堆

1991 年计算机先驱奖获得者、斯坦福大学计算机科学系教授罗伯特 •弗洛伊德(Robert W. Floyd)和威廉姆斯(J. Williams)在 1964 年共同发明了著名的堆排序算法(Heap Sort)

堆的定义：

n 个关键字序列 K_1，K_2，…，K_n 称为堆，当且仅当该序列满足如下性质(简称为堆性质)：

小根堆：(1) $k_i \leqslant K_{2i}$ 且 $k_i \leqslant K_{2i+1}$ $(1 \leqslant i \leqslant n)$

或者

大根堆：(2) $K_i \geqslant K_{2i}$ 且 $k_i \geqslant K_{2i+1}$ $(1 \leqslant i \leqslant n)$

若将此序列所存储的向量 R[1..n]看做是一棵完全二叉树的存储结构，则堆实质上是满足如下性质的完全二叉树：

树中任一非叶结点的关键字均不大于(或不小于)其左右孩子(若存在)结点的关键字。

堆排序过程：

给定两个基本操作：（以小根堆为例）

1)UP：对于一个结点，通过不断与父节点的比较达成调整大小顺序的操作。具体操作是与其父节点进行比较（由于完全二叉树的特殊性，该节点的父节点的位置是子节点位置的二分之一），如果子节点更小则将父节点与子节点进行交换。如果原来以子节点

为根的子树符合堆构成的树的性质，则将其根节点交换为原根节点的父节点所组成的子树也符合。原来以父节点为根的子树同理。效率 O(logn)

示例代码：

```
inline void Heap_up(int x)
{   int k,j=x;
    while (j>1)
    {     k=j/2;
            if (a[k]>a[j]) S(j,k);
            j=k;
            }
}
```

2)DOWN：对于一个父节点，通过不断与其子节点的比较达成调整大小顺序的操作。具体操作是将父节点与其所有子节点进行比较，择所有子节点中数值最小的作为新的父节点，原父节点交换到数值最小子节点的位置上。效率 O(logn)

示例代码：

```
inline void Heap_down(int x)
{   int k,j=x;
    while (j*2<=i)
    {     k=j*2;
        if ((k<i)&&(a[k]>a[k+1])) k++;
```

```
        if (a[k]<a[j]) S(j,k); j=k;
        }
}
```

【定义：inline void S(int i,int j) { int t=a[i];a[i]=a[j];a[j]=t; }】

可以联想到一些延伸操作：

1) 创建堆/加入新元素：如果要建立 n 个元素的堆，依次将 n 个元素加入堆。即对于第 i 个元素，作为一个堆的第 i 个单位也就是当前最后一个单位加入（即依完全二叉树顺次序），使用 UP 操作调整堆。

2) 删除元素：堆的有价值元素即堆的根，也就是 R[1]，删除 R[1]，其位置由 R[n]填充，然后对当前的 R[1]使用 DOWN 操作调整堆。

堆分为大根堆和小根堆，是完全二叉树。大根堆的要求是每个节点的值都不大于其父节点的值，即 A[PARENT[i]] >= A[i]。在数组的非降序排序中，需要使用的就是大根堆，因为根据大根堆的要求可知，最大的值一定在堆顶。

下面以一道简单的题目作为例子：

合并果子

题目描述 Description

在一个果园里，多多已经将所有的果子打了下来，而且按果子的不同种类分成了不同的堆。多多决定把所有的果子合成一堆。

每一次合并，多多可以把两堆果子合并到一起，消耗的体力等于两堆果子的重量之和。可以看出，所有的果子经过 n-1 次合并之后，就只剩下一堆了。多多在合并果子时总共消耗的体力等于每次合并所耗体力之和。

因为还要花大力气把这些果子搬回家，所以多多在合并果子时要尽可能地节省体力。假定每个果子重量都为 1，并且已知果子的种类数和每种果子的数目，你的任务是设计出合并的次序方案，使多多耗费的体力最少，并输出这个最小的体力耗费值。

例如有 3 种果子，数目依次为 1，2，9。可以先将 1、2 堆合并，新堆数目为 3，耗费体力为 3。接着，将新堆与原先的第

三堆合并，又得到新的堆，数目为 12，耗费体力为 12。所以多多总共耗费体力=3+12=15。可以证明 15 为最小的体力耗费值。

输入输出格式 Input/output

输入格式:

输入文件 fruit.in 包括两行，第一行是一个整数 n(1<=n<=10000)，表示果子的种类数。第二行包含 n 个整数，用空格分隔，第 i 个整数 ai(1<=ai<=20000)是第 i 种果子的数目。

输出格式:

输出文件 fruit.out 包括一行，这一行只包含一个整数，也就是最小的体力耗费值。输入数据保证这个值小于 2^31。

输入输出样例 Sample input/output

输入样例:

3

1 2 9

输出样例:

15

说明 description

对于 30％的数据，保证有 n<=1000：

对于 50％的数据，保证有 n<=5000；

对于全部的数据，保证有 n<=10000。

这是标准的用堆排序解决的题目。如果只是用一般的排序取最小值，我们会发现随着两个数的消失，会产生一个新的数，而这个新数的诞生，让整个有序列都受到了影响。而如果使用堆排序，我们每次从堆头连续取两次值，即最小的两个苹果堆，又继而将诞生的新数加入堆，每次维护的效率是 O（logn），轻松解决了这个问题。

练习：

poj2833

poj3253

poj2388

poj2823

hdu1040

hdu1425

hdu1280

并查集

在一些有 N 个元素的集合应用问题中，我们通常是在开始时让每个元素构成一个单元素的集合，然后按一定顺序将属于同一组的元素所在的集合合并，其间要反复查找一个元素在哪个集合中。这一类问题近几年来反复出现在信息学的国际国内赛题中，其特点是看似并不复杂，但数据量极大，若用正常的数据结构来描述的话，往往在空间上过大，计算机无法承受；即使在空间上勉强通过，运行的时间复杂度也极高，根本就不可能在比赛规定的运行时间（1～3 秒）内计算出试题需要的结果，只能用并查集来描述。

为了比较容易理解并查集的原理，我将借用一个有爱的例子。

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。

下面我们来看并查集的实现。 int fa[1000]; 这个数组，记录了每个大侠的上级是谁。大侠们从 1 或者 0 开始编号（依据题意而定），fa[15]=3 就表示 15 号大侠的上级是 3 号大侠。如果一个人的上级就是他自己，那说明他就是掌门人了，查找到此为止。也有孤家寡人自成一派的，比如欧阳锋，那么他的上级就是他自己。每个人都只认自己的上级。比如胡青牛同学只知道自己的上级是杨左使。张无忌是谁？不认识！要想知道自己的掌门是谁，只能一级级查上去。 find 这个函数就是找掌门用的，意义再清

楚不过了（路径压缩算法先不论，后面再说）。

```
int find(int x)                                    //查找我（x）的掌门
{
    int r=x;                                       //委托 r 去找掌门
    while (fa[r]!=r)                               //如果 r 的上级不是 r 自己（也就是说找到的大侠他不是掌门 ==）
   r=fa[r ] ;                                      // r 就接着找他的上级，直到找到掌门为止。
    return  r ;                                    //掌门驾到~~~
}
```

再来看看 join 函数，就是在两个点之间连一条线，这样一来，原先它们所在的两个板块的所有点就都可以互通了。这在图上很好办，画条线就行了。但我们现在是用并查集来描述武林中的状况的，一共只有一个 fa[]数组，该如何实现呢？ 还是举江湖的例子，假设现在武林中的形势如图所示。虚竹小和尚与周芷若 MM 是我非常喜欢的两个人物，他们的终极 boss 分别是玄慈方丈和灭绝师太，那明显就是两个阵营了。我不希望他们互相打架，就对他俩说：“你们两位拉拉勾，做好朋友吧。”他们看在我的面子上，同意了。这一同意可非同小可，整个少林和峨眉派的人

就不能打架了。这么重大的变化，可如何实现呀，要改动多少地方？其实非常简单，我对玄慈方丈说："大师，麻烦你把你的上级改为灭绝师太吧。这样一来，两派原先的所有人员的终极 boss 都是师太，那还打个球啊！反正我们关心的只是连通性，门派内部的结构不要紧的。"玄慈一听肯定火大了："我靠，凭什么是我变成她手下呀，怎么不反过来？我抗议！"抗议无效，上天安排的，最大。反正谁加入谁效果是一样的，我就随手指定了一个。这段函数的意思很明白了吧？

```
void join(int x,int y)                                //我想让虚竹和周芷若做朋友
{
    int fx=find(x),fy=find(y);                        //虚竹的老大是玄慈，芷若 MM 的老大是灭绝
    if(fx!=fy)                                        //玄慈和灭绝显然不是同一个人
    fa[fx ]=fy;                                       //方丈只好委委屈屈地当了师太的手下啦
}
```

再来看看路径压缩算法。建立门派的过程是用 join 函数两个人两个人地连接起来的，谁当谁的手下完全随机。最后的树状结构会变成什么胎唇样，我也完全无法预计，一字长蛇阵也有可能。这样查找的效率就会比较低下。最理想的情况就是所有人的直接

上级都是掌门，一共就两级结构，只要找一次就找到掌门了。哪怕不能完全做到，也最好尽量接近。这样就产生了路径压缩算法。设想这样一个场景：两个互不相识的大侠碰面了，想知道能不能揍。 于是赶紧打电话问自己的上级：“你是不是掌门？” 上级说：“我不是呀，我的上级是谁谁谁，你问问他看看。” 一路问下去，原来两人的最终 boss 都是东厂曹公公。 “哎呀呀，原来是记己人，西礼西礼，在下三营六组白面葫芦娃!” “幸会幸会，在下九营十八组仙子狗尾巴花！” 两人高高兴兴地手拉手喝酒去了。“等等等等，两位同学请留步，还有事情没完成呢!”我叫住他俩。 “哦，对了，还要做路径压缩。”两人醒悟。 白面葫芦娃打电话给他的上级六组长：“组长啊，我查过了，其习偶们的掌门是曹公公。不如偶们一起及接拜在曹公公手下吧，省得级别太低，以后查找掌门麻环。” “唔，有道理。” 白面葫芦娃接着打电话给刚才拜访过的三营长……仙子狗尾巴花也做了同样的事情。这样，查询中所有涉及到的人物都聚集在曹公公的直接领导下。每次查询都做了优化处理，所以整个门派树的层数都会维持在比较低的水平上。

在平常的操作中，路径压缩往往是对 find 函数的优化：

```
inline int find(int x)
{    if (x==fa[x]) return x; int t=fa[x];
     fa[x]=Find(fa[x]); id[x]=(id[x]+id[t])%3; return fa[x];
}
```

畅通工程

Time Limit: 4000/2000 MS (Java/Others)   Memory Limit: 65536/32768 K (Java/Others)
Total Submission(s): 36081   Accepted Submission(s): 19129

Problem Description

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？

Input

测试输入包含若干测试用例。每个测试用例的第 1 行给出两个正整数，分别是城镇数目 N (< 1000)和道路数目 M；随后的 M 行对应 M 条道路，每行给出一对正整数，分别是该条道路直接连通的两个城镇的编号。为简单起见，城镇从 1 到 N 编号。
注意:两个城市之间可以有多条道路相通,也就是说
3 3
1 2
1 2
2 1
这种输入也是合法的
当 N 为 0 时，输入结束，该用例不被处理。

Output

对每个测试用例，在 1 行里输出最少还需要建设的道路数目。

Sample Input

```
4 2
1 3
4 3
3 3
1 2
1 3
2 3
5 2
1 2
3 5
999 0
0
```

Sample Output

```
1
0
2
998
```

很基础的并查集。做法就是通过题目给定的数据建立互相的联系，然后寻找有几个分块——即有多少个掌门人。需要建立的路径数就是区块数-1。

练习：

Hdu1272、Poj1308、Poj1611

POJ 1182 食物链

POJ 1456 Supermarket

POJ 1733 Parity game

hdu 3038 How Many Answers Are Wrong

POJ 1417 True Liars

最小生成树

描述：一个有 n 个结点的连通图的生成树是原图的极小连通子图，且包含原图中的所有 n 个结点，并且有保持图连通的最少的边。最小生成树可以用 kruskal（克鲁斯卡尔）算法或 prim（普里姆）算法求出

kruskal 算法简单描述：

1).记 Graph 中有 v 个顶点，e 个边

2).新建图 $Graph_{new}$，$Graph_{new}$ 中拥有原图中相同的 e 个顶点，但没有边。

3).将原图 Graph 中所有 e 个边按权值从小到大排序。

4).循环：从权值最小的边开始遍历每条边，直至图 Graph 中所有的节点都在同一个连通分量中。如果这条边连接的两个节点于图 $Graph_{new}$ 中不在同一个连通分量中，则添加这条边到图 $Graph_{new}$ 中——这个操作可以用并查集优化

代码示例：

```
sort(&a[1],&a[n+1],cmp);
for (i=1;i<=n;i++) if (find(a[i].x)!=find(a[i].y))
{    fa[find(a[i].x)]=find(a[i].y);
     sum++;
     if (sum==m-1) break;
```

```
}
```

Prim 算法简单描述：

1).输入：一个加权连通图，其中顶点集合为 V，边集合为 E。

2).初始化：V_{new} = {x}，其中 x 为集合 V 中的任一节点（起始点），E_{new} = {},为空。

3).重复下列操作，直到 V_{new} = V：

a.在集合 E 中选取权值最小的边<u, v>，其中 u 为集合 V_{new} 中的元素，而 v 不在 V_{new} 集合当中，并且 v∈V（如果存在有多条满足前述条件即具有相同权值的边，则可任意选取其中之一）；

b.将 v 加入集合 V_{new} 中，将<u, v>边加入集合 E_{new} 中；

4).输出：使用集合 V_{new} 和 E_{new} 来描述所得到的最小生成树。

代码示例：

```
for (i=1;i<n;i++)
{    sum=2147483647;
     for (j=1;j<=n;j++) if (!F[j])
          for    (k=1;k<=n;k++)    if    ((F[k])&&(sum>f[j][k]))
sum=f[j][k],place=k;
     F[place]=false;
}
```

例题就不用讲了，随便几道练习：

kruskal：hdu1863 hdu1102 hdu1162

Prim：poj1251 hdu1875 hdu1233 hdu4463