华东师范大学数据科学与工程学院上机实践报告

课程名称：算法设计与分析　　**年级：**21 级　　**上机实践成绩：**

指导教师：金澈清　　**姓名：**杨茜雅

上机实践名称：并查集　　**学号：**10215501435　　**上机实践日期：**5.19

上机实践编号：No.11

一、目的

1．熟悉算法设计的基本思想
2．熟悉并查集

二、内容与设计思想

1. 解决 OJ 上的练习题，习题描述如下所述。
2. 简单描述优化复杂度的方案，比如按秩合并，路径压缩。
3. 如果要统计每个联通的城镇的数量，需要如何修改算法呢？

*注 1：练习题只有正确性测试点，请至少列出不同实现的代码的时间（多次求平均）。
*注 2：可以自己设计测试方式和数据进行比较。

有 N 个城镇，现有城镇道路统计表，表中列出了每条道路直接连通的城镇。目标是使任何两个城镇间都可以连通（但不一定有直接的道路相连，只要互相间接通过道路可达即可）。问最少还需要建设多少条道路？

每个测试点都有多组输入，对于每组输入：

第一行输入两个正整数 N(1 <= N<= 20000)和 M(1<= M <= 20000)， N 表示城镇个数，M 表示道路个数。

接下来 M 行，每行两个正整数 a,b (1<= a,b <= N)，用两个城镇表示一条道路。

在所有输入的结尾，有一个 0 单独成行，表示输入文件的结束。

保证每个测试点的输入组数不超过 50 组。

三、使用环境

推荐使用 C/C++集成编译环境。

不建议随意改变本文档以上部分的结构，以下部分可以按需扩充。但不要缺少“实验过程”和“总结”。

四、实验过程

路径压缩

```
//路径压缩
#include<iostream>
using namespace std;
int set[100000];//存每个元素的父节点
void Ini_set(int n)
{
    for(int i=1;i<=n;i++)
    {
        set[i]=i;//初始化并查集，把每个元素的父节点设为自己
    }
}
int find_set(int x)//找每个元素的根节点，找到的标志是父节点是其本身
{
    if(set[x]==x)
    return x;
    else
    {
        set[x]=find_set(set[x]);
        return set[x];
    }//按路径合并在 find_set 处做修改，如果一个元素不是那个集合的根节点，则找到其根节点使之成为该元素的父节点即可

}
void merge_set(int a,int b)//合并功能：如果两个元素的根节点不一样
{
    if(find_set(a)!=find_set(b))
    {
        set[find_set(a)]=find_set(b);//找到两个集合的根节点，将其中一个节点的父节点设置成另一个元素
    }
}
```

```
int main()
{

    int num;

    while(cin>>num)
{
        if (num==0)
        break;
    for(int i=1;i<=num;i++)
    {
        Ini_set(num);
    }
    int peer_num;
    cin>>peer_num;
    for(int h=1;h<=peer_num;h++)
    {
        int x,y;
        cin>>x>>y;
        merge_set(x,y);//把题目里提到的元素都合并起来
    }
    int set_number=0;
    for(int i=1;i<=num;i++)
    {
        if(set[i]==i)
        set_number++;//如果一个元素的父节点是其本身，代表它是
一个集合，并 cout 一共有多少不相交的集合
    }
    cout<<set_number-1<<endl;//连接 n 个不相交的集合需要 n-1 条
线，放在题目里就是 n-1 条路
}

    return 0;
```

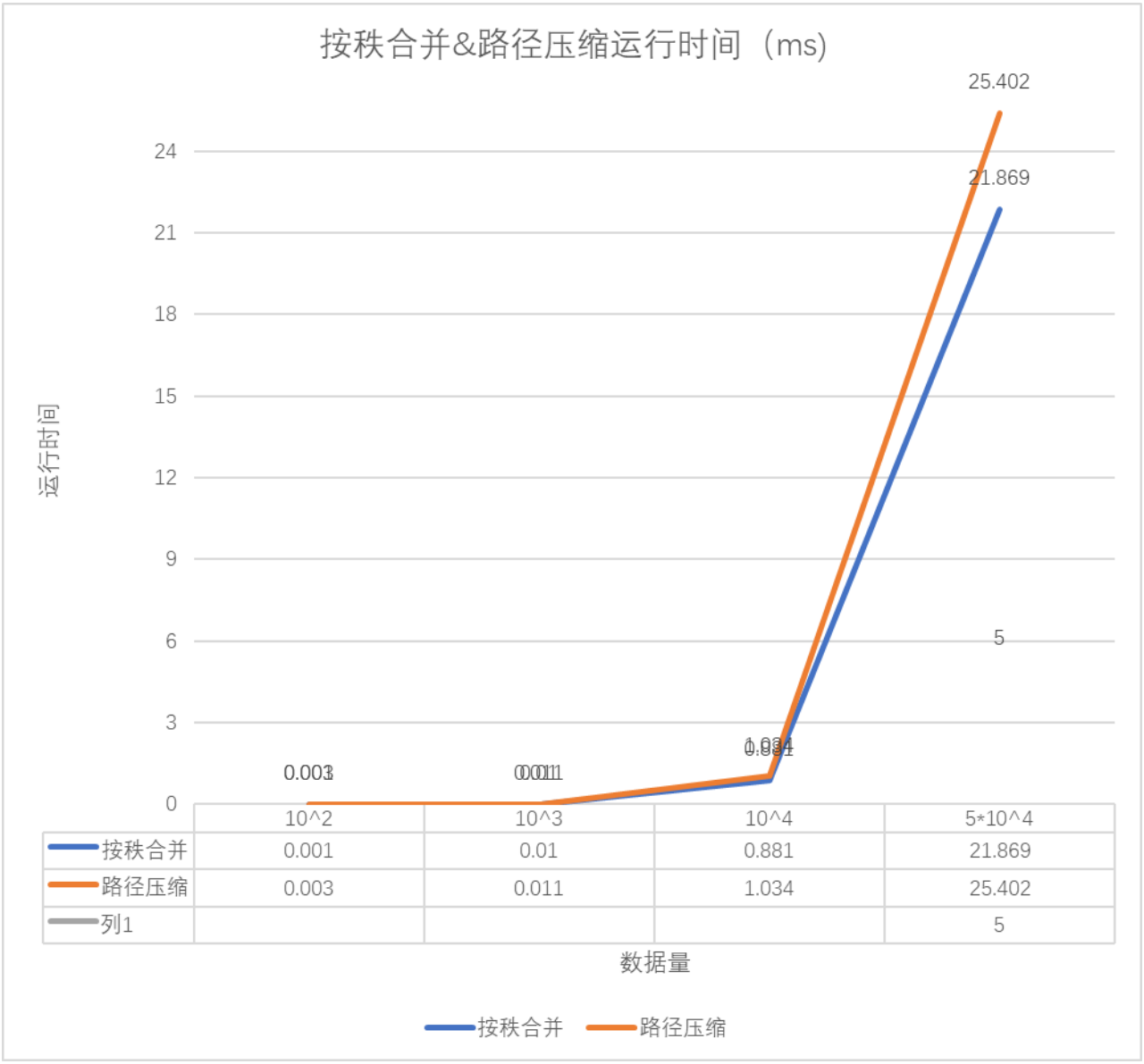
```
}
```

按秩合并

```
//按秩合并
#include<iostream>
using namespace std;
int set[100000];//存每个元素的父节点
int item_rank[100000];//每个元素的根节点对应的树的深度
void Ini_set(int n)
{
    for(int i=1;i<=n;i++)
    {
        set[i]=i;//初始化并查集，把每个元素的父节点设为自己
        item_rank[i]=1;//初始化把每个元素的深度都设为 1
    }
}
int find_set(int x)//找每个元素的根节点，找到的标志是父节点是其本身
{
   if(set[x]==x)
    return x;
    else
    return find_set(set[x]);//不是根节点的话就往上从其父节点再找一层
}
void merge_set(int a,int b)//合并功能：如果两个元素的根节点不一样
{
    int x=find_set(a),y=find_set(b);//先找到根节点
    if(item_rank[x]<=item_rank[y])
    {
        set[x]=y;//让 y 作为 x 的父节点
    }
    else if(item_rank[x]>item_rank[y])
    set[y]=x;
```

```
    if(item_rank[x]==item_rank[y]&&x!=y)
    item_rank[y]++;//思想是在合并时永远把简单的树往复杂的树上合并
}
int main()
{

    int num;

    while(cin>>num)
{
        if (num==0)
        break;
    for(int i=1;i<=num;i++)
    {
        Ini_set(num);
    }
    int peer_num;
    cin>>peer_num;
    for(int h=1;h<=peer_num;h++)
    {
        int x,y;
        cin>>x>>y;
        merge_set(x,y);//把题目里提到的元素都合并起来
    }
    int set_number=0;
    for(int i=1;i<=num;i++)
    {
        if(set[i]==i)
        set_number++;//如果一个元素的父节点是其本身，代表它是一个集合，并 cout 一共有多少不相交的集合
    }
    cout<<set_number-1<<endl;//连接 n 个不相交的集合需要 n-1 条线，放在题目里就是 n-1 条路
```

```
}

    return 0;

}
```



五、总结

最简单的并查集效率很低，所以有两种方法可以对其进行优化，即按秩合并和路径压缩

路径压缩：从具象上来看是把一个链状的并查集变成树状，这样查找一个元素的根节点的时候不用一层层从其父节点一次递归去找，因为在路径压缩方法里，我们只关心一个元素的根节点，并且我们希望一个元素到其根节点的距离尽量短，所以在查询的过程中，我们把沿途的每个结点的父节点都设为根节点。最后表现为一个集合的元素，它们的父节点就是这个元素的根节点。

按秩合并：按秩合并是一种树的形式，在普通的方法里，在写 merge 函数时，到底是把前者

作为后者的父节点还是后者作为前者的父节点是我们不关心的问题，但是在这里我们考虑每次都把简单的树并到复杂的树上，这样到根节点距离变长的节点个数会少一些。因此我们需要一个 rank[]数组用来记录每个节点的深度，最后合并完记得 rank++

随机测试代码：每次生成 item-num 个元素，暂且定有五组是联通的，至于谁和谁连通是 random 出来的，每次按秩合并和路径压缩采用同一组数据

```
#include <iostream>
#include<math.h>
#include <stdlib.h>
#include <time.h>
#include<fstream>
using namespace std;
 int main()
{
  srand((unsigned)time(NULL));
  ofstream outfile;
  int item_num;
  cin>>item_num;
  outfile.open("random_item.txt");
    for(int i=1;i<=5;i++)
    {
        int peer_num=1+rand()%item_num;
        outfile<<item_num<<" "<<peer_num<<endl;
        for(int i=1;i<=peer_num;i++)
        {
       int a=1+rand()%item_num;
       int b=1+rand()%item_num;
       outfile<<a<<" "<<b<<endl;
         }
    }


  outfile<<"0";
  outfile.close();
  return 0;
}
```