**5.最长回文子串**

1：找回文中心，向两端扩展

class Solution {

public:

pair<int,int> expend(const string& s,int left,int right)

{

while(left>=0 && right<s.size() && s[left] == s[right])

{

left--;

right++;

}

return {left+1,right-1};

}

string longestPalindrome(string s) {

int slen = s.size();

if(slen == 1)

return s;

int start=0,end=0;

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

{

auto[left1,right1] = expend(s,i,i);

auto[left2,right2] = expend(s,i,i+1);

if(right1-left1 > end-start)

{

start = left1;

end = right1;

}

if(right2-left2 > end-start)

{

start = left2;

end = right2;

}

}

return s.substr(start,end-start+1);

}

};

2：动态规划

class Solution {

public:

string longestPalindrome(string s) {

int n = s.size();

vector<vector<int>> dp(n, vector<int>(n));

string ans;

for (int l = 0; l < n; ++l) {

for (int i = 0; i + l < n; ++i) {

int j = i + l;

if (l == 0) {

dp[i][j] = 1;

}

else if (l == 1) {

dp[i][j] = (s[i] == s[j]);

}

else {

dp[i][j] = (s[i] == s[j] && dp[i + 1][j - 1]);

}

if (dp[i][j] && l + 1 > ans.size()) {

ans = s.substr(i, l + 1);

}

}

}

return ans;

}

};

3：马拉车算法（manacher）

class Solution {

public:

int expand(const string& s, int left, int right) {

while (left >= 0 && right < s.size() && s[left] == s[right]) {

--left;

++right;

}

return (right - left - 2) / 2;

}

string longestPalindrome(string s) {

int start = 0, end = -1;

string t = "#";

for (char c: s) {

t += c;

t += '#';

}

t += '#';

s = t;

vector<int> arm\_len;

int right = -1, j = -1;

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

int cur\_arm\_len;

if (right >= i) {

int i\_sym = j \* 2 - i;

int min\_arm\_len = min(arm\_len[i\_sym], right - i);

cur\_arm\_len = expand(s, i - min\_arm\_len, i + min\_arm\_len);

}

else {

cur\_arm\_len = expand(s, i, i);

}

arm\_len.push\_back(cur\_arm\_len);

if (i + cur\_arm\_len > right) {

j = i;

right = i + cur\_arm\_len;

}

if (cur\_arm\_len \* 2 + 1 > end - start) {

start = i - cur\_arm\_len;

end = i + cur\_arm\_len;

}

}

string ans;

for (int i = start; i <= end; ++i) {

if (s[i] != '#') {

ans += s[i];

}

}

return ans;

}

};

**6.Z字型转换**

1：

class Solution {

public:

string convert(string s, int numRows) {

int len = s.size();

if(len == 1 || numRows == 1)

return s;

int m = (numRows-1)\*2;

string res;

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

{

for(int j=0;j<len;j++)

{

if(j%m == i || j%m == m-i)

{

res += s[j];

}

}

}

return res;

}

};

2：

class Solution {

public:

string convert(string s, int numRows) {

if (numRows == 1) return s;

vector<string> rows(min(numRows, int(s.size())));

int curRow = 0;

bool goingDown = false;

for (char c : s) {

rows[curRow] += c;

if (curRow == 0 || curRow == numRows - 1) goingDown = !goingDown;

curRow += goingDown ? 1 : -1;

}

string ret;

for (string row : rows) ret += row;

return ret;

}

};

**8.字符串转换整数**

1：

class Solution {

public:

int myAtoi(string str) {

int len = str.size();

int isNeg = 1;

int res = 0 ,i = 0;

while(str[i] == ' ')

{

i++;

}

if(i == len)

return 0;

if(str[i] == '-')

{

isNeg = -1;

i++;

}

else if(str[i] == '+')

{

i++;

}

while(i < len)

{

if(str[i]<'0' || str[i]>'9')

break;

if(res > INT\_MAX/10 || (res == INT\_MAX/10 && (str[i]-'0') > INT\_MAX%10))

{

return INT\_MAX;

}

if(res < INT\_MIN/10 || (res == INT\_MIN/10 && -(str[i]-'0') < INT\_MIN%10))

{

return INT\_MIN;

}

res = res \* 10 + isNeg \* (str[i]-'0');

i++;

}

return res;

}

};

**9、回文数**

1：类型转换

using namespace std;

class Solution {

public:

    bool isPalindrome(int x)

    {

        long res;

        if(x<0 || (x%10 == 0 &&x!=0)) return false;

        string str = to\_string(x);

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

        stringstream out(str);

        out>>res;

        return x==res;

    }

};

2：翻转数字

class Solution {

public:

bool isPalindrome(int x) {

long y = 0; int quo = x;

while (quo) {

y = y \* 10 + quo % 10;

quo /= 10;

}

return x == y && x >= 0;

}

};

class Solution {

public:

bool isPalindrome(int x)

{

if(x<0 || (x%10==0 && x!=0))

{return false;}

int rev=0;

while(x>rev)

{

rev = rev\*10 + x%10;

x/=10;

}

return (x==rev || x==rev/10);

}

};

**13.罗马数字转整数**

1：switch case

class Solution {

public:

int romanToInt(string s) {

int res =0;

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

{

switch(s[i])

{

case 'I':

if(s[i+1] == 'V')

{

res+=4;

i++;

break;

}

else if(s[i+1] == 'X')

{

res+=9;

i++;

break;

}

else

{

res+=1;

break;

}

case 'V':

res+=5;

break;

case 'X':

if(s[i+1] == 'L')

{

res+=40;

i++;

break;

}

else if(s[i+1] == 'C')

{

res+=90;

i++;

break;

}

else

{

res+=10;

break;

}

case 'L':

res+=50;

break;

case 'C':

if(s[i+1] == 'D')

{

res+=400;

i++;

break;

}

else if(s[i+1] == 'M')

{

res+=900;

i++;

break;

}

else

{

res+=100;

break;

}

case 'D':

res+=500;

break;

case 'M':

res+=1000;

break;

}

}

return res;

}

};

2:哈希表

class Solution {

public:

int romanToInt(string s) {

int res = 0;

map<char,int> romanNum = {

{'I',1},

{'V',5},

{'X',10},

{'L',50},

{'C',100},

{'D',500},

{'M',1000}

};//初始化哈希表

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

{

if(romanNum[s[i]] < romanNum[s[i+1]])

res-=romanNum[s[i]];

else

res+=romanNum[s[i]];

}

return res;

}

};

**14.最长公共前缀**

1：纵向查找

using namespace std;

class Solution {

public:

string longestCommonPrefix(vector<string>& strs) {

if(!strs.size())

return "";

for(int i=0;i<strs[0].size();i++)

{

char c = strs[0][i];

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

{

if(strs[j].size()<=i || c!=strs[j][i])

{

return strs[0].substr(0,i);

}

}

}

return strs[0];

}

};

**15.三数之和**

**1：双指针**

class Solution {

public:

    vector<vector<int>> threeSum(vector<int>& nums) {

        int n = nums.size();

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

        vector<vector<int>> res;

        for(int first=0;first<n;first++)

        {

            if(first > 0 && nums[first] == nums[first-1])

            {

                continue;

            }

            int third = n-1;

            int target = -nums[first];

            for(int second = first+1;second<n;second++)

            {

                if(second > first+1 && nums[second] == nums[second-1])

                {

                    continue;

                }

                while(second<third && nums[second] + nums[third] > target)

                {

                    third--;

                    continue;

                }

                if(second == third)

                {

                    break;

                }

                if(nums[second] + nums[third] == target)

                {

                    res.push\_back({nums[first],nums[second],nums[third]});

                }

            }

        }

        return res;

    }

};

**16.最接近的三数之和**

**1：双指针**

int threeSumClosest(vector<int>& nums, int target)

{

int n = nums.size();

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

int min = 1e7;

// 定义一个小函数的新方法

// 根据差值的绝对值来更新答案

auto update = [&](int cur) {

if (abs(cur - target) < abs(min - target)) {

min = cur;

}

};

for (int first = 0; first < n; first++)

{

if (first > 0 && nums[first] == nums[first - 1])

{

continue;

}

int second = first + 1;

int third = n - 1;

while (second < third)

{

int sum = nums[first] + nums[second] + nums[third];

if (sum == target) return target;

update(sum);

if (sum < target)

{

int temp = second + 1;

while (temp < third && nums[second] == nums[temp])

temp++;

second = temp;

}

else

{

int temp = third - 1;

while (second < temp && nums[third] == nums[temp])

temp--;

third = temp;

}

}

}

return min;

}

17.电话号码的字母组合

1：哈希+回溯

map<char, string> phoneMap = {

{ '2',"abc" },

{ '3',"def" },

{ '4',"ghi" },

{ '5',"jkl" },

{ '6',"mno" },

{ '7',"pqrs" },

{ '8',"tuv" },

{ '9',"wxyz" }

};

vector<string> ans;

string current;

vector<string> Solution\_50::letterCombinations(string digits)

{

int n = digits.size();

if (n == 0) return ans;

DFS(0, digits);

return ans;

}

void DFS(int index , string digits)

{

//出口

if (index == digits.size())

{

ans.push\_back(current);

return;

}

//递归调用

//对于当前输入的第index号数字(digits[index])

//枚举其对应的所有字母(M[digits[index]][i])

for (int i = 0; i < phoneMap[digits[index]].size(); i++)

{

current.push\_back(phoneMap[digits[index]][i]);

DFS(index + 1, digits);

current.pop\_back();

}

}

**20.有效的括号**

1：超时了！！

class Solution {

public:

bool isValid(string s) {

if(s.size() == 0)

return true;

if(s.size()%2 == 1)

return false;

unordered\_map<char,int> map = {

{'(',1},

{'[',2},

{'{',3},

{'}',4},

{']',5},

{')',6},

};

do{

bool erase = false;

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

{

if(map[s[i]] + map[s[i+1]] == 7)

{

s.erase(i,2);

erase = true;

}

}

if(!erase && s.size()!=0)

{return false;}

}while(!s.size());

return true;

}

};

2：栈

class Solution {

public:

bool isValid(string s) {

if(s.size() == 0)

return true;

if(s.size()%2 == 1)

return false;

stack<char> st;

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

{

switch(s[i])

{

case '(':

case '[':

case '{':

st.push(s[i]);

break;

case ')':

if(st.empty())

return false;

if(st.top() == '(')

{

st.pop();

break;

}

else

return false;

case ']':

if(st.empty())

return false;

if(st.top() == '[')

{

st.pop();

break;

}

else

return false;

case '}':

if(st.empty())

return false;

if(st.top() == '{')

{

st.pop();

break;

}

else

return false;

}

}

return st.empty();

}

};

巧妙的入栈，放进去的都是右括号，就可以直接和当前取到的括号进行比较

class Solution {

public:

bool isValid(string s) {

stack <char> t;

for(char n : s)

{

if (n == '(') t.push(')');

else if (n == '{') t.push('}');

else if (n == '[') t.push(']');

else

{

if(t.empty() || t.top() != n) return false; //此时n为三种右括号之一，而栈顶也是三种右括号之一

else t.pop();

}

}

return t.empty();

}

};

**21.合并两个有序链表**

1：递归

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode() : val(0), next(nullptr) {}

\* ListNode(int x) : val(x), next(nullptr) {}

\* ListNode(int x, ListNode \*next) : val(x), next(next) {}

\* };

\*/

class Solution {

public:

ListNode\* mergeTwoLists(ListNode\* l1, ListNode\* l2) {

if(l1 == nullptr)

return l2;

else if(l2 == nullptr)

return l1;

else if(l1->val > l2->val)

{

l2->next = mergeTwoLists(l1,l2->next);

return l2;

}

else

{

l1->next = mergeTwoLists(l1->next,l2);

return l1;

}

}

};

2：建立新链表

class Solution {

public:

ListNode\* mergeTwoLists(ListNode\* l1, ListNode\* l2) {

ListNode\* head = new ListNode(-1);

ListNode\* pre = head;

while(l1!=nullptr && l2!=nullptr)

{

if(l1->val > l2->val)

{

pre->next = l2;

l2 = l2->next;

}

else

{

pre->next = l1;

l1 = l1->next;

}

pre = pre->next;

}

pre->next = (l1 == nullptr) ? l2 : l1;

return head->next;

}

};

**22.括号生成**

**1：**

class Solution {

public:

vector<string> Solution::generateParenthesis(int n)

{

vector<string> ans;

string cur;

generateParenthesisHelper(ans, cur, 0, 0, n);

return ans;

};

void Solution::generateParenthesisHelper(vector<string>& ans,string& cur,int left,int right,int n)

{

if (cur.size() == n\*2)

{

ans.push\_back(cur);

return;

}

if (left < n)

{

cur += '(';

generateParenthesisHelper(ans, cur, left + 1, right, n);

cur.pop\_back();

}

if (right < left)

{

cur += ')';

generateParenthesisHelper(ans, cur, left, right + 1, n);

cur.pop\_back();

}

}

**26.删除有序数组中的重复项**

1：

class Solution {

public:

int removeDuplicates(vector<int>& nums) {

int len=1,temp;

if(nums.size() == 0)

return 0;

temp = nums[0];

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

{

if(temp != nums[i])

{

temp = nums[i];

len++;

nums[len-1] = nums[i];

}

}

return len;

}

};

2：双指针

class Solution {

public:

int removeDuplicates(vector<int>& nums) {

if (nums.size() == 0) return 0;

int i = 0;

for (int j = 1; j < nums.size(); j++) {

if (nums[j] != nums[i]) {

i++;

nums[i] = nums[j];

}

}

return i + 1;

}

}

**27.移除元素**

1：双指针

class Solution {

public:

int removeElement(vector<int>& nums, int val) {

int i=0;

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

{

if(nums[j] != val)

{

nums[i] = nums[j];

i++;

}

}

return i;

}

};

2：更好的双指针

当我们遇到 nums[i]=valnums[i] = valnums[i]=val 时，我们可以将当前元素与最后一个元素进行交换，并释放最后一个元素。这实际上使数组的大小减少了 1。

请注意，被交换的最后一个元素可能是您想要移除的值。但是不要担心，在下一次迭代中，我们仍然会检查这个元素。

class Solution {

public:

int removeElement(vector<int>& nums, int val) {

int i = 0;

int n = nums.size();

while (i < n) {

if (nums[i] == val) {

nums[i] = nums[n - 1];

// reduce array size by one

n--;

} else {

i++;

}

}

return n;

}

**28.实现strstr()**

1:超时了

class Solution {

public:

int strStr(string haystack, string needle) {

if(needle.size()==0) return 0;

if(haystack.size()==0) return -1;

int i=0,idx=0;

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

{

if(haystack[j] == needle[i])

{

idx = j-i;

i++;

if(i>needle.size()-1)

{

return idx;

}

}

else

{

i=0;

j=idx+1;

}

}

return -1;

}

};

2：KMP

class Solution {

public:

vector<int> getnext(string str)

{

int len=str.size();

vector<int> next;

next.push\_back(-1);//next数组初值为-1

int j=0,k=-1;

while(j<len-1)

{

if(k==-1||str[j]==str[k])//str[j]后缀 str[k]前缀

{

j++;

k++;

next.push\_back(k);

}

else

{

k=next[k];

}

}

return next;

}

int strStr(string haystack, string needle) {

if(needle.empty())

return 0;

int i=0;//源串

int j=0;//子串

int len1=haystack.size();

int len2=needle.size();

vector<int> next;

next=getnext(needle);

while((i<len1)&&(j<len2))

{

if((j==-1)||(haystack[i]==needle[j]))

{

i++;

j++;

}

else

{

j=next[j];//获取下一次匹配的位置

}

}

if(j==len2)

return i-j;

return -1;

}

};

**35.搜索插入位置**

1：

class Solution {

public:

int searchInsert(vector<int>& nums, int target) {

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

{

if(nums[i] == target)

return i;

if(nums[i] > target)

return i;

}

return nums.size();

}

};

2：二分查找

class Solution {

public:

int searchInsert(vector<int>& nums, int target) {

int n = nums.size();

int left=0,right=n-1,ans=n;

while(left <= right)

{

int mid = (left+right)/2;

if(target<=nums[mid])

{

right = mid-1;

ans = mid;

}

else

{

left = mid+1;

}

}

return ans;

}

};

**38.外观数列**

1：

class Solution {

public:

string countAndSay(int n) {

if(n==1)

return "1";

string str = countAndSay(n-1);

string res;

int count = 1;

char c = str[0];

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

{

if(c == str[i])

{

count++;

}

else

{

res += to\_string(count) + c;

count = 1;

c = str[i];

}

}

res += to\_string(count) + c;

return res;

}

};

**43.字符串相乘**

**1：竖式**

int n1 = num1.size();

int n2 = num2.size();

vector<string> sum;

string ans;

if ((n1 == 1 && num1[0] == '0') || (n2 == 1 && num2[0] == '0'))

{

return "0";

}

int sign = 0;

for (int i = n1 - 1; i >= 0; i--)

{

int curNum1 = num1[i] - '0';

string tempstr;

for (int j = n2 - 1; j >= 0; j--)

{

int curNum2 = num2[j] - '0';

int mult = curNum1 \* curNum2 + sign;

sign = mult / 10;

mult = mult % 10;

tempstr = to\_string(mult) + tempstr;

}

if (sign != 0)

{

tempstr = to\_string(sign) + tempstr;

sign = 0;

}

for (int k = 0; k < n1 - i -1; k++)

{

tempstr = tempstr + '0';

}

sum.push\_back(tempstr);

}

int size = sum.size();

int len = sum[size-1].size();

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

{

int n = len - sum[i].size();

for (int j = 0; j < n; j++)

{

sum[i] = '0' + sum[i];

}

}

sign = 0;

for (int i = len-1; i >= 0; i--)

{

int temp = sign;

for (int j = 0; j < size; j++)

{

temp += sum[j][i] -'0';

}

sign = temp / 10;

temp = temp % 10;

ans = to\_string(temp) + ans;

}

if (sign != 0)

{

ans = to\_string(sign) + ans;

}

return ans;

**2：乘法**

class Solution {

public:

    string multiply(string num1, string num2) {

        if (num1 == "0" || num2 == "0") {

            return "0";

        }

        int m = num1.size(), n = num2.size();

        auto ansArr = vector<int>(m + n);

        for (int i = m - 1; i >= 0; i--) {

            int x = num1.at(i) - '0';

            for (int j = n - 1; j >= 0; j--) {

                int y = num2.at(j) - '0';

                ansArr[i + j + 1] += x \* y;

            }

        }

        for (int i = m + n - 1; i > 0; i--) {

            ansArr[i - 1] += ansArr[i] / 10;

            ansArr[i] %= 10;

        }

        int index = ansArr[0] == 0 ? 1 : 0;

        string ans;

        while (index < m + n) {

            ans.push\_back(ansArr[index]);

            index++;

        }

        for (auto &c: ans) {

            c += '0';

        }

        return ans;

    }

};

110.平衡二叉树

1：自底向上递归

class Solution {

public:

int height(TreeNode\* root) {

if (root == NULL) {

return 0;

}

int leftHeight = height(root->left);

int rightHeight = height(root->right);

if (leftHeight == -1 || rightHeight == -1 || abs(leftHeight - rightHeight) > 1) {

return -1;

} else {

return max(leftHeight, rightHeight) + 1;

}

}

bool isBalanced(TreeNode\* root) {

return height(root) >= 0;

}

};

2：自顶向下递归

class Solution {

public:

int height(TreeNode\* root) {

if (root == NULL) {

return 0;

} else {

return max(height(root->left), height(root->right)) + 1;

}

}

bool isBalanced(TreeNode\* root) {

if (root == NULL) {

return true;

} else {

return abs(height(root->left) - height(root->right)) <= 1 && isBalanced(root->left) && isBalanced(root->right);

}

}

};

130.被围绕的区域

1：

class Solution {

public:

    int n,m;

    void solve(vector<vector<char>>& board) {

        n = board.size();

        if(n == 0) return;

        m = board[0].size();

        for(int i=0;i<n;i++)

        {

            sereach(board,i,0);

            sereach(board,i,m-1);

        }

        for(int j=1;j<m-1;j++)

        {

            sereach(board,0,j);

            sereach(board,n-1,j);

        }

        for(int i=0;i<n;i++)

        {

            for(int j=0;j<m;j++)

            {

                if(board[i][j] == 'A')

                board[i][j] = 'O';

                else if(board[i][j] == 'O')

                board[i][j] = 'X';

            }

        }

    }

    void sereach(vector<vector<char>>& board,int i,int j)

    {

        if(i<0 || i>=n || j<0 || j>=m || board[i][j]!='O')

        return;

        board[i][j] = 'A';

        sereach(board,i+1,j);

        sereach(board,i-1,j);

        sereach(board,i,j+1);

        sereach(board,i,j-1);

    }

};