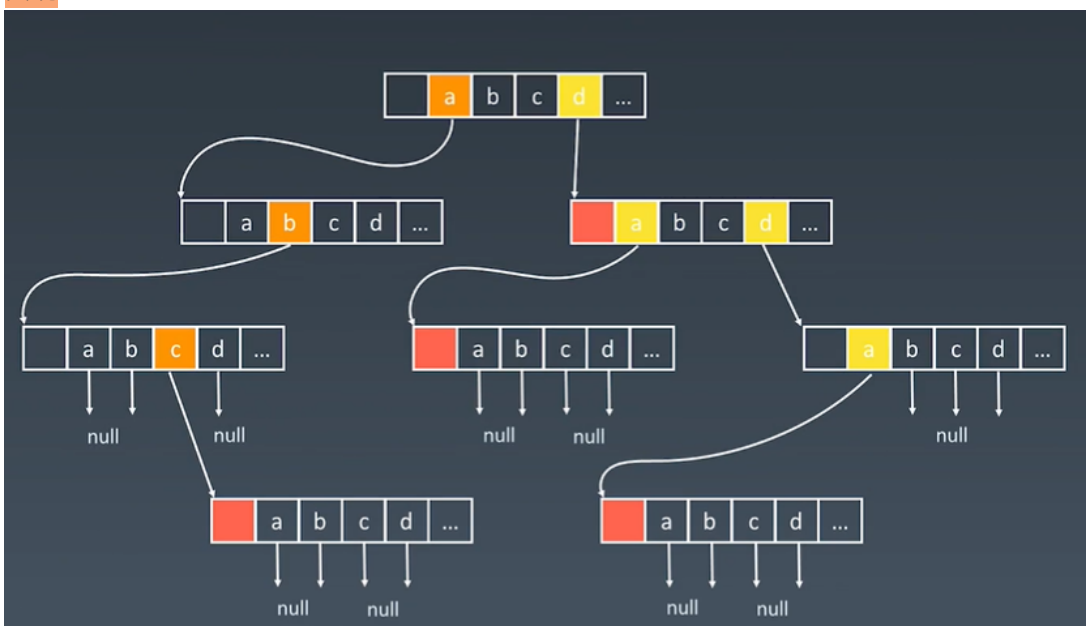
Week7

字典树

字典树原理比较简单，但是细节很考验代码功底，不小心就会写的又臭又长，要背住python实现




```
class TrieNode {
    // R links to node children，26个字母分别指向哪些节点
    private TrieNode[] links;
    private final int R = 26;//26个字符
    private boolean isEnd;//到这个点为止，有没有单词存在
    public TrieNode() {
        links = new TrieNode[R];
    }
    public boolean containsKey(char ch) {
        return links[ch -'a'] != null;
    }
    public TrieNode get(char ch) {
        return links[ch -'a'];
    }
    public void put(char ch, TrieNode node) {
        links[ch -'a'] = node;
    }
    public void setEnd() {
        isEnd = true;
    }
    public boolean isEnd() {
        return isEnd;
```

```
    }
}
```

向 Trie 树中插入键

```
class Trie {
    private TrieNode root;
    public Trie() {
        root = new TrieNode();
    }
    // Inserts a word into the trie.
    public void insert(String word) {
        TrieNode node = root;
        for (int i = 0; i < word.length(); i++) {
            char currentChar = word.charAt(i);
            if (!node.containsKey(currentChar)) {
                //如果找不到，就创建一个新节点挂在node上
                node.put(currentChar, new TrieNode());
            }
            node = node.get(currentChar);
        }
        node.setEnd();
    }
}
```

在Trie树中查找键

```
class Trie {
    // search a prefix or whole key in trie and
    // returns the node where search ends
    private TrieNode searchPrefix(String word) {
        TrieNode node = root;
        for (int i = 0; i < word.length(); i++) { //遍历单词中每个字符
           char curLetter = word.charAt(i);
           if (node.containsKey(curLetter)) {
               node = node.get(curLetter); //存在的话就去儿子节点
           } else {
               return null;
           }
        }
        return node;
    }
    // Returns if the word is in the trie.
    public boolean search(String word) {
       TrieNode node = searchPrefix(word);
       return node != null && node.isEnd(); //否则只是前缀
    }
}
```

Python实现Trie树（背住！）

```
class Trie(object):
```

```
    def __init__(self):
        self.root = {} #字典，key是26个字母，value是下一个节点的位置
        self.end_of_word = "#"

    def insert(self, word):
        node = self.root #一个嵌一个字典，把树形结构放到字典里
        for char in word: #循环word，对于每一个字符，去node里找有没有
            node = node.setdefault(char, {}) #setdefault，没有就空字典
        node[self.end_of_word] = self.end_of_word #标识符，完整单词结尾

    def search(self, word):
        node = self.root
        for char in word:
            if char not in node: #单词的字符是否在字典的key中
                return False
            node = node[char] #单词的字符在字典中，就把孩子赋给node
        return self.end_of_word in node

    def startsWith(self, prefix):
        node = self.root
        for char in prefix:
            if char not in node:
                return False
            node = node[char]
        return True
```

单词搜索II

此题字节跳动一面面试被问到过，要会字典树原理和代码
方法1：
遍历words，在board里面查找
先看单词的首字母在board里面有没有，如果有再四连通查找。
O(N * m * m * 4^k) ——最坏的情况下
N是单词数，board是m*m的，四连通，每个单词平均长度k
方法2：
Trie树
a. 所有单词，放到Trie里面，构建prefix
b. 对board进行DFS，看是不是Trie里面的前缀，如果不是就不往下查找了

```
import collections
class Solution:
    def __init__(self):
        self.dx = [-1, 1, 0, 0]
        self.dy = [0, 0, -1, 1]
        self.END_OF_WORD = "#"
    def findWords(self, board, words):
        if not board or not board[0]: return []
        if not words: return []
        self.result = set()
```

```
        # 构建trie
        root = collections.defaultdict()
        for word in words:
            node = root
            for char in word:
                node = node.setdefault(char, collections.defaultdict())
            node[self.END_OF_WORD] = self.END_OF_WORD
        self.m, self.n = len(board), len(board[0])
        #两重循环，遍历board，进行dfs
        #剪枝：如果board里的字符不是Trie当中任何单词的字母，不进行dfs
        for i in range(self.m):
            for j in range(self.n):
                if board[i][j] in root:
                    self._dfs(board, i, j, "", root)
        return list(self.result)
    def _dfs(self, board, i, j, cur_word, cur_dict):
        cur_word += board[i][j] #根据board值累加成一个cur_word
        cur_dict = cur_dict[board[i][j]] #Trie的下一层，判断下一层的字符在Trie里面有没有
        if self.END_OF_WORD in cur_dict:
            #如果有，并且END_OF_WORD也在，表示单词找到了，不然只是前缀
            self.result.add(cur_word)
        tmp, board[i][j] = board[i][j], '@'
        #先把board[i][j]保存成tmp，再用'@'替换用过的board[i][j]，防止走回来
        for k in range(4):
            x, y = i + self.dx[k], j + self.dy[k] #用四连通数组扩散四周
            if 0 <= x <self.m and 0 <= y <self.n and board[x][y] != '@' and board[x][y] in cur_dict:
                #x和y必须在board内，board[x][y]没被用过，且board[x][y]在Trie树的当前层级
                self._dfs(board, x, y, cur_word, cur_dict) #drill down
        board[i][j] = tmp #恢复board[x][y]，之前替换为'@'了
```

下面这段python代码要背下来，写的很简洁漂亮

一共三块内容：

1.构造字典树，要烂熟于心

2.DFS，上下左右变换坐标，也要烂熟于心

3.遍历二维数组

```
class Solution:
    def findWords(self, board: List[List[str]], words: List[str]) -> List[str]:
        trie = {}  # 构造字典树
        for word in words:
            node = trie
            for char in word:
                node = node.setdefault(char, {})
            node['#'] = True

```

```
        def search(i, j, node, pre, visited):  # (i,j)当前坐标，node当前trie树结点，pre前面的字符串，visited已访问坐标
            if '#' in node:  # 已有字典树结束
                res.add(pre)  # 添加答案
            for (di, dj) in ((-1, 0), (1, 0), (0, -1), (0, 1)):
                #python二元组
                _i, _j = i+di, j+dj
                if -1 < _i < h and -1 < _j < w and board[_i][_j] in node and (_i, _j) not in visited:  # 可继续搜索
                    search(_i, _j, node[board[_i][_j]], pre+board[_i][_j], visited | {(_i, _j)})  # dfs搜索
                    #visited | {(_i, _j)}，把(_i, _j)加到visited中，{(1, 2)} | {(1, 1)} = {(1, 1), (1, 2)}
        res, h, w = set(), len(board), len(board[0])
        for i in range(h):
            for j in range(w):
                if board[i][j] in trie:  # 可继续搜索
                    search(i, j, trie[board[i][j]], board[i][j], {(i, j)})  # dfs搜索
        return list(res)
```

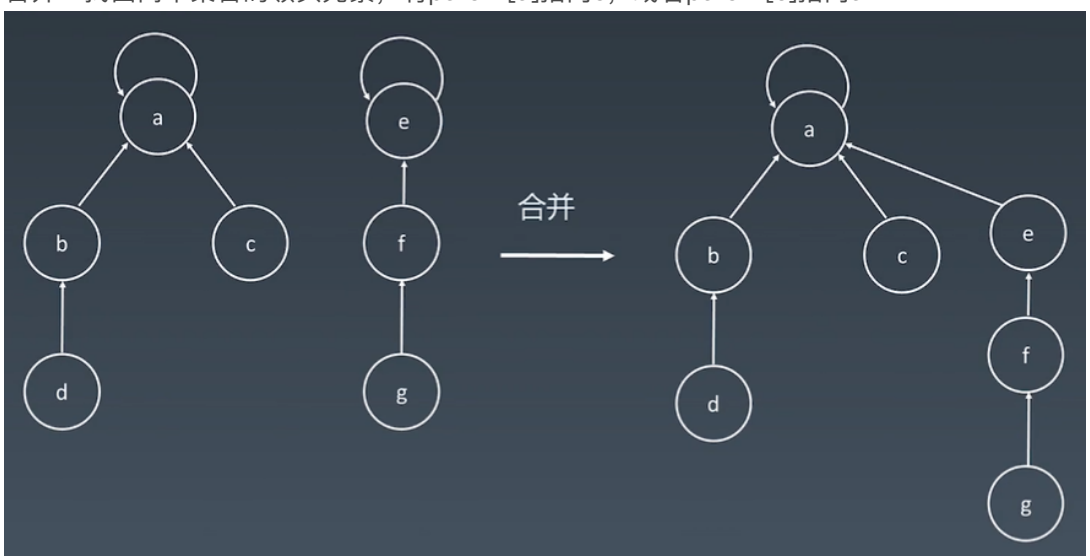
并查集

初始化：每个元素拥有一个parent数组指向自己，表示自己是自己的集合

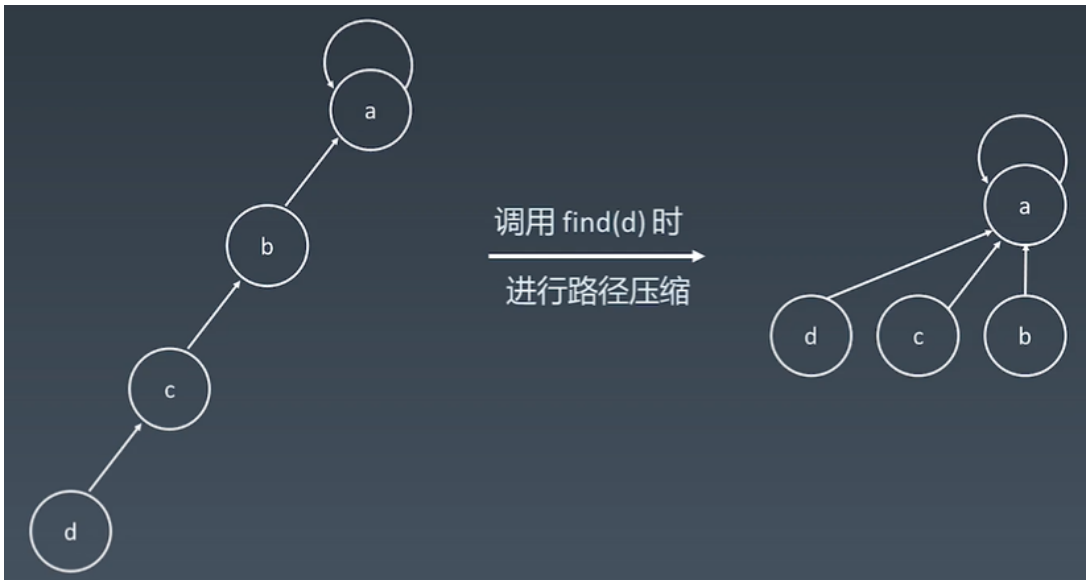
查询，合并

找parent，再找parent，直到parent等于自己，说明找到了领头元素。

合并：找出两个集合的领头元素，将parent[a]指向e，或者parent[e]指向a



路径压缩，查找O(1)



并查集模板

```
class UnionFind {
    private int count = 0; //size
    private int[] parent;
    public UnionFind(int n) {
        count = n;
        parent = new int[n]; //size为n，初始化一个长度为n的数组
        for (int i = 0; i < n; i++) {
            parent[i] = i;//初始化，自己是自己的领头元素
        }
    }
    public int find(int p) { //不断往上找parent
        while (p != parent[p]) { //集合领头元素的特点，parent[i] == i
            parent[p] = parent[parent[p]];
            p = parent[p];
        }
        return p;
    }
    public void union(int p, int q) {
        int rootP = find(p); //找p的领头元素
        int rootQ = find(q); //找q的领头元素
        if (rootP == rootQ) return;
        parent[rootP] = rootQ; //把rootQ作为rootP的领头元素
        count--;//独立的集合减少了一个
    }
}
```

python模板

```
def init(p):
    # for i = 0 .. n: p[i] = i;
    p = [i for i in range(n)]

def union(self, p, i, j): #找到各自集合的领头元素，合并
    p1 = self.parent(p, i)
    p2 = self.parent(p, j)
```

```
    p[p1] = p2

def parent(self, p, i):
    root = i
    while p[root] != root:
        root = p[root] #不断找parent
    while p[i] != i: # 路径压缩，把经过元素的parent都赋成root
        x = i; i = p[i]; p[x] = root
    return root
```

547朋友圈

方法一：DFS

```
class Solution{
    public int findCircleNum(int[][] M) {
        /**
        visited数组，表示这个学生是否被记过数了，依次判断每个节点
        如果其未访问，朋友圈数加1并对该节点进行dfs搜索标记所有访问到的节点
         */
        boolean[] visited = new boolean[M.length];
        int ret = 0;//return value，有多少个朋友圈
        //遍历所有学生，如果未访问过，ret++，并且dfs这个学生和他所有的朋友
        for (int i = 0; i < M.length; ++i) {
            if (!visited[i]) {
                dfs(M, visited, i);
                ret++;
            }
        }
        return ret;
    }
    private void dfs(int[][] m, boolean[] visited, int i) {
        for (int j = 0; j < m.length; ++j) {
            if (m[i][j] == 1 && !visited[j]) {
                //i 和 j 是朋友，且j没有被访问过，j也要被访问过
                visited[j] = true;
                //再从 j 继续扩散朋友
                dfs(m, visited, j);
            }
        }
    }
    //没有terminator，特殊情况，terminator就是visited都访问过了
    //当前学生是i，遍历所有学生，如果 i 和 j 是朋友，且j没有被访问过，j也要被染色，算作已访问
    //再从 j 继续扩散朋友
}
```

方法二：并查集

```
class Solution:
    def findCircleNum(self, M: List[List[int]]) -> int:
```

```
        if not M: return 0
        n = len(M)
        p = [i for i in range(n)]
        for i in range(n):
            for j in range(n):
                if M[i][j] == 1:
                    self._union(p, i, j) #合并i和j
        return len(set([self._parent(p, i) for i in range(n)]))
        #看n个节点中不同的parent有几个，每个parent就是一个孤立的群，用set判重
    def _union(self, p, i, j):
        p1 = self._parent(p, i)
        p2 = self._parent(p, j)
        p[p2] = p1
    def _parent(self, p, i):
        root = i
        while p[root] != root:
            root = p[root]
        while p[i] != i:
            x = i; i = p[i]; p[x] = root
        return root
```

高级搜索

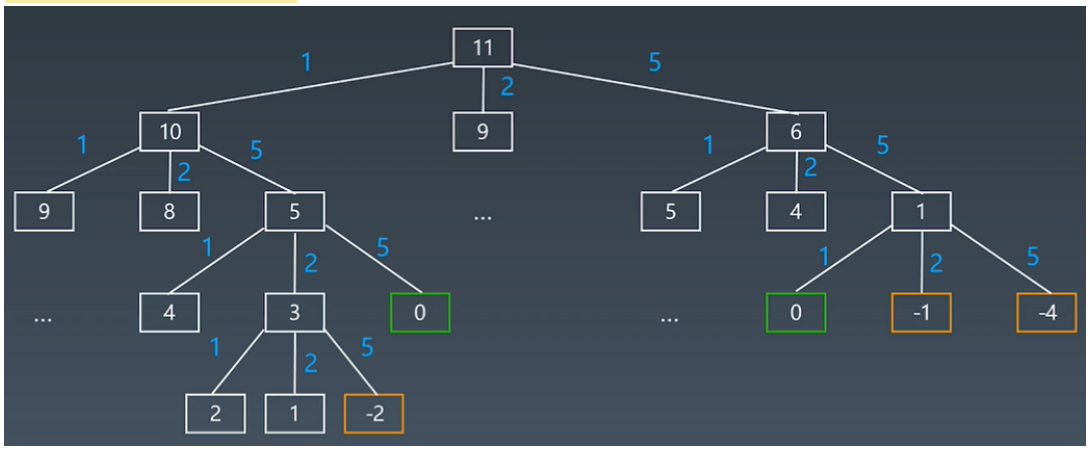
初级搜索

1. 朴素搜索：傻搜/暴力搜索
2. 优化方式：不重复（fibonacci）、剪枝（生成括号问题）
3. 搜索方向：

DFS: depth first search 深度优先搜索：一路不回头撞南墙

BFS: breadth first search 广度优先搜索

搜索问题一定要画搜索树



题目集合：

爬楼梯，零钱兑换 爬楼梯问题转换为零钱兑换问题

括号生成问题的剪枝

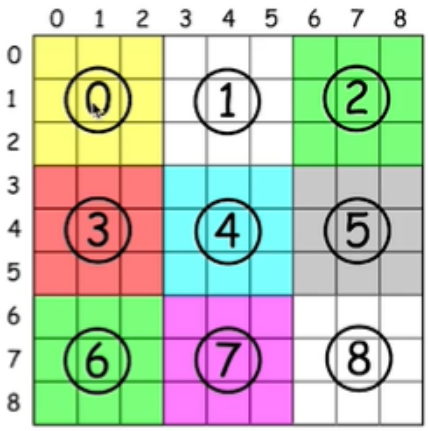
N皇后问题 剪枝

```python
def solveNQueens(self, n):
    def DFS(queens, xy_dif, xy_sum):
        p = len(queens)
        if p==n:
            result.append(queens)
            return None
        for q in range(n):
            if q not in queens and p-q not in xy_dif and p+q not in xy_sum:
                DFS(queens+[q], xy_dif+[p-q], xy_sum+[p+q])
    result = []
    DFS([],[],[])
    return [ ["."*i + "Q" + "."*(n-i-1) for i in sol] for sol in result]
```

有效的数独 看官方题解

37解数独

使用 `方块索引= (行 / 3) * 3 + 列 / 3`
其中 `/` 表示整数除法。



https://leetcode.com/problems/sudoku-solver/discuss/15752/Straight-Forward-Java-Solution-Using-Backtracking

```java
public class Solution {
    public void solveSudoku(char[][] board) {
        if(board == null || board.length == 0)
            return;
        solve(board);
    }

    public boolean solve(char[][] board){
        for(int i = 0; i < board.length; i++){
            for(int j = 0; j < board[0].length; j++){//遍历整个棋盘
                if(board[i][j] == '.'){ //棋盘为空位
                    for(char c = '1'; c <= '9'; c++){//trial. Try 1 through 9
                        if(isValid(board, i, j, c)){//是否合法
                            board[i][j] = c; //Put c for this cell，试着填上c

                            if(solve(board))//递归调用solve
```

```
                                return true; //如果solve能解决，说明整个棋盘都解决了return true。
                                board[i][j] = '.'; //Otherwise go back
                            }
                        }

                        return false;
                    }
                }
            }
            return true;
        }

        private boolean isValid(char[][] board, int row, int col, char c){
            for(int i = 0; i < 9; i++) {
                if(board[i][col] != '.' && board[i][col] == c) return false; //check row
                if(board[row][i] != '.' && board[row][i] == c) return false; //check column
                if(board[3 * (row / 3) + i / 3][ 3 * (col / 3) + i % 3] != '.' &&
board[3 * (row / 3) + i / 3][3 * (col / 3) + i % 3] == c) return false; //check 3*3 block
            }
            return true;
        }
}
```

https://leetcode.com/problems/sudoku-solver/discuss/15911/Less-than-30-line-clean-java-solution-using-DFS

```
public class Solution {
    public void solveSudoku(char[][] board) {
        dfs(board,0);
    }
    private boolean dfs(char[][] board, int d) {
        //9*9共81个格子，每个格子可以填0-9，相当于81层的递归
        if (d==81) return true; //found solution
        int i=d/9, j=d%9;
        if (board[i][j]!='.') return dfs(board,d+1);//prefill number skip

        boolean[] flag=new boolean[10];
        validate(board,i,j,flag);
        for (int k=1; k<=9; k++) {
            if (flag[k]) {
                board[i][j]=(char)('0'+k);
                if (dfs(board,d+1)) return true;
            }
        }
```

```
        board[i]
[j]='.'; //if can not solve, in the wrong path, change back to '.' and out
        return false;
    }
    private void validate(char[][] board, int i, int j, boolean[] flag) {
        Arrays.fill(flag,true);
        for (int k=0; k<9; k++) {
            if (board[i][k]!='.') flag[board[i][k]-'0']=false;
            if (board[k][j]!='.') flag[board[k][j]-'0']=false;
            int r=i/3*3+k/3;
            int c=j/3*3+k%3;
            if (board[r][c]!='.') flag[board[r][c]-'0']=false;
        }
    }
}
```