LeetCode Summary

tanglei

Published with GitBook

Table of Contents

ii. Add Two Numbers 题解
iii. Divide Two Integers 题解
iv. Gray Code 题解
v. Integer to Roman 题解
vi. Multiply Strings 题解
vii. Palindrome Number 题解
viii. Plus One 题解
ix. Pow(x, n) 题解
x. Reverse Integer 题解
xi. Roman to Integer 题解
xii. Sqrt(x) 题解
xiii. String to Integer (atoi) 题解
xiv. Valid Number 题解

8. string, 字符串处理相关
   i. Anagrams 题解
   ii. Count and Say 题解
   iii. Evaluate Reverse Polish Notation 题解
   iv. Implement strStr() 题解
   v. Length of Last Word 题解
   vi. Longest Common Prefix 题解
   vii. Longest Palindromic Substring 题解
   viii. Longest Substring Without Repeating Characters 题解
   ix. Longest Valid Parentheses 题解
   x. Minimum Window Substring 题解
   xi. Regular Expression Matching 题解
   xii. Reverse Words in a String 题解
   xiii. Simplify Path 题解
   xiv. Text Justification 题解
   xv. Valid Parentheses 题解
   xvi. Wildcard Matching 题解
   xvii. ZigZag Conversion 题解
9. combination and permutation, 排列组合相关
   i. Combinations 题解
   ii. Combination Sum 题解
   iii. Combination Sum II 题解
   iv. Letter Combinations of a Phone Number 题解
   v. Next Permutation 题解
   vi. Palindrome Partitioning 题解
   vii. Permutation Sequence 题解
   viii. Permutations 题解
   ix. Permutations II 题解
   x. Subsets 题解
   xi. Subsets II 题解
   xii. Unique Paths 题解
10. matrix, 二维数组, 矩阵相关
   i. Rotate Image 题解
   ii. Set Matrix Zeroes 题解
   iii. Spiral Matrix 题解

Leetcode Summary

这是本人之前陆陆续续刷leetcode留下的，这个Repository是截止到2014年10月为止Leetcode上的所有154道题目的解题思路和AC代码。

最开始的版本是一个网页，在这里，后来才用gitbook整理改写到这里。

以下题目的分类都是根据自己的理解分的，后来才发现leetcode官网也对题目进行了分类且题目有不断更新，以后有机会再慢慢加入和更新。现在整理下希望对其他人有所作用。对相关题目有所讨论和指正的欢迎采取以下方式进行联系：

1. 在github上发 pull request Leetcode summary 地址。//最直接的方式，推荐。
2. 直接在本gitbook每篇文章里留言。

另外，欢迎关注并订阅我的blog：tanglei's blog。

目 录

DP, 动态规划类

list, 链表相关

6. Linked List Cycle 题解
7. Linked List Cycle II 题解
8. Merge Two Sorted Lists 题解
9. Merge k Sorted Lists 题解
10. Partition List 题解
11. Remove Duplicates from Sorted List 题解
12. Remove Duplicates from Sorted List II 题解
13. Remove Nth Node From End of List 题解
14. Reorder List 题解
15. Reverse Linked List II 题解
16. Reverse Nodes in k-Group 题解
17. Rotate List 题解
18. Sort List 题解
19. Swap Nodes in Pairs 题解

binary tree, 二叉树相关

1. Balanced Binary Tree 题解
2. Binary Tree Inorder Traversal 题解
3. Binary Tree Level Order Traversal 题解
4. Binary Tree Level Order Traversal II 题解
5. Binary Tree Maximum Path Sum 题解
6. Binary Tree Postorder Traversal 题解
7. Binary Tree Preorder Traversal 题解
8. Binary Tree Zigzag Level Order Traversal 题解
9. Construct Binary Tree from Inorder and Postorder Traversal 题解
10. Construct Binary Tree from Preorder and Inorder Traversal 题解
11. Convert Sorted List to Binary Search Tree 题解
12. Flatten Binary Tree to Linked List 题解
13. Maximum Depth of Binary Tree 题解
14. Minimum Depth of Binary Tree 题解
15. Path Sum 题解
16. Path Sum II 题解
17. Populating Next Right Pointers in Each Node 题解
18. Populating Next Right Pointers in Each Node II 题解
19. Recover Binary Search Tree 题解
20. Same Tree 题解
21. Sum Root to Leaf Numbers 题解
22. Symmetric Tree 题解
23. Unique Binary Search Trees 题解
24. Unique Binary Search Trees II 题解
25. Validate Binary Search Tree 题解

sort, 排序相关

1. 3Sum Closest 题解
2. 3Sum 题解
3. 4Sum 题解

4. Insert Interval 题解
5. Longest Consecutive Sequence 题解
6. Merge Intervals 题解
7. Merge Sorted Array 题解
8. Remove Duplicates from Sorted Array 题解
9. Remove Duplicates from Sorted Array II 题解
10. Sort Colors 题解
11. Two Sum 题解

search, 搜索相关

1. First Missing Positive 题解
2. Find Minimum in Rotated Sorted Array 题解
3. Find Minimum in Rotated Sorted Array II 题解
4. Median of Two Sorted Arrays 题解
5. Search Insert Position 题解
6. Search a 2D Matrix 题解
7. Search for a Range 题解
8. Search in Rotated Sorted Array 题解
9. Search in Rotated Sorted Array II 题解
10. Single Number 题解
11. Single Number II 题解

math, 数学类相关

1. Add Binary 题解
2. Add Two Numbers 题解
3. Divide Two Integers 题解
4. Gray Code 题解
5. Integer to Roman 题解
6. Multiply Strings 题解
7. Palindrome Number 题解
8. Plus One 题解
9. Pow(x, n) 题解.md)
10. Reverse Integer 题解
11. Roman to Integer 题解
12. Sqrt(x) 题解.md)
13. String to Integer (atoi) 题解.md)
14. Valid Number 题解

string, 字符串处理相关

1. Anagrams 题解
2. Count and Say 题解
3. Evaluate Reverse Polish Notation 题解
4. Implement strStr() 题解.md)
5. Length of Last Word 题解
6. Longest Common Prefix 题解

7. Longest Palindromic Substring 题解
8. Longest Substring Without Repeating Characters 题解
9. Longest Valid Parentheses 题解
10. Minimum Window Substring 题解
11. Regular Expression Matching 题解
12. Reverse Words in a String 题解
13. Simplify Path 题解
14. Text Justification 题解
15. Valid Parentheses 题解
16. Wildcard Matching 题解
17. ZigZag Conversion 题解

combination and permutation, 排列组合相关

1. Combinations 题解
2. Combination Sum 题解
3. Combination Sum II 题解
4. Letter Combinations of a Phone Number 题解
5. Next Permutation 题解
6. Palindrome Partitioning 题解
7. Permutation Sequence 题解
8. Permutations 题解
9. Permutations II 题解
10. Subsets 题解
11. Subsets II 题解
12. Unique Paths 题解

matrix, 二维数组, 矩阵相关

1. Rotate Image 题解
2. Set Matrix Zeroes 题解
3. Spiral Matrix 题解
4. Spiral Matrix II 题解
5. Maximal Rectangle 题解

回溯, BFS/DFS

1. Clone Graph 题解
2. Generate Parentheses 题解
3. N-Queens 题解
4. N-Queens II 题解
5. Restore IP Addresses 题解
6. Sudoku Solver 题解
7. Surrounded Regions 题解
8. Word Ladder 题解
9. Word Ladder II 题解
10. Word Search 题解

greedy, 贪心

1. Best Time to Buy and Sell Stock II 题解
2. Jump Game 题解
3. Jump Game II 题解

其他

1. Candy 题解
2. Container With Most Water 题解
3. Gas Station 题解
4. Gray Code 题解
5. Max Points on a Line 题解
6. Pascal's Triangle 题解
7. Pascal's Triangle II 题解
8. Remove Element 题解
9. Trapping Rain Water 题解

DP, 动态规划类

1. Best Time to Buy and Sell Stock III 题解
2. Best Time to Buy and Sell Stock 题解
3. Climbing Stairs 题解
4. Decode Ways 题解
5. Distinct Subsequences 题解
6. Edit Distance 题解
7. Interleaving String 题解
8. Longest Palindromic Substring 题解
9. Maximum Product Subarray 题解
10. Maximum Subarray 题解
11. Minimum Path Sum 题解
12. Palindrome Partitioning 题解
13. Palindrome Partitioning II 题解
14. Scramble String 题解
15. Triangle 题解
16. Unique Binary Search Trees 题解
17. Unique Paths II 题解
18. Word Break 题解
19. Word Break II 题解

Climbing Stairs

题目来源：Climbing Stairs

> You are climbing a stair case. It takes n steps to reach to the top. Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

解题思路：

设f(x)=【剩x阶时，迈楼梯的方法总数】。首先迈出第一步，如果一次迈一阶，剩下x-1阶，方法总数为f(x-1)；如果一次迈两阶，剩下x-2阶，方法总数为f(x-2); f(x)=f(x-1)+f(x-2)。容易发现，f(1)=1，f(2)=2。

```
//TLE
int climbStairsTLE(int n)
{
    if(n <= 1) return n;
    if(n == 2) return 2;
    return climbStairs(n-1) + climbStairs(n-2);
}
```

```
int climbStairs(int n)
{
    if(n <= 1) return n;
    if(n == 2) return 2;
    int a = 1, b = 2, c = 2;
    for(int i = 2; i < n; i++)
    {
        c = a + b;
        a = b;
        b = c;
    }
    return c;
}
```

Decode Ways

题目来源：Decode Ways

> A message containing letters from A-Z is being encoded to numbers using the following mapping: 'A' -> 1 'B' -> 2 ... 'Z' -> 26 Given an encoded message containing digits, determine the total number of ways to decode it. For example, Given encoded message "12", it could be decoded as "AB" (1 2) or "L" (12). The number of ways decoding "12" is 2.

解题思路：

动态规划. dp[i] 表示 s[0:i-1]的结果.

1. 若当前字符s[i-1]为0
   - 若前面的字符是1或2, 则这1或2必须得跟0结合, dp[i] = dp[i-2] eg: XXX(10)
   - 若前面的字符不是1或者2, 拼接不起来了， 直接return 0.
2. 当前字符不为0
   - 前面的字符为1 或者 2且当前字符为1-6. 即可分两种情况拆分, dp[i] = dp[i-2] + dp[i-1], eg: XXX16: (XXX)(16) + (XXX1)6
   - 其他, 只能自己跟自己一组了. dp[i] = dp[i-1] eg: (XXX3)6

还可以将下面的O(n)空间化简成常数空间～记录之前上一次结果 和 上上一次的结果，当前的结果由此获得。

```
int numDecodings(string s)
{
    if( s.empty() || s[0] == '0') return 0;
    vector<int> dp(s.length()+1, 0);
    dp[0] = 1; dp[1] = 1;
    for(int i = 2; i <= s.length(); i++)
    {
        if(s[i-1] == '0')
        {
            if(s[i-2] == '1' || s[i-2] == '2')
                dp[i] = dp[i-2];
            else
                return 0;
        }else if(s[i-2] == '1' || (s[i-2]=='2' && s[i-1] <= '6'))
            dp[i] = dp[i-2] + dp[i-1];
        else
            dp[i] = dp[i-1];
    }
    return dp[s.length()];
}
```

Distinct Subsequences

题目来源：Distinct Subsequences

>

```
Given a string S and a string T, count the number of distinct subsequences of T
in S.
A subsequence of a string is a new string which is formed from the original
string by deleting some (can be none) of the characters without disturbing the
relative positions of the remaining characters. (ie, "ACE" is a subsequence of
"ABCDE" while "AEC" is not).

Here is an example:
S = "rabbbit", T = "rabbit"
Return 3.
```

解题思路：

动态规划，`dp[j][i]` 表示 `S[:i],T[:j]` 的结果, dp[j][i]至少为dp[j][i-1] 那么 若 `S[i] == T[j]`，则dp[j][i]还得加上dp[j-1][i-1]即S[:i-1],T[:j-1]的匹配结果。

```
int numDistinct(string S, string T)
{
    int n = S.length();
    int m = T.length();
    if(m >= n) return T == S ? 1 : 0;
    vector<vector<int> > dp(m, vector<int>(n, 0));
    //dp[j][i] S[:i] T[:j] matches
    for(int i = 0; i < n; i++)
        for(int j = 0; j < m; j++)
        {
            if(i >= 1)
                dp[j][i] = dp[j][i-1];
            if(S[i] == T[j])
            {
                if(j >= 1 && i >= 1)
                    dp[j][i] += dp[j-1][i-1];
                if(j == 0)
                    dp[j][i] += 1;
            }
        }
    return dp[m-1][n-1];
}
```

若以dp[j][i]中的i/j以长度来看的话，代码要简洁些。 初始化dp[0][0:i]=1 表示T中长度为0的串可以和S中任意长度匹配。

```
int numDistinct(string S, string T)
{
    int m = T.length();
    int n = S.length();
```

```
        if(m >= n) return T == S;
        vector<vector<int> > dp(m+1, vector<int>(n+1, 0));
        for(int i = 0; i <=n; i++)
            dp[0][i] = 1;
        for(int i = 1; i <= n; i++)
            for(int j = 1; j <= m; j++)
                dp[j][i] = dp[j][i-1] + (S[i-1] == T[j-1] ? dp[j-1][i-1] : 0);
        return dp[m][n];
    }
```

节省内存

```
int numDistinct(string S, string T)
{
    int m = T.length();
    int n = S.length();
    if (m > n) return 0;    // impossible for subsequence

    vector<int> path(m+1, 0);
    path[0] = 1;            // initial condition
    for (int j = 1; j <= n; j++) {
        // traversing backwards so we are using path[i-1] from last time step
        for (int i = m; i >= 1; i--) {
            path[i] = path[i] + (T[i-1] == S[j-1] ? path[i-1] : 0);
        }
    }
    return path[m];
}
```

这题参考了REF，其实跟Interleaving String 这道题差不多。

Edit Distance

题目来源：Edit Distance

> Given two words word1 and word2, find the minimum number of steps required to convert word1 to word2. (each operation is counted as 1 step.) You have the following 3 operations permitted on a word: a) Insert a character b) Delete a character c) Replace a character

解题思路：

编辑距离是动态规划里面的经典题目。 DP, DP[i][j] 表示word1[0:i-1] 与 word2[0:j-1]的编辑距离。Ref

$$d_{ij} = \begin{cases} d_{i-1,j-1} & a_j = b_i \\ \min \begin{cases} d_{i-1,j} + w_{\text{del}}(b_i) \\ d_{i,j-1} + w_{\text{ins}}(a_j) \\ d_{i-1,j-1} + w_{\text{sub}}(a_j, b_i) \end{cases} & a_j \neq b_i \end{cases}, \quad for\ 1 \leq i \leq m, 1 \leq j \leq n.$$

w(del), w(ins), w(sub) 分别是删除，插入，替换(substitute)的权重。

```
int minDistance(string word1, string word2)
{
    int m = word1.length();
    int n = word2.length();
    vector<vector<int> >dp(m+1, vector<int>(n+1, 0));
    //dp[i][j]: word1[0:i-1] -> word2[0:j-1]
    for(int i = 0; i <= m; i++)
        dp[i][0] = i;//insert
    for(int i = 0; i <= n; i++)
        dp[0][i] = i;
    for(int i = 1; i <= m; i++)
        for(int j = 1; j <= n; j++)
        {
            if(word1[i-1] == word2[j-1])
                dp[i][j] = dp[i-1][j-1];
            else{
                int insert = dp[i-1][j] + 1; //insert word2[j] to word1[i]
                int _delete = dp[i][j-1] + 1; //delete word1[i]
                int replace = dp[i-1][j-1] + 1; //replace word1[i] to word2[j]
                dp[i][j] = std::min(replace, std::min(insert, _delete));
            }
        }
    return dp[m][n];

}
```

关于更多编辑距离的算法及应用可参考 stanford 课件。

Interleaving String

题目来源：Interleaving String

> Given s1, s2, s3, find whether s3 is formed by the interleaving of s1 and s2. For example, Given: s1 = "aabcc", s2 = "dbbca", When s3 = "aadbbcbcac", return true. When s3 = "aadbbbaccc", return false.

解题思路：

递归

超时。

```
bool isInterleave(string s1, string s2, string s3)
{
    return isInterleave(s1, s1.length()-1, s2, s2.length()-1, s3, s3.length()-1);
}
bool isInterleave(string s1, int i1, string s2, int i2, string s, int i)
{
    if(i < 0 || i1 < 0 || i2 < 0) return i < 0 && i1 < 0 && i2 < 0;
    if((s1[i1] == s[i]) && isInterleave(s1, i1-1, s2, i2, s, i-1))
        return true;
    if((s2[i2] == s[i]) && isInterleave(s1, i1, s2, i2-1, s, i-1))
        return true;
    return false;
}
```

动态规划

DP。

用DP， 类似Distinct Subsequences 一样， dp[i][j]表示长度为i的s1[0:i-1],长度为j的s2[0:j-1]和s3[0:i+j-1]的匹配结果，那么

```
dp[i][j] = dp[i][j-1] && (s2[j-1]==s3[i+j-1])
       || dp[i-1][j] && (s1[i-1]==s3[i+j-1])
       or false.
```

注意边界的初始化条件.

```
bool isInterleave(string s1, string s2, string s3)
{
    int n1 = s1.length(); int n2 = s2.length(); int n3 = s3.length();
    if(n1 + n2 != n3) return false;
    if(n1 == 0) return s2 == s3;
    if(n2 == 0) return s1 == s3;
    vector<vector<bool> > dp(n1+1, vector<bool>(n2+1, false));
    dp[0][0] = true;
    for(int i = 1; i <= n1; i++)
        dp[i][0] = (s1[i-1] == s3[i-1]);
    for(int i = 1; i <= n2; i++)
        dp[0][i] = (s2[i-1] == s3[i-1]);
```

```
    for(int i = 1; i <= n1; i++)
        for(int j = 1; j <= n2; j++)
        {
            if(s1[i-1] == s3[i+j-1] && dp[i-1][j])
                dp[i][j] = true;
            else if(s2[j-1] == s3[i+j-1] && dp[i][j-1])
                dp[i][j] = true;
            else
                dp[i][j] = false;
        }
    return dp[n1][n2];
}
```

Longest Palindromic Substring

题目来源：Longest Palindromic Substring

> Given a string S, find the longest palindromic substring in S. You may assume that the maximum length of S is 1000, and there exists one unique longest palindromic substring.

解题思路：

暴力搜索, O(N^2)

最简单的方法就是选中i(0~n-1)，然后向两边扩展，复杂度为O(N^2) . 注意回文长度可能是奇数或者偶数，即 `aba or abba`

```
string longestPalindrome(string s)
{
  int n = s.length();
  if (n <= 1) return s;
  int start = 0; int maxLen = 1;
  for(int i = 0; i < n; i++)
  {
      //center: i
      int left = i - 1;
      int right = i + 1;
      while(left >= 0 && right < n
          && s[left] == s[right])
          --left, ++right;
      //s[left] != s[right] , s[left+1 : right-1] is palindrome
      int len = (right-1) - (left+1) + 1;
      if(len > maxLen)
          start = left+1, maxLen = len;
      //center: between s[i] and s[i+1]
      if(i+1 < n && s[i] == s[i+1])
      {
          left = i;
          right = i+1;
          while(left >= 0 && right < n
              && s[left] == s[right])
              --left, ++right;
          len = (right-1) - (left+1) + 1;
          if(len > maxLen)
              start = left+1, maxLen = len;
      }
  }
  return s.substr(start, maxLen);
}
```

或者可以这样，将中间相同的字符跳过，不考虑奇数还是偶数的串(其实还是考虑了，也包括在内了)。

```
abbbbbbbbba, left=1时,right一直走到最后一个b, 然后往两边判断是否相等.
```

```
string longestPalindrome(string s)
{
```

```
    int n = s.length();
    if (n <= 1) return s;
    int start = 0; int maxLen = 1;
    for(int i = 0; i < n; i++)
    {
        int left = i;
        int right = i;
        while(right+1 < n && s[right+1] == s[left])
            ++right;
        //s[right+1] != s[left], s[right]=s[left]
        i = right; //skip the same char
        //a[bbbbbbbb]a
        while(left-1 >= 0 && right+1 < n
            && s[left-1] == s[right+1])
            --left, ++right;
        //s[left : right] is palindrome
        int len = right - left + 1;
        if(len > maxLen)
            start = left, maxLen = len;
    }
    return s.substr(start, maxLen);
}
```

DP, O(N^2)

dp[i][j] 表示 s[i:j] 是回文, 当且尽当 `s[i] == [j] && dp[i+1][j-1]` , 即计算dp[i][j]时, dp[i+1][j-1]得先计算出来， 算dp[x][i]， 必须先把dp[x][i-1]先计算出来了来。

```
string longestPalindrome(string s)
{
    int n = s.length();
    if (n <= 1) return s;
    int start = 0; int maxLen = 1;
    //vector<vector<bool> > dp(n, vector<bool>(n, false));
    bool dp[1000][1000] = {false};
    for(int i = 0; i < n; i++)
        dp[i][i] = true;
    //dp[j][i]: s[j:i] is palindrome, dp[j+1][i-1] true + s[j]==s[i]
    for(int i = 1; i < n; i++)
        for(int j = 0; j < i; j++)
        {
            if(s[i] == s[j] && (j+1 > i-1 || dp[j+1][i-1]))
            {
                dp[j][i] = true;
                int len = i - j + 1;
                if(len > maxLen)
                    maxLen = len, start = j;
            }
        }
    return s.substr(start, maxLen);
}
```

计算s[:5]的结果，先得把所有s[:4]结尾的回文算出来了来。 上面比如在算dp[i][5]时，用到了dp[x][4]，在上面的循环中,dp[x][4]已经算出来了的。 另外， 虽然都是平方的算法，上面用vector还过不了， 用数组才能过。

O(n) 算法, Manacher 算法

felix021的文章讲得很清楚，这里“偷”过来。

>

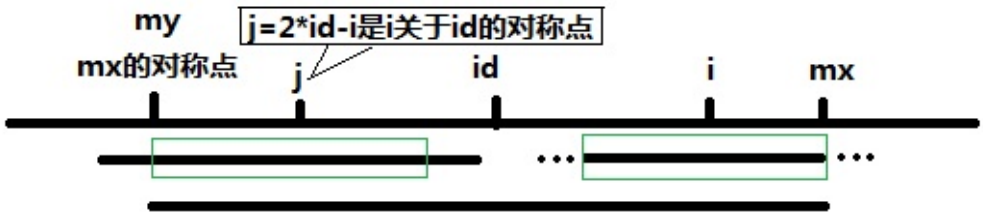
首先用一个非常巧妙的方式，将所有可能的奇数/偶数长度的回文子串都转换成了奇数长度：在每个字符的两边都插入一个

下面以字符串12212321为例，经过上一步，变成了 S[] = "$#1#2#2#1#2#3#2#1#";

然后用一个数组 P[i] 来记录以字符S[i]为中心的最长回文子串向左/右扩张的长度（包括S[i]，也就是把该回文串“对
```
S  #  1  #  2  #  2  #  1  #  2  #  3  #  2  #  1  #
P  1  2  1  2  5  2  1  4  1  2  1  6  1  2  1  2  1
```
(p.s. 可以看出，P[i]-1正好是原字符串中回文串的总长度)
那么怎么计算P[i]呢？该算法增加两个辅助变量（其实一个就够了，两个更清晰）id和mx，其中id表示最大回文子串中心

然后可以得到一个非常神奇的结论，这个算法的关键点就在这里了：如果mx > i，那么P[i] >= MIN(P[2 * id - i]
```
//记j = 2 * id - i，也就是说 j 是 i 关于 id 的对称点。
if (mx - i > P[j])
    P[i] = P[j];
else /* P[j] >= mx - i */
    P[i] = mx - i; // P[i] >= mx - i，取最小值，之后再匹配更新。
```
当然光看代码还是不够清晰，还是借助图来理解比较容易。
当 mx - i > P[j] 的时候，以S[j]为中心的回文子串包含在以S[id]为中心的回文子串中，由于 i 和 j 对称，以S

当 P[j] >= mx - i 的时候，以S[j]为中心的回文子串不一定完全包含于以S[id]为中心的回文子串中，但是基于对称



对于 mx <= i 的情况，无法对 P[i]做更多的假设，只能P[i] = 1，然后再去匹配了。

代码如下：

```
//Manacher O(n)
//ref1: http://leetcode.com/2011/11/longest-palindromic-substring-part-ii.html
//ref2: http://blog.csdn.net/pickless/article/details/9040293
//ref3: http://www.felix021.com/blog/read.php?2040
string longestPalindrome(string s)
{
   int len = (int)s.length() * 2 + 2;
   string news(len+1, ' ');
```

```
    news[0]='$';
    for(int i = 0; i < s.length(); i++)
    {
        news[2 * i + 1] = '#';
        news[2 * i + 2] = s[i];
    }
    news[len-1] = '#';
    news[len] = '\0';
    vector<int> p(len, 0);
    int mx = 0, id = 0;
    for(int i = 1; i < len; i++)
    {
        p[i] = mx > i ? min(p[2*id-i], mx-i) : 1;
        while(news[i + p[i]] == news[i - p[i]]) ++p[i];
        if(i + p[i] > mx)
        {
            mx = i + p[i];
            id = i;
        }
    }
    int maxLen = 0;
    int center = 0;
    for(int i = 0; i < len; i++)
    {
        if(p[i] > maxLen)
        {
            maxLen = p[i];
            center = i;
        }
    }
    return s.substr((center)/2 - maxLen/2, maxLen-1);
}
```

Maximum Product Subarray

题目来源：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.

解题思路：

暴力O(n^2)

超时

```
int maxProductN2(int A[], int n)
{
    assert(A != NULL && n != 0);
    if (n == 1) return A[0];
    int result = A[0];
    for(int i = 0; i < n; i++)
    {
        int t = 1;
        for(int j = i; j >= 0; j--)
        {
            t *= A[j];
            result = std::max(result, t);
        }
    }
    return result;
}
```

DP O(n)

记录到i为止的最大值和最小值，最小值乘以当前值可能反而变成最大值，不用去考虑当前A[i]的值的正负，分情况讨论，这样反而复杂。

```
int maxProduct(int A[], int n)
{
    assert(A != NULL && n != 0);
    if (n == 1) return A[0];
    int result = A[0];
    vector<int> dpMin(n, 0);
    vector<int> dpMax(n, 0);
    dpMax[0] = dpMin[0] = A[0];
    for(int i = 1; i < n; i++)
    {
        dpMax[i] = std::max(std::max(dpMax[i-1]*A[i], dpMin[i-1]*A[i]), A[i]);
        result = std::max(result, dpMax[i]);
        dpMin[i] = std::min(std::min(dpMin[i-1]*A[i], dpMax[i-1]*A[i]), A[i]);
    }
    return result;
}
```

可以用O(1)的空间，记录上一次的结果。代码就直接贴 discuss 里的了。

```
//[ref](https://oj.leetcode.com/discuss/11923/sharing-my-solution-o-1-space-o-n-runni
int maxProduct(int A[], int n)
{
    assert(A != NULL && n != 0);
    int maxherepre = A[0];
    int minherepre = A[0];
    int maxsofar = A[0];
    int maxhere = 0; int minhere = 0;

    for (int i = 1; i < n; i++) {
        maxhere = std::max(std::max(maxherepre * A[i], minherepre * A[i]), A[i]);
        minhere = std::min(std::min(maxherepre * A[i], minherepre * A[i]), A[i]);
        maxsofar = std::max(maxhere, maxsofar);
        maxherepre = maxhere;
        minherepre = minhere;
    }
    return maxsofar;
}
```

Maximum Subarray

题目来源：Maximum Subarray

> Find the contiguous subarray within an array (containing at least one number) which has the largest sum. For example, given the array [−2,1,−3,4,−1,2,1,−5,4], the contiguous subarray [4,−1,2,1] has the largest sum = 6. More practice: If you have figured out the O(n) solution, try coding another solution using the divide and conquer approach, which is more subtle.

解题思路：

注意此例是连续subarray,且最少得选1个。

若当前i，前面i-1的结果若为负的话，新序列就从当前A[i]开始算起了，不然就将当前A[i]附加上去。

DP, O(n) 空间

```
int maxSubArray(int A[], int n)
{
    assert(n != 0);
    vector<int> dp(n, 0);
    dp[0] = A[0];
    int result = A[0];
    for(int i = 1; i < n; i++)
    {
        if(dp[i-1] < 0)
            dp[i] = A[i];
        else
            dp[i] = dp[i-1] + A[i];
        result = std::max(result, dp[i]);
    }
    return result;
}
```

DP, O(1) 空间

上面的优化一下即可。

```
//Kadane's algorithm O(n)
//max_end_here是结束位置为i-1的最大子数组和
int maxSubArray(int A[], int n)
{
    int max_so_far = A[0];
    int max_end_here = A[0];
    for(int i = 1; i < n; i++)
    {
        if(max_end_here < 0)
            max_end_here = A[i];
        else
            max_end_here += A[i];
        max_so_far = std::max(max_so_far, max_end_here);
```

```
        }
        return max_so_far ;
    }
```

分治, O(nlogn)

分治算法：要么左半/右半，要么包括中间的和左右两边都有部分, 时间复杂度 `O(NlogN)` .

```
    //Divide and Conquer O(nlogn)
    int maxSubArrayDAC(int A[], int left, int right)
    {
        if(right < left) return INT_MIN;
        if(right == left)
            return A [left]; //at least 1 element
        int mid = left + ((right - left)>>1);
        //across left and right
        int crossSumLeft = A[mid]; //including mid
        int crossMaxLeft = A[mid];
        for(int i = mid-1; i >= left; i--)
        {
            crossSumLeft += A[i];
            crossMaxLeft = std::max(crossMaxLeft, crossSumLeft);
        }
        int crossSumRight = A[mid];
        int crossMaxRight = A[mid];
        for(int i = mid+1; i <= right; i++)
        {
            crossSumRight += A[i];
            crossMaxRight = std::max(crossMaxRight, crossSumRight);
        }
        int crossMax = crossMaxLeft + crossMaxRight - A[mid];
        int leftMax = maxSubArrayDAC( A, left , mid-1);
        int rightMax = maxSubArrayDAC( A, mid+1, right );
        return std::max(std::max(leftMax, rightMax), crossMax);
    }
    int maxSubArray(int A[], int n)
    {
        assert(n != 0);
        if(n == 1) return A[0];
        return maxSubArrayDAC(A, 0, n-1) ;
    }
```

参考资料:

- ref

Minimum Path Sum

题目来源：Minimum Path Sum

> Given a m x n grid filled with non-negative numbers, find a path from top left to bottom right which minimizes the sum of all numbers along its path. Note: You can only move either down or right at any point in time.

解题思路：

递归

用递归思路比较清晰，然后转成迭代。

```
int min(vector<vector<int> >&grid, int row, int col)
{
    if(row == 0)
    {
        int result = 0;
        for(int i = 0; i <= col; i++)
            result += grid[0][i];
        return result;
    }
    if(col == 0)
    {
        int result = 0;
        for(int i = 0; i <= row; i++)
            result += grid[i][0];
        return result;
    }
    int fromleft = min(grid, row, col-1);
    int fromup = min(grid, row-1, col);
    return std::min(fromleft, fromup);
}
```

动态规划, O(m+n)空间

```
int minPathSum(vector<vector<int> > &grid)
{
    int m = grid.size(); if(m == 0) return INT_MIN;
    int n = grid[0].size();
    vector<vector<int> > dp(m, vector<int>(n, 0));
    dp[0][0] = grid[0][0];
    for(int i = 1; i < n; i++)
        dp[0][i] = dp[0][i-1] + grid[0][i];
    for(int i = 1; i < m; i++)
        dp[i][0] = dp[i-1][0] + grid[i][0];
    for(int i = 1; i < m; i++)
        for(int j = 1; j < n; j++)
            dp[i][j] = std::min(dp[i-1][j], dp[i][j-1]) + grid[i][j];
    return dp[m-1][n-1];
}
```

动态规划, O(n)空间

更加节约点空间可以这样. 参考了leetcode-cpp

```
f[j] = min(f[j-1], f[j])+grid[i][j];
右边的f[j]是老的f[j]表示v[i-1][j], f[j-1]即为v[i][j-1]
左边f[j]是新的,即v[i][j].
```

```
    int minPathSum(vector<vector<int > > &grid)
    {
        int m = grid.size();  if (m == 0) return INT_MIN;
        int n = grid[0].size();
        vector< int> f(n, INT_MAX );
        f[0] = 0;
        for (int i = 0; i < m; i++)
        {
             f[0] += grid[i][0];
              for (int j = 1; j < n; j++)
                f[j] = std::min(f[j - 1], f[j]) + grid[i][j];
        }
        return f[n - 1];
    }
```

Palindrome Partitioning

题目来源：Palindrome Partitioning

>

```
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"]
]
```

解题思路：

直接暴力解决

枚举每种可能，去判读是否回文。跟排列组合算法一样。 还可以优化，把中间的某个子串是否回文用hash缓存下来。

```
    bool isPalindrome(string s)
    {
        int n = s.length();
        if(n <= 1) return true;
        int left = 0; int right = n-1;
        while(left < right)
        {
            if(s[left] == s[right])
            {
                left++;
                right--;
            }else
                return false;
        }
        return true;
    }
    void search(vector<string> &path, int start, string s, vector<vector<string> > &resul
    {
        int n = s.length();
        if(start > n) return;
        if(start == n)
        {
            result.push_back(path);
            return;
        }
        for(int i = start; i < n; i++)
        {
            string str = s.substr(start, i-start+1);
```

```
            if(isPalindrome(str))
            {
                path.push_back(str);
                search(path, i+1, s, result);
                path.pop_back();
            }
        }
    }
    vector<vector<string>> partition(string s)
    {
        vector<vector<string> > result;
        vector<string> path;
        search(path, 0, s, result);
        return move(result);
    }
```

利用动态规划 O(n^2)

`dp[i:j]` 表示 `s[i:j]` 是回文, 如果 `s[i] == s[j] and dp[i+1, j-1]` ,满足条件, 则dp[i:j]就是回文。 注意要先算dp[i+1][j-1]，所以循环的顺序。

```
    void search(vector<string> &path, int start, string s, vector<vector<bool> > &dp, vec
    {
        if(start == s.length()){
            result.push_back(path);
            return;
        }
        for(int i = start; i < s.length(); i++)
        {
            if(dp[start][i]){
                string sub = s.substr(start, i-start+1);
                path.push_back(sub);
                search(path, i+1, s, dp, result);
                path.pop_back();
            }
        }
    }
    vector<vector<string>> partitionDp(string s)
    {
        int n = s.length();
        vector<vector<bool> > dp(n, vector<bool>(n, false));
        for(int i = 0; i < n; i++) //init, single char is palindrome
            dp[i][i] = true;
        //dp[i:j] s[i:j] is palindrome,  need dp[i+1, j-1], i need i+1, should downto, j
        for(int i = n-1; i >= 0; i--)
            for(int j = i; j < n; j++)
            {
                if(s[i] == s[j] && ( (i+1 >j-1) || dp[i+1][j-1]))
                    dp[i][j] = true;
            }
        vector<vector<string> > result;
        vector<string> path;
        search(path, 0, s, dp, result);
        return move(result);
    }
```

其实像上面那样把每一个回文子串找出来后，就不用像排列组合那样去搜索了，可以直接构造。这个参考了leetcode-cpp。 `result[i]` 表示s[i:n]构成的回文串拆分结果。再走一遍dp就可以构造出来。方法如下: result[i]的结果为当前的回文串 插入每一个 result[i+1]构成。

```
vector<vector<string>> partitionDp(string s)
{
    int n = s.length();
    vector<vector<bool> > dp(n, vector<bool>(n, false));
    for(int i = 0; i < n; i++) //init, single char is palindrome
        dp[i][i] = true;
    //dp[i:j] s[i:j] is palindrome,  need dp[i+1, j-1], i need i+1, should downto, j
    for(int i = n-1; i >= 0; i--)
        for(int j = i; j < n; j++)
        {
            if(s[i] == s[j] && ( (i+1 >j-1) || dp[i+1][j-1]))
                dp[i][j] = true;
        }
    vector<vector<vector<string> >> result(n, vector<vector<string> >());
    for(int i = n-1; i >= 0; i--)
        for(int j = i; j < n; j++)
        {
            if(dp[i][j])
            {
                string str = s.substr(i, j-i+1);
                if(j+1 < n)
                {
                    vector<vector<string> > &next = result[j+1];
                    for(int k = 0; k < next.size(); k++)
                    {
                        auto v = next[k];
                        v.insert(v.begin(), str);
                        result[i].push_back(v);
                    }
                }
                else
                    result[i].push_back(vector<string>(1, str));
            }
        }
    return result[0];
}
```

Palindrome Partitioning II

题目来源：Palindrome Partitioning II

>

```
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.
```

解题思路：

>

```
Calculate and maintain 2 DP states:
pal[i][j] , which is whether s[i..j] forms a pal
d[i], which is the minCut for s[i..n-1]
Once we comes to a pal[i][j]==true:
    if j==n-1, the string s[i..n-1] is a Pal, minCut is 0, d[i]=0;
    else: the current cut num (first cut s[i..j] and then cut the rest s[j+1...n-1]) is 1
         compare it to the exisiting minCut num d[i], repalce if smaller.
d[0] is the answer.
```

第一步还是跟Palindrome Partitioning一样，用DP，任意i-j组合先计算好是否是回文；

第二步仍用DP，dp[i]表示s[i, len-1]最少的minCut, 对每一个palindrome[i][j]为true的:

- if j == len-1, 则 dp[i]=0;
- else s要拆分为s[i, j], [j+1, len-1],当前cut数=1+dp[j+1], 所以dp[i] = std::min(dp[i], 1+dp[j+1])

当然这两步也可以合在一起. ref

```
    int minCut(string s)
    {
        int n = s.length();
        vector<vector<bool> > dp(n, vector<bool>(n, false));
        for(int i = 0; i < n; i++)
            dp[i][i] = true;
        vector<int> cuts(n, 0);
        for(int i = n-1; i >= 0; i--)
        {
            cuts[i] = n-i-1;
            for(int j = i; j < n; j++)
            {
                if(s[i] == s[j] && ((i+1>j-1) || dp[i+1][j-1] ))
                {
                    dp[i][j] = true;
                    if(j == n-1)
                        cuts[i] = 0;
```

```
                    else //cut s[:j][j+1:n]
                        cuts[i] = std::min(cuts[i], 1 + cuts[j+1]);
                }
            }
        }
        return cuts[0];
    }
```

Scramble String

题目来源：Scramble String

> Given a string s1, we may represent it as a binary tree by partitioning it to two non-empty substrings recursively. Below is one possible representation of s1 = "great": great / \ gr eat / \ / \ g r e at / \ a t To scramble the string, we may choose any non-leaf node and swap its two children. For example, if we choose the node "gr" and swap its two children, it produces a scrambled string "rgeat". rgeat / \ rg eat / \ / \ r g e at / \ a t We say that "rgeat" is a scrambled string of "great". Similarly, if we continue to swap the children of nodes "eat" and "at", it produces a scrambled string "rgtae". rgtae / \ rg tae / \ / \ r g ta e / \ t a We say that "rgtae" is a scrambled string of "great". Given two strings s1 and s2 of the same length, determine if s2 is a scrambled string of s1.

解题思路：

用递归即可。 rg|tae gr|eat， rg和gr是scramble， tae和eat递归成 t|ae 和 ea|t， 因此最后满足条件。

```
bool isSameChar(string s1, string s2)
{
    int x[26] = {0};
    for(int i = 0; i < s1.length(); i++)
        ++x[s1[i]-'a'];
    for(int i = 0; i < s2.length(); i++)
        --x[s2[i]-'a'];
    for(int i = 0; i < 26; i++)
    {
        if(x[i] != 0) return false;
    }
    return true;
}
bool isScramble(const string &s1, const string &s2)
{
    if(s1.length() != s2.length()) return false;
    if(s1 == s2) return true;
    if(! isSameChar(s1, s2)) return false;
    int n = s1.length();
    for(int i = 1; i < n; i++)
    {
        if(isScramble(s1.substr(0, i), s2.substr(0, i)) && isScramble(s1.substr(i, n-
            return true;
        if(isScramble(s1.substr(0, i), s2.substr(n-i, i)) && isScramble(s1.substr(i,
            return true;
    }
    return false;
}
```

虽然能AC，但上面的代码效率确实～ 内存耗费不少吧，每次都去创建string出来。 参考下别人的代码，直接用迭代器来做，省掉了字符串的创建。 以上还可以把一些算过的用map cache起来, 学下 STL的tuple。

```
bool isSameChar(string::const_iterator first1, string::const_iterator first2, int len
{
```

```
        int x[26] = {0};
        for(auto i = first1; i != first1+len; i++)
            ++x[*i-'a'];
        for(auto i = first2; i != first2+len; i++)
            --x[*i-'a'];
        for(int i = 0; i < 26; i++)
        {
            if(x[i] != 0) return false;
        }
        return true;
    }
    bool isScramble(string::const_iterator first1, string::const_iterator first2, int len
    {
        if(len == 1) return *first1 == *first2;
        if(! isSameChar(first1, first2, len)) return false;
        for(int i = 1; i < len; i++)
        {
            if( (isScramble(first1, first2, i) && isScramble(first1+i, first2+i, len-i))
              ||(isScramble(first1, first2+len-i, i) && isScramble(first1+i, first2, len-
              return true;
        }
        return false;
    }
    bool isScramble(const string &s1, const string &s2)
    {
        if(s1.length() != s2.length()) return false;
        if(s1 == s2) return true;
        return isScramble(s1.begin(), s2.begin(), s1.length());
    }
```

设状态为 f[n][i][j]，表示长度为 n，起 点为 s1[i] 和起点为 s2[j] 两个字符串是否互为 scramble，则状态转移方程为

```
f[n][i][j] = (f[k][i][j] && f[n-k][i+k][j+k])
        || (f[k][i][j+n-k] && f[n-k][i+k][j])
```

跟上面递归的 `isScramble(string::const_iterator first1, string::const_iterator first2, int len)` 一致。

```
    bool isScramble(const string &s1, const string &s2)
    {
        if(s1.length() != s2.length()) return false;
        if(s1 == s2) return true;
        int n = s1.length();
        //dp[n][i][j], s1[i:i+n), s2[j:j+n) is scramble
        vector<vector<vector<bool> > > dp(n+1, vector<vector<bool>>(n, vector<bool>(n, fa
        for(int i = 0; i < n; i++)
            for(int j = 0; j < n; j++)
                dp[1][i][j] = s1[i] == s2[j];
        for(int len = 2; len <= n; len++)
            for(int i = 0; i <= n-len; i++)
                for(int j = 0; j <= n-len; j++)
                    for(int k = 1; k < len; k++)
                    {
                        if( (dp[k][i][j] && dp[len-k][i+k][j+k]) ||
                            (dp[k][i][j+len-k] && dp[len-k][i+k][j]))
```

```
                            {
                                dp[len][i][j] = true;
                                break;
                            }
                        }
            return dp[n][0][0];
        }
```

参考 leetcode-cpp

Triangle

题目来源：Triangle

>

```
Given a triangle, find the minimum path sum from top to bottom. Each step you may move to
For example, given the following triangle
[
     [2],
    [3,4],
   [6,5,7],
  [4,1,8,3]
]
The minimum path sum from top to bottom is 11 (i.e., 2 + 3 + 5 + 1 = 11).
Note:
Bonus point if you are able to do this using only O(n) extra space, where n is the total
```

解题思路：

动态规划，从下往上走～ `rows[col] = std::min(rows[col], rows[col+1]) + triangle[row][col];`

```
    int minimumTotal(vector < vector< int > > &triangle )
    {
        int m = triangle .size();
        if (m == 0) return 0;
        vector< int > dp(triangle [m-1]);
        for( int r = m-2; r >= 0; r--) //triangle, if m is 1, dp[0] is result
            for (int c = 0; c <= r; c++) //triangle[r].size() = r+1
                dp[c] = std::min(dp[c], dp[c+1]) + triangle [r][c];
        return dp[0];
    }
```

如果triangle值可以改变的话，可以O(1)的空间复杂度。

```
    //triangle can be changed
    int minimumTotal(vector < vector< int > > &triangle )
    {
        int m = triangle .size();
        if (m == 0) return 0;
        for( int r = m-2; r >= 0; r--) //triangle, if m is 1, dp[0] is result
            for (int c = 0; c <= r; c++) //triangle[r].size() = r+1
                triangle [r][c] += std::min(triangle [r+1][c], triangle[r+1][c+1]);
        return triangle [0][0];
}
```

Unique Binary Search Trees

题目来源：Unique Binary Search Trees

> Given n, how many structurally unique BST's (binary search trees) that store values 1...n? For example, Given n = 3, there are a total of 5 unique BST's. 1 3 3 2 1 \ / / / \ \ 3 2 1 1 3 2 / / \ \ 2 1 2 3

解题思路：

递归

递归比较好理解。比如 根节点数字为i, 比i小的左孩纸i-1个(子问题), 右孩纸n-i. 于是就有了下面的代码。

```
int numTrees(int n)
{
    if(n == 0) return 1;//recursion, maybe, real input 0 shoule return 0
    if(n == 1) return 1;
    int r = 0;
    for(int i = 1; i <= n; i++)
        r += numTrees(i-1)*numTrees(n-i);
    return r;
}
```

动态规划

其实可以缓存下, 用动态规划。

```
int f(int n)
{
    const int size = n+1;
    vector<int>  cache(size, 1);;
    for(int i = 2; i <=n; i++)
    {
        int result = 0;
        for(int j = 1; j <=i; j++)
            result += cache[j-1] * cache[i-j];
        cache[i] = result;
    }
    return cache[n];
}
int numTrees(int n)
{
    if(n == 0) return 0;
    return f(n);
}
```

数学公式法

其实这个问题有公式可以直接算的，参考卡塔兰数 。

Unique Paths II

题目来源：Unique Paths II

> Follow up for "Unique Paths": Now consider if some obstacles are added to the grids. How many unique paths would there be? An obstacle and empty space is marked as 1 and 0 respectively in the grid. For example, There is one obstacle in the middle of a 3x3 grid as illustrated below. [[0,0,0], [0,1,0], [0,0,0]] The total number of unique paths is 2. Note: m and n will be at most 100.

解题思路： 跟 minimum-path-sum差不多. 这里就省却递归方法了， 直接用DP。

```
int uniquePathsWithObstacles(vector<vector<int> > &grid)
{
    int m = grid.size(); if (m == 0) return 0;
    int n = grid[0].size();
    if(grid[0][0] == 1) return 0;
    vector<vector<int> > dp(m, vector<int>(n, 0));
    for(int i = 0; i < n; i++)
    {
        if(grid[0][i] == 1)
            break;
        dp[0][i] = 1;
    }
    for(int i = 0; i < m; i++)
    {
        if(grid[i][0] == 1)
            break;
        dp[i][0] = 1;
    }
    for(int i = 1; i < m; i++)
        for(int j = 1; j < n; j++)
        {
            if(grid[i][j] == 1)
                dp[i][j] = 0;
            else
                dp[i][j] = dp[i-1][j] + dp[i][j-1];
       }
    return dp[m-1][n-1];
}
```

Word Break

题目来源：Word Break

>

```
Given a string s and a dictionary of words dict, determine if s can be
segmented into a space-separated sequence of one or more dictionary words.

For example, given
s = "leetcode",
dict = ["leet", "code"].

Return true because "leetcode" can be segmented as "leet code".
```

解题思路：

首先想到的就是DFS，直接挨个搜索，能走到结尾就OK。不过这样做超时了。

```
//TLE
bool dfs(string s, int startIndex, unordered_set<string> &dict)
{
    if(startIndex > s.length()) return false;
    if(startIndex == s.length()) return true;
    for(int i = startIndex; i < s.length(); i++)
    {
        string pre = s.substr(startIndex, i - startIndex + 1);
        if(dict.find(pre) != dict.end())
        {
            if(dfs(s, i+1, dict))
                return true;
        }
    }
    return false;
}
bool wordBreak(string s, unordered_set<string> &dict)
{
    if(s.length() == 0) return false;
    return dfs(s, 0, dict);
}
```

然后用DP, `dp[i]` 表示s[0:i]都跟dict对应了, 对于更长的j, `dp[j]` 为true的话，肯定存在i使得 `dp[i]` 为true和 `s[i:j]` 能在dict中找到。
因此得到如下代码：

```
bool dp(string s, unordered_set<string> &dict)
{
    int n = s.length();
    vector<bool> dp(n+1, false);
    dp[0] = true;
    for(int j = 1; j <= n; j++)
        for(int i = 0; i < j; i++)
```

```
            {
                if(dp[i]) //dp[i], true, dp[i] && s[i:j]==> dp[j]
                {
                    string str = s.substr(i, j-i);
                    if(dict.find(str) != dict.end())
                    {
                        dp[j] = true;
                        break;
                    }
                }
            }
        return dp[n];
    }
```

Word Break II

题目来源：Word Break II

>

```
Given a string s and a dictionary of words dict, add spaces in s to construct a sentence

Return all such possible sentences.

For example, given
s = "catsanddog",
dict = ["cat", "cats", "and", "sand", "dog"].

A solution is ["cats and dog", "cat sand dog"].
```

解题思路：

沿用Word break的思路，dp[i]表示s中0到i的串能在dict中对应，首先想到的是用vector把dp[i]为true的情况都保存下来(即以i为结尾的单词)，最后组装回去可得到结果。注意不能每次得到dp[i]为true就去枚举结果，这样会超时(最后可能没有成功走到结尾也浪费了时间在这里去拼装)。

代码如下：

```
 void searchResult(string input, vector<vector<string> >&dp, int len, vector<string>
{
    if(len <= 0)
    {
        if(len == 0)
            result.push_back(input);
        return;
    }
    for(int i = 0; i < dp[len].size(); i++)
    {
        string str = dp[len][i];
        if(input.length() > 0)
            searchResult(str + " " + input, dp, len - str.length(), result);
        else
            searchResult(str, dp, len - str.length(), result);
    }
}
vector<string> dp(string s, unordered_set<string> &dict)
{
    int n = s.length();
    vector<bool> dp(n+1, false);
    dp[0] = true;
    vector<vector<string> > dpStrings(n+1, vector<string>());
    for(int j = 1; j <= n; j++)
        for(int i = 0; i < j; i++)
        {
            if(dp[i]) //dp[i], true, dp[i] && s[i:j]==> dp[j]
            {
                string str = s.substr(i, j-i);
                if(dict.find(str) != dict.end())
```

```
                    {
                        dp[j] = true;
                        //break;, can NOT break, for wordbreak II should return all the p
                        dpStrings[j].push_back(str);
                    }
                }
            }
        vector<string> result;
        if(dp[n])
            searchResult("", dpStrings, n, result);
        return result;
    }
    vector<string> wordBreak(string s, unordered_set<string> &dict) {
        return dp(s, dict);
    }
```

其实，按照上面提的思路一边dp的时候就去枚举结果也是可以的，测试用例中就那一个较长的过不了，先按照wordbreak的思路detective一下再枚举就可以AC。(一般人我不告诉他) :)

```
    //中途的复杂度记录dps的复杂度较高，用例（aaaaaaaaaaaaaaaa*b, aaaaaa...aaa）可能最后没有结果 但
    //先detect一下 可以通过oj
    vector<string> wordBreak2(string s, unordered_set<string> &dict)
    {
        if(! wordBreak(s, dict))
            return vector<string>();
        size_t len = s.length();
        vector<bool> dp(len+1, false);
        dp[0] = true; //dp[i] : s[0:i] is ok
        vector< vector<string> > dps(len+1);
        for(int i = 1; i <= len; i++)
        {
            for(int j = 0; j < i; j++)
            {
                if(dp[j]) // dp[0:j] ok + s[j:i] ok ---> dp[i] is ok
                {
                    string sub =  s.substr(j, i-j);
                    if(dict.find(sub) != dict.end())
                    {
                        dp[i] = true;
                        if(dps[j].size() > 0)
                        {
                            for(auto it = dps[j].begin(); it != dps[j].end(); it++)
                                dps[i].push_back(string((*it) + " "+ sub));
                        }else
                        {
                            dps[i].push_back(sub);
                        }

                    }
                }
            }
        }
        return dps[len];
    }
```

list, 链表相关

1. Add Two Numbers 题解
2. Convert Sorted List to Binary Search Tree 题解
3. Copy List with Random Pointer 题解
4. Insertion Sort List 题解
5. LRU Cache 题解
6. Linked List Cycle 题解
7. Linked List Cycle II 题解
8. Merge Two Sorted Lists 题解
9. Merge k Sorted Lists 题解
10. Partition List 题解
11. Remove Duplicates from Sorted List 题解
12. Remove Duplicates from Sorted List II 题解
13. Remove Nth Node From End of List 题解
14. Reorder List 题解
15. Reverse Linked List II 题解
16. Reverse Nodes in k-Group 题解
17. Rotate List 题解
18. Sort List 题解
19. Swap Nodes in Pairs 题解

Add Two Numbers

题目来源：Add Two Numbers

> 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

解题思路：

递归版

```
ListNode *addTwoNumbers(ListNode *l1, ListNode *l2, int carry)
{
    if(l1 == NULL && l2 == NULL && carry == 0) //only comes the carry,like 5, 5 = 10,
        return NULL;
    ListNode * result = new ListNode(carry);
    if(l1 != NULL)
        result->val += l1->val;
    if(l2 != NULL)
        result->val += l2->val;
    result->next = addTwoNumbers(l1 == NULL ? NULL : l1->next, l2 == NULL ? NULL : l2
    result->val %= 10;
    return result;
}

ListNode *addTwoNumbers(ListNode *l1, ListNode *l2)
{
    if(l1 == NULL && l2 == NULL)
        return NULL;
    if(l1 == NULL)
        return l2;
    if(l2 == NULL)
        return l1;
    return addTwoNumbers(l1, l2, 0);
}
```

迭代版

```
ListNode *addTwoNumbers(ListNode *l1, ListNode *l2)
{
    if(l1 == NULL && l2 == NULL) return NULL;
    if(l1 == NULL || l2 == NULL) return l1 == NULL ? l2 : l1;
    ListNode dummy(-1);
    ListNode * pre = &dummy;
    while(l1 && l2)
    {
        pre->next = new ListNode(l1->val + l2->val);
        pre = pre->next;
        l1 = l1->next;
        l2 = l2->next;
```

```
        }
        while(l1){
            pre->next = new ListNode(l1->val);
            pre = pre->next;
            l1 = l1->next;
        }
        while(l2){
            pre->next = new ListNode(l2->val);
            pre = pre->next;
            l2 = l2->next;
        }
        pre = dummy.next;
        while(pre){
            int t = pre->val;
            if(t >= 10) {
                pre->val = t % 10;
                if(pre->next)
                    pre->next->val += t / 10;
                else
                    {pre->next = new ListNode(t / 10); break;}
            }
            pre = pre->next;
        }
        return dummy.next;
    }
```

Convert Sorted List to Binary Search Tree

题目来源：Convert Sorted List to Binary Search Tree

> Given a singly linked list where elements are sorted in ascending order, convert it to a height balanced BST.

解题思路：

tricky 方法, 另外取O(n)空间

偷懒方法，另外取另外取O(n)空间把list的数据取出来放入数组，然后跟题目一样用数组的方式去做。代码就略过了。虽然不是出题者的本意～ 但..... 你咬我呀.

O(nlogn)时间

每次用O(len/2)的时间去把中间的节点找出来。然后跟数组一样的方式解决。时间复杂度为O(nlogn).中途找mid不跟数组一样O(1).

```
int length(ListNode * head)
{
    int len = 0;
    while(head)
    {
        ++len;
        head = head->next;
    }
    return len;
}
TreeNode * convert(ListNode * head, int len)
{
    if(head == NULL || len == 0) return NULL;
    if(len == 1) return new TreeNode(head->val);
    int mid = len>>1;
    ListNode * pre = head;
    int i = mid;
    while(--i)
        pre=pre->next;
    int leftlen = mid-1; int rightlen = len-mid;//even
    if(len & 0x1)
    {
        pre = pre->next;
        leftlen = mid;
        rightlen = len-mid-1;
    }
    auto root = new TreeNode(pre->val);
    root->left = convert(head, leftlen);
    root->right = convert(pre->next, rightlen);
}

TreeNode *sortedListToBST(ListNode *head)
{
    int len = length(head);
    return convert(head, len);
```

```
    }
```

从底至上构造Tree, 递归(得忽略递归调用的时间/空间消耗)调用，递归中传同一个链表，链表不停往前走，通过下标关系来控制左右子树。进入递归时链表指向头节点，结束递归时，链表指向尾节点的next。

下面代码中，每次递归调用开始时，节点指针都指向区间第一个，结束时节点的指针指向区间末尾的后一个。每次递归调用时，分成左右两部分，左边构造完时，正好指针指向mid，创建一下root，继续构造右部分子树。ref

```
int length(ListNode * head)
{
    int len = 0;
    while(head)
    {
        ++len;
        head = head->next;
    }
    return len;
}
TreeNode * convert(ListNode * &head, int start, int end)
{
    assert(start <= end);
    if(start == end) {
        auto result = new TreeNode(head->val);
        head=head->next;
        return result;
    }
    TreeNode* left = NULL;
    int mid = start + ((end-start)>>1);
    if(start <= mid-1)
        left = convert(head, start, mid-1);
    TreeNode * root = new TreeNode(head->val);
    head = head->next;
    root->left = left;
    if(mid+1 <= end)
        root->right = convert(head, mid+1, end);
    return root;
}

TreeNode *sortedListToBST(ListNode *head)
{
    if(head == NULL) return NULL;
    int len = length(head);
    return convert(head, 0, len-1);
}
```

或许 convert这样写更简洁。

```
TreeNode * convert(ListNode * &head, int start, int end)
{
    if(start > end) return NULL;
    int mid = start + ((end-start)>>1);
    TreeNode* left = convert(head, start, mid-1);
    TreeNode * root = new TreeNode(head->val);
    head = head->next;
```

```
        root->left = left;
        root->right = convert(head, mid+1, end);
        return root;
    }
```

Copy List with Random Pointer

题目来源：Copy List with Random Pointer

>

```
A linked list is given such that each node contains an additional random
pointer which could point to any node in the list or null.

Return a deep copy of the list.
```

```
struct RandomListNode
{
    int label;
    RandomListNode *next, *random;
    RandomListNode(int x) : label(x), next(NULL), random(NULL) {}
};
```

解题思路：

传统方法用hashmap

主要是解决random pointer的问题，可以用一个map把copy过的存起来，下次碰到的时候直接从map中取。

```
RandomListNode *copyRandomListUseMap(RandomListNode *head)
{
    unordered_map<RandomListNode*, RandomListNode*> copyed;
    RandomListNode * head1 = new RandomListNode(head->label);
    copyed[head] = head1;
    RandomListNode * headbak = head1;
    //node's next can not construct a cycle.
    while(head)
    {
        if(head->next){
            if (copyed.find(head->next) != copyed.end())
                head1->next = copyed[head->next];
            else
            {
                head1->next = new RandomListNode(head->next->label);
                copyed[head->next] = head1->next;
            }
        }
        if(head->random){
            if (copyed.find(head->random) != copyed.end())
                head1->random =  copyed[head->random];
            else
            {
                head1->random = new RandomListNode(head->random->label);
                copyed[head->random] = head1->random;
            }
        }
        head = head->next;
        head1 = head1->next;
```

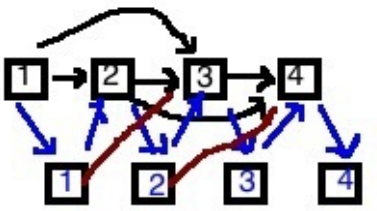
```
        }
        return headbak;
    }

    RandomListNode *copyRandomList(RandomListNode *head)
    {
        if(head == NULL) return NULL;
        return copyRandomListUseMap(head);
    }
```

常数空间神奇妙解

上面的的方法用了额外的空间，网上总是有些高人能想出牛B的解法。下面就是一个。

不用map等数据结构常数空间解Copy List with Random Pointer，该方法分3步：

```
1. 直接遍历原链表，copy一遍，random指向跟原来一样的node。将copy的node0插入到原来链表的node0和node1
2. 根据第一步有，nodei->next是node的copy的那个(i为偶数)，现在要解决copy的node的random问题，即 nod
3. 将这个连接在一起的链表分开，注意要断开。
```



举例个例子, 上面的一排数据1,2,3,4原始数据前后通过next连接, 然后1和2的random指针分别指向3和4.

第1步,copy并把copy得到的数据放到原来的后面.得到下一排数据.

然后观察, 1->next->random = ? 即 copy得到的蓝色1的random指针应该指向哪? 其实就是原来的黑色1->random(黑色3)->next(蓝色3),即 `nodei->next->random = nodei->random->next`

第3步,再断开，得到copy后的链表。

AC代码如下:

```
    RandomListNode *copyRandomListConstantSpace(RandomListNode *head)
    {
        RandomListNode * src = head;
        while(src)
        {
            RandomListNode* copy = new RandomListNode(src->label);
            copy->random = src->random;
            //insert between src, src->next;
            auto next = src->next;
            copy->next = next;
            src->next = copy;
            src = next;
        }
        //step 2, deal with random pointer
        src = head;
```

```
        while(src)
        {
            //src->next is the copyed one, src->random->next is the copied src->random
            src->next->random = src->random == NULL ? NULL : src->random->next;
            src = src->next->next;//src->next is surely not NULL
        }
        //split
        src = head;
        auto result = src->next;
        auto cpy = result;
        while(src->next)
        {
            src->next = src->next->next;
            src = src->next;
            if(src == NULL) break;
            cpy->next = cpy->next->next;
            cpy = cpy->next;
        }
        return result;
    }
```

Insertion Sort List

题目来源：Insertion Sort List

> Sort a linked list using insertion sort.

解题思路： 发现有逆序的就将这个插入到前面的有序序列里。

```
ListNode *insertionSortList(ListNode *head)
{
    if(NULL == head || NULL == head->next) return head;
    ListNode  result(1);
    result.next = head;
    ListNode * nodei = head;
    while(nodei && nodei->next)
    {
        ListNode * nodej = nodei->next;
        if(nodej->val >= nodei->val)
        {
            nodei = nodei->next;
            continue;
        }
        //insert nodej to result
        ListNode * pos = &result;
        while(pos->next->val < nodej->val)
            pos = pos->next;
        //[pos, pos->next]  -->[pos, nodej, pos->next]
        nodei->next = nodej->next;
        nodej->next = pos->next;
        pos->next = nodej;
    }
    return result.next;
}
```

顺便把选择排序也写下.

```
ListNode * selectionSort(ListNode *head)
{
    ListNode * nodei = head;
    while(nodei)
    {
        ListNode * nodemin = nodei;
        ListNode * nodej = nodei;
        while(nodej)
        {
            if(nodej->val < nodemin->val)
                nodemin = nodej;
            nodej = nodej->next;
        }
        std::swap(nodei->val, nodemin->val);
        nodei = nodei->next;
    }
    return head;
}
```

LRU Cache

题目来源：LRU Cache

> Design and implement a data structure for Least Recently Used (LRU) cache. It should support the following operations: get and set. > get(key) - Get the value (will always be positive) of the key if the key exists in the cache, otherwise return -1. set(key, value) - Set or insert the value if the key is not already present. When the cache reached its capacity, it should invalidate the least recently used item before inserting a new item.

解题思路：双向链表 + hashmap

ref

```
Once the data with key K is queried, the function get(K) is first called.
If the data of key K is in the cache, the cache just returns the data and refresh the dat
To refresh data T in the list, we move the item of data T to the head of the list. Otherw
if the data T of key K is not in the cache, we need to insert the pair into the list.
If the cache is not full, we insert into the hash map, and add the item at the head of th
If the cache is already full, we get the tail of the list and update it with , then move
```

ref1

```
LRU的典型实现是hash map + doubly linked list， 双向链表用于存储数据结点，并且它是按照结点最近被使用的
我们有理由相信它在接下来的一段时间被访问的概率要大于其它结点。于是， 我们把它放到双向链表的头部。当我们往双
同样把它插入到头部。 我们使用这种方式不断地调整着双向链表，链表尾部的结点自然也就是最近一段时间， 最久没有
需要替换掉的就是双向链表中最后的那个结点(不是尾结点，头尾结点不存储实际内容)。

如下是双向链表示意图，注意头尾结点不存储实际内容：

头 --> 结 --> 结 --> 结 --> 尾
结     点     点     点     结
点 <-- 1  <-- 2 <-- 3  <-- 点
假如上图Cache已满了，我们要替换的就是结点3。

哈希表的作用是什么呢？如果没有哈希表，我们要访问某个结点，就需要顺序地一个个找， 时间复杂度是O(n)。使用哈希
或者返回未找到。
```

```
class LRUCache
{
public:
    struct Node
    {
        int value;
        int key;
        Node * prev;
        Node * next;
        Node(int k, int v): key(k),value(v),prev(NULL), next(NULL)
        {}
    };
```

```
    LRUCache(int capacity):capacity_(capacity)
    {
        head = new Node(-1,-1);
        tail = new Node(-1,-1);
        head->next = tail;
        tail->prev = head;
    }

    ~LRUCache()
    {
        delete head;
        delete tail;
        for(auto it = kv_.begin(); it!=kv_.end(); it++)
            delete it->second;
    }

    int get(int key)
    {
        auto it = kv_.find(key);
        if(it != kv_.end())
        {
            removeFromList(it->second);
            move2Head(it->second);
            return it->second->value;
        }
        return -1;
    }

    void set(int key, int value)
    {
        auto it = kv_.find(key);
        if(it != kv_.end())//replace
        {
            removeFromList(it->second);
            it->second->value = value;
            move2Head(it->second);
        }else//insert new
        {
            if(kv_.size() >= capacity_)//remove tail
            {
                auto node = tail->prev;
                removeFromList(node);//!!Do not forget
                kv_.erase(node->key);
                node->key = key;
                node->value = value;
                kv_[key] = node;
                move2Head(node);
            }else //add
            {
                auto node = new Node(key, value);
                kv_[key] = node;
                move2Head(node);
            }
        }
    }
private:
    Node * head; //head and tail do NOT store data
    Node * tail;
    unordered_map<int, Node*> kv_;
    int capacity_;
    void removeFromList(Node* node)
    {
```

```
            node->prev->next = node->next;
            node->next->prev = node->prev;
        }
        void move2Head(Node* node)
        {
            Node* old = head->next;
            old->prev = node;
            node->next = old;
            head->next = node;
            node->prev = head;
        }
    };
```

看看人家用stl中的list写的，代码多短啊。 ref leetcode-cpp list中的方法

> splice (iterator position, list& x, iterator i); Transfers elements from x into the container, inserting them at position. list api

```
    class LRUCache
    {
    public:
        struct Node
        {
            int value;
            int key;
            Node(int k, int v): key(k),value(v)
            {}
        };

        LRUCache(int capacity):capacity_(capacity)
        {
        }

        int get(int key)
        {
            auto it = kv_.find(key);
            if(it != kv_.end())
            {
                //transfer: remove it from dataList, add to dataList.begin()
                //std::list::splice http://www.cplusplus.com/reference/list/list/splice/
                dataList.splice(dataList.begin(), dataList, it->second);
                it->second = dataList.begin();
                return it->second->value;
            }
            return -1;
        }

        void set(int key, int value)
        {
            auto it = kv_.find(key);
            if(it != kv_.end())//replace
            {
                dataList.splice(dataList.begin(), dataList, it->second);
                it->second = dataList.begin();
                it->second->value = value;
            }else//insert new
            {
                if(kv_.size() >= capacity_)//remove tail
                {
```

```
                    kv_.erase(dataList.back().key);
                    auto tail = dataList.back();
                    dataList.pop_back();
                    tail.key = key; tail.value = value;
                    dataList.push_front(tail);
                    kv_[key] = dataList.begin(); //insert
                }else
                {
                    dataList.push_front(Node(key, value));
                    kv_[key] = dataList.begin(); //insert
                }
            }
        }
    private:
        unordered_map<int, list<Node>::iterator> kv_;
        int capacity_;
        list<Node> dataList;
    };
```

Linked List Cycle

题目来源：Linked List Cycle

>

```
Given a linked list, determine if it has a cycle in it.
```

解题思路：

简单的快慢指针.

```
    bool hasCycle(ListNode *head)
    {
        if(head == NULL || head->next == NULL) return false;
        ListNode * fast = head;
        ListNode * slow = head;
        while(fast != NULL && fast->next != NULL){
            fast = fast->next->next;
            slow = slow->next;
            if(fast == slow) return true;
        }
        return false;
    }
```

Linked List Cycle II

题目来源：Linked List Cycle II

>

```
Given a linked list, return the node where the cycle begins. If there is no cycle, return
```

解题思路：

跟Linked List Cycle 题解一样,运用快慢指针, 先判断是否存在环。然后再找环的起点。 假设存在环，不妨设这个带环的链表长成这个样子(将就看)：

```
s-------a->-------
        |         |
        |-<-c-----|
```

如图示，链表起点s, a是环开始的点，假设快慢指针第一次相遇的点在c处。我们将这个环分为3段，

1. s-->a: 链表起点到还的起点，假设距离为a;
2. a-->c: 环的起点到快慢指针首次相交的点c，设距离为b;
3. c-->a: 快慢指针首次相交的点，再转回环起点a的距离设为c;

那么，得到环的长度为 `b+c` , 到快慢指针首次相交时，快/慢指针走的距离设fast/slow分别为：

- fast = a + b + n*(b + c), n >= 1, (可能快指针比慢指针多走了不止1圈)
- slow = a + b

又因为fast每次走两步，slow每次走一步，所以有

```
fast = 2 * slow
a + b + n*(b+c) = 2 * (a+b)
(n-1)*(b+c) + (b+c) = a+b
(n-1)*(b+c) + c = a;
```

观察这个式子，若首次相交时，将fast指针若回到链表头，然后fast、slow都一步一步走，那么他们将在哪相遇?

fast走到a时，slow恰好走了(n-1)圈+c，刚好回到了环起点a处。

因此首次相遇后，再次相遇的点即为环的起点。

代码如下：

```
    ListNode *detectCycle(ListNode *head)
    {
        if(head == NULL || head->next == NULL) return NULL;
```

```
        ListNode * fast = head;
        ListNode * slow = head;
        while(fast != NULL && fast->next != NULL)
        {
            fast = fast->next->next;
            slow = slow->next;
            if(fast == slow) //find intersection node
            {
                fast = head;
                while(fast != slow)
                {
                    fast = fast->next;
                    slow = slow->next;
                }
                return fast;
            }
        }
        return NULL;
    }
```

Merge Two Sorted Lists

题目来源：Merge Two Sorted Lists

> 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.

解题思路：

用一个多余的节点在头部，使得代码简洁，不然得去内部判断到底哪个节点当作新list的head。

```
ListNode *mergeTwoLists(ListNode *l1, ListNode *l2)
{
    ListNode dummy(-1);
    ListNode * pre = &dummy;
    while(l1 && l2)
    {
        if (l1->val <= l2->val)
            pre->next = l1, l1 = l1->next;
        else
            pre->next = l2, l2 = l2->next;
        pre = pre->next;
    }
    if (l1 != NULL)
        pre->next = l1;
    else
        pre->next = l2;
    return dummy.next;
}
```

Merge k Sorted Lists

题目来源：Merge k Sorted Lists

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

解题思路：

跟merge-two-sorted-lists 一样, 一个一个merge即可。

```
ListNode * merge2List(ListNode * list1, ListNode* list2)
{
   ListNode dummy(-1);
   ListNode * pre = &dummy;
   while(list1 && list2)
   {
       if(list1->val <= list2->val)
           pre->next = list1, list1 = list1->next;
       else
           pre->next = list2, list2 = list2->next;
       pre = pre->next;
   }
   if(list1)
       pre->next = list1;
   else
       pre->next = list2;
   return dummy.next;
}

ListNode *mergeKLists(vector<ListNode *> &lists)
{
   if(lists.size() == 0) return NULL;
   ListNode * result = lists[0];
   for(int i = 1; i < lists.size(); i++)
       result = merge2List(result, lists[i]);
   return result;
}
```

不过超时了. 从超时的testcase可以看出,全是短的链表， 加到结果集的链表后，新的短链表加进去运气不好又得将长链表遍历完后才能加到结果链表中。

改成如下代码就可以AC了。

```
ListNode *mergeKLists(vector<ListNode *> &lists)
{
   if(lists.size() == 0) return NULL;
   vector<ListNode*> merged(lists);
   int size = (int)merged.size();
   while(size > 1)
   {
       for(int i = 0; i < size / 2; i++)
           merged[i] = merge2List(merged[i*2], merged[i*2+1]);
       if((size & 0x1) == 1)
           merged[size / 2 - 1] = merge2List(merged[size / 2 - 1], merged[size-1]);
       size /= 2;
   }
```

```
    return merged[0];
}
```

Partition List

题目来源：Partition List

> Given a linked list and a value x, partition it such that all nodes less than x come before nodes greater than or equal to x. You should preserve the original relative order of the nodes in each of the two partitions. For example, Given 1->4->3->2->5->2 and x = 3, return 1->2->2->4->3->5.

解题思路：

注意一些边界情况， 要保持以前的两个节点顺序。不然就可以把小于x的一个一个往最前面插入。

```
ListNode *partition(ListNode *head, int x)
{
    ListNode dummy1(-1), dummy2(-1);
    ListNode * left = &dummy1;
    ListNode * right = &dummy2;
    ListNode * node = head;
    while(node)
    {
        if(node->val < x)
        {
            left->next = node;
            left = left->next;
        }else
        {
            right->next = node;
            right = right->next;
        }
        node = node->next;
    }
    left->next = dummy2.next;
    right->next = NULL;
    return dummy1.next;
}
```

Remove Duplicates from Sorted List

题目来源：Remove Duplicates from Sorted List

> Given a sorted linked list, delete all duplicates such that each element appear only once. For example, Given 1->1->2, return 1->2. Given 1->1->2->3->3, return 1->2->3.

解题思路：

```
ListNode *deleteDuplicates(ListNode *head)
{
    ListNode * result = head;
    ListNode * resultBak = result;
    while(head)
    {
        while(head != NULL && head->val == result->val)
            //free 'head'
            head = head->next;
        result->next = head;///result->val != head
        result = result->next;
    }
    return resultBak;
}
```

把与上一个节点相同的值略过， [1] {1 1} 2 2 ... 上面代码保留相同中的第一个， 会造成内存泄漏。 下面代码是保留相同中的最后一个，之前的都delete掉。

```
ListNode *deleteDuplicates(ListNode *head)
{
    if(head == NULL || head->next == NULL) return head;
    ListNode dummy(-1);
    ListNode * result = &dummy;
    while(head)
    {
        while(head != NULL && head->next != NULL && head->val == head->next->val)
        {
            auto next = head->next;  //1 1 1 2
            delete head;//free 'head'
            head = next;
        }
        result->next = head;
        result = result->next;
        if(head) head = head->next;
    }
    return dummy.next;
}
```

Remove Duplicates from Sorted List II

题目来源：Remove Duplicates from Sorted List II

> Given a sorted linked list, delete all nodes that have duplicate numbers, leaving only distinct numbers from the original list. For example, Given 1->2->3->3->4->4->5, return 1->2->5. Given 1->1->1->2->3, return 2->3.

解题思路：

用一个变量标记是否有相同的节点，直到不同的才连接到result中。

```
ListNode *deleteDuplicates(ListNode *head)
{
    ListNode dummy(-1);
    ListNode * result = &dummy;
    while(head)
    {
        auto node = head->next;
        bool rep = false;
        while(node != NULL && node->val == head->val)
        {
            auto next = node->next;
            delete node;
            node = next;
            rep = true;
        }
        if(rep) //1 1 1 2 2
            {delete head; head = node;}
        else //1 2 2
            {result->next = head; result = result->next;  head = head->next;}
    }
    result->next = NULL; //!!IMPORTANT
    return dummy.next;
}
```

别忘了最后的节点->next需要置空。

Remove Nth Node From End of List

题目来源：Remove Nth Node From End of List

> Given a linked list, remove the nth node from the end of list and return its head. For example, Given linked list: 1->2->3->4->5, and n = 2. After removing the second node from the end, the linked list becomes 1->2->3->5. Note: Given n will always be valid. Try to do this in one pass.

解题思路：

快慢指针的思路, 先走n步，然后另外一个节点从头开始，直到先走的那个节点到达结尾，第二次从头开始的那个点极为要删除的那个节点。 注意边界条件可能要删除头。

```
ListNode *removeNthFromEnd(ListNode *head, int n)
{
    assert(head != NULL);
    //n is valid
    int k = 0;
    ListNode *headbak = head;
    while(k++ < n)
        head = head->next;
    ListNode * pre = headbak;
    if(head == NULL) //delete head
    {
        ListNode * result = headbak->next;
        delete pre;
        return result;
    }
    while(head->next)
    {
        pre = pre->next;
        head = head->next;
    }
    //delete pre->next;
    auto deleted = pre->next;
    auto next = deleted->next;
    pre->next = next;
    delete deleted;
    return headbak;
}
```

Reorder List

题目来源：Reorder List

>

```
Given a singly linked list L: L0→L1→…→Ln-1→Ln,
reorder it to: L0→Ln→L1→Ln-1→L2→Ln-2→…

You must do this in-place without altering the nodes' values.

For example,
Given {1,2,3,4}, reorder it to {1,4,2,3}.
```

解题思路：

`O(n)` 时间 + `O(n)` 空间

将node都copy出来放到数组里，后半段逆序(或者直接通过下标不用逆序)连接前半段。

```
    //O(n) + O(n)
    void reorderList2(ListNode* head)
    {
        vector<ListNode*> nodes;
        while(head)
        {
            nodes.push_back(head);
            head = head->next;
        }
        int n = (int)nodes.size();
        int mid = n >> 1;
        std::reverse(nodes.begin()+mid, nodes.end());
        for(int i = 0; i < mid; i++)
        {
            nodes[i]->next = nodes[i+mid];
            nodes[i+mid]->next = nodes[i+1];
        }
        if((n & 0x1) == 0x1)//odd 0,1,2,3,4 | 0,1,4,3,2 | 0->4->1->3->4 ==>0->4->1->3->2-
        {
            nodes[n-2]->next = nodes[n-1];
            nodes[n-1]->next = NULL;
        }//even 0,1,2,3 | 0,1,3,2| 0->3->1->2
        else
            nodes[n-1]->next = NULL;
    }
```

`O(n)` 时间 + `O(1)` 空间

这才是出题者的意图，同样后半段逆序，但通过指针的方式就地逆序，然后与前半段连接。

```
```

```
ListNode * reverseList(ListNode * head)
{
    if(head == NULL) return NULL;
    ListNode * pre = head;
    ListNode * node = head->next;
    while(node){
        ListNode* next = node->next;
        node->next = pre;
        pre = node;
        node = next;
    }
    head->next = NULL; //!!Do not forget
    return pre;
}

//O(n) time + O(1) space
void reorderList(ListNode *head)
{
    if(head == NULL || head->next == NULL) return ;

    ListNode * first = head;
    ListNode * fast = head;
    while(fast != NULL && fast->next != NULL)
    {
        fast = fast->next->next;
        head = head->next;
    }
    ListNode * second = head->next;
    head->next = NULL;
    second = reverseList(second);

    while(second != NULL)
    {
        ListNode * tmp = second->next;
        second->next = first->next;
        first->next = second;
        first = first->next->next;
        second = tmp;
    }
}
```

Reverse Linked List II

题目来源：Reverse Linked List II

> Reverse a linked list from position m to n. Do it in-place and in one-pass. For example: Given 1->2->3->4->5->NULL, m = 2 and n = 4, return 1->4->3->2->5->NULL. Note: Given m, n satisfy the following condition: 1 ≤ m ≤ n ≤ length of list.

解题思路： 画个图 较清晰。

```
1-> 2->3 -> 4->5->NULL
pre m cur next
pre不变, 一个一个插入到pre后面.
```

```
ListNode *reverseBetween(ListNode *head, int m, int n)
{
    if(m == n) return head;
    ListNode dummy(-1);
    ListNode * pre = &dummy;
    pre->next = head;
    int i = 1;
    while(i++ < m)
        pre = pre->next;

    //reverse between [m n]
    ListNode * mthNode = pre->next;
    ListNode * cur = mthNode->next, *next = NULL;
    while(m++ < n)
    {
        next = cur->next;
        cur->next = pre->next;
        pre->next = cur;
        cur=next;
    }
    mthNode->next = next;
    return dummy.next;
}
```

Reverse Nodes in k Group

题目来源：Reverse Nodes in k-Group

> Given a linked list, reverse the nodes of a linked list k at a time and return its modified list. If the number of nodes is not a multiple of k then left-out nodes in the end should remain as it is. You may not alter the values in the nodes, only nodes itself may be changed. Only constant memory is allowed. For example, Given this linked list: 1->2->3->4->5 For k = 2, you should return: 2->1->4->3->5 For k = 3, you should return: 3->2->1->4->5

解题思路：

每遇到k个，就reverse一下。值得注意的是：

- 输入list长度是否大于k个;
- 最后不足的k个;
- reverse 中间的一段前之前得backup一下当前段(tail)的下一个, 以防止reverse内部改变了tail->next值.

```
//reverse [start, tail], return newhead
ListNode* reverse(ListNode* head, ListNode* tail)
{
    ListNode * pre = NULL;
    while(head != tail)
    {
        auto next = head->next;
        head->next = pre;
        pre = head;
        head = next;
    }
    head->next = pre; //do not forget tail
    return tail;
}
ListNode *reverseKGroup(ListNode *head, int k)
{
    if(k <= 1 || head == NULL) return head;
    ListNode * lastStart = NULL;
    ListNode * result = NULL;
    while(head)
    {
        int i = 1;
        auto start = head;
        while(i++ < k && head != NULL)
            head = head->next;
        if(head != NULL)
        {
            auto end = head;
            auto nextbak = head->next;
            auto segStart = reverse(start, end);
            if (lastStart == NULL)
                result = segStart;
            else
                lastStart->next = segStart;
            lastStart = start;
            head = nextbak;
        }else//last segment
        {
```

```
            if(lastStart)
                lastStart->next = start;
            else //node len less than k
                return start;
        }
    }
    return result;
}
```

Rotate List

题目来源：Rotate List

> Given a list, rotate the list to the right by k places, where k is non-negative. For example: Given 1->2->3->4->5->NULL and k = 2, return 4->5->1->2->3->NULL.

解题思路：

找出分隔点，注意对输入k值的处理，k可能超过链表长度。另外别忘了链表结尾要手动置为NULL。

```
int length(ListNode* head)
{
    int len = 0;
    while(head)
        ++len, head = head->next;
    return len;
}
ListNode *rotateRight(ListNode *head, int k)
{
    if(head == NULL || head->next == NULL || k == 0) return head;
    int len = length(head);
    k = k % len;
    if(k == 0) return head;
    ListNode * newHead = NULL;
    ListNode * headbak = head;
    int i = 1;
    while(i < len-k)
        head = head->next, i++;
    newHead = head->next;
    head->next = NULL; //!!! important
    ListNode *tail = newHead;
    while(tail->next)
        tail = tail->next;
    tail->next = headbak;
    return newHead;
}
```

Sort List

题目来源：Sort List

> Sort a linked list in O(n log n) time using constant space complexity.

解题思路： 可以用归并或者快排. merge就比较简单，一个一个比较然后将较小的放到上一个的后面。 用一个dummy省去第一个head的确定。

```
ListNode* merge(ListNode* head1, ListNode* head2)
{
    ListNode dummy(-1);
    ListNode * p = &dummy;
    while(head1 && head2)
    {
        if(head1->val < head2->val)
        {
            p->next = head1;
            head1 = head1->next;
        }else
        {
            p->next = head2;
            head2 = head2->next;
        }
        p = p->next;
    }
    if(head1 != NULL)
        p->next = head1;
    else
        p->next = head2;
    return dummy.next;
}
```

然后就是递归一半一半来，如下所示：

```
ListNode * mergesort(ListNode * head)
{
    if (head == NULL || head->next == NULL)
        return head;
    int len = list_len(head);
    if(len <= 2)
    {
        if(len == 1) return head;
        if(len == 2 && head->val > head->next->val)
        {
            swap(head, head->next);//or std::swap(head->val, head->next->val);
            return head;
        }
    }
    int mid = len >> 1;
    ListNode * left_tail = head;
    int i = mid-1;
    while(i)
    {
        left_tail = left_tail->next;
        i -= 1;
```

```
    }
    ListNode * right = left_tail->next;
    left_tail->next = NULL;
    ListNode * left_result = mergesort(head);
    ListNode * right_result = mergesort(right);
    return merge(left_result, right_result);
}
```

当然也可以用快慢指针去找中间的那个。

```
ListNode *sortList(ListNode *head)
    {
        if(NULL == head || NULL == head->next) return head;
        ListNode* fast = head;
        ListNode* slow = head;
        while(NULL != fast->next && NULL != fast->next->next){
            slow = slow->next;
            fast = fast->next->next;
        }
        //slow is mid, ignore fast: (last or last but one)
        ListNode * mid = slow;
        slow = slow->next; //the right part
        mid->next = NULL; // cut the left part
        ListNode * left = sortList(head);
        ListNode * right = sortList(slow);
        return merge(left, right);
    }
```

刚开始写的快排，有的testcase过不了(数字范围全是1-3那个共30293个数)，把这个testcase排除掉后，也能AC。

```
 ListNode* quick_sort_(ListNode * head)
{
    if(head == NULL || head->next == NULL)
        return head;
    if(len(head) <= 1000) return mergeSort(head);
    ListNode   lefthead = ListNode(-1);
    ListNode   righthead = ListNode(-1);
    ListNode * left = &lefthead;
    ListNode * right = &righthead;

    ListNode * pivot =  head; //or random select one
    ListNode * t = head->next;
    while(t != NULL)
    {
        if(t->val <= pivot->val)
        {
            left->next = t;
            left = left->next;
        }else
        {
            right->next = t;
            right = right->next;
        }
        t = t->next;
    }
    if(left != NULL) left->next = NULL;
    if(right != NULL) right->next = NULL;
```

```
    ListNode * left_result = quick_sort_(lefthead.next);
    ListNode * right_result = quick_sort_(righthead.next);
    ListNode * newresult_head = left_result;
    if(left_result != NULL)
    {
        while(left_result->next != NULL)
            left_result = left_result->next;
        //left_result->next is NULL
        left_result->next = pivot;
    }else
    {
        newresult_head = pivot;
    }
    pivot->next = right_result;

    return newresult_head;
}

ListNode * quick_sort(ListNode * head)
{
    if(head == NULL || head->next == NULL)
        return head;
    //if(len(head) == 30293) return countSort(head);
    return quick_sort_(head);
}
```

Swap Nodes in Pairs

题目来源：Swap Nodes in Pairs

> Given a linked list, swap every two adjacent nodes and return its head. For example, Given 1->2->3->4, you should return the list as 2->1->4->3. Your algorithm should use only constant space. You may not modify the values in the list, only nodes itself can be changed.

解题思路：

reverseKGroup

前面刚写了 reverse-nodes-in-k-group，直接调用一下，传参数2即可。

```
//reverse [start, tail], return newhead
ListNode* reverse(ListNode* head, ListNode* tail)
{
    ListNode * pre = NULL;
    while(head != tail)
    {
        auto next = head->next;
        head->next = pre;
        pre = head;
        head = next;
    }
    head->next = pre; //do not forget tail
    return tail;
}
ListNode *reverseKGroup(ListNode *head, int k)
{
    if(k <= 1 || head == NULL) return head;
    ListNode * lastStart = NULL;
    ListNode * result = NULL;
    while(head)
    {
        int i = 1;
        auto start = head;
        while(i++ < k && head != NULL)
            head = head->next;
        if(head != NULL)
        {
            auto end = head;
            auto nextbak = head->next;
            auto segStart = reverse(start, end);
            if (lastStart == NULL)
                result = segStart;
            else
                lastStart->next = segStart;
            lastStart = start;
            head = nextbak;
        }else//last segment
        {
            if(lastStart)
                lastStart->next = start;
            else //node len less than k
                return start;
        }
```

```
    }
    return result;
}
ListNode *swapPairs(ListNode *head)
{
    return reverseKGroup(head, 2);
}
```

递归版本

> next(p1->p2->p3->p4...) =
p1->next = next(p3->p4...); p2->next = p1; return p2;

代码如下:

```
ListNode* next(ListNode*p1, ListNode* p2)
{
    if (p1 == NULL) return NULL;
    if (p1 != NULL && p2 == NULL) return p1;
    p1->next = next(p2->next, p2->next ? p2->next->next : NULL);
    p2->next = p1;
    return p2;
}
ListNode *swapPairs(ListNode *head)
{
    if(head == NULL || head->next == NULL) return head;
    return next(head, head->next);
}
```

迭代版本

```
ListNode *swapPairs(ListNode *head)
{
    if(head == NULL || head->next == NULL) return head;
    //1     2   3       4 5
    //1st   2nd next
    ListNode dummy(-1);
    ListNode *pre = &dummy;
    ListNode *first = head, *second = NULL, *next = NULL;
    while(first && first->next)
    {
        second = first->next;
        next = second->next;

        pre->next = second;
        second->next = first;
        first->next = next;

        pre = first;
        first = next;
    }
    return dummy.next;
}
```

binary tree, 二叉树相关

1. Balanced Binary Tree 题解
2. Binary Tree Inorder Traversal 题解
3. Binary Tree Level Order Traversal 题解
4. Binary Tree Level Order Traversal II 题解
5. Binary Tree Maximum Path Sum 题解
6. Binary Tree Postorder Traversal 题解
7. Binary Tree Preorder Traversal 题解
8. Binary Tree Zigzag Level Order Traversal 题解
9. Construct Binary Tree from Inorder and Postorder Traversal 题解
10. Construct Binary Tree from Preorder and Inorder Traversal 题解
11. Convert Sorted List to Binary Search Tree 题解
12. Flatten Binary Tree to Linked List 题解
13. Maximum Depth of Binary Tree 题解
14. Minimum Depth of Binary Tree 题解
15. Path Sum 题解
16. Path Sum II 题解
17. Populating Next Right Pointers in Each Node 题解
18. Populating Next Right Pointers in Each Node II 题解
19. Recover Binary Search Tree 题解
20. Same Tree 题解
21. Sum Root to Leaf Numbers 题解
22. Symmetric Tree 题解
23. Unique Binary Search Trees 题解
24. Unique Binary Search Trees II 题解
25. Validate Binary Search Tree 题解

Balanced Binary Tree

题目来源：Balanced Binary Tree

>

```
Given a binary tree, determine if it is height-balanced.

For this problem, a height-balanced binary tree is defined as a binary tree in which the
```

解题思路：

递归.

```
int depth(TreeNode * root){
    if(root == NULL) return 0;
    return std::max(depth(root->left),depth(root->right))+1;
}
bool isBalanced(TreeNode *root)
{
    if(root == NULL) return true;
    int left = depth(root->left);
    int right = depth(root->right);
    if (abs(left - right) > 1) return false;
    return isBalanced(root->left) && isBalanced(root->right);
}
```

其实递归过程中好多都算重复了。 可以将判断过程融合在算高度的方法里。

```
int depth(TreeNode * root){
    if(root == NULL) return 0;
    int left = depth(root->left);
    int right = depth(root->right);
    if(left == -1 || right == -1 || abs(left-right) > 1)
        return -1;
    return std::max(left, right)+1;
}
bool isBalanced(TreeNode *root)
{
    if(root == NULL) return true;
    return depth(root) >= 0;
}
```

Binary Tree Inorder Traversal

题目来源： Binary Tree Inorder Traversal

> Given a binary tree, return the inorder traversal of its nodes' values. For example: Given binary tree {1,#,2,3}, 1 \ 2 / 3 return [1,3,2]. Note: Recursive solution is trivial, could you do it iteratively?

解题思路：

思路一: 直接递归(略)

思路二: 用stack.

```
vector<int> inorderNormal(TreeNode * root)
{
    vector<int> result;
    stack<TreeNode*> stacks;
    TreeNode * cur = root;
    while(true)
    {
        if(cur != NULL)
        {
            stacks.push(cur);
            cur = cur->left;
        }else
        {
            if(stacks.empty()) break;
            cur = stacks.top(); stacks.pop();
            result.push_back(cur->val);
            if(stacks.empty())
                break;
            cur = cur->right;

        }
    }
    return move(result);
}
vector<int> inorderTraversal(TreeNode *root)
{
    if(root == NULL) return vector<int>();
    return inorderNormal(root);
}
```

思路三： Morris遍历. `O(1)` 空间 + `O(n)` 时间

利用线索二叉树, 利用叶子节点的空指针指向前驱后继来记住状态。算法仍参考Morris Traversal，里面讲了详细的案例。

具体算法如下:

```
步骤：
1. 如果当前节点的左孩子为空，则输出当前节点并将其右孩子作为当前节点。
2. 如果当前节点的左孩子不为空，在当前节点的左子树中找到当前节点在中序遍历下的前驱节点。
```

a）如果前驱节点的右孩子为空，将它的右孩子设置为当前节点。当前节点更新为当前节点的左孩子。
b）如果前驱节点的右孩子为当前节点，将它的右孩子重新设为空（恢复树的形状）。输出当前节点。当前节点更新为
3．重复以上1、2直到当前节点为空。

```
vector<int> inorderMorris(TreeNode * root)
{
    vector<int> result;
    TreeNode * cur = root;
    while(NULL != cur)
    {
        if(cur->left == NULL)
        {
            result.push_back(cur->val);
            cur = cur->right;
        }else
        {
            auto pre = cur->left;
            while(pre->right != NULL && pre->right != cur)
                pre = pre->right;
            if(pre->right == NULL)
            {
                pre->right = cur;
                cur = cur->left;
            }else //reset
            {
                pre->right = NULL;
                result.push_back(cur->val);
                cur = cur->right;
            }
        }
    }
    return move(result);
}
```

Binary Tree Level Order Traversal

题目来源：Binary Tree Level Order Traversal

>

```
Given a binary tree, return the level order traversal of its nodes' values. (ie, from lef

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]
]
```

解题思路：

常规方法, 两个queue交替

```
    vector<vector<int> > levelOrder0(TreeNode *root)
    {
        vector<vector<int> > result;
        queue<TreeNode*> q1, q2;
        q1.push(root);
        while(! q1.empty())
        {
            vector<int> level;
            while(!q1.empty())
            {
                auto node = q1.front(); q1.pop();
                level.push_back(node->val);
                if(node->left) q2.push(node->left);
                if(node->right) q2.push(node->right);
            }
            std::swap(q1, q2);
            result.push_back(level);
        }
        return move(result);
    }
```

单queue+隔板

前面word ladder ii就提到过bfs，用隔板将各层之间隔离出来。只用一个queue就能知道某层是否已经遍历完毕。

```
vector<vector<int> > levelOrder(TreeNode *root)
{
    vector<vector<int> > result;
    queue<TreeNode*> q;
    q.push(root);
    TreeNode * split = nullptr;
    while(! q.empty())
    {
        q.push(split);
        vector<int> level;
        while(q.front() != split)
        {
            auto node = q.front(); q.pop();
            level.push_back(node->val);
            if(node->left) q.push(node->left);
            if(node->right) q.push(node->right);
        }
        result.push_back(level);
        q.pop();//pop split;
    }
    return move(result);
}
```

递归

递归写起来就是简单。

```
void levelOrderRecusion(TreeNode *root, int level, vector<vector<int> > &result)
{
    if(level >= result.size())
        result.push_back(vector<int>());
    result[level].push_back(root->val);
    if(root->left)
        levelOrderRecusion(root->left, level+1, result);
    if(root->right)
        levelOrderRecusion(root->right, level+1, result);
}
vector<vector<int> > levelOrder(TreeNode *root)
{
    vector<vector<int> > result;
    if(root == NULL) return result;
    levelOrderRecusion(root, 0, result);
    return move(result);
}
```

Binary Tree Level Order Traversal II

题目来源：Binary Tree Level Order Traversal II

>

```
Given a binary tree, return the bottom-up level order traversal of its nodes' values. (ie

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]
]
```

解题思路：

跟前一题Binary Tree Level Order Traversal唯一的区别就是这个将最后结果reverse一下。 这里就只列了其中一种代码了。

常规方法, 两个queue交替

参见Binary Tree Level Order Traversal。

单queue+隔板

前面word ladder ii就提到过bfs，用隔板将各层之间隔离出来。只用一个queue就能知道某层是否已经遍历完毕。

```
     vector<vector<int> > levelOrderBottom(TreeNode *root)
    {
        vector<vector<int> > result;
        if(root == NULL) return result;
        queue<TreeNode*> q;
        q.push(root);
        while(! q.empty())
        {
            q.push(NULL);
            vector<int> level;
            while(q.front() != NULL)
            {
                auto node = q.front(); q.pop();
                level.push_back(node->val);
                if(node->left) q.push(node->left);
                if(node->right) q.push(node->right);
```

```
            }
            result.push_back(level);
            q.pop();//pop NULL
        }
        std::reverse(result.begin(), result.end());
        return move(result);
    }
```

递归

参见Binary Tree Level Order Traversal。

Binary Tree Maximum Path Sum

题目来源：Binary Tree Maximum Path Sum

>

```
Given a binary tree, find the maximum path sum.
The path may start and end at any node in the tree.
For example:
Given the below binary tree,
  1
 / \
2   3
Return 6.
```

解题思路：

path路径能以任意节点开头或结尾。注意maxPathSum(root) != max{ maxPathSum(root.left), maxPathSum(root.right), maxPathSum(root.left) + maxPathSum(root.right) + root.val }， 右/左 子树的最大结果很有可能不能跟当前节点连在一起。

```
    // ended/started with root, the max path
    int subMax(TreeNode *root, int &global_max)
    {
        if(root == NULL) return 0;
        int l_max = 0;
        int r_max = 0;
        l_max = std::max(subMax(root->left, global_max), 0); //if 0, not choose left chil
        r_max = std::max(subMax(root->right, global_max), 0);
        int max_ = root->val + l_max + r_max; //global_max: connected current node
        if(global_max < max_)
            global_max = max_;
        return std::max(0, std::max(l_max, r_max) + root->val);
    }
    //Attention, for the max(root.right) may not connected with root
    //maxPathSum(root) != max{ maxPathSum(root.left), maxPathSum(root.right), maxPathSum(

    int maxPathSum(TreeNode *root)
    {
        int global_max = INT_MIN;
        subMax(root, global_max);
        return global_max;
    }
```

Binary Tree Postorder Traversal

题目来源： Binary Tree Postorder Traversal

> Given a binary tree, return the postorder traversal of its nodes' values. For example: Given binary tree {1,#,2,3}, 1 \ 2 / 3 return [3,2,1]. Note: Recursive solution is trivial, could you do it iteratively?

解题思路：

下文用了5种方法实现了对二叉树进行后序遍历。

思路一: 直接递归

```
void postRecursion(TreeNode * root, vector<int> &path)
{
    if(root == NULL) return;
    if(root->left != NULL)
        postRecursion(root->left, path);
    if(root->right != NULL)
        postRecursion(root->right, path);
    path.push_back(root->val);
}

vector<int> postorderTraversal(TreeNode *root)
{
    vector<int> result;
    postRecursion(root, result);
    return move(result);
}
```

思路二: 非递归. 仿造先序,因为先序的非递归很好写.

(来自寝室哥们ZZ大神的思路)

```
先序: `中左右`
后序: `左右中`
```

发现 先序.reverse = `右左中` 将 `右左` 交换就得到 `左右中` . 即将原来的先序变通下就有了下面的算法.

```
vector<int> postFakePre(TreeNode * root)
{
    stack<TreeNode*> stacks;
    stacks.push(root);
    vector<int> result;
    while(! stacks.empty())
    {
        auto node = stacks.top();
        result.push_back(node->val); stacks.pop();
        //pre: push right then left,
        //fake pre: push left, then right
```

```
            if(node->left != NULL)
                stacks.push(node->left);
            if(node->right != NULL)
                stacks.push(node->right);
        }
        std::reverse(result.begin(), result.end());
        return move(result);
    }
```

思路三：传统方法

用一个指针last记录上一次访问的节点来区分右孩纸是否已经访问过了该回归到父节点。代码如下

```
    vector<int> postNormal(TreeNode * root)
    {
        stack<TreeNode*> stacks;
        vector<int> result;
        TreeNode* last = NULL, * cur = root;
        while(true)
        {
            if(cur != NULL)//go to left most
            {
                stacks.push(cur);
                cur = cur->left;
            }else //leaf node
            {
                auto peak = stacks.top();
                if(peak->right != NULL && peak->right != last) // right child has not bee
                {
                    cur = peak->right;
                }else{
                    result.push_back(peak->val);
                    stacks.pop();
                    last = peak;
                    if(stacks.empty())
                        break;
                }
            }
        }
        return move(result);
    }
```

思路四：改进的传统方法

下面的方法来自网络(但忘了具体出处了). 比较好理解。

> 要保证根结点在左孩子和右孩子访问之后才能访问，因此对于任一结点P，先将其入栈。 如果P不存在左孩子和右孩子，则可以直接访问它；或者P存在左孩子或者右孩子，但是其左孩子和右孩子都已被访问过了，则同样可以直接访问该结点。 若非上述两种情况，则将P的右孩子和左孩子依次入栈，这样就保证了每次取栈顶元素的时候，左孩子在右孩子前面被访问，左孩子和右孩子都在根结点前面被访问。

代码如下:

```
vector<int> postNormalBetter(TreeNode * root)
{
    stack<TreeNode*> stacks;
    vector<int> result;
    TreeNode* last = NULL;
    stacks.push(root);
    while(! stacks.empty())
    {
        auto cur = stacks.top();
        if ( (NULL == cur->left && NULL == cur->right) //is leaf node or
            || (last && (last == cur->left || last == cur->right)) )// children have
            {
                result.push_back(cur->val);
                stacks.pop();
                last = cur;
            }
        else//has children, push right then left
        {
            if(cur->right)
                stacks.push(cur->right);
            if(cur->left)
                stacks.push(cur->left);
        }
    }
    return move(result);
}
```

思路五：Morris遍历

以上都用了 `O(n)` 的时间+ `O(n)` 的空间. 还有就是传说中的利用了线索二叉树 `O(1)` 的空间的 `Morris遍历算法` . 主要就是利用了叶子节点的孩纸指针, 指向后继节点记录回退的位置。 这篇文章讲得清晰，我就不重复了。 将后续遍历的算法copy过来.

> 后序遍历稍显复杂，需要建立一个临时节点dump，令其左孩子是root。并且还需要一个子过程，就是倒序输出某两个节点之间路径上的各个节点。 步骤： 当前节点设置为临时节点dump。

```
1. 如果当前节点的左孩子为空，则将其右孩子作为当前节点。
2. 如果当前节点的左孩子不为空，在当前节点的左子树中找到当前节点在中序遍历下的前驱节点。
   a) 如果前驱节点的右孩子为空，将它的右孩子设置为当前节点。当前节点更新为当前节点的左孩子。
   b) 如果前驱节点的右孩子为当前节点，将它的右孩子重新设为空。倒序输出从当前节点的左孩子到该前驱节点这条路
3. 重复以上1、2直到当前节点为空。
```

```
vector<int> reverse(TreeNode * from, TreeNode * to)
{
    vector<int> path;
    while(true)
    {
        path.push_back(from->val);
        if(from == to)
            break;
        from = from->right;
    }
```

```
        std::reverse(path.begin(), path.end());
    return move(path);
    }
    vector<int> postMorris(TreeNode * root)
    {
        TreeNode dump(-1);
        dump.left = root;
        TreeNode * cur = &dump;
        vector<int> result;
        while(cur != NULL)
        {
            if(NULL == cur->left) //1
                cur = cur->right;
            else//2
            {
                auto pre = cur->left; //pre: left child's right most
                while(pre->right != NULL && pre->right != cur)
                    pre = pre->right;
                if(pre->right == NULL)//2.a
                {
                    pre->right = cur;
                    cur = cur->left;
                }else // 2.b
                {
                    pre->right = NULL;
                    auto path = reverse(cur->left, pre);
                    std::copy(path.begin(), path.end(), back_insert(result));
                    cur = cur->right;
                }
            }
        }
        return move(result);
    }
```

Binary Tree Preorder Traversal

题目来源： Binary Tree Preorder Traversal

> Given a binary tree, return the preorder traversal of its nodes' values. For example: Given binary tree {1,#,2,3}, 1 \ 2 / 3 return [1,2,3]. Note: Recursive solution is trivial, could you do it iteratively?

解题思路：

思路一: 直接递归(略)

思路二: 用stack.

```
vector<int> preNormal(TreeNode * root)
{
    stack<TreeNode*> stacks;
    stacks.push(root);
    vector<int> result;
    while(! stacks.empty())
    {
        auto node = stacks.top();
        result.push_back(node->val); stacks.pop();
        if(node->right != NULL)
            stacks.push(node->right);
        if(node->left != NULL)
            stacks.push(node->left);
    }
    return move(result);
}
```

思路三： Morris遍历. `O(1)` 空间 + `O(n)` 时间

利用线索二叉树, 利用叶子节点的空指针指向前驱后继来记住状态。算法仍参考Morris Traversal，里面讲了详细的案例。

具体算法如下:

```
步骤：
1. 如果当前节点的左孩子为空，则输出当前节点并将其右孩子作为当前节点。
2. 如果当前节点的左孩子不为空，在当前节点的左子树中找到当前节点在中序遍历下的前驱节点。
   a)如果前驱节点的右孩子为空，将它的右孩子设置为当前节点。输出当前节点（**在这里输出，这是与中序遍历唯一
   b)如果前驱节点的右孩子为当前节点，将它的右孩子重新设为空。当前节点更新为当前节点的右孩子。
3. 重复以上1、2直到当前节点为空。
```

```
vector<int> preMorris(TreeNode * root)
{
    vector<int> result;
    TreeNode * cur = root;
```

```
        while(cur != NULL)
        {
            if(cur->left == NULL)
            {
                result.push_back(cur->val);
                cur = cur->right;
            }else
            {
                auto pre = cur->left;
                while(pre->right != NULL && pre->right != cur)
                    pre = pre->right;
                if(pre->right == NULL)//2.a
                {
                    pre->right = cur;
                    result.push_back(cur->val);
                    cur = cur->left;
                }else // 2.b
                {
                    pre->right = NULL;//reset
                    cur = cur->right;
                }
            }
        }
        return move(result);
    }
```

Binary Tree Zigzag Level Order Traversal

题目来源：Binary Tree Zigzag Level Order Traversal II

>

```
Given a binary tree, return the zigzag level order traversal of its nodes' values. (ie, f
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]
]
```

解题思路：

跟前面的题Binary Tree Level Order Traversal 以及Binary Tree Level Order Traversal II。 区别就是这个将第偶数层的结果reverse一下。 这里就只列了其中一种代码了。

常规方法, 两个queue交替

参见Binary Tree Level Order Traversal。

单queue+隔板

前面word ladder ii就提到过bfs，用隔板将各层之间隔离出来。只用一个queue就能知道某层是否已经遍历完毕。

```
     vector<vector<int> > zigzagLevelOrder(TreeNode *root)
    {
        vector<vector<int> > result;
        if(root == NULL) return result;
        queue<TreeNode*> q;
        q.push(root);
        int i = 0;
        while(! q.empty())
        {
            q.push(NULL);
            vector<int> level;
            while(q.front() != NULL)
            {
                auto node = q.front(); q.pop();
                level.push_back(node->val);
                if(node->left) q.push(node->left);
                if(node->right) q.push(node->right);
```

```
            }
            if(i++ & 0x1)
                std::reverse(level.begin(), level.end());
            result.push_back(level);
            q.pop();//pop NULL
        }
        return move(result);
    }
```

递归

参见Binary Tree Level Order Traversal。

Construct Binary Tree from Inorder and Postorder Traversal

题目来源：Construct Binary Tree from Inorder and Postorder Traversal

> Given inorder and postorder traversal of a tree, construct the binary tree. Note: You may assume that duplicates do not exist in the tree.

解题思路：

仍跟上题一样construct-binary-tree-from-preorder-and-inorder-traversal,同样假设输入合理。 若输入不合法，可参考树重建进行处理。

```
中：左  中  右
后：左  右  中
在中序中查找后序的最后一个得index为left
left_len = left - in_start
对左子树而言：
中序左: [in_start, left - 1]
后序左: [post_start, post_start + left_len -1]
同理 右子树
中序右: [left+1, in_end]
后序右: [post_start + left_len, post_end-1]
```

```
    TreeNode * buildRecursion(vector<int> &postorder, vector<int> &inorder,
                      int postStart, int postEnd, int inStart, int inEnd)
        {
            if(postStart > postEnd) return NULL;
            if(postEnd == postStart)
                return new TreeNode(postorder[postStart]);
            TreeNode * root = new TreeNode(postorder[postEnd]);
            int index = inStart;
            while(inorder[index] != root->val) //assuming can get, or else should check i
                index++;
            int leftLen = index - inStart;
            int rightLen = inEnd - index;
            root->left = buildRecursion(postorder, inorder, postStart, postStart+leftLen-
            root->right = buildRecursion(postorder, inorder, postStart+leftLen, postEnd-1
            return root;
        }
    TreeNode *buildTree(vector<int> &inorder, vector<int> &postorder)
    {
        return    buildRecursion(postorder, inorder, 0, postorder.size()-1, 0, inorder.si
    }
```

Construct Binary Tree from Preorder and Inorder Traversal

题目来源：Construct Binary Tree from Preorder and Inorder Traversal

>

```
Given preorder and inorder traversal of a tree, construct the binary tree.
Note:
You may assume that duplicates do not exist in the tree.
```

解题思路：

递归比较好～但注意下标别搞错. 这里假设输入一定可以构造合理的二叉树。所以没有考虑一些异常情况。这里树重建有考虑不能成功构造出二叉树的情况。

```
先序：中     左     右
中序：左     中     右
先序左：[pre_start+1, pre_start + 1 + left_len - 1]
中序左：[in_start, left-1]

先序右：[pre_start+left_len+1, pre_end]
中序右：[left+1, in_end]
```

```
    //assuming always can get right result
    TreeNode * buildRecursion(vector<int> &preorder, vector<int> &inorder,
                              int preStart, int preEnd, int inStart, int inEnd)
    {
        if(preStart > preEnd) return NULL;
        if(preEnd == preStart) // assert(inStart == inEnd)
            return new TreeNode(preorder[preStart]);
        TreeNode * root = new TreeNode(preorder[preStart]);
        int index = inStart;
        while(inorder[index] != root->val) //assuming can get, or else should check if in
            index++;
        int leftLen = index - inStart;
        int rightLen = inEnd - index;
        root->left = buildRecursion(preorder, inorder, preStart+1, preStart+leftLen, inSt
        root->right = buildRecursion(preorder, inorder, preStart+leftLen+1, preEnd, index
        return root;
    }
    TreeNode *buildTree(vector<int> &preorder, vector<int> &inorder)
    {
        return buildRecursion(preorder, inorder, 0, preorder.size()-1, 0, inorder.size()-
    }
```

Convert Sorted List to Binary Search Tree

题目来源：Convert Sorted List to Binary Search Tree

> Given a singly linked list where elements are sorted in ascending order, convert it to a height balanced BST.

解题思路：

tricky 方法, 另外取O(n)空间

偷懒方法，另外取另外取O(n)空间把list的数据取出来放入数组，然后跟题目一样用数组的方式去做。代码就略过了。虽然不是出题者的本意～ 但..... 你咬我呀.

O(nlogn)时间

每次用O(len/2)的时间去把中间的节点找出来。然后跟数组一样的方式解决。时间复杂度为O(nlogn).中途找mid不跟数组一样O(1).

```
int length(ListNode * head)
{
    int len = 0;
    while(head)
    {
        ++len;
        head = head->next;
    }
    return len;
}
TreeNode * convert(ListNode * head, int len)
{
    if(head == NULL || len == 0) return NULL;
    if(len == 1) return new TreeNode(head->val);
    int mid = len>>1;
    ListNode * pre = head;
    int i = mid;
    while(--i)
        pre=pre->next;
    int leftlen = mid-1; int rightlen = len-mid;//even
    if(len & 0x1)
    {
        pre = pre->next;
        leftlen = mid;
        rightlen = len-mid-1;
    }
    auto root = new TreeNode(pre->val);
    root->left = convert(head, leftlen);
    root->right = convert(pre->next, rightlen);
}

TreeNode *sortedListToBST(ListNode *head)
{
    int len = length(head);
    return convert(head, len);
```

```
    }
```

从底至上构造Tree, 递归(得忽略递归调用的时间/空间消耗)调用，递归中传同一个链表，链表不停往前走，通过下标关系来控制左右子树。进入递归时链表指向头节点，结束递归时，链表指向尾节点的next。

下面代码中，每次递归调用开始时，节点指针都指向区间第一个，结束时节点的指针指向区间末尾的后一个。每次递归调用时，分成左右两部分，左边构造完时，正好指针指向mid，创建一下root，继续构造右部分子树。ref

```
int length(ListNode * head)
{
    int len = 0;
    while(head)
    {
        ++len;
        head = head->next;
    }
    return len;
}
TreeNode * convert(ListNode * &head, int start, int end)
{
    assert(start <= end);
    if(start == end) {
        auto result = new TreeNode(head->val);
        head=head->next;
        return result;
    }
    TreeNode* left = NULL;
    int mid = start + ((end-start)>>1);
    if(start <= mid-1)
        left = convert(head, start, mid-1);
    TreeNode * root = new TreeNode(head->val);
    head = head->next;
    root->left = left;
    if(mid+1 <= end)
        root->right = convert(head, mid+1, end);
    return root;
}

TreeNode *sortedListToBST(ListNode *head)
{
    if(head == NULL) return NULL;
    int len = length(head);
    return convert(head, 0, len-1);
}
```

或许 convert这样写更简洁。

```
TreeNode * convert(ListNode * &head, int start, int end)
{
    if(start > end) return NULL;
    int mid = start + ((end-start)>>1);
    TreeNode* left = convert(head, start, mid-1);
    TreeNode * root = new TreeNode(head->val);
    head = head->next;
```

```
        root->left = left;
        root->right = convert(head, mid+1, end);
        return root;
    }
```

Flatten Binary Tree to Linked List

题目来源：Flatten Binary Tree to Linked List

>

```
Given a binary tree, flatten it to a linked list in-place.
For example,
Given

         1
        / \
       2   5
      / \   \
     3   4   6
The flattened tree should look like:
   1
    \
     2
      \
       3
        \
         4
          \
           5
            \
             6
Hints: If you notice carefully in the flattened tree, each node's right child points to t
```

解题思路：

先序遍历，可以遍历完后再连接，也可以在遍历过程中连接。

```
    void flatten(TreeNode *root)
    {
        if(root == NULL) return;
        vector<TreeNode*> prefs;
        stack<TreeNode*> stacks;
        stacks.push(root);
        while(! stacks.empty())
        {
            auto node = stacks.top(); stacks.pop();
            prefs.push_back(node);
            if(node->right)
                stacks.push(node->right);
            if(node->left)
                stacks.push(node->left);
        }
        for(int i = 0; i < prefs.size()-1; i++)
        {
            prefs[i]->left = NULL;
            prefs[i]->right = prefs[i+1];
        }
    }
```

或者

```
void flatten(TreeNode *root)
{
    if(root == NULL) return;
    stack<TreeNode*> stacks;
    stacks.push(root);
    TreeNode tmp(-1);
    TreeNode* last = &tmp;
    while(! stacks.empty())
    {
        auto node = stacks.top(); stacks.pop();
        last->right = node;
        last->left = NULL;
        last = node;
        if(node->right)
            stacks.push(node->right);
        if(node->left)
            stacks.push(node->left);
    }
}
```

或者用其他binary tree pre traverse 的方法都行。

Maximum Depth of Binary Tree

题目来源：Maximum Depth of Binary Tree

>

```
Given a binary tree, find its maximum depth.
The maximum depth is the number of nodes along the longest path from the root node down t
```

解题思路：

跟Minimum Depth of Binary Tree思路一样，这个更简答。

```
    int depth(TreeNode* root){
        if(root->left == NULL && root->right == NULL) return 1;
        int left = 0; int right = 0;
        if(root->left) left = depth(root->left);
        if(root->right) right = depth(root->right);
        return std::max(left, right) + 1;
    }
    int maxDepth(TreeNode *root)
    {
        if(root == NULL) return 0;
        return depth(root);
    }
```

Minimum Depth of Binary Tree

题目来源：Minimum Depth of Binary Tree

>

```
Given a binary tree, find its minimum depth.
The minimum depth is the number of nodes along the shortest path from the root
node down to the nearest leaf node.
```

解题思路：

穷举所有路径即可。

```
    void depthRecursion(TreeNode *root, int curDepth, int &minDepth){
        if(root->left == NULL && root->right == NULL)
        {
            minDepth = std::min(minDepth, curDepth+1);
            return;
        }
        if(root->left) depthRecursion(root->left, curDepth+1, minDepth);
        if(root->right) depthRecursion(root->right, curDepth+1, minDepth);
    }
    int minDepth(TreeNode *root)
    {
        if(root == NULL) return 0;
        int result = INT_MAX;
        depthRecursion(root, 0, result);
        return result;
    }
```

或者这样，递归时把是否有兄弟节点传进去。ref.

```
    int minDepth2(TreeNode* node, bool hasBrother)
    {
        if(node == NULL) return hasBrother ? INT_MAX : 0;
        return std::min(minDepth2(node->left, node->right != NULL),
                        minDepth2(node->right, node->left != NULL))+1;
    }
    int minDepth(TreeNode *root)
    {
        return minDepth2(root, false);
    }
```

Path Sum

题目来源：Path Sum

> Given a binary tree and a sum, determine if the tree has a root-to-leaf path such that adding up all the values along the path equals the given sum. For example: Given the below binary tree and sum = 22, 5 / \ 4 8 / / \ 11 13 4 / \ \ 7 2 1 return true, as there exist a root-to-leaf path 5->4->11->2 which sum is 22.

解题思路：

递归最简单了。

```
bool solveRecusive(TreeNode *root, int sum)
{
    int v = root->val;
    if(root->left == NULL && root->right == NULL)
        return v == sum;
    return (root->left && solveRecusive(root->left, sum-v)) ||
           (root->right && solveRecusive(root->right, sum-v));
}
bool hasPathSum(TreeNode *root, int sum)
{
    if(root == NULL) return false;//test case rule
    return solveRecusive(root, sum);
}
```

Path Sum II

题目来源：Path Sum II

>

```
Given a binary tree and a sum, find all root-to-leaf paths where each path's sum equals t
For example:
Given the below binary tree and sum = 22,
              5
             / \
            4   8
           /   / \
          11  13  4
         /  \    / \
        7    2  5   1
return
[
   [5,4,11,2],
   [5,8,4,5]
]
```

解题思路：

跟Path Sum思路一样，不过这题把路径存起来。

```
    void search(vector<int> &path, vector<vector<int> >&result, TreeNode* node, int targe
    {
        assert(node != NULL);
        int v = node->val;// Attention 0
        if(node->left == NULL && node->right == NULL)
        {
            if(target == v)
            {
                path.push_back(v); //Attention 1
                result.push_back(path);
                path.pop_back();
            }
            return;
        }
        if(node->left)
        {
            path.push_back(v); //Attention 1
            search(path, result, node->left, target - v);
            path.pop_back();
        }
        if(node->right)
        {
            path.push_back(v); //Attention 1
            search(path, result, node->right, target - v);
            path.pop_back();
        }
    }
    vector<vector<int> > pathSum(TreeNode *root, int sum)
```

```
    {
        vector<vector<int> > result;
        if(root == NULL) return result;
        vector<int> path;
        search(path, result, root, sum);
        return move(result);
    }
```

注意别被code中的表象所迷惑，将`//Attention 1`的代码提取到`//Attention 0`处。 path先后push_back会反映到递归调用里面去的。

不然应该下面这样写。

```
    void search2(vector<int> &path, vector<vector<int> >&result, TreeNode* node, int targ
    {
        if(node == NULL) return;//cannot check if target is 0 then putinto result, for th
        int v = node->val;
        path.push_back(v);
        if(node->left == NULL && node->right == NULL)
        {
            if(target == v)
                result.push_back(path);
        }
        search2(path, result, node->left, target - v);
        search2(path, result, node->right, target - v);
        path.pop_back();
    }
```

Populating Next Right Pointers in Each Node

题目来源：Populating Next Right Pointers in Each Node

>

```
Given a binary tree

struct TreeLinkNode {
  TreeLinkNode *left;
  TreeLinkNode *right;
  TreeLinkNode *next;
}
Populate each next pointer to point to its next right node. If there is no next right nod

Initially, all next pointers are set to NULL.

Note:

You may only use constant extra space.
You may assume that it is a perfect binary tree (ie, all leaves are at the same level, an
For example,
Given the following perfect binary tree,
         1
       /  \
      2    3
     / \  / \
    4  5  6  7
After calling your function, the tree should look like:
         1 -> NULL
       /  \
      2 -> 3 -> NULL
     / \  / \
    4->5->6->7 -> NULL
```

解题思路：

1种方法是领取数组把每一层存起来，然后连接。不过非常数空间。 另外就是常数空间解法。

```
void connect(TreeLinkNode *root)
{
   while(root)
   {
       auto left = root->left;
       while(root) //go right
       {
           if(root->left && root->right)
               root->left->next = root->right;
           if(root->right && root->next)
               root->right->next = root->next->left;
           root = root->next;
       }
       root = left;//go down
   }
```

```
}
```

还有的写法就是 Populating Next Right Pointers in Each Node II了，完全可以用于解这个题目。

populating next right pointers in each node ii

题目来源：Populating Next Right Pointers in Each Node II

>

```
Follow up for problem "Populating Next Right Pointers in Each Node".
What if the given tree could be any binary tree? Would your previous solution still work?

Note:
You may only use constant extra space.
For example,
Given the following binary tree,
         1
       /  \
      2    3
     / \    \
    4   5    7
After calling your function, the tree should look like:
         1 -> NULL
       /  \
      2 -> 3 -> NULL
     / \    \
    4-> 5 -> 7 -> NULL
```

解题思路：

跟 Populating Next Right Pointers in Each Node思路一致，值得注意的是

- 在填充下一层next时，当前层的next要找完。//Attention 0
- 往下走时，可能前面的节点没有孩纸节点，也要找完同一层的节点是否存在孩纸节点。//Attention 1

```
void connect(TreeLinkNode *root)
{
   while(root)
   {
       auto down = root->left;
       if(NULL == down) down = root->right;
       if(NULL == down) down = root->next;////Attention 1: no child of root, but maybe th
       while(root)//go right
       {
           auto next = root->next; //go right to find next's children
           if(root->left)
           {
               if(root->right)
                   root->left->next = root->right;
               else{
                   while (next && (next->left == NULL && next->right == NULL)) //Attentio
                       next = next->next;
                   if(next != NULL)
                       root->left->next = next->left == NULL ? next->right : next->left;
               }
           }
           if(root->right)
           {
```

```
                while (next && (next->left == NULL && next->right == NULL))
                    next = next->next;
                if(next != NULL)
                    root->right->next = next->left == NULL ? next->right : next->left;
            }
            root = next;
        }
        root = down;//go down
    }
}
```

上面代码确实比较丑陋～ 其实着眼于同一层之前通过pre来记录同一层的前一个节点的话，代码就好看得多。 leetcode-cpp

```
void connect(TreeLinkNode *root)
{
    while(root)
    {
        TreeLinkNode * next = NULL;//next level
        TreeLinkNode * pre = NULL;
        while(root)//go right
        {
            if(next == NULL) next = root->left ? root->left : root->right;
            if(root->left)
            {
                if(pre) pre->next = root->left;
                pre = root->left;
            }
            if(root->right)
            {
                if(pre) pre->next = root->right;
                pre = root->right;
            }
            root = root->next;
        }
        root = next;//goto next level
    }
}
```

Recover Binary Search Tree

题目来源：Recover Binary Search Tree

> Two elements of a binary search tree (BST) are swapped by mistake. Recover the tree without changing its structure. Note: A solution using O(n) space is pretty straight forward. Could you devise a constant space solution?

解题思路：

求得中序遍历结果,再两边向中间扫描

O(2*n) 空间解法～ 直接中序遍历，然后分别从前往后、从后往前找非升序、非降序的两个node，交换其值即可。

```
void inorder1(vector<TreeNode*> &result, TreeNode* root)
{
    if(root->left) inorder1(result, root->left);
    result.push_back(root);
    if(root->right) inorder1(result, root->right);
}

void recoverTree1(TreeNode *root)
{
    if (root == NULL) return;
    vector<TreeNode*> inorder_result;
    inorder1(inorder_result, root);
    TreeNode* firstWrong = NULL, *secondWrong = NULL;
    vector<TreeNode*>::iterator it;
    for(it = inorder_result.begin(); it != inorder_result.end()-1; it++)
    {
        if((*it)->val >= (*(it+1))->val)
        {
            firstWrong = *it;
            break;
        }
    }
    for(auto it2 = inorder_result.end()-1; it2 != it; it2--)
    {
        if((*it2)->val <= (*(it2-1))->val)
        {
            secondWrong = *it2;
            break;
        }
    }
    //swap
    int tmp = firstWrong->val;
    firstWrong->val = secondWrong->val;
    secondWrong->val = tmp;
}
```

中序遍历一边遍历，一边扫描。

当两个节点都找到后，即可退出中序遍历流程。

```
void recoverTree(TreeNode *root)
{
    if(root == NULL) return;
    stack<TreeNode*> stacks;
    TreeNode tmp(INT_MIN);
    TreeNode * last = &tmp;
    TreeNode * node1 = NULL, *node2 = NULL;
    TreeNode * node = root;
    while(true)
    {
        if(node)
        {
            stacks.push(node);
            node = node->left;
        }
        else
        {
            if(stacks.empty()) break;
            node = stacks.top(); stacks.pop();
            if(last->val >= node->val)
            {
                if(node1 == NULL)
                    {node1 = last; node2=node;}//3,2
                else
                    node2 = node;
            }
            last = node;
            node = node->right;
        }
    }
    std::swap(node1->val, node2->val);
}
```

Morris遍历，常数空间。

算法解释见binary-tree-inorder-traversal;

注意找出两个*node*后还得让遍历走完～以避免之前的改动*revert*完毕，否则可能会造成oj check时死循环(传入的树结构修改后不对).

```
void recoverTree(TreeNode *root)
{
    if(root == NULL) return;
    TreeNode* node1 = NULL, *node2 = NULL;
    TreeNode * pre = NULL;
    TreeNode * node = root;
    TreeNode  tmp(INT_MIN);
    TreeNode * last = &tmp;
    while(node)
    {
        if(node->left == NULL)
        {
            //visit node
            if(last->val >= node->val){
```

```
                    if(node1 == NULL)
                        {node1 = last; node2 = node;}
                    else
                        node2 = node;//can not break;
                }
                last = node;
                node = node->right;
            }
            else
            {
                pre = node->left;
                while(pre->right != NULL && pre->right != node)
                    pre = pre->right;
                if(pre->right == NULL)
                {
                    pre->right = node;
                    node = node->left;
                }
                else
                {
                    pre->right = NULL;
                    //visit node
                    if(last->val >= node->val){
                        if(node1 == NULL)
                            {node1 = last; node2 = node;}
                        else
                            node2 = node;//can not break;
                    }
                    last = node;
                    node = node->right;
                }
            }
        }
        std::swap(node1->val, node2->val);
    }
```

Same Tree

题目来源：Same Tree

> Given two binary trees, write a function to check if they are equal or not. Two binary trees are considered equal if they are structurally identical and the nodes have the same value.

解题思路：

思路跟上题对称树一样, 仍分递归和迭代两种方法。

递归

```
bool isSameTreeRecursion(TreeNode *p, TreeNode *q)
{
    if(p == NULL && q == NULL) return true;
    if(p == NULL || q == NULL) return false;
    if(p->val != q->val) return false;
    return isSameTree(p->left, q->left) && isSameTree(p->right, q->right);
}
```

迭代

```
bool isSameTree(TreeNode *p, TreeNode *q)
{
    stack<TreeNode*> stacks;
    stacks.push(p); stacks.push(q);
    while(! stacks.empty())
    {
        auto n1 = stacks.top(); stacks.pop();
        auto n2 = stacks.top(); stacks.pop();
        if(n1 == NULL && n2 == NULL) continue;
        if(n1 == NULL || n2 == NULL) return false;
        if(n1->val != n2->val) return false;
        stacks.push(n1->left);
        stacks.push(n2->left);
        stacks.push(n1->right);
        stacks.push(n2->right);
    }
    return true;
}
```

Sum Root to Leaf Numbers

题目来源：Sum Root to Leaf Numbers

>

```
Given a binary tree containing digits from 0-9 only, each root-to-leaf path
could represent a number.
An example is the root-to-leaf path 1->2->3 which represents the number
123.
Find the total sum of all root-to-leaf numbers.
For example,
   1
  / \
 2   3
 The root-to-leaf path 1->2 represents the number 12.
 The root-to-leaf path 1->3 represents the number 13.
 Return the sum = 12 + 13 = 25.
```

解题思路：

直接dfs，将到达叶节点代表的数字加起来即可。

```
    void dfs(long long input, TreeNode* node, long long &result)
    {
        assert(node != NULL);
        long long cur = input * 10 + node->val;
        if(node->left == NULL && node->right == NULL) //leaf
        {
            result += cur;
            return;
        }
        if(node->left)
            dfs(cur, node->left, result);
        if(node->right)
            dfs(cur, node->right, result);
    }
    int sumNumbers(TreeNode *root)
    {
        if(root == NULL) return 0;
        long long result = 0L;
        dfs(0, root, result);
        return (int)result;
    }
```

Symmetric Tree

题目来源：Symmetric Tree

> Given a binary tree, check whether it is a mirror of itself (ie, symmetric around its center). For example, this binary tree is symmetric: 1 / \ 2 2 / \ / \ 3 4 4 3 But the following is not: 1 / \ 2 2 \ \ 3 3 Note: Bonus points if you could solve it both recursively and iteratively.

解题思路： 左节点的左子树 = 右节点的由子树。 解题方法跟same tree差不多。

递归

```
bool isSymmetric(TreeNode* node1, TreeNode* node2)
{
    if(node1 == NULL && node2 == NULL) return true;
    if(node1 == NULL || node2 == NULL) return false;
    if(node1->val != node2->val ) return false;
    return isSymmetric(node1->left, node2->right) && isSymmetric(node1->right, node2-
}
bool isSymmetric(TreeNode *root)
{
    if(root == NULL) return true;
    return isSymmetric(root->left, root->right);
}
```

迭代

```
bool isSymmetric(TreeNode *root)
{
    if(root == NULL) return true;
    stack<TreeNode*> q;
    q.push(root->left);
    q.push(root->right);
    while(! q.empty())
    {
        auto right = q.top(); q.pop();
        auto left = q.top(); q.pop();
        if(left == NULL && right == NULL) continue;
        if(left == NULL || right == NULL) return false;
        if(left->val != right->val) return false;
        q.push(right->left);
        q.push(left->right);
        q.push(left->left);
        q.push(right->right);
    }
    return true;
}
```

Unique Binary Search Trees

题目来源：Unique Binary Search Trees

> Given n, how many structurally unique BST's (binary search trees) that store values 1...n? For example, Given n = 3, there are a total of 5 unique BST's. 1 3 3 2 1 \ / / / \ \ 3 2 1 1 3 2 / / \ \ 2 1 2 3

解题思路：

递归

递归比较好理解。比如 根节点数字为i, 比i小的左孩纸i-1个(子问题), 右孩纸n-i. 于是就有了下面的代码。

```
int numTrees(int n)
{
    if(n == 0) return 1;//recursion, maybe, real input 0 shoule return 0
    if(n == 1) return 1;
    int r = 0;
    for(int i = 1; i <= n; i++)
        r += numTrees(i-1)*numTrees(n-i);
    return r;
}
```

动态规划

其实可以缓存下, 用动态规划。

```
int f(int n)
{
    const int size = n+1;
    vector<int>  cache(size, 1);;
    for(int i = 2; i <=n; i++)
    {
        int result = 0;
        for(int j = 1; j <=i; j++)
            result += cache[j-1] * cache[i-j];
        cache[i] = result;
    }
    return cache[n];
}
int numTrees(int n)
{
    if(n == 0) return 0;
    return f(n);
}
```

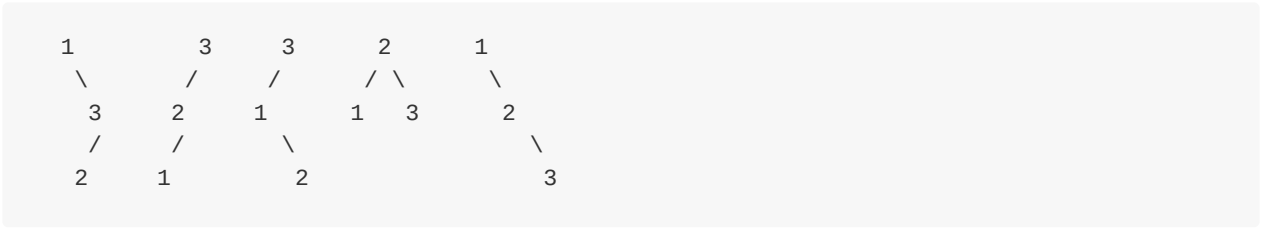
数学公式法

其实这个问题有公式可以直接算的，参考卡塔兰数 。

Unique Binary Search Trees II

题目来源：Unique Binary Search Trees II

> Given n, generate all structurally unique BST's (binary search trees) that store values 1...n. For example, Given n = 3, your program should return all 5 unique BST's shown below.



解题思路：

跟上题Unique Binary Search Trees一样，这个题目需要返回具体的结果。

```
vector<TreeNode*> generateTreesRec(int start, int end)
{
    vector<TreeNode *> result;
    if(start >= end)
    {
        if(start > end)
           result.push_back(NULL);
       else
           result.push_back(new TreeNode(start));
       return move(result);
    }
    for(int i = start; i <= end; i++)
    {
        auto left = generateTreesRec(start, i-1);
        auto right = generateTreesRec(i+1, end);
        for(auto it1 = left.begin(); it1 != left.end(); it1++)
           for(auto it2 = right.begin(); it2 != right.end(); it2++)
           {
               TreeNode* root = new TreeNode(i);
               root->left = *it1;
               root->right = *it2;
               result.push_back(root);
           }
    }
    return move(result);
}

vector<TreeNode *> generateTrees(int n)
{
   return generateTreesRec(1, n);//n==0, return TreeNode is NULL, included.
}
```

注意当end>start的时候也要返回一个空的Node，因为后面的遍历时，直接用left/right都有的情况才生成新的node。

Validate Binary Search Tree

题目来源：Validate Binary Search Tree

> Given a binary tree, determine if it is a valid binary search tree (BST). Assume a BST is defined as follows:
The left subtree of a node contains only nodes with keys less than the node's key. The right subtree of a node contains only nodes with keys greater than the node's key. Both the left and right subtrees must also be binary search trees.

解题思路：

递归判断节点值是否满足条件

```
bool _isBST(TreeNode * node, int min_, int max_)
{
    if(node == NULL) return true;
    if(node->val <= min_ || node->val >= max_) return false;
    return _isBST(node->left, min_, node->val)
         && _isBST(node->right, node->val, max_);
}
bool isValidBST(TreeNode *root)
{
    return _isBST(root, INT_MIN, INT_MAX);
}
```

中序遍历

BST 中序遍历结果是升序。 中序遍历的方法就多了，有递归、迭代、Morris遍历等，详情可以参考binary-tree-inorder-traversal, 下面就只列一种了。

```
bool isValidBST(TreeNode *root)
{
    stack<TreeNode*> stacks;
    TreeNode * node = root;
    int last = INT_MIN;
    while(true)
    {
        if(node)
        {
            stacks.push(node);
            node = node->left;
        }else
        {
            if(stacks.empty()) break;
            node = stacks.top(); stacks.pop();
            if(last >= node->val) return false;
            last = node->val;
            node = node->right;
        }
    }
```

```
        return true;
    }
```

再训练下Morris遍历。 跟recover-binary-search-tree一样，还是得强调下，Morris遍历中找到不是升序了，也不能return。因为修改了原来树的结构，必须rollback完毕才OK。

```
    bool isValidBST(TreeNode *root)
    {
        TreeNode * node = root;
        int last = INT_MIN;
        bool result = true;
        while(node)
        {
            if(node->left == NULL)
            {
                if(last >= node->val) result = false;;
                last = node->val;
                node = node->right;
            }else
            {
                auto pre = node->left;
                while(pre->right != NULL && pre->right != node)
                    pre = pre->right;
                if(pre->right == NULL)
                {
                    pre->right = node;
                    node= node->left;
                }else
                {
                    if(last >= node->val) result = false;//cannot return. return false;
                    pre->right = NULL;//reset
                    last = node->val;
                    node = node->right;
                }
            }
        }
        return result;
    }
```

sort, 排序相关

1. 3Sum Closest 题解
2. 3Sum 题解
3. 4Sum 题解
4. Insert Interval 题解
5. Longest Consecutive Sequence 题解
6. Merge Intervals 题解
7. Merge Sorted Array 题解
8. Remove Duplicates from Sorted Array 题解
9. Remove Duplicates from Sorted Array II 题解
10. Sort Colors 题解
11. Two Sum 题解

3Sum Closest

题目来源：3Sum Closest

> Given an array S of n integers, find three integers in S such that the sum is closest to a given number, target. Return the sum of the three integers. You may assume that each input would have exactly one solution. For example, given array S = {-1 2 1 -4}, and target = 1. The sum that is closest to the target is 2. (-1 + 2 + 1 = 2).

解题思路：

跟3-sum思路一致。

```
int threeSumClosest(vector<int> &num, int target)
{
   int n = num.size();
   int result = 0;
   int closest = INT_MAX;
   std::sort(num.begin(), num.end());
   for(int i = 0; i < n-2; i++)
   {
      if (i > 0 && num[i-1] == num[i]) continue;
      int begin = i+1;
      int end = n - 1;
      while(begin < end)
      {
          int sum = num[i] + num[begin] + num[end];
          if(closest > abs(sum-target))
               result = sum, closest = abs(sum-target);
          if(sum > target)
              --end;
          else if (sum < target)
              ++begin;
          else
               return sum;
       }
    }
   return move(result);
}
```

3Sum

题目来源：3Sum

> Given an array S of n integers, are there elements a, b, c in S such that a + b + c = 0? Find all unique triplets in the array which gives the sum of zero. Note: Elements in a triplet (a,b,c) must be in non-descending order. (ie, a ≤ b ≤ c) The solution set must not contain duplicate triplets. For example, given array S = {-1 0 1 2 -1 -4}, A solution set is: (-1, 0, 1) (-1, -1, 2)

解题思路：

可以沿用2sum的思路。 排序后以当前值target为基准，从当前值后面的值中找等于-target的所有对。注意去重～

- 当前值i的去重
- search对数的时候去重

```
void search(vector<std::pair<int, int> > &pairs, int start, const int target, const vecto
{
   int end = num.size() - 1;
   while(start < end)
   {
       int sum = num[start] + num[end];
       if(sum == target)
       {
           pairs.push_back(make_pair(num[start], num[end]));
           while(start < end && num[start] == num[start+1])
               ++start;
           ++start;//[start] != [start+1]
           while(start < end && num[end] == num[end-1])
               --end;
           --end;
       }else if (sum > target)
           --end;
       else
           ++start;
   }
}

vector<vector<int> > threeSum(vector<int> &num)
{
   int n = num.size();
   vector<vector<int> > result;
   std::sort(num.begin(), num.end());
   for(int i = 0; i < n-2; i++)
   {
       int target = - num[i];
       vector<std::pair<int, int> > pairs;
       search(pairs, i+1, target, num);
       if (!(pairs.empty()))
       {
           for(auto it = pairs.begin(); it != pairs.end(); ++it)
               result.push_back(vector<int>{num[i], it->first, it->second});
       }
       while (i < n-2 && num[i+1] == num[i]) i++;
   }
```

```
    return move(result);
}
```

或者直接这样：

```
vector<vector<int> > threeSum(vector<int> &num)
{
    int n = num.size();
    vector<vector<int> > result;
    std::sort(num.begin(), num.end());
    for(int i = 0; i < n-2; i++)
    {
        int begin = i+1;
        int end = n - 1;
        while(begin < end)
        {
            int sum = num[i] + num[begin] + num[end];
            if(sum > 0)
                --end;
            else if (sum < 0)
                ++begin;
            else
            {
                result.push_back(vector<int>{num[i], num[begin], num[end]});
                while(begin < end && num[begin] == num[begin+1])
                    ++begin;
                ++begin;
                while(begin < end && num[end] == num[end-1])
                    --end;
                --end;
            }
        }
        while (i < n-2 && num[i+1] == num[i]) i++;
    }
    return move(result);
}
```

4Sum

题目来源：4Sum

> Given an array S of n integers, are there elements a, b, c, and d in S such that a + b + c + d = target? Find all unique quadruplets in the array which gives the sum of target. Note: Elements in a quadruplet (a,b,c,d) must be in non-descending order. (ie, a ≤ b ≤ c ≤ d) The solution set must not contain duplicate quadruplets. For example, given array S = {1 0 -1 0 -2 2}, and target = 0. A solution set is: (-1, 0, 0, 1) (-2, -1, 1, 2) (-2, 0, 0, 2)

解题思路：

跟3-sum思路一样，只不过这个是\(N^2 \) 两两组合，再去找2sum, 得注意去重。

```
void search(vector<std::pair<int, int> > &pairs, int start, const int target, const vecto
{
   int end = num.size() - 1;
  while(start < end)
  {
      int sum = num[start] + num[end];
      if(sum == target)
      {
          pairs.push_back(make_pair(num[start], num[end]));
          while(start < end && num[start] == num[start+1])
              ++start;
          ++start;//[start] != [start+1]
          while(start < end && num[end] == num[end-1])
              --end;
          --end;
      }else if (sum > target)
          --end;
      else
          ++start;
  }
}
vector<vector<int> > fourSum(vector<int> &num, int target)
{
   int n = num.size();
   vector<vector<int> > result;
   if(n <= 3) return result;
   std::sort(num.begin(), num.end());
   for(int i = 0; i < n-3; i++)
   {
       if(num[i] + num[i+1] + num[i+2] + num[i+3] > target) break;
       for(int j = i+1; j < n-2; j++)
       {
           int sum = num[i] + num[j];
           vector<std::pair<int, int> > pairs;
           search(pairs, j+1, target-sum, num);
           if (!(pairs.empty()))
           {
               for(auto it = pairs.begin(); it != pairs.end(); it++)
                   result.push_back(vector<int>{num[i], num[j], it->first, it->second});
           }
           while (j+1 < n-2 && num[j+1] == num[j]) j++;
       }
       while (i < n-3 && num[i+1] == num[i]) i++;
```

```
    }
    return move(result);
}
```

Insert Interval

题目来源：Insert Interval

>

```
Given a set of non-overlapping intervals, insert a new interval into the intervals (merge
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]
```

解题思路：

可以先加进去，然后再按照Merge Intervals的算法merge一下就行。

```
    vector<Interval> merge2(vector<Interval> &intervals)
    {
        vector<Interval> result;
        if(intervals.size() == 1) return intervals;
        int i = 1;
        auto start = intervals[0];
        while(i < intervals.size())
        {
            if(intervals[i].end <= start.end)
            {
                i++;
            }else if(intervals[i].start <= start.end && intervals[i].end >= start.end)
            {
                start.end = intervals[i].end;
                i++;
            }
            else
            {
                result.push_back(start);
                if(i < intervals.size())
                    start = intervals[i];
                else
                    return result;
            }
        }
        result.push_back(start);
        return move(result);
    }
    vector<Interval> insert(vector<Interval> &intervals, Interval newInterval)
    {
        if(intervals.size() == 0) return vector<Interval>(1, newInterval);
         int i = 0;
        vector<Interval> newIntervals(intervals.size()+1);

        while(i < intervals.size())
        {
            if(intervals[i].start < newInterval.start)
            {
```

```
                i++;
            }else if (intervals[i].start == newInterval.start)
            {
                if(intervals[i].end >= newInterval.end)
                    ;
                else
                    i++; //the newInterval first
                break;
            }else
            {
                break;
            }
        }
        int index = 0;
        while(index < i)
        {
            newIntervals[index] = intervals[index];
            index++;
        }
        newIntervals[index] = newInterval;
        while(index < intervals.size())
        {
            newIntervals[index+1] = intervals[index];
            index++;
        }
        return merge2(newIntervals);
    }
```

或者 直接加。参考了 leetcode-cpp 原文中的代码貌似有个testcase 过不了TLE。

```
    vector<Interval> insert(vector<Interval> &intervals, Interval newInterval)
    {
        if(intervals.size() == 0) return vector<Interval>(1, newInterval);
        int n = intervals.size();
        vector<Interval> result;
        for(int i = 0; i < n; i++)
        {
            auto &it = intervals[i];
            if(newInterval.end < it.start)
            {
                result.push_back(newInterval);
                std::copy(intervals.begin()+i, intervals.end(), std::back_inserter(result
                return move(result);
            }else if(newInterval.start > it.end)
                result.push_back(it);
            else
            {
                newInterval.start = std::min(newInterval.start, it.start);
                newInterval.end = std::max(newInterval.end, it.end);
            }
        }
        result.push_back(newInterval);
        return move(result);
    }
```

Longest Consecutive Sequence

题目来源：Longest Consecutive Sequence

>

```
Given an unsorted array of integers, find the length of the longest consecutive
elements sequence.

For example,
Given [100, 4, 200, 1, 3, 2],
The longest consecutive elements sequence is [1, 2, 3, 4]. Return its length:
4.

Your algorithm should run in O(n) complexity.
```

解题思路：

利用hashmap

用一个set/map记录每个数，然后挨个找相邻的数字，每找到一个就从原set/map中去掉，直到全部遍历完毕。

```
    int longestConsecutive(vector<int> &num)
    {
        int result = 0;
        unordered_set<int> data(num.begin(), num.end());
        while(! data.empty())
        {
            int v = *(data.begin());
            data.erase(data.begin());
            int i = 1;
            int len = 1;
            while(data.find(v-i) != data.end())
            {
                ++len;
                data.erase(data.find(v-i));
                ++i;
            }
            i = 1;
            while(data.find(v+i) != data.end())
            {
                ++len;
                data.erase(data.find(v+i));
                ++i;
            }
            result = std::max(result, len);
        }
        return result;
    }
```

先利用O(n)的排序

这也是参考了discuss的答案。 先用一个O(n)的排序算法，然后挨个左右看就是。 注意数组中可能含有相同的数字以及负数。

这里用基数排序radixsort，注意基数排序中内部计数排序时注意，输入可能含有负数，因此映射的下标不能是[0,9],而是还得把负数的另外一半算上即[0,18],-9->0, 9->18.

```
//-9 ---> index is 0 //9 --->index is 18
int getBucket(int n, int base)
{
    return n / base % 10 + 9;
}

//按照个位(base=1)、十位(base=10)排序
void countSort(vector<int> &num, int base)
{
    vector<int> numback(num);
    vector<int> counts(19, 0);
    for(int i = 0; i < numback.size(); i++)
    {
        int bucket = getBucket(numback[i], base);
        ++counts[bucket];
    }
    for(int j = 1; j < counts.size(); j++)
        counts[j] += counts[j-1];
    for(int j = (int)numback.size()-1; j >= 0; j--)
    {
        int index = getBucket(numback[j], base);
        num[counts[index]-1] = numback[j];
        counts[index]--;
    }
}
//O(N) sort, then scan to get the result
void radixSort(vector<int> &num)
{
    int max = INT_MIN;
    for(int i = 0; i < num.size(); i++)
        max = std::max(max, abs(num[i])); //!! abs
    int base = 1;
    while(max / base)
    {
        countSort(num, base);
        base *= 10;
    }
}
//ref https://oj.leetcode.com/discuss/2731/this-problem-has-a-o-n-solution?show=4368#
int longestConsecutive2(vector<int> &num)
{
    if(num.size() <= 1) return num.size();
    radixSort(num);
    int max = 1;
    int len = 1;
    for(int i = 1; i < num.size(); i++)
    {
        if(num[i] == num[i-1])//!!
            continue;
        if(num[i] == num[i-1] + 1)
            len++;
        else
        {
            max = std::max(max, len);
```

```
                len = 1;
            }
        }
        return std::max(max, len);
    }
```

Merge Intervals

题目来源：Merge Intervals

> 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].

解题思路：

先对start排序, 排序时这样 [1,4] [1,5]时,[1,5][1,4] 统一处理[1,4][2,4]吃掉。 然后数轴上对前后两个interval作讨论，3种情况，一种including，i+1包含在i里面; i+1和i相交，i+1和i相隔。

```
/**
 * Definition for an interval.
 * struct Interval {
 *     int start;
 *     int end;
 *     Interval() : start(0), end(0) {}
 *     Interval(int s, int e) : start(s), end(e) {}
 * };
 */
int cmp(const Interval &i1, const Interval &i2)
{
    if(i1.start == i2.start) return i1.end > i2.end; // make second bigger first
    return i1.start < i2.start;
}
class Solution
{
public:
    vector<Interval> merge(vector<Interval> &intervals)
    {
        vector<Interval> result;
        if(intervals.size() <= 1) return intervals;
        std::sort(intervals.begin(), intervals.end(), cmp);
        int i = 1;
        auto start = intervals[0];
        while(i < intervals.size())
        {
            if(intervals[i].end <= start.end)
            {
                i++;
            }else if(intervals[i].start <= start.end && intervals[i].end >= start.end
            {
                start.end = intervals[i].end;
                i++;
            }
            else
            {
                result.push_back(start);
                if(i < intervals.size())
                    start = intervals[i];
                else
                    return result;
            }
        }
        result.push_back(start);
        return move(result);
```

```
        }
    };
```

当然，也可以借助 Insert Interval 一个一个插入到结果里面去。

```
    vector<Interval> merge(vector<Interval> &intervals)
    {
        vector<Interval> result;
        if(intervals.size() <= 1) return intervals;
        for(int i = 0; i < intervals.size(); i++)
            result = insert(result, intervals[i]);
        return move(result);
    }
    vector<Interval> insert(vector<Interval> &intervals, Interval newInterval)
    {
        if(intervals.size() == 0) return vector<Interval>(1, newInterval);
        int n = intervals.size();
        vector<Interval> result;
        for(int i = 0; i < n; i++)
        {
            auto &it = intervals[i];
            if(newInterval.end < it.start)
            {
                result.push_back(newInterval);
                std::copy(intervals.begin()+i, intervals.end(), std::back_inserter(result
                return move(result);
            }else if(newInterval.start > it.end)
                result.push_back(it);
            else
            {
                newInterval.start = std::min(newInterval.start, it.start);
                newInterval.end = std::max(newInterval.end, it.end);
            }
        }
        result.push_back(newInterval);
        return move(result);
    }
```

Merge Sorted Array

题目来源：Merge Sorted Array

> 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 (size that is greater or equal to m + n) to hold additional elements from B. The number of elements initialized in A and B are m and n respectively.

解题思路： 因为A空间足够，所以直接从后往前定位就可以。只需要将B放完即可，若B放完，A也已经呆在应该待的位置了。

```
void merge(int A[], int m, int B[], int n)
{
    int tot = m + n - 1;
    int i = m-1;
    int j = n-1;
    while(j >= 0)
    {
        if(i>=0 && A[i] >= B[j])
            A[tot--] = A[i--];
        else
            A[tot--] = B[j--];
    }
}
```

Remove Duplicates from Sorted Array

题目来源：Remove Duplicates from Sorted Array

> Given a sorted array, remove the duplicates in place such that each element appear only once and return the new length. Do not allocate extra space for another array, you must do this in place with constant memory. For example, Given input array A = [1,1,2], Your function should return length = 2, and A is now [1,2].

解题思路：

```
int removeDuplicates(int A[], int n)
{
    if(n <= 1) return n;
    int index = 0;
    int i = 0;
    while(i < n)
    {
        while(i+1 < n && A[i+1] == A[i])
            i++;
        //A[i+1] != A[i] // 1 1 2
        A[index++] = A[i++];
    }
    return index; //index
}
```

Remove Duplicates from Sorted Array II

题目来源：Remove Duplicates from Sorted Array II

> Follow up for "Remove Duplicates": What if duplicates are allowed at most twice? For example, Given sorted array A = [1,1,1,2,2,3], Your function should return length = 5, and A is now [1,1,2,2,3].

解题思路：

```
int removeDuplicates(int A[], int n)
{
    if(n <= 2) return n;
    int index = 0;
    int i = 0;
    while(i < n)
    {
        int count = 1;
        while (i+1 < n && A[i+1] == A[i])
        {
            count++; i++;
        }
        if (count >= 2)
        {
            A[index++] = A[i]; //A[i+1] != A[i]
            A[index++] = A[i++];
        } else
            A[index++] = A[i++];
    }
    return index;
}
```

Sort Colors

题目来源：Sort Colors

> Given an array with n objects colored red, white or blue, sort them so that objects of the same color are adjacent, with the colors in the order red, white and blue. Here, we will use the integers 0, 1, and 2 to represent the color red, white, and blue respectively.
Note: You are not suppose to use the library's sort function for this problem. A rather straight forward solution is a two-pass algorithm using counting sort. First, iterate the array counting number of 0's, 1's, and 2's, then overwrite array with total number of 0's, then 1's and followed by 2's. Could you come up with an one-pass algorithm using only constant space?

解题思路：

countSort= O(2*n)

按照提示，分别数数0,1,2各有多少个，然后填充进去即可。简单的countsort.

```
    void sortColors(int A[], int n)
    {
        int one = 0, two = 0, zero = 0;
        for(int i = 0; i < n; i++)
        {
            if(A[i] == 0)
                ++zero;
            else if(A[i] == 1)
                ++one;
            else
                ++two;
        }
        int i = 0;
        while(i < zero)
            A[i++] = 0;
        while(i < zero+one)
            A[i++] = 1;
        while(i < zero+one+two)
            A[i++] = 2;
    }
```

O(1*n) 算法

设前面的数字已经排好序. 0000 111 222 *1*\02*...

记录第一次出现1的index，第一次出现2的index， 当前搜索的数是1的话，将first2改为1，当前index的数改为2即可...即:

```
current = 1: first2 = 1, current=2;
current = 0: first1=0,first2=1,current=2;
current = 2: continue即可。
```

注意修改方式从后往前即可避免繁琐的边界值(first2=first1等情况) 这个主要是参考了anyone with one pass and constant space solution.

```
 void sortColors(int A[], int n)
{
    if(n <= 1) return;
    int zero = 0;
    int one = 0;
    int two = 0;
    for(int cur = 0; cur < n; cur++)
    {
        if(A[cur] == 2){
            A[two++]=2;
        }else if(A[cur] == 1){
            A[two++]=2;
            A[one++]=1;
        }else{ //0
            A[two++]=2;
            A[one++]=1;
            A[zero++]=0;
        }
    }
}
```

O(1*n) 算法

双指针算法, 参考 leetcode-cpp.

```
zero记录最后一个0的的后一个index(可能是1或2), two记录最开始一个2的位置。
cur == 0: 将zero的后一个与cur交换, 二者都+1;
cur == 2: 将two和当前对应值交换, two是放2的位置, 下一次放2,放到two-1的地方, 交换的值可能是0, 所以cur
```

```
void sortColors(int A[], int n)
{
    if(n <= 1) return;
    int zero = 0; int two = n-1;
    int i = 0;
    while(i <= two)
    {
        if(A[i] == 0)
            std::swap(A[zero++], A[i++]);
        else if(A[i] == 2)
            std::swap(A[two--], A[i]);//i cannot ++, maybe A[two] is zero.
        else
            i++;
    }
}
```

Two Sum

题目来源:leetcode-two-sum

> 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

解答：

1、先排序（得记录index），i->0, j->n 相加结果sum<target, i++ 否则 j—

```
vector<int> twoSum(vector<int> &numbers, int target)
{
    vector<pair<int, int> > num_index_map;
    for(int i = 0; i < numbers.size(); i++)
        num_index_map.push_back(pair<int, int>(numbers[i], i+1));
    std::sort(num_index_map.begin(), num_index_map.end(), [](const pair<int,int> &a, cons
    int i = 0;
    int j = numbers.size() - 1;
    while(i < j)
    {
        int tmp = num_index_map[i].first + num_index_map[j].first;
        if( tmp == target)
        {
            vector<int> result(2); //capacity, then push back, becomes 3
            //quick sort is not stable.
            if(num_index_map[i].second < num_index_map[j].second)
            {
                result[0] = (num_index_map[i].second);
                result[1] = (num_index_map[j].second);
            }else
            {
                result[1] = (num_index_map[i].second);
                result[0] = (num_index_map[j].second);
            }
            return result;
        }else if(tmp < target)
        {
            i++;
        }else
        {
            j--;
        }
    }
    return vector<int>();
}
```

2、用map存起来～直接找对应的另一半

```
vector< int> twoSum(vector< int > &numbers, int target)
{
    unordered_map< int , int > numIndex;
    int index = 0;
    for( auto num : numbers)
    {
        if (numIndex.find(target - num) != numIndex.end())
        {
            return vector<int >{numIndex[target-num]+1, index+1};
        }
        numIndex[num] = index++;
    }
}
```

search, 搜索相关

1. First Missing Positive 题解
2. Find Minimum in Rotated Sorted Array 题解
3. Find Minimum in Rotated Sorted Array II 题解
4. Median of Two Sorted Arrays 题解
5. Search Insert Position 题解
6. Search a 2D Matrix 题解
7. Search for a Range 题解
8. Search in Rotated Sorted Array 题解
9. Search in Rotated Sorted Array II 题解
10. Single Number 题解
11. Single Number II 题解

First Missing Positive

题目来源：First Missing Positive

> Given an unsorted integer array, find the first missing positive integer. For example, Given [1,2,0] return 3, and [3,4,-1,1] return 2. Your algorithm should run in O(n) time and uses constant space.

解题思路：

注意有可能含有重复的，若非重复的可以用应该得的正数和和实际正数和之差得到miss的正数，如果为0的话，就是本来就是连续的数，first missing的就是max+1. 含有重复的话可以用

这里用原地改变下数组的位置来算，例如 3 3 1 4 0,将index所在的数（[0,n]的数）放到应该放的数里面去（数-1），比如这里应该是放到 3 应该放到index＝2，1放到index＝0，4放到index＝3，所有数的放好后，再从头开始扫描数组，第一个下标所得的数不是index+1，那么差的就是这个数。复杂度不超过O（2*n） A里面的数，若A[i]是正数0-n之间的，则把TA放到位置i-1处。即最后达到的效果是 A[i]=i+1，不然就是负数 或者大于n的数。第一遍遍历把所有的数归位，第二遍找，哪个位置差了，就那个位置对应的数i+1就是差的正数。

第一版本的代码确实很戳啊。

```
int firstMissingPositive(int A[], int n)
{
    if (n == 0) return 1;
    if (n == 1) return A[0] > 0 ? (A[0] == 1 ? 2 : 1) : 1;

    int index = 0;
    while(index < n)
    {
        int tmp = A[index];
        bool goon = true;
        while(goon)
        {
            if(tmp-1 >=0 && tmp-1 < n)
            {
                if(A[tmp-1] == tmp)
                {
                    goon = false;
                    index++;
                }
                else
                {
                    int tmpbak = A[tmp-1];
                    A[tmp-1] = tmp;
                    tmp = tmpbak;
                }
            }else
            {
                break;
            }
        }
        while(index < n && (A[index] == index+1 ||
              A[index] <= 0 || A[index] >= n))
            index++;
    }
```

```
        index = 0;
        while(index < n)
        {
            if(A[index] != index+1)
                return index+1;
            index++;
        }
        return n+1;
    }
```

其实跟下面的代码一个意思。

```
    int firstMissingPositive(int A[], int n)
    {
        if(n == 0) return 1;
        int i = 0;
        while(i < n)
        {
            while(i < n && A[i] != i+1)
            {
                if(A[i] <= 0 || A[i] > n || A[i] == A[A[i]-1])
                    i++;
                else
                    std::swap(A[i], A[A[i]-1]);
            }
            i++;
        }
        for(int i = 0; i < n; i++)
            if(A[i] != (i+1)) return i+1;
        return n+1;
    }
```

find minimum in rotated sorted array

题目来源：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.

解题思路：

rotate总是至少有一半是有序的，可以根据这一半有序的值去二分。跟Search in Rotated Sorted Array一样。

```
int findMin(vector<int> &num)
{
   assert(num.size() != 0);
   if(num.size() == 1) return num[0];
   int result = INT_MAX;
   int left = 0; int right = num.size() - 1;
   while(left <= right)
   {
       int mid = left + ((right - left)>>1);
       if (num[mid] < num[right])//right is sorted
       {
           result = std::min(result, num[mid]);
           right = mid-1;
       }else if(num[left] <= num[mid]) //left is sorted, left maybe equals mid
       {
           result = std::min(result, num[left]);
           left = mid+1;
       }
   }
   return result;
}
```

find minimum in rotated sorted array II

题目来源：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.

解题思路：

跟Search in Rotated Sorted Array II一样。

```
int findMin(vector<int> &num)
{
   assert(num.size() != 0);
   if(num.size() == 1) return num[0];
   int result = INT_MAX;
   int left = 0; int right = num.size() - 1;
   while(left <= right)
   {
       int mid = left + ((right - left)>>1);
       if (num[mid] < num[right])//right is sorted
       {
           result = std::min(result, num[mid]);
           right = mid-1;
       }else if(num[left] < num[mid]) //left is sorted
       {
           result = std::min(result, num[left]);
           left = mid+1;
       }else if(num[mid] == num[left])
           ++left, result = std::min(result, num[mid]);
       else
           --right, result = std::min(result, num[mid]);
   }
   return result;
}
```

Median of Two Sorted Arrays

题目来源：Median of Two Sorted Arrays

> There are two sorted arrays A and B of size m and n respectively. Find the median of the two sorted arrays. The overall run time complexity should be O(log (m+n)).

解题思路：

`log` 得二分了。 思想是将A[k/2-1] 与 B[k/2-1]比较：如果 `A[k/2-1] < B[k/2-1]` 意味着, A[0: k/2-1] 不会大于合并后的第k个数。

```
//kth number, increase. k starts from 1
int findkth(int *a, int m, int *b, int n, int k)
{
    if(m > n) return findkth(b, n, a, m, k);
    //m < n
    if(m == 0) return b[k-1];
    if(k == 1) return std::min(a[0], b[0]);
    int ka = std::min(m, k>>1);
    int kb = k - ka;
    if(a[ka-1] < b[kb-1])
        return findkth(a+ka, m-ka, b, n, k-ka);
    else if (a[ka-1] > b[kb-1])
        return findkth(a, m, b+kb, n-kb, k-kb);
    return a[ka-1];
}

double findMedianSortedArrays(int a[], int m, int b[], int n)
{
    assert(!(m == 0 && n == 0));
    if ((m+n)&0x1)
        return findkth(a, m, b, n, ((m+n)>>1)+1);
    return  (findkth(a, m, b, n, ((m+n)>>1)+1) + findkth(a, m, b, n, (m+n)>>1))*0.5;
}
```

Search Insert Position

题目来源：Search Insert Position

> Given a sorted array and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order. You may assume no duplicates in the array. Here are few examples. [1,3,5,6], 5 → 2 [1,3,5,6], 2 → 1 [1,3,5,6], 7 → 4 [1,3,5,6], 0 → 0

解题思路：

注意运算符优先级。mid = start + ((end - start) >> 1); 后面的括号非常有必要！>> 优先级比+低！ 参考 lowerBound.

```
int searchInsert(int A[], int n, int target)
{
    assert(A != NULL && n != 0);
    int left = 0; int right = n;
    while(left != right)
    {
        int mid = left + ((right - left)>>1);
        if(A[mid] == target) return mid;
        if(A[mid] < target)
            left = mid+1;
        else
            right = mid; //NOT mid-1
    }
    return left;
}
```

Search a 2D Matrix

题目来源：Search a 2D Matrix

> Write an efficient algorithm that searches for a value in an m x n matrix. This matrix has the following properties: Integers in each row are sorted from left to right. The first integer of each row is greater than the last integer of the previous row. For example, Consider the following matrix: [[1, 3, 5, 7], [10, 11, 16, 20], [23, 30, 34, 50]] Given target = 3, return true.

解题思路：
二分查找，将index转化为matrix的row/col即可。

```
int cmp(vector<vector<int > > &matrix, const int m, const int n, int index, int targe
{
    int row = index / n;
    int col = index % n;
    if(matrix[row][col] == target)
        return 0;
    if(matrix[row][col] < target)
        return -1;
    return 1;
}

bool searchMatrix(vector<vector< int> > &matrix, int target)
{
    if(matrix.size() == 0) return false ;
    int m = matrix.size();
    int n = matrix[0].size();
    int left = 0; int right = m * n - 1;
    while(left <= right)
    {
        int mid = left + ((right - left)>>1);
        int c = cmp(matrix, m, n, mid, target);
        if (c == 0)
            return true ;
        else if (c < 0)
            left = mid+1;
        else
            right = mid-1;
    }
    return false ;
}
```

Search for a Range

题目来源：Search for a Range

> Given a sorted array of integers, find the starting and ending position of a given target value. Your algorithm's runtime complexity must be in the order of O(log n). If the target is not found in the array, return [-1, -1]. For example, Given [5, 7, 7, 8, 8, 10] and target value 8, return [3, 4].

解题思路：

二分搜索,搜到了后前后找相同的,非Log(n)算法。 lower返回插入点(相等的最小的index)的位置，upper返回比target大的位置。right从n开始，非n-1；

std::lower_bound.

std::upper_bound.

```
int lower(int *A, int n, int target)
{
    int left = 0;
    int right = n;
    while(left != right)
    {
        int mid = left + ((right - left)>>1);
        if(A[mid] < target)
            left = mid+1;
        else
            right = mid;//NOT mid-1
    }
    return left;
}
int upper(int *A, int n, int target)
{
    int left = 0;
    int right = n;
    while(left != right)
    {
        int mid = left + ((right - left)>>1);
        if(A[mid] <= target)
            left = mid+1;
        else
            right = mid;//NOT mid-1
    }
    return left;
}

vector<int> searchRange(int A[], int n, int target)
{
    assert(A != NULL && n != 0);
    int left = lower(A, n, target);
    int right = upper(A, n, target);
    if(left == n || A[left] != target)
        return vector<int>(2, -1);
    return vector<int>{left, right-1};

}
```

Search in Rotated Sorted Array

题目来源：Search 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). You are given a target value to search. If found in the array return its index, otherwise return -1. You may assume no duplicate exists in the array.

解题思路：

rotate总是至少有一半是有序的，可以根据这一半有序的值去二分。

```
int search(int A[], int n, int target)
{
    assert(n != 0);
    int left = 0; int right = n-1;
    while(left <= right)
    {
        int mid = left + ((right - left)>>1);
        if(A[mid] == target) return mid;
        if(A[left] <= A[mid])//left is sorted, left may equal mid
        {
            if(target >= A[left] && target < A[mid])//target in A[left,mid]
                right = mid - 1;
            else
                left = mid + 1;
        }else //right is sorted
        {
            if(target > A[mid] && target <= A[right]) //target in A[mid,right]
                left = mid + 1;
            else
                right = mid - 1;
        }
    }
    return -1;
}
```

Search in Rotated Sorted Array II

题目来源：Search in Rotated Sorted Array II

> Follow up for "Search in Rotated Sorted Array": What if duplicates are allowed?
Would this affect the run-time complexity? How and why? Write a function to determine if a given target is in the array.

解题思路： 跟Search in Rotated Sorted Array相比， 不能通过A[left] <= A[mid] 判断这段有序。

```
bool search(int A[], int n, int target)
{
    assert(n != 0);
    int left = 0; int right = n - 1;
    while(left <= right)
    {
        int mid = left + ((right-left)>>1);
        if(A[mid] == target) return true;
        if(A[left] < A[mid])//left is sorted
        {
            if(A[left] <= target && target < A[mid])
                right = mid-1;
            else
                left = mid+1;
        }else if(A[mid] < A[right])//right is sorted
        {
            if(A[mid] < target && target <= A[right])
                left = mid+1;
            else
                right = mid-1;
        }else if (A[left] == A[mid])//1 3 1 1 1
            ++left;
        else//A[mid] == A[right]
            --right;
    }
    return false;
}
```

Single Number

题目来源：Single Number

>

```
Given an array of integers, every element appears twice except for one. Find
that single one.

Note:
Your algorithm should have a linear runtime complexity. Could you implement it
without using extra memory?
```

解题思路：

普通程序员方法

用一个hashmap数数，再遍历一次即可。

```
int singleNumber(int A[], int n)
{
    unordered_map<int, int> count;
    for(int i = 0; i < n; i++)
        count[A[i]]++;
    for(int i = 0; i < n; i++)
    {
        if(count[A[i]] != 2)
            return A[i];
    }
    //error
    return 0;
}
```

文艺程序员方法

看题目要求不用额外的存储~ 然后所有数字出现2次～ 然后想想位运算。能想到位运算应该就差不多了。
`1^1 = 0`

```
int singleNumber(int A[], int n)
{
    assert(n != 0);
    int r = A[0];
    for(int i = 1; i < n; i++)
        r ^= A[i];
    return r;
}
```

Single Number II

题目来源：Single Number II

>

```
Given an array of integers, every element appears three times except for one. Find that s

Note:
Your algorithm should have a linear runtime complexity. Could you implement it without us
```

解题思路：

普通程序员方法

用一个hashmap数数，再遍历一次即可。代码就略了。

文艺程序员方法

有了Single Number的思路，可能你会想想用位运算。不过一时半会貌似想不太出来。没关系，开一个32位数组，每个数字出现3次，相应的位肯定出现3次的整数倍。剩下的那些数对应的那个应该就是要找的了。

```
int singleNumber(int A[], int n)
{
    int count[32] = {0};
    for(int i = 0; i < n; i++)
        for(int j = 0; j < 32; j++)
        {
            if(A[i] & (1<<j))
                count[j] = (count[j]+1) % 3;
        }
    int r = 0;
    for(int i = 0; i < 32; i++)
    {
        if(count[i] != 0)
            r |= (1<<i);
    }
    return r;
}
```

极品程序员

从discuss看到极品程序员的答案。值得学习，不过有时候容易搞混。个人认为上面第1种(文艺)程序员的方法就不错。

```
int singleNumber(int A[], int n)
{
    int ones = 0, twos = 0, threes = 0;
```

```
        for(int i = 0; i < n; i++)
        {
            threes = twos & A[i]; //已经出现两次并且再次出现
            twos = twos | (ones & A[i]); //曾经出现两次的或者曾经出现一次但是再次出现的
            ones = ones | A[i]; //出现一次的

            twos = twos & ~threes; //当某一位出现三次后，我们就从出现两次中消除该位
            ones = ones & ~threes; //当某一位出现三次后，我们就从出现一次中消除该位
        }
        return ones; //twos, threes最终都为0. ones是只出现一次的数
    }
```

另外，有题目是数组中有2个数不一样，其他都出现2次，找出这两个数。 思路是全部异或得到一个数～然后从这个数中找到一个二进制位为1的位置～（说明原来的两个数这个位不一样，一个为0，一个为1）然后根据这个位可以将原始的数组分成2组。再根据single number i的思路单独每组异或得到结果。

math, 数学类相关

1. Add Binary 题解
2. Add Two Numbers 题解
3. Divide Two Integers 题解
4. Gray Code 题解
5. Integer to Roman 题解
6. Multiply Strings 题解
7. Palindrome Number 题解
8. Plus One 题解
9. Pow(x, n) 题解.md)
10. Reverse Integer 题解
11. Roman to Integer 题解
12. Sqrt(x) 题解.md)
13. String to Integer (atoi) 题解.md)
14. Valid Number 题解

Add Binary

题目来源：Add Binary

> Given two binary strings, return their sum (also a binary string). For example, a = "11" b = "1" Return "100".

解题思路：

跟前面的 plus-one 差不多。 这里注意char和int的转换，别搞错了。

```
string addBinary(string a, string b)
{
    int i = 0;
    while(a[i] == ' ')  i++;
    a = a.substr(i, a.length()-i);
    i = 0;
    while(b[i] == ' ') i++;
    b = b.substr(i, b.length()-i);
    int m = a.length(); int n = b.length();
    if(m < n) return addBinary(b, a);
    if(n == 0) return a;
    string result(a);
    for(int i = n-1, j = m-1; i >= 0; i--,j--)
        result[j] = (result[j] - '0') + (b[i] - '0') + '0';
    for(int i = m-1; i >= 1; i--)
    {
        int c = result[i] - '0';
        result[i] = (c % 2) + '0';
        result[i-1] += (c / 2);
    }
    int first = result[0] - '0';
    if(first > 1)
    {
        result[0] = (first % 2) + '0';
        result = "1" + result;
    }
    return result;
}
```

Add Two Numbers

题目来源：Add Two Numbers

> 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

解题思路：

递归版

```
ListNode *addTwoNumbers(ListNode *l1, ListNode *l2, int carry)
{
    if(l1 == NULL && l2 == NULL && carry == 0) //only comes the carry,like 5, 5 = 10,
        return NULL;
    ListNode * result = new ListNode(carry);
    if(l1 != NULL)
        result->val += l1->val;
    if(l2 != NULL)
        result->val += l2->val;
    result->next = addTwoNumbers(l1 == NULL ? NULL : l1->next, l2 == NULL ? NULL : l2
    result->val %= 10;
    return result;
}

ListNode *addTwoNumbers(ListNode *l1, ListNode *l2)
{
    if(l1 == NULL && l2 == NULL)
        return NULL;
    if(l1 == NULL)
        return l2;
    if(l2 == NULL)
        return l1;
    return addTwoNumbers(l1, l2, 0);
}
```

迭代版

```
ListNode *addTwoNumbers(ListNode *l1, ListNode *l2)
{
    if(l1 == NULL && l2 == NULL) return NULL;
    if(l1 == NULL || l2 == NULL) return l1 == NULL ? l2 : l1;
    ListNode dummy(-1);
    ListNode * pre = &dummy;
    while(l1 && l2)
    {
        pre->next = new ListNode(l1->val + l2->val);
        pre = pre->next;
        l1 = l1->next;
        l2 = l2->next;
```

```
        }
        while(l1){
            pre->next = new ListNode(l1->val);
            pre = pre->next;
            l1 = l1->next;
        }
        while(l2){
            pre->next = new ListNode(l2->val);
            pre = pre->next;
            l2 = l2->next;
        }
        pre = dummy.next;
        while(pre){
            int t = pre->val;
            if(t >= 10) {
                pre->val = t % 10;
                if(pre->next)
                    pre->next->val += t / 10;
                else
                    {pre->next = new ListNode(t / 10); break;}
            }
            pre = pre->next;
        }
        return dummy.next;
    }
```

Divide Two Integers

题目来源：Divide Two Integers

> Divide two integers without using multiplication, division and mod operator.

解题思路：

只能用减法了，一个一个减比较慢。 注意考虑溢出的情况～

> In 2's complement systems, the absolute value of the most-negative value is out of range, e.g. for 32-bit 2's complement type int, INT_MIN is -2147483648, but the would-be result 2147483648 is greater than INT_MAX, which is 2147483647.

long的表达是4个字节～要用llabs才能AC。 cppreference 关于abs几个函数的说明在此.

```
int       abs( int n );
long      abs( long n );
long long abs( long long n );
(since C++11)
long       labs( long n );
long long llabs( long long n );
```

labs参数和返回值是long, oj 4个字节不够～ 用下面的或者用llabs都可以。

```
long long  m = abs((long long)divisor);
long long  n = abs((long long)dividend);
```

```
int divide(int dividend, int divisor)
{
   int result = 0;
   int sign = ( (dividend > 0 && divisor > 0) || (dividend < 0 && divisor < 0) ) ? 1 : -1
   //    long long  m = labs(divisor); this not work in leetcode's oj
 //   long long  n = labs(dividend);
   long long  m = llabs(divisor);
   long long  n = llabs(dividend);
   if(m == 1) return sign * n;
   if(m == n) return sign;
   if(n < m) return 0;
   while(n >= m)
   {
       int pow = 1;
       int mm = m;
       while(n >= mm && mm > 0) // mm > 0 important
       {
           n -= mm;
           result += pow;
           pow <<= 1;
           mm <<= 1;
       }
   }
   return sign * result;
}
```

Gray Code

题目来源：Gray Code

> The gray code is a binary numeral system where two successive values differ in only one bit. Given a non-negative integer n representing the total number of bits in the code, print the sequence of gray code. A gray code sequence must begin with 0. For example, given n = 2, return [0,1,3,2]. Its gray code sequence is: 00 - 0 01 - 1 11 - 3 10 - 2 Note: For a given n, a gray code sequence is not uniquely defined. For example, [0,2,3,1] is also a valid gray code sequence according to the above definition.
> For now, the judge is able to judge based on one instance of gray code sequence. Sorry about that.

解题思路：

逆序

注意观察，n每增加1，即是在n-1的结果之上，最高位加1，并按照n-1的逆序。

```
n = 1
0
1
n=2
0 0
0 1
---
1 1
1 0
n=3
0 0 0
0 0 1
0 1 1
0 1 0
------
1 1 0
1 1 1
1 0 1
1 0 0
```

```
    vector<int> grayCode(int n)
    {
        assert(n>=0);
        if(n == 0) return std::move(vector<int>(1,0));
        vector<int> result;
        result.push_back(0); result.push_back(1);
        for(int i = 1; i < n; i++)
        {
            int len = result.size();
            for(int j = len-1; j >=0; j--)
                result.push_back(result[j] + (1<<i));
        }
        return move(result);
    }
```

公式法

格雷码

```
G：格雷码  B：二进制码
G(N) = (B(n)/2) XOR B(n)
Binary Code(1011)要转换成Gray Code = (1011 >> 1) ^ 1011 = 1110
```

```
    vector<int> grayCode(int n)
    {
        assert(n>=0);
        int len = 1 << n;//std::pow(2,n);
        vector<int> result(len, 0);
        for(int i = 1; i < len; i++)
            result[i] = (i>>1)^i;
        return move(result);
    }
```

Integer to Roman

题目来源：Integer to Roman

> Given an integer, convert it to a roman numeral. Input is guaranteed to be within the range from 1 to 3999.

解题思路：

跟roman-to-integer一样，搞一个map对应关系，这个题目把一些特殊的比如4/9之类的也放进map里，然后遍历得到, 不然会搞得很复杂。 参考了Discuss.

```
string intToRoman(int num)
{
    string vs[] = {"I", "IV", "V", "IX", "X", "XL", "L", "XC", "C", "CD", "D", "CM", "M"};
    int ks[] = {1, 4, 5, 9, 10, 40, 50, 90, 100, 400, 500, 900, 1000};
    int len = sizeof(ks)/sizeof(int);

    int index = len-1;
    int value = num;
    string result = "";
    while(index >= 0 && value != 0)
    {
        if(value >= ks[index])
            result += vs[index], value -= ks[index];
        else
            index--;
    }
    return result;
}
```

Multiply Strings

题目来源：Multiply Strings

> Given two numbers represented as strings, return multiplication of the numbers as a string. Note: The numbers can be arbitrarily large and are non-negative.

解题思路：

大正整数乘法, 题目说了非负了。 跟 plus-one、 add-binary差不多。

```
string multiply(string num1, string num2)
{
    if(num1.length() * num2.length() == 0) return "";
    if(num1.length() < num2.length())
        return multiply(num2, num1);
    if(num1.length() == 1)
    {
        if(num1[0] == '0') return "0";
        if(num1[0] == '1') return num2;
    }
    if(num2.length() == 1)
    {
        if(num2[0] == '0') return "0";
        if(num2[0] == '1') return num1;
    }
    //num1.length > num2.length : num1 * num2
    int total_len = num1.length() + num2.length();
    int * result = new int[total_len];
    memset(result, 0, sizeof(int) * total_len);

    for(int i = num2.length()-1; i >=0; i--)
    {
        int n2 = num2[i] - '0';
        int index = total_len - (num2.length() - i);
        for(int j = num1.length()-1, jindex = 0; j >= 0; j--,jindex++)
        {
            int n1 = num1[j] - '0';
            int m = n2 * n1;
            result[index - jindex] += m;
        }
    }
    int index = total_len-1;
    while(index -1 >= 0)
    {
        int m = result[index];
        result[index] = m % 10;
        result[index-1] += (m / 10);
        index--;
    }
    string str = "";
    int i = 0;
    while(result[i] == 0) //skip first 0
        i++;
    while(i < total_len)
    {
        str += (result[i] + '0');
```

```
            i++;
        }
        delete [] result;
        return str;
    }
```

Palindrome Number

题目来源：Palindrome Number

> Determine whether an integer is a palindrome. Do this without extra space. Some hints: Could negative integers be palindromes? (ie, -1) If you are thinking of converting the integer to string, note the restriction of using extra space. You could also try reversing an integer. However, if you have solved the problem "Reverse Integer", you know that the reversed integer might overflow. How would you handle such case? There is a more generic way of solving this problem.

解题思路：

常量空间~ 关键得到长度,然后求reverse的数字，判断是否相等。

```
bool isPalindrome(int x)
{
   if (x < 0) return false;
   if (x < 10) return true;
   int len = 0;
   long long base = 10L;
   int xbak = x;
   while(x/base)
       base *= 10;
   base /= 10;
   x = xbak;
   long long reverse = 0;
   int base2 = 1;
   while(base)
   {
       reverse += (x / base % 10) * base2;
       base2 *= 10;
       base /= 10;
   }
   return reverse == x;
}
```

Plus One

题目来源：Plus One

> Given a non-negative number represented as an array of digits, plus one to the number. The digits are stored such that the most significant digit is at the head of the list.

解题思路：

直接从后往前加即可。

```
vector<int> plusOne(vector<int> &digits)
{
    int n = digits.size();
    if(n == 0) return digits;
    digits[n-1] += 1;
    for(int i = n-1; i >= 1; i--)
    {
        int t = digits[i];
        digits[i] = t % 10;
        if(t < 10) break;
        digits[i-1] += t / 10;
    }
    if(digits[0] < 10)
        return move(digits);
    else
    {
        digits.insert(digits.begin(), digits[0]/10);
        digits[1] = digits[1] % 10;
    }
    return move(digits);
}
```

Reverse Integer

题目来源：Reverse Integer

> Reverse digits of an integer. Example1: x = 123, return 321 Example2: x = -123, return -321 Have you thought about this? Here are some good questions to ask before coding. Bonus points for you if you have already thought through this! If the integer's last digit is 0, what should the output be? ie, cases such as 10, 100. Did you notice that the reversed integer might overflow? Assume the input is a 32-bit integer, then the reverse of 1000000003 overflows. How should you handle such cases? Throw an exception? Good, but what if throwing an exception is not an option? You would then have to re-design the function (ie, add an extra parameter).

解题思路：

按照 palindrome-number的思路，一样。

```
int reverse(int x)
{
  int sign = x > 0 ? 1 : -1;
  x = abs(x);
  long long base = 10L;
  int xbak = x;
  while(x/base)
      base *= 10;
  base /= 10;
  x = xbak;
  long long reverse = 0;
  int base2 = 1;
  while(base)
  {
      reverse += (x / base % 10) * base2;
      base2 *= 10;
      base /= 10;
  }
  return reverse * sign;
}
```

Roman to Integer

题目来源：Roman to Integer

> Given a roman numeral, convert it to an integer. Input is guaranteed to be within the range from 1 to 3999.

解题思路：

IV， I比V对应的数值小，结果就是result + （-I)，不然就是直接加 I.

学习下unordered_map的初始化

末尾加个0，让串都解析完毕。

```
int romanToInt(string s)
{
   unordered_map<char, int> kv({ {'I', 1}, {'V',5}, {'X',10}, {'L',50}, {'C',100}, {'D',5
   int result = 0;
   s += "0";
   for(int i = 0; i < s.length()-1; i++)
       if(kv[s[i]] < kv[s[i+1]])
           result -= kv[s[i]];
       else
           result += kv[s[i]];
   return result;
}
```

Valid Number

题目来源：Valid Number

> Validate if a given string is numeric. Some examples: "0" => true " 0.1 " => true "abc" => false "1 a" => false "2e10" => true Note: It is intended for the problem statement to be ambiguous. You should gather all requirements up front before implementing one.

解题思路：

粗暴方法

自己写的代码丑陋无比，一种情况一种情况试，实在是无参考价值。 主要是各种情况，例如：

input	result
.123	true
12.	True
47e+6	True
__1.__	True
.	False
46.e3	OJ里的 .2e81 true 0.e false 46.e3 true
+3	True
-.3	True
6e6.5	false

```
bool isNumber(const char *s)
{
    if(s == NULL) return false;
    if(*s == '\0') return false;
    while(*s == ' ')
        s++; //skip 空格
    if(*s == '\0') return false;
    if(*s == '-' || *s == '+') s++;
    bool firstnumber = false;
    if((*s >= '0' && *s <= '9') || *s == '.')
        firstnumber = true;
    if(! firstnumber) return false;
    bool has_e_before = false;
    bool has_dot_before = *s == '.';
    if(has_dot_before)
    {
        s++;
        if (!(*s >= '0' && *s <= '9')) return false; // "."
    }
    while(*s != '\0')
    {
        if(*s >= '0' && *s <= '9')
        {
            s++;
        }else if(*s == 'e')
```

```
            {
                if(has_e_before) return false;//.2e81  true  //0.e false  //46.e3 true
                has_e_before = true;
                s++;
                if(*s == '+' || *s == '-') s++; //005047e+6 ok
                if (!(*s >= '0' && *s <= '9')) return false; //"e." "e 1"
            }else if(*s == ' ')
            {
                while(*s != '\0' && *s == ' ')
                    s++;
                if(*s == '\0')//"  1.2  "
                    return true;
                return false; //" 2.3  3"
            }else if(*s == '.')
            {
                if(has_dot_before || has_e_before) return false;
                has_dot_before = true;
                s++;
            }else
            {
                return false;
            }
        }
        return true;
    }
```

整理下上面的代码，可以更好看些。

```
    bool isNumber(const char *s)
    {
        if(s == NULL ) return false;
        bool isNum = false; bool isDot = false; bool isExp = false;
        const char* end = s;
        while(*end != '\0') end++;
        --end;
        while(*end == ' ') --end;
        while(*s == ' ') s++;

        if(*s == '+')s++;
        else if(*s == '-')s++;
        while(s <= end)
        {
            if(*s>= '0' && *s <= '9'){
                isNum = true;
            }else if(*s == '.'){
                if (isDot || isExp) return false;
                isDot = true;
            }else if(*s == 'e'){
                if(isExp||!isNum) return false;
                isExp = true;
                isNum = false;//e 后面必须得有数字
            }else if(*s == '+'||*s == '-'){
                if(*(s-1) != 'e') return false;
            }else
                return false;
            s++;
        }
        return isNum;
    }
```

利用**strtod.**

利用函数**strtod**.

```
double      strtod( const char          *str, char          **str_end );
```

能够一步一步提取str中能够组成的double, str_end为提取后剩下的串。这里找了一份实现, 有兴趣的可以参考下 strtod的源码.

```
bool isNumber(const char *s)
{
    char * end;
    strtod(s, &end);
    if(end == s) return false; // "  "
    while(*end != '\0')
    {
        if(*end != ' ')
            return false;
        end++;
    }
    return true;
}
```

利用自动机

可参考 自动机实现valid-number.

```
注释一下本题分多少状态吧：
0初始无输入或者只有space的状态
1输入了数字之后的状态
2前面无数字，只输入了Dot的状态
3输入了符号状态
4前面有数字和有dot的状态
5'e' or 'E'输入后的状态
6输入e之后输入Sign的状态
7输入e后输入数字的状态
8前面有有效数输入之后，输入space的状态
共9种状态了，难设计的是6,7,8状态。
分好之后就好办了，设计出根据输入进行状态转换就OK了。
```

```
class Solution {
public:
     bool isNumber(const char *s) {
          enum InputType {
               INVALID,          // 0 Include: Alphas, '(', '&' ans so on
               SPACE,          // 1
               SIGN,          // 2 '+','-'
               DIGIT,          // 3 numbers
               DOT,               // 4 '.'
               EXPONENT,          // 5 'e' 'E'
          };
```

```
            int transTable[][6] = {
            //0INVA,1SPA,2SIG,3DI,4DO,5E
                  -1,  0,  3,  1,  2, -1,//0初始无输入或者只有space的状态
                  -1,  8, -1,  1,  4,  5,//1输入了数字之后的状态
                  -1, -1, -1,  4, -1, -1,//2前面无数字，只输入了Dot的状态
                  -1, -1, -1,  1,  2, -1,//3输入了符号状态
                  -1,  8, -1,  4, -1,  5,//4前面有数字和有dot的状态
                  -1, -1,  6,  7, -1, -1,//5'e' or 'E'输入后的状态
                  -1, -1, -1,  7, -1, -1,//6输入e之后输入Sign的状态
                  -1,  8, -1,  7, -1, -1,//7输入e后输入数字的状态
                  -1,  8, -1, -1, -1, -1,//8前面有有效数输入之后，输入space的状态
            };
            int state = 0;
            while (*s)
            {
                  InputType input = INVALID;
                  if (*s == ' ') input = SPACE;
                  else if (*s == '+' || *s == '-') input = SIGN;
                  else if (isdigit(*s)) input = DIGIT;
                  else if (*s == '.') input = DOT;
                  else if (*s == 'e' || *s == 'E') input = EXPONENT;
                  state = transTable[state][input];
                  if (state == -1) return false;
                  ++s;
            }
            return state == 1 || state == 4 || state == 7 || state == 8;
      }
};
```

string, 字符串处理相关

1. Anagrams 题解
2. Count and Say 题解
3. Evaluate Reverse Polish Notation 题解
4. Implement strStr() 题解.md)
5. Length of Last Word 题解
6. Longest Common Prefix 题解
7. Longest Palindromic Substring 题解
8. Longest Substring Without Repeating Characters 题解
9. Longest Valid Parentheses 题解
10. Minimum Window Substring 题解
11. Regular Expression Matching 题解
12. Reverse Words in a String 题解
13. Simplify Path 题解
14. Text Justification 题解
15. Valid Parentheses 题解
16. Wildcard Matching 题解
17. ZigZag Conversion 题解

Anagrams

题目来源：Anagrams

> Given an array of strings, return all groups of strings that are anagrams. Note: All inputs will be in lower-case.

解题思路：

变位词sort后是一样的，因此可用map存起来。

```
////[from discuss] single one is not counted yet!
vector<string> anagrams(vector<string> &strs)
{
    unordered_map<string, vector<string>> sortedKeyValue;
    for(int i = 0; i < strs.size(); i++)
    {
        string tmp = strs[i];
        std::sort(tmp.begin(), tmp.end());
        sortedKeyValue[tmp].push_back(strs[i]);
    }
    vector<string> result;
    auto it = sortedKeyValue.begin();
    for(; it != sortedKeyValue.end(); it++)
    {
        if((*it).second.size() <= 1) continue;
        std::copy((*it).second.begin(), (*it).second.end(), back_inserter(result));
    }
    return move(result);
}
```

Count and Say

题目来源：Count and Say

> The count-and-say sequence is the sequence of integers beginning as follows: 1, 11, 21, 1211, 111221, ...
1 is read off as "one 1" or 11. 11 is read off as "two 1s" or 21. 21 is read off as "one 2, then one 1" or 1211. Given an integer n, generate the nth sequence. Note: The sequence of integers will be represented as a string.

解题思路：

递归数数即可。

```
string to_str(int a)
{
    stringstream ss;
    ss << a;
    string result;
    ss >> result;
    return result;
}

string countAndSay(int n)
{
    if(n == 0) return "";
    const string str[] = {"1", "11", "21", "1211", "111221"};
    int num = sizeof(str) / sizeof(string);
    if(n-1 < num) return str[n-1];
    string last = countAndSay(n-1);
    int i = 0;
    int len = last.length();
    string result;
    while(i < len)
    {
        int count = 1;
        while(i+1 < len && last[i] == last[i+1])
            ++count, ++i;
        result += to_str(count);
        result += to_str(last[i]-'0');
        i++;
    }
    return result;
}
```

Evaluate Reverse Polish Notation

> Evaluate the value of an arithmetic expression in Reverse Polish Notation. Valid operators are +, -, *, /. Each operand may be an integer or another expression.

> Some examples:
>
> ["2", "1", "+", "3", "*] -> ((2 + 1) 3) -> 9
> ["4", "13", "5", "/", "+"] -> (4 + (13 / 5)) -> 6

解题思路：后缀表达式求值，用一个stack即可。 字符串转数字其实可以用库函数: int stoi (const string& str, size_t* idx = 0, int base = 10);

```
bool isOperator(string s){
    if(s.length() == 1 &&
       (s[0] == '+' || s[0] == '-' || s[0] == '*' || s[0] == '/'))
        return true;
    return false;
}
int toInt(string s){
    stringstream ss(s);
    int r;
    ss >> r;
    return r;
}
int evalRPN(vector<string> &tokens) {
    stack<int> op;
    for(int i = 0; i < tokens.size(); i++){
        string t = tokens[i];
        int op2 = 0; int op1 = 0;
        if(isOperator(t)){
            switch (t[0]){
                case '+':
                    assert(op.size()>=2);
                    op2 = op.top(); op.pop();
                    op1 = op.top(); op.pop();
                    op.push(op1+op2);
                    break;
                case '-':
                    assert(op.size()>=2);
                    op2 = op.top(); op.pop();
                    op1 = op.top(); op.pop();
                    op.push(op1-op2);
                    break;
                case '*':
                    assert(op.size()>=2);
                    op2 = op.top(); op.pop();
                    op1 = op.top(); op.pop();
                    op.push(op1*op2);
                    break;
                case '/':
                    assert(op.size()>=2);
                    op2 = op.top(); op.pop();
                    op1 = op.top(); op.pop();
                    assert(op2 != 0);
                    op.push(op1/op2);
                    break;
```

```
                default:
                    assert(false);
                    break;
            }
        }else{
            op.push(toInt(t));
        }
    }
    return op.top();
}
```

Length of Last Word

题目来源：Length of Last Word

> Given a string s consists of upper/lower-case alphabets and empty space characters ' ', return the length of last word in the string. If the last word does not exist, return 0. Note: A word is defined as a character sequence consists of non-space characters only. For example, Given s = "Hello World", return 5.

解题思路：

去掉首位的空格, 从最后往前走到第一个空格或开头停止。

```
int lengthOfLastWord(const char *s)
{
    char * start = const_cast<char*>(s);
    while(*start == ' ' && *start != '\0')
        start++;
    if(*start == '\0') return 0;
    char * end = start;
    while(*(end+1) != '\0')
        end++;
    while(*end == ' ') --end;
    int len = 0;
    while(end >= start && *end !=' ')
        ++len, --end;
    return len;
}
```

这个思路用STL写就简单了, ref。 std::ptr_fun 模版取函数指针.

```
    int lengthOfLastWord(const char *s)
    {
        if(s == NULL || *s == '\0') return 0;
        string str(s);
        auto start = std::find_if(str.rbegin(), str.rend(), std::ptr_fun<int, int>(std::i
        auto last = std::find_if_not(start, str.rend(), std::ptr_fun<int, int>(std::isalp
        return std::distance(start, last);
    }
```

直接从前往后，记录每一个单词的长度，后面的单词长度会取代前面的长度，注意得跳过中间的连续空格。

```
    int lengthOfLastWord(const char *s)
    {
        if(s == NULL || *s == '\0') return 0;
        while(*s != '\0' && *s == ' ') s++;
        if(*s == '\0') return 0;
        int len = 0;
        while(*s != '\0')
```

```
        {
            if(*s != ' ')
                len++;
            else
            {
                while(*s == ' ')
                    s++;
                if(*s == '\0')
                    break;
                len = 1; //*s != ' '
            }
            s++;
        }
        return len;
    }
```

Longest Common Prefix

题目来源：Longest Common Prefix

> Write a function to find the longest common prefix string amongst an array of strings.

解题思路：

从前往后一个一个对比就是。

```
string longestCommonPrefix(vector<string> &strs)
{
   if (strs.size() == 0) return "";
   if (strs.size() == 1) return strs[0];
   if (strs[0].length() == 0) return "";
   int size = 0;
   while(size < strs[0].length())
   {
       char ch = strs[0][size];
       for(int i = 1; i < strs.size(); i++)
           if(size >= strs[i].length() || strs[i][size] != ch)
               return strs[0].substr(0, size);
       size++;
   }
   return strs[0].substr(0, size);
}
```

Longest Palindromic Substring

题目来源：Longest Palindromic Substring

> Given a string S, find the longest palindromic substring in S. You may assume that the maximum length of S is 1000, and there exists one unique longest palindromic substring.

解题思路：

暴力搜索, \(O(N^2) \)

最简单的方法就是选中i(0~n-1)，然后向两边扩展，复杂度为\(O(N^2) \) . 注意回文长度可能是奇数或者偶数， 即 `aba or abba`

```
string longestPalindrome(string s)
{
  int n = s.length();
  if (n <= 1) return s;
  int start = 0; int maxLen = 1;
  for(int i = 0; i < n; i++)
  {
      //center: i
      int left = i - 1;
      int right = i + 1;
      while(left >= 0 && right < n
          && s[left] == s[right])
          --left, ++right;
      //s[left] != s[right] , s[left+1 : right-1] is palindrome
      int len = (right-1) - (left+1) + 1;
      if(len > maxLen)
          start = left+1, maxLen = len;
      //center: between s[i] and s[i+1]
      if(i+1 < n && s[i] == s[i+1])
      {
          left = i;
          right = i+1;
          while(left >= 0 && right < n
              && s[left] == s[right])
              --left, ++right;
          len = (right-1) - (left+1) + 1;
          if(len > maxLen)
              start = left+1, maxLen = len;
      }
  }
  return s.substr(start, maxLen);
}
```

或者可以这样，将中间相同的字符跳过，不考虑奇数还是偶数的串(其实还是考虑了，也包括在内了)。

```
abbbbbbbbba, left=1时,right一直走到最后一个b, 然后往两边判断是否相等.
```

```
string longestPalindrome(string s)
{
```

```
    int n = s.length();
    if (n <= 1) return s;
    int start = 0; int maxLen = 1;
    for(int i = 0; i < n; i++)
    {
        int left = i;
        int right = i;
        while(right+1 < n && s[right+1] == s[left])
            ++right;
        //s[right+1] != s[left], s[right]=s[left]
        i = right; //skip the same char
        //a[bbbbbbbb]a
        while(left-1 >= 0 && right+1 < n
            && s[left-1] == s[right+1])
            --left, ++right;
        //s[left : right] is palindrome
        int len = right - left + 1;
        if(len > maxLen)
            start = left, maxLen = len;
    }
    return s.substr(start, maxLen);
}
```

DP, \(O(N^2) \)

dp[i][j] 表示 s[i:j] 是回文, 当且尽当 `s[i] == [j] && dp[i+1][j-1]` , 即计算dp[i][j]时, dp[i+1][j-1]得先计算出来， 算dp[x][i]，必须先把dp[x][i-1]先计算出来了来。

```
string longestPalindrome(string s)
{
    int n = s.length();
    if (n <= 1) return s;
    int start = 0; int maxLen = 1;
    //vector<vector<bool> > dp(n, vector<bool>(n, false));
    bool dp[1000][1000] = {false};
    for(int i = 0; i < n; i++)
        dp[i][i] = true;
    //dp[j][i]: s[j:i] is palindrome, dp[j+1][i-1] true + s[j]==s[i]
    for(int i = 1; i < n; i++)
        for(int j = 0; j < i; j++)
        {
            if(s[i] == s[j] && (j+1 > i-1 || dp[j+1][i-1]))
            {
                dp[j][i] = true;
                int len = i - j + 1;
                if(len > maxLen)
                    maxLen = len, start = j;
            }
        }
    return s.substr(start, maxLen);
}
```

计算s[:5]的结果，先得把所有s[:4]结尾的回文算出来了来。 上面比如在算dp[i][5]时，用到了dp[x][4]，在上面的循环中,dp[x][4]已经算出来了的。 另外，虽然都是平方的算法，上面用vector还过不了，用数组才能过。

\(O(n) \) 算法, Manacher 算法

felix021的文章讲得很清楚，这里“偷”过来。

>

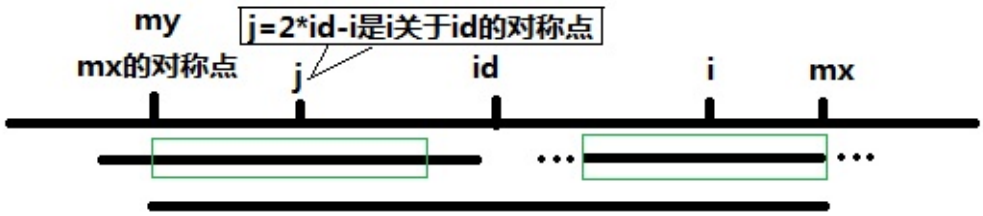
首先用一个非常巧妙的方式，将所有可能的奇数/偶数长度的回文子串都转换成了奇数长度：在每个字符的两边都插入一个

下面以字符串12212321为例，经过上一步，变成了 S[] = "$#1#2#2#1#2#3#2#1#";

然后用一个数组 P[i] 来记录以字符S[i]为中心的最长回文子串向左/右扩张的长度（包括S[i]，也就是把该回文串“对
```
S  #  1  #  2  #  2  #  1  #  2  #  3  #  2  #  1  #
P  1  2  1  2  5  2  1  4  1  2  1  6  1  2  1  2  1
```
(p.s. 可以看出，P[i]-1正好是原字符串中回文串的总长度)
那么怎么计算P[i]呢？该算法增加两个辅助变量（其实一个就够了，两个更清晰）id和mx，其中id表示最大回文子串中心

然后可以得到一个非常神奇的结论，这个算法的关键点就在这里了：如果mx > i，那么P[i] >= MIN(P[2 * id - i]
```
//记j = 2 * id - i，也就是说 j 是 i 关于 id 的对称点。
if (mx - i > P[j])
    P[i] = P[j];
else /* P[j] >= mx - i */
    P[i] = mx - i; // P[i] >= mx - i，取最小值，之后再匹配更新。
```
当然光看代码还是不够清晰，还是借助图来理解比较容易。
当 mx - i > P[j] 的时候，以S[j]为中心的回文子串包含在以S[id]为中心的回文子串中，由于 i 和 j 对称，以S

当 P[j] >= mx - i 的时候，以S[j]为中心的回文子串不一定完全包含于以S[id]为中心的回文子串中，但是基于对称



对于 mx <= i 的情况，无法对 P[i]做更多的假设，只能P[i] = 1，然后再去匹配了。

代码如下：

```
//Manacher O(n)
//ref1: http://leetcode.com/2011/11/longest-palindromic-substring-part-ii.html
//ref2: http://blog.csdn.net/pickless/article/details/9040293
//ref3: http://www.felix021.com/blog/read.php?2040
string longestPalindrome(string s)
{
   int len = (int)s.length() * 2 + 2;
   string news(len+1, ' ');
```

```
    news[0]='$';
    for(int i = 0; i < s.length(); i++)
    {
        news[2 * i + 1] = '#';
        news[2 * i + 2] = s[i];
    }
    news[len-1] = '#';
    news[len] = '\0';
    vector<int> p(len, 0);
    int mx = 0, id = 0;
    for(int i = 1; i < len; i++)
    {
        p[i] = mx > i ? min(p[2*id-i], mx-i) : 1;
        while(news[i + p[i]] == news[i - p[i]]) ++p[i];
        if(i + p[i] > mx)
        {
            mx = i + p[i];
            id = i;
        }
    }
    int maxLen = 0;
    int center = 0;
    for(int i = 0; i < len; i++)
    {
        if(p[i] > maxLen)
        {
            maxLen = p[i];
            center = i;
        }
    }
    return s.substr((center)/2 - maxLen/2, maxLen-1);
}
```

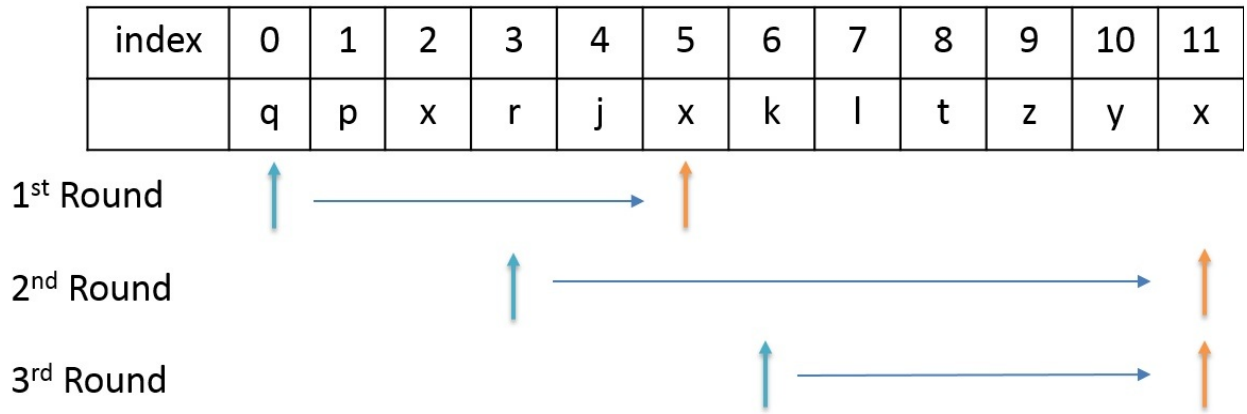
Longest Substring Without Repeating Characters

题目来源：Longest Substring Without Repeating Characters

> Given a string, find the length of the longest substring without repeating characters. For example, the longest substring without repeating letters for "abcabcbb" is "abc", which the length is 3. For "bbbbb" the longest substring is "b", with the length of 1.

解题思路：

记录上一次出现同字符到当前字符的长度，如下图。




```
int lengthOfLongestSubstring(string s)
{
   if(s.length() <= 1) return s.length();
   vector<int> table(256, -1);
   int start = 0;
   int result = 0;
   for(int i = 0; i < s.length(); i++)
   {
       if(table[s[i]] >= start)
           start = table[s[i]]+1;
       result = std::max(result, i-start+1);
       table[s[i]]=i;
   }
   return result;
}
```

注意数组初始化，`int table[256]={-1}; //只有第一个为-1，其他为0.`

我不会告诉你我参考了这篇文章的.

Longest Valid Parentheses

题目来源：Longest Valid Parentheses

> Given a string containing just the characters '(' and ')', find the length of the longest valid (well-formed) parentheses substring. For "(()", the longest valid parentheses substring is "()", which has length = 2. Another example is ")()())", where the longest valid parentheses substring is "()()", which has length = 4.

解题思路：

找连续合法的括号对数。

O(2*N)

1、用一个数组记录每个括号的配对状态，借助stack找配对的index，最后再扫描一遍，找连续配对的数量max.ref1.

```
int longestValidParentheses(string s)
{
    stack<int> left;
    vector<bool> match(s.length(), false);
    for(int i = 0; i < s.length(); i++)
    {
        if(s[i] == '(')
            left.push(i);
        else//')'
        {
            if(! left.empty())
            {
                match[left.top()] = true; left.pop();
                match[i] = true;
            }//else default false
        }
    }
    int result = 0;
    for(int i = 0; i < s.length(); i++)
    {
        int len = 0;
        while(i < s.length() && match[i]) i++, len++;
        result = std::max(result, len);
    }
    return result;
}
```

O(N)

用last记录上一个还没配对的右括号")", 用一个栈记录下"("的index, 遇到")", 配对时pop掉，记录其长度, pop完时，长度为当前 `index-last` , 没完时，长度为当前 `index-stack.top()` . ref2

例如:

```
`)(()())` last=0, index=3时, pop(),len=3-left.top()=2,
```

index=5时, pop(),len=5-left.top()=4,
index=6时, empty() len=6-last=6.

```
    int longestValidParentheses(string s)
    {
        stack<int> left;
        int result = 0;
        int last = -1;
        for(int i = 0; i < s.length(); i++)
        {
            if(s[i] == '('
                left.push(i);
            else//')'
            {
                if(left.empty())
                    last = i;
                else
                {
                    left.pop();
                    int len = 0;
                    if(left.empty())
                        len = i - last;
                    else
                        len = i - left.top();
                    result = std::max(result, len);
                }
            }
        }
        return result;
    }
```

Minimum Window Substring

题目来源：Minimum Window Substring

> Given a string S and a string T, find the minimum window in S which will contain all the characters in T in complexity O(n). For example, S = "ADOBECODEBANC" T = "ABC" Minimum window is "BANC". Note: If there is no such window in S that covers all characters in T, return the emtpy string "". If there are multiple such windows, you are guaranteed that there will always be only one unique minimum window in S.

解题思路：

始终先找到一个包含所有T中的串，记录其window长度，然后继续往后扫面，到第二个合法的window的时候，前面begin指针往后，删除window中冗余的，期间比较window的size，记录一个最小的。有了一个window之后，后面的变化始终保持window中包含一个合法的T。 用has[]表示找到的相应字符的数量，need[]表示T中应该有的数量。 例如：

```
S= acbbaca, T = aba
begin,end扫描，[acbba]ca 第一个合法window后，end继续
到[acbbaca]发现第二个合法，begin此时指向了a，而此时has[a]=3>need[a]=2，a冗余，后移begin，has[a]--
ac[bbaca]: c不包含在T中，直接略过，到b，has[b]=2>need[b]=1,b冗余，继续后移，并has[b]—;
acb[baca]: has[b]=need[b]不能减少了，记录这个短的window,并于当前比较。
其中end/start都只+不-，复杂度为O(n).
```

代码如下：

```
string minWindow(string S, string T)
{
    if(S.length() < T.length()) return "";
    vector<int> need(256, 0);
    for(int i = 0; i < T.length(); i++)
        need[T[i]]++;
    vector<int> has(256, 0);
    int count = 0;
    int windowStart = 0;
    int minLen = INT_MAX;
    int minStart = 0;
    for(int windowEnd = 0; windowEnd < S.length(); windowEnd++)
    {
        if(need[S[windowEnd]] == 0) continue;//skip
        if(has[S[windowEnd]] < need[S[windowEnd]])
            ++count;
        ++has[S[windowEnd]];
        if(count == T.length())//a window found
        {
            while(windowStart < windowEnd)
            {
                if(need[S[windowStart]] == 0)//skip
                {
                    ++windowStart;
                }else if(has[S[windowStart]] > need[S[windowStart]])
                {
```

```
                        --has[S[windowStart]];
                        ++windowStart;
                    }else
                    {
                        break;
                    }
                }

                int len = windowEnd - windowStart + 1;
                if(len < minLen)
                {
                    minLen = len;
                    minStart = windowStart;
                }
            }
        }
        if(minLen == INT_MAX) return "";
        return S.substr(minStart, minLen);
    }
```

Ref

Regular Expression Matching

题目来源：Regular Expression Matching

> Implement regular expression matching with support for '.' and '*'. '.' Matches any single character. * Matches zero or more of the preceding element. The matching should cover the entire input string (not partial). The function prototype should be: bool isMatch(const char *s, const char *p) Some examples: isMatch("aa","a") → false isMatch("aa","aa") → true isMatch("aaa","aa") → false isMatch("aa", "a*") → true isMatch("aa", ".*") → true isMatch("ab", ".*") → true isMatch("aab", "c*a*b") → true

解题思路：

注意理解请题意，* 和前面的字符是一个整体, c* 可以表示'', 'c','ccccc' 等。'.*' 表示 '[.....]'

```
bool isMatch(const char *s, const char *p)
{
    if(s == NULL && p == NULL) return true;
    if(s == NULL || p == NULL) return false;
    if(*p == '\0') return *s == '\0';
    if(*(p+1) == '*')
    {
        //.* ----> .[.....]
        while(*s == *p || (*s != '\0' && *p == '.'))
        {
            if(isMatch(s, p+2))
                return true;
            ++s;
        }
        return isMatch(s, p+2);
    }else if(*s == *p || (*s != '\0' && *p == '.'))
        return isMatch(s+1, p+1);
    return false;
}
```

Reverse Words in a String

> Given an input string, reverse the string word by word. For example, Given s = "the sky is blue", return "blue is sky the".

解题思路：类似矩阵转置的操作 (AB)^t = B^tA^t

例如 “hello world”.reverse = “dlrow olleh” 然后针对每一个word进行reverse，拼接而成得到”world hello”

注意的地方:

1. 全是空格的情况；
2. 中间单词间隔多个空格；
3. reverse最后一个单词。

```
void reverseWords(string &s)
{
    int start = 0;
    while(s[start] == ' ') start++;
    int end = (int)s.length() - 1;
    while(s[end] == ' ') end--;
    if(end < start)//1.
    {
        s = "";
        return;
    } //if all blank
    s = s.substr(start, end-start+1); //trim
    std::reverse(s.begin(), s.end());
    s += ' '; //3. for the last word
    string result;
    start = 0;
    int i = 0;
    while(i < s.length())
    {
        if(s[i] == ' '){
            string tmp = s.substr(start, i-start+1);//tmp including '_'(blank)
            std::reverse(tmp.begin(), tmp.end());//'_word1_word2'
            result += tmp;
            while(i < s.length() && s[i] == ' ') i++; //2.
            if(i == s.length())
                break;
            start = i;
        }else
            i++;
    }
    s = result.substr(1, result.length()-1);
}
```

从discuss里面还看到了简短的代码～值得学习。 通过stringstream 一次提取一个单词出来, 然后将这个单词与上一次的结果连接(逆序)。

```
void reverseWords2(string &s)
{
    stringstream ss(s);
    string tmp = "";
```

```
    string result = "";
    while(ss >> tmp)
    {
        tmp += " ";
        tmp += result;
        result = tmp;
    }
    s = result.substr(0, result.length()-1);
}
```

Simplify Path

题目来源：Simplify Path

> Given an absolute path for a file (Unix-style), simplify it. For example, path = "/home/", => "/home" path = "/a/./b/../../c/", => "/c" Corner Cases: Did you consider the case where path = "/../"? In this case, you should return "/". Another corner case is the path might contain multiple slashes '/' together, such as "/home//foo/". In this case, you should ignore redundant slashes and return "/home/foo".

解题思路：

把"//"替换成"/"然后split一下，用stack记录，".."之类的就去掉。 这个Java写起来方便些。

主要就是一些testcase能否想到。

```
public  String simplifyPath(String path)
{
    if(path == null || path.length() <= 1) return path;
    //assuming always start with "/", //skip the first "/"
    if(path.charAt(path.length()-1) == '/' )
        path = path.substring(1, path.length()-1);
    path = path.replaceAll( "//", "/" );
    String[] strs = path.split( "/");
    Stack<String> ss = new Stack<String>();
    for (int i = 0; i < strs.length; i++)
    {
        if (strs[i].equals("." ))
            continue;
        if (strs[i].equals(".." ))
        {
            if (ss.size() > 0)
                ss.pop();
        } else
        {
            if(strs[i].length()>0)
                ss.push(strs[i]);
        }
    }
    Stack<String> ss2 = new Stack<String>();
    while(ss.size()>0)
    {
        ss2.push(ss.pop());
    }
    String result = "";
    if(ss2.size() == 0) return "/";
    while(ss2.size() > 0)
        result += "/" + ss2.pop();
    return result;
}
```

C++ 一样的。

```
string simplifyPath(string path)
{
```

```
        vector<string> splits;
        int start = 0;
        int i = 0;
        path += "/"; //make sure the last segment can put into splits.
        while(i < path.length())
        {
            if(path[i] == '/')
            {
                if(i - start > 0){
                    splits.push_back(path.substr(start, i-start)); //no including the cur
                    start = i;
                }
                while(i+1 < path.length() && path[i+1] == '/') // ignore remain '/' of "/
                {
                    i += 1;
                    start = i;
                }
            }
            ++i;
        }
        vector<string> result;
        for(int i = 0; i < splits.size(); i++)
            if(splits[i] == "/..")
            {
                if (result.size()>0)
                    result.pop_back();
            }
            else if(splits[i] == "/.")
                ;
            else
                result.push_back(splits[i]);
        string str_result = "";
        for(int i = 0; i < result.size(); i++)
            str_result += result[i];
        if(str_result.length() == 0)
            return "/";
        return str_result;
    }
```

Text Justification

题目来源：Text Justification

> Given an array of words and a length L, format the text such that each line has exactly L characters and is fully (left and right) justified. You should pack your words in a greedy approach; that is, pack as many words as you can in each line. Pad extra spaces ' ' when necessary so that each line has exactly L characters. Extra spaces between words should be distributed as evenly as possible. If the number of spaces on a line do not divide evenly between words, the empty slots on the left will be assigned more spaces than the slots on the right. For the last line of text, it should be left justified and no extra space is inserted between words. For example, words: ["This", "is", "an", "example", "of", "text", "justification."] L: 16.
Return the formatted lines as: ["This is an", "example of text", "justification. "] Note: Each word is guaranteed not to exceed L in length. Corner Cases: A line other than the last line might contain only one word. What should you do in this case? In this case, that line should be left-justified.

解题思路：

一步一步来即可。注意最后一行需要特殊处理。最后一行的空格尽量留在行末，而其他行是多余的空格尽量分布在行首。

```
vector<string> fullJustify(vector<string> &words, int L)
{
    if(words.size() == 0 ) return vector<string>();
    if(words.size() == 1 && words[0].length() == L) {return vector<string>(words);}
    vector<string> result;
    int curIndex = 0;
    int lastIndex = 0;
    int curLen = 0;
    int realwordLen = 0;
    while(curIndex < words.size())
    {
        while(curIndex < words.size() && curLen + words[curIndex].length() <= L)
        {
            curLen += words[curIndex].length();
            realwordLen += words[curIndex].length();
            if(curLen != L)
                ++curLen;//blank
            ++curIndex;
        }
        //curLen + words[curIndex].length() > L, can not use curIndex
        {
            int blank = L - realwordLen;
            int wordCount = curIndex - lastIndex;
            int eachblank = wordCount == 1 ? blank : blank / (wordCount-1);
            int moreblank = wordCount == 1 ? 0 : blank % (wordCount-1);
            string line;
            string eachblankstr;
            for(int i = 0; i < eachblank; i++)
                eachblankstr += ' ';
            for(int j = 0, i = lastIndex; i < curIndex; i++,j++)
            {
                line += words[i];
                if(wordCount == 1 || i != curIndex-1) //last one
```

```
                    {
                        line += eachblankstr;
                        if(j < moreblank)
                            line += ' ';
                    }
                }
                result.push_back(line);
                lastIndex = curIndex;
                curLen = realwordLen = 0;
            }
        }
        //special deal with last line
        string lastline = result[result.size() - 1];
        string lastlineshould(lastline.size(), ' ');
        bool blank = true;
        int i = 0; int j = 0;
        while(i < L)
        {
            if(lastline[i] != ' ')
            {
                lastlineshould[j++] = lastline[i];
                blank = true;
            }
            else if(blank)
            {
                blank = false;
                lastlineshould[j++] = ' ';
            }
            ++i;
        }
        result[result.size()-1] = lastlineshould;
        return move(result);
    }
```

Valid Parentheses

题目来源：Valid Parentheses

> Given a string containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid. The brackets must close in the correct order, "()" and "()[]{}" are all valid but "(]" and "([)]" are not.

解题思路：

用stack.

```
bool isValid(string s)
{
   stack<char> ss;
   for(int i = 0; i < s.length(); i++)
   {
       switch(s[i])
       {
           case '(':
           case '[':
           case '{':
               ss.push(s[i]);
               break;
           case ')':
               if(!ss.empty() && ss.top() == '(')
                   ss.pop();
               else
                   return false;
               break;
           case ']':
               if(!ss.empty() && ss.top() == '[')
                   ss.pop();
               else
                   return false;
               break;
           case '}':
               if(!ss.empty() && ss.top() == '{')
                   ss.pop();
               else
                   return false;
               break;
           default:
               return false;
       }
   }
   return ss.empty();
}
```

Wildcard Matching

题目来源：Wildcard Matching

> Implement wildcard pattern matching with support for '?' and '*'. *'?' Matches any single character. "* Matches any sequence of characters (including the empty sequence). The matching should cover the entire input string (not partial). The function prototype should be: bool isMatch(const char *s, const char* p) Some examples: isMatch("aa","a") → false isMatch("aa","aa") → true isMatch("aaa","aa") → false isMatch("aa", "*") → true isMatch("aa", "a*") → true isMatch("ab", "?*") → true isMatch("aab", "c*a*b") → false

解题思路：

跟 regular-expression-matching 类似。

递归

主要是考虑 “*” 匹配任意字符的问题， 下面代码超时了。

```
bool isMatch(const char *s, const char *p)
{
    if(s == NULL && p == NULL) return true;
    if(s == NULL || p == NULL) return false;
    if(*s == '\0') return *p == '\0' || (*p == '*' && *(p+1) == '\0');
    if(*p == '\0') return *s == '\0';
    if(*p == '*')
    {
        while(*(p+1) == '*') p++;//skip all continous *
        while(*s != '\0')
        {
            if(isMatch(s, p+1))
                return true;
            s++;
        }
        return *(p+1) == '\0';
    }
    while(*s != '\0' && *p != '\0' && (*s == *p || *p == '?'))
    {
        s++; p++;
    }
    if(*s == '\0' && *p == '\0') return true;
    return false;
}
```

迭代

Key point, compare char one by one, if not matched, and '*' matched before, then pattern backtrace to '*', and string backtrace to the later one of compared char of last iterative time. 参考了 discuss.leetcode.

```
bool isMatch(const char *s, const char *p)
{
    if(s == NULL && p == NULL) return true;
    if(s == NULL || p == NULL) return false;
    if(*s == '\0') return *p == '\0' || (*p == '*' && *(p+1) == '\0');
```

```
    if(*p == '\0') return *s == '\0';
    const char *star_p=NULL,*star_s=NULL;
    while(*s)
    {
        if(*p == '?' || *p == *s)
        {
            ++p,++s;
        }else if(*p == '*')
        {
            //skip all continuous '*'
            while(*p == '*') ++p;

            if(!*p) return true; //if end with '*', its match.

            star_p = p; //star_p is later char of '*',
            star_s = s; //store '*' pos for string
        }else if((!*p || *p != *s)  && star_p)
        {
            s = ++star_s; //skip non-match char of string, regard it matched in '*'
            p = star_p; //pattern backtrace to later char of '*'
        }else
            return false;
    }
    //check if later part of p are all '*'
    while(*p)
        if(*p++ != '*')
            return false;
    return true;
}
```

ZigZag Conversion

题目来源：ZigZag Conversion

> The string "PAYPALISHIRING" is written in a zigzag pattern on a given number of rows like this: (you may want to display this pattern in a fixed font for better legibility) P A H N A P L S I I G Y I R And then read line by line: "PAHNAPLSIIGYIR" Write the code that will take a string and make this conversion given a number of rows: string convert(string text, int nRows); convert("PAYPALISHIRING", 3) should return "PAHNAPLSIIGYIR".

解题思路：

对勾的形式一个周期～hash一下到具体哪一个row～

```
0           0           0
1       5   1       5
2   4       2   4
3           3
```

```
string convert(string s, int nRows)
{
   if(nRows == 1) return s;
   vector<vector<char> > rows(nRows);
   int T = nRows + nRows - 2;
   for(int i = 0; i < s.length(); i++)
   {
       int bucket = i % T;
       bucket = bucket >= nRows ? (T - bucket) : bucket;
       rows[bucket].push_back(s[i]);
   }
   string result="";
   for(int i = 0; i < nRows; i++)
   {
       for(int j = 0; j < rows[i].size(); j++)
           result += rows[i][j];
   }
   return result;
}
```

combination and permutation, 排列组合相关

1. Combinations 题解
2. Combination Sum 题解
3. Combination Sum II 题解
4. Letter Combinations of a Phone Number 题解
5. Next Permutation 题解
6. Palindrome Partitioning 题解
7. Permutation Sequence 题解
8. Permutations 题解
9. Permutations II 题解
10. Subsets 题解
11. Subsets II 题解
12. Unique Paths 题解

Combinations

题目来源：Combinations

> Given two integers n and k, return all possible combinations of k numbers out of 1 ... n. For example, If n = 4 and k = 2, a solution is:

```
[
  [2,4],
  [3,4],
  [2,3],
  [1,2],
  [1,3],
  [1,4],
]
```

解题思路：

```
    void search(vector<vector<int> > &result, vector<int> &input, int start, int k, int n
    {
        if(input.size() == k){
            result.push_back(input);
            return;
        }
        for(int i = start; i <= n; i++){
            input.push_back(i);
            search(result, input, i+1, k, n);
            input.pop_back();
        }
    }
    vector<vector<int> > combine(int n, int k)
    {
        vector<vector<int> > result;
        assert(k <= n);
        vector<int> input;
        search(result, input, 1, k, n);
        return move(result);
    }
```

Combination Sum

题目来源：combination Sum

> Given a set of candidate numbers (C) and a target number (T), find all unique combinations in C where the candidate numbers sums to T. The same repeated number may be chosen from C unlimited number of times. Note: All numbers (including target) will be positive integers. Elements in a combination (a1, a2, … , ak) must be in non-descending order. (ie, a1 ≤ a2 ≤ … ≤ ak). The solution set must not contain duplicate combinations. For example, given candidate set 2,3,6,7 and target 7, A solution set is: [7] [2, 2, 3]

解题思路：

深搜, 排序后一个一个往里面加，注意是组合，比如当前加到2了，2还可以加，但2以前的就不能加了。

```
void dfs(vector<vector<int> > &result, vector<int> &path, const vector<int> &num, int
{
    if(sum > target) return;
    if(sum == target)
    {
        result.push_back(path);
        return;
    }
    for(int i = 0 ; i < num.size(); i++)
    {
        if(sum < target && (path.size() == 0 || num[i] >=path[path.size()-1]))
        {
            sum += num[i];
            path.push_back(num[i]);
            dfs(result, path, num, sum, target);
            path.pop_back();
            sum -= num[i];
        }
    }
}

vector<vector<int> > combinationSum(vector<int> &candidates, int target)
{
    vector<vector<int> > result;
    if(candidates.size() == 0) return result;
    std::sort(candidates.begin(), candidates.end());
    vector<int> path;
    dfs(result, path, candidates, 0, target);
    return move(result);
}
```

Combination Sum II

题目来源：combination Sum II

> Given a collection of candidate numbers (C) and a target number (T), find all unique combinations in C where the candidate numbers sums to T. Each number in C may only be used once in the combination. Note: All numbers (including target) will be positive integers. Elements in a combination (a1, a2, … , ak) must be in non-descending order. (ie, a1 ≤ a2 ≤ … ≤ ak). The solution set must not contain duplicate combinations. For example, given candidate set 10,1,2,7,6,1,5 and target 8, A solution set is: [1, 7] [1, 2, 5] [2, 6] [1, 1, 6]

解题思路：

跟 combination-sum 一样, DFS，在它的基础上，将用过的跳过即可。重复的数字，后面的要跳过，不然结果有重复的。

```
void dfs(vector<vector<int> > &result, vector<int> &path, const vector<int> &num, int
{
    if(sum > target) return;
    if(sum == target)
    {
        result.push_back(path);
        return;
    }
    for(int i = start ; i < num.size(); i++)
    {
        if(sum < target)
        {
            sum += num[i];
            path.push_back(num[i]);
            dfs(result, path, num, sum, target, i+1);
            path.pop_back();
            sum -= num[i];
        }
        while(i+1 < num.size() && num[i] == num[i+1]) i++; ////ignore the next same o
    }
}

vector<vector<int> > combinationSum2(vector<int> &num, int target)
{
     vector<vector<int> > result;
    if(num.size() == 0) return result;
    std::sort(num.begin(), num.end());
    vector<int> path;
    dfs(result, path, num, 0, target, 0);
    return move(result);
}
```

Letter Combinations of a Phone Number

题目来源：Letter Combinations of a Phone Number

> Given a digit string, return all possible letter combinations that the number could represent. A mapping of digit to letters (just like on the telephone buttons) is given below. 

> Input:Digit string "23" Output: ["ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"]. Note: Although the above answer is in lexicographical order, your answer could be in any order you want.

解题思路：

排列组合问题.

```
void dfs(vector<string> &result, string &str, int start, const string &input, const vecto
{
    if (start == input.length())
    {
        result.push_back(str);
        return;
    }
    int num = input[start] - '0';
    for(int i = 0; i < numMap[num].size(); i++)
    {
        str[start] = numMap[num][i];
        dfs(result, str, start+1, input, numMap);
    }
}

vector<string> letterCombinations(string digits)
{
    vector<string> numMap({"","","abc","def","ghi","jkl","mno","pqrs","tuv","wxyz"});
    vector<string> result;
    int n = digits.length();
    if (n == 0) return vector<string>(1, "");//special case in oj
    string str(n, '.');
    dfs(result, str, 0, digits, numMap);
    return move(result);
}
```

Next Permutation

题目来源：Next Permutation

> Implement next permutation, which rearranges numbers into the lexicographically next greater permutation of numbers. If such arrangement is not possible, it must rearrange it as the lowest possible order (ie, sorted in ascending order). The replacement must be in-place, do not allocate extra memory. Here are some examples. Inputs are in the left-hand column and its corresponding outputs are in the right-hand column. 1,2,3 → 1,3,2 3,2,1 → 1,2,3 1,1,5 → 1,5,1

解题思路：

这里 permutations-ii其实已经实现过一次了。

```
void nextPermutation(vector<int> &num)
{
    int n = num.size();
    int index = n - 1;//2 [3] 5 4 1
    while(index-1 >= 0 && num[index-1] >= num[index])
        --index;
    if(index == 0)
    {
        std::reverse(num.begin(), num.end());
        return;
    }
    --index; //[3]
    int bigger = n-1; //find[4]
    while(bigger != index && num[bigger] <= num[index])
        --bigger;
    std::swap(num[index], num[bigger]);
    std::reverse(num.begin()+index+1, num.end());
}
```

Palindrome Partitioning

题目来源：Palindrome Partitioning

>

```
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"]
]
```

解题思路：

直接暴力解决

枚举每种可能，去判读是否回文。跟排列组合算法一样。 还可以优化，把中间的某个子串是否回文用hash缓存下来。

```
    bool isPalindrome(string s)
    {
        int n = s.length();
        if(n <= 1) return true;
        int left = 0; int right = n-1;
        while(left < right)
        {
            if(s[left] == s[right])
            {
                left++;
                right--;
            }else
                return false;
        }
        return true;
    }
    void search(vector<string> &path, int start, string s, vector<vector<string> > &resul
    {
        int n = s.length();
        if(start > n) return;
        if(start == n)
        {
            result.push_back(path);
            return;
        }
        for(int i = start; i < n; i++)
        {
            string str = s.substr(start, i-start+1);
```

```
            if(isPalindrome(str))
            {
                path.push_back(str);
                search(path, i+1, s, result);
                path.pop_back();
            }
        }
    }
    vector<vector<string>> partition(string s)
    {
        vector<vector<string> > result;
        vector<string> path;
        search(path, 0, s, result);
        return move(result);
    }
```

利用动态规划 O(n^2)

`dp[i:j]` 表示 `s[i:j]` 是回文, 如果 `s[i] == s[j] and dp[i+1, j-1]` ,满足条件, 则dp[i:j]就是回文。 注意要先算dp[i+1][j-1]，所以循环的顺序。

```
    void search(vector<string> &path, int start, string s, vector<vector<bool> > &dp, vec
    {
        if(start == s.length()){
            result.push_back(path);
            return;
        }
        for(int i = start; i < s.length(); i++)
        {
            if(dp[start][i]){
                string sub = s.substr(start, i-start+1);
                path.push_back(sub);
                search(path, i+1, s, dp, result);
                path.pop_back();
            }
        }
    }
    vector<vector<string>> partitionDp(string s)
    {
        int n = s.length();
        vector<vector<bool> > dp(n, vector<bool>(n, false));
        for(int i = 0; i < n; i++) //init, single char is palindrome
            dp[i][i] = true;
        //dp[i:j] s[i:j] is palindrome,  need dp[i+1, j-1], i need i+1, should downto, j
        for(int i = n-1; i >= 0; i--)
            for(int j = i; j < n; j++)
            {
                if(s[i] == s[j] && ( (i+1 >j-1) || dp[i+1][j-1]))
                    dp[i][j] = true;
            }
        vector<vector<string> > result;
        vector<string> path;
        search(path, 0, s, dp, result);
        return move(result);
    }
```

其实像上面那样把每一个回文子串找出来后，就不用像排列组合那样去搜索了，可以直接构造。这个参考了leetcode-cpp。 `result[i]` 表示s[i:n]构成的回文串拆分结果。再走一遍dp就可以构造出来。方法如下: result[i]的结果为当前的回文串 插入每一个 result[i+1]构成。

```
vector<vector<string>> partitionDp(string s)
{
    int n = s.length();
    vector<vector<bool> > dp(n, vector<bool>(n, false));
    for(int i = 0; i < n; i++) //init, single char is palindrome
        dp[i][i] = true;
    //dp[i:j] s[i:j] is palindrome,  need dp[i+1, j-1], i need i+1, should downto, j
    for(int i = n-1; i >= 0; i--)
        for(int j = i; j < n; j++)
        {
            if(s[i] == s[j] && ( (i+1 >j-1) || dp[i+1][j-1]))
                dp[i][j] = true;
        }
    vector<vector<vector<string> >> result(n, vector<vector<string> >());
    for(int i = n-1; i >= 0; i--)
        for(int j = i; j < n; j++)
        {
            if(dp[i][j])
            {
                string str = s.substr(i, j-i+1);
                if(j+1 < n)
                {
                    vector<vector<string> > &next = result[j+1];
                    for(int k = 0; k < next.size(); k++)
                    {
                        auto v = next[k];
                        v.insert(v.begin(), str);
                        result[i].push_back(v);
                    }
                }
                else
                    result[i].push_back(vector<string>(1, str));
            }
        }
    return result[0];
}
```

Permutation Sequence

题目来源：Permutation Sequence

> The set [1,2,3,…,n] contains a total of n! unique permutations. By listing and labeling all of the permutations in order, We get the following sequence (ie, for n = 3): "123" "132" "213" "231" "312" "321" Given n and k, return the kth permutation sequence. Note: Given n will be between 1 and 9 inclusive.

解题思路：

```
康托展开的逆运算
既然康托展开是一个双射，那么一定可以通过康托展开值求出原排列，即可以求出n的全排列中第x大排列。
如n=5,x=96时：
首先用96-1得到95，说明x之前有95个排列.(将此数本身减去！)
用95去除4！得到3余23，说明有3个数比第1位小，所以第一位是4.
用23去除3！得到3余5，说明有3个数比第2位小，所以是4，但是4已出现过，因此是5.
用5去除2!得到2余1，类似地，这一位是3.
用1去除1!得到1余0，这一位是2.
最后一位只能是1.
所以这个数是45321.
```

代码如下：

```
void jiecheng(vector<int> &f, int n)
{
    f.resize(n+1);
    f[0]=1;
    for(int i = 1; i <= n; i++)
        f[i] = f[i-1]*i;
}
string getPermutation(int n, int k)
{
    vector<int> f;
    jiecheng(f, n);
    string result;
    vector<int> small(n, 0);
    for(int i = 1; i <= n; i++)
        small[i-1] = i;
    k -= 1;
    for(int i = 0; i < n; i++)
    {
        int bigger = k / f[n-i-1];
        k = k % f[n-i-1];
        result += (*(small.begin()+bigger) + '0');
        small.erase((small.begin()+bigger));
    }
    return result;
}
```

参考

- 康托展开
- Leetcode: Permutation Sequence http://blog.csdn.net/MrRoyLee/article/details/34981399

Permutations

题目来源：Permutations

> Given a collection of numbers, return all possible permutations. For example, [1,2,3] have the following permutations: [1,2,3], [1,3,2], [2,1,3], [2,3,1], [3,1,2], and [3,2,1].

解题思路：

置换法

递归, 一个一个与第一个交换。

```
void dfs(vector<vector<int> > &result, vector<int> &num, int start)
{
    if(start >= num.size()) // >= ">"also should be put in, for the last ele.
    {
        result.push_back(num);
        return;
    }
    for(int i = start; i < num.size(); i++)
    {
        std::swap(num[i], num[start]);
        dfs(result, num, start+1);
        std::swap(num[i], num[start]);
    }
}
vector<vector<int> > permute(vector<int> &num)
{
    vector<vector<int> > result;
    dfs(result, num, 0);
    return move(result);
}
```

增量构造法

可以跟 combinations 类似, 一个一个往里面加。

```
void dfs(vector<vector<int> > &result, const vector<int> &num, vector<int> &path )
{
    if(path.size() == num.size())
    {
        result.push_back(path);
        return;
    }
    for(int i = 0; i < num.size(); i++)
    {
        if(std::find(path.begin(), path.end(), num[i]) == path.end())
        {
            path.push_back(num[i]);
            dfs(result, num, path);
            path.pop_back();
```

```
            }
        }
    }
    vector<vector<int> > permute(vector<int> &num)
    {
        vector<vector<int> > result;
        vector<int> path;
        dfs(result, num, path);
        return move(result);
    }
```

nextPermunation

参考 permutations-ii.

Permutations II

题目来源：Permutations II

> Given a collection of numbers that might contain duplicates, return all possible unique permutations. For example, [1,1,2] have the following unique permutations: [1,1,2], [1,2,1], and [2,1,1].

解题思路：

跟 Permutations思路差不多，分为下面几种解法。

置换法

跟Permutations一样，每一个与第一个交换～用set存结果，将重复的去掉，中途剪枝下即可AC。

```
void dfs(set<vector<int> > &result, vector<int> &num, int start)
{
    if(start >= num.size()) // >= ">"also should be put in, for the last ele.
    {
        result.insert(num);
        return;
    }
    for(int i = start; i < num.size(); i++)
    {
        if(i != start && num[i] == num[i-1]) continue; //culling
        std::swap(num[i], num[start]);
        dfs(result, num, start+1);
        std::swap(num[i], num[start]);
    }
}

vector<vector<int> > permuteUnique(vector<int> &num)
{
    set<vector<int> > result;
    std::sort(num.begin(), num.end());
    dfs(result, num, 0);
    return vector<vector<int>>(result.begin(), result.end());
}
```

增量构造

或者跟permutation的方法，增量构造, 这里需要用一个map存下数量。

```
void dfs(vector<vector<int> > &result, const int n, vector<int> &path, unordered_map<
{
    if(path.size() == n)
    {
        result.push_back(path);
        return;
    }
     //for(int i = 0; i < num.size(); i++)//num has redundant numbers,cannot use this
    for(auto it = countMap.begin(); it != countMap.end(); it++)
```

```
        {
            int cnt = 0;
            for(auto itj = path.begin(); itj != path.end(); itj++)
            {
                if(*itj == it->first) ++cnt;
            }
            if(cnt < it->second)
            {
                path.push_back(it->first);
                dfs(result, n, path, countMap);
                path.pop_back();
            }
        }
    }

    vector<vector<int> > permuteUnique(vector<int> &num)
    {
        vector<vector<int> > result;
        unordered_map<int, int> countMap;
        for(auto it = num.begin(); it != num.end(); it++)
            ++countMap[*it];
        vector<int> path;
        dfs(result, num.size(), path, countMap);
        return move(result);
    }
```

next_permunation

自然序的下一个：1 3 5 4 2，从后往前找，找到第一个降序(从后往前看)的数字3，然后找后面的比3大的最小的数字4，交换，1 4 5 3 2，然后交换index后面的序列逆序 532->235，构成下一个自然序：1 4 2 3 5。

```
    bool nextPermutation(vector<int> &current)
    {
        int end = current.size() - 1;
        while(end-1>=0 && current[end-1] >= current[end])
            end--;
        if(end == 0) //5 4 3 2 1
            return false;
        //[end-1] < [end] 2 3 5 4 1
        int start = end-1;//3
        end = current.size() - 1;
        while(current[end] <= current[start])
            end--;
        //[end] > [start] 4 > 3
        std::swap(current[start], current[end]); // 2 4 5 3 1
        std::reverse(current.begin()+start+1, current.end()); //2 4 1 3 5
        return true;
    }

    vector<vector<int> > permuteUnique3(vector<int> &num)
    {
        vector<vector<int> > result;
        std::sort(num.begin(), num.end());
        result.push_back(num);
        while(nextPermutation(num))
            result.push_back(num);
        return std::move(result);
```

```
    }
```

Subsets

题目来源：Subsets

> Given a set of distinct integers, S, return all possible subsets. Note: Elements in a subset must be in non-descending order. The solution set must not contain duplicate subsets. For example, If S = [1,2,3], a solution is: [[3], [1], [2], [1,2,3], [1,3], [2,3], [1,2], []]

解题思路：

注意输出的每个子集要有序。

DFS搜索

跟Combinations一样。

```
void search(vector<vector<int> > &result, vector<int> &S, vector<int> &input, int sta
{
    if(input.size() == k){
        result.push_back(input);
        return;
    }
    for(int i = start; i < S.size(); i++){
        input.push_back(S[i]);
        search(result, S, input, i+1, k);
        input.pop_back();
    }
}
vector<vector<int> > subsets(vector<int> &S)
{
    std::sort(S.begin(), S.end());
    vector<vector<int> > result;
    vector<int> input;
    for(int i = 0; i <= S.size(); i++)
        search(result, S, input, 0, i);
    return move(result);
}
```

0 二进制组合

每个元素都有0/1两种状态，全部排列一下即可。例如1,2,3,4一共有2^4=16种子集，第15种(2^0+2^1+2^2+2^3)为1-4都取, 第7种`(1*(2^0)+1*(2^1)+1*(2^2)+0*(2^3))`为[1,2,3]. ref. 注意得先将S排序(当然也可以先加到result中，最后再来排序), 不然结果中的子集顺序不是升序的。

```
vector<vector<int> > subsets(vector<int> &S)
{
    std::sort(S.begin(), S.end());
    int n = std::pow(2, S.size());
    vector<vector<int> > result(n, vector<int>());
    for(int j = 0; j < S.size(); j++)
```

```
        {
            for(int i = 0; i < n; i++)
            {
                if((1 << j) & i)
                    result[i].push_back(S[j]);
            }
        }
        return move(result);
    }
```

Subsets II

题目来源：Subsets II

> Given a collection of integers that might contain duplicates, S, return all possible subsets. Note: Elements in a subset must be in non-descending order. The solution set must not contain duplicate subsets. For example, If S = [1,2,2], a solution is: [[2], [1], [1,2,2], [2,2], [1,2], []]

解题思路：

可以跟上题Subsets一样做，只是在add到result的时候先判断result中是否存在。 或者最后将result unique一下都可以。

```
void searchWithDup(vector<vector<int> >& result, vector<int> &sub, vector<int> &S, in
{
    if(sub.size() == k)
    {
        //also can add, at last, remove the duplicated
        if(std::find(result.begin(), result.end(), sub) != result.end())
            return;
        result.push_back(sub);
        return;
    }
    for(int i = start; i < S.size(); i++)
    {
        sub.push_back(S[i]);
        searchWithDup(result, sub, S, k, i+1);
        sub.pop_back();
    }
}
vector<vector<int> > subsetsWithDup(vector<int> &S)
{
    std::sort(S.begin(), S.end());
    vector<vector<int> > result;
    for(int i = 0; i <= S.size(); i++)
    {
        vector<int> sub;
        searchWithDup(result, sub, S, i, 0);
    }
    //remove the duplicated, if didnot check when add in searchWithDup
    //std::sort(result.begin(), result.end());
    //result.erase(std::unique(result.begin(), result.end()), result.end());
    return move(result);
}
```

另外，考虑到会对输入的S排序，可以在递归搜索的时候，若有连续相邻的元素，则只需要搜索一个即可。ref

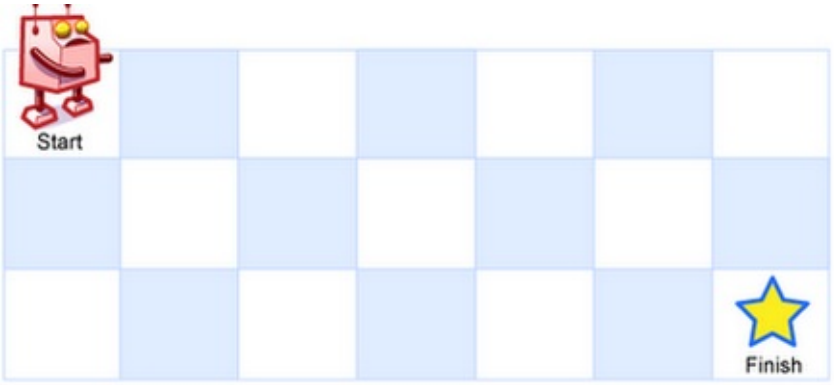
```
void search(vector<vector<int> > &result, const vector<int> &S, vector<int> &input, i
{
    if(input.size() == k){
        result.push_back(input);
```

```
            return;
        }
        for(int i = start; i < S.size(); i++){
            if(i != start && S[i] == S[i-1]) continue;
            input.push_back(S[i]);
            search(result, S, input, i+1, k);
            input.pop_back();
        }
    }
    vector<vector<int> > subsetsWithDup(vector<int> &S)
    {
        std::sort(S.begin(), S.end());
        vector<vector<int> > result;
        vector<int> input;
        for(int k = 0; k <= S.size(); k++)
            search(result, S, input, 0, k);
        return move(result);
    }
```

Unique Paths

题目来源：Unique Paths

> A robot is located at the top-left corner of a m x n grid (marked 'Start' in the diagram below). The robot can only move either down or right at any point in time. The robot is trying to reach the bottom-right corner of the grid (marked 'Finish' in the diagram below).



> How many possible unique paths are there? Above is a 3 x 7 grid. How many possible unique paths are there? Note: m and n will be at most 100.

解题思路：

一共m-1+n-1步，其中任意选择m-1作为竖着走即可。

```
//C_n ^m
int C(int n, int m)
{
    if(m == 0) return 1;
    if(n-m < m) return C(n, n-m);
    int i = 1;
    long long result = 1;
    while(i <= m)
    {
        result *= n-(i-1);
        result /= i;
        i++;
    }
    return (int)result;
}

int uniquePaths(int m, int n)
{
    return C(m-1 + n-1, m-1);
}
```

matrix, 二维数组, 矩阵相关

1. Rotate Image 题解
2. Set Matrix Zeroes 题解
3. Spiral Matrix 题解
4. Spiral Matrix II 题解
5. Maximal Rectangle 题解

Rotate Image

题目来源：Rotate Image

> You are given an n x n 2D matrix representing an image. Rotate the image by 90 degrees (clockwise). Follow up: Could you do this in-place?

解题思路：

常规

```
    void rotate(vector<vector<int> > &matrix)
    {
        int n = matrix.size();
        for(int row = 0; row < n / 2; row++)
        {
            for(int col = row; col < n - row - 1; col++)
            {
                int top = matrix[row][col]; // back top
                //left->top
                matrix[row][col] = matrix[n-col-1][row];
                //bottom->left
                matrix[n-col-1][row] = matrix[n-row-1][n-col-1];
                //right->bottom
                matrix[n-row-1][n-col-1] = matrix[col][n-row-1];
                //top->right
                matrix[col][n-row-1] = top;
            }
        }
    }
```

高级解法

参考discuss

```
//[discurs]
//(1) write your matrix on a paper.
//(2) flip (not rotate) the paper upside down. (reverse)
//(3) flip again, but this time bottom edge to the right. (swap)
123     789    741
456  –> 456 –> 852 –>OK
789     123    963
```

```
    void rotate(vector<vector<int> > &matrix)
    {
        int m = matrix.size(); if(m == 0) return;
        int n = matrix[0].size();
        std::reverse(matrix.begin(), matrix.end());
        for(int i = 0; i < m; i++)
            for(int j = 0; j < i; j++)
                std::swap(matrix[i][j], matrix[j][i]);
```

```
    }
```

Set Matrix Zeroes

题目来源：Set Matrix Zeroes

> Given a m x n matrix, if an element is 0, set its entire row and column to 0. Do it in place. click to show follow up. Follow up: Did you use extra space? A straight forward solution using O(mn) space is probably a bad idea. A simple improvement uses O(m + n) space, but still not the best solution. Could you devise a constant space solution?

解题思路：

O(m + n) 空间

另用数组记录哪些行/列有0.

```
void setZeroes(vector<vector<int> > &matrix)
{
    int m = matrix.size();
    if(m == 0) return;
    int n = matrix[0].size();
    vector<bool> row(m, false);
    vector<bool> col(n, false);
    for(int i = 0; i < m; i++)
        for(int j = 0;j < n; j++)
        {
            if(matrix[i][j] == 0)
                row[i] = col[j] = true;
        }
    for(int i = 0; i < m; i++)
        for(int j = 0;j < n; j++)
        {
            if(row[i] || col[j])
                matrix[i][j] = 0;
        }
}
```

常数空间

假设第i行j列是0，那么第0行的j列、第0列第i行 肯定要设置为0。 所以可以用两个变量记录下第0行0列是否有，然后把其他行列的信息往这写。

```
 void setZeroes(vector<vector<int> > &matrix)
{
    int m = matrix.size();
    if(m == 0) return;
    int n = matrix[0].size();
    bool firstRow = false;
    bool firstCol = false;
    for(int i = 0; i < n; i++)
    {
        if(matrix[0][i] == 0)
```

```
            {
                firstRow = true;
                break;
            }
        }
        for(int i = 0; i < m; i++)
        {
            if(matrix[i][0] == 0)
            {
                firstCol = true;
                break;
            }
        }

        for(int i = 1; i < m; i++)
            for(int j = 1;j < n; j++)
            {
                if(matrix[i][j] == 0)
                    matrix[i][0] = matrix[0][j] = 0;
            }
        for(int i = 1; i < m; i++)
            for(int j = 1;j < n; j++)
            {
                if(matrix[i][0] == 0 || matrix[0][j] == 0)
                    matrix[i][j] = 0;
            }
         if(firstRow)
         {
             for(int i = 0; i < n; i++)
                matrix[0][i]=0;
         }
         if(firstCol)
         {
             for(int i = 0; i < m; i++)
                matrix[i][0]=0;
         }
    }
```

Spiral Matrix

题目来源：Spiral Matrix

> Given a matrix of m x n elements (m rows, n columns), return all elements of the matrix in spiral order. For example, Given the following matrix: [[1, 2, 3], [4, 5, 6], [7, 8, 9]] You should return [1,2,3,6,9,8,7,4,5].

解题思路：

暴力模拟～

```
vector<int> spiralOrder(vector<vector<int> > &matrix)
{
    vector<int> result;
    int row = matrix.size();
    if(row == 0) return result;
    int colume = matrix[0].size();
    int size = row * colume;
    int rowindex = 0;
    int columeindex = colume-1;
    while(true)
    {
        int toleft_start = rowindex;
        int toleft_end = columeindex;
        int todown_start = rowindex + 1;
        int todown_end = row - rowindex - 1;

        for(int i = toleft_start; i <= toleft_end; i++)
            result.push_back(matrix[rowindex][i]);
        if(result.size() == size)
            break;
        for(int i = todown_start; i <= todown_end; i++)
            result.push_back(matrix[i][columeindex]);
        if(result.size() == size)
            break;
        for(int i = toleft_end-1; i>= toleft_start; i--)
            result.push_back(matrix[row - rowindex - 1][i]);
        if(result.size() == size)
            break;
        for(int i = todown_end-1; i >= todown_start; i--)
            result.push_back(matrix[i][colume-columeindex-1]);
        if(result.size() == size)
            break;
        rowindex++;
        columeindex--;
    }
    return move(result);
}
```

Spiral Matrix II

题目来源：Spiral Matrix II

> Given an integer n, generate a square matrix filled with elements from 1 to n2 in spiral order.
For example, Given n = 3, You should return the following matrix: [[1, 2, 3], [8, 9, 4], [7, 6, 5]]

解题思路：

跟 spiral-matrix一样。

```
vector<vector<int> > generateMatrix(int n)
{
    if(n == 0) return vector<vector<int> >();
    vector<vector<int> > matrix(n, vector<int>(n, 0));
    int rowindex = 0;
    int columeindex = n-1;
    int row = n, colume = n;
    int index = 1;
    int size = n * n;
    while(true)
    {
        int toleft_start = rowindex;
        int toleft_end = columeindex;
        int todown_start = rowindex + 1;
        int todown_end = row - rowindex - 1;

        for(int i = toleft_start; i <= toleft_end; i++)
            matrix[rowindex][i] = index++;
        if(index == size+1)
            break;
        for(int i = todown_start; i <= todown_end; i++)
            matrix[i][columeindex] = index++;
        if(index == size+1)
            break;
        for(int i = toleft_end-1; i>= toleft_start; i--)
            matrix[row - rowindex - 1][i] = index++;
        if(index == size+1)
            break;
        for(int i = todown_end-1; i >= todown_start; i--)
            matrix[i][colume-columeindex-1] = index++;
        if(index == size+1)
            break;
        rowindex++;
        columeindex--;
    }
    return move(matrix);
}
```

Maximal Rectangle

题目来源：Maximal Rectangle

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

解题思路：

O(n^3) 算法

一种直接的方法是: 横向记录从左到i, 包括i为1的连续1的长度，然后再纵向去查找以这个连续1长度作为min宽的最大的高度,得到面积。O(n^3)

```
int getMaxHeight(const vector<vector<int> > &ones, int row, int col)
{
    int minWidth = ones[row][col];
    int height = 0;
    for(int i = row; i >= 0; i--) //up
    {
        if(ones[i][col] >= minWidth)
            ++height;
        else
            break;
    }
    for(int i = row+1; i < ones.size(); i++)
    {
        if(ones[i][col] >= minWidth)
            ++height;
        else
            break;
    }
    return height;
}
int maximalRectangle(vector<vector<char> > &matrix)
{
    int m = matrix.size(); if(m == 0) return 0;
    int n = matrix[0].size();
    vector<vector<int> > ones(m, vector<int>(n, 0));//[i][j], 第i行从左到j, 包括j连续1的
    for(int i = 0; i < m; i++)
        for(int j = 0; j < n; j++)
        {
            if(j == 0) ones[i][0] = matrix[i][j] == '1' ? 1 : 0;
            else ones[i][j] = matrix[i][j] == '1' ? ones[i][j-1]+1 : 0;
        }
    int result = 0;
    for(int i = 0; i < m; i++)
        for(int j = 0; j < n; j++)
        {
            if(ones[i][j] > 0)
                result = std::max(result, ones[i][j] * getMaxHeight(ones, i, j));
        }
    return result;
}
```

O(n^2)算法

一行一行处理，每一行，按照柱状图那道题目 Largest Rectangle in Histogram `O(n)` 算法处理，总体复杂度O(n^2).

用栈维护了一个递增(非递减)的序列，当当前索引的元素比栈顶小时，取栈顶元素（并出栈），并将这个元素的高度和当前索引端(快降低了)构成的矩形面积，栈中上升的那段都可以出栈并计算。

```
int largestRectangleArea(const vector<int> &height)
{
    int result = 0;
    stack<int> index;
    for(int i = 0; i < height.size(); i++)
    {
        if(index.empty() || height[index.top()] <= height[i] )
            index.push(i);
        else
        {
            while(!index.empty() && height[index.top()] >= height[i])
            {
                auto t = index.top();
                index.pop();
                result = std::max(result, height[t] * (index.empty() ? i : (i-1)-inde
            }
            index.push(i);//Do not forget.
        }
    }
    return result;
}

int maximalRectangle(vector<vector<char> > &matrix)
{
    int m = matrix.size(); if(m == 0) return 0;
    int n = matrix[0].size();
    vector<vector<int> > ones(m, vector<int>(n, 0));
    int result = 0;
    for(int i = 0; i < m; i++)
    {
        for(int j = 0; j < n; j++)
        {
            if(i == 0) ones[0][j] = matrix[i][j] == '1' ? 1 : 0;
            else ones[i][j] = matrix[i][j] == '1' ? ones[i-1][j]+1 : 0;
        }
        ones[i].push_back(-1);
        result = std::max(result, largestRectangleArea(ones[i]));
    }
    return result;
}
```

以高度 `[2,5,3,4,1]` 为例, 2[index=0], 5[index=2]进栈, 当前高度为3, 以5为最矮的计算面积为5,然后5出栈, 此时把**3[index=2]**进栈 (注意对比下Largest Rectangle in Histogram的写法), 一直到index=4时，1最小了, 依次计算面积,

```
[0,2,3,]-->[0,2,]
4*(4-1-2)=8.
```

```
[0,2] --> [0]
3*(4-1-0)=9. !!max
[0] -->[]
2*(4)=8.
```

上面代码还可以优化下内存空间, 用 `O(n)` n为列数量。

```
int maximalRectangle2(vector <string > & matrix)
{
    int m = matrix.size(); if (m == 0) return 0;
    int n = matrix[0].size(); if (n == 0) return 0;
    vector< int> height(n+1, 0);
    int maxArea = 0;
    for(int row = 0; row < m; row++)
    {
        for(int col = 0; col < n; col++)
        {
            if(matrix [row][col] == '1')
                height[col] += 1;
            else
                height[col] = 0;
        }
        height[n] = -1; //dummy one
        maxArea = std::max(maxArea, largestRectangleArea(height));
    }
    return maxArea;
}
```

O(n^2)算法 思路2

参考了leetcode-cpp。 思路是对当前高度h, 找左边比他小的最大的index,设为i, 右边比h小最小的index,设为j,则以h为最小高度的面积应该为 `(j-i-1)*h`. eg : [2,5,3,4,1], 当前高度3, 则, left=0, right = 4, area = 3*(4-0-1)=9.

```
int maximalRectangle(vector<vector<char> > &matrix)
{
    int m = matrix.size(); if(m == 0) return 0;
    int n = matrix[0].size();
    int result = 0;
    vector<int> leftMax(n, -1);
    vector<int> rightMin(n, n);
    vector<int> height(n, 0);
    for(int i = 0; i < m; i++)
    {
        int left = -1, right = n;
        for(int j = 0; j < n; j++)
        {
            if(matrix[i][j] == '1'){
                ++height[j];
                leftMax[j] = std::max(leftMax[j], left);
            }else{
                left = j;
                height[j]=0; leftMax[j]=-1; rightMin[j]=n;
            }
        }
```

```
            for(int j = n-1; j >= 0; j--)
            {
                 if(matrix[i][j] == '1'){
                     rightMin[j] = std::min(rightMin[j], right);
                     result = std::max(result, height[j]*(rightMin[j]-leftMax[j]-1));
                 }else{
                     right = j;
                 }
            }
        }
        return result;
    }
```

回溯, BFS/DFS

1. Clone Graph 题解
2. Generate Parentheses 题解
3. N-Queens 题解
4. N-Queens II 题解
5. Restore IP Addresses 题解
6. Sudoku Solver 题解
7. Surrounded Regions 题解
8. Word Ladder 题解
9. Word Ladder II 题解
10. Word Search 题解

Clone Graph

题目来源：Clone Graph

>

```
Clone an undirected graph. Each node in the graph contains a label and a list of its neig


OJ's undirected graph serialization:
Nodes are labeled uniquely.

We use # as a separator for each node, and , as a separator for node label and each neigh
As an example, consider the serialized graph {0,1,2#1,2#2,2}.

The graph has a total of three nodes, and therefore contains three parts as separated by

First node is labeled as 0. Connect node 0 to both nodes 1 and 2.
Second node is labeled as 1. Connect node 1 to node 2.
Third node is labeled as 2. Connect node 2 to node 2 (itself), thus forming a self-cycle.
Visually, the graph looks like the following:

       1
      / \
     /   \
    0 --- 2
         / \
         \_/
```

解题思路：

用BFS+hashmap解决。

```
    //Definition for undirected graph.
    struct UndirectedGraphNode
    {
        int label;
        vector<UndirectedGraphNode *> neighbors;
        UndirectedGraphNode(int x) : label(x) {};
    };

    UndirectedGraphNode *cloneGraph(UndirectedGraphNode *node)
    {
        if(node == NULL) return NULL;
        unordered_map<UndirectedGraphNode*, UndirectedGraphNode*> copies;
        queue<UndirectedGraphNode*> todo;
        todo.push(node);
        copies[node] = new UndirectedGraphNode(node->label);
        while(!todo.empty())
        {
            auto origin = todo.front(); todo.pop();
            UndirectedGraphNode* cpy = copies[origin];
            for(auto it = origin->neighbors.begin(); it != origin->neighbors.end(); it++)
            {
```

```
                if(copies.find(*it) == copies.end())
                {
                    auto node1 = new  UndirectedGraphNode((*it)->label);
                    copies[*it] = node1;
                    todo.push(*it);
                }
                cpy->neighbors.push_back(copies[*it]);
            }
        }
        return copies[node];
    }
```

Generate Parentheses

题目来源：Generate Parentheses

> Given n pairs of parentheses, write a function to generate all combinations of well-formed parentheses. For example, given n = 3, a solution set is: "((()))", "(()())", "(())()", "()(())", "()()()"

解题思路：

用递归，一个括号一个括号放，只要有左括号在，随时都可以放；放右括号时，已经放好的左括号数量要多余已放好的右括号才可以。

```
void gen(vector<string> &result, string pre, int left, int right)
{
   if(left < 0 || right < 0) return;
   if(left == 0 && right == 0)
   {
       result.push_back(pre);
       return;
   }
   if(left > 0)
       gen(result, pre+"(", left-1, right);
   if(left < right) //  (()
       gen(result, pre+")", left, right-1);
}

vector<string> generateParenthesis(int n)
{
   vector<string> result;
   string pre;
   gen(result, pre, n, n);
   return move(result);
}
```

N Queens

题目来源：N-Queens

> The n-queens puzzle is the problem of placing n queens on an n×n chessboard such that no two queens attack each other. Given an integer n, return all distinct solutions to the n-queens puzzle. Each solution contains a distinct board configuration of the n-queens' placement, where 'Q' and '.' both indicate a queen and an empty space respectively. For example, There exist two distinct solutions to the 4-queens puzzle: [[".Q..", // Solution 1 "...Q", "Q...", "..Q."], ["..Q.", // Solution 2 "Q...", "...Q", ".Q.."]]

解题思路：

八(N)皇后问题，经典回溯法。

```
vector<string> genResult(const vector<int> &queen)
{
    int n = queen.size();
    vector<string> solution(n, string(n, '.'));
    for(int i = 0; i < n; i++)
        solution[i][queen[i]] = 'Q';
    return move(solution);
}

bool canPut(const vector<int> &queen, int row, int col)
{
    int n = queen.size();
    //for(int i = 0; i < n; i++) //no need, check after row
    for(int i = 0; i < row; i++)
    {
        if(queen[i] == -1) continue;
        if(queen[i] == col //|| queen[row] != -1 row one by one, no need
        || abs(i - row) == abs(queen[i] - col))
            return false;
    }
    return true;
}
void dfs(vector<vector<string>> &result, vector<int> &queen, int startRow)
{
    int n = queen.size();
    if(startRow == n)
    {
        result.push_back(genResult(queen));
        return;
    }
    for(int col = 0; col < n; col++)
    {
        if(canPut(queen, startRow, col))
        {
            queen[startRow] = col;
            dfs(result, queen, startRow+1);
            queen[startRow] = -1;
        }
    }
}

vector<vector<string> > solveNQueens(int n)
```

```
    {
        assert(n != 0);
        vector<vector<string> > result;
        vector<int> queen(n, -1);
        dfs(result, queen, 0);
        return move(result);
    }
```

N Queens II

题目来源：N-Queens II

> Follow up for N-Queens problem. Now, instead outputting board configurations, return the total number of distinct solutions.

解题思路：

跟上一题 n-queens一模一样。

```
bool canPut(const vector<int> &queen, int row, int col)
{
    int n = queen.size();
    for(int i = 0; i < n; i++)
    {
        if(queen[i] == -1) continue;
        if(queen[i] == col //|| queen[row] != -1 row one by one, no need
        || abs(i - row) == abs(queen[i] - col))
            return false;
    }
    return true;
}
void dfs(int &result, vector<int> &queen, int startRow)
{
    int n = queen.size();
    if(startRow == n)
    {
        ++result;
        return;
    }
    for(int col = 0; col < n; col++)
    {
        if(canPut(queen, startRow, col))
        {
            queen[startRow] = col;
            dfs(result, queen, startRow+1);
            queen[startRow] = -1;
        }
    }
}

int totalNQueens(int n)
{
    assert(n != 0);
    int result = 0;
     vector<int> queen(n, -1);
    dfs(result, queen, 0);
    return result;
}
```

Restore IP Addresses

题目来源：Restore IP Addresses

> Given a string containing only digits, restore it by returning all possible valid IP address combinations.
For example: Given "25525511135", return ["255.255.11.135", "255.255.111.35"]. (Order does not matter)

解题思路： 在原串中加点，每个位置都去试探，直到3个点加完毕，若满足规则就是。 注意以0开头的段。

```
bool check(const string &str)
{
    if(str.length() == 0) return false;
    int num = std::stoi(str);
    if(str[0] == '0' && str.length() != 1)
        return false;
    if(num >= 0 && num <= 255)
        return true;
    return false;
}

void search(vector<string> &result, string prefix, const string &input, int start, in
{
    if(input.length() - start > (4-dot) * 3) return;
    if(dot == 3)
    {
        string str = input.substr(start, input.length()-start);
        if(check(str))
            result.push_back(prefix + str);
        return;
    }

    for(int i = start; i < (start+3) && i < input.length(); i++)
    {
        string str = input.substr(start, i-start+1);
        if(check(str))
            search(result, prefix + str + ".", input, i+1, dot+1);

    }
}

vector<string> restoreIpAddresses(string s)
{
    vector<string> result;
    if(s.length() < 4 || s.length() > 12) return result;
    string prefix = "";
    search(result, prefix, s, 0, 0);
    return move(result);
}
```

Sudoku Solver

题目来源：Sudoku Solver

> Write a program to solve a Sudoku puzzle by filling the empty cells. Empty cells are indicated by the character '.'. You may assume that there will be only one unique solution.

解题思路：

记录需要填充的每个位置，然后用1-9一个一个去试～ 深搜即可。 类似的问题还有八皇后等。

```
//check if can put x in board[row][col]
bool canPut(vector<vector<char> > &board, int row, int col, char x)
{
    for(int i = 0; i < 9; i++)
    {
        if(board[i][col] == x) return false;
        if(board[row][i] == x) return false;
        int r = 3 * (row / 3) + i / 3;
        int c = 3 * (col / 3) + i % 3;
        if(board[r][c] == x) return false;
    }
    return true;
}
bool search(vector<vector<char> > &board, const vector<std::pair<int, int>> &need, in
{
    if(start == need.size())
        return true;
    int r = need[start].first;
    int c = need[start].second;
    for(char x = '1'; x <= '9'; x++)
    {
        if(canPut(board, r, c, x))
        {
            board[r][c] = x;
            if(search(board, need, start+1))
                return true;
            board[r][c] = '.'; //rollback
        }
    }
}
void solveSudoku(vector<vector<char> > &board)
{
    vector<std::pair<int, int>> need;
    for(int i = 0; i < 9; i++)
        for(int j = 0; j < 9; j++)
            if(board[i][j] == '.')
                need.push_back(make_pair(i, j));
    search(board, need, 0);
}
```

Surrounded Regions

题目来源：Surrounded Regions

>

```
Given a 2D board containing 'X' and 'O', capture all regions surrounded by 'X'.

A region is captured by flipping all 'O's into 'X's in that surrounded region.

For example,
X X X X
X O O X
X X O X
X O X X
After running your function, the board should be:

X X X X
X X X X
X X X X
X O X X
```

解题思路：

这个题关键在与能想到跟最外面的'O'连通的就能“幸免于难”, 因此可以从从最外层的O开始往里dfs/bfs搜索，把连在一起的O记录下来(比如暂时改成另外的符号)，遍历完之后，再把所有的搜索一遍，这时仍然是'O'的就要变成'X'了，是之前暂存的，要还原成'O'. ref

```
    void dfs(vector<vector<char>> &board, int row, int col)
    {
        int m = board.size();
        assert(m >= 1);
        int n = board[0].size();
        if(row >= m || col >= n || row < 0 || col < 0)
            return;
        if(board[row][col] == 'O')
        {
            board[row][col] = '.';
            dfs(board, row-1, col); //up
            dfs(board, row+1, col); //down
            dfs(board, row, col-1); //left
            dfs(board, row, col+1); //right
        }
    }

    void solve(vector<vector<char>> &board)
    {
        int m = board.size();
        if(m == 0) return;
        int n = board[0].size();
        for(int col = 0; col < n; col++)//top and bottom
        {
            dfs(board, 0, col);
            dfs(board, m-1, col);
        }
```

```
        for(int row = 0; row < m; row++) //left and right
        {
            dfs(board, row, 0);
            dfs(board, row, n-1);
        }
        //all the 'O' the outmost has been changed to '.',
        //now rollback and change the others 'O' -> 'X'
        for(int i = 0; i < m; i++)
            for(int j = 0; j < n; j++)
            {
                if(board[i][j] == '.')
                    board[i][j] = 'O';
                else if(board[i][j] == 'O')
                    board[i][j] = 'X';
            }
    }
```

或者将上面的dfs换成bfs也一样。

```
    bool check(int row, int col, vector<vector<char> >&board)
    {
        if(row < 0 || col < 0 || row >= board.size() || col >= board[0].size())
            return false;
        if(board[row][col] == 'O')
            return true;
        return false;
    }

    void bfs(vector<vector<char> >&board, int row, int col)
    {
        queue<std::pair<int, int> > coords;
        coords.push(std::pair<int, int>(row, col));
        while(coords.size() > 0)
        {
            int r = coords.front().first;
            int c = coords.front().second;
            coords.pop();
            if(check(r, c, board))
            {
                board[r][c] = '.';//mark
                if(check(r-1, c, board))//or check later (the up 'if')
                    coords.push(std::pair<int,int>(r-1, c));
                if(check(r+1, c, board))
                    coords.push(std::pair<int,int>(r+1, c));
                if(check(r, c-1, board))
                    coords.push(std::pair<int,int>(r, c-1));
                if(check(r, c+1, board))
                    coords.push(std::pair<int,int>(r, c+1));
            }
        }
    }
```

Word Ladder

题目来源：Word Ladder

>

```
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.
```

解题思路：

bfs

用BFS搜索，记录从开始到当前路径长度。注意遍历map/set删除满足条件的element的写法。第一个用BFS搜索到的肯定是最短的之一。DFS则不是哦。

```
    bool diff1char(string s1, string s2)
    {
        assert(s1.length() == s2.length());
        int diff = 0;
        for(int i = 0; i < s1.length(); i++)
            if (s1[i] != s2[i]) {
                ++diff;
                if(diff == 2) return false;
            }
        return true;
    }
    int ladderLength(string start, string end, unordered_set<string> &dict)
    {
        queue<std::pair<string,int>> queues;
        queues.push(std::pair<string, int>(start,1));
        auto it = dict.find(start);
        if(it != dict.end()) dict.erase(start);
        while (queues.size() > 0)
        {
            string s = queues.front().first;
            int curLen = queues.front().second;
            queues.pop();
            if (diff1char(s, end))
                return curLen + 1;
            for(auto it = dict.begin(); it != dict.end(); )
```

```
            {
                if (diff1char(s, *it))
                {
                    queues.push(std::pair<string, int>(*it, curLen+1));
                    it = dict.erase(it);
                }else
                    ++it;
            }
        }
        return -1;
    }
```

悲剧的是，上面的过不了～ testcase中dict太大，而word相对较短，去从dict去搜索相邻的单词，耗时太久。改为变动word中的每一个字符(26个一个一个试)，然后再去dict中判断是否存在。这样就能AC。ref, 代码如下：

```
    void getDiff1chars(string s1, queue<std::pair<string,int>> &next, int nextLen, unorde
    {
        int n = s1.length();
        for(int i = 0; i < n; i++)
        {
            string s2(s1);
            for(char c = 'a'; c <= 'z'; c++)
            {
                if(s2[i] != c){
                    s2[i] = c;
                    auto it = dict.find(s2);
                    if(it != dict.end())
                    {
                        next.push(std::pair<string,int>(s2, nextLen));
                        dict.erase(it);
                    }
                }
            }
        }
    }
    int ladderLength(string start, string end, unordered_set<string> &dict)
    {
        queue<std::pair<string,int>> queues;
        queues.push(std::pair<string, int>(start,1));
        auto it = dict.find(start);
        if(it != dict.end()) dict.erase(start);
        while (queues.size() > 0)
        {
            string s = queues.front().first;
            int curLen = queues.front().second;
            queues.pop();
            if (diff1char(s, end))
                return curLen + 1;
            getDiff1chars(s, queues, curLen+1, dict);
        }
        return 0;
    }
```

Word Ladder II

题目来源：Word Ladder II

>

```
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.
```

解题思路：

先BFS把邻接图构造出来，即比如与hit相差紧1个字符的单词有哪些~ 然后再dfs把所有结果搜索出来。有了word ladder的经验，这次就直接用 `26*length(word)` 去查找相邻的单词而不取dict搜索了。 其实到可以判断下 dict大小和 `26*length(word)` 之间的关系决定选用哪种方法。

```
    void getDiff1s(string s, unordered_set< string> &adjlist , const unordered_set< strin
    {
        for(int i = 0; i < s.length(); i++)
        {
            string strback(s );
            for(char c = 'a'; c <= 'z'; c++)
            {
                strback[i] = c;
                auto it = dict .find(strback);
                if(it != dict .end() && *it != s && adjlist.find(*it) == adjlist .end())
                    adjlist.insert(strback);
            }
        }
    }
    void genResult(int level, int targetLen , string end, vector <string> & path, std::un
    {
        string curStr = path[path .size() - 1];
        if(level == targetLen)
        {
            if(curStr == end )//!!IMPORTANT
                result.push_back(path );
            return;
        }
        for(auto it = adjList[curStr].begin(); it != adjList [curStr].end(); ++it)
        {
```

```
            path.push_back(*it);
            genResult( level+1, targetLen , end, path, adjList, result);
            path.pop_back();
        }
    }


    vector<vector <string>> findLadders( string start , string end, unordered_set <string
    {
        std::unordered_map< string, unordered_set <std::string>> adjList;
        if(dict.find( end) != dict .end()) dict.insert( end);
        if(dict.find( start) != dict .end()) dict.erase( start);
        //build adjList
        unordered_set< string> twoLevels[2];
        twoLevels[0].insert( start);
        int level = 0;
        while ( true)
        {
            unordered_set<string > &lastLevel = twoLevels[level % 2];
            unordered_set<string > &nextLevel = twoLevels[(level+1) % 2];
            nextLevel.clear();
            for(auto it = lastLevel.begin(); it != lastLevel.end(); ++it)
            {
                unordered_set<string > adj;
                getDiff1s(*it, adj, dict);
                adjList[*it] = adj;
                nextLevel.insert(adj.begin(), adj.end()); //if the same, will not insert
            }
            if(nextLevel.size() == 0)
                return vector <vector< string>>();// no result
            //can remove the ones in dict of the current level,
            for(auto it = nextLevel.begin(); it != nextLevel.end(); ++it)
                dict.erase(*it);//erase by key
            ++level;
            if(nextLevel.find(end ) != nextLevel.end())//find the smallest path
                break;
        }
        vector< vector<string > > result;
        vector< string> path(1, start );
        //adjList contains the smallest path, but not all the path is valid,
        //valid: path's length is level AND the last one is end
        genResult(0, level, end, path, adjList, result);
        return move(result);
    }
```

get了一种bfs的新技能，用一个queue，不用像上面那样两层之间交换。一层一层之间加个特殊的节点表示层次之间的隔板（比如空指针啊，空串等）。上面其他逻辑不变，bfs部分改变后的代码如下，也能AC。针对本体的逻辑，注意最内层for循环(//!!!!!)，不能直接加到q1中去，因为这样操作q1中可能含有重复的单词，会超时。

```
    vector<vector <string>> findLadders( string start , string end, unordered_set <string
    {
        std::unordered_map< string, unordered_set <std::string>> adjList;
        if(dict.find( end) != dict .end()) dict.insert( end);
        if(dict.find( start) != dict .end()) dict.erase( start);
        //build adjList
        queue< string> q1;
        q1.push( start);
```

```
        int level = 0;
        while (!q1.empty())
        {
            q1.push(""); //"" or NULL to split cur level and next level
            bool toEnd = false;
            unordered_set<string> toDelete;
            while(q1.front() != "")
            {
                string it = q1.front(); q1.pop();
                unordered_set<string > adj;
                getDiff1s(it, adj, dict);
                adjList[it] = adj;
                for(auto it = adj.begin(); it != adj.end(); it++)
                {
                    //q1.push(*it); //!!!!! this way may TLE
                    toDelete.insert(*it);
                    if(*it == end) toEnd = true;
                }
            }
            if(toDelete.size() == 0) return vector<vector <string>>();
            for(auto it = toDelete.begin(); it != toDelete.end(); it++)
            {
                q1.push(*it);
                dict.erase(*it);
            }
            q1.pop(); // pop ""
            ++level;
            if(toEnd)//find the smallest path
                break;
        }
        vector< vector<string > > result;
        vector< string> path(1, start );
        //adjList contains the smallest path, but not all the path is valid,
        //valid: path's length is level AND the last one is end
        genResult(0, level, end, path, adjList, result);
        return move(result);
    }
```

Word Search

题目来源：Word Search

> Given a 2D board and a word, find if the word exists in the grid. The word can be constructed from letters of sequentially adjacent cell, where "adjacent" cells are those horizontally or vertically neighboring. The same letter cell may not be used more than once. For example, Given board = [["ABCE"], ["SFCS"], ["ADEE"]] word = "ABCCED", -> returns true, word = "SEE", -> returns true, word = "ABCB", -> returns false.

解题思路：

DFS， 不能重复用，所以得标记一下某个字符是否用过。

```
bool search(vector <string> & board, int row, int col, string word, int index, vector<vec
{
    if(index == word.length())
        return true ;
    if((row-1) >= 0 && col >=0)//up
    {
        if(!used [row-1][ col] && board [row-1][ col] == word [index])
        {
            used[row -1][col] = true;
            if(search(board , row-1, col, word , index+1, used))
                return true ;
            used[row -1][col] = false;
        }
    }
    if(row >= 0 && ( col-1)>=0) //left
    {
        if(!used [row][col-1] && board[row ][col-1] == word[index ])
        {
            used[row ][col-1] = true;
            if(search(board , row, col-1, word, index +1, used))
                return true ;
            used[row ][col-1] = false;
        }
    }
    if(row+1 < board.size() && col >= 0) //below
    {
        if(!used [row+1][ col] && board [row+1][ col] == word [index])
        {
            used[row +1][col] = true;
            if(search(board , row+1, col, word , index+1, used))
                return true ;
            used[row +1][col] = false;
        }
    }
    if(row >= 0 && col+1 < board [0].size())//right
    {
        if(!used [row][col+1] && board[row ][col+1] == word[index ])
        {
            used[row ][col+1] = true;
            if(search(board , row, col+1, word, index +1, used))
                return true ;
            used[row ][col+1] = false;
```

```
        }
    }
    return false;
}

bool exist(vector <string> & board, string word)
{
    int row = board.size();
    if(row == 0) return false ;
    int col = board[0].size();
    if(row * col < word.length()) return false;
    vector< vector<bool > > used(row, vector< bool>(col, false ));


    int index = 0;
    for(int i = 0; i < row; i++)
    {
        for(int j = 0; j < col; j++)
        {
            if(board [i][j] == word[index])
            {
                used[i][j] = true;
                if(search(board , i, j, word, index+1, used))
                    return true ;
                used[i][j] = false;
            }
        }
    }
    return false;
}
```

上面代码难看了点，其实可以像下面这样，简单些。另外，可以借助Surrounded Regions的思想，在原有矩阵的情况下加标记，最后还原即可。可以省去O(m*n)的空间。

```
bool dfs(vector<vector<char> > &board, int i, int j,  const string &word, int index)
{
   if(index == word.length()) return true;
   if(i < 0 || j < 0 || i >= board.size() || j >= board[0].size()) return false;
   if(board[i][j] == word[index])
   {
       board[i][j] = '.';//used
       bool r = false;
       r = dfs(board, i-1, j, word, index+1);//up
       if(!r && dfs(board, i+1, j, word, index+1))//below
           r = true;
       if(!r && dfs(board, i, j-1, word, index+1))//left
           r = true;
       if(!r && dfs(board, i, j+1, word, index+1)) //right
           r = true;
       board[i][j] = word[index];//reset
       if(r) return true;
   }
   return false;
}
bool exist(vector<vector<char> > &board, string word)
{
   assert(board.size() > 0 && board[0].size() > 0);
   for(int i = 0; i < board.size(); i++)
```

```
        for(int j = 0; j < board[0].size(); j++)
            {
                if(dfs(board, i, j, word, 0))
                return true;
            }
    return false;
}
```

greedy, 贪心

1. Best Time to Buy and Sell Stock II 题解
2. Jump Game 题解
3. Jump Game II 题解

Best Time to Buy and Sell Stock II

题目来源：Best Time to Buy and Sell Stock II

>

```
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 as many
transactions as you like (ie, buy one and sell one share of the stock multiple
times). However, you may not engage in multiple transactions at the same time
(ie, you must sell the stock before you buy again).
```

解题思路： 这题目跟前一题的区别在于，这个题允许多次交易。

思路就是在一个递增序列里，最矮的点买进，最高的点卖出。这样得到的利润最大。于是算法就是找一段一段的递增序列。

```
int maxProfit(vector<int> &prices)
{
    int n = prices.size();
    if (n <= 1) return 0;
    int i = 0;
    int result = 0;
    while(i < n)
    {
        int j = i;
        while(j+1 < n && prices[j] <= prices[j+1])
            j++;
        if(j == n-1)
        {
            result += std::max(prices[n-1] - prices[i], 0);
            return result;
        };
        //prices[j]> prices[j+1]
        if(j > i)
            result += (prices[j] - prices[i]);
        i = std::max(i+1, j+1);
    }
    return result;
}
```

后来看人家代码，才两三行，就觉得奇怪了。再分析下，举个例子，序列 1 3 4 5, 1进5出（一天只能交易一次）。相当于1进3出3进4出4进5出。于是就有了下面的代码。

```
int maxProfit(vector<int> &prices)
{
    int n = prices.size();
    int result = 0;
    for(int i = 1; i < n; i++)
        result += std::max(prices[i]-prices[i-1], 0);
```

```
        return result;
    }
```

Jump Game

题目来源：Jump Game

> Given an array of non-negative integers, you are initially positioned at the first index of the array. Each element in the array represents your maximum jump length at that position. Determine if you are able to reach the last index. For example: A = [2,3,1,1,4], return true. A = [3,2,1,0,4], return false.

解题思路：

贪心

过每个index查看能到的最远的index，若当前最远的比遍历index还小或者相等时就走不下去了。

```
bool canJump(int A[], int n)
{
    if(n == 0) return true;
    int max_sofar = 0;
    for(int i = 0; i < n; i++)
    {
        max_sofar = std::max(max_sofar, i + A[i]);
        if(max_sofar >= n-1) return true;
        if(max_sofar <= i) return false;
    }
    return false;
}
```

动归

f[i]表示走到第A[i]时, 多余的最大步数。 `f[i] = max(f[i-1], A[i-1])-1`

```
bool canJump(int A[], int n)
{
    if(n == 0) return true;
    vector<int> dp(n, 0);
    for(int i = 1; i < n; i++)
    {
        dp[i] = std::max(dp[i-1], A[i-1]) - 1;
        if(dp[i] < 0) return false;
    }
    return dp[n-1] >= 0;
}
```

Jump Game II

题目来源：Jump Game II

> Given an array of non-negative integers, you are initially positioned at the first index of the array. Each element in the array represents your maximum jump length at that position. Your goal is to reach the last index in the minimum number of jumps. For example: Given array A = [2,3,1,1,4] The minimum number of jumps to reach the last index is 2. (Jump 1 step from index 0 to 1, then 3 steps to the last index.)

解题思路：

跟 jump-game一样, 可以贪心。每次选择当前可以跳的范围内，下一次能跳得最远的

```
int selectMaxIndex(int *A, int start, int range, int n)
{
    int max_ = A[start];
    int max_index = start;
    int i = start+1;
    while(i < n && i <= start + range)
    {
        if(i + A[i] > max_)
        {
            max_ = i + A[i];
            max_index = i;
        }
        i++;
    }
    if(i >= n) return n;//goal achived
    return max_index;
}
int jump(int A[], int n)
{
    if(n == 0) return 0;
    int step = 0;
    int cur = 0;
    int goal = n - 1;
    while(cur < goal)
    {
        int range = A[cur];
        int next = selectMaxIndex(A, cur, range, n); ////每次选择当前可以跳的范围内，下一次
        if(next <= cur) return -1;
        cur = next;
        ++step;
    }
    return step;
}
```

学习下 人家的代码就是简单.

```
int jump(int A[], int n)
{
    if(n == 0 || n == 1) return 0;
```

```
        int step = 0;
        int last = 0;
        int cur = 0;
        for(int i = 0;  i < n; i++)
        {
            if(i > last)
            {
                last = cur;
                ++step;
            }
            cur = std::max(cur, i + A[i]);
            if(cur >= n-1) return ++step;
            if(cur <= i) return -1;
        }
        return step;
    }
```

其他

1. Candy 题解
2. Container With Most Water 题解
3. Gas Station 题解
4. Gray Code 题解
5. Max Points on a Line 题解
6. Pascal's Triangle 题解
7. Pascal's Triangle II 题解
8. Remove Element 题解
9. Trapping Rain Water 题解

Candy

题目来源：Candy

>

```
There are N children standing in a line. Each child is assigned a rating value.

You are giving candies to these children subjected to the following
requirements:

Each child must have at least one candy.
Children with a higher rating get more candies than their neighbors.
What is the minimum candies you must give?
```

解题思路：

解析：注意理解题意 [3,2,2,3,1] 糖数量: 2,1,1,2,1; [4,2,3,4,1] 结果是 2,1,2,3,1.

每次找最低点，再往回确定糖数量

遍历一两遍即可,每次找下一次最低点，最低点的糖数量为1，再从最低的遍历到当前点得到结果。下面代码用了一个数组保存了每个child的结果，实际上只需用几个变量记录即可。 按照这个思路写了下面的比较戳的代码。

```
//nextLowest and max index
int nextLowest(vector<int> &ratings, int startIndex)
{
    if(startIndex == ratings.size() - 1) return startIndex;
    while(ratings[startIndex] >= ratings[startIndex+1])
    {
        startIndex++;
        if(startIndex == ratings.size() - 1)
            return startIndex;
    }
    return startIndex;
}

//use two other variables to store candies[index-1] candies[index+1]
//can change the space complexity to O(1)
int candy(vector<int> &ratings)
{
    size_t len = ratings.size();
    if(len <= 1) return (int)len;
    int curIndex = 0;
    int nextLow = nextLowest(ratings, curIndex);
    int sum = 0;
    vector<int> candies(len);
    while(curIndex < len)
    {
        if(curIndex == nextLow)
        {
            if(curIndex == 0)
                candies[curIndex] = 1;
```

```
                else
                    candies[curIndex] = candies[curIndex-1] + 1;
                sum += candies[curIndex];
            }else
            {
                int index = nextLow;
                while(index-1 >= curIndex)
                {
                    if(ratings[index-1] == ratings[index])
                    {
                        if(index+1 <= nextLow && ratings[index] > ratings[index+1])
                            candies[index] = candies[index+1] + 1;
                        else
                            candies[index] = 1;
                        sum += candies[index];
                    }else // ratings[index-1] > ratings[index]
                    {
                        if(index == nextLow || (index+1 <= nextLow && ratings[index] == r
                            candies[index] = 1;
                        else
                            candies[index] = candies[index+1] + 1;
                        sum += candies[index];
                    }
                    --index;
                    if(index == curIndex)
                    {
                        //4 2 3 [4] 1
                        if(ratings[index] > ratings[index+1] && (index-1 >= 0 && ratings[
                            candies[index] = std::max(candies[index+1], candies[index-1])
                        else if(ratings[index] > ratings[index+1])
                            candies[index] = candies[index+1] + 1;
                        else if(ratings[index] > ratings[index-1])
                            candies[index] = candies[index-1] + 1; //1 [2] 2
                        else
                            candies[index] = 1;
                        sum += candies[index];
                        break;
                    }
                }
            }
            curIndex = nextLow+1;
            nextLow = nextLowest(ratings, curIndex);
        }
        return sum;
    }
```

从左到右从右到左双向遍历

从discuss看到的答案，短小精悍的代码。思路也很清晰。

>

```
1. From left to right, to make all candies satisfy the condition if ratings[i] > ratings[
2. From right to left, almost like step 1, get a solution rightCandies[i] which just sati
3. For now, we have leftCandies[i] and rightCandies[i], so how can we satisfy the real co
```

即把整个过程分为两个步骤，第一步从左到右，只要右边的ratings大于自己，右边的糖数量就＝自己+1,(先不管左边大于右边的情况)，这一步完成之后有条件,`ratings[i] > ratings[i - 1] && candies[i] > candies[i - 1]` 然后再从右往左，一样的思路，使得`ratings[i] > ratings[i + 1] && candies[i] > candies[i + 1]`。 最后candies再取两个中的max，这样就同时满足这两个条件。问题解决。

```
int candy2(vector<int> &ratings)
{
    int n = ratings.size();
    if(n <= 1) return n;
    vector<int> candies(n, 1);
    //left to right
    for(int i = 1; i < n; i++)
    {
        if(ratings[i] > ratings[i-1])
            candies[i] = candies[i-1]+1;
        else
            candies[i] = 1;
    }
    int total = candies[n-1];
    for(int i = n-2; i >= 0; i--)
    {
        if(ratings[i] > ratings[i+1])
            candies[i] = std::max(candies[i+1]+1, candies[i]);
        total += candies[i];
    }
    return total;
}
```

备忘录法

这个方法参考了leetcode-cpp。即用递归的方式使得分得candy数量同时满足以上两个条件。

```
int f(vector<int> &ratings, vector<int> &cache, int index){
    if(cache[index] == 0)//has not been calculated before
    {
        cache[index] = 1;
        if(index+1 < ratings.size() && ratings[index] > ratings[index+1])
            cache[index] = std::max(cache[index], f(ratings, cache, index+1)+1);
        if(index-1 >= 0 && ratings[index] > ratings[index-1])
            cache[index] = std::max(cache[index], f(ratings, cache, index-1)+1);
    }
    return cache[index];}
int candy3(vector<int> &ratings){
    int n = ratings.size();
    if(n <= 1) return n;
    int total = 0;
    vector<int> cache(n, 0);
    for(int i = 0; i < n; i++)
        total += f(ratings, cache, i);
    return total;
}
```

Container With Most Water

题目来源：Container With Most Water

> Given n non-negative integers a1, a2, ..., an, where each represents a point at coordinate (i, ai). n vertical lines are drawn such that the two endpoints of line i is at (i, ai) and (i, 0). Find two lines, which together with x-axis forms a container, such that the container contains the most water. Note: You may not slant the container

解题思路：

从 discuss 得到的答案。

```
int maxArea(vector<int> &height)
{
    if(height.size() == 0) return 0;
    int result = 0;
    int left = 0;
    int right = height.size() - 1;
    while(left < right)
    {
        result = std::max(result, (right-left) * std::min(height[left], height[right]
        if(height[left] < height[right])
            ++left;
        else
            --right;
    }
    return result;
}
```

怎么证明是对的?

> Suppose the returned result is not the optimal solution. Then there must exist an optimal solution, say a container with aol and aor (left and right respectively), such that it has a greater volume than the one we got. Since our algorithm stops only if the two pointers meet. So, we must have visited one of them but not the other. WLOG, let's say we visited aol but not aor. When a pointer stops at a_ol, it won't move until >

- The other pointer also points to aol. In this case, iteration ends. But the other pointer must have visited aor on its way from right end to aol. Contradiction to our assumption that we didn't visit aor.
- The other pointer arrives at a value, say arr, that is greater than aol before it reaches aor. In this case, we does move aol. But notice that the volume of aol and arr is already greater than aol and aor (as it is wider and heigher), which means that aol and aor is not the optimal solution -- Contradiction!

或者

v[low, high] 表示(low, hight)和x轴围成的容器装水的结果，假设height[low] < height[high]，那么算法将low++, 这意味着v[low, high-1],v[low, high-2]等被忽略。v[low, high-1],v[low, high-2]……不会大于v[low, high]，因为装水的容量是由宽度和短的那个height[low]决定的(low是固定的)，宽度显然(low, high)更宽。

Gas Station

题目来源：Gas Station

>

```
There are N gas stations along a circular route, where the amount of gas at
station i is gas[i].

You have a car with an unlimited gas tank and it costs cost[i] of gas to travel
from station i to its next station (i+1). You begin the journey with an empty
tank at one of the gas stations.

Return the starting gas station's index if you can travel around the circuit
once, otherwise return -1.

Note:
The solution is guaranteed to be unique.
```

解题思路：

暴力法

一个一个试～ `O(N^2)` 能AC。

```
bool ok(vector<int> &gas, vector<int> &cost, int startIndex)
{
    int n = gas.size();
    int i = 0;
    int remain = 0;
    int index = startIndex;
    while(i < n)
    {
        if (remain + gas[index] - cost[index] < 0)
            return false;
        remain += gas[index] - cost[index];
        i++;
        index  = (index+1)%n;
    }
    if(i == n) return true;
}
int canCompleteCircuit(vector<int> &gas, vector<int> &cost)
{
    int n = gas.size();
    for(int i = 0; i  < n; i++)
    {
        if(ok(gas, cost, i))
            return i;
    }
    return -1;
}
```

O(N)解法

从discuss看来的答案。思路如下：

>

```
1. If car starts at A and can not reach B. Any station between A and B can not reach B.(B
2. If the total number of gas is bigger than the total number of cost. There must be a so
3. Every time a fail happens, accumulate the amount of gas that is needed to overcome the
```

```
 int canCompleteCircuit(vector<int> &gas, vector<int> &cost)
{
    int n = gas.size();
    int remain = 0;
    int sum = 0;
    int start = 0;
    for(int i = 0; i < n; i++)
    {
        if (remain + gas[i] - cost[i] < 0)
        {
            start = i+1;
            remain = 0;
        }else
            remain += gas[i] - cost[i];
        sum += gas[i] - cost[i];
    }
    return sum >= 0 ? start : -1;
}
```

Gray Code

题目来源：Gray Code

> The gray code is a binary numeral system where two successive values differ in only one bit. Given a non-negative integer n representing the total number of bits in the code, print the sequence of gray code. A gray code sequence must begin with 0. For example, given n = 2, return [0,1,3,2]. Its gray code sequence is: 00 - 0 01 - 1 11 - 3 10 - 2 Note: For a given n, a gray code sequence is not uniquely defined. For example, [0,2,3,1] is also a valid gray code sequence according to the above definition.
> For now, the judge is able to judge based on one instance of gray code sequence. Sorry about that.

解题思路：

逆序

注意观察，n每增加1，即是在n-1的结果之上，最高位加1，并按照n-1的逆序。

```
n = 1
0
1
n=2
0 0
0 1
---
1 1
1 0
n=3
0 0 0
0 0 1
0 1 1
0 1 0
------
1 1 0
1 1 1
1 0 1
1 0 0
```

```
    vector<int> grayCode(int n)
    {
        assert(n>=0);
        if(n == 0) return std::move(vector<int>(1,0));
        vector<int> result;
        result.push_back(0); result.push_back(1);
        for(int i = 1; i < n; i++)
        {
            int len = result.size();
            for(int j = len-1; j >=0; j--)
                result.push_back(result[j] + (1<<i));
        }
        return move(result);
    }
```

公式法

格雷码

```
G：格雷码  B：二进制码
G(N) = (B(n)/2) XOR B(n)
Binary Code(1011)要转换成Gray Code = (1011 >> 1) ^ 1011 = 1110
```

```
    vector<int> grayCode(int n)
    {
        assert(n>=0);
        int len = 1 << n;//std::pow(2,n);
        vector<int> result(len, 0);
        for(int i = 1; i < len; i++)
            result[i] = (i>>1)^i;
        return move(result);
    }
```

Max Points on a Line

题目来源：Max Points on a Line

> Given n points on a 2D plane, find the maximum number of points that lie on the same straight line.

解题思路：

一种解法是用map，斜率作为key，对每一个点，找过此点斜率相同的点数量。double数值作为map的key貌似不好，这样总感觉挺别扭的……。

注意，斜率不存在的情况，(所有)点重合的情况。 感觉还是用新写一个直线类比较好，下面的方法二就采取了一种类似的方法～或者用c++里的pair, 存斜率，不过得约下分。

第一种方法若用Java写就要注意了，Double作为key时得注意，斜率+0和-0用Double作为key时不一样。你可以试着输出Double +.0 和 Double -.0的hashcode，明显是不一样的。

```
int maxPoints(vector<Point> &points)
{
    int n = points.size();
    if(n <= 1) return n;
    unordered_map<double, int> slopes;
    int result = 1;
    for(int i = 0; i < n-1; i++)
    {
        slopes.clear();
        int dup = 1;
        for(int j = i+1; j < n; j++)
        {
            if(points[i].x == points[j].x && points[i].y == points[j].y)
            {
                ++dup;
                continue;
            }
            double deltaY = points[j].y - points[i].y;
            double deltaX = points[j].x - points[i].x;
            double slope = (deltaX == 0 ? INT_MAX: deltaY / deltaX); //yline's slope INT_
            //if(deltaY == 0) // +0.0 != -0.0
            //    slope = .0;
            if(slopes.find(slope) != slopes.end())
                slopes[slope]++;
            else
                slopes.insert(std::pair<double, int>(slope, 1));
        }
        if(result < dup) //all point is the same
            result = dup;
        for(auto it = slopes.begin(); it != slopes.end(); ++it)
        {
            if(it->second + dup > result)
                result = it->second + dup;
        }
    }
    return result;
}
```

方法二，自己新写一个类，科学的写法是斜率+斜率上一点构成直线，然后用另外的直线斜率相等且也过相同的一个点才判断两条直线是重合的，这里就偷懒了，构造直线的时候总用同一个点。 写Hash函数的时候得注意下, 相同的直线(斜率)映射的hash函数要一致才OK。即== 为*true*时，*hash*必须一样。

```
struct DeltaPoint
{
    int xx;
    int yy;
    DeltaPoint(int x, int y):xx(x), yy(y)
    {}
    DeltaPoint():xx(0), yy(0){}
    bool operator == (const DeltaPoint &l) const
    {
        if(l.xx == 0 && xx == 0) return true;
        if(l.xx == 0 || xx == 0) return false;
        if(abs(l.yy * 1.0 / l.xx - yy * 1.0 / xx) < 1e-7)
            return true;
        return false;
    }
};

struct hash_func
{
    size_t operator()(const DeltaPoint &l) const
    {
        hash<int> h;
        if(l.xx == 0)
            return 1;
        return h(l.yy / l.xx);
    }
};

bool operator == (const Point &a, const Point &b)
{
    return a.x == b.x && a.y == b.y;
}

int maxPoints2(vector<Point> &points)
{
    int result = 0;
    int n = (int)points.size();
    if(n <= 1) return n;
    unordered_map<DeltaPoint, int,  hash_func> countMap;
    for(int i = 0; i < n-1; i++)
    {
        Point &p0 = (points[i]);
        int same = 1;
        countMap.clear();
        for(int j = i+1; j < n; j++)
        {
            Point &p1 = points[j];
            if (p1 == p0)
                {++same; continue;}
            DeltaPoint pp(p1.x - p0.x, p1.y - p0.y);
            countMap[pp]++;
        }
        result = std::max(result, same);//if only the same points
        for(auto it = countMap.begin(); it != countMap.end(); it++)
        {
            if(it->second + same > result)
```

```
                result = it->second + same;
            }
        }
        return result;
    }
```

Remove Element

题目来源：Remove Element

> Given an array and a value, remove all instances of that value in place and return the new length. The order of elements can be changed. It doesn't matter what you leave beyond the new length.

解题思路：

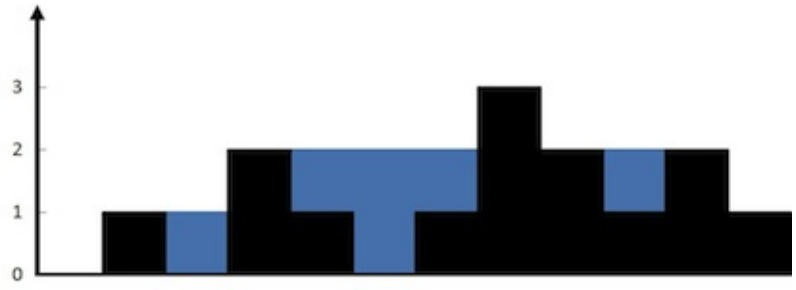
直接两个index, 跟目标不同就复制下, 不然只移动后面一个index. 跟Remove Duplicates from Sorted Array这道题类似.

```
int removeElement(int A[], int n, int elem)
{
    int index = 0;
    for(int i = 0; i < n; i++)
        if(A[i] != elem)
            A[index++] = A[i];
    return index;
}
```

Trapping Rain Water

题目来源：Trapping Rain Water

> Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it is able to trap after raining. For example, Given [0,1,0,2,1,0,1,3,2,1,2,1], return 6.



解题思路：

O(2*n)

先找打最高的柱子, 然后从两边往中间走, 如从左到右时, maxHeight记录到当前位置最高的柱子, 若当前高度cur小于maxHeight, 则 water += maxHeight - cur;

参考了 soulmachine 's leetcode;

```
int trap(int A[], int n)
{
    if(n == 0) return 0;
    int maxIndex = 0;
    for(int i = 1; i < n; i++)
        if(A[i] > A[maxIndex]) maxIndex = i;
    int lastMaxHeight = 0;
    int water = 0;
    for(int i = 0; i < maxIndex; i++)
    {
        if(A[i] > lastMaxHeight)
            lastMaxHeight = A[i];
        else
            water += lastMaxHeight - A[i];
    }
    lastMaxHeight = 0;
    for(int i = n-1; i > maxIndex; i--)
    {
        if(A[i] > lastMaxHeight)
            lastMaxHeight = A[i];
        else
            water += lastMaxHeight - A[i];
    }
    return water;
}
```

O(n)

> one pass and constant space,one point starts from left,another starts from right,and store the level at present,calculate the area of rectangle "all",and remove the area of block "block".It's the answer.

先假设没有中间的柱子，只有最边上的两根，算出容量，然后往中间走，一个一个的加柱子，加容量，计算柱子挡住的容量，最后减去即可。

参考 discuss.

```
int trap(int A[], int n)
{
    if(n == 0) return 0;
    int left = 0; int right = n-1;
    int cur = 0;
    int total = 0; int block = 0;
    while(left <= right)
    {
        int tmp = std::min(A[left], A[right]);
        if(tmp > cur)
        {
            total += (tmp - cur)*(right - left + 1);
            cur = tmp;
        }
        if(A[left] < A[right])
            block += A[left++];
        else
            block += A[right--];
    }
    return total - block;
}
```

类似的题目还有 Container With Most Water.