数据结构进阶

复习：几种基本的数据结构

- 栈：先进后出结构
- 队列：先进先出结构
- 链表：易维护不易查询

栈

- Stack
- 使用数组模拟：

  a[];

  top=0;

  入栈： a[top]=x;top++;

  出栈： top--;x=a[top];

  判断栈是否为空： top==0?

栈

- 更简易的实现方法：

STL 栈

- STL = Standard Template Library，标准模板库
- STL 栈须引用 <stack> 头文件 --- 》 C++ 内容

栈

- STL 栈基本操作

  stack<int>s;

  s.push(); 入栈

  s.pop(); 出栈

  s.top(); 查看栈顶元素

  s.size(); 查看栈的大小

  s.empty(); 判断栈是否为空

  自己建立一个源程序文件试验一下这 5 个操作，时间 5 分钟

队列

- queue
- 使用数组模拟：

  a[];

  l=0,r=0;

  入队列： a[r]=x;r++;

  出队列： x=a[l];l++;

  判断队列是否为空： l==r？

队列

- 更简易的实现方法：

STL 队列

- STL = Standard Template Library，标准模板库
- STL 队列须引用 <queue> 头文件 --- 》 C++ 内容

队列

- STL 队列基本操作

  queue<int>q;

  q.push(); 入队列

  q.pop(); 出队列

  q.front(); 查看队列前端元素

  q.end(); 查看对列尾端元素

  q.size(); 查看队列的大小

  q.empty(); 判断队列是否为空

  自己建立一个源程序文件试验一下这 6 个操作，时间 5 分钟

可变长数组

- 定义数组通常需要限定大小

例如 :int a[10]

- 虽然我们可以 resize，但是很不方便
- STL 中存在一个叫做 vector 的东西
- vector 能够像容器一样存放各种类型的对象，简单地说， vector 是一个能够存放任意类型的动态数组，能够增加和压缩数据。
- 须引用 <vector> 头文件

vector

- vector 基本操作

  vector<int>v;

  v.push_back(); 向 vector 后面添加元素

  v.clear(); 清空 vector

  v[x]; 访问第 x+1 个元素

  v.size(); 查看 vector 的大小

  vector 的用法很多，大家可以自己去查找资料学习

  自己建立一个源程序文件试验一下这 4 个操作，时间 5 分钟

STL

- STL 中还包含例如 map 、优先队列等容器，有兴趣的同学可以自己上网查询一下相关的信息，还有迭代器的用法。建议大家学好 stl ，对你以后做题会有很大帮助。
- 另外大家可以尝试着在容器内部使用结构体。

我们今天的主要内容

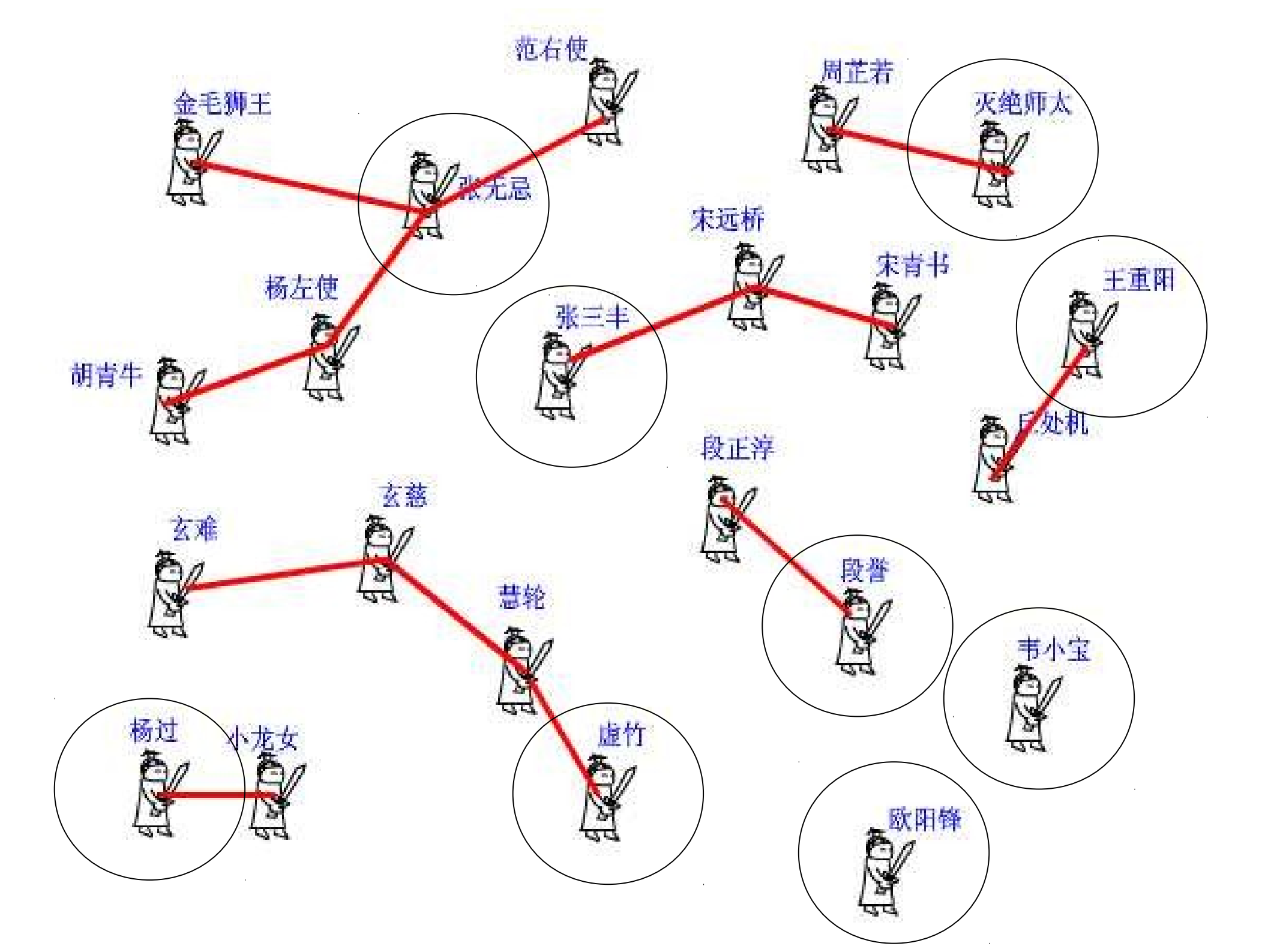
- 并查集
- 并查集是一种树型的数据结构，用于处理一些不相交集合（Disjoint Sets）的合并及查询问题。

这里借用一下别人的例子

- 话说江湖上散落着各式各样的大侠，有上千个之多。他们没有什么正当职业，整天背着剑在外面走来走去，碰到和自己不是一路人的，就免不了要打一架。但大侠们有一个优点就是讲义气，绝对不打自己的朋友。而且他们信奉“朋友的朋友就是我的朋友”，只要是能通过朋友关系串联起来的，不管拐了多少个弯，都认为是自己人。这样一来，江湖上就形成了一个一个的群落，通过两两之间的朋友关系串联起来。而不在同一个群落的人，无论如何都无法通过朋友关系连起来，于是就可以放心往死了打。但是两个原本互不相识的人，如何判断是否属于一个朋友圈呢？

- 我们可以在每个朋友圈内推举出一个比较有名望的人，作为该圈子的代表人物，这样，每个圈子就可以这样命名“齐达内朋友之队”“罗纳尔多朋友之队”……两人只要互相对一下自己的队长是不是同一个人，就可以确定敌友关系了。

- 但是还有问题啊，大侠们只知道自己直接的朋友是谁，很多人压根就不认识队长，要判断自己的队长是谁，只能漫无目的的通过朋友的朋友关系问下去：“你是不是队长？你是不是队长？”这样一来，队长面子上挂不住了，而且效率太低，还有可能陷入无限循环中。于是队长下令，重新组队。队内所有人实行分等级制度，形成树状结构，我队长就是根节点，下面分别是二级队员、三级队员。每个人只要记住自己的上级是谁就行了。遇到判断敌友的时候，只要一层层向上问，直到最高层，就可以在短时间内确定队长是谁了。由于我们关心的只是两个人之间是否连通，至于他们是如何连通的，以及每个圈子内部的结构是怎样的，甚至队长是谁，并不重要。所以我们可以放任队长随意重新组队，只要不搞错敌友关系就好了。于是，门派产生了。


范右使
金毛狮王
张无忌
周芷若
灭绝师太
宋远桥
宋青书
王重阳
杨左使
张三丰
胡青牛
段正淳
玄慈
玄难
段誉
韦小宝
慧轮
杨过
小龙女
虚竹
欧阳锋

用程序来描述这个过程

- 定义:
- Int father[i] 为第 i 个人的上级
- 初始每个人的上级为自己

谁是我的上级?

```
int find_father(int x)
{
    int r=x;
    while (father[r ]!=r)
            r=father[r ] ;
    return  r ;
}
```

如何建立门派关系呢?

```
void join(int x,int y)
{
    int fx=find_father(x),fy=find_father(y);
    if(fx!=fy)
    father[fx]=fy;
}
```

那么一共有多少门派呢?

- Int ans=0;
- for(int i=1;i<=n;i++)
- {

```
 if(father[i]==i)
      ans++;
}
```

考虑这么一个问题

- 某省调查城镇交通状况，得到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。省政府“畅通工程”的目标是使全省任何两个城镇间都可以实现交通（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?

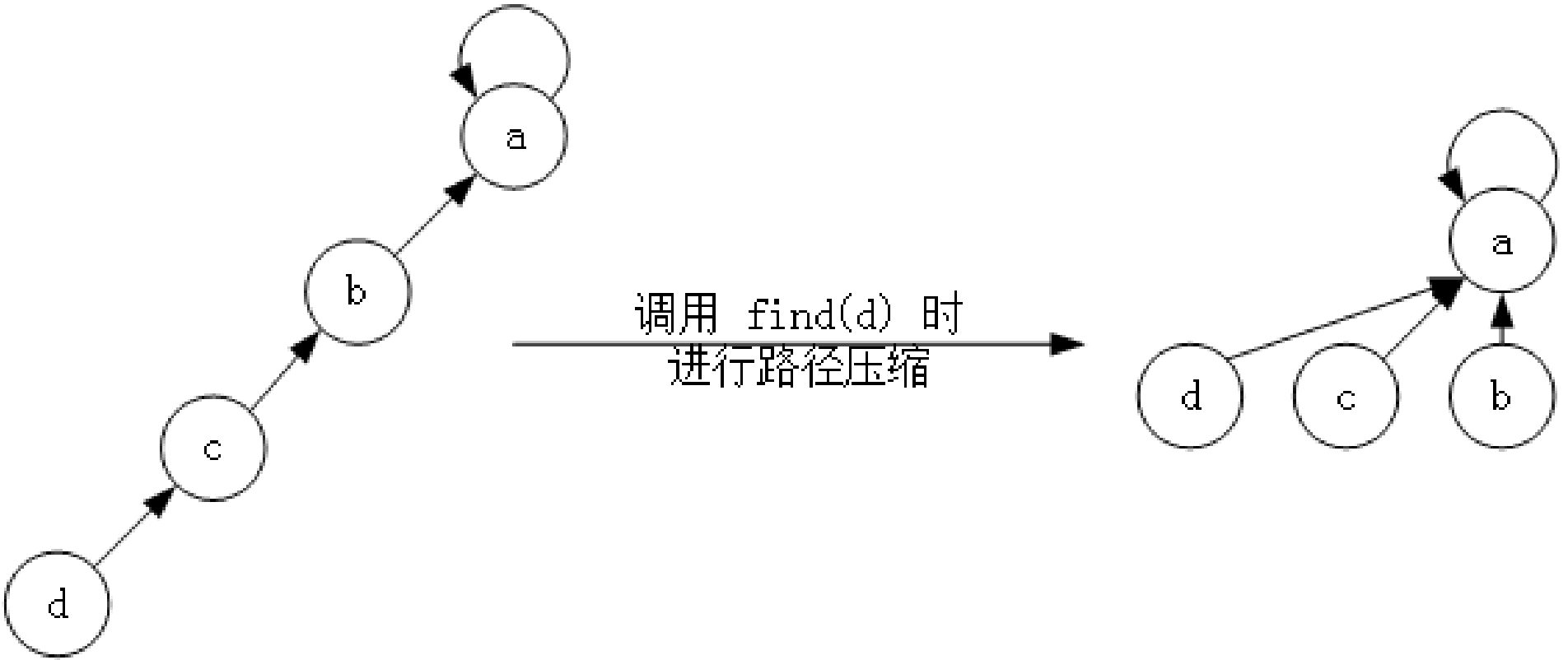
问题分析

- 把这个问题和上文的门派相类比，则能得出下列结论：
- 把城镇看作是每个人，道路关系看作门派关系，最后要建设的道路数目就是门派数目减一。

代码如下

自己先练习一下～～

路径压缩



想想代码应该怎么写?

```
int find_father(int x)
{
    int r=x;
    while (father[r ]!=r)
            r=father[r ] ;
    return  r ;
}
```

想想代码应该怎么写?

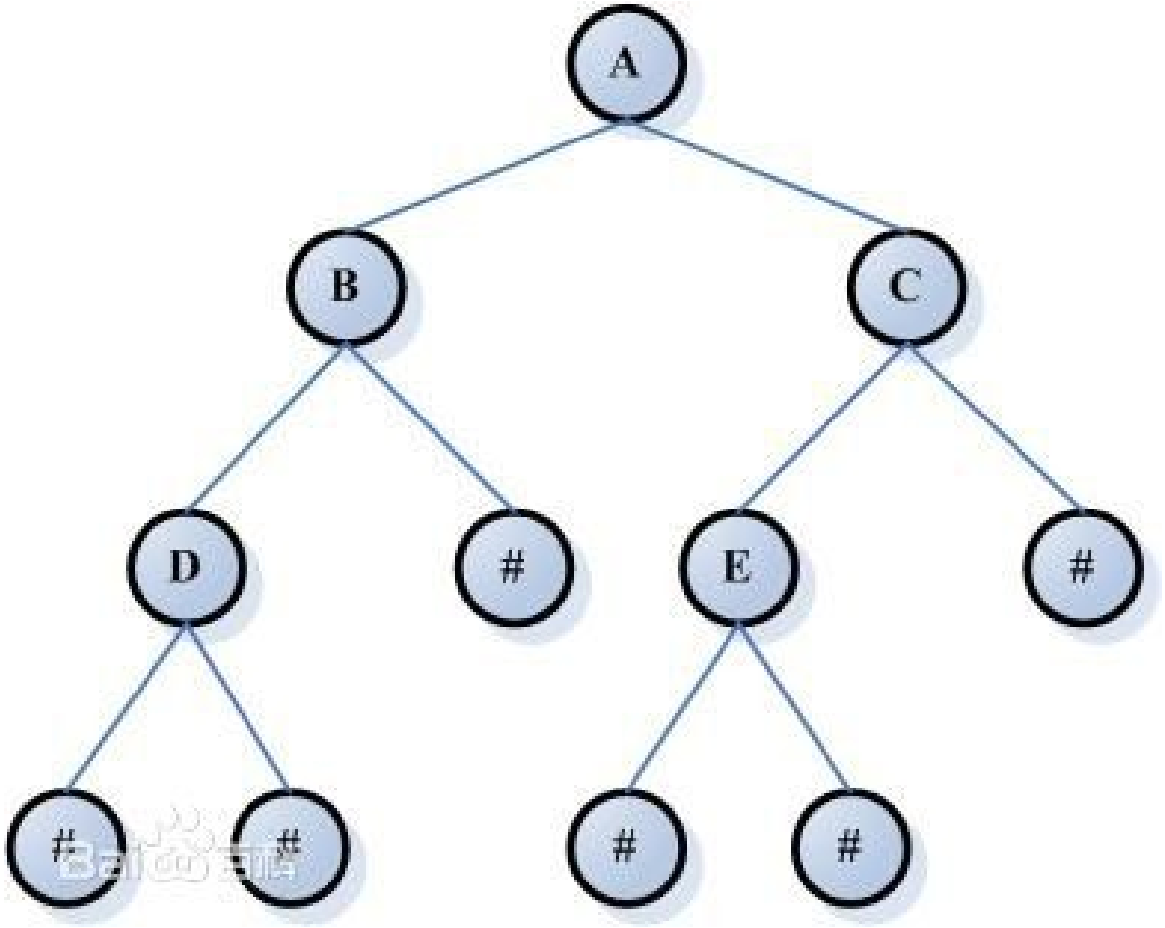
```
int find_father(int x)
{
    while (father[x ]!=x)
            father[x]=find_father(father[x]) ;

    return  father[x];
}
```

堆

先说说树

二叉树



二叉树

父亲和儿子

上面的叫父亲（祖先）

下面的叫儿子

儿子还分左儿子和右儿子

当然，左面的叫左儿子，右面的叫右儿子

树的第一层那一个结点叫做根，最后一层的那一排结点叫叶子

满二叉树

我需要画个图!

完全二叉树

我还需要画个图!

有了二叉树我们能干什么呢?

推广到一般树

树的根，树的叶子，树的儿子和树的父亲

树的实现需要用指针 + 结构体

说到了堆

堆是什么?

堆是计算机科学中一类特殊的数据结构的统称。堆通常是一个可以被看做一棵树的数组对象。堆总是满足下列性质:

堆中某个节点的值总是不大于或不小于其父节点的值;

堆总是一棵完全树。

堆所需的数据结构

（堆也是种数据结构）

完全树

对堆的操作

build: 建立一个空堆；

insert: 向堆中插入一个新元素；

update： 将新元素提升使其符合堆的性质；

get： 获取当前堆顶元素的值；

delete： 删除堆顶元素；

heapify： 使删除堆顶元素的堆再次成为堆。

建空堆

画图！

加元素

画图！

Update

画图!

删除后的堆更新操作

把最后一个元素提到堆顶然后与儿子比较，向下递推

画图!

有了堆，我们就有了优先队列

普通的队列是一种先进先出的数据结构，元素在队列尾追加，而从队列头删除。在优先队列中，元素被赋予优先级。当访问元素时，具有最高优先级的元素最先删除。优先队列具有最高级先出（largest-in， first-out）的行为特征。

其实优先队列的 STL 实现就是用堆！

STL 中包含优先队列容器（ priority_queue ）

heap 在 algorithm 中作为算法出现

由于这些数据结构涉及算法较多，故以后为大家留练习题，大家可以自己上网找一些 STL 优先队列的用法自己实验着学习一下。

剩余时间： http://www.cplusplus.com/reference/stl/
自己看看对应容器的操作

下午 12： 00 开题，望大家能够 AK