

## Graph with BFS and DFS

2019年10月25日 下午 2:28

### Common Methods:

DFS, BFS, Union Find, Topologic Sorting

核心是想办法把一个问题看成graph问题

★ 图的DFS遍历，用hashmap存每个节点的neighbors

```
class Solution {
public:
    Node* cloneGraph(Node* node) {
        unordered_map<Node*, Node*> m;
        return helper(node, m);
    }
    Node* helper(Node* node, unordered_map<Node*, Node*>& m) {
        if (!node) return NULL;
        if (m.count(node)) return m[node];
        Node* clone = new Node(node->val);
        m[node] = clone;
        for (auto& neighbor: node->neighbors)
        {
            clone->neighbors.push_back(helper(neighbor, m));
        }
        return clone;
    }
};
```

★ 图BFS遍历，用hashmap存每个节点的neighbors，queue去遍历

```
class Solution {
public:
    Node* cloneGraph(Node* node) {
        if (!node) return NULL;
        unordered_map<Node*, Node*> m;
        queue<Node*> q{{node}};
        Node *clone = new Node(node->val);
        m[node] = clone;
        while (!q.empty()) {
            Node *t = q.front(); q.pop();
            for (Node *neighbor : t->neighbors) {
                if (!m.count(neighbor)) {
                    m[neighbor] = new Node(neighbor->val);
                    q.push(neighbor);
                }
                m[t->neighbors.push_back(m[neighbor]);
            }
        }
        return clone;
    }
};
```

Leetcode 200 islands or 695 以及 547

Friend Circles 一模一样的题目，联通分量，DFS

只不过双重for循环，一个for循环放在dfs里面

visited 一维数组表示friends circles

```
1 class Solution {
2 public:
3     int findCircleNum(vector<vector<int>>& M) {
4         if (M.empty()) return 0;
5         int n = M.size();
6         int ans = 0;
7         vector<int> visited(n, 0);
8         for (int i = 0; i < n; ++i) {
9             if (visited[i]) continue;
10            ++ans;
11            dfs(M, i, n, visited);
12        }
13        return ans;
14    }
15 private:
16    void dfs(vector<vector<int>>& M, int curr, int n, vector<int>& visited) {
17        // Visit all friends (neighbors)
18        if (visited[curr]) return;
19        visited[curr] = 1;
20        for (int i = 0; i < n; ++i) {
21            if (M[curr][i] && !visited[i]) dfs(M, i, n, visited);
22        }
23        return;
24    }
25};
```

18行不是多余的，当recursive调用时很有用

用visited表示这个人的所有朋友都访问过，所以不用再访问；

18 行不是多余的，当recursive调用的时候很有用

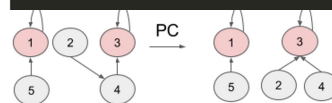
用visited,表示这个人的所有朋友都访问过，所以不用再访问；

★ 能返回int值就返回int值的比较快，不用传入值进去；

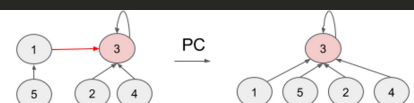
★ 上下进行比较 void dfs(max...) or int dfs()

```
1 class Solution {
2 public:
3     int maxAreaOfIsland(vector<vector<int>>& grid) {
4         if (grid.empty()) return 0;
5         int max = 0;
6         int m = grid.size();
7         int n = grid[0].size();
8         for(int i = 0; i < m; ++i)
9         {
10             for(int j = 0; j < n; ++j)
11             {
12                 int cur = 0;
13                 if(grid[i][j]) dfs(grid,cur,i,j,m,n);
14                 max = cur>max?cur:max;
15             }
16         }
17         return max;
18     }
19 private:
20     void dfs(vector<vector<int>>& grid,int& cur, int r, int c, int m, int n)
21     {
22         if(r < 0 || c < 0 || r >= m || c >= n) return;
23         if(!grid[r][c]) return;
24         grid[r][c] = 0;
25         cur++;
26         dfs(grid,cur,r,c-1,m,n);
27         dfs(grid,cur,r,c+1,m,n);
28         dfs(grid,cur,r-1,c,m,n);
29         dfs(grid,cur,r+1,c,m,n);
30     }
31 }
```

```
1 class Solution {
2 public:
3     int maxAreaOfIsland(vector<vector<int>>& grid) {
4         int h = grid.size();
5         if (h == 0) return 0;
6         int w = grid[0].size();
7
8         int ans = 0;
9         for (int i = 0; i < h; ++i)
10             for (int j = 0; j < w; ++j)
11                 ans = max(ans, area(grid, j, i, w, h));
12         return ans;
13     }
14 private:
15     int area(vector<vector<int>>& grid, int x, int y, int w, int h) {
16         if (x < 0 || y < 0 || x >= w || y >= h || grid[y][x] == 0) return 0;
17
18         grid[y][x] = 0;
19
20         return area(grid, x - 1, y, w, h)
21             + area(grid, x + 1, y, w, h)
22             + area(grid, x, y - 1, w, h)
23             + area(grid, x, y + 1, w, h)
24             + 1;
25     }
26 };
```



Union(2, 4), Find(2) = 3



Union(5, 2) = Union(Find(5), Find(2))

★ 遇到图想着用 adjacency lists 来表示

```
map<int, vector<int>> graph --> graph.at(边
1);
```

Union Find!! 查找cycles

<https://zxi.mytechroad.com/blog/data-structure/sp1-union-find-set/>

Ref: <https://www.cs.princeton.edu/~rs/AlgsDS07/01UnionFind.pdf>

PC = path compression

```
class UnionFindSet {
public:
    UnionFindSet(int n) {
        ranks_ = vector<int>(n + 1, 0);
        parents_ = vector<int>(n + 1, 0);

        for (int i = 0; i < parents_.size(); ++i)
            parents_[i] = i;
    }

    // Merge sets that contains u and v.
    // Return true if merged, false if u and v are already in one set.
    bool Union(int u, int v) {
```

## Leetcode 684 & 685 redundant connection 1 and 2

Leetcode

Leetcode 990 Statifiability of equality equations

### more concise Union Find

```
class Solution {
public:
    bool equationsPossible(vector<string>& equations) {
        iota(begin(parents_), end(parents_), 0);
        for (const auto& eq : equations)
            if (eq[1] == '=')
                parents_[find(eq[0])] = find(eq[3]);
        for (const auto& eq : equations)
            if (eq[1] == '!' && find(eq[0]) == find(eq[3]))
                return false;
        return true;
    }
private:
    array<int, 128> parents_;
    int find(int x) {
        if (x != parents_[x])
            parents_[x] = find(parents_[x]);
        return parents_[x];
    }
};
```

iota 用顺序递增的值存储在指定范围内

0, ....., 126,127,128

```
Class UnionFind{
    create parents_(size+1);
    for (int i = 0; i < size; ++i) // first, all members'parent are themselves
    {
        parents_[i] = i;
    }
    for (auto &edge: edges)//Union
    {
        if (find[edge[0]] == find[edge[1]]) return there is a cycle;
        parents_[find[edge[0]]] = find[edge[1]];
    }

    functions to check whether there is an edge or not?

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

```
int pu = Find(u);
int pv = Find(v);
if (pu == pv) return false;

// Meger low rank tree into high rank tree
if (ranks_[pv] > ranks_[pu])
    parents_[pu] = pv;
else if (ranks_[pu] > ranks_[pv])
    parents_[pv] = pu;
else {
    parents_[pv] = pu;
    ranks_[pv] += 1;
}

return true;
}

// Get the root of u.
int Find(int u) {
    // Compress the path during traversal
    if (u != parents_[u])
        parents_[u] = Find(parents_[u]);
    return parents_[u];
}
private:
    vector<int> parents_;
    vector<int> ranks_;
};
```

```
class Solution {
public:
    vector<int> findRedundantConnection(vector<vector<int>>& edges) {
        UnionFindSet s(edges.size());

        for(const auto& edge: edges)
            if (!s.Union(edge[0], edge[1]))
                return edge;

        return {};
    }
};
```

来自 <<https://zxi.mytechroad.com/blog/tree/leetcode-684-redundant-connection/>>

### Templates

```
class UnionFindSet {
public:
    UnionFindSet(int n) {
        ranks_ = vector<int>(n + 1, 0);
        parents_ = vector<int>(n + 1, 0);

        for (int i = 0; i < parents_.size(); ++i)
            parents_[i] = i;
    }

    // Merge sets that contains u and v.
    // Return true if merged, false if u and v are already in one set.
    bool Union(int u, int v) {
        int pu = Find(u);
        int pv = Find(v);
        if (pu == pv) return false;

        // Meger low rank tree into high rank tree
        if (ranks_[pv] < ranks_[pu])
            parents_[pv] = pu;
        else if (ranks_[pu] < ranks_[pv])
            parents_[pu] = pv;
```

```

else {
    parents_[pv] = pu;
    ranks_[pu] += 1;
}

return true;
}

// Get the root of u.
int Find(int u) {
    // Compress the path during traversal
    if (u != parents_[u])
        parents_[u] = Find(parents_[u]);
    return parents_[u];
}
private:
    vector<int> parents_;
    vector<int> ranks_;
};

```

来自 <<https://zxi.mytechroad.com/blog/data-structure/sp1-union-find-set/>>

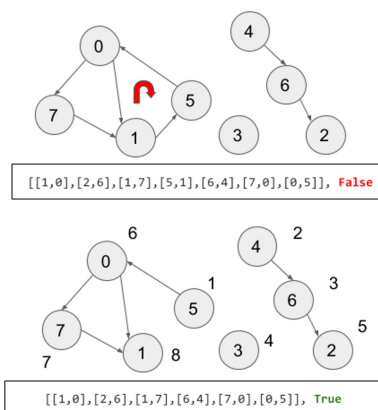
### Leetcode course schedule 207

<https://zxi.mytechroad.com/blog/graph/leetcode-207-course-schedule/>

there are **cycles**, there is impossible

花花酱 LeetCode huahualetcode

207. Course Schedule

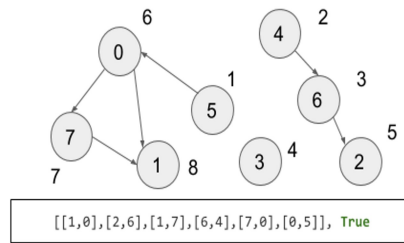


Solution:  
1. Topological sorting with DFS  $O(n)$

Topological sorting:

for each node:  
if not marked:  
if (dfs(node) == CYCLE) return CYCLE  
return OK

dfs(node):  
if node is marked as **visited**: return OK  
if node is marked as **visiting**: return CYCLE  
mark node as **visiting**  
for each new\_node in node.neighbors:  
if dfs(new\_node) == CYCLE: return CYCLE  
mark node as **visited**  
add node to the head of order\_list  
return OK

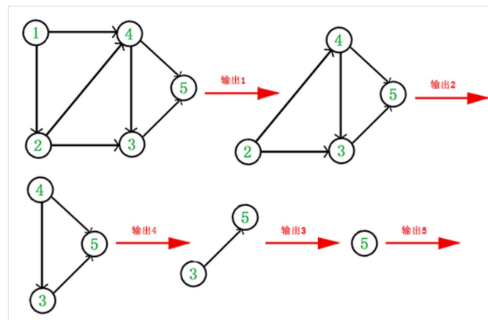


n	visiting	visited	ordered list
0	{0}	{}	{}
1	{0,1}	{}	{}
0	{0}	{1}	{1}
7	{0,7}	{1}	{1}
0	{0}	{1,7}	{7,1}
-	{}	{1,7,0}	{0,7,1}
2	{2}	{1,7,0}	{0,7,1}
-	{}	{1,7,0,2}	{2,0,7,1}
3	{3}	{1,7,0,2}	{2,0,7,1}
-	{}	{1,7,0,2,3}	{3,2,0,7,1}
4	{4}	{1,7,0,2,3}	{3,2,0,7,1}
6	{4,6}	{1,7,0,2,3}	{3,2,0,7,1}
4	{4}	{1,7,0,2,3,6}	{6,3,2,0,7,1}
-	{}	{1,7,0,2,3,6,4}	{4,6,3,2,0,7,1}
5	{5}	{1,7,0,2,3,6,4}	{4,6,3,2,0,7,1}
-	{}	{1,7,0,2,3,6,4,5}	{5,4,6,3,2,0,7,1}

### topological Sorting Templates 1 -> 0 一直寻找入度为0 的节点 indegree = 0

它是一个 DAG 图，那么如何写出它的拓扑排序呢？这里说一种比较常用的方法：

1. 从 DAG 图中选择一个 没有前驱（即入度为0）的顶点并输出。
2. 从图中删除该顶点和所有以它为起点的有向边。
3. 重复 1 和 2 直到当前的 DAG 图为空或当前图中不存在无前驱的顶点为止。后一种情况说明有向图中必然存在环。



### ★ DFS topologic Sorting

这个题目是完成事件1需要先完成事件2，也就是走1 之前需要2，那么转化成图，用topological sorting

```
bool canFinish(int n, vector<pair<int, int>>& pre)
{
    vector<vector<int>> adj(n, vector<int>());
    vector<int> degree(n, 0);
    for (auto &p: pre) {
        adj[p.second].push_back(p.first);
        degree[p.first]++;
    }
    queue<int> q;
    for (int i = 0; i < n; i++)
        if (degree[i] == 0) q.push(i);
    while (!q.empty()) {
        int curr = q.front(); q.pop(); n--;
        for (auto next: adj[curr])
            if (--degree[next] == 0) q.push(next);
    }
    return n == 0;
}
```

★ 用双重map可以达到二维数组访问的效果，只不过index可以是字母！！！！

```
unordered_map<string, unordered_map<string, double>> g;
```