写在前面：这里面的 7 道题都是比较经典的 STL 题目，而且都是紫书当中提及到的题目，故希望各位同学能尽量独立完成。

A - Where is the Marble?

这道题直接使用 sort 函数对 n 个数进行排序，然后再逐个遍历查找即可。
排序之后也可以使用 lower_bound()函数;用于查找大于或等于 x 的第一个位置.查找速度会快点
参数为:　lower_bound(数组的开始位置,结束位置,查找的数据)

代码：

```
#include<bits/stdc++.h>
#define maxn 10000
using namespace std;
int s[maxn];
int main(){
    int N,Q,kase=0;;
    while(scanf("%d%d",&N,&Q)==2&&N){
     printf("CASE# %d:\n",++kase);
     for(int i=0;i<N;i++)
          cin>>s[i];
     sort(s,s+N);

     for(int j=1;j<=Q;j++){
          int t,k,cnt=0,flag=0;
          cin>>k;
          for(t=0;t<N;t++){
               if(s[t]==k){
                  flag=1;}
               if(flag)
                    break;
          }
          if(flag)
               printf("%d found at %d\n",k,t+1);
          else
               printf("%d not found\n",k);}
       /*
          while(Q--){
               int k；
               cin>>k;
               int p=lower_bound(s,s+N,k)-s;
               if(s[p]==k)
```

```
                printf("%d found at %d\n",k,p+1);
            else
                printf("%d not found\n",k);
        }
     */
    }
    return 0;
}
```

B - The Blocks Problem

这是一道模拟题，题目比较繁琐，代码量相对较大，但是难度并不是很大，这道题可以使用 stack 来做，也可以使用 vector 来做,用数组存储每个木块当前所在的位置，每次移动都要更新。另外，如果 a 和 b 木块在同一个位置就不需要移动。

代码：

```
#include<bits/stdc++.h>
using namespace std;

const int maxn =25;
vector<int> pile[maxn];
int n;
void find_block(int a, int& p, int& h) {   //找到木块所在的位置和高度（从 0 开始）
    for (p = 0; p < n; p++) {
        for (h = 0; h < pile[p].size(); h++) {
            if (pile[p][h] == a) return;
        }
    }
}

void clear_above(int p, int h) {   //将 p 位置高度 h 以上的木块移回原位
    for (int i = h + 1; i < pile[p].size(); i++) {
        int b = pile[p][i];
        pile[b].push_back(b);
    }
    pile[p].resize(h + 1);   //保留  0-h 高度的木块
}

void pile_onto(int p, int p2,int h) {     //将 p 位置处从 h 高度的木块全部移动到 p2 顶部
    for (int i = h; i < pile[p].size(); i++) {
        pile[p2].push_back(pile[p][i]);
```

```
        }
        pile[p].resize(h);    //保留高度 0~h-1 的木块
}

void print() {    //输出全部操作介绍后各个位置木块的信息
        for (int i = 0; i < n; i++) {
                cout << i << ':';
                for (int j = 0; j < pile[i].size(); j++)
                        cout << ' ' << pile[i][j];
                cout << endl;
        }
}

int main() {
        int ha, pa, hb, pb;
        cin >> n;
        for (int i = 0; i < n; i++) pile[i].push_back(i);
        string s1, s2;
        int a, b;
        while (cin >> s1 ) {
                if(s1 == "quit") break;
                cin>> a >> s2 >> b;
                find_block(a, pa, ha);
                find_block(b, pb, hb);
                if (pa == pb) continue;
                if (s1 == "move") clear_above(pa, ha);
                if (s2 == "onto") clear_above(pb, hb);
                pile_onto(pa, pb, ha);
        }
        print();
        return 0;
}
```

C - Rails

这道题是一个典型的栈问题，因为中转站中是符合先进后出的顺序的，故只需使用 stack 来模拟即可。

代码：

```
#include<bits/stdc++.h>
using namespace std;
const int N = 1e3 + 10;
```

```
int n;
int a[N];
stack<int> s, t;

int main()
{
    while(cin >> n){
        if(n == 0){
            break;
        }
        int m;
        while(cin >> m){
            if(m == 0){
                break;
            }
            while(!s.empty())
                s.pop();
            a[1] = m;
            for (int i = 2; i <= n; i++){
                cin >> a[i];
            }
            int start = 1, end = 1;                 //start 代表入站的顺序，end 代表出站的顺序
            int flag = 1;
            while(end <= n){
                if(start == a[end]){                //从入站进入一节，直接出战
                    start++;
                    end++;
                }else if(!s.empty() && s.top() == a[end]){          //从中间站台出站
                    s.pop();
                    end++;
                }else if(start <= n){               //当前的出战节点和入站，中间站，都不一样，从入站进入一节，后面接着判断
                    s.push(start++);
                }else{
                    flag = 0;
                    break;
                }
            }
            printf("%s\n", flag ? "Yes" : "No");
        }
        cout << endl;
    }
    return 0;
```

}

D - Team Queue

模拟题，简单的出队和入队问题。
用 map 来映射每个人与其所在的队伍，
用两个队列存，队列一是每个人的队伍编号
队列二是一个队列数组，存的是每个团队的人构成一个队列

代码：

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    map<int,int>ans;
    queue<int>q;                  //队列 1，每个成员的队伍编号
    queue<int>q2[2000];        //队列 2，每个团队成员构成一个队列
    int n,m,x;
    string s;
    int coun=0;
    while(cin>>n)
    {   if(n==0) break;
        cout<<"Scenario #"<<++coun<<endl;
    for(int i=0;i<n;i++)
    {
        cin>>m;
        while(m--)
        {
            cin>>x;
            ans[x]=i;        //给每个队伍成员映射一个队伍编号 （map）
        }
    }
    while(cin>>s)
    {
        if(s[0]=='S') break;
        else if(s[0]=='E')
        {
            cin>>x;
            int t=ans[x];
            if(q2[t].empty()) q.push(t);   //如果编号为 t 的队伍没有人，就在队列添加一个
编号 t 的队伍
```

```
                q2[t].push(x);              //在编号 t 的队伍里添加一个成员
            }
            else
            {
                int t=q.front();
                cout<<q2[t].front()<<endl;q2[t].pop();
                if(q2[t].empty()) q.pop();
            }
        }
        for(int i=0;i<n;i++)                                    //每次 STOP 后的数据都要清空!!!!
            while(q2[i].empty()==0) q2[i].pop();      //  因为初学不熟练，这三行的清空导致 wa
了 3 次。。
        while(q.empty()==0) q.pop();                    //  看了网上代码才想起来还要这个
        cout<<endl;
        }
        return 0;
}
```

E - Ugly Numbers

利用优先队列每次取出最小的那个数字，然后乘上 2,3,5，在判断这个数是否在集合当中即可。注意要使用 long long

代码：

```
#include<bits/stdc++.h>
using namespace std;
typedef long long LL;

priority_queue<LL, vector<LL>, greater<LL>> Q;
set<LL> S;

int main()
{
        int a[3]={2,3,5};
        Q.push(1);
        S.insert(1);
        for(int i=1;;i++)
        {
                long long now=Q.top();
                Q.pop();
                if(i==1500)
```

```
            {
                cout<<"The 1500'th ugly number is "<<now<<".\n";
                break;
            }
            for(int j=0;j<3;j++)
            {
                LL next=now*a[j];
                if(!S.count(next))
                {
                    S.insert(next);
                    Q.push(next);
                }
            }
        }
        return 0;
}
```

F - Ananagrams

这道可以很好的使用 map 来解决，首先将输入的字符串全部变成小写然后排好序放入 map 中，如果这个组合在 map 中只出现 1 次，那么绝对不可能重排得到别的单词

代码：

```
#include<bits/stdc++.h>
using namespace std;

string re(string s)
{
    for(int i=0;i<s.size();i++)
        s[i]=tolower(s[i]);
    sort(s.begin(),s.end());
    return s;
}

int main()
{
    string s;
    vector<string> words,ans;
    map<string,int> ma;
    while(cin>>s)
    {
```

```
        if(s[0]=='#') break;
        words.push_back(s);
        string x=re(s);
        if(!ma.count(x)) ma[x]=0;
        ma[x]++;
    }
    for(int i=0;i<words.size();i++)
    {
        if(ma[re(words[i])]==1){
            ans.push_back(words[i]);
        }
    }
    sort(ans.begin(),ans.end());
    for(int i=0;i<ans.size();i++) {
        cout<<ans[i]<<endl;
    }
    return 0;
}
```

G - The SetStack Computer

本题的集合是集合的集合。可以先为不同的集合分配一个唯一的 ID，则每个集合都可以表示成所包含元素的 ID 集合，这样就可以使用 STL 的 set<int>来表示了，而整个栈则是一个 stack<int>

代码：

```
#include<bits/stdc++.h>
using namespace std;
#define ALL(x) x.begin(),x.end()
#define INS(x) inserter(x, x.begin())
typedef set<int> Set;
map<Set,int> IDcache;
vector<Set> Setcache;

int ID(Set x)
{
    if(IDcache.count(x)) return IDcache[x];
    Setcache.push_back(x);
    return IDcache[x] = Setcache.size() - 1;
}
```

```
int main()
{
	int t, n;
	while(~scanf("%d",&t))
	{
		while(t--)
		{
			stack<int> s;
			scanf("%d",&n);
			while(n--)
			{
				string op;
				cin >> op;
				if(op[0] == 'P') s.push(ID(Set()));
				else if (op[0] == 'D') s.push(s.top());
				else
				{
					Set x1 = Setcache[s.top()]; s.pop();
					Set x2 = Setcache[s.top()]; s.pop();
					Set x;
					if(op[0] == 'U') set_union(ALL(x1), ALL(x2), INS(x));
					if(op[0] == 'I') set_intersection(ALL(x1), ALL(x2), INS(x));
					if(op[0] == 'A') { x = x2; x.insert(ID(x1)); }
					s.push(ID(x));
				}
				cout << Setcache[s.top()].size() << endl;
			}
			cout << "***" << endl;
		}
	}
	return 0;
}
```