LeetCode 题解

戴方勤 (soulmachine@gmail.com)

https://gitcafe.com/soulmachine/LeetCode

最后更新 2013-9-11

版权声明

内容简介

本书的目标读者是准备去北美找工作的码农，也适用于在国内找工作的码农，以及刚接触 ACM 算法竞赛的新手。

本书包含了 LeetCode Online Judge(http://leetcode.com/onlinejudge) 所有题目的答案，所有代码经过精心编写，编码规范良好，适合读者反复揣摩，模仿，甚至在纸上默写。

全书的代码，使用 C++ 11 的编写，并在 LeetCode Online Judge 上测试通过。本书中的代码规范，跟在公司中的工程规范略有不同，为了使代码短（方便迅速实现）：

- 所有代码都是单一文件。这是因为一般 OJ 网站，提交代码的时候只有一个文本框，如果还是按照标准做法，比如分为头文件.h 和源代码.cpp，无法在网站上提交；
- 大量使用 STL，让代码更短，shorter is better；
- 不提倡防御式编程。不需要检查 malloc()/new 返回的指针是否为 NULL；不需要检查内部函数入口参数的有效性；使用纯 C 基于对象编程时，调用对象的成员方法，不需要检查对象自身是否为 NULL。

本手册假定读者已经学过《数据结构》[①]，《算法》[②] 这两门课，熟练掌握 C++ 或 Java。

GitCafe 地址

本书是开源的，项目地址：https://gitcafe.com/soulmachine/LeetCode

[①]《数据结构》，严蔚敏等著，清华大学出版社，http://book.douban.com/subject/2024655/

[②]《Algorithms》，Robert Sedgewick, Addison-Wesley Professional, http://book.douban.com/subject/4854123/

目录

第 1 章 线性表

这类题目考察线性表的操作，例如，数组，单链表，双向链表等。

1.1 Add Two Numbers

描述

You are given two linked lists representing two non-negative numbers. The digits are stored in reverse order and each of their nodes contain a single digit. Add the two numbers and return it as a linked list.

Input: `(2 -> 4 -> 3) + (5 -> 6 -> 4)`

Output: `7 -> 0 -> 8`

分析

跟 Add Binary（见 §2.1）很类似

代码

```
struct ListNode {
    int val;
    ListNode *next;
    ListNode(int x) : val(x), next(NULL) { }
};

//LeetCode, Add Two Numbers
//跟 Add Binary 很类似
class Solution {
public:
    ListNode *addTwoNumbers(ListNode *l1, ListNode *l2) {
        ListNode* head = new ListNode(-1); // 头节点
        ListNode* pre = head;
        ListNode *pa = l1, *pb = l2;
        int carry = 0;
        while (pa != NULL || pb != NULL) {
            int av = pa == NULL ? 0 : pa->val;
            int bv = pb == NULL ? 0 : pb->val;
            ListNode* node = new ListNode((av + bv + carry) % 10);
            carry = (av + bv + carry) / 10;
            pre->next = node; // 尾插法
            pre = pre->next;
            pa = pa == NULL ? NULL : pa->next;
            pb = pb == NULL ? NULL : pb->next;
        }
        if (carry > 0)
            pre->next = new ListNode(1);
        pre = head->next;
        delete head;
        return pre;
    }
};
```

相关题目

- Add Binary，见 §2.1

第 2 章
字符串

这类题目考察字符串的操作。

2.1 Add Binary

描述

Given two binary strings, return their sum (also a binary string).

For example,

```
a = "11"
b = "1"
```

Return `"100"`.

分析

无

代码

```
//LeetCode, Add Binary
class Solution {
public:
    string addBinary(string a, string b) {
        string result;
        int maxL = a.size() > b.size() ? a.size() : b.size();
        std::reverse(a.begin(), a.end());
        std::reverse(b.begin(), b.end());
        int carry = 0;
        for (int i = 0; i < maxL; i++) {
            int ai = i < a.size() ? a[i] - '0' : 0;
            int bi = i < b.size() ? b[i] - '0' : 0;
            int val = (ai + bi + carry) % 2;
            carry = (ai + bi + carry) / 2;
            result.insert(result.begin(), val + '0');
        }
        if (carry == 1) {
            result.insert(result.begin(), '1');
        }
        return result;
    }
};
```

相关题目

- Add Two Numbers，见 §1.1

第 3 章 树

3.1 Binary Tree Level Order Traversal

描述

Given a binary tree, return the level order traversal of its nodes' values. (ie, from left to right, level by level).

For example: Given binary tree `{3,9,20,#,#,15,7}`,

```
    3
   / \
  9  20
    /  \
   15   7
```

return its level order traversal as:

```
[
  [3],
  [9,20],
  [15,7]
]
```

分析

无

代码

```
struct TreeNode {
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};

//LeetCode, Binary Tree Level Order Traversal
class Solution {
public:
    vector<vector<int> > levelOrder(TreeNode *root) {
        vector<vector<int> > result;
        if(root == NULL) return result;

        queue<TreeNode*> queToPush, queToPop;
        queToPop.push(root);
        while (!queToPop.empty()) {
            vector<int> level; // elments in level level

            while (!queToPop.empty()) {
                TreeNode* node = queToPop.front();
                queToPop.pop();
                level.push_back(node->val);
                if (node->left != NULL) queToPush.push(node->left);
                if (node->right != NULL) queToPush.push(node->right);
            }
            result.push_back(level);
            swap(queToPush, queToPop); //!!! how to use swap
        }
        return result;
    }
};
```

相关题目

- Binary Tree Level Order Traversal II，见 §3.2
- Binary Tree Zigzag Level Order Traversal，见 §3.3

3.2 Binary Tree Level Order Traversal II

描述

Given a binary tree, return the bottom-up level order traversal of its nodes' values. (ie, from left to right, level by level from leaf to root).

For example: Given binary tree `{3,9,20,#,#,15,7}`,

```
   3
  / \
 9  20
   /  \
  15   7
```

return its bottom-up level order traversal as:

```
[
  [15,7]
  [9,20],
  [3],
]
```

分析

在 Binary Tree Level Order Traversal I（见 §3.1）的基础上，用一个 list 作为栈，每次在头部插入，就可以实现倒着输出

代码

```
struct TreeNode {
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) { }
};

//LeetCode, Binary Tree Level Order Traversal II
//在 Binary Tree Level Order Traversal I 的基础上，用一个 list 作为栈
//每次在头部插入，就可以实现倒着输出
class Solution {
public:
    vector<vector<int> > levelOrderBottom(TreeNode *root) {
        list<vector<int> > retTemp;

        queue<TreeNode *> q;
        q.push(root);
        q.push(NULL); // level separator

        vector<int> level;  // elements in one level
        while(!q.empty()) {
            TreeNode *cur = q.front();
            q.pop();
            if(cur) {
                level.push_back(cur->val);
                if(cur->left) q.push(cur->left);
                if(cur->right) q.push(cur->right);
            } else {
                if(level.size() > 0) {
                    retTemp.push_front(level);
                    level.erase(level.begin(),level.end());
                    q.push(NULL);
                }
            }
        }

        vector<vector<int> > ret;
        for(list<vector<int> >::iterator it = retTemp.begin();
                it != retTemp.end(); ++it) {
            ret.push_back(*it);
        }
        return ret;
    }
};
```

相关题目

- Binary Tree Level Order Traversal，见 §3.1
- Binary Tree Zigzag Level Order Traversal，见 §3.3

3.3 Binary Tree Zigzag Level Order Traversal

描述

Given a binary tree, return the zigzag level order traversal of its nodes' values. (ie, from left to right, then right to left for the next level and alternate between).

For example: Given binary tree `3,9,20,#,#,15,7`,

```
    3
   / \
  9  20
    /  \
   15   7
```

return its zigzag level order traversal as:

```
[
  [3],
  [20,9],
  [15,7]
]
```

分析

广度优先遍历，用一个 bool 记录是从左到右还是从右到左，每一层结束就翻转一下。

代码

```
struct TreeNode {
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode(int x) :
            val(x), left(NULL), right(NULL) {
    }
};

//LeetCode, Binary Tree Zigzag Level Order Traversal
//广度优先遍历，用一个 bool 记录是从左到右还是从右到左，每一层结束就翻转一下。
class Solution {
public:
    vector<vector<int> > zigzagLevelOrder(TreeNode *root) {
        vector<vector<int> > result;
        if (NULL == root) return result;
        queue<TreeNode*> q;
        bool l2r = true;  //left to right
        vector<int> level;  // one level's elements

        q.push(root);
        q.push(NULL);  // level separator
        while (!q.empty()) {
            TreeNode *cur = q.front();
            q.pop();
            if (cur) {
                level.push_back(cur->val);
                if (cur->left) q.push(cur->left);
                if (cur->right) q.push(cur->right);
            } else {
                if (l2r) {
                    result.push_back(level);
                } else {
                    vector<int> temp;
                    for (int i = level.size() - 1; i >= 0; --i) {
                        temp.push_back(level[i]);
                    }
                    result.push_back(temp);
                }
                level.clear();
                l2r = !l2r;

                if (q.size() > 0) q.push(NULL);
            }
        }

        return result;
    }
};
```

相关题目

- Binary Tree Level Order Traversal，见 §3.1
- Binary Tree Level Order Traversal II，见 §3.2

第 4 章
排序

4.1 Merge Sorted Array

描述

Given two sorted integer arrays A and B, merge B into A as one sorted array.

Note: You may assume that A has enough space to hold additional elements from B. The number of elements initialized in A and B are m and n respectively.

分析

无

代码

```
//LeetCode, Merge Sorted Array
class Solution {
public:
    void merge(int A[], int m, int B[], int n) {
        int ia = m - 1, ib = n - 1, icur = m + n - 1;
        while(ia >= 0 && ib >= 0) {
            A[icur--] = A[ia] >= B[ib] ? A[ia--] : B[ib--];
        }
        while(ib >= 0) {
            A[icur--] = B[ib--];
        }
    }
};
```

相关题目

- Merge Two Sorted Lists，见 §4.2
- Merge k Sorted Lists，见 §4.3

4.2 Merge Two Sorted Lists

描述

Merge two sorted linked lists and return it as a new list. The new list should be made by splicing together the nodes of the first two lists.

分析

无

代码

```
struct ListNode {
    int val;
    ListNode *next;
    ListNode(int x) :
            val(x), next(NULL) {
    }
};

//LeetCode, Merge Two Sorted Lists
class Solution {
public:
    ListNode *mergeTwoLists(ListNode *l1, ListNode *l2) {
        ListNode* head = new ListNode(-1);
        ListNode* p = head;
        while (l1 != NULL || l2 != NULL) {
            int val1 = l1 == NULL ? INT_MAX : l1->val;
            int val2 = l2 == NULL ? INT_MAX : l2->val;
            if (val1 <= val2) {
                p->next = l1;
                l1 = l1->next;
            } else {
                p->next = l2;
                l2 = l2->next;
            }
            p = p->next;
        }
        p = head->next;
        delete head;
        return p;
    }
};
```

相关题目

- Merge Sorted Array §4.1
- Merge k Sorted Lists，见 §4.3

4.3 Merge k Sorted Lists

描述

Merge k sorted linked lists and return it as one sorted list. Analyze and describe its complexity.

分析

可以复用 Merge Two Sorted Lists（见 §4.2）的函数

代码

```
struct ListNode {
    int val;
    ListNode *next;
    ListNode(int x) :
            val(x), next(NULL) {
    }
};

//LeetCode, Merge k Sorted Lists
class Solution {
public:
    ListNode *mergeKLists(vector<ListNode *> &lists) {
        if (lists.size() == 0)
            return NULL;
        ListNode *p = lists[0];
        for (int i = 1; i < lists.size(); i++) {
            p = mergeTwoLists(p, lists[i]);
        }
        return p;
    }

    // Merge Two Sorted Lists
    ListNode *mergeTwoLists(ListNode *l1, ListNode *l2) {
        ListNode* head = new ListNode(-1);
        ListNode* p = head;
        while (l1 != NULL || l2 != NULL) {
            int val1 = l1 == NULL ? INT_MAX : l1->val;
            int val2 = l2 == NULL ? INT_MAX : l2->val;
            if (val1 <= val2) {
                p->next = l1;
                l1 = l1->next;
            } else {
                p->next = l2;
                l2 = l2->next;
            }
            p = p->next;
        }
        p = head->next;
        delete head;
        return p;
    }
};
```

相关题目

- Merge Sorted Array §4.1
- Merge Two Sorted Lists，见 §4.2

第 5 章
分治法

5.1 Pow(x,n)

描述

Implement pow(x, n).

分析

二分法，$x^n = x^{n/2} \times x^{n/2} \times x^{n\%2}$

代码

```
//LeetCode, Pow(x, n)
// 二分法，$x^n = x^{n/2} * x^{n/2} * x^{n\%2}$
class Solution {
private:
    double power(double x, int n) {
        if (n == 0) return 1;
        double v = power(x, n / 2);
        if (n % 2 == 0) return v * v;
        else return v * v * x;
    }
public:
    double pow(double x, int n) {
        if (n < 0) return 1.0 / power(x, -n);
        else return power(x, n);
    }
};
```

相关题目

- 无

第 6 章 动态规划

6.1 Palindrome Partitioning II

描述

Given a string s, partition s such that every substring of the partition is a palindrome.

Return the minimum cuts needed for a palindrome partitioning of s.

For example, given `s = "aab"`,

Return 1 since the palindrome partitioning `["aa","b"]` could be produced using 1 cut.

分析

定义状态 `f(i,j)` 表示区间 `[i,j]` 之间最小的 cut 数，则状态转移方程为

$$f(i,j) = \min\left\{f(i,k)+f(k+1,j)\right\}, i \leq k \leq j, 0 \leq i \leq j < n$$

这是一个二维函数，实际写代码比较麻烦。

所以要转换成一维 DP。如果每次，从 i 往右扫描，每找到一个回文就算一次 DP 的话，就可以转换为 `f(i)=` 区间 `[i, n-1]` 之间最小的 `cut` 数，n 为字符串长度，则状态转移方程为

$$f(i) = \min\left\{f(j+1)+1\right\}, i \leq j < n$$

一个问题出现了，就是如何判断 `[i,j]` 是否是回文？每次都从 i 到 j 比较一遍？太浪费了，这里也是一个 DP 问题。

定义状态 `P[i][j] = true if [i,j]` 为回文，那么

```
P[i][j] = str[i] == str[j] && P[i+1][j-1]
```

代码

```
//LeetCode, Palindrome Partitioning II
class Solution {
public:
    int minCut(string s) {
        const int len = s.size();
        int f[len+1];
        bool p[len][len];
        //the worst case is cutting by each char
        for (int i = 0; i <= len; i++)
            f[i] = len - 1 - i; // 最后一个 f[len]=-1
        for (int i = 0; i < len; i++)
            for (int j = 0; j < len; j++)
                p[i][j] = false;
        for (int i = len - 1; i >= 0; i--) {
            for (int j = i; j < len; j++) {
                if (s[i] == s[j] && (j - i < 2 || p[i + 1][j - 1])) {
                    p[i][j] = true;
                    f[i] = min(f[i], f[j + 1] + 1);
                }
            }
        }
        return f[0];
    }
};
```

相关题目

- Palindrome Partitioning，见 §8.1

6.2 Maximal Rectangle

描述

Given a 2D binary matrix filled with 0's and 1's, find the largest rectangle containing all ones and return its area.

分析

无

代码

```
// LeetCode, Maximal Rectangle
class Solution {
public:
    int maximalRectangle(vector<vector<char> > &matrix) {
        if (matrix.empty())  return 0;

        const int m = matrix.size();
        const int n = matrix[0].size();
        vector<int> H(n, 0);
        vector<int> L(n, 0);
        vector<int> R(n, n);

        int ret = 0;
        for (int i = 0; i < m; ++i) {
            int left = 0, right = n;
            // calculate L(i, j) from left to right
            for (int j = 0; j < n; ++j) {
                if (matrix[i][j] == '1') {
                    ++H[j];
                    L[j] = max(L[j], left);
                } else {
                    left = j+1;
                    H[j] = 0; L[j] = 0; R[j] = n;
                }
            }
            // calculate R(i, j) from right to left
            for (int j = n-1; j >= 0; --j) {
                if (matrix[i][j] == '1') {
                    R[j] = min(R[j], right);
                    ret = max(ret, H[j]*(R[j]-L[j]));
                } else {
                    right = j;
                }
            }
        }
        return ret;
    }
};
```

相关题目

- 无

第 7 章 广度优先搜索

当题目看不出任何规律，既不能用分治，贪心，也不能用动规时，这时候万能方法——搜索，就派上用场了。搜索分为广搜和深搜，广搜里面又有普通广搜，双向广搜，A* 搜索等。深搜里面又有普通深搜，回溯法等。

广搜和深搜非常类似（除了在扩展节点这部分不一样），二者有相同的框架，如何表示状态？如何扩展状态？如何判重？尤其是判重，解决了这个问题，基本上整个问题就解决了。

7.1 Word Ladder

描述

Given two words (start and end), and a dictionary, find the length of shortest transformation sequence from start to end, such that:

- Only one letter can be changed at a time
- Each intermediate word must exist in the dictionary

For example, Given:

```
start = "hit"
end = "cog"
dict = ["hot","dot","dog","lot","log"]
```

As one shortest transformation is `"hit" -> "hot" -> "dot" -> "dog" -> "cog"`, return its length 5.

Note:

- Return 0 if there is no such transformation sequence.
- All words have the same length.
- All words contain only lowercase alphabetic characters.

分析

代码

```
//LeetCode, Word Ladder
class Solution {
```

```
public:
    int ladderLength(string start, string end, unordered_set<string> &dict) {
        if (start.size() != end.size()) return 0;
        if (start.empty() || end.empty()) return 1; return 0;
        int level = 1;  // 层次
        queue<string> queToPush, queToPop;

        queToPop.push(start);
        while (dict.size() > 0 && !queToPop.empty()) {
            while (!queToPop.empty()) {
                string str(queToPop.front());
                queToPop.pop();
                for (int i = 0; i < str.size(); i++) {
                    for (char j = 'a'; j <= 'z'; j++) {
                        if (j == str[i]) continue;

                        const char temp = str[i];
                        str[i] = j;
                        if (str == end) return level + 1; //找到了

                        if (dict.count(str) > 0) {
                            queToPush.push(str);
                            dict.erase(str); // 删除该单词，防止死循环
                        }
                        str[i] = temp; // 恢复该单词
                    }
                }
            }
            swap(queToPush, queToPop); //!!! 交换两个队列
            level++;
        }
        return 0; // 所有单词已经用光，还是找不到通向目标单词的路径
    }
};
```

相关题目

- Word Ladder II，见 §7.2

7.2 Word Ladder II

描述

Given two words (start and end), and a dictionary, find all shortest transformation sequence(s) from start to end, such that:

- Only one letter can be changed at a time
- Each intermediate word must exist in the dictionary

For example, Given:

```
start = "hit"
end = "cog"
dict = ["hot","dot","dog","lot","log"]
```

Return

```
[
    ["hit","hot","dot","dog","cog"],
    ["hit","hot","lot","log","cog"]
]
```

Note:

- All words have the same length.
- All words contain only lowercase alphabetic characters.

分析

跟 Word Ladder 比，这题是求路径本身，不是路径长度，也是 BFS，略微麻烦点

代码

```
//LeetCode, Word Ladder II
//跟 Word Ladder 比，这题是求路径本身，不是路径长度，也是 BFS，略微麻烦点
class Solution {
public:
    std::vector<std::vector<std::string> > findLadders(std::string start,
            std::string end, std::unordered_set<std::string> &dict) {
        result_.clear();
        std::unordered_map<std::string, std::vector<std::string>> prevMap;

        for (auto iter = dict.begin(); iter != dict.end(); ++iter) {
            prevMap[*iter] = std::vector<std::string>();
        }

        std::vector<std::unordered_set<std::string>> candidates(2);

        int current = 0;
        int previous = 1;

        candidates[current].insert(start);

        while (true) {
            current = !current;
            previous = !previous;

            // 从 dict 中删除 previous 中的单词，避免自己指向自己
            for (auto iter = candidates[previous].begin();
                    iter != candidates[previous].end(); ++iter) {
                dict.erase(*iter);
            }

            candidates[current].clear();
            for (auto iter = candidates[previous].begin();
                    iter != candidates[previous].end(); ++iter) {
                for (size_t pos = 0; pos < iter->size(); ++pos) {
                    std::string word = *iter;
                    for (int i = 'a'; i <= 'z'; ++i) {
                        if (word[pos] == i) continue;

                        word[pos] = i;

                        if (dict.count(word) > 0) {
                            prevMap[word].push_back(*iter);
                            candidates[current].insert(word);
                        }
                    }
                }
            }

            if (candidates[current].size() == 0) return result_;  // 此题无解
            if (candidates[current].count(end)) break; // 没看懂
        }

        std::vector<std::string> path;
        GeneratePath(prevMap, path, end);

        return result_;
    }

private:
    std::vector<std::vector<std::string>> result_;

    void GeneratePath(
            std::unordered_map<std::string, std::vector<std::string>> &prevMap,
            std::vector<std::string>& path, const std::string& word) {
        if (prevMap[word].size() == 0) {
            path.push_back(word);
            std::vector<std::string> curPath = path;
            reverse(curPath.begin(), curPath.end());
            result_.push_back(curPath);
            path.pop_back();
            return;
        }

        path.push_back(word);
        for (auto iter = prevMap[word].begin(); iter != prevMap[word].end();
                ++iter) {
            GeneratePath(prevMap, path, *iter);
        }
        path.pop_back();
    }
};
```

相关题目

- Word Ladder，见 §9.1

第 8 章
深度优先搜索

8.1 Palindrome Partitioning

描述

Given a string s, partition s such that every substring of the partition is a palindrome.

Return all possible palindrome partitioning of s.

For example, given `s = "aab"`, Return

```
[
  ["aa","b"],
  ["a","a","b"]
]
```

分析

在每一步都可以判断中间结果是否为合法结果，用回溯法。

一个长度为 n 的字符串，有 n+1 个地方可以砍断，每个地方可断可不断，因此复杂度为 $O(2^{n+1})$

代码

```
//LeetCode, Palindrome Partitioning
class Solution {
public:
    vector<vector<string>> partition(string s) {
        vector<vector<string>> result;
        vector<string> output;  // 一个 partition 方案
        DFS(s, 0, output, result);
        return result;
    }
    // 搜索必须以 s[start] 开头的 partition 方案
    // 如果一个字符串长度为 n, 则可以插入 n+1 个隔板，复制度为 O(2^{n+1})
    void DFS(string &s, int start, vector<string>& output,
            vector<vector<string>> &result) {
        if (start == s.size()) {
            result.push_back(output);
            return;
        }
        for (int i = start; i < s.size(); i++) {
            if (isPalindrome(s, start, i)) { // 从 i 位置砍一刀
                output.push_back(s.substr(start, i - start + 1));
                DFS(s, i + 1, output, result);  // 继续往下砍
                output.pop_back(); // 撤销上一个 push_back 的砍
            }
        }
    }
    bool isPalindrome(string &s, int start, int end) {
        while (start < end) {
            if (s[start] != s[end]) return false;
            start++;
            end--;
        }
        return true;
    }
};
```

相关题目

- Palindrome Partitioning II，见 §6.1

第 9 章
基本实现题

这类题目不考特定的算法，纯粹考察写代码的熟练度。

9.1 Two Sum

描述

Given an array of integers, 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`

分析

方法 1：暴力，复杂度 $O(n^2)$，会超时

方法 2：hash。用一个哈希表，存储每个数对应的下标，复杂度 $O(n)$

代码

```
//LeetCode, Two Sum
// 方法 1：暴力，O(n^2)
// 方法 2：hash。用一个哈希表，存储每个数对应的下标，复杂度 O(n)，代码如下，
class Solution {
public:
    vector<int> twoSum(vector<int> &numbers, int target) {
        unordered_map<int, int> mapping;
        vector<int> result;
        for (int i = 0; i < numbers.size(); i++) {
            mapping[numbers[i]] = i;
        }
        for (int i = 0; i < numbers.size(); i++) {
            const int gap = target - numbers[i];
            if (mapping.find(gap) != mapping.end()) {
                result.push_back(i + 1);
                result.push_back(mapping[gap] + 1);
                break;
            }
        }
        return result;
    }
};
```

相关题目

- 无

9.2 Insert Interval

描述

Given a set of non-overlapping intervals, insert a new interval into the intervals (merge if necessary).

You may assume that the intervals were initially sorted according to their start times.

Example 1: Given intervals `[1,3],[6,9]`, insert and merge `[2,5]` in as `[1,5],[6,9]`.

Example 2: Given `[1,2],[3,5],[6,7],[8,10],[12,16]`, insert and merge `[4,9]` in as `[1,2],[3,10],[12,16]`.

This is because the new interval `[4,9]` overlaps with `[3,5],[6,7],[8,10]`.

分析

无

代码

```
struct Interval {
    int start;
    int end;
    Interval() : start(0), end(0) { }
    Interval(int s, int e) : start(s), end(e) { }
};

//LeetCode, Insert Interval
class Solution {
public:
    vector<Interval> insert(vector<Interval> &intervals, Interval newInterval) {
        vector<Interval>::iterator it = intervals.begin();
        while (it != intervals.end()) {
            if (newInterval.end < it->start) {
                intervals.insert(it, newInterval);
                return intervals;
            } else if (newInterval.start > it->end) {
                it++;
                continue;
            } else {
                newInterval.start = min(newInterval.start, it->start);
                newInterval.end = max(newInterval.end, it->end);
                it = intervals.erase(it);
            }
        }
        intervals.insert(intervals.end(), newInterval);
        return intervals;
    }
};
```

相关题目

- Merge Intervals，见 §9.3

9.3 Merge Intervals

描述

Given a collection of intervals, merge all overlapping intervals.

For example, Given `[1,3]`,`[2,6]`,`[8,10]`,`[15,18]`, return `[1,6]`,`[8,10]`,`[15,18]`

分析

复用一下 Insert Intervals 的解法即可，创建一个新的 interval 集合，然后每次从旧的里面取一个 interval 出来，然后插入到新的集合中。

代码

```
struct Interval {
    int start;
    int end;
    Interval() : start(0), end(0) { }
    Interval(int s, int e) : start(s), end(e) { }
};

//LeetCode, Merge Interval
//复用一下 Insert Intervals 的解法即可
class Solution {
public:
    vector<Interval> merge(vector<Interval> &intervals) {
        vector<Interval> result;
        for (int i = 0; i < intervals.size(); i++) {
            insert(result, intervals[i]);
        }
        return result;
    }
private:
    vector<Interval> insert(vector<Interval> &intervals, Interval newInterval) {
        vector<Interval>::iterator it = intervals.begin();
        while (it != intervals.end()) {
            if (newInterval.end < it->start) {
                intervals.insert(it, newInterval);
                return intervals;
            } else if (newInterval.start > it->end) {
                it++;
                continue;
            } else {
                newInterval.start = min(newInterval.start, it->start);
                newInterval.end = max(newInterval.end, it->end);
                it = intervals.erase(it);
            }
        }
        intervals.insert(intervals.end(), newInterval);
        return intervals;
    }
};
```

相关题目

- Insert Interval，见 §9.2