# Problem A. 众数

https://vpn.bupt.edu.cn/http/10.105.242.80/problem/p/270/

## 题目描述

给定一个长度为N的非降数列，求数列中出现次数最多的数。如果答案不唯一，输出其中最小的数。

## 输入格式

输入数据第一行是一个整数T(1≤T≤100)，表示测试数据的组数。对于每组测试数据:

第一行是一个正整数N(1≤N≤100)，表示数列长度。

第二行有N个整数，整数之间用空格隔开，所有的整数都不超过105，表示这个数列。

## 输出格式

对于每组测试数据，输出一个整数。

## 输入样例

2  
4  
1 1 1 2  
5  
1 1 2 2 3

## 输出样例

1  
1

## AC代码

### Hash计数数组

如果数组开小了，会导致RE（运行时错误）

用时：10ms

#include <bits/stdc++.h>  
#define FF(a,b) for(int a=0;a<b;a++)  
#define F(a,b) for(int a=1;a<=b;a++)  
#define LEN 110  
#define INF 2000  
#define bug(x) cout<<#x<<"="<<x<<endl;  
   
using namespace std;  
typedef long long ll;  
const double pi=acos(-1);  
   
//map<int,int> cnt;//超时  
int cnt[100001];  
   
int main()  
{  
// freopen("./in","r",stdin);  
 int T,N;  
 scanf("%d",&T);  
 int t;  
 while(T--){  
 scanf("%d",&N);  
// cnt.clear();//超时  
 memset(cnt,0,sizeof cnt);  
 int maxCnt=0;  
 int maxNum=0;  
 while(N--){  
 scanf("%d",&t);  
 cnt[t]=cnt[t]+1;  
 if(cnt[t]>maxCnt || (cnt[t]==maxCnt && t<maxNum)){  
 maxCnt=cnt[t];  
 maxNum=t;  
 }  
 }  
 printf("%d\n",maxNum);  
 }  
 return 0;  
}

### STL自带的红黑树

需要注意的是，对于每组测试如果用cnt.clear()来初始化，会用时2000ms导致**超时**。

必须使用 cnt=map<int,int>()**强行初始化**。

用时：4ms

#include <bits/stdc++.h>  
#define FF(a,b) for(int a=0;a<b;a++)  
#define F(a,b) for(int a=1;a<=b;a++)  
#define LEN 110  
#define INF 2000  
#define bug(x) cout<<#x<<"="<<x<<endl;  
  
using namespace std;  
typedef long long ll;  
const double pi=acos(-1);  
  
map<int,int> cnt;  
//int cnt[100001];  
  
int main()  
{  
// freopen("./in","r",stdin);  
 int T,N;  
 scanf("%d",&T);  
 int t;  
 while(T--){  
 scanf("%d",&N);  
// cnt.clear();//超时  
 cnt=map<int,int>();  
 int maxCnt=0;  
 int maxNum=0;  
 while(N--){  
 scanf("%d",&t);  
 cnt[t]=cnt[t]+1;  
 if(cnt[t]>maxCnt || (cnt[t]==maxCnt && t<maxNum)){  
 maxCnt=cnt[t];  
 maxNum=t;  
 }  
 }  
 printf("%d\n",maxNum);  
 }  
 return 0;  
}

# Problem B. 旋转图像

https://vpn.bupt.edu.cn/http/10.105.242.80/problem/p/271/

## 题目描述

在图形学中，我们经常需要对具体的图像进行一些处理。在这个问题中，你的任务是将一幅只包含01像素点的图片进行顺时针旋转。旋转的角度仅包含0度，90度，180度和270度。

## 输入格式

输入的第行是一个整数T (T≤50)，表示输入的数据组数。

每组测试数据的第一.行是两个整数N和M(1≤N,M≤50),表示图片的高度和宽度。

接下来N行，每行是一个长度为M的01串，表示图片的像素点。最后一行是一个整数angle,表示旋转的角度。

## 输出格式

对于每组测试数据，输出旋转后得到的图片。请注意不要输出多余的空格或空行。

## 输入样例

2  
2 3  
111  
000  
90  
3 3  
111  
101  
111  
180

## 输出样例

01  
01  
01  
111  
101  
111

## AC代码

### 编制3个旋转函数：低时间复杂度

用时：31ms

针对90°，180°，270°各编制一个旋转函数，相比法2时间复杂度要低

#include <bits/stdc++.h>  
#define FF(a,b) for(int a=0;a<b;a++)  
#define F(a,b) for(int a=1;a<=b;a++)  
#define LEN 60  
#define INF 2000  
#define bug(x) cout<<#x<<"="<<x<<endl;  
  
using namespace std;  
typedef long long ll;  
const double pi=acos(-1);  
  
typedef struct Mat{  
 int N,M;//height,width  
 char data[LEN][LEN];  
 void getSize(){  
 scanf("%d%d",&N,&M);  
 }  
 void getData(){  
 FF(i,N){  
 scanf("%s",data[i]);  
 }  
 }  
 Mat rot90(){  
 Mat ans;  
 ans.N=M;ans.M=N;  
 FF(i,N)FF(j,M)ans.data[j][N-1-i]=data[i][j];  
 return ans;  
 }  
 Mat rot180(){  
 Mat ans;  
 ans.N=N;ans.M=M;  
 FF(i,N)FF(j,M)ans.data[N-1-i][M-1-j]=data[i][j];  
 return ans;  
 }  
 Mat rot270(){  
 Mat ans;  
 ans.N=M;ans.M=N;  
 FF(i,N)FF(j,M)ans.data[M-1-j][i]=data[i][j];  
 return ans;  
 }  
 Mat rot(int rot){  
 switch(rot){  
 case 0:  
 return \*this;  
 case 90:  
 return rot90();break;  
 case 180:  
 return rot180();break;  
 case 270:  
 return rot270();break;  
 }  
 }  
 void print(){  
 FF(i,N){  
 FF(j,M)  
 putchar(data[i][j]);  
 puts("");  
 }  
 }  
}Mat;  
  
int main()  
{  
// freopen("./in","r",stdin);  
 int T,N;  
 scanf("%d",&T);  
 int rot;  
 while(T--){  
 Mat a;  
 a.getSize();  
 a.getData();  
 scanf("%d",&rot);  
 Mat b=a.rot(rot);  
 b.print();  
 }  
 return 0;  
}

### 编制1个旋转函数循环调用：低编码复杂度

用时：22ms

根据角度的不同，对rot90()调用0-3次

很尴尬的发现比法1还快… …

其实在小数据量的情况下，一些优化看不出明显的效果，比如dijkstra的堆优化，小数据量的情况下甚至会因为优先队列的开销比普通法还要慢，但是到了大数据量下，优势就显现了。

#include <bits/stdc++.h>  
#define FF(a,b) for(int a=0;a<b;a++)  
#define F(a,b) for(int a=1;a<=b;a++)  
#define LEN 60  
#define INF 2000  
#define bug(x) cout<<#x<<"="<<x<<endl;  
  
using namespace std;  
typedef long long ll;  
const double pi=acos(-1);  
  
typedef struct Mat{  
 int N,M;//height,width  
 char data[LEN][LEN];  
 void getSize(){  
 scanf("%d%d",&N,&M);  
 }  
 void getData(){  
 FF(i,N){  
 scanf("%s",data[i]);  
 }  
 }  
 Mat rot90(){  
 Mat ans;  
 ans.N=M;ans.M=N;  
 FF(i,N)FF(j,M)ans.data[j][N-1-i]=data[i][j];  
 return ans;  
 }  
 Mat rot(int rot){  
 FF(i,rot/90)  
 \*this=rot90();  
 return \*this;  
 }  
 void print(){  
 FF(i,N){  
 FF(j,M)  
 putchar(data[i][j]);  
 puts("");  
 }  
 }  
}Mat;  
  
int main()  
{  
// freopen("./in","r",stdin);  
 int T,N;  
 scanf("%d",&T);  
 int rot;  
 while(T--){  
 Mat a;  
 a.getSize();  
 a.getData();  
 scanf("%d",&rot);  
 Mat b=a.rot(rot);  
 b.print();  
 }  
 return 0;  
}

# Problem C. 网络的核

https://vpn.bupt.edu.cn/http/10.105.242.80/problem/p/272/

## 题目描述

给定一个无向网络G,网络中共有N个节点(从1编号到N), M条边，求网络的核。

**网络的核**: 到网络中其他节点的距离之和最小的节点。且对于不连通的两点，它们之间的距离为N。

如果有多组解，输出编号最小的节点。

## 输入格式

输入的第一行是一个整数T(T≤25)，表示输入的数据组数。

对于每组测试数据:

* 第一行有两个整数N,M(1≤N≤50, 0≤M≤N\*(N-1),表示网络中有N个点，M条边。
* 接下来M行，每行两个整数u,v(1≤u,v≤N, u≠v),表示点u和点v之间有一条距离为1的边。任意两个点之间最多只会有一条边相连。

## 输出格式

对于每组测试数据，输出网络的核。

## 输入样例

2  
3 3  
1 2  
1 3  
2 3  
4 2  
1 2  
2 3

输出样例

1  
2

## AC代码

### Floyd算法

时间：10ms

#include <bits/stdc++.h>  
#define FF(a,b) for(int a=0;a<b;a++)  
#define F(a,b) for(int a=1;a<=b;a++)  
#define LEN 60  
#define INF 100000  
#define bug(x) cout<<#x<<"="<<x<<endl;  
  
using namespace std;  
typedef long long ll;  
const double pi=acos(-1);  
  
int g[LEN][LEN];  
int N,M;  
  
int calc\_dist(int h){  
 int ans=0;  
 F(i,N)if(i!=h)ans+=g[h][i];  
 return ans;  
}  
  
int main()  
{  
// freopen("./in","r",stdin);  
 int T;  
 scanf("%d",&T);  
  
 int u,v;  
 while(T--){  
 scanf("%d%d",&N,&M);  
 F(i,N)F(j,N) g[i][j]=N;  
 while(M--){  
 scanf("%d%d",&u,&v);  
 g[u][v]=g[v][u]=1;  
 }  
 //floyd  
 F(k,N)F(i,N)F(j,N)if(g[i][k]+g[k][j]<g[i][j]) g[i][j]=g[i][k]+g[k][j];  
 //遍历寻找网络的核  
 int minD=INF,minV=0;  
 F(i,N){ //可以看到顶点i从1遍历到N，从小到大，所以不需要对顶点字典序进行判断  
 int d=calc\_dist(i);  
 if(d<minD){  
 minV=i;  
 minD=d;  
 }  
 }  
 printf("%d\n",minV);  
  
 }  
 return 0;  
}

### Dijkstra算法

理论上dijkstra算法+堆优化求N个顶点的dist数组的时间复杂读为**O(N2log2N)**

时间有限，我就不编码了。如果读者完成了这个思路的代码请您留言给我。

# Problem D. Python List

https://vpn.bupt.edu.cn/http/10.105.242.80/problem/p/273/

## 题目描述

在Python中，List (列表)是一种非常重要的数据结构。它与C/C++/Java中的数组有些类似，但支持添加新元素时的动态扩展。在这个问题中，你需要处理如下的几种对List的操作。

* L=[]: 将名字为L的List清空。在这里，List 的名字是长度为1到10之间

的字符串(只包括大小写字母)。如果L原来不存在，这个语句相当于定义了一个名字为L的空列表。

* L.append(x):向L的末端插入元素x。为方便起见，这里的x只会是[0, 65536]之间的整数。
* L.sort():将L中的元素按升序排序。
* L[id]:返回L中下标为id(id≥0)的值。下标是从0开始计数的。

给定若干Python语句，你的任务是对于每个形如L[id]的语句，输出它返回的值。

## 输入格式

输入数据包含多组测试数据。请注意各组测试数据之间是相互独立的。输入的第一行是一个整数T(T≤100)，表示测试数据的组数。

每组测试数据第一行是语句的数量N (N≤100)。接下来N行,每行一个python语句。测试数据保证只会出现上述四种语句，语句中间不会出现空格。一个List在被使用前一定会被先定义。

## 输出格式

对于每个查询，输出查找的L[id]的值。如果id超出了当前List 的下标范围，;输出一行ERROR。

## 样例输入

2  
5  
a=[]  
a.append(0)  
a.append(1)  
a[0]  
a[1]  
8  
lista=[]  
lista.append(123)  
lista.append(65)  
lista[0]  
lista.sort()  
lista[0]  
listb=[]  
listb[0]

## 样例输出

0  
1  
123  
65  
ERROR

## AC代码

本题属于复杂模拟，有csp第三题的既视感

#include <bits/stdc++.h>  
#define FF(a,b) for(int a=0;a<b;a++)  
#define F(a,b) for(int a=1;a<=b;a++)  
#define LEN 60  
#define INF 100000  
#define bug(x) cout<<#x<<"="<<x<<endl;  
  
using namespace std;  
typedef long long ll;  
const double pi=acos(-1);  
  
#define V vector<int>  
#define MP map<string,vector<int> >  
  
MP nm2lst;//name to list  
  
int str2int(string s){  
 int ans;  
 sscanf(s.c\_str(),"%d",&ans);  
 return ans;  
}  
  
void init(string s){  
 int pos=s.find("=");  
 string name=s.substr(0,pos);  
 nm2lst[name]=V();  
}  
void append(string s){  
 int pos=s.find(".");  
 int len=s.length();  
 string name=s.substr(0,pos);  
 int l=pos+8,r=len-2;  
 string s\_num=s.substr(l,r-l+1);//[pos+7,len-2]  
 nm2lst[name].push\_back(str2int(s\_num));  
}  
void \_sort(string s){  
 int pos=s.find(".");  
 string name=s.substr(0,pos);  
 V &v=nm2lst[name];  
 sort(v.begin(),v.end());  
}  
void \_index(string s){  
 int pos=s.find("[");  
 string name=s.substr(0,pos);  
 int len=s.length();  
 if(nm2lst.count(name)){  
 int l=pos+1,r=len-2;  
 string s\_num=s.substr(l,r-l+1);//[pos+7,len-2]  
 int idx=str2int(s\_num);  
 V &v=nm2lst[name];  
 if(idx>=0 && idx<v.size())  
 printf("%d\n",v[idx]);  
 else  
 puts("ERROR");  
 }else{  
 puts("ERROR");  
 }  
  
}  
int main()  
{  
// freopen("./in","r",stdin);  
 int T,N;  
 scanf("%d",&T);  
 char buf[1000];  
 while(T--){  
 nm2lst=MP();//强行初始化  
 scanf("%d",&N);  
 getchar();  
 while(N--){  
 gets(buf);  
 string str(buf);  
 if(str.find("=[]")!=string::npos){//赋初始值  
 init(str);  
 }else if(str.find("append")!=string::npos){//末尾添加  
 append(str);  
 }else if(str.find("sort")!=string::npos){//排序  
 \_sort(str);  
 }else{ //下标访问  
 \_index(str);  
 }  
 }  
 }  
 return 0;  
}