*root)** {  if (root == NULL) return NULL;  TreeNode \*new\_root = NULL;  upsideDownBinaryTree\_recur(root, new\_root);  return new\_root;  }  TreeNode \*upsideDownBinaryTree\_recur(TreeNode \*root, TreeNode\*& new\_root) {  if (root->left == NULL) {  new\_root = root;  }  else {  TreeNode \*tail = upsideDownBinaryTree\_recur(root->left, new\_root);  tail->left = root->right;  tail->right = root;  root->left = root->right = NULL;  }  return root;  } |

# 

# 157. Read N Characters Given Read4 [Locked]

|  |
| --- |
| The API: int read4(char \*buf) reads 4 characters at a time from a file.  The return value is the actual number of characters read. For example, it returns 3 if there is only 3 characters left in the file.  By using the read4 API, implement the function int read(char \*buf, int n) that reads n characters from the file.  Note:  The read function will only be called once for each test case. |
| interface:  /\*\*  \* @param buf Destination buffer  \* @param n Maximum number of characters to read  \* @return The number of characters read  \*/  int read(char \*buf, int n) |

**Sol: modularity**

文件大小可能大于n也可能小于n。因此，最多读(n-1)/4+1次。每次读进字符时，复制min(buf+n-p, m)个数。这里p为当前global buffer的写指针起始位置（初始为buf，每循环+4)，m为read4的返回值。

|  |
| --- |
| **int read(char \*buf, int n)** {  const int nchar = 4;  int len = 0; //total number of char read  int nread = nchar; //number of char that was actually read to the local buffer  int nwrite = nchar; //number of char that was actually written to the global buffer  char \*p = buf;    while (nread == nchar && len < n) {  char lbuf[nchar];  nread = read4(lbuf);  nwrite = min(nread, n-len);  copy(lbuf, lbuf + nwrite, p);  len += nwrite;  p += nwrite;  }  return len;  } |

# 158. Read N Characters Given Read4 II - Call multiple times [Locked]

|  |
| --- |
| The API: int read4(char \*buf) reads 4 characters at a time from a file.  The return value is the actual number of characters read. For example, it returns 3 if there is only 3 characters left in the file.  By using the read4 API, implement the function int read(char \*buf, int n) that reads n characters from the file. |

**Sol: class member**

把lbuf,m,nwrite改为类成员。这样当进入一个新的call时，首先从lbuf开始读min(n, m-nwrite)个字符并把n-min(n, m-nwrite)。如果还要读入更多字符，则还是call上一题的循环。p的初始值为buf + min(n, m-nwrite).

|  |
| --- |
| **class Solution {**  **public:**  **int read(char \*buf, int n)** {  // read remaining characters from local buffer without calling read4  int len = min(nread-nwrite, n);  copy(lbuf+nwrite, lbuf + nwrite + len, buf);  nwrite += len;  // call read4 iteratively  char \*p = buf+len;  while (nread == nchar && len < n) {  nread = read4(lbuf);  nwrite = min(nread, n-len);  copy(lbuf, lbuf + nwrite, p);  len += nwrite;  p += nwrite;  }  return len;  }  private:  static const int nchar = 4;  char lbuf[nchar];  int nwrite = nchar;  int nread = nchar;  **};** |

# 159. Longest Substring with At Most Two Distinct Characters [Locked]

|  |
| --- |
| Given a string, find the length of the longest substring T that contains at most 2 distinct characters. |
| For example, Given s = "eceba",  T is "ece" which its length is 3. |

**Sol: sliding window**

维护一个sliding window，以及这个window里的两个字母c1, c2以及它们的最后一次出现的位置p1, p2。当出现一个新字母时，把sliding window的头移到p1, p2中较前面那个的后一位。

循环不变量: p1= p2=-1 or p1<p2

|  |
| --- |
| **int lengthOfLongestSubstringTwoDistinct(string s)** {  char c1, c2;  int p1 = -1, p2 = -1;  int start = 0, maxlen = 0;  for (int k = 0; k < s.size(); ++k) {  char c = s[k];  if (c == c1) p1 = k;  else if (c == c2) p2 = k;  else {  start = p1+1;  c1 = c;  p1 = k;  }  if (p1 > p2) {  swap(c1, c2);  swap(p1, p2);  }    maxlen = max(maxlen, k - start + 1);  }  return maxlen;  } |

# 160. Intersection of Two Linked Lists

|  |
| --- |
| Write a program to find the node at which the intersection of two singly linked lists begins.  Notes:  If the two linked lists have no intersection at all, return null.  The linked lists must retain their original structure after the function returns.  You may assume there are no cycles anywhere in the entire linked structure.  Your code should preferably run in O(n) time and use only O(1) memory. |
| For example, the following two linked lists:  begin to intersect at the red node. |

**Sol1: hashing**

把链表中的所有结点装进unordered\_set，扫描链表B，返回第一个出现在unordered\_set中的结点。若不存在该结点，则返回NULL。

|  |
| --- |
| ListNode \*getIntersectionNode(ListNode \*headA, ListNode \*headB) {  unordered\_set<ListNode \*> nodes\_in\_A;  for (ListNode \*p = headA; p; p = p->next) nodes\_in\_A.insert(p);  for (ListNode \*p = headB; p; p = p->next)  if (nodes\_in\_A.find(p) != nodes\_in\_A.end()) return p;  return NULL;  } |

**Sol2: fast-slow pointer**

想法是把tail结点链接到第二个链表头，则本问题转化为一个检测链表环的问题。

|  |
| --- |
| **ListNode \*getIntersectionNode1(ListNode \*headA, ListNode \*headB)** {  if (headA == NULL || headB == NULL) return NULL;    // find tail of list A and attach it to the head of listB  ListNode \*tail = headA;  while (tail->next) tail = tail->next;  tail->next = headB;    ListNode \*intersection = detectCycle(headA);  tail->next = NULL;  return intersection;  }  ListNode \*detectCycle(ListNode \*head) {  if (head == NULL) return NULL;  ListNode \*fast, \*slow;  for (fast = slow = head; fast;) {  fast = fast->next;  if (!fast) return NULL;  slow = slow->next;  fast = fast->next;  if (fast == slow) {  for (ListNode \*p = head; p != fast; p = p->next, fast = fast->next);  return fast;  }  }  return NULL;  } |

**Sol3：two pointers**

令两个指针pab先访问第一个链表再访问第二个链表, pba先访问第二个链表，再访问第一个链表。因此两个访问序列的尾部重合。只要检测两者是否相等就好。

|  |
| --- |
| **ListNode \*getIntersectionNode3(ListNode \*headA, ListNode \*headB)** {  ListNode \*pAB, \*pBA;  if (headA == NULL || headB == NULL) return NULL;  for (pAB= headA, pBA = headB; pAB != pBA; ) {  pAB = pAB?pAB->next:headB;  pBA = pBA?pBA->next:headA;  }  return pAB;  } |

# 161. One Edit Distance [Locked]

|  |
| --- |
| Given two strings S and T, determine if they are both one edit distance apart. |

**Sol: simple iteration**

考虑以下可能性：

1. |S.size() - T.size()|>1:直接返回false
2. S.size()=T.size()：检查是否只有一个字母不同
3. S.size()=T.size()+1：找到第一个S和T不同的字母,假设这个字母是插入字母，检查后序的部分。

|  |
| --- |
| **bool isOneEditDistance(string s, string t)** {  if (s.size() < t.size()) return isOneEditDistance(t, s);  if (s.size() > t.size()+1) return false;  int k, j;  bool one\_edit = false;  for (k = 0, j = 0; k < s.size() && j < t.size(); k++, j++) {  if (s[k] !=t[j]) {  if (one\_edit) return false;  if (s.size() > t.size()) j--; //skip  one\_edit = true;  }  }    return j == t.size() && ((k == s.size() && one\_edit) || k==s.size()-1);  } |

# 162. Find Peak Element

|  |
| --- |
| A peak element is an element that is greater than its neighbors.  Given an input array where num[i] ≠ num[i+1], find a peak element and return its index.  The array may contain multiple peaks, in that case return the index to any one of the peaks is fine.  Your solution should be in logarithmic complexity. |
| You may imagine that num[-1] = num[n] = -∞.  For example, in array [1, 2, 3, 1], 3 is a peak element and your function should return the index number 2. |

**Sol: binary search**

我们需要确保范围[first, last]只间有一个peak。考虑几种情况：

1. nums[mid] < nums[first]：此时或者从first->mid单调递减，或者先增后减，因此[first, mid-1]满足此条件。
2. nums[mid] < nums[second]: 与(1)对称
3. nums[last]<=nums[mid]<nums[mid+1]:因此peak必出现在[mid+1, last]
4. nums[first]<=nums[mid]<nums[mid-1]:与(3)对称
5. nums[mid]是局部极大值点：返回mid

在此过程中我们需要保证first, last之间至少间隔1,从而条件(3)(4)的判断不会出问题

|  |
| --- |
| **int findPeakElement(vector<int>& nums)** {  int first = 0, last = nums.size()-1;  while (first < last-1) {  int mid = (first + last)/2;  if (nums[mid] < nums[first]) last = mid-1;  else if (nums[mid] < nums[last]) first = mid + 1;  else if (nums[mid] < nums[mid+1]) first = mid + 1;  else if (nums[mid] < nums[mid-1]) last = mid-1;  else return mid;  }    return nums[first] > nums[last]? first : last;  } |

# 163. Missing Ranges [Locked]

|  |
| --- |
| Given a sorted integer array where the range of elements are [0, 99] inclusive, return its missing ranges.  For example, given [0, 1, 3, 50, 75], return [“2”, “4->49”, “51->74”, “76->99”] |

**Sol: simple iteraton**

循环地先搜索missing range的起点，从当前已覆盖位置last（初始为-1)开始，每次检查nums[k]<=last+1，如果是则更新last=nums[k]，否则设置start为已覆盖位置+1。最后一个数为nums[k+1]-1，或99.

|  |
| --- |
| **vector<string> findMissingRanges(vector<int>& nums, int lower, int upper)** {  int last = lower-1;  vector<string> result;  for (int v : nums) {  if (v > last+1) result.push\_back(rangeStr(last+1, v-1));  last = v;  }  if (last < upper) result.push\_back(rangeStr(last+1, upper));  }  inline string rangeStr(int left, int right) {  string ret = to\_string(left);  if (right > left) ret += "->" + to\_string(right);  return ret;  } |

# 164. Maximum Gap

|  |
| --- |
| Given an unsorted array, find the maximum difference between the successive elements in its sorted form.  Try to solve it in linear time/space.  Return 0 if the array contains less than 2 elements.  You may assume all elements in the array are non-negative integers and fit in the 32-bit signed integer range. |

**Sol: bucket sort**

数值的range是[minv, maxv]，假设有N个数，我们知道最大距离至少是(maxv-minv)/(N-1)。因此我们设置N个连续、不重叠的bucket放置数组中的数，每个bucket表示一个宽为width的区间。通过扫描一遍数组我们可以知道每个区间的最大数、最小数、以及有没有数。然后我们只需要检查相邻两有数区间的左边最大数和右边最小数之差即可（桶内的值间距离可以忽略不计）。

|  |
| --- |
| **int maximumGap(vector<int>& nums)** {  if (nums.size() < 2) return 0;  int maxv = 0, minv = INT\_MAX;    for (int v : nums) {  maxv = max(maxv, v);  minv = min(minv, v);  }    if (maxv == minv) return 0;    int width = max(1, (maxv - minv)/(int(nums.size())-1));  int nbuckets = (maxv - minv)/width + 1;  vector<int> maxvb(nbuckets, minv);  vector<int> minvb(nbuckets, maxv);  for (int v : nums) {  int bucket\_id = (v-minv) / width;  maxvb[bucket\_id] = max(maxvb[bucket\_id], v);  minvb[bucket\_id] = min(minvb[bucket\_id], v);  }    int max\_gap = 0;  int last\_max = maxvb[0];  for (int k = 1; k <nbuckets; ++k) {  if (minvb[k] <= maxvb[k]) {  max\_gap = max(max\_gap, minvb[k] - last\_max);  last\_max = maxvb[k];  }  }  return max\_gap;  } |

# 165. Compare Version Numbers

|  |
| --- |
| Compare two version numbers version1 and version2.  If version1 > version2 return 1, if version1 < version2 return -1, otherwise return 0.  You may assume that the version strings are non-empty and contain only digits and the . character.  The . character does not represent a decimal point and is used to separate number sequences.  For instance, 2.5 is not "two and a half" or "half way to version three", it is the fifth second-level revision of the second first-level revision. |
| Here is an example of version numbers ordering:  0.1 < 1.1 < 1.2 < 13.37 |

**corner cases：**

(1)1.01 = 1.1?

(2)1 = 1.0

**Sol: string ops**

依次分割出数字进行比较即可。下面的实现利用replace, istringstream来做，代码比较简洁。

|  |
| --- |
| **int compareVersion(string version1, string version2)** {  replace(version1.begin(), version1.end(), '.', ' ');  replace(version2.begin(), version2.end(), '.', ' ');  istringstream iss1(version1), iss2(version2);  while (true) {  int n1, n2;  bool e1 = (iss1 >> n1), e2 = (iss2 >> n2);  if (!e1 && !e2) return 0;  if (!e1) n1 = 0;  if (!e2) n2 = 0;  if (n1 > n2) return 1;  if (n1 < n2) return -1;  }  return 0;  } |

# 166. Fraction to Recurring Decimal

|  |
| --- |
| Given two integers representing the numerator and denominator of a fraction, return the fraction in string format.  If the fractional part is repeating, enclose the repeating part in parentheses. |
| For example,  Given numerator = 1, denominator = 2, return "0.5".  Given numerator = 2, denominator = 1, return "2".  Given numerator = 2, denominator = 3, return "0.(6)". |

**corner cases：**

1. 被除数为0：返回"0"
2. 除数为0: 抛出异常
3. 两操作数符号不同：结果以"-"开始
4. 两操作数整除：不需要小数部分
5. 溢出：INT\_MIN/-1需输出正确字符串

**Sol: math ops+hashing**

以上情况决定了小数点以前的部分。

下面我们考虑小数点以后的部分：我们用正常的数字计算来一位位算出小数部分。对每一位，我们把当前余数＊10, 整除被除数的结果加入输出，余数留给下一位。

如果我们发现余数重复出现，说明发现循环节。这可以用一个unordered\_map定位。

如果余数为0,则我们不需要插入括号，直接返回输出即可。

|  |
| --- |
| **string fractionToDecimal(int numerator, int denominator)** {  //we ignore the case denominator is 0  if (numerator == 0) return "0";  if (denominator == 0) throw invalid\_argument("denominator cannot be 0");  long long num = llabs(numerator), denom = llabs(denominator);  string ipart = ((numerator ^ denominator) < 0? "-":"") + to\_string(num/denom);  num %= denom;  string fpart = num? comp\_fpart(num, denom): "";  return ipart + fpart;  }  string comp\_fpart(long long num, long long denom) {  string retval(".");  unordered\_map<int, size\_t> pos;  for (size\_t k = 1; num>0; ++k, num%=denom) {  auto it = pos.find(num);  if (it == pos.end()) {  pos[num] = k;  num \*= 10;  retval += ('0'+num/denom);  }  else {  retval += ')';  retval.insert(it->second, "(");  break;  }  }  return retval;  } |

# 167. Two Sum II - Input array is sorted [Locked]

|  |
| --- |
| Given an array of integers that is already sorted in ascending order, find two numbers such that they add up to a specific target number.  The function twoSum should return indices of the two numbers such that they add up to the target, where index1 must be less than index2. Please note that your returned answers (both index1 and index2) are not zero-based.  You may assume that each input would have exactly one solution.  Input: numbers={2, 7, 11, 15}, target=9  Output: index1=1, index2=2 |

**Sol: two pointer**

两头夹逼即可。之前3sum已经用到过。

|  |
| --- |
| **vector<int> twoSum(vector<int>& numbers, int target)** {  for (int l = 0, r = numbers.size()-1; l < r; ) {  if (numbers[l]+numbers[r] == target) return vector<int>{l+1, r+1};  if (numbers[l]+numbers[r] < target) l++; else r--;  }  return vector<int>{}; //in case there's no solution  } |

# 168. Excel Sheet Column Title

|  |
| --- |
| Given a positive integer, return its corresponding column title as appear in an Excel sheet. |
| For example:  1 -> A  2 -> B  3 -> C  ...  26 -> Z  27 -> AA  28 -> AB |

**Sol: math ops**

本题类似是十进制数到26进制。每次通过模26试图确定最低位,由于可能的取值是1-26,因此要减1.

|  |
| --- |
| **string convertToTitle(int n)** {  string result("");  for (; n > 0; n /= 26) result += 'A' + (--n%26);  reverse(result.begin(), result.end());  return result;  } |

# 169. Majority Element

|  |
| --- |
| Given an array of size n, find the majority element. The majority element is the element that appears more than ⌊ n/2 ⌋ times.  You may assume that the array is non-empty and the majority element always exist in the array. |

**Sol1: histogram(hashing)**

维护一个histogram，从而确定majority element。

|  |
| --- |
| **int majorityElement(vector<int>& nums)** {  if (nums.size() <= 2) return nums[0];  int thresh = nums.size()/2;  unordered\_map<int, int> hist;  for (int v : nums) {  auto it = hist.find(v);  if (it == hist.end()) hist[v] = 1;  else {  it->second++;  if (it->second > thresh) return v;  }  }  return -1;  } |

**Sol2: voting**

如果我们把值不相同的数尽量结对，则可以抵消所有minority elements和部分majority elements。因为数组中数可能连续出现，所以我们维护一个计数器和当前的majority candidate，当出现candiate时计数器加一（还未结对），出现非candidate时计数器减一(和candidate结对抵消），当计数器为0时，设置当前元素为candidate。

|  |
| --- |
| **int majorityElement(vector<int>& nums)** {  int candidate = 0, count = 0;  for (int v : nums) {  if (count == 0) {  candidate = v;  count = 1;  }  else {  if (v == candidate) count++;  else count--;  }  }  return candidate;  } |

# 170. Two Sum III - Data structure design [Locked]

|  |
| --- |
| Design and implement a TwoSum class. It should support the following operations:add and find.  add - Add the number to an internal data structure.  find - Find if there exists any pair of numbers which sum is equal to the value. |
| For example,  add(1); add(3); add(5);  find(4) -> true  find(7) -> false |

**Sol1: histograming(hashing)**

用一个unordered\_map<int, int>存储每个元素的count。在find时遍历所有元素v，并寻找target-v。如target==2\*v，则检查是否元素个数>=2。这个算法空间复杂度是O(n)。时间复杂度是find:(O(n))，add: O(1)

|  |
| --- |
| **class TwoSum** {  unordered\_map<int, int> cnt;  **public:**  **void add(int number)** {  cnt[number]++;  }  **bool find(int value)** {  for (auto p : cnt) {  if (p.first\*2!=value && cnt.find(value-p.first) != cnt.end()) return true;  if (p.first\*2==value && p.second>1) return true;  }  return false;  }  **};** |

**Sol2: hashing**

用一个unordered\_set<int>存储所有可能的和。add时增加新产生的和。这个算法空间复杂度O(n2)，时间复杂度add是O(n)的，find是O(1)的。

|  |
| --- |
| **class TwoSum** {  list<int> nums;  unordered\_set<int> sums;  **public:**  **void add(int number)** {  for (int v : nums) sums.insert(v + number);  nums.push\_back(number);  }  **bool find(int value)** {  return sums.find(value) != sums.end();  }  **};** |

# 171. Excel Sheet Column Number

|  |
| --- |
| Given a column title as appear in an Excel sheet, return its corresponding column number. |
| For example:  A -> 1  B -> 2  C -> 3  ...  Z -> 26  AA -> 27  AB -> 28 |

**Sol: combinatorial math**

从右数第k位的contribution是n\*26k-1,从右到左扫描所有位，并累计它们对结果的贡献。

|  |
| --- |
| **int titleToNumber(string s)** {  int sum = 0, base = 1;  for (auto it = s.rbegin(); it != s.rend(); ++it, base \*= 26) sum += (\*it-'A'+1) \* base;  return sum;  } |

# 172. Factorial Trailing Zero

|  |
| --- |
| Given an integer n, return the number of trailing zeroes in n!.  Note: Your solution should be in logarithmic time complexity. |

**Sol: combinatorial math**

由于因子2比5频繁得多，我们只需要数有多少5。5k累计k次。

|  |
| --- |
| **int trailingZeroes(int n)** {  int sum = 0;  for (; n; n/=5) sum += n/5;  return sum;  } |

# 173. Binary Search Tree Iterator

|  |
| --- |
| Implement an iterator over a binary search tree (BST). Your iterator will be initialized with the root node of a BST.  Calling next() will return the next smallest number in the BST.  Note: next() and hasNext() should run in average O(1) time and uses O(h) memory, where h is the height of the tree. |

**Sol: inorder traversal with stack**

|  |
| --- |
| **class BSTIterator** {  **public:**  **BSTIterator(TreeNode \*root)** {  r = root;  }  /\*\* @return whether we have a next smallest number \*/  **bool hasNext()** {  return (r != NULL) || (!stk.empty());  }  /\*\* @return the next smallest number \*/  **int next()** {  int retval = 0;  if (r == NULL) {  retval = stk.top()->val;  r = stk.top()->right;  stk.pop();  }  else {  while (r->left) {  stk.push(r);  r = r->left;  }  retval = r->val;  r = r->right;  }  return retval;  }  private:  stack<TreeNode \*> stk;  TreeNode \*r;  **};** |

# 174. Dungeon Game

|  |
| --- |
| The demons had captured the princess (P) and imprisoned her in the bottom-right corner of a dungeon. The dungeon consists of M x N rooms laid out in a 2D grid. Our valiant knight (K) was initially positioned in the top-left room and must fight his way through the dungeon to rescue the princess.  The knight has an initial health point represented by a positive integer. If at any point his health point drops to 0 or below, he dies immediately.  Some of the rooms are guarded by demons, so the knight loses health (negative integers) upon entering these rooms; other rooms are either empty (0's) or contain magic orbs that increase the knight's health (positive integers).  In order to reach the princess as quickly as possible, the knight decides to move only rightward or downward in each step.  Write a function to determine the knight's minimum initial health so that he is able to rescue the princess.  Notes:  The knight's health has no upper bound.  Any room can contain threats or power-ups, even the first room the knight enters and the bottom-right room where the princess is imprisoned. |
| For example, given the dungeon below, the initial health of the knight must be at least 7 if he follows the optimal path RIGHT-> RIGHT -> DOWN -> DOWN.   |  |  |  | | --- | --- | --- | | -2  (K) | -3 | 3 | | -5 | -10 | 1 | | 10 | 30 | -5  (P) | |

**Sol: DP**

从右下角到左上角计算从点(k,j)到达右下角需要的最少的生命值。分两种情况：下一步是下方、右方。因此cost[k][j] = max(1, max(1-dungeon[k][j], min(cost[k+1][j], cost[k][j+1])-dungeon[k][j])) = max(1, min(cost[k+1][j], cost[k][j+1])-dungeon[k][j]).

|  |
| --- |
| **int calculateMinimumHP(vector<vector<int>>& dungeon)** {  if (dungeon.size() == 0 && dungeon[0].size() == 0) return 0;  const int M = dungeon.size(), N = dungeon[0].size();  vector<vector<int>> cost(M, vector<int>(N, 0));    for (int k = M-1; k >= 0; --k) {  for (int j = N-1; j >= 0; --j) {  if (k == M-1 && j == N-1) cost[k][j] = max(1, 1-dungeon[k][j]);  else if (k == M-1) cost[k][j] = max(1, cost[k][j+1]-dungeon[k][j]);  else if (j == N-1) cost[k][j] = max(1, cost[k+1][j]-dungeon[k][j]);  else cost[k][j] = max(1, min(cost[k][j+1], cost[k+1][j])-dungeon[k][j]);  }  }  return cost.front().front();  } |

# 175 - 178. [Database problems]

# 179. Largest Number

|  |
| --- |
| Given a list of non negative integers, arrange them such that they form the largest number.  Note: The result may be very large, so you need to return a string instead of an integer. |
| For example, given [3, 30, 34, 5, 9], the largest formed number is 9534330. |

**Sol: sorting + comparator**

比较两个数的大小：把它们串联比较起来即可。这样可以实现一个comparator，然后我们用这个comparator sort并串联起来。特殊情况：数组中都是0,返回0.

|  |
| --- |
| bool smaller(int n1, int n2) {  string s1 = to\_string(n1) + to\_string(n2);  string s2 = to\_string(n2) + to\_string(n1);  return stoll(s1, NULL) > stoll(s2, NULL);  }  **string largestNumber(vector<int>& nums)** {  sort(nums.begin(), nums.end(), smaller);  if (nums[0] == 0) return "0";  string s("");  for (int v : nums) s += to\_string(v);  return s;  } |

# 180-185: [Database problems]

# 186. Reverse Words in a String II [Locked]

|  |
| --- |
| Given an input string, reverse the string word by word. A word is defined as a sequence of non-space characters.  The input string does not contain leading or trailing spaces and the words are always separated by a single space.  For example,  Given s = "the sky is blue",  return "blue is sky the".  Could you do it in-place without allocating extra space? |

**Sol: global + local reverse**

同151题。

|  |
| --- |
| **void reverseWords(string &s)** {  for (int k = 0; k < s.size();) {  int j;  for (j = k+1; j < s.size() && s[j] != ' '; ++j);  reverse(s.begin()+k, s.begin()+j);  k = j+1;  }  reverse(s.begin(), s.end());  } |

# 187. Repeated DNA Sequences

|  |
| --- |
| All DNA is composed of a series of nucleotides abbreviated as A, C, G, and T, for example: "ACGAATTCCG". When studying DNA, it is sometimes useful to identify repeated sequences within the DNA.  Write a function to find all the 10-letter-long sequences (substrings) that occur more than once in a DNA molecule. |
| For example,  Given s = "AAAAACCCCCAAAAACCCCCCAAAAAGGGTTT",  Return:  ["AAAAACCCCC", "CCCCCAAAAA"]. |

**Sol: hashing**

从左到右扫描，把子串存储到hash table。因为长度为10的字符串共有410种可能性，我们用一个32位长整数描述它。

|  |
| --- |
| **vector<string> findRepeatedDnaSequences(string s)** {  if (s.size() <= 10) return vector<string>{};  unordered\_map<long, int> hist;    // scan windows of length of 10 and encode each window into a number  int k = 0, key = 0;  for (k = 0; k < 10; ++k) updateKey(s[k], key);  hist[key] = 1;  for (k = 10; k < s.size(); ++k) {  updateKey(s[k], key);  auto it = hist.find(key);  if (it == hist.end()) hist[key] = 1; else it->second++;  }    // scan all windows and output those with multiple occurrence  vector<string> result;  for (auto it = hist.begin(); it != hist.end(); it++) {  if (it->second > 1) result.push\_back(decode(it->first));  }  return result;  }  void updateKey(char c, int &key) {  key &= ((1 << 18) - 1);  int v = 0;  switch (c) {  case 'A': v = 0; break;  case 'C': v = 1; break;  case 'G': v = 2; break;  case 'T': v = 3;  }  key = (key << 2) + v;  }  string decode(long code) {  string s("");  for (int k = 0; k < 10; ++k, code /= 4) {  char c = ' ';  switch (code%4) {  case 0: c = 'A'; break;  case 1: c = 'C'; break;  case 2: c = 'G'; break;  case 3: c = 'T';  };  s += c;  }  reverse(s.begin(), s.end());  return s;  } |

# 

# 188. Best Time to Buy and Sell Stock IV

|  |
| --- |
| Say you have an array for which the ith element is the price of a given stock on day i.  Design an algorithm to find the maximum profit. You may complete at most k transactions.  Note:  You may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again). |

**Sol: DP**

我们用矩阵global[i][k]表示到第i天为止k次交易的最大利润。用矩阵local[i][k]表示第天结束第k次交易的最大利润。则

global[i][k] = max(global[i-1][k], local[i][k]) (最优的交易结束于第k天或第k天以前）

local[i][k] = max(global[i-1][k-1]+diff, local[i-1][k]+diff) (第i-1天没有持有，或者有持有）

需要注意的是如果天数少于k,可以简化为不限制交易次数的版本。

|  |
| --- |
| **int maxProfit(int k, vector<int>& prices)** {  int N = prices.size();  if (N < 2 || k <= 0) return 0;  if (N-1 <= k) return maxProfit2(prices);    vector<vector<int>> local(N, vector<int>(k+1, 0));  vector<vector<int>> global(N, vector<int>(k+1, 0));  for (int j = 1; j < N; ++j) {  int diff = prices[j] - prices[j-1];  for (int l = 1; l <= k; ++l) {  local[j][l] = max(global[j-1][l-1]+diff, local[j-1][l]+diff);  global[j][l] = max(global[j-1][l], local[j][l]);  }  }  return global.back().back();  }  int maxProfit2(vector<int>& prices) {  int sum = 0;  for (int k = 1; k < prices.size(); ++k) sum += max(prices[k]-prices[k-1], 0);  return sum;  } |

# 189. Rotate Array

|  |
| --- |
| Rotate an array of n elements to the right by k steps.  For example, with n = 7 and k = 3, the array [1,2,3,4,5,6,7] is rotated to [5,6,7,1,2,3,4].  Note:  Try to come up as many solutions as you can, there are at least 3 different ways to solve this problem. |

**Sol: global+local reverese**

首先k%=n以确保0<=k<n。这时我们可以把问题转化为置换左边n-k个数和右边k个数。这可以通过三次逆转实现。

|  |
| --- |
| **void rotate(vector<int>& nums, int k)** {  if (nums.size() == 0) return;  k %= nums.size();  reverse(nums.begin(), nums.end());  reverse(nums.begin(), nums.begin()+k);  reverse(nums.begin()+k, nums.end());  } |

# 190. Reverse Bits

|  |
| --- |
| Reverse bits of a given 32 bits unsigned integer.  Follow up:  If this function is called many times, how would you optimize it? |
| For example,  given input 43261596 (represented in binary as 00000010100101000001111010011100),  return 964176192 (represented in binary as 00111001011110000010100101000000). |

**Sol1: math ops**

顺序扫描每一位，并把从右数第k位(0 based)的值放入从右数第31-k位。

|  |
| --- |
| uint32\_t reverseBits(uint32\_t n) {  int result = 0;  for (int k = 31; n; --k, n/=2) result |= (n%2)<<k;  return result;  } |

**Sol2: hashing**

预存8位的逆转到一个char array，然后把n的第1－4个字节的逆转分别放在n的第4－1个字节。

# 191. Number of 1 Bits

|  |
| --- |
| Write a function that takes an unsigned integer and returns the number of ’1' bits it has (also known as the Hamming weight).  For example, the 32-bit integer ’11' has binary representation 00000000000000000000000000001011, so the function should return 3. |

**Sol1: bit ops**

从右到左取出每一位。如果是1，则计数器加1.

|  |
| --- |
| **int hammingWeight(uint32\_t n)** {  int count = 0;  for (int k = 0; k<32; ++k) count += (n>>k)%2;  return count;  } |

**Sol2: (more advanced) bit ops**

循环讲数最右一位的1置0直到该数为0, 计数器加1。

|  |
| --- |
| **int hammingWeight(uint32\_t n)** {  int count = 0;  for (; n; n &= (n-1)) count ++;  return count;  } |

# 192-195: [Shell problems]

# 196-197: [Database problems]

# 198. House Robber

|  |
| --- |
| You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security system connected and it will automatically contact the police if two adjacent houses were broken into on the same night.  Given a list of non-negative integers representing the amount of money of each house, determine the maximum amount of money you can rob tonight without alerting the police. |

**Sol: DP**

令M[i]为第i家为最后一家的最大收益，则M[i] = max( M[i-1], M[i-2]+money[i]). 由于递归结构是local的且我们只要最后一个M，因此不用存储M.

|  |
| --- |
| **int rob(vector<int>& nums)** {  int last = 0, last2 = 0;  for (int v : nums) {  int cur = max(last2 + v, last);  last2 = last; last = cur;  }  return last;  } |

# 199. Binary Tree Right Side View

|  |
| --- |
| Given a binary tree, imagine yourself standing on the right side of it, return the values of the nodes you can see ordered from top to bottom. |
| For example:  Given the following binary tree,  1 <---  / \  2 3 <---  \ \  5 4 <---  You should return [1, 3, 4]. |

**Sol: BFS**

遍历时输出每层最后一个数。

|  |
| --- |
| **vector<int> rightSideView(TreeNode\* root)** {  vector<int> result;  vector<TreeNode\* > pre;  if (root) pre.push\_back(root);    while (!pre.empty()) {  vector<TreeNode\* > cur;  result.push\_back(pre.back()->val);  for (TreeNode\* t : pre) {  if (t->left) cur.push\_back(t->left);  if (t->right) cur.push\_back(t->right);  }  swap(pre, cur);  }  return result;  } |

# 200. Number of Islands

|  |
| --- |
| Given a 2d grid map of '1's (land) and '0's (water), count the number of islands. An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water. |
| Example 1:  11110  11010  11000  00000  Answer: 1  Example 2:  11000  11000  00100  00011  Answer: 3 |

**Sol1：flood fill**

可以用BFS或DFS实现。

|  |
| --- |
| **int numIslands(vector<vector<char>>& grid)** {  if (grid.size() == 0 || grid[0].size() == 0) return 0;  int count = 0;  for (int k = 0; k < grid.size(); ++k) {  for (int j = 0; j < grid[0].size(); ++j) {  if (grid[k][j] != '1') continue;  eraseIsland(grid, k, j);  ++count;  }  }  return count;  }  void eraseIsland(vector<vector<char>>& grid, int r, int c) {  const int R = grid.size(), C = grid[0].size();  stack<pair<int, int>> stk;  stk.push(make\_pair(r, c));    while (!stk.empty()) {  int r = stk.top().first;  int c = stk.top().second;  grid[r][c] = '.';  stk.pop();  if (r>0 && grid[r-1][c] == '1') stk.push(make\_pair(r-1, c));  if (r<R-1 && grid[r+1][c] == '1') stk.push(make\_pair(r+1, c));  if (c>0 && grid[r][c-1] == '1') stk.push(make\_pair(r, c-1));  if (c<C-1 && grid[r][c+1] == '1') stk.push(make\_pair(r, c+1));  }  } |

**Sol2： union-find**

见305 Number of Islands II