**Leetcode Log**

**2016.7.23**

001\_TwoSum 2Y

Given a series of numbers and a target. The problems requires to get the index of the two numbers whose sum is the target.

Not much to say, two-layer-loop to enumerate all conditions and terminate when ever there is a solution. The time complexity is O(n^2)。

At the first time when I submitted my solution, I didn’t write “return XXX” at the end of the function. Though I passed the test on my computer, but got an compilation error on leetcode because under some circumstances there will be no return value even if the problem told us there must be exactly one solution. The same error may happen if we don't write a return line after “else”.

By using unordered map we could reduce the time complexity of find the position to O(1), thus the total complexity can be reduced to O(n).(Detailed in code)

**2016.8.2**

002\_AddTwoNumbers 1Y

Review the basic operation of list and pointer.

004\_MedianOfTwoSortedArrays 1Y

005\_LongestPalindromicSubstring 1Y

006\_ZigZagCoversion 3Y

1st WA forget to consider empty string && forget to consider when numRows == 1.

2st Memory Exceeded didn’t calculate the size of the pattern array.

Watch out the initialization of 2-D arrays. We can not do it dynamically.

007\_ReverseInteger 2Y

1st WA forget to consider when reversed, the result could be larger than MAX\_INT.

008\_StringToInteger Too Many Was

Consider WhiteSpaces at the beginning.Consider ‘+’ and ‘-’, and the conversion break when there is a character which is not a number. Finally you have to consider the overload of MAX\_INT and MIN\_INT.

**2016.8.3**

009\_PalindromeNumber 2Y

Forget to consider negative number and all negative numbers are not palindrome.

011\_ContainerWithMostWater

1 and 2 TLE Search all conditions, O(n^2).

Greedy algorithm. The area was related to the lower one. So just start from the beginning and the end. It’s of no use that we change the higher one’s index closer to the lower one because the area will only decrease. Thus each time we just need to change the index of the lower one. And we can assure that once the two indexes become the same we have already find the maximum area.

**2016.8.4**

010\_RegularExpressionMatching

Recursively solve the problem

First, consider p[p\_index + 1] is ‘\*’;

Then consider where there is a ‘.’ or s and p has the same character at their indexes.

**2016.8.5**

012\_IntegerToRoman 1Y

013\_RomanToInteger 1Y

There are only two types of digits, IX (X minus I) or else(Add all together).

014\_LongestCommonPrefix 1Y

015\_3Sum 3Y

1 TLE Search O(n^3)

2 TLE Hashmap O(n^2)

3Y Passed the same numbers. Then accepted.

017\_LetterCombinationOfAPhoneNumber 1Y

Typical DFS

019\_RemoveNthNodeFromEndOfList 1Y

The best way to solve this problem is to set two pointer at the begining. The first one go n steps first and then they go together while the first on goes to the end.

020\_ValidParentheses 1Y

Use a stack to pair the brackets.

021\_MergeTwoSortedLists 1Y

**2016.8.6**

023\_MergeKSortedLists 1Y

Use MergeTwoSortedLists n times.

**2016.8.9**

022\_GenerateParentheses nWA

The parentheses follow the rule that if the unpaired left ones are zero, you couldn’t put a right one. Besides, if n \* 2 - pos == leftn, then you can only put right ones.

**2016.8.10**

026\_RemoveDuplicatesFromSortedArray 1Y

027\_RemoveElement 1Y

**2016.8.11**

028\_ImplementStrStr() 1Y

003\_LongestSubstringWithoutRepeatingCharacters 2Y

1 TLE Binary Search

2 Greedy Algorithm

024\_SwapNodesInPairs 1Y

**2016.8.27**

031\_NextPermutation 2Y

for(auto first = nums.end() - 2; first >= nums.begin(); first--){

for(auto last = nums.end() - 1; last != first; last--){

if(\*last > \*first){

swap(\*last, \*first);

sort(first + 1, nums.end());

return;

}

}

}

032\_LongestValidParentheses 1TLE 2Y

for(int pos = 0; pos < s.length(); pos++ ){

if(s[pos] == '(')

unpair++;

else

if(unpair == 0)

lastPos = pos;

else{

unpair--;

if(unpair == 0)

if(pos - lastPos > max)

max = pos - lastPos;

}

}

use this method twice.

**2016.8.28**

029\_DivideTwoIntegers

The divisor can be doubled every time to accelerate. Besides, mind the overload situation

Example Code:





034\_SearchForARange

Binary search, if(nums[head] == num[tail]) return [head, tail] or [-1, -1] for pruning.

035\_SearchInsertPosition

Binary search

int binarySearch(const vector<int> nums, int target, int head, int tail){

int mid = (head + tail) / 2 + 1;

if(head == tail)

if(nums[head] >= target)

return head;

else

return head + 1;

if(target >= nums[mid])

return binarySearch(nums, target, mid, tail);

else

return binarySearch(nums, target, head, mid - 1);

}

036\_ValidSudoku 1Y

**2016.8.30**

038\_CountAndSay 1Y

039\_CombinationSum 2Y

1 WA didn't sort the result and remove the duplicates

sort(result.begin(), result.end());

vector<vector<int>>::iterator end\_unique = unique(result.begin(), result.end());

result.erase(end\_unique, result.end());

040\_CombinationSumII 2Y

1 TLE use the method in previous problem, erase the item you selected everytime in candidates.

2Y DFS

041\_FirstMissingPositive 1Y

int firstMissingPositive(vector<int>& nums) {

vector<pair<int, int>> result;

pair<int, int> init = make\_pair(1, INT\_MAX);

result.push\_back(init);

for(auto i = nums.begin(); i != nums.end(); i++){

if(\*i <= 0)

continue;

for(int j = 0; j < result.size(); j++){

if(\*i >= result[j].first && \*i <= result[j].second){

pair<int, int> left, right;

left = make\_pair(result[j].first, \*i - 1);

right = make\_pair(\*i + 1, result[j].second);

result.erase(result.begin() + j);

if(left.first <= left.second)

result.insert(result.begin() + j++, left);

if(right.first <= right.second)

result.insert(result.begin() + j, right);

break;

}

}

}

return result[0].first;

}

042\_TrappingRainWater 2Y

1 WA Forgot when size is 0, couldn’t initialize.

while(last < height.size() - 1){

int i;

for(i = last + 1; i < height.size(); i++){

if(height[i] >= lastHeight)

if(lastHeight == 0){

lastHeight = height[i];

last = i;

} else {

for(int j = last + 1; j < i; j++){

result += lastHeight - height[j];

}

last = i;

lastHeight = height[last];

break;

}

else

continue;

}

if(i == height.size()){

lastHeight--;

}

}

**2016.8.31**

043\_MultiplyStrings 2Y

Similar to high accuracy multiplication. Cannot directly use char array or string because their ranges are from [-127, 128]

045\_JumpGameII 3Y

1 TLE DP

2 TLE Refined DP

3 Greedy Algorithm

int step = 0, last = 0, curr = 0, max = 0;

if(nums.size() == 1)

return 0;

for(int i = 0; i < nums.size(); i++){

if(i > last){

step++;

last = max;

}

curr = i + nums[i];

if(curr >= nums.size() - 1)

return step + 1;

else

if(curr > max)

max = curr;

else {}

}

return step;

046\_Permutations 1Y

dfs could solve

047\_PermutationsII 2Y

dfs TLE, but it can be passed by pruning.

Otherwise we can use next\_permutation function.

**2016.9.1**

050\_pow(x, n) 2Y

divide and conquer, mind overload condition.

051\_N-Queens 1Y

052\_N-QueensII 1Y

dfs

**2016.9.2**

053\_MaximumSubarray 2Y

Greedy, when current sum is lower than 0, let it be 0.

054\_SpiralMatrix 1Y

Mind all possible conditions.

058\_LengthOfLastWord 3Y

Forgot to consider the space could be at the end and there could be multiple spaces.

059\_SpiralMatrixII 1Y

If the parameter should be 2-D array, better use like f[x\*m + y], where f is a m x n matrix;

**2016.9.3**

062\_UniquePaths 1Y

063\_UniquePathsII 1Y

dp

**2016.9.4**

064\_MinimumPathSum 1Y

dp

060\_PermutationSquence 2Y

1 TLE One by one

2 recursion

066\_PlusOne 1Y

067\_AddBinary 1Y

069\_Sqrt(x) 2Y

1 WA Overflow when compare i \* i <= x;

070\_ClimbingStaris 1Y

DP

2016.9.6

056\_MergeIntervals

for(int i = 0; i < intervals.size() - 1; i++)

for(int j = i + 1; j < intervals.size(); j++)

if(!(intervals[i].start > intervals[j].end) && !(intervals[i].end < intervals[j].start)){

//Overlapped

Interval a = intervals[i], b = intervals[j];

Interval newInterval(min(a.start, b.start), max(a.end, b.end));

intervals.erase(intervals.begin() + j);

intervals.erase(intervals.begin() + i);

intervals.insert(intervals.begin() + i, newInterval);

j = i; //Until convergent

}

return intervals;