差分 前缀和

前缀和

一维前缀和

定义 $s_i=\sum_{j=1}^{i}a_j$,等价于 $s_i=a_i+s_{i-1}(s_0=0)$
作用：对于固定数组a快速求区间和
$\sum_{i=l}^{r}a_i=s[r]-s[l-1]$
代码

```
int a[N],s[N];
for(int i=1;i<=n;i++){
  cin>>a[i];
  s[i]=s[i-1]+a[i];
}//O(n)预处理
int q;
cin>>q;
while(q--){
  int l,r;
  cin>>l>>r;
  int ans=s[r]-s[r-1];
  cout<<ans<<endl;
  }//q次O(1)询问
```

二维前缀和

定义

$$\begin{aligned}s[n][m]&=\sum_{i=1}^{n}\sum_{j=1}^{m}a_[i][j]\\&=s[n-1][m]+s[n][m-1]-s_[n-1][m-1]+a[n][m]\end{aligned}$$

常见于矩阵，方阵求和：

左上角为(x_1,y_1)右下角为(x_2,y_2)的矩阵中所有元素之和可表示为:

$$\sum_{i=x_1}^{x_2}\sum_{j=y_1}^{y_2}a[i][j]=s[x_2][y_2]-s[x_1-1][y_2]-s[x_2][y_1-1]+s[x_1-1][y_1-1]$$

(容斥原理)
代码

```
int a[N][N],s[N][N];
for(int i=1;i<=n;i++){
  for(int j=1;j<=m;j++){
    cin>>a[i][j];
    s[i][j]=s[i][j-1]+a[i-1][j]-s[i-1][j-1]+a[i][j];
  }
}//O(nm)预处理
int q;
cin>>q;
while(q--){
int f_x,s_x,f_y,s_y;//first_x,second_x,first_y,second_y
  cin>>f_x>>f_y>>s_x>>s_x;
  int ans=s[s_x][s_y]-s[f_x-1][x_y]-s[s_x][f_y-1]+s[f_x-1][f_y-1];
  cout<<ans<<endl;
}//q次O(1)询问
```

差分

从本质上说，差分是前缀和的逆运算

一阶差分

定义: $b_i=a_i-a_{i-1}$
作用：离线处理区间加减
如将数组 a 区间 $[l,r]$ 中的数都加上 x 只需将 b_l 加上 x将 b_{r+1} 减去 x 再对 b 做前缀和还原 a 即可
代码

```
int a[N],b[N];
int n;
cin>>n;
for(int i=1;i<=n;i++){
  cin>>a[i];
  b[i]=a[i]-a[i-1];
}
int q;
cin>>q;
whlie{
  int l,r,x;
  cin>>l>>r>>x;
  b[l]+=x;
  b[r+1]-=x;
}// O(1)处理
for(int i=1;i<=n;i++){
  a[i]=b[i]+a[i-1];
}// O(n)还原
```

- ## 二阶差分

与一阶没有本质区别,运用容斥原理就出来了
定义：$b[i][j]=a[i][j]-a[i-1][j]-a[i][j-1]+a[i-1][j+1]$

```
int a[N][N],b[N][N];
int nm;
cin>>n>>m;

for(int i=1;i<=n;i++){
  for(int j=1;j<=m;j++){
    cin>>a[i][j];
    b[i][j]=a[i][j]-a[i-1][j]-a[i][j-1]+a[i-1][j-1];
  }
}
int q;
cin>>q;
while(q--){
  int f_x,f_y,s_x,s_y,data;
  cin>>f_x>>f_y>>s_x>>s_y>>data;
  b[f_x][f_y]+=data;
  b[f_x][s_y+1]-=data;
  b[s_x+1][f_y]-=data;
  b[s_x+1][s_y+1]+=data;
} // O(1)处理
for(int i=1;i=n;i++){
  for(int j=1;j<=m;j++){
    a[i][j]=b[i][j]+a[i-1][j]+a[i][j-1]-a[i-1][j-1];
  }
}// O(nm)还原
```

关于更多

更多运算

以上所有结论完全可以将$+,-$替换为某运算与其逆运算
如$*,/$或$\oplus,\oplus$(异或是自身的逆运算)仍然成立

更高维

非常稀有
如果要求,那么直接写 n 个循环会把你累死

可以先把 k 维的数组$A[s_1][s_2]...[s_k]$压缩成一个大小 $\prod_{i=1}^{k} s_i$的一维数组
用 dfs 钦定第 i 维度,然后递归去搜
具体算的时候用容斥原理

更高阶

比较稀有
对一维数组 a 的前缀和数组 s 再做一次前缀和得到 s_2 , 就是二阶前缀和
以此类推 $s_i[j] = \sum_{k=1}^{j} s_{i-1}[k]$
(差分同理)
可以实现一些与组合数学序列有关操作
先说结论:
令 $a[1]$ 对 $s_i[j]$ 的贡献为 $k_i[j]$ 倍 $a[1]$ 则

$$\begin{aligned} k_i[j] &= \sum_{p=1}^{j} k_{i-1}[p] \\ &= k_{i-1}[j] + k_i[j-1] \\ &= \binom{j+i-2}{i-1} (i>0, j>0) \end{aligned}$$

关于 $a[x]$ 则可看作 $a[1]$ 向右平移 x 个单位
直接用 $k_i[j-x+1]$,如若 $j-x+1<0$ 直接令 $k_i[j-x+1]$ 为0就好了
可得

$$\begin{aligned} s_i[j] &= \sum_{p=1}^{j} k_i[j-p+1]a[p] \\ &= \sum_{p=1}^{j} \binom{j+i-p-2}{i-1} a[p] \end{aligned}$$

推导

我们不妨将 k 列出来

$i\backslash j$	1	2	3	5	...	n
1	1	1	1	1	...	1
2	1	2	3	4	...	n
3	1	3	6	10	...	$\frac{n(n+1)}{2}$
4	1	4	10	20	...	$\frac{n(n+1)(n+2)}{6}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
m	1	m	$\frac{m(m+1)}{2}$	$\frac{m(m+1)(m+2)}{6}$	...	$\frac{(n+m-2)!}{(n-1)!(m-1)!}$

这显然是杨辉三角(斜着看)

~~直接注意到结论~~

看不出也没关系,手推一下

$$\forall k_1[j]=1$$

$$k_i[j]=\sum_{p=0}^{j}k_{i-1}[p] \qquad p从0或1开始都没问题\ \forall k_j[0]=0$$

$$\sum_{p=0}^{n}\binom{p+j}{p}=\binom{j+n}{n} \quad 朱世杰恒等式$$

剩下的数学归纳法就够了

高维&高阶

依旧考虑 $a[1][1]...[1]$ 对 $s_i[d_1][d_2]...[d_n]$ 的贡献为 $k_i[d_1][d_2]...[d_n]$ 倍

$aa[1][1]...[1]$

发现：当阶数为2时

$$k_i[d_1][d_2]...[d_n]=\prod_{q=1}^{n}d_q$$

又因为

$$\sum_{q=1}^{n}\prod_{q=1}^{n}d_q=\prod_{q=1}^{n}\sum_{q=1}^{n}d_q$$

所以d_q都是独立的
因此

$$k_i[d_1][d_2]...[d_n]=k_i[d_1][1]...[1]*k_i[1][d_2]...[1]*...*k_i[1][1]...[d_n]$$

显然 $k_i[d_1][1]...[1]$ 等价于一维时的情况 而我们已经有了求玩意的公式

$$\begin{aligned}k_i[j]&=\sum_{p=1}^{j}k_{i-1}[p]\\&=k_{i-1}[j]+k_i[j-1]\\&=\binom{j+i-2}{i-1}(i>0,j>0)\end{aligned}$$

因此

$$k_i[d_1][d_2]...[d_n]=\prod_{p=1}^{n}\binom{d_p+i-2}{d_p-1}$$

当然 将数组压缩为一维也是不错的选择

2026.01.24 12:35