# 差分前缀和

本讲主要了解前缀和数组和差分数组，统称为差分前缀和。

## 一维前缀和数组

前缀和数组是指，对于给定长度为 n 的数列 a[i]，我们设定数组 s，其中 s[i]=a[1]+a[2]+……+a[i]，即 s[i]表示前 i 项的和。

| 下标 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| A[] | 5 | 21 | 7 | 32 | 22 | 3 | 6 | 12 | 9 |
| S[] | 5 | 26 | 33 | 65 | 87 | 90 | 96 | 108 | 117 |

更新前缀和数组，利用递推式 s[i]=s[i-1]+a[i]更新即可。

```
#include<iostream>
using namespace std;
int n,a[10001],s[10001];
int main(){
    cin>>n;
    for(int i=1;i<=n;i++){
        cin>>a[i];
        s[i]=s[i-1]+a[i];
    }
}
```

如果需要要保存 a 数组，还可以更简洁，如下：

```
#include<iostream>
using namespace std;
int n,a[10001],s[10001];
int main(){
    cin>>n;
    for(int i=1;i<=n;i++){
        cin>>s[i];
        s[i]+=s[i-1];
    }
}
```

利用前缀和数组，我们可以很容易求出区间和。比如求区间[l,r]的和，我们可以直接用 s[r]-s[l-1]求得。

# 二维前缀和数组

前缀和数组可以扩展到二维数组上。设数组 s[i][j]表示 a[1][1]+……+a[1][j]+a[2][1]+……+a[2][j]+……+a[i][1]+……+a[i][j]的和。

| | | | |
|---|---|---|---|
| 5 | 3 | 1 | 5 |
| 9 | 8 | 6 | 1 |
| 2 | 9 | 8 | 7 |
| 3 | 4 | 5 | 6 |

A 数组

| | | | |
|---|---|---|---|
| 5 | 8 | 9 | 14 |
| 14 | 25 | 32 | 38 |
| 16 | 36 | 51 | 64 |
| 19 | 43 | 63 | 82 |

S 数组

S 数组的更新：S[i][j]=s[i-1][j]+s[i][j-1]-s[i-1][j-1]+a[i][j]，如下图:

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | | |
| | | | S[i-1][j-1] | S[i-1][j] | | |
| | | | S[i][j-1] | S[i][j] | | |
| | | | | | | |
| | | | | | | |

代码实现：

```
#include<iostream>
using namespace std
int n,m,a[1001][1001],s[1001][1001];
int main(){
    cin>>n>>m;
    for(int i=1;i<=n;i++){
        for(int j=1;j<=m;j++){
            cin>>a[i][j];
            s[i][j]=s[i-1][j]+s[i][j-1]-s[i-1][j-1]+a[i][j];
        }
    }
}
```

利用二维前缀和求子矩阵的和

已知坐标点 x1，y1 和 x2，y2，可以由二维前缀和数组直接求出由这两个坐标点构成的子矩阵的和。其和为 s[x2][y2]-s[x2][y1-1]-s[x1-1][y2]+s[x1-1][y1-1]，如下图所示：

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | | |
| | | S[x1-1][y1-1] | | | S[x1-1][y2] | |
| | | | S[x1][y1] | | | |
| | | | | | | |
| | | S[x2][y1-1] | | | S[x2][y2] | |

# 差分数组.

对于数组 c，对于长度为 n 的数列 a，我们令 c[i]=a[i]-a[i-1]，这里 c 数组就是差分数组。差分数组有一些优美的性质，比如，a[i]=c[1]+c[2]+……+c[i]
证明：由差分数组的定义得 c[1]=a[1]-a[0]，c[2]=a[2]-a[1]，……，c[i]=a[i]-a[i-1]，那么 c[1]+c[2]+……+c[i]=(a[1]-a[0])+(a[2]-a[1])+……+(a[i]-a[i-1])=a[i]-a[0]，因为 a[0]=0，所以 a[i]=c[1]+c[2]+……+c[i]。

利用差分数组进行区间修改单点询问

问题：给出一个长度为 n 的线段和 m 个操作，操作分两种，操作 1：将区间[s,t]的值加 1；操作 2：问点 x 的值为多少。

对于每个操作，对于差分数组，我们将 c[s]++,c[t+1]--，那么对于点 x 的值为 c[1]+c[2]+……+c[x]。

证明：
1、如果 x<s，a[x]=c[1]+c[2]+……+c[x]，c[s]++对询问没有影响。
2、如果 s<=x<=t，那么 a[x]=c[1]+c[2]+……+c[s]+……+c[x]，因为 c[s]加 1，所以 a[x]多加了 1。
3、如果 x>t，那么 a[x]=c[1]+c[2]+……+c[s]+……+c[t+1]+……+c[x]，虽然 c[s]多加了 1，但是 c[t+1]减去了 1，所有对最后查询结果没有影响。

# 拼不出的数

【题目描述】
3 个元素的集合 {5, 1, 2} 的所有子集的和分别是 0, 1, 2, 3, 5, 6, 7, 8。发现最小的不能由该集合子集拼出的数字是 4。
现在给你一个 n 个元素的集合，问你最小的不能由该集合子集拼出的数字是多少。
注意 32 位数字表示范围。
【输入】
第一行一个整数 n。
第二行 n 个正整数 ai，表示集合内的元素。
【输出】
一行，一个整数答案。
【样例输入】
3
5 1 2
【样例输出】
4
【提示】
对于 30% 的数据，满足 n ≤ 15。
对于 60% 的数据，满足 n ≤ 1000。

对于 100% 的数据，满足 n ≤ 100000, 1 ≤ ai ≤ 10^9。
【分析】
因为每个数只能用一次，所以前 i 个数所能组成的最大和为 a[1]+a[2]...+a[i]，若 a[i+1]>a[1]+a[2]...+a[i]+1，则因为比 a[1]+a[2]...+a[i]大的下一个数必>=a[i+1]，所以 a[1]+a[2]...+a[i]+1 为答案。
【代码】

```
#include<iostream>
#include<algorithm>
using namespace std;
int n,a[100001];
long long s[100001];
int main(){
    cin>>n;
    for(int i=1;i<=n;i++)cin>>a[i];
    sort(a+1,a+n+1);
    for(int i=1;i<=n;i++){
        if(s[i-1]+1<a[i]){
            cout<<s[i-1]+1;
            return 0;
        }
        s[i]=s[i-1]+a[i];
    }
    cout<<s[n]+1;
}
```

# 羊羊吃草

【题目描述】
小羊们上完课后，纷纷到草场上吃草。而羊村现在正在进行特色示范羊村检查，领导们想看看羊村的草场。
羊村的草场是连续分布的，每块草场上都有数量不等的羊在吃草。领导们想要查看连续若干个草场，但是又不想看到超过 T 只羊。而村长希望领导们多看看羊村的风貌，尽可能多参观几个草场。
现在，请你帮村长决定，带领导们去参观哪一段草场，满足领导和村长的要求。
【输入】
第一行一个整数 N 和 T，表示羊村共有多少个连续草场，以及领导们希望看到羊数量的最大值。
第二行 N 个整数，两个整数间用一个空格分开，第 i 个数 ai 表示第 i 个草场上有 ai 只羊在吃草。编号从 1 到 N。
【输出】
输出一行，共两个数，表示参观的起点编号和终点编号，中间用空格分开。走的方向总是从编号小的到编号大的。另外，若有长度相同的可能性，输出起点编号较小的答案。数据保证至少有答案存在。

【样例输入】
5 10
6 3 2 1 7
【样例输出】
2 4
【样例解释】
要满足连续个总和不超过 10，有 3+2+1 和 2+1+7 两种可能性，优先输出 2 到 4 个草场。
【数据范围】
30%的数据，1<=N<=100;
60%的数据，1<=N<=1000;
100%的数据，1<=N<=100000, 0<=ai<=10^9，0<=T<=2^31-1。
【分析】
利用前缀和，可以 o(1)时间查询区间和。对于每个终点 i，查询起点 j，如果[j,i]的区间和在 t 以内，且长度更长，则更新答案。代码如下，时间复杂度 O（n^2）,超时。

```
#include<iostream>
using namespace std;
int n,t,x,s[100001],ans,l,r;
int main(){
    cin>>n>>t;
    for(int i=1;i<=n;i++){
        cin>>x;
        s[i]=s[i-1]+x;
        int j;
        for(j=1;j<=i;j++){
            if(s[i]-s[j-1]<=t)break;
        }
        if(ans<i-j+1){
            ans=i-j+1;
            l=j;
            r=i;
        }
    }
    cout<<l<<" "<<r<<endl;
}
```

优化的方法，对于当前区间[j,i]，已知其是区间和小于 t 的最大值，那么对于[j,i+1],不可能存在[j-1,i+1]小于 t，如果[j,i+1]依然小于 t，则比较其区间长度，如果[j,i+1]大于 t，则增大 j,使得[j',i+1]为以 i+1 为终点的最长的小于 t 的区间长度。时间复杂度为 O（2*n）。
【代码】

```
#include<iostream>
using namespace std;
int n,t,x,ans,l,r,j=1;
long long s[100001];
```

```
int main(){
    cin>>n>>t;
    for(int i=1;i<=n;i++){
        cin>>x;
        s[i]=s[i-1]+x;
        while(s[i]-s[j-1]>t)j++;
        if(ans<i-j+1){
            ans=i-j+1;
            l=j;
            r=i;
        }
    }
    cout<<l<<" "<<r<<endl;
}
```

# 家的范围

【题目描述】
农民约翰在一片边长是 N (2 <= N <= 250)英里的正方形牧场上放牧他的奶牛。(因为一些原因，他的奶牛只在正方形的牧场上吃草。)遗憾的是,他的奶牛已经毁坏一些土地。( 一些 1 平方英里的正方形)
农民约翰需要统计那些可以放牧奶牛的正方形牧场(至少是 2x2 的,在这些较大的正方形中没有一个点是被破坏的，也就是说，所有的点都是“1”)。
你的工作要在被供应的数据组里面统计所有不同的正方形放牧区域(>=2x2)的个数。当然，放牧区域可能是重叠。
【输入】
第 1 行:N,牧区的边长。
第 2 到 n+1 行:N 个没有空格分开的字符。
0 表示 "那一个区段被毁坏了";1 表示 " 准备好被吃"。
【输出】
输出那些存在的正方形的边长和个数，一种一行。
【样例输入】
6
101111
001111
111111
001111
101101
111001
【样例输出】
2 10
3 4
4 1
【分析】

【代码】

```
#include<iostream>
using namespace std;
long long n,ans,g[251][251],s[251][251];
char ch;
int main(){
    cin>>n;
    for(int i=1;i<=n;i++){
        for(int j=1;j<=n;j++){
            cin>>ch;
            if(ch=='1')g[i][j]=1;
            if(ch=='0')g[i][j]=-9999999;
            s[i][j]=s[i-1][j]+s[i][j-1]-s[i-1][j-1]+g[i][j];
        }
    }
    for(int k=2;k<=n;k++){
        ans=0;
        for(int i=k;i<=n;i++){
            for(int j=k;j<=n;j++){
                int x=i-k+1;
                int y=j-k+1;
                if(s[i][j]-s[i][y-1]-s[x-1][j]+s[x-1][y-1]>0)ans++;
            }
        }
        if(ans!=0)cout<<k<<" "<<ans<<endl;
    }
}
```

# 堆叠草堆

【题目描述】

Bessie 对她自己最近在农场周围的恶作剧感到抱歉，于是她同意帮助 Farmer John 堆叠新送达的一批干草捆。

开始时有 N(N 是奇数)个空草堆，标记为 1..N。FJ 将会给 Bessie 包含 K 条指令的序列，每条指令的格式为"A B"，表示 Bessie 应该在草堆 A..B 中每一个草堆顶部新放一捆干草。例如，如果 Bessie 听到指令"10 13"，那么她应该在草堆 10,11,12 和 13 上各新放一捆干草。

在 Bessie 完成了 FJ 的所有指令后，FJ 想要知道 N 个草堆高度的中位数——也就是说，所有草堆排序后中央草堆(由于 N 是奇数，这个草堆是唯一的)的高度。请帮助 Bessie 确定 FJ 问题的答案。

【输入】

第 1 行：两个被空格分隔的整数 N 和 K。

第 2..1+K 行：每行包含一条 FJ 的指令，其格式为两个被空格分隔的整数 A 和 B。

【输出】
输出共一行一个整数， Bessie 完成所有指令后草堆高度的中位数。
【样例输入】
7 4
5 5
2 4
4 6
3 5
【样例输出】
1
【输入说明】有 N=7 个草堆，FJ 给出了 K=4 条指令。第一条指令要求增加草堆 5 的干草捆，第二条指令要求增加草堆 2..4 的干草捆，等等。
【输出说明】Bessie 完成任务后，草堆的高度为 0,1,2,3,3,1,0。高度的中位数是 1，因为 1 是排序后结果 0,0,1,1,2,3,3 的中间数。
【数据规模】
对于 20%的数据：1≤N(N 为奇数)≤100；1≤K≤100；
对于 40%的数据：1≤N(N 为奇数)≤1,000；1≤K≤5,000；
对于 60%的数据：1≤N(N 为奇数)≤50,000；1≤K≤10,000；
对于 100%的数据：1≤N(N 为奇数)≤1,000,000；1≤K≤25,000；1≤A≤B≤N；
【分析】

【代码】

```
#include<iostream>
#include<algorithm>
using namespace std;
int n,a[1000001],c[1000001],k,x,y;
int main(){
    cin>>n>>k;
    for(int i=1;i<=k;i++){
        cin>>x>>y;
        c[x]+=1;
        c[y+1]-=1;
    }
    for(int i=1;i<=n;i++){
        a[i]=a[i-1]+c[i];
    }
    sort(a+1,a+n+1);
    cout<<a[n/2+1];
}
```

# 车票

【题目描述】

考虑一条高铁线路 G39，这条线路依次经过 N 个火车站，编号为 1，2，3，……，n。
现在你得到了关于未来 K 天的 M 条订票信息，问你每天至少安排多少个座位可以满足这些订票信息。
每条订票信息形如 i j k，代表在第 i 天出行，从 j 火车站上车，从 k 火车站下车。
注意座位可以重复使用，但不能两个人坐一个座位，如果一条信息是 1 1 3，另一条信息是 1 3 5，那么只安排一个座位即可 (第一位乘客在 3 号火车站下车，同时第二位乘客在 3 号火车站上车)。
【输入】
第一行三个整数 N,M,K。
接下来 M 行，每行三个整数 i j k，意义见题目描述。
【输出】
K 行，每行一个整数，表示当天至少需要安排多少个座位。
【样例输入】
10 5 3
1 1 3
1 3 5
2 1 4
1 7 8
2 2 10
【样例输出】
1
2
0
【提示】
对于 30% 的数据， M ≤ 100, N ≤ 100。
对于 60% 的数据， M ≤ 3000, N ≤ 3000。
对于 100% 的数据， 1 ≤ K ≤ 50, 1 ≤ M ≤ 10^6,2 ≤ N ≤ 10^6, 满足 1 ≤ i ≤ K; 1 ≤ j < k ≤ N。
【分析】

【代码】

```cpp
#include<iostream>
using namespace std;
int n,m,k,a[51][100001],d,x,y;
int main(){
    cin>>n>>m>>k;
    for(int i=1;i<=m;i++){
        cin>>d>>x>>y;
        a[d][x]++;
        a[d][y]--;
    }
    for(int i=1;i<=k;i++){
        int ans=0,s=0;
        for(int j=1;j<=n;j++){
```

```
            s+=a[i][j];
            ans=max(ans,s);
        }
        cout<<ans<<endl;
    }
}
```

# 切割三角形

【题目描述】
贝克金宝有 N 个三角形。他的儿子乔伊有一个很大的刀，准备切割这些三角形。每一次切割，乔伊能够用与 x 轴平行的直线（y = c）或与 y 轴平行的直线（x = c）对这些三角形进行切割。
你的任务是计算每一次切割，有多少个三角形会被切割。
注意：
被切割过的三角形仍看作一个三角形；
三角形被切割当且仅当三角形在直线以左的部分的面积和在直线以右的部分的面积，都应大于 0。
【输入】
第一行是一个正整数 N（2<=N<=100000），三角形的个数。
第 2 到 N+1 行 每行包含 6 个整数 x1,y1,x2,y2,x3,y3 表示三角形的三个顶点(x1,y1)，(x2,y2)，(x3,y3)（保证这三个点不共线） 0<=x1,y1,x2,y2,x3,y3<=1000000
第 N+2 行是一个正整数 M（2<=M<=100000），切割的次数。
第 N+3 到 N+M+2 行 包含一个直线方程“x = c”或“y = c”（注意等号两旁的空格）
0<=C<=1000000。
【输出】
每一次切割，输出一行包含切割的三角形数量。
【样例输入】
Input 1
3
1 0 0 2 2 2
1 3 3 5 4 0
5 4 4 5 4 4
4
x = 4
x = 1
y = 3
y = 1
Input 2
4
2 7 6 0 0 5
7 1 7 10 11 11
5 10 2 9 6 8
1 9 10 10 4 1

4
y = 6
x = 2
x = 4
x = 9
【样例输出】
Output 1
0
1
1
2
Output 2
3
2
3
2
【数据范围】
至少 40%的数据 M<=300
另外 40%的数据 三角形顶点的坐标小于 1000
【分析】

【代码】

```
#include<iostream>
#include<cstdio>
using namespace std;
int n,m,x1,y1,x2,y2,x3,y3,cx[1000001],ax[1000001],cy[1000001],ay[1000001],k;
char ch,s[10];
int main(){
    scanf("%d",&n);
    for(int i=1;i<=n;i++){
        scanf("%d %d %d %d %d %d",&x1,&y1,&x2,&y2,&x3,&y3);
        int minx=min(x1,min(x2,x3));
        int maxx=max(x1,max(x2,x3));
        cx[minx+1]++;
        cx[maxx]--;
        int miny=min(y1,min(y2,y3));
        int maxy=max(y1,max(y2,y3));
        cy[miny+1]++;
        cy[maxy]--;
    }
    for(int i=1;i<=1000000;i++){
        ax[i]=ax[i-1]+cx[i];
        ay[i]=ay[i-1]+cy[i];
```

```
    }
    scanf("%d",&m);
    for(int i=1;i<=m;i++){
        scanf("%s = %d",s,&k);
        if(s[0]=='x')printf("%d\n",ax[k]);
        if(s[0]=='y')printf("%d\n",ay[k]);
    }
}
```