A题:

数据范围很小，暴力

AC代码:

#include<bits/stdc++.h>

#define ll long long

#define INF 0x7f7f7f7f //2139062143

#define llINF 9223372036854775807

#define IOS ios::sync\_with\_stdio(0); cin.tie(0); cout.tie(0);

**using** **namespace** std;

**const** **int** maxn=1e6+7;

**const** **double** eps=1e-6;

**const** **int** mod=1e9+7;

**int** num[6];

**int** main(){

    IOS

**int** n;cin>>n;

**int** tot;//tot代表当前的数字有几位

**for**(**int** i=1;1;i++,n-=tot){

**int** temp=i;

        tot=0;

**while**(temp){

            num[tot++]=temp%10;

            temp/=10;

        }

**if**(tot>=n) {cout<<num[tot-n]<<endl;**break**;}

    }

}

B题:

数据范围很小，暴力，也可以不用map直接暴力比对后过掉

AC代码:

#include<bits/stdc++.h>

#define ll long long

#define INF 0x7f7f7f7f //2139062143

#define llINF 9223372036854775807

#define IOS ios::sync\_with\_stdio(0); cin.tie(0); cout.tie(0);

**using** **namespace** std;

**const** **int** maxn=1e6+7;

**const** **double** eps=1e-6;

**const** **int** mod=1e9+7;

map<string,**int**>M;

**int** main(){

    IOS

    string s1,s2;

    cin>>s1;

**for**(**int** i=0;i<s1.size();i++){

        s2.clear();

**for**(**int** j=i;j<s1.size();j++) s2+=s1[j];

**for**(**int** j=0;j<i;j++) s2+=s1[j];

        M[s2]=1;

    }

    cout<<M.size()<<endl;

}

C题:

给你一个数组，让你选定一个整数x，对这个数组的某些数字进行一次+x或者-x的操作，问能否通过这样的操作使得整个数组的数字都相等。

对于初始相等的值，我们可以进行一样的操作，所以先去重，整个数组中有k个不同的整数，按照k的值进行分类讨论。

当k=1的时候，不需要操作已经相等，此时答案为YES

当k=2的时候，我们可以把其中一个数全部变为另一个数，此时答案为YES

当k=3的时候，三个数为a<b<c，当且仅当a+c=b\*2的时候，可以让a和c一个进行加法操作一个进行减法操作后使得整个数组都为b。

当k>=4的时候无解。

AC代码:

#include<bits/stdc++.h>

#define ll long long

#define INF 0x7f7f7f7f //2139062143

#define llINF 9223372036854775807

#define IOS ios::sync\_with\_stdio(0); cin.tie(0); cout.tie(0);

**using** **namespace** std;

**const** **int** maxn=1e5+7;

**const** **double** eps=1e-6;

**const** **int** mod=1e9+7;

**int** n;

**int** num[maxn],cas[maxn];//cas为num去重后的数组

**int** tot=0;//tot记录cas数组中有几个数

**int** main(){

    IOS

    cin>>n;

**for**(**int** i=0;i<n;i++) cin>>num[i];

    sort(num,num+n);

**for**(**int** i=0;i<n;i++){

**if**(i==0||num[i]!=num[i-1]) cas[tot++]=num[i];

    }

**if**(tot<=2||tot==3&&cas[0]+cas[2]==cas[1]\*2) cout<<"YES"<<endl;

**else** cout<<"NO"<<endl;

}

D题:

实际上就是问你拓扑图有没有环，有环输出NO，无环输出YES

AC代码:  
#include<map>

#include<set>

#include<cmath>

#include<queue>

#include<stack>

#include<bitset>

#include<cstdio>

#include<string>

#include<vector>

#include<cstdlib>

#include<cstring>

#include<fstream>

#include<iomanip>

#include<sstream>

#include<iostream>

#include<algorithm>

#include<functional>

#define ll long long

#define llINF 9223372036854775807

**using** **namespace** std;

**int** n,m;

**int** ru[107];//ru[i]代表第i个点的入度是多少

vector<**int**>dir[107];//dir[i]保存了从第i个点出发有哪些边

**bool** tuopu(){

**int** tot=0;//统计拓扑排序过程中，排序了几个点

    queue<**int**>Q;

**for**(**int** i=0;i<n;i++) **if**(!ru[i]) Q.push(i);

**while**(Q.size()){

**int** now=Q.front();

        Q.pop();

        tot++;

**for**(**int** i=0;i<dir[now].size();i++){

**int** to=dir[now][i];

            ru[to]--;

**if**(!ru[to]) Q.push(to);

        }

    }

**return** tot==n;//如果无环，则拓扑排序能把所有n个点都排序出来

    //反之则有环

}

**int** main(){

**while**(1){

        scanf("%d%d",&n,&m);

**if**(n==0&&m==0) **break**;

        memset(ru,0,**sizeof**(ru));

**for**(**int** i=0;i<m;i++){

**int** a,b;scanf("%d%d",&a,&b);

            dir[a].push\_back(b);

            ru[b]++;

        }

**if**(tuopu()) printf("YES\n");

**else** printf("NO\n");

**for**(**int** i=0;i<n;i++) dir[i].clear();

    }

}

E题:

最短路模板题，这次刻意选了一个数据规模很小的题，点非常少才1000个，边最多也只有2000条，暴力版本的迪杰斯特拉是可以过的，spfa应该也可以。暴力版本或者spfa过的同学回去学一下堆优化的迪杰斯特拉，优化掉一个n为logn。不然数据规模大一点就过不去了。

AC代码:(堆优化迪杰斯特拉，vector存图，链式前向星（链表）存图也可以自己去学习下）

#include<map>

#include<set>

#include<cmath>

#include<queue>

#include<stack>

#include<bitset>

#include<cstdio>

#include<string>

#include<vector>

#include<cstdlib>

#include<cstring>

#include<fstream>

#include<iomanip>

#include<sstream>

#include<iostream>

#include<algorithm>

#include<functional>

#define ll long long

#define llINF 9223372036854775807

**using** **namespace** std;

**int** t,n;

**struct** Edge{

**int** to,dis;

};

vector<Edge>dir[1007];//dir[i]保存了与第i个点有哪些边

**int** dis[1007];//保存了与起点(第一个点)的最短距离

**struct** Node{

**int** now,dis;//now为当前位置，dis为与起点距离

**bool** **friend** **operator** < (Node a,Node b){

**return** a.dis>b.dis;

    }//重载运算符，使得priority\_queue从大根堆变为小根堆

    //从而在logn复杂度时间提供出，没走过的点中，当前距离最小的点

};

**void** Dij(){

    memset(dis,0x3f,**sizeof**(dis));//初始距离为无穷

    dis[1]=0;//起点与自身距离为0

    priority\_queue<Node>Q;

    Q.push({1,0});

**while**(Q.size()){

        Node temp=Q.top();

        Q.pop();

**if**(dis[temp.now]<temp.dis) **continue**;

**for**(**int** i=0;i<dir[temp.now].size();i++){

**int** to=dir[temp.now][i].to;

**int** D=dir[temp.now][i].dis+temp.dis;

**if**(D<dis[to]){

                dis[to]=D;

                Q.push({to,D});

            }

        }

    }

}

**int** main(){

    scanf("%d%d",&t,&n);

**for**(**int** i=0;i<t;i++){

**int** a,b,c;scanf("%d%d%d",&a,&b,&c);

        dir[a].push\_back({b,c});

        dir[b].push\_back({a,c});

    }

    Dij();

    printf("%d\n",dis[n]);

}

F题:

逆向思维，切割变成合并。

对于最后切割出来的部分，每次贪心选择最短的两个部分合并在一起成为新的一段，计算消耗，再放回去。重复操作。

同样利用优先队列降低一个n复杂度到logn

AC代码:  
#include<map>

#include<set>

#include<cmath>

#include<queue>

#include<stack>

#include<bitset>

#include<cstdio>

#include<string>

#include<vector>

#include<cstdlib>

#include<cstring>

#include<fstream>

#include<iomanip>

#include<sstream>

#include<iostream>

#include<algorithm>

#include<functional>

#define ll long long

#define llINF 9223372036854775807

**using** **namespace** std;

**int** n;

priority\_queue<ll>Q;

**int** main(){

    scanf("%d",&n);

**for**(**int** i=0;i<n;i++){

        ll x;scanf("%lld",&x);

        Q.push(-x);//由于priority\_queue默认是大根堆

        //因此这里把负数存进去，那么弹出来的数的绝对值实际上就是原本数中最小的那个数了

        //这是个取巧的方法，类似上一题堆优化Dij中重载运算符的方法最好是学一下

    }

    ll ans=0;

**while**(Q.size()>1){

        ll a=Q.top();Q.pop();

        ll b=Q.top();Q.pop();

        ans-=a+b;//存的是负数，所以加法变减法

        Q.push(a+b);

    }

    printf("%lld\n",ans);

}

G题:

原先的每个数字，我们忽略掉数值，用他们初始所在的位置来表示，那么原数组就是[1,2,3,4,....n]。

对于每次交换操作，我们会使得整个数组的逆序对数量+1，在最优的情况下我们的每次操作都应该使得逆序对+1，否则就是做了无用操作。

因此我们只需要处理出最后翻转后的数组，每个位置的数初始的位置是多少，算出用这个值构成的数组的逆序对数量，就是最少操作次数了。

而翻转后的数组每个数的初始位置，我们可以用贪心的原则来处理，对于同一个数字，原本越左边的，就去翻转后该数可在位置中越左边的。

比如原数组[1,3,4,3,3]，对于数字3来说，原本它所在的位置是[2,4,5]，翻转后应该在的位置是[1,2,4]，

那么原本位置2的3应该到翻转后的位置1，原本位置4 的3应该到翻转后的位置2，原本位置5的3应该到翻转后的位置4。

由此我们可以得到最后翻转后的数组，每个数在原数组中所在位置的值，所构成的数组应当为[2,4,3,5,1]。意思是翻转后的数组[3,3,4,3,1]中，第1个数字3在原数组中所在位置是2，第2个数字3在原数组中所在位置是4，第3个数字4在原数组中所在位置是3，第4个数字3在原数组中所在位置是5，第5个数字1在原数组中所在位置是1。

而数组[2,4,3,5,1]的逆序对数，就是我们要执行的最少操作次数。

AC代码:  
#include<bits/stdc++.h>

#define ll long long

#define llINF 9223372036854775807

#define IOS ios::sync\_with\_stdio(0); cin.tie(0); cout.tie(0);

**using** **namespace** std;

**const** ll maxn=2e5+7;

vector<**int**>origin[26];

**int** revese[maxn];//reverse[i]记录翻转后的第i个位置，在原来的字符串中是第几个位置

**int** n;

string s;

**int** tree[maxn];

**void** add(**int** x,**int** v){

**for**(;x<=n;x+=x&-x) tree[x]+=v;

}

**int** sum(**int** x){

**int** ret=0;

**for**(;x>0;x-=x&-x) ret+=tree[x];

**return** ret;

}

**int** main(){

    IOS;

    cin>>n>>s;

**for**(**int** i=0;i<n;i++) origin[s[i]-'a'].push\_back(i);//26个字母记录从左往右的下标

**for**(**int** i=0;i<26;i++){

**int** len=origin[i].size();

**for**(**int** j=0;j<len;j++) revese[n-origin[i][len-j-1]]=origin[i][j]+1;//同一个字母，按照翻转后所在的位置，从左往右依次贪心放进去

        //由于要用树状数组，所以这里让下标变成从1开始

    }

    ll ans=0;

**for**(**int** i=1;i<=n;i++){//利用树状数组求个逆序对，就是冒泡从原序列变成当前序列所需的最少操作次数了

        ans+=i-1-sum(revese[i]);

        add(revese[i],1);

    }

    cout<<ans<<endl;

}

H题:

首先要读懂题意。

有两个数组a[]和b[]，长度均为n，我们要给出n个操作顺序，其中1-n要都恰好操作一次。

ans初始时候为0。

每次操作的时候，如果选定了i，那么ans+=ai，且如果b[i]!=-1则a[b[i]]+=a[i]。

实际上我们可以根据b[]数组，把b[]数组的信息看成一条边，b[i]代表a数组中一条从位置i指向位置b[i]的有向边。

我们希望最后的ans尽可能大，那么对于正数的a[i]来说，拓扑序越靠前的，我们对他们优先操作会使得后续的值更大，因此a[i]是正数的i，按照拓扑序输出。

而对于负数的a[i]来说，拓扑序越靠前的，我们对他们优先操作会使得后续的值更小，因此a[i]是负数的i，按照拓扑序逆序输出。

AC代码:

#include<bits/stdc++.h>

#define ll long long

#define IOS ios::sync\_with\_stdio(0); cin.tie(0); cout.tie(0);

**using** **namespace** std;

**const** ll maxn=2e5+7;

ll n;

ll a[maxn],b[maxn];

ll ru[maxn];//ru[i]保存结点i的入度

**int** main(){

    IOS;

    cin>>n;

**for**(ll i=1;i<=n;i++) cin>>a[i];

**for**(ll i=1;i<=n;i++){

        cin>>b[i];

**if**(b[i]!=-1) ru[b[i]]++;

    }

    ll ans=0;

    queue<ll>Q;

    vector<ll>ans1,ans2;

**for**(ll i=1;i<=n;i++)

**if**(!ru[i]) Q.push(i);

**while**(Q.size()){///拓扑排序过程

        ll now=Q.front();

        Q.pop();

        ll to=b[now];

**if**(to!=-1){

            ru[to]--;

**if**(!ru[to]) Q.push(to);

**if**(a[now]>0) a[to]+=a[now];//如果当前结点的值是正数，那么加到下一个对应结点的值上

        }

        ans+=a[now];//入度为0的结点的值加到ans上

**if**(a[now]>=0) ans1.push\_back(now);

**else** ans2.push\_back(now);

    }

    cout<<ans<<endl;

**for**(ll i=0;i<ans1.size();i++) cout<<ans1[i]<<' ';//正数结点的下标按照拓扑序输出

    ll len=ans2.size();

**for**(ll i=0;i<ans2.size();i++) cout<<ans2[len-i-1]<<' ';//负数结点的下标按照拓扑序的逆序输出

    cout<<endl;

}