基础算法

20计科官昕

目录

前缀和

partial Sum

前缀和是什么?

- 所谓前缀和，其实就是数组中某个下标之前所有数组元素的总和
- 举个例子:
  - 数组 a[]={1,2,3,4,5}
  - 则其前缀和数组为 s[]={1,3,6,10,15}
  - 用公式来表示的话就是：$S[i]=\Sigma a[j]\ ,\ j\in[1,i]$

前缀和有什么用?

- 通过预处理降低程序时间复杂度，便于后续程序编写和运行
- 比如需要频繁求取数组中某一区间[l,r]的总和时:
  - 如果直接进行求取，每一次都要对区间内每个元素进行遍历，很容易出现多次重复计算某一区间的和的情况，这是一种浪费，而且在询问次数过多时有超时的危险
  - 而使用前缀和的话，只需要一次O(n)的预处理，此后每次询问的时间复杂度均为O(1)，即s[r]-s[l-1]。在需要多次查询结果时时间优势极大

差分

differentiation

什么是差分?

- 差分与前缀和是互逆运算，对前缀和序列取差分即为原序列
  - 即：a[i]=s[i]-s[i-1]
- 差分和前缀和一样，是一种预处理的方式，经过差分后，数组记录的内容为相邻两位之间的差值，请思考一下这种处理方式有什么优点?

差分的作用

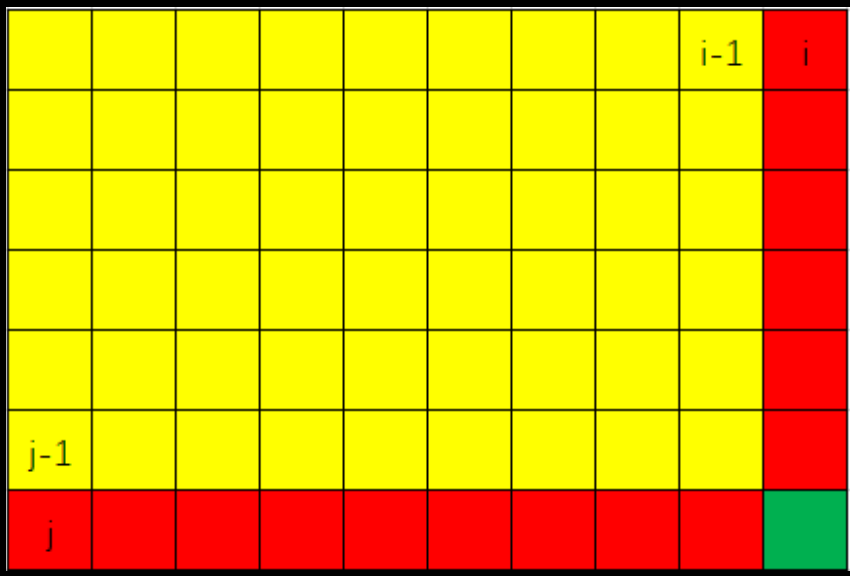
- 如前所述，差分后的数组为相邻两元素的差值，即元素的相对大小
  - 所以在整体对某一区间[l,r]加或减去某一个固定值时，差分序列能在O(1)的时间复杂度下完成这一操作
  - 设该区间都增加一个数n，操作为：s[l]+n，s[r+1]-n
- 可以说，如果能够熟练的运用前缀和与差分，在解决一些问题时将会便利许多，其实比起前缀和与差分，这种预处理的思路可以说使用的更为广泛也更加重要，希望同学们能够借这两种预处理的思路理解并熟悉预处理的思路

二维前缀和与差分

- 除了之前我们说的一位数组上的前缀和与差分，有些时候还需要用到二维上的前缀和与差分，二维和一维定义相同，操作类似，不过有一些细节需要注意：
  - 计算前缀和与差分时，要同时考虑行和列
  - 可能存在重复计算的情况，要合理运用容斥原理
- 大家可以试试自己推导一下二维上的公式

二维前缀和

- 如右图所示，设绿色部分为第j行第i列
  - 设原数组为a，前缀和数组为s，则有：
    - 定义式：S[i,j]=Σa[x,y] , x∈[1,i] j∈[1,j]
    - 递推式：S[i,j]=S[i-1,j]+S[i,j-1]-S[i-1][j-1]+A[i,j]
    - 变形式：A[i,j]=S[i,j]-S[i-1,j]-S[i,j-1]-S[i-1][j-1]



二维差分

- 二维差分比起一维差分，区间增减时处理元素数从2个增加到了4个
- 差分与前缀和类似，在二维上需要考虑行和列的互相影响
  - 话不多说，直接上代码：

```
void insert(int x1,int y1,int x2,int y2,int c)
{
    val[x1][y1]+=c;
    val[x2+1][y1]-=c;
    val[x1][y2+1]-=c;
    val[x2+1][y2+1]+=c;
    return ;
}
```

结构体

struct

结构体的定义

- 结构体是一种自定义的构造数据类型，可以通过将基本数据类型进行有机组合，得到更便于自己理解和使用的新类型
- 定义语句格式

```
struct 结构体类型名称
{
    成员内容;
};
```

结构体的定义

- 举个例子：

```
struct student
{
    char name[20];  //姓名
    int No;         //学号
    float score[9]; //成绩
};
```

- 这里我们定义了一个有三个成员变量的结构体student，这样，我们就可以把这三个属性当做一个整体来使用和修改，无论是编写代码的难度还是可读性都得到了改善

结构体的使用

- 使用结构体时，可以将其当做正常的数据类型来使用，不过由于定义的不完善，可能有些部分无法正常使用，如：比较大小
- 使用时对于这种无法正常使用的部分，可以考虑通过重载运算符等方式完善定义，不过较为复杂，这里就不展开说了，感兴趣的同学可以自行查阅资料
- 调用成员变量的方式为：
  - (结构体变量名).(成员变量名)
  - 结构体成员变量可以直接按其类型正常使用

结构体的使用

- 举个例子：

```
int main()
{
    student s[2];    //结构体变量的定义
    s[1].name="YYS" //成员变量可以直接赋值
    cin>>s[1].No;    //也可以从用户输入
    s[0]=s[1];       //同类型结构体变量可以互相赋值
    s[0].name="zls"
    s[0].NO++;       //也可以对成员变量进行各种修改
    cout<<s[0].No;   //当然，输出自然是没问题的
}
```

其他

others

时间复杂度

- 我们经常说的某算法的时间复杂度即该算法的用时随数据规模而增长的趋势，是衡量算法效率的一种重要方式
  - 时间复杂度在竞赛上的应用主要为按照数据规模推断算法运行时间，辅助我们进行算法的选择并提前规避TLE
  - 在进行算法的时间复杂度的计算时，我们看中的是算法用时随数据规模变化的趋势，因此往往忽略常数，只保留其量级特点
  - 尽管对于不同基本操作在复杂度的计算上无影响，但实际上运行速度有差别，在一些对时间卡的比较死的题目中，可能复杂度相同的算法不一定都能通过，这时可以考虑改写基本操作

时间复杂度

- 时间复杂度的计算方式简单来说可以看做基本操作的执行次数，具体的以后会讲到，这一讲就只是简单说一下
  - 举个例子：之前的前缀和因为将数组遍历了一遍，设数组大小为n，则时间复杂度为$O(n)$。同理，若二维数组大小为$n*m$，二维前缀和复杂度即为$O(n*m)$。而调用某一个前缀和的值时，因为是直接调用，所以复杂度为常数级，即$O(1)$
  - 其他常见的复杂度:$O(n^2), O(n^3), O(logn), O(n!), O(\sqrt{n})$等

sort函数

- sort函数是c++STL库中的排序函数，可以对数组直接进行快速排序，在无特殊要求或限制的情况下，一般排序时直接调用此函数即可
  - 所需头文件：<algorithm>
  - 格式：sort(数组头指针,数组尾指针[,比较函数]);
    - 例如对a数组前n个元素进行排序为：sort(a,a+n);
    - 数组尾指针指向数组中需排序元素的后一位
    - 比较函数是可选参数，不加默认升序排序
    - 被排序的数据类型必须定义'<'运算符

sort函数的比较函数

- 那我们如果要进行降序排序，或者对没有定义'<'运算符的类型进行排序要怎么办呢？最简单的办法是用sort的第三个参数，即比较函数
- 比较函数是一个返回值为bool类型,两个参数为需排序类型变量的函数
  - 以之前的student类型为例，按学号升序排序的比较函数如下：

```
bool cmp(student a,student b)//sort函数的第三个参数是比较函数的名字，这个例子中即cmp
{
    return a.NO<b.No;
}
```

讲解结束，同学们可以开始快乐做题了