5 DFS/BFS

5.1 problem list

- 来源列表:深度优先搜索

id	title	label
1	p104. 二叉树的最大深度 easy	
2	p101. 对称二叉树 easy	
3	p733. 图像渲染 easy	
4	p111.二叉树的最小深度 easy	
5	p690. 员工的重要性 easy	
6	p841. 钥匙和房间 medium	
7	p113. 路径总和 II medium	
8	p130. 被围绕的区域 medium	
9	p417. 太平洋大西洋水流问题 medium	
10	p542. 01 矩阵 medium	
11	p473. 火柴拼正方形 medium	
12	p773 滑动谜题 hard	

5.2 unofficial solution

5.2.1 p104. 二叉树的最大深度

递归：当前结点的最大深度 = 子节点的最大深度 + 1

O(N)

```
class Solution {
public:

    int maxDepth(Node* root) {
```

```
        if(root==NULL)
            return 0;
        int mxsondepth = 0;
        for(int i = 0; i < root->children.size(); ++i)
        {
            Node *son = root->children[i];
            int sondepth = maxDepth(son);
            mxsondepth = max(mxsondepth,sondepth);
        }
        return mxsondepth+1;
    }
};
```

5.2.2 p101. 对称二叉树

递归

- 判断左子树与右子树是否对称
- 判断两棵树是否对称，即判断树1的左（右）子树与树2的右（左）子树是否对称

```
class Solution {
private:
    bool isTreesSymmetric(TreeNode* root1, TreeNode* root2) {
        if (!root1 && !root2)  return true;
        if (root1 && root2 && root1->val == root2->val) {
            return isTreesSymmetric(root1->left, root2->right)
                && isTreesSymmetric(root1->right, root2->left);
        }
        return false;
    }
public:
    bool isSymmetric(TreeNode* root) {
        return !root || isTreesSymmetric(root->left, root->right);
    }
};
```

5.2.3 p733. 图像渲染

解法一:dfs

```
class Solution {

public:
    vector<vector<int>> floodFill(vector<vector<int>>& image, int sr, int sc, int newColor) {

        int oldColor = image[sr][sc];
        if(oldColor!=newColor)
            dfs(image,sr,sc,newColor,oldColor);
        return image;
    }

private:
    void dfs(vector<vector<int>>& image, int sr, int sc, int newColor,int oldColor)
    {

        int r = image.size();
        int c = image[0].size();
        int dirx[] = {-1,0,1,0};
        int diry[] = {0,-1,0,1};

        image[sr][sc]=newColor;
        for(int i = 0; i < 4; ++i)
        {
            int x = sr+dirx[i];
            int y = sc+diry[i];
            if(x>=r||x<0||y>=c||y<0||image[x][y]!=oldColor)
                continue;

            dfs(image, x, y, newColor,oldColor);
        }

    }
};
```

解法二:bfs

```
class Solution {

public:
    vector<vector<int>> floodFill(vector<vector<int>>& image, int sr, int sc, int newColor) {

        int oldColor = image[sr][sc];
        if(oldColor!=newColor)
            bfs(image,sr,sc,newColor,oldColor);
        return image;
    }

private:
    void bfs(vector<vector<int>>& image, int sr, int sc, int newColor,int oldColor)
    {
```

```
        int r = image.size();
        int c = image[0].size();
        int dirx[] = {-1,0,1,0};
        int diry[] = {0,-1,0,1};

        queue<pair<int,int> >que;
        pair<int,int>u(sr,sc),v;
        que.push(u);
        while(!que.empty())
        {
            u = que.front();que.pop();
            sr = u.first;
            sc = u.second;
            image[sr][sc]=newColor;
            for(int i = 0; i < 4; ++i)
            {
                int x = sr+dirx[i];
                int y = sc+diry[i];
                if(x>=r||x<0||y>=c||y<0||image[x][y]!=oldColor)
                    continue;

                v.first = x,v.second = y;
                que.push(v);
            }
        }
    }
};
```

5.2.4 p111.二叉树的最小深度

递归

从树根开始往下bfs的过程中碰到的第一个叶子节点的深度即为最小深度

```
class Solution {
public:
    int minDepth(TreeNode* root) {
        if (!root)  return 0;
        if (!root->left)  return minDepth(root->right) + 1;
        if (!root->right)  return minDepth(root->left) + 1;
        return min(minDepth(root->left), minDepth(root->right)) + 1;
    }
};
```

5.2.5 p690. 员工的重要性

给定一棵树，每一个结点有一个权值，同时输入一个结点id，要求输出这个以这个结点为根的子数的所有结点权值的和。 做法：实际上构造出这棵树之后，可以用BFS/DFS遍历这棵树，假设是用DFS做的，那么就是在第一次访问到id的时候，开始记录权值和，接下去遍历到的结点权值就加入权值和，直到递归完成，离开id之后不再统计。

```
class Solution {
public:
    int getImportance(vector<Employee*> employees,int id)
    {
        Employee* rt;
        for (auto i:employees)
        {
            if (i->id==id)
            {
                rt=i; break;
            }
        }
        int sum=rt->importance;
        for (int i:rt->subordinates) sum+=getImportance(employees,i);
        return sum;
    }
};
```

5.2.6 p841. 钥匙和房间

解法一:dfs暴力搜索

```
class Solution {
public:
    bool canVisitAllRooms(vector<vector<int>>& rooms) {
        int sz = rooms.size();
        bool enable_rooms[sz] = {false};
        dfs(0,rooms,enable_rooms);
        return (Sum(enable_rooms,sz) == sz);
    }
private:
    int Sum(bool *enable_rooms,int sz)
    {

        int res = 0;
```

```
        for(int i = 0; i < sz; ++i)
            res += (enable_rooms[i]==true);
        return res;
    }
    void dfs(int u,vector<vector<int>>& rooms,bool *enable_rooms)
    {
        if(enable_rooms[u]==true)
            return ;
        enable_rooms[u] = true;
        for(int i = 0; i < rooms[u].size(); ++i)
        {
            int v = rooms[u][i];
            dfs(v,rooms,enable_rooms);
        }
    }
};
```

解法二:bfs

```
class Solution {
public:
    bool canVisitAllRooms(vector<vector<int>>& rooms) {
        int sz = rooms.size();
        bool enable_rooms[sz] = {false};
        queue<int>que;que.push(0);
        while(!que.empty())
        {
            int u = que.front();que.pop();
            enable_rooms[u] = true;
            for(int i = 0; i < rooms[u].size(); ++i)
            {
                int v = rooms[u][i];
                if(enable_rooms[v] == true)
                    continue;
                que.push(v);
            }
        }

        return (Sum(enable_rooms,sz) == sz);
    }
private:
    int Sum(bool *enable_rooms,int sz)
    {
        int res = 0;
        for(int i = 0; i < sz; ++i)
            res += (enable_rooms[i]==true);
        return res;
    }
};
```

5.2.7 p113. 路径总和 II

解法

dfs,从上往下记录路径权值和,,然后从下往上传递路径。

从根节点开始往下记录路径,如果到叶子时路径总和等于给定目标和.则把该路径传递给父节点,父节点在所有子节点传递的路径开头加上自身权值,继续向上传递。写起来有点复杂。

```
class Solution {
public:
    vector<vector<int>> pathSum(TreeNode* root, int sum) {
        vector<vector<int>>  ans;ans.clear();
        if(root!=NULL) ans = dfs(root,0,sum);
        return ans;
    }
private:
    bool is_leaf(TreeNode* node)
    {
        return (node->left==NULL) && (node->right==NULL);
    }
    void get_way(TreeNode* node,int &val,vector<vector<int>> &res,int nowsum,int sum)
    {
        vector<vector<int>> t = dfs(node,nowsum,sum);
        for(int i = 0; i < t.size(); ++i)
        {
            vector<int> &way = t[i];
            way.insert(way.begin(),val);
            res.push_back(way);
        }
    }
    vector<vector<int>> dfs(TreeNode* node, int nowsum,int sum)
    {
        nowsum += node->val;
        vector<vector<int>> res;res.clear();
        if(is_leaf(node))
        {
            if(nowsum==sum)
            {
                vector<int>way;way.clear();
                way.push_back(node->val);
                res.push_back(way);
            }

            return res;
        }

        if(node->left!=NULL)
            get_way(node->left,node->val,res,nowsum,sum);
        if(node->right!=NULL)
            get_way(node->right,node->val,res,nowsum,sum);
        return res;

    }
```

```
};
```

5.2.8 p130. 被围绕的区域

bfs

- 被围绕的区间不会存在于边界上
- 任何边界上的 O 都不会被填充为 X
- 任何不在边界上，或不与边界上的 O 相连的 O 最终都会被填充为 X

```
class Solution {
public:
    void solve(vector<vector<char>>& board) {
        if (board.size() == 0 || board[0].size() == 0)  return;
        int n = board.size(), m = board[0].size();
        vector<pair<int, int>> dir({{0, 1}, {0, -1}, {1, 0}, {-1, 0}});
        queue<pair<int, int>> Q;

        // 把所有边界上的'O'放入队列
        for (int i = 0; i < n; i++) {
            if (i == 0 || i == n - 1) {
                for (int j = 0; j < m; j++)
                    if (board[i][j] == 'O'){
                        Q.push({i, j});
                        board[i][j] = 'C';
                    }
            }
            else {
                if (board[i][0] == 'O') {
                    Q.push({i, 0});
                    board[i][0] = 'C';
                }
                if (board[i][m - 1] == 'O') {
                    Q.push({i, m - 1});
                    board[i][m - 1] = 'C';
                }
            }
        }

        while (!Q.empty()) {
            auto tmp = Q.front();
            Q.pop();
            for (int i = 0; i < 4; i++) {
                int x = tmp.first + dir[i].first;
                int y = tmp.second + dir[i].second;
                if (x < 0 || x == n || y < 0 || y == m)  continue;

                if (board[x][y] == 'O'){
```

```
                    Q.push({x, y});
                    board[x][y] = 'C';
                }
            }
        }
        for (int i = 0; i < n; i++)
            for (int j = 0; j < m; j++)
                board[i][j] = (board[i][j] == 'C' ? 'O' : 'X');
    }
};
```

5.2.9 p417. 太平洋大西洋水流问题

bfs

- 一种比较自然的思路是依次对每个点进行 bfs 看其是否既能到达太平洋又能到达大西洋，这种思路会导致做很多重复的操作以致超时
- 转换思路，从目标出发去寻找源，可以从与太平洋和大西洋直接相邻的点（也就是矩阵边缘上的点）出发进行 bfs 去遍历矩阵中所有的点，则遍历到的点肯定是能够到达某个海洋的点

```
class Solution {
private:
    void bfs(vector<vector<int>>& matrix, queue<pair<int, int>>& Q, vector<vector<int>>& visit)
{
        vector<pair<int, int>> dir({{0, 1}, {0, -1}, {1, 0}, {-1, 0}});
        int n = matrix.size(), m = matrix[0].size();
        while (!Q.empty()) {
            auto tmp = Q.front();
            Q.pop();
            for (int i = 0; i < 4; i++) {
                int x = tmp.first + dir[i].first;
                int y = tmp.second + dir[i].second;
                if (x < 0 || x == n || y < 0 || y == m || visit[x][y])  continue;
                if (matrix[x][y] >= matrix[tmp.first][tmp.second]) {
                    visit[x][y] = 1;
                    Q.push({x, y});
                }
            }
        }
    }
public:
    vector<pair<int, int>> pacificAtlantic(vector<vector<int>>& matrix) {
        vector<pair<int, int>> ans;
        if (matrix.size() == 0 || matrix[0].size() == 0)  return ans;
        int n = matrix.size(), m = matrix[0].size();


        // Pacific
```

```
        queue<pair<int, int>> pQ;
        vector<vector<int>> pVisit(n, vector<int>(m, 0));
        for (int j = 0; j < m; j++) {
            pVisit[0][j] = 1;
            pQ.push({0, j});
        }
        for (int i = 1; i < n; i++) {
            pVisit[i][0] = 1;
            pQ.push({i, 0});
        }
        bfs(matrix, pQ, pVisit);

        // Atlantic
        queue<pair<int, int>> aQ;
        vector<vector<int>> aVisit(n, vector<int>(m, 0));
        for (int j = 0; j < m; j++) {
            aVisit[n - 1][j] = 1;
            aQ.push({n - 1, j});
        }
        for (int i = 0; i < n - 1; i++) {
            aVisit[i][m - 1] = 1;
            aQ.push({i, m - 1});
        }
        bfs(matrix, aQ, aVisit);

        for (int i = 0; i < n; i++)
            for (int j = 0; j < m; j++)
                if (pVisit[i][j] && aVisit[i][j])
                    ans.push_back({i, j});
        return ans;
    }
};
```

5.2.10 p542. 01 矩阵

给定一个01矩阵，找出每个元素到最近的0的距离。 做法：BFS，floodfill的思想，一开始把所有的0的位置加入队列中，然后进行BFS

```
class Solution {
public:
    vector<vector<int>> updateMatrix(vector<vector<int>>& matrix)
    {
        typedef pair<int, int> pii;
        vector<pair<int,int>> dir{{1,0},{-1,0},{0,1},{0,-1}};
        int n,m;
        n=matrix.size(); m=matrix[0].size();

        vector<vector<int>> ans=matrix;
```

```
        queue<pii> q;
        for (int i=0;i<n;i++)
        {
            for (int j=0;j<m;j++)
            {
                if (matrix[i][j]==0)
                {
                    ans[i][j]=0;
                    q.push(pii(i,j));
                }
                else ans[i][j]=-1;
            }
        }
        while (!q.empty())
        {
            pii now=q.front();
            q.pop();
            int x=now.first;
            int y=now.second;
            for(auto&v:dir)
            {
                int tox=x+v.first;
                int toy=y+v.second;
                if (tox>=0&&tox<n&&toy>=0&&toy<m&&ans[tox][toy]==-1)
                {
                    ans[tox][toy]=ans[x][y]+1;
                    q.push(pii(tox,toy));
                }
            }
        }
        return ans;
    }
};
```

5.2.11 p473. 火柴拼正方形

给定N（最多15）个火柴，已知所有火柴长度，问是否可以用所有的火柴拼成一个正方形。所有的火柴都需要用上，并且不可以折断火柴。 做法：DFS。由于我们知道了所有火柴的长度，从而知道了正方形的周长就是所有火柴的长度和，从而知道了正方形的边长。这里我们可以想到一个剪枝就是只需要判断是否可以拼出3条边，因为如果拼出了3条，那么剩下的那个边一定可以拼出来。还有一个优化是我们尽量先放长度大的边，因为长度大的边灵活性比较差，这样可以有效减少递归次数。

```
class Solution {
    bool dfs(const vector<int>& nums,vector<int>& len,int id,int side)
    {
        if (id==nums.size()) return len[0]==side&&len[1]==side&&len[2]==side;
        for (int i=0;i<4;i++)
```

```
        {
            if (len[i]+nums[id]<=side)
            {
                len[i]+=nums[id];
                if(dfs(nums,len,id+1,side)) return 1;
                len[i]-=nums[id];
            }
        }
        return 0;
    }
public:
    bool makesquare(vector<int>& nums)
    {
        vector<int> len(4,0);
        int total=accumulate(nums.begin(),nums.end(),0);
        if (total%4||nums.size()<4) return 0;
        sort(nums.begin(),nums.end(),greater<int>());
        return dfs(nums,len,0,total/4);
    }
};
```

5.2.12 p773. 滑动谜题

给定一个2X3的板，上面有5块可以滑动的方块，标有数字1~5，还有一个空缺0，然后我们每一次操作可以移动一块板（即数字0与上下左右的其中一个数字交换）。问把板移动成题目描述的状态最少次数。 做法：其实就是八数码的简化版本，可以用A*，但是对于求解最少次数的题目，实际上也可以想到用BFS做。状态最多是6!,可以直接用数组存下来所有的状态。

```
class Solution {
public:
    int slidingPuzzle(vector<vector<int>>& board) {
        string now="";
        string des="123450";
        for (int i=0;i<board.size();i++)
            for(int j=0;j<board[0].size();j++) now+=to_string(board[i][j]);
        if (now==des) return 0;
        queue<string> q;
        vector<pair<int,int>> dir{{1,0},{-1,0},{0,1},{0,-1}};
        set<string> s;
        s.insert(now);q.push(now);
        int step=0;
        while (!q.empty())
        {
            int sz=q.size();
            for (int i=0;i<sz;i++)
            {
                auto t=q.front();
```

```
                q.pop();
                int pos=t.find('0');
                int x=pos/3;
                int y=pos%3;
                for(auto&v:dir)
                {
                    int tox=x+v.first;
                    int toy=y+v.second;
                    if (tox>=0&&tox<2&&toy>=0&&toy<3)
                    {
                        string to=t;
                        swap(to[pos],to[tox*3+toy]);
                        if (to==des) return step+1;
                        if (s.find(to)==s.end())
                        {
                            s.insert(to);
                            q.push(to);
                        }
                    }
                }
            }
            step++;
        }
        return -1;
    }
};
```

End.