缘起

- 刷题的链接，https://blog.csdn.net/goldarmour/article/details/129807099?spm=1001.2014.3001.5502
- 42道，目前AC了28道，待debug4道，把手上现有的题，按**知识点**整理一下，毕竟meida的题，我目前的水平，还不是说很轻松的done，目前只是算学习，还是需要**不断理解**，之前知道**概念或思想**，现在通过做题来不断强化。
- 未做的题（9）
    - 37、字符串化繁为简
    - 40、代码编辑器
    - 39、树状结构查询
    - 逻辑分析的4道，**我看到逻辑分析，就头疼，其实也可能说明，它们不好归类**
    - 13、转骰（tou、shai）子
    - 31、计算误码率
    - 34、不开心的小朋友
    - 非逻辑分析
    - 20、区间交集
    - 35、跳房子2
    - 38、评论转换输出
- debug的题（4）
    - 9、
    - 11、
    - 22、找出两个整数数组中同时出现的整数
    - 29、
- 43也算是bfs，只是没统计，
- 未提交的题（1）
    - 16还没写到代码中
- 单纯的序号
    - 9、
    - 11、13、16、
    - 20、22、29
    - 31、34、35、37、38、39、
    - 40、

内容

数据结构：字符串、栈

- 字符串，看着简单，但应该不会考那么简单，都是杂着一起，不行就暴力，**遇到会的技巧，就用技巧**

字符串（3，1）

- 25、查字典（done）
    - 比较简单

```
void od_a25() {
    //单词前缀+字典长度+字典  b 3 a b c  【字典是个有序单词数组】
```

```
    //输出b  【所有含有该前缀的单词】

    //abc 4 a ab abc abcd
    //输出 abc abcd，多个单词换行输出

    string in;
    vector<string> input;
    vector<string> res;

    while (cin >> in) {
        input.push_back(in);
        if (cin.get() == '\n')     //lionel，这个输入输出确实要花时间整一下
            break;
    }
    string pre = input[0];
    int len = stoi(input[1]);

    while (len--) {
        //input.push_back(in);
        //lionel，也可以直接处理
        if (input[len].substr(0, pre.size()) == pre) {
            res.push_back(input[len]);
        }
    }


    if (res.empty()) {
        cout << "null" << endl;
    } else {
        for (auto c : res) {
            cout << c << endl;
        }
    }
}
```

- 37、字符串化繁为简
  - *题能看懂，不确定是不是用到并查集*
  - *把 () 的都合并到一个string里，然后去匹配，找到就换成第1个?*
- 40、代码编辑器
  - *未看*

栈（4，3）

- 2、找最小数
  - *这个核心在于，case通过率的问题*

```
void od_a02_impl(string input, int k) {
    //健壮性
    if (k == 0) {
        cout << input << endl;
    }

    if (input.size() == 0 || k > input.size()) {
        cout << "0" << endl;
```

```
    }

    stack<char> res;
    for (int i = 0; i < input.size(); i++) {
        char current = input[i];
        while (!res.empty() && res.top() > current && k>0){
            res.pop();
            k--;
        }
        res.push(current);//lionel，啥时候用呢，这是最小值，要是取最大值呢？
    }

    while (k > 0) {
        res.pop();  //也可以用for，如果还需要移动数目
        k--;  //这个忘记写了，其实可以写成while(k--) res.pop();  这种直接抄代码啊，确实没法看，当然，最开始还写的是for循环呢
    }

    //把栈转成string，并转置一下
    string s;
    while (!res.empty()) {
        s += res.top();
        res.pop();
    }
    reverse(s.begin(), s.end());

    //把转置前面的0给去掉，如果有0的话
    int pos = 0;
    while (pos < s.size() && s[pos] == '0') {
        pos++;
    }
    if (pos == s.size()) {
        cout << "0" << endl;
    }

    cout << s.substr(pos) << endl;  //lionel，substr()的新用法
}
```

- 3、解压报文

```
void od_a03_impl(string input) {
    stack<int> stk;
    string replace;
    string repeat;
    string val;

    for (char c : input) {
        if (c == ']') {
            while ((stk.size() > 0) && (stk.top() != '[')) {
                replace += stk.top();
                stk.pop();
            }

            reverse(replace.begin(), replace.end());
```

```
            if (stk.size() > 0 && stk.top() == '[') {   //lionel,这个if是在
if(c==']')的里面
                stk.pop();//把`[`出掉
                while (stk.size() > 0 && isdigit(stk.top())) {
                    repeat += stk.top();
                    stk.pop();
                }


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

                cout << repeat << endl;//lionel，第2个case有问题，用例写错了
                int num = stoi(repeat);
                repeat.clear();
                while (num) {
                    val += replace;
                    num--;
                }
                replace.clear();
                //cout << val << endl;

                for (int i = 0; i < val.size(); i++) {
                    stk.push(val[i]);  //lionel，这时候把替换过的字符再插入回去
                }
                val.clear();  //lionel，卡在了，每次都clear()掉才行
                continue;//lionel，这个continue是我没想到的
            }
        }

        stk.push(c);  //表示，只有不为`[`的时候，才push进来
    }

    string res;
    while (!stk.empty()) {
        res += stk.top();
        stk.pop();
    }
    reverse(res.begin(), res.end());
    cout << res << endl;
}
```

- 11、仿LISP计算（debug）
- 17、阿里巴巴找黄金宝箱VI

```
//lionel，不确定，这个地方，input是不是也要改为引用
void findNextBig(vector<int> inputs, stack<pair<int, int>>& stk, vector<int>&
res) {
    for (int i = 0; i < inputs.size(); i++) {
        int ele = inputs[i];
        while (1) {
            if (stk.size() == 0) {
                stk.push(make_pair(ele, i));
                break;
            } else {
                pair<int, int> val = stk.top();
```

```
                int peekEle = val.first;
                int peekIdx = val.second;
                if (ele > peekEle) {
                    res[peekIdx] = ele;
                    stk.pop();
                } else {
                    stk.push(make_pair(ele, i));
                    break;
                }
            }
        }
    }
}

void od_a17() {
    vector<int> input{ 2,5,2 };  //lionel，这个怎么输入，自己写个
    stack<pair<int, int>> stk;
    vector<int>res(input.size(), -1);
    findNextBig(input, stk, res);
    if (stk.size() != 1)
        findNextBig(input, stk, res);  //lionel，为啥这么写？

    string str;
    for (auto c : res) {
        str += to_string(c);
        str += ",";
    }
    cout << str << endl;
}
```

- 自己整理的模板

```
//卡在哪里呢，就是处理完后，要把新值再push回去；第2个就是stack<int>，可以是一个结构体，这个思路要打开
 "3[m2[c]]"，处理好里的后mcc后，要push回去
```

bfs、dfs、回溯、dijkstra

- 这几个，**各有异同，同时还有模板**，我就放在一起理解了
- dfs和bfs，可以相互转换使用，*我目前，dfs，还只会用递归*

bfs（4，3）

- 01宜居星球改造计划

```
int dirs01[4][2] = { {-1,0},{1,0},{0,-1},{0,1} };
using pii = pair<int, int>;

void bfs(queue<pii> &q, vector<vector<string>> &input,int row, int col, int &no_count) {
    int result = 0;
    while (q.size() && no_count) {
        queue<pii> newq;
```

```
        //lionel，知道错在哪了，是每个宜居区，这个地方就必须是for()了  【这个地方太扯了，没想清楚】
        for (int i = 0; i < q.size(); i++) {
            int x = q.front().first;  //lionel，队列没有top，只有front
            int y = q.front().second;

            q.pop();
            for (auto dir : dirs01) {
                int mx = x + dir[0];
                int my = y + dir[1];

                cout << mx << "," << my << "before" << endl;
                if (mx >= 0 && mx < row && my >= 0 && my < col && input[mx][my] == "NO") {
                    input[mx][my] = "YES";
                    cout << mx << "," << my << endl;
                    newq.push(make_pair(mx, my));
                    no_count--;
                }
            }

        }
        cout << "result" << result << endl;
        result++;
        while (!q.empty()) q.pop();
        q = newq;
    }

    if (no_count == 0)
        cout << result << endl;
}


void od_a01_impl(vector<vector<string>>& input) {
    int row = input.size();
    int col = input[0].size();
    queue<pii> q;
    int no_count = 0;

    for (int i = 0; i < row; ++i) {
        for (int j = 0; j < col; ++j) {
            if (input[i][j] == "YES") {
                q.push(make_pair(i, j));
            } else if (input[i][j] == "NO") {
                no_count++;
            }
        }
    }
    bfs(q, input, row, col, no_count);
}
```

- 32、返回矩阵中非1元素的个数

```
```

```
using vvi = vector < vector<int >>; //lionel，自己还不太会用这种，把顺序搞混了， 倒过来的写法 是typedef的用法
using pii = pair<int, int>;
int dirs32[4][2] = { {1,0},{-1,0},{0,-1},{0,1} };
void od_a32_impl(int m, int n, vvi &input) {
    //将数组所有成员随机初始化为0或2,再将矩阵的[0,0]元素修改成1  --lionel，要看题干
    input[0][0] = 1;
    queue<pii> q;
    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < n; ++j) {
            if (input[i][j] == 1) {
                q.push(make_pair(i, j));//lionel，这个make_pair，不是必须吧
            }
        }
    }
    int count = 1;

    while (q.size()) {  //lionel，这个地方是while()
        int x = q.front().first;  //lionel，队列没有top，只有front
        int y = q.front().second;
        q.pop();
        for (auto dir : dirs32) {
            int mx = x + dir[0];
            int my = y + dir[1];

            if (mx >= 0 && mx < m && my >= 0 && my < n && input[mx][my] == 0) {
                input[mx][my] = 1;
                count++;
                q.push(make_pair(mx, my));
            }
        }
    }
    cout << m * n - count << endl;
}

void od_a32() {
    int m1 = 4, n1 = 4;
    vvi inputs1{ {0,0,0,0},{0,2,2,2},{0,2,0,0},{0,2,0,0} };
    od_a32_impl(m1, n1, inputs1);  //预期9，输出9
}
```

- 39、树状结构查询
  - *对树结构，不太熟悉，投得也相对少一些*，理解还是OK的，就是结构体怎么去定义，包括图也是一样的
- 41、周末爬山
  - *这个还是用dfs做的，后面要改成bfs*

```
void od_a41() {
    int max = input1[0][0];

    int steps = 0;
    minSteps[max] = 0;

    dfs(0, 0, 0);  //从开始的(i,j)也就是(0,0)已经走了steps步，第一次steps是0
```

```
    for (auto c : minSteps) {
        if (c.first > max) {
            max = c.first;
            steps = c.second;
        }
    }

    cout << max << "," << steps << endl; //输出是2,2  lionel，关注一下
}

void dfs(int i, int j, int steps) {
    //当前山峰高度
    int curHeight = input1[i][j];
    //4个方向
    for (auto d : direction) {
        int mx = i + d[0];
        int my = j + d[1];

        //cout << mx << "," << my << "," << visited[i][j] << endl;
        //lionel，关注一下，应该是mx>=input1.size()，我写成mx>input1.size()，两个式子，
都小写了一个=号，导致core掉了
        if (mx<0 || mx>=input1.size() || my<0 || my>=input1[0].size() ||
visited[i][j]) {
            continue;
        }

        //下一步山峰
        int nextHeight = input1[mx][my];
        if (abs(curHeight - nextHeight) <= k1) {
            steps++;
            if (!minSteps[nextHeight] || minSteps[nextHeight] > steps) {
                minSteps[nextHeight] = steps;
            }

            //标记为已访问
            visited[i][j] = true;
            //下一个
            dfs(mx, my, steps);
            //回退
            visited[i][j] = false;  //lionel，这就很回溯的思想了
            steps--;
        }
    }
}
```

- 自己整理的模板（*网上说的都太简单了*），**目前都是矩阵，还没用链图**

```
int dirs01[4][2] = { {-1,0},{1,0},{0,-1},{0,1} }; //上下左右的方向

//q是创建的队列，像输入的数组，行列式，看情况是不是要通过变量传
//其它变量，比如要统计一些值（非1元素的个数）
void bfs(q,其它变量){
    while(q.size()){  //队列不空
```

```
        //lionel，01移居星球这套题，就得在这进行for()循环  【这个时候，就要插入到新队列newq里去，最后newq替换原有的q】

        auto val = q.front();//取栈顶元素，一般是pair<int,int>，也就是x,y的坐标
        q.pop();//出队

        for(auto directions: dirs01){
            int mx = x+directions[0];
            int my = y+directions[1];

            //校验mx和my的是否越界，以及visisted[mx][my]是否被访问过，以及要判断input[mx][my]是不是自己要的
            if(mx>=0&& mx <input.size() ...){
                //符合条件的再放到队列里去
                q.push(mx,my);
            }
        }

        //lionel，这就是一个块
    }
}

void init(){
    //根据题意，bfs一般是队列，2层for循环
    vector<vector<bool>> visited ; //创建一个访问标识
    q.push_back(x,y); //把坐标放入队列
//一般不会在for里面调用bfs()
    bfs(q,其它变量);
}
```

dfs（3，2）

- 15、欢乐的周末
  - *用了2个dfs，感觉不太会，想重新思考一下*

```
int m = 4, n = 4;  //长，宽

void dfs(vector<vector<int>> input, int x, int y, vector<vector<int>>& point, vector<vector<int>> &person) {
    if (x < 0 || y < 0 || x >= m || y >= n || input[x][y] == 1 || person[x][y] == 1) {
        return;
    }

    if (input[x][y] == 3 && person[x][y] == 0) {
        point[x][y]++;//访问次数+1
    }

    person[x][y] = 1;  //已经访问过

    //lionel，用for和dirs[4][2]是一样的
    dfs(input, x + 1,y, point, person);
    dfs(input, x, y+1, point, person);
    dfs(input, x - 1, y, point, person);
    dfs(input, x, y-1, point, person);
```

```
}

void od_a15_impl(int m, int n, vector<vector<int>> input){
    //vector<vector<int>> v(m, vector<int>(n, 0)); //感觉就是input啊
    int sum = 0;
    vector<vector<int>> inn(m, vector<int>(n, 0));
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {  //lionel，这个地方j<n，写成了i<n，导致越界访问
            if (input[i][j] == 2)
                inn.push_back({ i, j });  //lionel，我没看懂，这个意义在哪里
        }
    }

    vector<vector<int>> point(m, vector<int>(n, 0));
    vector<vector<int>> hua(m, vector<int>(n, 0));
    vector<vector<int>> wei(m, vector<int>(n, 0));

    dfs(input, inn[0][0], inn[0][1], point, hua);
    dfs(input, inn[1][0], inn[1][1], point, wei);

    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            if (point[i][j] >= 2)
                sum++;
        }
    }

    cout << sum << endl;
}

void od_a15() {
    int m1 = 4, n1 = 4;  //长，宽
    vector<vector<int>> input1{ {2,1,0,3},{0,1,2,1},{0,3,0,0},{0,0,0,0} };  //3表明聚餐的地方，2表示（小华，小为的位置），0可通行，1不能通行
    od_a15_impl(m1, n1, input1);  //输出2
}
```

- 21、寻找最大价值矿堆

```
void dfs(vector<vector<int>>& input, int x, int y, int& island) {
    int row = input.size();
    int col = input[0].size();

    //sum = max(sum, ++cnt);
    island += input[x][y];

    input[x][y] = 0; //修改一下，相当于visited[][]


    for (auto dir : dirs21) {
        int mx = x + dir[0];
        int my = y + dir[1];
        if (mx >= 0 && mx < row && my >= 0 && my < col && input[mx][my]) {
            dfs(input, mx, my, island);
        }
```

```
    }
}

void od_a21_impl(vector<vector<int>> &input) {
    int row = input.size();
    int col = input[0].size();

    int res = 0, island = 0;
    //vector<vector<int>> visited(row, vector<int>(col, 0));
    for (int i = 0; i < row; i++) {
        for (int j = 0; j < col; j++) {
            //lionel，在于这个逻辑不太熟悉
            if (input[i][j]) {
                dfs(input, i, j, island);
                res = max(res, island);
                island = 0;
            }
        }
    }
    cout << res << endl;
}
void od_a21() {
    vector<vector<int>>inputs1{ {2,2,2,2,0},{0,0,0,0,0},{0,0,0,0,0},{0,1,1,1,1}
};
    inputs = inputs1;
    od_a21_impl(inputs1);
}
```

- 29、战场索敌（debug中）
- 自己整理的模板

```
void init(){
    //前面定义与bfs相同，差别在于2层for中调用，无需定义quue【以21最大矿堆为例】
    dfs(i,j,island);
    res=max(res,island);
    island=0;
}

//i，j是坐标，island是要算的值，必须整成引用
void dfs(int i, int j, int &island){
    //因为有当前的坐标i和j了，先计算值，
    island+=input[i][j];
    visitied[i][j]=true;  //表示已经访问过

    for(auto directions: dirs01){  //继续4个方向
            int mx = x+directions[0];
            int my = y+directions[1];

            //校验mx和my的是否越界，以及visisted[mx][my]是否被访问过，以及要判断
input[mx][my]是不是自己要的
            if(mx>=0&& mx <input.size() ...){
                //符合条件的继续递归
                dfs(mx,my,island);
            }
        }
```

```
}
```

回溯（2，1）

- 8、考古学家
  - *lionel，这是一道经典的组合题，***要区分一下，组合、排列，怎么做**

```
void swap08(vector<char> &input, int index, int i) {  //lionel，问题出在这个地方没写成引用，导致交换后没有变化
    if (index != i) {
        char tmp = input[index];
        input[index] = input[i];
        input[i] = tmp;
    }
}

void helper(vector<char> input, int index, vector<string>& result) {
    if (index == 3) {  //lionel，index==input.size()，其实就是3
        string cur;
        for (auto c : input) {
            cur.push_back(c);
        }
        result.emplace_back(cur);
    } else {
        for (int i = index; i < 3; ++i) {  //lionel，排列是交换顺序
            //cout << index << "," << i << endl;
            swap08(input, index, i);
            helper(input, index + 1, result);
            swap08(input, index, i);
        }
    }
}

void od_a08_impl(int n, vector<char> input) {
    //n个全排列，就需要n步，但第一个数字时，就有n个选择，生成第2个数字，就有n-1个选择.....
    vector<string> result;
    helper(input, 0, result);
    for (auto s : result) {
        cout << s << endl;
    }
}

void od_a08() {
    int n1 = 3;  //碎石个数
    vector<char> input1{ 'a','b','c'};
    od_a08_impl(n1, input1);
}
```

- 9、叠积木（debug中）
- 自己整理的模板（*我目前，还没有整理出来，自己还有点懵的*）
  - 41周末爬山，用的的回溯 **【差异体现在最后2行】**

- **如果想思考一下，回溯与dfs的异同，可以用这个点**

```
//我只记得处理后，要回退一下，这是核心
//用的是dfs的模板壳，差异在哪呢，就是dfs()之后，会有回退的动作
void dfs(int i, int j, int steps){
    //当前高度
    int cur = input[i][j];
    //4个方向
    for(){
        int mx;
        int my;
        if(mx>=0&& mx <input.size() ...){//也有visited
            continue;
        }

        //取一个山峰的值
        int next = input[mx][my];
        for(abs(cur-next)<=k){
            steps++;
            //更新相关值
            visited[i][j]=true;

            dfs(mx,my,steps);

            //重点是这个【lionel】  **差异体现在最后2行**
            visited[i][j]=false;
            steps--;
        }
    }
}
```

dijkstra（2，2）

- 05最长广播效应

```
void od_a05() {
    int N, T;
    cin >> N >> T;
    //节点之间互通
    int u, v;
    map<int, set<int>>mp;  //边与边的关系，lionel，这个我不太熟悉
    for (int i = 0; i < T; i++) {
        cin >> u >> v;
        mp[u].insert(v);
        mp[v].insert(u);  //lionel，这是啥意思，保持节点互通？    【记录，当前节点，与邻接节点，互为邻接的关系】
    }

    int start;
    cin >> start;

    queue<int>q;
    set<int> visited;   //lionel，为啥用set？
    int layer = 0; //层数
```

```
    q.push(start);
    visited.insert(start);

    //lionel，差别在这儿？
    while(visited.size()<N){
        int len = q.size();
        for (int i = 0; i < len; i++) {
            int cur = q.front();
            q.pop();

            for (auto val : mp[cur]) {
                if (0 == visited.count(val)) {
                    q.push(val);
                    visited.insert(val);
                }
            }
        }
        ++layer;
    }

    cout << layer * 2 << endl;

#if 0
    //lionel，这个再消化一下
    vector<int> d(N+1);
    //dijkstra的做法
    while (!q.empty()) {
        //每次取最近的节点
        int node = q.front();
        for (auto n : mp[node]) {
            if (!visited.find(n)) {
                visited.add(n);
                d[n] = d[node] + 1;
                q.push(n);
            }
        }
    }

    int res = 0;
    for (int i = 0; i < N + 1; i++) {
        res = max(res, d[i]);
    }
    cout << res * 2 << endl;
#endif
}
```

- 42、最小传输时延1
    - *我理解，这是一道经典的dijkstra的题*
    - 参考**LC743**

```
void od_a42() {
    int n = 3, n = 3;
    int start = 1, end = 3;
    const int inf = INT_MAX / 2;
    vector<vector<int>> tmp_edge{ {1,2,21},{2,3,13},{1,3,50} };
```

```
    vector<vector<int>> g(n, vector<int>(n, inf));

    //lionel，x和y相当于u和v，相当从0开始，所以每次都减1
    for (auto& t : tmp_edge) {
        int x = t[0] - 1;
        int y = t[1] - 1;
        g[x][y] = t[2];
    }

    //lionel，题目上写的，已确定的点，和不确定的点

    vector<int> dist(n, inf);
    dist[start - 1] = 0;
    vector<int> used(n);//把n个元素赋值为0，  [used(n,3)，把n个元素赋值为3]
    for (int i = 0; i < n; i++) {
        int x = -1;
        for (int y = 0; y < n; y++) {
            if (!used[y] && (x == -1 || dist[y] < dist[x])) {
                x = y;
            }
        }
        used[x] = true;
        for (int y = 0; y < n; y++) {
            dist[y] = min(dist[y], dist[x] + g[x][y]);
        }
    }

    cout << dist[end - 1] << endl; //lionel，因为这个地方有终点，如果无终点取所有最小

#if 0
    int ans = *max_element(dist.begin(), dist.end());
    return ans == inf ? -1 : ans;
#endif
}
```

- 自己整理的模板
    - *我怀疑dijistra，其实也可以用bfs去更新的*

```
void init(){
    //从入参u,v，w中去更新edges[][]，二维数组

    //创建一个dist[]，一维数组
    dist[start-1]=0;//初始化
    used[];//是否访问过

    for(int i=0;i<n;i++){
        int x=-1; //开始的u
        for(int y=0;y<n;y++){//v
            //v没有被访问，并且u没有被更新【还是在这个for里】或者 u的值，大于v的值，*后面是啥意思，不太明确*
            if(!used[y] && (x == -1 || dist[y] < dist[x])){ //lionel，这个条件比较重要
                x=y;
            }
```

```
            }
            used[x]=true;//lionel，跳出来再置为true
            for(int y=0;y<n;y++){
                dist[y]=min(dist[y],dist[x]+g[x][y]);  //这个方法是从leetcode上看出来的，但有些人会用bfs的方式来实现
            }
        }

        //lionel，要想一下，教科书上的实现，是怎么写的？
    }

    //最长广播效应用的就是bfs的方式
    把start放入queue里，又不断的找map<int,int>，*lionel，甚至它为啥定义成这样的map,我没太懂，它这个区别在于，没有表示weight，至少没有明面的表示weight
```

算法：贪心、动规、0-1背包

贪心（3，3）

- 07导师请吃火锅

```
//问题1：采用什么样的贪心策略？【x+y是总时间】
void od_a07_impl(int n, int m, vector<pair<int, int>> input) {
    vector<int> val(n, 0);
    for (auto c : input) {
        val.push_back(c.first + c.second);
    }

    sort(val.begin(), val.end());  //贪心就是排序

    int res = 1;//lionel，第1个菜必吃
    int pre = 0;
    for (int i = 1; i < n; i++) {
        if (val[i] > val[pre] + m) {
            res++;
            pre = i;//lionel，这个pre，不就相当于n-1嘛
        }
    }
    cout << res << endl;
}

void od_a07(){
    int n1 = 2;//2个菜；，也就是inputs1的大小
    int m1 = 1;//手速，1秒后才能再捞
    vector<pair<int, int>> inputs1{ {1,2},{2,1} };
    od_a07_impl(n1, m1, inputs1);
}
```

- 14最少面试官数

```
//思路是，1、按结束时间排序，逐个遍历，如果下一个的开始时间，大于等于 上一个的结束时间，就count++
```

```
bool cmp(pair<int, int>m1, pair<int, int> m2) {
    if (m2.second > m1.second) {
        return 1;
    }  //没说结束时间相等的场景，就先这样
    return 0;
}

void od_a14_impl(int m_count, int n_total, vector<pair<int, int>> input) {
    sort(input.begin(), input.end(), cmp);

    int lastTime = 0;
    int interviewCount = 0;
    for (int i = 0; i < n_total; i++) {
        if (input[i].first >= lastTime) {
            interviewCount++;
            lastTime = input[i].second;
        }
    }

    cout << (interviewCount / m_count) + 1 <<endl;  //lionel，如果不是正好，加1是可以的，如果正好，加1就不对
}

void od_a14() {
    int m_count1 = 2;//面试官最多面试人次
    int n_total1 = 5;//总的面试场次

    vector<pair<int, int>> input1 = { {1,2},{2,3},{3,4},{4,5},{5,6} };//每场的开始和结束时间
    od_a14_impl(m_count1, n_total1, input1);
}
```

- 27、数字序列比大小

```
bool cmp27(vector<int> &m1, vector<int> &m2) {  //lionel，要变成 引用 格式，不然局部变量不会影响input的变化

    sort(m1.begin(), m1.end());  //lionel，因为默认是从小到大排列，最后比较的时候，也是逐个比较
    sort(m2.begin(), m2.end());
#if 0
    for (int i = 0; i < 3; i++) {
        if (m2[i] > m1[i]) {
            return 1;
        } else {
            return 0;
        }
    }
#endif
    return 0;
}

void od_a27_impl(int size, vector<vector<int>> input) {
    sort(input.begin(), input.end(), cmp27);
```

```
    int res=0;
    for (size_t i = 0; i < input.size()-1; i++) {
        for (int j = 0; j < size; j++) {
            if (input[i][j] > input[i + 1][j])
                res++;
        }
    }

    cout << res << endl;
}

void od_a27() {
    int size1 = 3;
    vector<vector<int>> input1{ {4,8,10},{3,6,4} };
    od_a27_impl(size1, input1);
}
```

- 自己整理的思路

```
//相当于是自定义排序的加强版
```

动规（3，3）

- 12、高效会议安排

```
void od_a12_impl(int m, int n, vector<pair<int, int>>& input) {
    sort(input.begin(), input.end(), [&](auto& i1, auto& i2) {return i1.first >=
i2.first; });

#if 0
    int len = 0;
    int maxnum = 0;
#endif

    int len = input.size();
    vector<int> dp(len);
    dp[0] = input[0].first + input[0].second;
    for (int i = 1; i < len; i++) {
        //这是动规的转移方程，lionel，我抄的，我没想到
        dp[i] = max(dp[i - 1], dp[i - 1] - input[i - 1].second + input[i].first
+ input[i].second);
    }
    cout << dp[n - 1] << endl;
}

void od_a12() {
    //vector<std::pair<int,int>,std::allocator<std::pair<int,int> > > 不是变量模板
    //https://www.acwing.com/community/content/621232/  【人家也有这个错】
    int m1 = 1, n1 = 1;
    vector<pair<int, int>>inputs1 = { {2,2} };// { make_pair(2, 2) };
//lionel，这个地方是由于我把这行，写到上一行去了，认为它是一个int类型导致的报错
    od_a12_impl(m1, n1, inputs1);  //4，输出4
```

```
    //lionel，代表了，我的输入输出设计得有问题
#if 0
    int m2 = 2, n2 = 2;  //m2表示2组任务，n2=2，表示其中的一组任务，需要2个机器，
    vector<pair<int, int>>inputs2 = { {2,2} };// { make_pair(2, 2) };
    od_a12_impl(m2, n2, inputs1);
#endif
}
```

- 28、跳格子1

```
void od_a28_impl(vector<int> input) {
    //本来写的是res，vector<int> res;

    vector<int> dp(input.size()); //lionel，这个地方不写个dp(input.size())还会crash掉，要注意下
#if 0
    if (input.size() == 1 || input.size() == 2) {
        cout << input[input.size()] << endl;
        res[input.size()] = input[input.size() - 1];
    }
#endif
    //sort(input.begin(), input.end());

    dp[0] = input[0];
    if (input.size() > 1) {
        dp[1] = max(input[0], input[1]);
    }

    //这是人家写的，可以参考下
#if 0
    if (input.size() >= 1) {
        dp[0] = input[0];
    }

    if (input.size() >=2) {
        dp[1] = max(input[0], input[1]);
    }
#endif

    //lionel，动态规划，怎么调整i变化的？
    for (int i = 2; i < input.size(); i++) {
        dp[i] = max(dp[i - 2] + input[i], input[i - 1]);
    }

    sort(dp.begin(), dp.end());  //lionel，这个地方是不是要排个序  [人家用的map类型，说明已经排过序了]

    cout << dp[input.size() - 1] << endl;  //lionel，否则会啥非是最后一个是最大值，尤其在这个条件下
}

void od_a28() {
    vector<int> inputs1{ 1,2,3,1 }; //4
    od_a28_impl(inputs1);
    vector<int> inputs2{ 2,7,9,3,1 }; //12
```

```
    od_a28_impl(inputs2);
    vector<int> inputs3{ 9,1,2,9 };//18
    od_a28_impl(inputs3);
}
```

- 30、跳格子2

```
void od_a30() {
    string s;//2 3 2, lionel，怎么输入到vector<int>中
    getline(cin, s);
    vector<int> nums{ 2,3,2 };

    //2个dp
    vector<vector<int>> dp(2, vector<int>(nums.size(), 0));
    dp[0][0] = nums[0];
    if (nums.size() > 1) {
        dp[0][1] = max(nums[0], nums[1]);
    }

    for (int i = 2; i < nums.size() - 1; i++) {
        dp[0][i] = max(dp[0][i - 2] + nums[i], dp[0][i - 1]);
    }

    sort(dp[0].begin(), dp[0].end());

    dp[1][0] = nums[1];
    if (nums.size() > 1) {
        dp[1][1] = max(nums[1], nums[2]);
    }

    for (int i = 3; i < nums.size(); i++) {
        dp[1][i] = max(dp[0][i - 2] + nums[i], dp[0][i - 1]);
    }

    sort(dp[1].begin(), dp[1].end());

    int res = max(dp[0][nums.size() - 1], dp[1][nums.size() - 1]);
    cout << res << endl;
}
```

- 整理的模板
  - **一维的都比较简单**，找到递归式子

```
//30是个进阶版，相当于循环，拆分成2个一维，这个leetcode有经典题目
```

0-1背包（3，3）

- 18、通过软盘拷贝文件

```
void od_a18_impl(int num, vector<int> input) {
    int target = 1474560;
    int sum = 0;
```

```
    sort(input.begin(), input.end());
    for(int i=0;i<num;i++){
        sum += input[i];
        if (sum > target) {
            cout << sum - input[i] << endl;
            break;
        }
    }
}


void od_a18() {
    int num1 = 3;
    vector<int> input1{737270,737272,737288};
    od_a18_impl(num1, input1);  //输出 1474542

    int num2 = 6;
    vector<int> input2{ 400000,200000,200000,200000,400000,400000 };
    od_a18_impl(num2, input2);  //输出，1400000，它这个case问题，我感觉单纯用 贪心 还不
一定行
}
```

- 19、代表团坐车

```
void od_a19_impl(vector<int> input) {
    //想法是用二维数组，空间大小正好是数量，不过又回到 回溯上，当前的数组正好为10，加入到
result里去


    //lionel，核心问题是没有想到如何构建 表达式
    sort(input.begin(), input.end());
    vector<vector<int>> dp(input.size(), vector<int>(10+1, 0));

    //第一列初始化为1【原因，空间为0时，不论可选元素范围是多少，只有一种方案-不选取任何元素】
    for (int i = 0; i < input.size(); i++) {
        dp[i][0] = 1;
    }

    dp[0][input[0]] = 1; //多了一列的原因？

    for (int i = 1; i < input.size(); i++) {
        for (int j = 0; j < 10 + 1; j++) {
            if (j - input[i] >= 0) {//10与input[i]逐个比较的过程？
                dp[i][j] = dp[i - 1][j] + dp[i - 1][j - input[i]];
            } else {
                dp[i][j] = dp[i - 1][j];
            }
        }
    }

    cout << dp[input.size() - 1][10] << endl;
}

//作者dp[i][j]的思路：i表示input中的0到i的元素，j表示客车容纳的人数  dp[i][j]表示限定的几种
方法
```

```
void od_a19() {
    vector<int> input1{ 5,4,2,3,2,4,9 };
    int target = 10;
    //输出4，坐满汽车的解决方案数量，【lionel,相当于最多值】，想了一下，只等是等于10，它应该是组合，2+3+5；4+4+2；2+3+5,2+4+4
    od_a19_impl(input1);
}
```

- 26、MELON的难题

```
void od_a26_impl(int m, vector<int> input) {
    int sum = 0;
    sum = accumulate(input.begin(), input.end(), sum);

    sort(input.begin(), input.end(),greater<int>()); //从到到小排

    vector<int> val;
    int target = sum / 2;
    for (int i = 0; i < input.size(); i++) {
        if (input[i] <= target) {
            target -= input[i];
            val.push_back(input[i]);  //lionel，我的有问题，我只算出一种，题目意思应该是所有场景都找出来，然后找最小的情况
        }
    }

    int res = min(input.size() - val.size(), val.size());
    cout << res << endl;
}
void od_a26() {
    int m1 = 4;
    vector<int> inputs1{ 1,1,2,2 };
    //od_a26_impl(m1, inputs1);  //2

    int m2 = 10;
    vector<int> inputs2{ 1,1,1,1,1,9,8,3,7,10 };
    od_a26_impl(m2, inputs2);  //3，答案是3，我输出的4
}
```

- 整理的模板

```
//这种题，一般都可以用贪心、动规 来完成
```

新学的：并查集、KMP

并查集（2，2）

- 04We Are a team

```
vector<int> fa;//没想到要用 typedef 或者 using，来省略一下，lionel
void init(int len) {
    fa.resize(len+1);//lionel，这个地方给的size太小了，导致越界，但根因是不是这个，不确定
    for (int i = 0; i < len; i++) {
        fa[i] = i;
    }
}

int find(int x) {
    if (x != fa[x]) {
        fa[x] = find(fa[x]);
    }
    return fa[x];
}

void merge(int x, int y) {

    int mx = find(x);
    int my = find(y);
    //lionel，不需要判断？
    fa[x] = y;
    //lionel，后面是我根据正规写的，也是可以运行
#if 0
    x = find(x);
    y = find(y);
    if (x > y) {
        fa[x] = y;
    } else {
        fa[y] = x;
    }
#endif
}

void od_a04_impl(int n, int m, vector<vector<int>> input) {
    init(n);//总共多少人
    for (int i = 0; i < m; i++) {
        int x = input[i][0];
        int y = input[i][1];
        int p = input[i][2];

        //lionel，核心在这个地方，我没懂搞的是，啥时候要merge()，啥时候要find()，后来看代码，又看看题意，逻辑都在题干上，要多看看
        if (x<1 || x>n || y<1 || y>m) {
            cout << "da pian zi" << endl;
        } else {
            if (p == 0) {
                merge(x, y);
            } else if (p == 1) {
                if (find(x) == find(y)) {
                    cout << "we are team" << endl;
                } else {
                    cout << "we are not team" << endl;
                }
```

```
            } else {
                cout << "da pian zi" << endl;
            }
        }

    }
}

void od_a04() {
    int n1 = 5, m1 = 7;
    vector<vector<int>> inputs1{ {1,2,0},{4,5,0},{2,3,0},{1,2,1},{2,3,1},
{4,5,1},{1,5,1} };
    od_a04_impl(n1,m1,inputs1);
}
```

- 6、服务失效判断

```
class UF {
public:
    string find(string word, bool create = false) {
        if (!parents.count(word)) {
            if (!create)
                return word;
            return parents[word] = word; //lionel，不是太懂这个
        }

        string w = word;
        while (w != parents[w]) {
            parents[w] = parents[parents[w]];
            w = parents[w];
        }
        return parents[w];
    }

    bool merge(string s1, string s2) {
        string p1 = find(s1,true);  //lionel，忘记写这个
        string p2 = find(s2,true);

        if (p1 == p2)
            return false;
        parents[p1] = p2;
        return true;
    }
private:
    unordered_map<string, string> parents;
};

void od_a06() {
    UF a;

    vector<pair<string, string>> inputs{ {"a1","a2"},{"a5","a6"},{"a2","a3"} };
    for (auto c : inputs) {
        a.merge(c.first, c.second);
    }
```

```
    vector<string> found{ "a5","a2" };
    for (auto c : found) {
        cout << a.find(c) << endl;
    }
}
```

- 整理的模板
    - 2个进阶：1、简单的并、查，**可以压缩**，这个还没看
    - 2、现在都是int型，**string型，怎么办**

```
//简单的并、查实现
vector<int> fa;//没想到要用 typedef 或者 using，来省略一下，lionel
void init(int len) {
    fa.resize(len+1);
    for (int i = 0; i < len; i++) {
        fa[i] = i;
    }
}

int find(int x) {
    if (x != fa[x]) {
        fa[x] = find(fa[x]);
    }
    return fa[x];
}

void merge(int x, int y) {

    int mx = find(x);
    int my = find(y);
    fa[x] = y;
}

//lionel，抄的这个
class UF {
public:
    string find(string word, bool create = false) {
        if (!parents.count(word)) {
            if (!create)
                return word;
            return parents[word] = word; //lionel，不是太懂这个
        }

        string w = word;
        while (w != parents[w]) {
            parents[w] = parents[parents[w]];
            w = parents[w];
        }
        return parents[w];
    }

    bool merge(string s1, string s2) {
        string p1 = find(s1,true);  //lionel，忘记写这个
        string p2 = find(s2,true);
```

```
        if (p1 == p2)
            return false;
        parents[p1] = p2;
        return true;
    }
private:
    unordered_map<string, string> parents;
};
```

KMP（1，1）

- 33、最小循环子数组

```
void getNext(int *next, string s) {
    next[0] = 0;
    int j = 0;
    for (int i = 1; i < s.size(); i++) {
        while (j > 0 && s[i] != s[j]) {
            j = next[j - 1];
        }
        if (s[i] == s[j]) {  //lionel，next数组这个差不多了，这个地方写错了，把==写成了=
            j++;
        }
        next[i] = j;
    }
}

void od_a33_impl(int n, vector<int> nums) {
    if (nums.size() == 0) {
        cout << "null" << endl;
        return;
    }

    int next[10]; //lionel，必须是常量
    string tmp;
    for (auto c : nums) {
        tmp += to_string(c);
    }
    getNext(next, tmp);
    int len = tmp.size();
    int L = len - next[len - 1];//lionel，重点还在于 理解 什么是**最小循环节**
    while (L--) {
        cout << tmp[L] << " ";
    }

#if 0
    //这个不知道抄的谁的
    cout << tmp[next[1]] << "," << next[2] << "," << next[3] << endl;
    //if (next[len - 1] != 0 && len % (len - next[len - 1]) == 0)
        //cout << next << endl;
#endif
}

void od_a33() {
    int n = 9;
```

```
    vector<int> nums{ 1,2,1,1,2,1,1,2,1 };
    od_a33_impl(n, nums);
}
```

- 整理的模板

```
void getNext(int *next, string s) {
    next[0] = 0;
    int j = 0;
    for (int i = 1; i < s.size(); i++) {
        while (j > 0 && s[i] != s[j]) {
            j = next[j - 1];
        }
        if (s[i] == s[j]) {
            j++;
        }
        next[i] = j;
    }
}
```

其它

模拟入队出队

- 10、打印任务排序
    - *其实没太懂*，我没想到的点是**vector里去存了结构体**

```
struct Item {
    int prio;
    int idx;
};

void od_a10() {
    string str;  // 输入9,3,5
    getline(cin, str);

    //分割逗号，这个输入可能会用到
    vector<Item> inputvi;
    str += ",";
    int id = 0;
    while (str.find(",") != string::npos) {
        int idx = str.find(",");
        string t = str.substr(0, idx);
        str = str.substr(idx + 1);
        inputvi.push_back({ stoi(t), id++ });// 插入结构体
    }

    vector<Item> outvi;
    while (inputvi.size()) {
        int prio = inputvi[0].prio;

        if (inputvi.size() == 1) {  //lionel，这个还没懂，啥意思
            outvi.push_back(inputvi[0]);
            inputvi.clear();
```

```
            break;
        }

        bool find = false;

        for (auto it = inputvi.begin() + 1; it != inputvi.end(); it++) {
            if (it->prio > prio) {
                find = true;
                break;
            }
        }

        if (find) {
            inputvi.push_back(*inputvi.begin());
            inputvi.erase(inputvi.begin());  //lionel，也没太懂
        }

        outvi.push_back(inputvi[0]);
        inputvi.erase(inputvi.begin());
    }

    for (int i = 0; i < outvi.size(); ++i) {
        cout << outvi[i].idx;   //输出0，2，1
        if (i != outvi.size() - 1) {
            cout << ",";
        }
    }
}
```

逻辑分析

- 13、转骰（tou、shai）子
- 22、找出两个整数数组中同时出现的整数
- 31、计算误码率
    - *这个看到有原型*，暂未做
- 34、不开心的小朋友

二分

- 16、最佳植树距离
    - *lionel，其实做了，只不过没写到代码仓里*，写在某个纸上了

区间合并

- 20、区间交集

双指针（1，1）

- 23、数据最节约的备份方法

```
void od_a23_impl(vector<int> input) {
    sort(input.begin(), input.end());

    int left = 0, right = input.size() - 1;
```

```
    int target = 500;
    int res = 0;

    while (left < right) {
        if (input[right] > target) {
            right--;
        } else if (input[right] == target) {
            right--;
            res++;
        } else if (input[right] + input[left] <= target) {
            res++;
            right--;
            left++;
        }
    }

    cout << res << endl;
}

void od_a23() {
    vector<int> input1{ 100,500,300,200,400 }; //3
    od_a23_impl(input1);
    vector<int> input2{ 1,100,200,300 };  //2
    od_a23_impl(input2);
    vector<int> input3{ 1,100,200,200,200,300 };//3
    od_a23_impl(input3);
}
```

前缀和（1，1）

- 24、数字游戏

```
void od_a24_impl(int n, int m, vector<int> input) {
    vector<int> val(n, 0);  //lionel，遇到个问题，没有初始化会core掉，初始化方法不对，我用的是{n,0}表示初始化2个元素，只有使用()才行，表示把n个元素初始化为0
    for (int i = 0; i < input.size(); i++) {
        if (i == 0) {
            val[i] = input[i];
        } else {
            val[i] = val[i - 1] + input[i];
        }
    }

    for (auto num : val) {
        if (num % m == 0) {
            cout << "1" << endl;
            return;
        }
    }
    cout << "0" << endl;
}

void od_a24() {
    int n1 = 6, m1 = 7;
    vector<int> inputs1{ 2,12,6,3,5,5 };
```

```
    od_a24_impl(n1, m1, inputs1);

    int n2 = 10, m2 = 11;
    vector<int> inputs2{ 1,1,1,1,1,1,1,1,1,1 };
    od_a24_impl(n2, m2, inputs2);
}
```

三数之和

- 35、跳房子2

二叉树后序遍历（1，1）

- 36、完全二叉树非叶子部分后续序列

```
void dfs(vector<int>& inputs, int root, vector<int>& res) {
    int left = 2 * root + 1;
    int right = 2 * root + 2;   //lionel，这种套跑，我是没想到的
    if (inputs.size() > left) {
        dfs(inputs, left, res);
        if (inputs.size() > right){
            dfs(inputs, right, res);  //这种dfs方式，我也不会，以及怎么组织？
        }

        res.push_back(inputs[root]);  //lionel，这个地方就写错了，写成了
res.push_back(root)了，应该是值，inputs[root]
    }
}

void od_a36() {
    vector<int> inputs{ 1,2,3,4,5,6,7 };
    vector<int> res;

    dfs(inputs, 0, res);

    for (auto c : res) {
        cout << c<<" ";
    }
}
```

递归

- 38、评论转换输出

最后

- 代码目前提交在，https://gitee.com/fewolflion/BookNote/tree/master/00leetcode/00hw-od/2023yearB/02%E8%BF%9B%E9%98%B6%E9%A2%98
- 后面会放到github上，估计能AC到35道的样子，10月31号左右