并查集

ZeitHaum

2023 年 4 月 1 日

目录

1 概述

并查集是一种存储非重叠和不相交元素集合的数据结构。其支持查找和合并两种基本操作。

2 如何查找?

对于每个并查集，选择一个元素作为其**代表元素**。选择代表元素的方式有很多，一种基本方式是选择最大并查集中或者最小的元素。另外规定一个并查集的代表元素为其本身，且此元素必须属于并查集。

查找两个元素是否属于一个并查集只需检查其代表元素是否一致即可。

3 如何合并?

先找到两个并查集的代表元素，然后修改其中一个元素的代表元素为另外一个即可。

4 性能优化

4.1 按尺寸合并

合并时将大小小的并查集合并到尺寸大的并查集上。

4.2 按秩合并

定义每个并查集的秩为其树状结构的高度，合并时将低秩的并查集合并到高秩并查集是更优的选择。

4.3 路径压缩

每次查询时将查询节点直接和代表元素相连，将降低重复查询相同元素的复杂度。

4.4 复杂度

1. 若没有任何性能优化，单次查询和合并复杂度为 $O(n)$.

2. 若只带按秩合并或按尺寸合并优化，对于 n 个元素和 m 次操作，其复杂度为 $O(m\log(n))$.

3. 若只带路径压缩优化，对于 n 次合并和 f 次查询操作，其复杂度为 $O(n+f\cdot\log_{2+f/n}(n))$.

4. 若既有按秩合并和路径压缩，对于 m 次操作和 n 次合并，其复杂度为 $O(m\alpha(n))$. 其中 $\alpha(n)$ 为反 Ackerman 函数，对于物理世界中任意可以写下的数，$\alpha(n)$ 不会超过 4。

4.5 应用

1. 最小生成树算法 (Kruskal).

2. 图论判环.

3. $\cdots$.

4.6 实现

可用负数标记每个并查集的代表元素，为了高效利用空间，可令并查集代表元素的标记为并查集大小或者秩的相反数。由此根据按尺寸和按秩的不同有两种实现:

```
#include<bits/stdc++.h>
using namespace std;

struct DSU_size{
    vector<int> e;
    DSU_size(int N){
        e = vector<int>(N,-1);
    }
    int find(int x){return e[x]<0?x:e[x] = find(e[x]);}
    int size(int x){return -e[find(x)];}
    bool check_same(int x,int y){return find(x)==find(y);}
    bool unite(int x,int y){
        x = find(x),y = find(y);
        if(x==y) return false;
        if(e[x]<e[y]) swap(x,y);//小的往大的合入。
        e[y] += e[x];//注意更新顺序。
```

```
        e[x] = y;
        return true;
    }
};

struct DSU_rank{
    vector<int>e;
    DSU_rank(int N){e = vector<int>(N,0);}
    int find(int x){return e[x]<=0?x:e[x] = find(e[x]);}
    int rank(int x){return -e[find(x)];}
    bool check_same(int x,int y){return find(x)==find(y);}
    bool unite(int x,int y){
        x = find(x),y = find(y);
        if(x==y) return false;
        if(e[x]<e[y]) swap(x,y);
        e[y] += e[x]==e[y]?0:-1;
        e[x] = y;
        return true;
    }
};
```

5 相关例题

5.1 例题 1

PTA 天体赛-练习集:L2-024 部落

5.1.1 Solution

该题重点是 DSU 个数的求取，观察到每成功合并一次总的 DSU 个数将会减一，可记录总的减少值最后加上数组长度即是答案。

5.1.2 Code

按尺寸合并版本:

```
#include<bits/stdc++.h>
using namespace std;
```

```
3
4  struct DSU_size{
5      vector<int> e;
6      DSU_size(int N){
7          e = vector<int>(N,-1);
8      }
9      int find(int x){return e[x]<0?x:e[x] = find(e[x]);}
10     int size(int x){return -e[find(x)];}
11     bool check_same(int x,int y){return find(x)==find(y);}
12     bool unite(int x,int y){
13         x = find(x),y = find(y);
14         if(x==y) return false;
15         if(e[x]<e[y]) swap(x,y);//小的往大的合入。
16         e[y] += e[x];
17         e[x] = y;
18         return true;
19     }
20 };
21
22
23 //带按尺寸合并和路径压缩
24 void solve(){
25     int n;
26     int MAX_SIZE = 10001;
27     DSU_size dsu = DSU_size(MAX_SIZE);
28     int actual_size = 0;
29     cin>>n;
30     int merge_cnt = 0;
31     for(int i = 0;i<n;i++){
32         int K;
33         cin>>K;
34         int rep;
35         cin>>rep;
36         actual_size = max(rep,actual_size);
37         int temp;
38         for(int j = 1;j<K;j++){
39             cin>>temp;
40             actual_size = max(temp,actual_size);
41             if(dsu.unite(temp,rep)) merge_cnt--;
```

```
        }
    }
    int Q;
    cin>>Q;
    int x,y;
    cout<<actual_size<<" "<<merge_cnt+actual_size<<endl;
    for(int i = 0;i<Q;i++){
        cin>>x>>y;
        if(dsu.check_same(x,y)) cout<<"Y"<<endl;
        else cout<<"N"<<endl;
    }
}


int main(){
    solve();
}
```

按秩合并版本:

```
#include<bits/stdc++.h>
using namespace std;

struct DSU_rank{
    vector<int>e;
    DSU_rank(int N){e = vector<int>(N,0);}
    int find(int x){return e[x]<=0?x:e[x] = find(e[x]);}
    int rank(int x){return -e[find(x)];}
    bool check_same(int x,int y){return find(x)==find(y);}
    bool unite(int x,int y){
        x = find(x),y = find(y);
        if(x==y) return false;
        if(e[x]<e[y]) swap(x,y);
        e[y] += e[x]==e[y]?0:-1;
        e[x] = y;
        return true;
    }
};


```

```
21 //带按秩合并和路径压缩
22 void solve(){
23     int n;
24     int MAX_SIZE = 10001;
25     DSU_rank dsu = DSU_rank(MAX_SIZE);
26     int actual_size = 0;
27     cin>>n;
28     int merge_cnt = 0;
29     for(int i = 0;i<n;i++){
30         int K;
31         cin>>K;
32         int rep;
33         cin>>rep;
34         actual_size = max(rep,actual_size);
35         int temp;
36         for(int j = 1;j<K;j++){
37             cin>>temp;
38             actual_size = max(temp,actual_size);
39             if(dsu.unite(temp,rep)) merge_cnt--;
40         }
41     }
42     int Q;
43     cin>>Q;
44     int x,y;
45     cout<<actual_size<<" "<<merge_cnt+actual_size<<endl;
46     for(int i = 0;i<Q;i++){
47         cin>>x>>y;
48         if(dsu.check_same(x,y)) cout<<"Y"<<endl;
49         else cout<<"N"<<endl;
50     }
51 }
52
53 int main(){
54     solve();
55 }
```

5.2 例题 2

力扣-岛屿数量

5.2.1 Solution

本题关键点在于并查集对象不是 int 类型而是一个 int 对。可使用 Std 标准库的 map 替换 vector, 注意需要重载的操作符对象以及拷贝构造函数的写法标准，诀窍在于参数中尽量考虑是否能添加 & 和 const 关键字。

本题需要考虑如何处理枚举合并顺序，保证不漏的情况下尽量减少重复。

5.2.2 Code

```
1  class Solution {
2  public:
3      struct point{
4          int x;
5          int y;
6          point(int x,int y):x(x),y(y) {}
7          point(): x(-1),y(-1){}
8          bool operator==(const point& p)const{
9              return this->x==p.x && this->y==p.y;
10         }
11         bool operator<(const point&p )const{
12             if(this->x == p.x) return this->y<p.y;
13             else return this->x<p.x;
14         }
15         point(const point& p): x(p.x),y(p.y){}
16         void swap(point& p){
17             int temp_x = p.x;
18             int temp_y = p.y;
19             p.x = this->x;
20             p.y = this->y;
21             this->x = temp_x;
22             this->y = temp_y;
23         }
24         friend ostream& operator<<(ostream& out,const point& p)
               {
```

```
25              return out<<p.x<<" "<<p.y<<" ";
26          }
27      };
28
29      struct DSU_size{
30          map<point,point>e;
31          DSU_size(vector<vector<char>>& grid){
32              for(int i = 0;i<grid.size();i++){
33                  for(int j = 0;j<grid[0].size();j++){
34                      if(grid[i][j]=='1'){
35                          point p{i,j};
36                          point* l = new point();
37                          e[p] = *l;
38                      }
39                  }
40              }
41          }
42          point find(point& p){return e[p].x<0?p:e[p] = find(e[p
                ]);}
43          bool check_same(point& p1,point& p2){return find(p1)==
                find(p2);}
44          bool size(point& p){return -find(p).x;}
45          int unite(int x1,int y1,int x2,int y2){
46              point p1{x1,y1};
47              point p2{x2,y2};
48              p1 = find(p1), p2 = find(p2);
49              if(p1==p2) return 0;
50              if(e[p1].x<e[p2].x) p1.swap(p2);
51              e[p2].x += e[p1].x;
52              e[p1] = p2;
53              return -1;
54          }
55
56      };
57
58      int numIslands(vector<vector<char>>& grid) {
59          DSU_size dsu(grid);
60          int n = grid.size();
61          int m = grid[0].size();
```

```
62          int land_cnt = 0;
63          for(int i = 0;i<n;i++){
64              for(int j =0;j<m;j++){
65                  land_cnt+= grid[i][j]=='1'?1:0;
66              }
67          }
68          if(grid.size()==0) return 0;
69          int merge_cnt = 0;
70          for(int i =0;i<n;i++){
71              for(int j = 0;j<m;j++){
72                  if(grid[i][j]!='1') continue;
73                  if(i>0 && grid[i-1][j]=='1') merge_cnt+=dsu.
                        unite(i,j,i-1,j);
74                  if(j>0 && grid[i][j-1]=='1') merge_cnt+=dsu.
                        unite(i,j,i,j-1);
75              }
76          }
77          // for(auto iter = dsu.e.begin();iter!=dsu.e.end();iter
                ++){
78          //      cout<<iter->first<<iter->second<<endl;
79          // }
80          return land_cnt+merge_cnt;
81      }
82  };
```

6 参考资料

[1]. USACO-Guide

[2]. geeksforgeeks

[3]. wikipedia

[4]. 算法导论 (原书第三版)