## 7 整数反转

思路：源自官网

<https://leetcode-cn.com/problems/reverse-integer/solution/zheng-shu-fan-zhuan-by-leetcode/>

比如要把数字123，反转成数字321。思路就是从数字末尾取处一个数字，然后放在头部，有点像pop和push

取出：

int pop = x % 10;

x /= 10;

放入：

rev = rev \* 10 + pop;

问题就在这个rev = rev \* 10 + pop;这个地方，要考虑整形溢出的问题，什么时候会溢出呢？

如果rev > INT\_MAX / 10, 那铁定会溢出

如果rev < INT\_MAX / 10, 那铁定不溢出

如果rev == INT\_MAX / 10, 这时候就是+7、-8两种情况之一了

class Solution {

public:

    int reverse(int x) {

        int pop, rev = 0;

        int TOP = INT\_MAX / 10;

        while(x) {

            pop = x % 10;

            x /= 10;

            if(rev > TOP || (rev == TOP && pop > 7)) return 0;

            if(rev < -TOP || (rev == -TOP && pop < -8)) return 0;

            rev = rev \* 10 + pop;

        }

        return rev;

    }

};

/\*

思路：源自官网

https://leetcode-cn.com/problems/reverse-integer/solution/zheng-shu-fan-zhuan-by-leetcode/

比如要把数字123，反转成数字321。思路就是从数字末尾取处一个数字，然后放在头部，有点像pop和push

取出：

int pop = x % 10;

x /= 10;

放入：

rev = rev \* 10 + pop;

问题就在这个rev = rev \* 10 + pop;这个地方，要考虑整形溢出的问题，什么时候会溢出呢？

如果rev > INT\_MAX / 10, 那铁定会溢出

如果rev < INT\_MAX / 10, 那铁定不溢出

如果rev == INT\_MAX / 10, 这时候就是+7、-8两种情况之一了

\*/

## 8 字符串转换整数atoi

class Solution {

public:

    int myAtoi(string s) {

        int i=0;

        int sign = 1, base = 0;

        while(s[i] == ' ') ++i;

        if(s[i] == '+') {

            sign = 1;

            ++i;

        }

        else if(s[i] == '-') {

            sign = -1;

            ++i;

        }

        while(i < s.size() && s[i] >= '0' && s[i] <= '9') {

            int temp = s[i] - '0';

            if(base > INT\_MAX/10 || (base == INT\_MAX/10 && temp > 7)) {

                return sign > 0 ? INT\_MAX : INT\_MIN;

            }

            base = base \* 10 + temp;

            ++i;

        }

        return sign \* base;

    }

};

/\*

思路：

顺序执行，先过滤空格 : while(s[i] == ' ') ++i;

接下来是符号！

接下来是数字，还是考虑整形溢出即可。

这里7和8的地方很微妙也很巧妙

\*/

## 39 组合总和

class Solution {public:

vector<vector<int>> combinationSum(vector<int>& candidates, int target) {

vector<int> path;

vector<vector<int>> ans;

backtrack(candidates, 0, 0, target, path, ans);

return ans;

}

void backtrack(vector<int>& candidates, int pos, int sum, int target,

vector<int>& path,

vector<vector<int>>& ans) {

int n = candidates.size();

if(pos == n) return;

if(sum == target) {

ans.push\_back(path);

return;

}

for(int i=pos; i<n; ++i) {

if(sum > target) continue;

path.push\_back(candidates[i]);

backtrack(candidates, i, sum + candidates[i], target, path, ans);

path.pop\_back();

}

}

};

## 40 组合总和II

class Solution {public:

vector<vector<int>> combinationSum2(vector<int>& candidates, int target) {

vector<int> path;

vector<vector<int>> ans;

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

backtrack(candidates, 0, 0, target, path, ans);

return ans;

}

void backtrack(vector<int>& candidates, int pos, int sum, int target,

vector<int>& path,

vector<vector<int>>& ans) {

int n = candidates.size();

// if(pos == n) return;

if(sum == target) {

ans.push\_back(path);

return;

}

for(int i=pos; i<n; ++i) {

if(sum > target) continue;

if(i > pos && candidates[i] == candidates[i-1]) continue;

path.push\_back(candidates[i]);

backtrack(candidates, i+1, sum + candidates[i], target, path, ans);

path.pop\_back();

}

}

};

## 50 Pow(x,n)

class Solution {

public:

    double myPow(double x, int n) {

        if(x == 0) return 0;

        if(n == 0) return 1;

        if(n == INT\_MIN) return 1 / (myPow(x, INT\_MAX) \* x);

        if(n < 0) return 1 / myPow(x, -n);

        if(n % 2 == 1) return x \* myPow(x, n-1);

        double temp = myPow(x, n >> 1);

        return temp \* temp;

    }

};

/\*

思路：

常规思路，如果n<0, 则返回1 / myPow(x, -n)

接下来是正数，分奇偶两种

如果奇数，则为myPow(x, n-1) \* x

如果偶数，则为myPow(x, n>>1) 的平方

最后是边界情况 x==0 和n==0

还有一种易错的情况，就是n = INT\_MIN

\*/

## 54 螺旋矩阵

方法一：方向数组 + visited标记

class Solution {

public:

    vector<int> spiralOrder(vector<vector<int>>& matrix) {

        int m = matrix.size(), n = m ? matrix[0].size() : 0;

        if(m == 0 || n == 0) return vector<int>();

        vector<pair<int,int>> directions{{0,1}, {1,0}, {0,-1}, {-1,0}};

        vector<vector<bool>> visited(m, vector<bool>(n, false));

        vector<int> ans;

        ans.push\_back(matrix[0][0]);

        int i = 0, j = 0;

        visited[0][0] = true;

        int dir = 0;

        for(int k=1; k<m\*n; ++k) {

            int r = i + directions[dir].first;

            int c = j + directions[dir].second;

            if(r<0 || r>=m || c<0 || c>=n || visited[r][c]) {

                dir = (dir + 1) % 4;

                r = i + directions[dir].first;

                c = j + directions[dir].second;

            }

            ans.push\_back(matrix[r][c]);

            visited[r][c] = true;

            i = r;

            j = c;

        }

        return ans;

    }

};

里面就是一些细节而已

方法二 ： 层层剥开

class Solution {

public:

    vector<int> spiralOrder(vector<vector<int>> &matrix) {

        int m = matrix.size(), n = m ? matrix[0].size() : 0;

        if(m == 0 || n == 0) return vector<int>();

        vector<int> ans;

        int k = min(m, n)-1;

        int sz = m\*n;

        for(int i=0; i<=k/2; ++i) {

            // 两种边界情形，走完就可以退出了

            if(i == n-1-i) {

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

                    ans.push\_back(matrix[j][i]);

                }

                break;

            }

            if(i == m-1-i) {

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

                    ans.push\_back(matrix[i][j]);

                }

                break;

            }

            // 正常情形

            // r : i; c : i~n-i-2

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

                ans.push\_back(matrix[i][j]);

            }

            // r : i ~ m-i-2 c : n-i-1

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

                ans.push\_back(matrix[j][n-i-1]);

            }

            // r : m-i-1  c : n-i-2 ~ i+1

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

                ans.push\_back(matrix[m-i-1][j]);

            }

            // r : m-i-2~i+1  c : i

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

                ans.push\_back(matrix[j][i]);

            }

        }

        return ans;

    }

};

以上两个代码都是自己写的，而且都写的非常好，建议自己把它背住！

## 59 螺旋矩阵II

方法一：方向数组 + visited标记

class Solution {

public:

    vector<vector<int>> generateMatrix(int n) {

        vector<vector<int>> ans(n, vector<int>(n,0));

        vector<int> directions{0,1,0,-1,0};

        vector<vector<bool>> visited(n, vector<bool>(n, false));

        ans[0][0] = 1;

        int i = 0, j = 0;

        int dir = 0;

        visited[0][0] = true;

        for(int k=2; k<=n\*n; ++k) {

            int r = i + directions[dir];

            int c = j + directions[dir+1];

            if(r<0 || r>=n || c<0 || c>=n || visited[r][c]) {

                dir = (dir+1) % 4;

                r = i + directions[dir];

                c = j + directions[dir+1];

            }

            ans[r][c] = k;

            i = r;

            j = c;

            visited[r][c] = true;

        }

        return ans;

    }

};

// 方法一：方向数组 + visited标记

方法二：层层剥开

class Solution {

public:

    vector<vector<int>> generateMatrix(int n) {

        vector<vector<int>> ans(n, vector<int>(n,0));

        int k = 0;

        for(int i=0; i<=(n-1)/2; ++i) {

            if(i==n-1-i) {

                ans[i][i] = ++k;

                break;

            }

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

                ans[i][j] = ++k;

            }

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

                ans[j][n-1-i] = ++k;

            }

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

                ans[n-1-i][j] = ++k;

            }

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

                ans[j][i] = ++k;

            }

        }

        return ans;

    }

};

## 66 加一

常规操作

class Solution {

public:

    vector<int> plusOne(vector<int>& digits) {

        int n = digits.size()-1;

        while(n>=0) {

            if(digits[n] < 9) {

                ++digits[n];

                return digits;

            }

            // 等于9怎么办？

            digits[n] = 0;

            --n;

        }

        // 至此还没有返回，说明是10000

        vector<int> res(digits.size()+1, 0);

        res[0] = 1;

        return res;

    }

};

## 100 相同的数

class Solution {

public:

    bool isSameTree(TreeNode\* p, TreeNode\* q) {

        if(!p && !q) return true;

        if(!p || !q) return false;

        if(p->val != q->val) return false;

        return isSameTree(p->left, q->left) && isSameTree(p->right, q->right);

    }

};

突出一个简单