小明陪小红去看钻石，他们从一堆钻石中随机抽取两颗并比较她们的重量。这些钻石的重量各不相同。在他们们比较了一段时间后，它们看中了两颗钻石 g1 和 g2。现在请你根据之前比较的信息判断这两颗钻石的哪颗更重。
给定两颗钻石的编号 g1,g2，编号从 1 开始，同时给定关系数组 vector,其中元素为一些二元组，第一个元素为一次比较中较重的钻石的编号，第二个元素为较轻的钻石的编号。最后给定之前的比较次数 n。请返回这两颗钻石的关系，若 g1 更重返回 1，g2 更重返回-1，无法判断返回 0。输入数据保证合法，不会有矛盾情况出现。
测试样例：

```
2,3,[[1,2],[2,4],[1,3],[4,3]],4

返回：1
```

```
1   //就是一个森林,关系存在就是以 g2 为根节点的树下面的节点中有 g1,
2   //或者以 g1 为根节点的树的下面的节点包含 g2
3   //我们采取层序遍历的方式遍历以 g1 开头的整棵树,和以 g2 开头的整棵树.
4   #include<unordered_map>
5   class Cmp {
6       bool judge(int g1,int g2,unordered_map<int,vector<int>> ans){
7       //查找 g1 是否比 g2 重.
8           queue<int>q;
9           unordered_map<int, bool>mark;//用于标记当前节点是否遍历过
10          q.push(g1);
11          while (!q.empty()) {
12              int cur = q.front();
13              q.pop();
14              mark[cur] = true;
15              if(cur==g2)
16                  return true;
17              for(int i=0;i<ans[cur].size();++i){
18                  if(!mark[ans[cur][i]])//没有遍历过
19                      q.push(ans[cur][i]);
20              }
21          }
22          return false;
23      }
24  public:
25      int cmp(int g1, int g2, vector<vector<int> > records, int n) {
26          unordered_map<int,vector<int>>ans;
27          for (int i=0; i<n; ++i)
28              ans[records[i][0]].push_back(records[i][1]);
29          if(judge(g1, g2, ans))
30              return 1;
```

```
31                  else{
32                          if(judge(g2, g1, ans))
33                                  return -1;
34                          else
35                                  return 0;
36                  }
37          }
38  };
```

有一棵二叉树，树上每个点标有权值，权值各不相同，请设计一个算法算出权值最大的叶节点到权值最小的叶节点的距离。二叉树每条边的距离为 1，一个节点经过多少条边到达另一个节点为这两个节点之间的距离。
给定二叉树的根节点 root，请返回所求距离。

进制编码题，算出叶子节点二进制编码，再比编码，计算后缀长度和

```
.util.*;


s TreeNode {
 val = 0;
eNode left = null;
eNode right = null;
lic TreeNode(int val) {
     this.val = val;



s Tree {
vate int  max=0;
vate int min=99999;
vate StringBuilder maxcodec;
vate StringBuilder mincodec;
    void PreOrder(TreeNode T,char code,StringBuilder codec){
    if(T!=null){
            codec.append(code);
            if(T.left==null && T.right==null)
            {
                if(max<T.val)
                {
                    max=T.val;
                    maxcodec = codec;
                }

                if(min>T.val)
                {
                    min=T.val;
                    mincodec = codec;
                }
            }
            PreOrder(T.left,'0',new StringBuilder(codec));
            PreOrder(T.right,'1',new StringBuilder(codec));
    }

lic int getDis(TreeNode root) {
     PreOrder(root,'0',new StringBuilder());
     int index=0;
     for(index=0;
odec.length()>mincodec.length()?maxcodec.length():mincodec.length());index++)
```

```
        {
                if(maxcodec.charAt(index)!=mincodec.charAt(index))
                        break;
        }
        return
ubstring(index).length()+mincodec.substring(index).length());

        // write code here
```

有一个整数数组，请你根据快速排序的思路，找出数组中第 K 大的数。
给定一个整数数组 a,同时给定它的大小 n 和要找的 K(K 在 1 到 n 之间)，请返回第 K 大的数，保证答案存在。
测试样例：

```
[1,3,5,2,2],5,3
```

返回：2

这题应该是用快排的思想：例如找 49 个元素里面第 24 大的元素，那么按如下步骤：

1.进行一次快排（将大的元素放在前半段，小的元素放在后半段）,假设得到的中轴为 p

2.判断 p - low + 1 == k ，如果成立，直接输出 a[p]，（因为前半段有 k - 1 个大于a[p]的元素，故 a[p]为第 K 大的元素）

3.如果 p - low + 1 > k， 则第 k 大的元素在前半段，此时更新 high = p - 1，继续进行步骤 1

4.如果 p - low + 1 < k， 则第 k 大的元素在后半段， 此时更新 low = p + 1，且 k = k - (p - low + 1)，继续步骤 1.

由于常规快排要得到整体有序的数组，而此方法每次可以去掉“一半”的元素，故实际的复杂度不是 o(nlgn), 而是 o(n)。

附上代码：

```
public class Finder {
    public int findKth(int[] a, int n, int K)
        { return findKth(a, 0, n-1, K);
    }

    public int findKth(int[] a, int low, int high, int k)
        { int part = partation(a, low, high);

        if(k == part - low + 1) return a[part];
        else if(k > part - low + 1) return findKth(a, part + 1, high, k - part + low -1);
        else return findKth(a, low, part -1, k);

    }

    public int partation(int[] a, int low, int high)
        { int key = a[low];

        while(low < high) {
            while(low < high && a[high] <= key) high--;
```

```
            a[low] = a[high];
            while(low < high && a[low] >= key) low++;
            a[high] = a[low];
        }
        a[low] = key;
        return low;
    }
}
```