2022.11.9

每日一题 764 最大加号标志

非常经典的前缀数组的应用。

首先我们想对每一个位置都去无脑向四个方向搜索，因为这个题的 n 很小其实这样能过。不过我们仔细考虑下发现有很多地方被重复搜索过，因此启发我们能不能在遍历前就把这些信息预处理出来。这就是前缀树组的来由。

我们记数组 $left[i][j]$ 表示在(i,j) 这个位置向左看去，能连接的最多的 1 的个数，因此有：

$$left[i][j+1] = \begin{cases} 0, & grid[i][j] = 0 \ or \ grid[i][j+1] = 0 \\ left[i][j]+1, & else \end{cases}$$

同样地，我们也可以预处理出来 $up[i][j], down[i][j], right[i][j]$ ，最后的结果只需在四个数组里取最小值即可。时间复杂度$O(n^2)$

```
class Solution {
public:
    int orderOfLargestPlusSign(int n, vector<vector<int>>& mines) {
        vector<vector<int>> maps(n+1,vector<int>(n+1,1));
        for(auto points : mines)    maps[points[0]][points[1]] = 0;

        vector<vector<int>> up(n+1,vector<int>(n+1,0));
        vector<vector<int>> down(n+1,vector<int>(n+1,0));
        vector<vector<int>> left(n+1,vector<int>(n+1,0));
        vector<vector<int>> right(n+1,vector<int>(n+1,0));
        for(int i = 0; i < n; i++)
            for(int j = 1; j < n; j++) {
                left[i][j] = maps[i][j-1] == 1 ? left[i][j-1] + 1 : 0;
                up[j][i] = maps[j-1][i] == 1 ? up[j-1][i] + 1 : 0;
                if(!maps[i][j]) {up[i][j] = 0; left[i][j] = 0;}
            }

       for(int i = 0; i < n; i++)
            for(int j = n-2; j >= 0; j--) {
                right[i][j] = maps[i][j+1] == 1 ? right[i][j+1] + 1 : 0;
                down[j][i] = maps[j+1][i] == 1 ? down[j+1][i] + 1 : 0;
                if(!maps[i][j]) {down[i][j] = 0; right[i][j] = 0;}
            }

        int ans = 0;
        for(int j = 0; j < n; j++)
            for(int i = 0; i < n; i++) {
                int tmp1 = min(up[i][j],down[i][j]);
                int tmp2 = min(left[i][j],right[i][j]);
                int tmp = min(tmp1,tmp2);
                if(tmp!=0) ans = max(ans,tmp+1);
                if(maps[i][j]) ans = max(ans,1);
            }
        return ans;
    }
```

```
};
```

这也是一类常规的**预处理+模拟**的类型题。或者也有一种说法，像这种求最大面积类问题均可往 DP 上思考。例如：

84 柱状图中最大的矩形

85 最大矩形

221 最大正方形

均为此类的典型题，在此不多做赘述。

430 扁平化多级双向链表

简单的 DFS 或者也可以看做是栈的应用、不多说直接上代码。

```
/*
// Definition for a Node.
class Node {
public:
    int val;
    Node* prev;
    Node* next;
    Node* child;
};
*/

class Solution {
public:
    Node* flatten(Node* head) {
        stack<Node*> st;
        st.push(head);
        Node *pre = NULL;
        while(!st.empty()){
            Node* now = st.top();   st.pop();
            if(pre && pre->next == NULL) {
                pre->next = now;
                now->prev = pre;
            }
            if(now && now->next) st.push(now->next);
            if(now && now->child) {
                st.push(now->child);
                now->next = now->child;
                now->child->prev = now;
                now->child = NULL;
            }
            pre = now;
        }
        return head;
    }
};
```

或者使用 DFS 写：

```cpp
class Solution {
public:
    Node* flatten(Node* head) {

        function<Node*(Node*)> dfs = [&](Node* node) {  //类似于一个内部函数
            Node* cur = node;   // 记录链表的最后一个节点
            Node* last = nullptr;
            while (cur) {
                Node* next = cur->next; //  如果有子节点，那么首先处理子节点
                if (cur->child) {
                    Node* child_last = dfs(cur->child);
                    next = cur->next;   //  将 node 与 child 相连
                    cur->next = cur->child;
                    cur->child->prev = cur;
                    if (next) { //  如果 next 不为空，就将 last 与 next 相连
                        child_last->next = next;
                        next->prev = child_last;
                    }
                    // 将 child 置为空
                    cur->child = nullptr;
                    last = child_last;
                }
                else last = cur;
                cur = next;
            }
            return last;
        };

        dfs(head);
        return head;
    }
};
```