二 分

1、二分查找

```
#include<bits/stdc++.h>//1-1236-1    二分查找    wintereta
using namespace std;
int a[1000005];
int n,x;
```

```
int bs(int x)
{
	int left=1;
	int right=n;
	while(left<=right)
	{
		int mid=(left+right)/2;
		if(a[mid]==x)
		{
			return mid;
		}
		if(a[mid]>x)
		{
			right=mid-1;
		}
		if(a[mid]<x)
		{
			left=mid+1;
		}
	}
	return -1;
}
int main()
{
	cin>>n;
	for(int i=1;i<=n;i++)
	{
		cin>>a[i];
	}
	cin>>x;
	cout<<bs(x);
	return 0;
}
```

二分循环

```
#include <bits/stdc++.h>//1-1236-2    二分查找      R_S
using namespace std;
// 声明数组
int a[1000005];
int n,x;
int main()
{
	// 完成输入过程
	cin >> n;
	for(int i=0;i<n;i++)
	{
		cin >> a[i];
	}
	cin >> x;
	// 二分查找
	int left,right,mid;
	left = 0;
	right = n-1;
	mid = (left+right)/2;
	// 什么情况下查找呢？
	//left <= right
```

```
	//什么情况下查找呢？
	//left <= right
	while(left <= right)
	{
		if(a[mid]==x)
		{
			cout << mid+1;
			return 0;
		}
		else if(a[mid]>x)
		{
			//证明要查找的数在左半边
			right = mid-1;
		}
		else if(a[mid]<x)
		{
			//证明要查找的数在右半边
			left = mid+1;
		}
		//更新中心点
		mid = (left+right)/2;
	}
	cout << -1;
	return 0;
}
```

二分递归

```
#include <bits/stdc++.h>//1-1236-3    二分查找     R_S
using namespace std;
// 声明数组
int a[1000005];
int n,x;

void fun(int l,int r)
{
	int mid = (l+r)/2;
	// 递归出口
	if(l>r)
	{
		cout << -1;
		return;
	}
	// 比较
	if(a[mid] == x)
	{
		cout << mid+1;
		return;
	}
	else if(a[mid]>x)
	{
		fun(l,mid-1);
	}
	else if(a[mid]<x)
	{
		fun(mid+1,r);
	}
}
```

```
int main()
{
	//完成输入过程
	cin >> n;
	for(int i=0;i<n;i++)
	{
		cin >> a[i];
	}
	cin >> x;
	//递归二分
	fun(0,n-1);

	return 0;
}
```

```
// 二分，递归。
#include<bits/stdc++.h>//2-1894     二分查找左侧边界     zhangjin
using namespace std;
int a[1000001],n,x;
int Find(int low,int high)
{
	if(a[low]>x||a[high]<x||low>high)
		return -1;

	int mid=(low+high)/2;
	if(a[mid]==x)
	{
		if(low==high) // 虽然找到 a[mid]=x，但是它不一定是首次出现的，还要继续找
			return mid;
		else
			return Find(low,mid);
	}
	else if(a[mid]<x)
		return Find(mid+1,high);
	else
		return Find(low,mid-1);
}
```

```
int main()
{
	int i,q;
	scanf("%d",&n);
	for(i=1;i<=n;i++)
		scanf("%d",&a[i]);

	scanf("%d",&q);

	while(q--)
	{
		scanf("%d",&x);
		printf("%d",Find(1,n));
	}


	return 0;
}
```

```
// 二分，非递归
#include<bits/stdc++.h>//3-1895-1    二分查找右侧边界    zhangjin
using namespace std;
int a[1000001],n,x;
int main()
{
	int i,q;
	scanf("%d",&n);
	for(i=1;i<=n;i++)
		scanf("%d",&a[i]);

	scanf("%d",&q);
	int low,mid,high;
```

```
	while(q--)
	{
		scanf("%d",&x);

		low = 1;
		high = n;
		while(low+1<high&&a[low]<=x&&a[high]>=x)  //请特别留意这个地方，意思是 low 和 high 之间还有 1 个数
		{
			mid=(low+high)/2;
			if(a[mid]==x)
				low =mid;
			else if(a[mid]<x)
				low = mid+1;
			else
				high = mid-1;
		}

		//还剩下 low 和 high 两个数，分别枚举
		if(a[high]==x)
			printf("%d",high);
		else if(a[low]==x)
			printf("%d",low);
		else
			printf("-1 ");

	}
	return 0;
}
```

```
// 二分查找右侧边界
#include<bits/stdc++.h>//3-1895-2  二分查找右侧边界    a623483487
using namespace std;
int a[100005],n,m;
int Search(int a[],int n,int key)
{

	int low = 1;
	int high = n;
	while(low <= high)
	{
		int mid =(high + low) / 2;
		if(key < a[mid])
		{
			high = mid - 1;
		}
		else
		{
			low = mid + 1;
		}
	}
	if(low <= n + 1 && key == a[low - 1])
	{
		return low - 1;
	}
	else
	{
		return -1;
	}
}
```

```
int main()
{

	cin >> n;
	for(int i = 1;i <= n;i ++)
	{
		cin >> a[i];
	}
	cin >> m;
	for(int i = 1;i <= m;i ++)
	{
		int x;
		cin >> x;
		cout << Search(a,n,x) << " ";
	}

	return 0;
}
```

```cpp
#include<bits/stdc++.h>//4-1896    二分查找满足条件的数    a623483487
using namespace std;
int a[100005],n,m;
int Search(int a[],int n,int key)
{

	int low = 1;
	int high = n;
	while(low <= high)
	{
		int mid =(high + low) / 2;
		if(key <= a[mid])
		{
			high = mid - 1;
		}
		else
		{
			low = mid + 1;
		}
	}
	if(low <= n)
	{
		return low;
	}
	else
	{
		return -1;
	}
}
```

```
int main()
{

	cin >> n;
	for(int i = 1;i <= n;i ++)
	{
		cin >> a[i];
	}
	cin >> m;
	for(int i = 1;i <= m;i ++)
	{
		int x;
		cin >> x;
		cout << Search(a,n,x) << " ";
	}

	return 0;
}
```

```
// 按题意找左右边界。
#include<bits/stdc++.h>//5-2078      起止位置    javacn
using namespace std;
const int N = 1e5 + 10;
int a[N];
int n,q,x;
// 找左侧边界
int fun1(int x)
{
      int l = 1,r = n,mid;
      while(l <= r)
      {
        mid = l + r >> 1;
        if(x <= a[mid]) r = mid - 1;
        else l = mid + 1;
      }

      if(l>=1&&l<=n&&a[l] == x) return l;
      else return -1;
}

// 找右侧边界
```

```
//找右侧边界
int fun2(int x)
{
	int l = 1,r = n,mid;
	while(l <= r)
	{
		mid = l + r >> 1;
		if(x < a[mid]) r = mid - 1;
		else l = mid + 1;
	}

	if(l-1>=1&&l-1<=n&&a[l-1] == x) return l-1;
	else return -1;
}

int main()
{
	scanf("%d%d",&n,&q);
    //读入 n 个问题
	for(int i = 1;i <= n;i++)
	{
		scanf("%d",&a[i]);
	}

	while(q--)
	{
		scanf("%d",&x);
		printf("%d %d\n",fun1(x),fun2(x));
	}
	return 0;
}
```

```
// 使用二分查找实现：
#include <bits/stdc++.h>//6-1898    同时出现的数      javacn
using namespace std;

/*
有 10^5 个数如果顺序查找，每个数最多找 10^5 次能找到
那么循环总次数：10^10

如果二分找，大概找 17 次
总循环次数：17 * 10^6

思路：
将第二组数排序，能够实现从小到大找
将第一组数排序，能够实现二分查找

逐个判断第 2 组数的每个数在第 1 组数中是否出现过
*/
int a[100010],b[100010];
int n,m;//n 代表 a 数组元素个数，m 代表 b 数组元素个数

// 判断 x 在 a 数组中是否存在
bool findx(int x)
{
	int l = 1,r = n,mid;
	while(l <= r)
	{
		mid = (l + r) / 2;
		if(x > a[mid]) l = mid + 1;
		else if(x < a[mid]) r = mid - 1;
		else return true;// 找到
	}

	return false;// 找不到
}
```

```
int main()
{
	cin>>n>>m;
	for(int i = 1;i <= n;i++)
	{
		cin>>a[i];
	}

	for(int i = 1;i <= m;i++)
	{
		cin>>b[i];
	}

	sort(a+1,a+n+1);
	sort(b+1,b+m+1);

	// 逐个判断 b 数组的每个数在 a 数组中是否出现过
	for(int i = 1;i <= m;i++)
	{
		if(findx(b[i]) == true)
		{
			cout<<b[i]<<" ";
		}
	}
	return 0;
}
```

```
// 二分，查找最后一个小于等于 bi，和第一个大于等于 bi 的数
#include<bits/stdc++.h>//7-1899    最满意的方案    zhangjin
using namespace std;
int a[1000001],b,m,n;
int Find1(int low,int high,int x)
{
	// 找最后一个小于等于 x 的数
	if(low+1==high||low==high)
	{
		if(a[high]<=x)
			return a[high];
		else
			return a[low];
	}

	int mid=(low+high)/2;
	if(a[mid]<=x)
		return Find1(mid,high,x);
	else
		return Find1(low,mid-1,x);
}
```

```
int Find2(int low,int high,int x)
{
	//找第一个大于等于 x 的数
	if(low+1==high||low==high)
	{
		if(a[low]>=x)
			return a[low];
		else
			return a[high];
	}

	int mid=(low+high)/2;
	if(a[mid]>=x)
		return Find2(low,mid,x);
	else
		return Find2(mid+1,high,x);
}
```

```
int main()
{
	int i,ans=0,t1,t2,MIN;
	scanf("%d%d",&m,&n);
	for(i=1;i<=m;i++) //学校的预计分数
		scanf("%d",&a[i]);
	sort(a+1,a+1+m);

	for(i=1;i<=n;i++) //学生的估计分数
	{
		scanf("%d",&b);
		if(a[1]>=b)
			ans += (a[1]-b);
		else if(b>a[m])
			ans += (b-a[m]);
		else
		{
			//来到这里，a数组里面肯定存在一个数是小于等于 b，又肯定存在一个数是大于等于b的
			t1 = Find1(1,m,b); //最后一个小于等于 b 的数
			t2 = Find2(1,m,b); //第一个大于等于b的数

			MIN = min(b-t1,t2-b);
			ans += MIN;
		}
	}

	printf("%d",ans);

	return 0;
}
```

```cpp
// 二分查找
// 对分数排序，再二分查找每个分数的排名。

#include <bits/stdc++.h>//8-1542-1      小 X 算排名    javacn
using namespace std;
const int N = 1e5 + 10;
int a[N],b[N];
int n;

bool cmp(int x,int y)
{
	return x > y;
}

int main()
{
	scanf("%d",&n);
	for(int i = 1;i <= n;i++)
	{
		scanf("%d",&a[i]);
		b[i] = a[i];
	}

	sort(b+1,b+n+1,cmp);

	// 找 a 数组每个数第一次出现的位置
	for(int i = 1;i <= n;i++)
	{
		int p = lower_bound(b+1,b+n+1,a[i],cmp) - b;// 获取下标
		printf("%d\n",p);
	}

	return 0;
}
```

// 统计 >= 每个分数的人数，从而统计每个分数的排名。

由于本题中每个分数的值都≤ 100000，因此可以先统计每个分数出现的次数，从而统计出 >= 每个分数的人数，从而计算排名。

```
#include <bits/stdc++.h>//8-1542-2      小 X 算排名    javacn
using namespace std;
95  96  97  98  99  100
每个分数的人数：
 1    1    0     0   2      1
>= 每个分数的人数：
 5    4    3      3    3    1
96 分的排名 =  ≥ 97 分的总人数 + 1
const int N = 1e5 + 10;
int a[N],b[N];
int n;
int main()
{
	scanf("%d",&n);
	for(int i = 1;i <= n;i++)
	{
		scanf("%d",&a[i]);
		b[a[i]]++;// 求每个分数的人数
	}
	// 求 >= 每个分数的总人数，也就是从右向左求前缀和
	for(int i = 100000;i >= 1;i--)
	{
		b[i] = b[i+1] + b[i];//>=i 分的人数 = i 分的人数 + ≥ i+1 分的人数
	}

	// 求每个分数的排名
	for(int i = 1;i <= n;i++)
	{
		printf("%d\n",b[a[i]+1] + 1);
	}
	return 0;
}
```

```cpp
//map 求解：
#include <bits/stdc++.h>//8-1542-3      小 X 算排名     javacn
using namespace std;
// 存放分数（key）以及分数出现的次数（value）
map<int,int,greater<int> > m;
int n,i,c = 0,a[100100],t;
int main()
{
	cin>>n;
	for(i = 0;i < n;i++)
	{
		cin>>a[i];
		// 判断该分数在 map 中是否存在
		if(m.count(a[i]) == 0)
		{
			m[a[i]] = 1;    // 将分数存入 map，记录值为 1（分数出现的次数）
		}
		else
		{
			m[a[i]]++;
		}
	}
	map<int,int>::iterator it;     // 修正 map 的 value 为排名
	for(it = m.begin();it != m.end();it++)
	{
		t = it->second;
		m[it->first] = c + 1;// 修正排名
		c = c + t;// 统计每个分数之前有几个人
	}
	for(i = 0;i < n;i++)     // 输出每个分数的排名
	{
		cout<<m[a[i]]<<endl;
	}
    return 0;
}
```

```cpp
#include<bits/stdc++.h>//8-1542-4      小 X 算排名    w2016010182
using namespace std;
int n;
map<int,int> m;
int main()
{
	scanf("%d",&n);
	int a[n+5],b[n+5];
	for(int i=1;i<=n;i++)
	{
		scanf("%d",&a[i]);
		b[i]=a[i];
	}
	sort(b+1,b+n+1);
	for(int i=1;i<=n;i++)
	{
		m[b[i]]=n-i+1;
	}
	for(int i=1;i<=n;i++)
	{
		printf("%d\n",m[a[i]]);
	}
	return 0;
}
```

将原来的 dp 数组的存储以每个数结尾的 LIS 序列的长度，修改为存储上升子序列长度为 i 的上升子序列的最小末尾数值。

原理：LIS 长度如果已经确定，那么如果这种长度的子序列的结尾元素越小，后面可能续的元素会更多！

```
#include <bits/stdc++.h>//9-1893  最长上升子序列 LIS（2）    javacn
using namespace std;

//dp: 长度为 i 的 LIS 的最后一位最小值是多少
int a[100100],dp[100100];
int i,n,l,r,mid;
```

```
int main()
{
	scanf("%d",&n);			//读入
	for(i = 1; i <= n; i++)
	{
		scanf("%d",&a[i]);
	}
	dp[1] = a[1];				//边界
	int len = 1;//LIS 的长度

	for(i = 2; i <= n; i++)		//从第 2 个数开始求解
	{
		//如果 a[i] 比 dp 最后一位大，a[i] 直接续上去，增加 LIS 的长度
		if(a[i] > dp[len])
		{
			len++;
			dp[len] = a[i];
		}
		else
		{
	//二分查找到 dp 数组中第 1 个 >=a[i] 的元素下标，替换（dp 数组一定是递增的）
			l = 1;
			r = len;
			while(l <= r)
			{
				mid = (l + r) / 2;
				if(a[i] <= dp[mid]) r = mid - 1;
				else l = mid + 1;
			}
			dp[l] = a[i];//替换
		}
	}
	printf("%d",len);
}
```

我们可以用 dp[i][j] 来表示第一个串的前 i 位，第二个串的前 j 位的 LCS 的长度，那么递推出状态转移方程：

如果当前的 a [i] 和 b[j] 相同（即是有新的公共元素） 这说明该元素一定位于公共子序列中。因此，现在只需要找：a 数组 1~i-1 和 b 数组 1~j-1 的最长公共子序列：

dp[i][j]=max(dp[i][j],dp[i-1][j-1]+1);

如果不相同，说明最后一个元素肯定不是公共子序列中的元素，那么考虑找 a 数组 1~i-1 和 b 数组 1~j 的 LCS，或者找：a 数组的 1~i 和 b 数组的 1~j-1 的 LCS，那么，状态转义方程如下：

dp[i][j]=max(dp[i-1][j],dp[i][j-1]);

```
#include <bits/stdc++.h>//10-1821    最长公共子序列(LCS)(1)    javacn
using namespace std;

/*
a[i]==b[j], 方程：dp[i-1][j-1]+1
a[i]!=b[j], 方程：max(dp[i][j-1],dp[i-1][j])
*/
const int N = 1010;// 常量，表示数组大小
int a[N],b[N],dp[N][N];
int n,i,j;
```

```
int main()
{
	cin>>n;
	for(i = 1;i <= n;i++) cin>>a[i];
	for(i = 1;i <= n;i++) cin>>b[i];

	//递推
	for(i = 1;i <= n;i++)
	{
		for(j = 1;j <= n;j++)
		{
			if(a[i] == b[j]) dp[i][j] = dp[i-1][j-1] + 1;
			else dp[i][j] = max(dp[i-1][j],dp[i][j-1]);
		}
	}

	cout<<dp[n][n];
	return 0;
}
```

1、因为两个序列都是1~n的全排列，那么两个序列元素互异且相同，也就是说只是位置不同;

2、通过c数组将b序列的数字在a序列中的位置求出；

3、如果b序列每个元素在a序列中的位置递增，说明b中的这个数在a中的这个数整体位置偏后，可以考虑纳入LCS； 4、从而就可以转变成求用来记录新的位置的c数组中的LIS。

```
#include <bits/stdc++.h>//11-1822      最长公共子序列(LCS)(2)    javacn
using namespace std;
const int N = 100100;
int a[N],b[N],c[N],dp[N];
int n,i;
```

```
int main()
{
	cin>>n;
	for(i = 1; i <= n; i++)
	{
		scanf("%d",&a[i]);
		c[a[i]] = i;// 求出 a 数组的每个数的位置
	}
	for(i = 1; i <= n; i++) scanf("%d",&b[i]);
	dp[1] = c[b[1]];// 边界    // 求 b 数组的每个数在 a 数组的位置 (c[b[i]]) 的 LIS
	int len = 1;
	int l,r,mid;
	for(i = 2; i <= n; i++)        // 从第 2 个数开始讨论
	{
		if(c[b[i]] > dp[len])          // 增加 LIS 的长度
		{
			len++;
			dp[len] = c[b[i]];
		}
		else
		{
			l = 1;
			r = len;
			while(l <= r)
			{
				mid = (l + r) / 2;
				if(c[b[i]] <= dp[mid]) r = mid - 1;
				else l = mid + 1;
			}
			dp[l] = c[b[i]];
		}
	}
	cout<<len;
	return 0;
}
```

思路：求最少的修改次数， 那就是要找出需要修改的数字，而且越少也好。逆向思维， 找最长的上升子序列（LIS）。然后，用总个数减去上升的，即需要修改的数字。

```
#include<bits/stdc++.h>//12-1902    最少的修改次数  dragoncatter
using namespace std;
const int N = 1e5 + 10 ;
int n, cnt = 1;
int a[N], dp[N];
int main()
{
	cin >> n;
	for (int i = 1; i <= n; i++)
	{
		cin >> a[i];
	}
	dp[1] = a[1];
	for (int i = 2; i <= n; i++)
	{
		if (a[i] > dp[cnt])
		{
			dp[++cnt] = a[i];
		}
		else
		{
			int k = lower_bound(dp+1, dp + cnt+1, a[i], less<int>()) - dp;
			dp[k] = a[i];
		}
	}
	cout << n - cnt;
	return 0;
}
```

1、二分答案

二分答案模板题：

```
#include <bits/stdc++.h>//1-1908    伐木工      javacn
using namespace std;

/*
本题：读入的数据在 int 范围，但求和可能会超过 int
因此定义 long long。
思考：找左边界，还是右边界；
本题：在高度为 mid 的情况下，如果能得到 >=m 米的木材，升高高度
因此本题求右边界；
*/

const int N = 1000010;
long long a[N];
long long n,m,l = 1,r,mid;

// 检验锯片高度为 x 的情况下，能够得到的木材量是否 >=m
```

```
// 检验锯片高度为 x 的情况下，能够得到的木材量是否 >=m
bool check(long long x)
{
	long long s = 0;// 能够锯到的木材量
	for(int i = 1;i <= n;i++)
	{
		// 如果 x< 树的高度，才能锯到木材
		if(x < a[i]) s = s + a[i] - x;
		// 如果得到 >=m 米的木材，就可以停止返回
		if(s >= m) return true;
	}
	return false;
}
int main()
{
	cin>>n>>m;
	// 读入木材的高度
	for(int i = 1;i <= n;i++)
	{
		cin>>a[i];
		r = max(a[i],r);//r 的值应该是：所有树的最高高度
	}

	// 二分答案
	while(l <= r)
	{
		mid = l + r >> 1;
		// 如果高度为 mid 的情况下，能够得到 >=m 的木材
		if(check(mid)) l = mid + 1;
		else r = mid - 1;
	}

	cout<<l - 1;// 右边界
	return 0;
}
```

```cpp
#include <iostream>//2-1909  跳石头  kevinh
using namespace std;
int l,n,m,a[505000],i;

bool check(int mid)
{
	int p=0,c=0;
	for(int i=0;i<n;i++)
	{
		if(a[i]-p<mid)
		{
			c++;
		}
		else
		{
			p = a[i];
		}
	}
	if(l-p<mid) c++;
	return c<=m;
}
```

```
int main()
{
	cin>>l>>n>>m;
	for(i=0;i<n;i++)
	{
	  cin>>a[i];
	}

	int left=0,right=l,mid;
	while(left<=right)
	{
		mid = (left+right)/2;
		if(check(mid))
		{
			left = mid+1;
		}
		else
		{
			right = mid-1;
		}
	}
	cout<<left-1<<endl;
	return 0;
}
```

二维数组求解：

```
#include <bits/stdc++.h>//3-1561-1    买木头   javacn
using namespace std;

int main(){
	int n,m,l_1,s_1,i,j;
	 // 记录 n 个商家每个商家的木头的长度和数量
	int a[10010][2];
	int r;// 最长木头长度
	cin>>n>>m>>l_1>>s_1;
    // 第一个商家木头长度和数量
	a[1][1] = l_1;
	a[1][2] = s_1;
	int max = a[1][1];// 最长木头的长度

	for(i = 2;i <= n;i++)
	{
		a[i][1] = ((a[i - 1][1] * 37011 + 10193) % 10000) + 1;
		a[i][2] = ((a[i - 1][2] * 73011 + 24793) % 100) + 1;

		if(a[i][1] > max)
		{
			max = a[i][1];
		}
	}
```

```
    //输出每个商家的木头的长度和数量
//  for(i = 1;i <= n;i++){
//      cout<<a[i][1]<<"  "<<a[i][2]<<endl;
//  }

    //计算在每个长度下各个商家能供多少根木头
      int c;//在每个长度下，各个供应商能供多少根木头
      for(i = max;i >= 1;i--)
      {
            c = 0;
        //循环每个供应商
            for(j = 1;j <= n;j++)
            {
                  c += a[j][1] / i * a[j][2];
            }

        //如果根数够
            if(c >= m)
            {
                  r = i;
                  break;
            }
      }

      cout<<r<<endl;
      return 0;
}
```

二分求解：

```
#include <bits/stdc++.h>//3-1561-2    买木头   javacn
using namespace std;
const int N = 10010;
int len[N],cnt[N];
int n,m;

// 检验切割长度为 mid，能否切出 m 根以上的木头
bool check(int mid)
{
	int c = 0;
	for(int i = 1;i <= n;i++)
	{
		c += len[i]/mid*cnt[i];
	}
	return c >= m;
}
```

```cpp
int main()
{
	cin>>n>>m>>len[1]>>cnt[1];

	int l = 1,r = len[1],mid;
	//递推出每个供货商的木头长度和数量
	for(int i = 2;i <= n;i++)
	{
		len[i]=((len[i-1]*37011+10193)%10000)+1;
		cnt[i]=((cnt[i-1]*73011+24793)%100)+1;
		r = max(len[i],r);
	}

	//二分可能的长度
	while(l <= r)
	{
		mid = l + r >> 1;
		//如果长度为 mid 能切出 >=m 个木头
		//放长
		if(check(mid)) l = mid + 1;
		else r = mid - 1;
	}

	cout<<l-1;
	return 0;
}
```

二分两头牛最大距离的最小值，如果两头牛的最大距离为 mid，能够容纳 >=c 头牛，说明检验成功，将答案放大一点再试试。否则，将答案缩小。

```
#include<bits/stdc++.h>//4-1910-1  愤怒的奶牛    javacn
using namespace std;
const int N = 1e5 + 10;
int a[N],l,r,mid,n,c;

// 测试距离为 mid，可以放几头牛
bool check(int mid)
{
	int tot = 1;// 默认能放 1 头
	//last: 表示上 1 个牛栏
	int last = 1;// 第 1 个牛栏肯定有牛
	for(int i = 2; i <= n; i++)
	{
		if(a[i] - a[last] >= mid)
		{
			tot++;// 牛栏数自增
			last = i;
		}
	}

	if(tot >= c) return true;
	else return false;
}
```

```
int main()
{
	//n 个牛栏，c 头要分隔的奶牛
	cin>>n>>c;
	for(int i = 1; i <= n; i++)
	{
		cin>>a[i];
	}
	// 排序
	sort(a+1,a+1+n);
	// 在 [l,r] 之间查找最大的最近距离
	l=1,r=a[n]-a[1];
	// 找右边界
	while(l <= r)
	{
		mid = (l + r) >> 1;
		if(check(mid)) l = mid + 1;
		else r = mid - 1;
	}

	cout<<l-1;
	return 0;
}
```

```cpp
#include<bits/stdc++.h>//4-1910-2    hasome
using namespace std;
int n,c,a[100010],l,r=INT_MIN,mid;
bool check(int x)
{
	int sum=c-1;
	int j=2;
	int i=1;
	while(sum>0)
	{
		while(a[j]-a[i]<x)
		{
			j++;
			if(j==n+1) return false;
		}
		i=j;
		sum--;
	}

	return true;
}
```

```
int main()
{
	cin>>n>>c;
	for(int i=1;i<=n;i++)
	{
		cin>>a[i];
		r=max(a[i],r);
	}
	sort(a+1,a+n+1);
	l=1;
	r=r-1;
	while(l<=r)
	{
		mid=l+r>>1;
		if(check(mid))
		{
			l=mid+1;
		}
		else
		{
			r=mid-1;
		}
	}
	cout<<l-1<<endl;
	return 0;
}
```

二分答案，检验如果最小的空旷指数为 x 的情况下，路标数量是否 <= 规定的 k 。

```
#include<bits/stdc++.h>//5-1912      最小的空旷指数      javacn
using namespace std;
int l,n,k;
int a[100010];
// 检验如果最小的空旷指数为 x 的情况下
// 路标数量是否 <= 规定的 k
bool check(int x)
{
	int cnt=0;
	for(int i=2; i<=n; i++)
	{
		int d=a[i]-a[i-1];
		if(d>x)
		{
			cnt+=d/x;
			if(d % x == 0) cnt--;
		}
	}
	return cnt<=k;
}
```

```
int main()
{
	scanf("%d%d%d",&l,&n,&k);
	for(int i=1; i<=n; i++)
	{
		scanf("%d",&a[i]);
	}
	//读入无序，排序
	sort(a+1,a+1+n);
	//二分答案，如果 left、right 定义在 main 的外面
	//记得不能用 left、right 这样的函数名
	int left=1,right=l;
	while(left<=right)
	{
		int mid=(left+right)>>1;
		if(check(mid)) right=mid-1;
		else left=mid+1;
	}
	//输出左边界
	cout<<left;
}
```

二分可能的伤害值，如果伤害值为 mid 的情况下，能够通过迷阵，则降低伤害值继续尝试。

判断伤害值为 mid 的情况下，能否通过迷阵的方法是：广搜，如果某个点的伤害值值<=mid，则认为该点可行。

参考解法：

```
#include <bits/stdc++.h>//6-1916    防御迷阵    lavacn
using namespace std;

/*
第 1 行是入口，第 n 行是出口
伤害值：经过所有点的伤害值的最大
求：最小的伤害代价（求最大值的小的问题）

二分答案：
1. 答案具备单调性——可以二分
2. 思考二分的范围：答案在什么范围内
3. 思考左边界、右边界
*/

int a[1010][1010];
int n,m;
int l = INT_MAX,r = INT_MIN,mid;
// 广搜变量准备
int q[1000100][3];
int fx[5] = {0,0,1,0,-1};
int fy[5] = {0,1,0,-1,0};
bool f[1010][1010];// 标记是否走过

//check(mid)：判断伤害值为 mid 的情况下，能否通过迷阵
// 检验方法：采用广搜，从 1,1 出发，走 a[i][j]<=mid 的点，看能否走到第 n 行
```

```
//check(mid)：判断伤害值为 mid 的情况下，能否通过迷阵
// 检验方法：采用广搜，从 1,1 出发，走 a[i][j]<=mid 的点，看能否走到第 n 行
bool check(int v)
{
	// 重设标记数组的值，因为要广搜多次
	memset(f,false,sizeof(f));
	int h = 1,t = 1;// 指针
	// 交代起始点
	q[1][1] = 1;
	q[1][2] = 1;
	f[1][1] = true;// 标记走过
	int tx,ty;// 要去的点
	while(h <= t)
	{
		// 尝试四方向
		for(int i = 1;i <= 4;i++)
		{
			tx = q[h][1] + fx[i];
			ty = q[h][2] + fy[i];
			// 如果该点可行
			//! 表示取反
			if(tx>=1&&tx<=n&&ty>=1&&ty<=m&&!f[tx][ty]&&a[tx][ty]<=v)
			{
				t++;
				q[t][1] = tx;
				q[t][2] = ty;
				f[tx][ty] = true;// 走过标记
				// 判断是否出迷阵
				if(tx == n) return true;
			}
		}
		h++;
	}
	return false;
}
```

```
int main()
{
	cin>>n>>m;
	for(int i = 1;i <= n;i++)
	{
		for(int j = 1;j <= m;j++)
		{
			cin>>a[i][j];
			// 如果不是第 1 行和最后一行，才是伤害值
			if(i != 1 && i != n)
			{
				l = min(l,a[i][j]);
				r = max(r,a[i][j]);
			}
		}
	}

	// 二分
	while(l <= r)
	{
		mid = l + r >> 1;
		// 如果伤害值为 mid，能通过，减少伤害值，再尝试
		if(check(mid)) r = mid - 1;
		else l = mid + 1;
	}

	cout<<l;// 左边界
	return 0;
}
```