问题 A：简单题 暴力在主串中依次查找字母序列：heidi

```
#include<bits/stdc++.h>
using namespace std;
char str[1100];
queue<char>q;
int main(){
	freopen("delstr.in","r",stdin);
	freopen("delstr.out","w",stdout);
	q.push('h');q.push('e');q.push('i');
	q.push('d'),q.push('i');
	cin>>str;
	int len=strlen(str);
	for(int i=0;i<len;i++)
		if(!q.empty())
			if(str[i]==q.front()) q.pop();
	if(q.empty()) puts("YES");
	else puts("NO");
	return 0;
}
```

问题 B：递推

依次构造 1,11,111,1111,…2,22,222,2222,…

每一串直到>n 停止，统计合法个数。

```
#include<bits/stdc++.h>
using namespace std;
int main(){
	freopen("nums.in","r",stdin);
	freopen("nums.out","w",stdout);
	int T,n;
	cin>>T;
	while(T--){
		scanf("%d",&n);
		int res=0;
		for(int i=1;i<=9;i++){
			long long t=i;
			while(n>=t){
				res++;
				t=t*10+i;
			}
		}
		printf("%d\n",res);
	}
	return 0;
}
```

问题 C：搜索

n 个点,m 条边，每个个都至多有一个入水口和一个出水口，要求找到所有的水塔以及路径中最短的边。

```
#include<bits/stdc++.h>
using namespace std;
const int N=1010;
struct Node{
    int u,v,d;
}ans[N];
int cnt,n,m;
int ne[N],cost[N],pre[N];
bool st[N];
void dfs(int u,int flow){
    st[u]=true;
    if(ne[u]==0){
        ans[cnt].d=flow;
        ans[cnt].v=u;
        return;
    }
    int j=ne[u];
    if(cost[j]<flow&&cost[j]!=0)
        dfs(j,cost[j]);
    else dfs(j,flow);
}
int main(){
    freopen("water.in","r",stdin);
    freopen("water.out","w",stdout);
    cin>>n>>m;
    for(int i=1;i<=m;i++){
        int a,b,k;
        cin>>a>>b>>k;
        ne[a]=b;
        pre[b]=a;
        cost[a]=k;
    }
    for(int i=1;i<=n;i++)
        if(!st[i]&&pre[i]==0&&ne[i]!=0){
            ans[cnt].u=i;
            dfs(i,cost[i]);
            cnt++;
        }
    cout<<cnt<<endl;
    for(int i=0;i<cnt;i++)
       cout<<ans[i].u<<" "<<ans[i].v<<" "<<ans[i].d<<endl;
```

```
    return 0;
}
```

问题 D：BFS　贪心

因为奇数行和偶数行的朝向是固定的，所以即使该行已经没有草了，但是由于下一行需要你还是往前走，所以每行需要走的那一列就得由当前行和下一行的草来决定，还有如果当前行推完后已经没有草了，就没必要去下一行了，所有开个 sum 来记录还剩多少草

```
#include<bits/stdc++.h>
using namespace std;
const int N=160;
char mp[N][N];
int main(){
    freopen("grass.in","r",stdin);
    freopen("grass.out","w",stdout);
    int n,m;
    scanf("%d%d",&n,&m);
    int sum=0;
    for(int i=1;i<=n;i++){
        scanf("%s",mp[i]+1);
        for(int j=1;j<=m;j++)
            if(mp[i][j]=='W') sum++;
    }
    int ans=0,c=1;
    for(int i=1;i<=n;i++){
        if(i%2){
            int t=c,num=0;
            for(int j=c;j<=m;j++){
                if(mp[i][j]=='W') t=j,num++;
                if(i<n&&mp[i+1][j]=='W') t=j;
            }
            sum-=num;
            ans+=t-c+(sum==0?0:1);
            c=t;
        }
        else{
            int t=c,num=0;
            for(int j=c;j>=1;j--){
                if(mp[i][j]=='W') t=j,num++;
                if(i<n&&mp[i+1][j]=='W') t=j;
            }
            sum-=num;
            ans+=c-t+(sum==0?0:1);
            c=t;
        }
```

```
            if(!sum) break;
        }
        printf("%d\n",ans);
        return 0;
}
```

问题 E:网络供应

这题是比较有趣的二分。我们观察题目发现，如果我们固定了一个网络基站如何向两边分配，那其实所有的网络基站如何分配就已经决定了。比如我们加入 n 号网络基站向 1 号家庭分配了 x 的网络,那 1 号电站就需要给 1 号家庭分配 a[1]-x 的网络，向 2 号家庭分配 b[1]-(a[1]-x)的网络……

那我们发现，如果 n 号网络基站向 1 号家庭分配了 x 的网络，如果这个 x 很大，那可能跑到最后 n 号网络基站向 n 号家庭分配的网络就不够了，不能满足需求；但如果如果这个 x 太小了，这时 1 号网络基站就要尽量供应 1 号家庭，这时 2 号网络基站就要尽量供应 2 号家庭……也就是我们的网络基站趋向于向前供应，那就有跑到中间 i 号家庭的时候就无法满足。我们利用这个性质来二分我们的 x，如果对于当前 x 我们发现是 n 号家庭无法满足，那就是 x 太大了；如果是中间 i 号家庭没法满足，那就是 x 太小了。这样二分加上判断就能完成这道题。

```
#include <bits/stdc++.h>
using namespace std;
const int N=1100000;
int t,n,flag=0;
long long a[N],b[N],aa[N],bb[N];
long long l,r,mid;
int pd(long long x){
```

```
for(int i=1;i<=n;i++){
    aa[i]=a[i];
    bb[i]=b[i];
}
long long temp;
bb[0]=bb[n]-x;
for(int i=1;i<=n; i++) {
    aa[i]-=bb[i-1];
    if(aa[i]<0) continue;
    if(i==n){
        if(aa[i]>x) return(1);
        else return(0);
    }
    else{
        if(aa[i]>bb[i]) return(2);
        else{
            bb[i]-=aa[i];
            aa[i]=0;
        }
    }
}
return 0;
```

```
}
int main() {
    freopen("network.in","r",stdin);
    freopen("network.out","w",stdout);
    scanf("%d",&t);
    while(t--){
        scanf("%d",&n);
        flag=0;
        for(int i=1;i<=n;i++) scanf("%lld",&a[i]);
        for(int i=1;i<=n;i++) scanf("%lld",&b[i]);
        l=0;r=b[n];
        while(l<r){
            mid=(l+r+1)/2;
            int temp=pd(mid);
            if(temp==0){
                flag=1;
                break;
            }
            else if(temp==1) l=mid;
            else r=mid-1;
        }
        if(pd(l)==0) flag=1;
```

```
        if(flag) printf("YES\n");
        else printf("NO\n");
    }
    return 0;
}
```

问题 F：二色线段

题目给了我们黑白两种线段，我们把它们变成点，黑色的点放左边白色的点放右边，然后能产生不好的黑白线段对连一条边，这样我们就得到了一张二分图，我们的题目就是要求一个最大的独立集。

然后根据图论知识我们知道，最大独立集 = 点数 - 最大匹配 ，要求解最大独立集本质上就是要去求最大匹配，这时我们一看数据范围（1<=n<=2*10^5），不要说跑匈牙利了光建图时间就受不住了。所以我们需要挖掘一下这个二分图的性质。

我们首先先按每个线段的右端点为第一关键字排序，搞两个 multiset ，一个记录白线段，一个记录黑线段，用来记录我们选择了哪些区间。我们要求的是最大匹配，不失一般性我们考虑现在插入一条白色的线段，如果我们在黑色的 multiset 中，不存在与我们现在考虑的这个线段形成“不好线段对”的线段的话，我们就直接把这个白色线段的右端点加入到白色的 multiset 中，代表展示没有匹配。反之假若存在与我们现在考虑的这个白色线段形成“不好线段”的线段的话，我们就选择这两个点之间的这条边作为最大匹配中的一条边，然后再黑色 multiset 中把那个黑色线段所代表的点给抹掉，并且现在的这个区间也不加入到

白色 multiset 中，因为我们这两个点已经完成匹配了，不能再出现在最大匹配的边里了，这是最大匹配的定义要求的。下面我们考虑，如果我们的这条白色线段能与多条黑色线段匹配的话，我们应该匹配哪条呢？这很简单，贪心的想我们应该匹配右端点最小的那个，也就是右端点大于等于白色线段左端点的第一条黑色线段。这是因为我们尽可能消耗右端点小的线段，才能把更多右端点大的也就是更有机会在之后匹配的线段留给后面，这样才能产生最大匹配！对于插入黑色线段也同理。

```
#include <bits/stdc++.h>
using namespace std;
const int N=220000;
struct node{
    int l,r,color;
    //color=1->white
    //color=2->black
}a[N];
int n,num,ans;

multiset<int>white,black;
bool comp(node x,node y){
    if(x.r<y.r) return(true);
    if(x.r==y.r&&x.l<y.l) return(true);
    return(false);
```

```
}
int main(){
    freopen("bicolored.in","r",stdin);
    freopen("bicolored.out","w",stdout);
    scanf("%d",&n);
    for(int i=1;i<=n;i++)
        scanf("%d%d%d",&a[i].l,&a[i].r,&a[i].color);
    sort(a+1,a+n+1,comp);

    num=0;
    for(int i=1;i<=n;i++){
        if(a[i].color==1){
            multiset<int>::iterator it=black.lower_bound(a[i].l);
            if(it!=black.end()){
                num++;
                black.erase(it);
            }
            else white.insert(a[i].r);
        }
        else{
            multiset<int>::iterator it=white.lower_bound(a[i].l);
            if(it!=white.end()){
```

```
                num++;
                white.erase(it);
            }
            else black.insert(a[i].r);
        }
    }
    ans=n-num;
    printf("%d",ans);
    return 0;
}
```