深度遍历

079单词搜索（easy）

- **题目**
  给定一个二维网格和一个单词，找出该单词是否存在于网格中。
  单词必须按照字母顺序，通过相邻的单元格内的字母构成，其中“相邻”单元格是那些水平相邻或垂直相邻的单元格。同一个单元格内的字母不允许被重复使用。

  > 示例:
  > board =
  > [
  > ['A','B','C','E'],
  > ['S','F','C','S'],
  > ['A','D','E','E']
  >]
  > 给定 word = "ABCCED", 返回 true.
  > 给定 word = "SEE", 返回 true.
  > 给定 word = "ABCB", 返回 false.

- **思路**
  采用深度遍历，主程序对二维数组的每个字母作为第一个进行深度搜索。在深度搜索过程中，如果查找的单词长度与字符串长度相等时，说明已经匹配到。否则对i，j的值进行边界判断，以及board[i][j]是否等于字符串的当前字符，满足这些不符合的条件时，返回false。接着保存当前字符，并更换为'#',接着对当前位置的

其他四个位置进行深度搜索。把当前的 '#'更换为原来的字符。并返回当前深度搜索的结果。

- **代码实现**

```
class Solution {
public:
 bool exist(vector<vector<char>>& board, string word) {
  if (board.empty() || board[0].empty()) return false;
  int len_x = board[0].size(), len_y = board.size();
  for (int i = 0; i < len_y; ++i)
  for (int j = 0; j < len_x; ++j){
   if (search_word(board, word, 0, i, j))
    return true;
  }
  return false;
 }

 bool search_word(vector<vector <char>> &board, string word, int word_ptr, int i, int j){
  if (word_ptr == word.size())
   return true;
  if (i < 0 || j < 0 || i >= board.size() || j >= board[0].size() || board[i][j] != word[word_ptr])
   return false;
  char temp_c = board[i][j];
  board[i][j] = '#';
  bool res = search_word(board, word, word_ptr + 1, i - 1, j) || search_word(board, word, word_ptr + 1, i + 1, j) ||
   search_word(board, word, word_ptr + 1, i, j - 1) || search_word(board, word, word_ptr + 1, i, j + 1);
  board[i][j] = temp_c;
  return res;
 }
};
```

位运算

190 颠倒二进制位（easy）

- **题目**

  颠倒给定的 32 位无符号整数的二进制位。

  > 示例 1：
  > 输入: 00000010100101000001111010011100
  > 输出: 00111001011110000010100101000000
  > 解释: 输入的二进制串 00000010100101000001111010011100 表示无符号整数 43261596，
  > 因此返回 964176192，其二进制表示形式为 00111001011110000010100101000000。

- **思路**

  对于数n，(n >> i) & 1判断出第 i 位为 0 或者 1，<< (31 - i)为 i 位逆置的位置。

- **代码实现**

```
class Solution {
public:
    uint32_t reverseBits(uint32_t n) {
        uint32_t res = 0;
        for (int i = 0; i < 32; ++i) {
            res |= ((n >> i) & 1) << (31 - i);
        }
        return res;
    }
};
```

二分查找（各种边界的总结）

- 查找与目标相同的值

```
int find(vector<int>& nums, int target) {
    int left = 0, right = nums.size();
    while (left < right) {
        int mid = left + (right - left) / 2;
        if (nums[mid] == target) return mid;  //不一定有目标值的时候删去此判断
        else if (nums[mid] < target) left = mid + 1; //其他位置代码固定，根据实际需求只改此处的判断条件
```

```
        else right = mid;
    }
    return -1;
}
```

若high初始化为nums.size()，那么就必须用 low < high，而最后的 high 的赋值必须用 high = mid。

- 查找第一个大于等于目标值的数

```
if (nums[mid] < target) left = mid + 1;
```

- 查找 第一个小于等于目标数的值

069 x 的平方根（easy）

- **题目**
  实现 int sqrt(int x) 函数。
  计算并返回 x 的平方根，其中 x 是非负整数。
  由于返回类型是整数，结果只保留整数的部分，小数部分将被舍去。

  > 示例 1:
  > 输入: 4
  > 输出: 2

- **思路**
  相当与求第一个大于目标值的二分查找
- **代码实现**

```
class Solution {
public:
    int mySqrt(int x) {
        if(x <= 1)
            return x;
        int low = 0,high = x;
        while(low < high){
            int mid = (low + high) >> 1;
            if( mid <= x / mid)
```

```
                low = mid + 1;
            else
                high = mid;
        }
        return low - 1;
    }
};
```

数学问题

050 Pow(x, n) (middle)

- **题目**
  实现 pow(x, n) ，即计算 x 的 n 次幂函数。

  > 示例 1:
  > 输入: 2.00000, 10
  > 输出: 1024.00000

- **思路**
  只要 n 不为 0，那么每个 循环n/=2（这里如果用移位运算符的话不适用 n 为负数的情况）。

- **代码实现**

```
class Solution {
public:
    double myPow(double x, int n) {
        if( n < 0 && x == 0)
            return -1;
        double res = 1.0;
        for (int i = n; i != 0; i /= 2) {
            if (i & 1) res *= x;
            x *= x;
        }
        return n < 0 ? 1 / res : res;
    }
};
```

洗牌算法

时间复杂度为O(n),空间复杂度为O(1),缺点必须知道数组长度n.

```
void Knuth_Durstenfeld_Shuffle(vector<int>&arr){
 for (int i=arr.size()-1;i>=0;--i) {
  srand((unsigned)time(NULL));//根据时间初始化随机种子
  swap(arr[rand()%(i+1)],arr[i]);
 }
}
```