Alogrithm in C

基础知识

第1章 引言

对于连通性问题(Dynamic Connectivity)，书中首先提出的是快速查找算法(quick-find algorithm)。大意是把连通的id值赋值为同一个，用id值的等价来表示它的连通。

过程大致如下：

1. 初始化数组
2. 获取输入
    - 如果输入的两个值的id相等，则跳出（已连通）
    - 否则，就把所有id值为id[p]的元素变为id[q]（标志连通属性）

则有代码：

快速查找算法

```c
#include <stdio.h>
#define N 10000
int main(){
  int i, p, q, t, id[N];
  // 初始化数组
  for(i = 0; i < N; i++)
    id[i] = i;
  while(scanf(" %d %d\n", &p, &q) == 2){
    // 如果输入的两个值得id相等，则跳出（已连通）
    if(id[p] == id[q])
      continue;
    // 检查所有id值为id[p]的元素，并将其值置为id[q]
    for(t = id[p], i = 0; i < N; i++)
      if(id[i] == t)
        id[i] = id[q];
    printf(" %d %d\n", p, q);
  }
}
```

但也可按照根(root)查找，根一定是指向自身的，每一次输入，都是“枝”的生长、合并，当查找的时候，找的是位于“树根”（枝的尽头）处的值，只要“树根”处的值位于一个集合中，则它们在一个“树”上，那我们就可以说，它们是连通的，这就是快速合并算法。

那么此时的数组就比第一种更加抽象了，数组的id值其实是该节点的“上一个值”，即父节点，通过数组的不停迭代上溯查找，来找到“树根”。

则有代码：

快速合并算法

```c
#include <stdio.h>
#define N 10000
int main(){
  int i, j, p, q, t, id[N];
  // 初始化数组,每一个元素都是自己的“根”
  for(i = 0; i < N; i++)
    id[i] = i;
  while(scanf(" %d %d\n", &p, &q) == 2){
    // 两个for循环都是为了找到各自的“根”。如果i和id值相等，那么
    // 自然有i == id[i]，即找到了“根”，退出循环；如果i和id值不
    // 等，则此值一定会通过“上溯”的方法找到“根”的位置，即，把其
id
    // 值作为新的查找值，直至找出其“根”。
    for(i = p; i != id[i]; i = id[i])
      ;
    for(j = q; j != id[j]; j = id[j])
      ;
    // 如果“根”相同，则说明它们在同一个集合中；否则，就将其归入
    // 另外一个集合中。
    if(i == j)
      continue;
    id[i] = j;
    printf(" %d %d\n", p, q);
  }
}
```

但是考察一下第二个算法，当进行树的合并(id[i]=j)的时候，如果把较大的树合并到较小的树上，就会造成查找的耗时。而我们可以通过增加一个判断，使较小的树总是连接到较大的树上，从而达到优化算法的目的。

这样的话，就需要一个size数组来记录每一个树的大小，在连接时进行比较，由此，将较小的树连接到较大的树上。

则有代码：

加权快速合并算法

```c
#include <stdio.h>
#define N 10000
int main(){
  int i, j, p, q, t, id[N], sz[N];
  for(i = 0; i < N; i++){
    id[i] = i;
    sz[i] = 1; // 每一个独立单元都是一个size为1的树
  }
  while(scanf(" %d %d\n", &p, &q) == 2){
    for(i = p; i != id[i]; i = id[i])
      ;
    for(j = q; j != id[j]; j = id[j])
      ;
    if(i == j)
      continue;
    // 加权合并
    if(sz[i] < sz[j]){
      id[i] = j;
      sz[j] += sz[i];
    }else{
      id[j] = i;
      sz[i] += sz[j];
    }
    printf(" %d %d\n", p, q);
  }
}
```

但对于这个算法而言，我们可以把一个树进行路径压缩，使树“平扁化”。

等分路径压缩

```c
#include <stdio.h>
#define N 10000
int main(){
  int i, j, p, q, t, id[N], sz[N];
  for(i = 0; i < N; i++){
    id[i] = i;
    sz[i] = 1; // 每一个独立单元都是一个size为1的树
  }
  while(scanf(" %d %d\n", &p, &q) == 2){
    for(i = p; i != id[i]; i = id[i])
      id[i] = id[id[i]];
    for(j = q; j != id[j]; j = id[j])
      id[j] = id[id[j]];
    if(i == j)
      continue;
    // 加权合并
    if(sz[i] < sz[j]){
      id[i] = j;
      sz[j] += sz[i];
    }else{
      id[j] = i;
      sz[i] += sz[j];
    }
    printf(" %d %d\n", p, q);
  }
}
```