分块算法

1.1 分块思想

在信息学竞赛中，常见有序列维护的数据结构，这类问题通常会采取分治类的数据结构来完成，但有些信息分治结构中很难快速合并，这时候可以采取分块。分块算法是将序列（序列长度为 N）进行分块，通常设置一个上限 K，每一块有至多 K 个元素。在序列分块问题上，一般会严格要求每个块都要有 K 个元素，这样就会分成约 N/K 块。（最后一个块除外）

分块方法的优势在于不需要对问题的性质进行过多的分析就可以进行维护，通用性强。缺陷是实现比较繁琐，时间复杂度有时难以承受，但在实际运用中性价比还是非常高。

下面就通过几个例子来逐一展开学习了解。

1.1.1 区间修改，单点查值 A2121

这是一个序列维护的简单题，这里主要用来学习如何分块。

直接处理，设块大小 K，b[i] 表示每个点的块编号，v[i] 是原来的值，lazy[i] 是每个块的标记

参考代码：

```
#include<bits/stdc++.h>
using namespace std;
#define N 1000010
#define ll long long
long long a[N],v[N],b[N];
//v       養a      æ¬ b¼¼ÿ¸  Ŀ
long long n,m,k;
inline void read(long long &x){
    x=0;int f=1;char ch=getchar();
     while(ch<'0'||ch>'9'){if(ch=='-')f=-1;ch=getchar();}
     while(ch>='0'&&ch<='9'){x=x*10+ch-'0';ch=getchar();}
     x*=f;
}
void add(ll x,ll y,ll z){
  for(int i=x;i<=min(b[x]*k,y);i++)
    v[i]+=z;//µ  ¶  µ¥¶4¦
  if(b[x]!=b[y])
    for(int i=(b[y]-1)*k+1;i<=y;i++)
      v[i]+=z;//     ¶  µ¥¶4¦
  for(int i=b[x]+1;i<=b[y]-1;i++)
    a[i]+=z;//     ½
}
int main(){
  read(n);read(m) ;
  k=sqrt(n);
  for(int i=1;i<=n;i++)read(v[i]);
  for(int i=1;i<=n;i++)b[i]=(i-1)/k+1;//³  »¯ÿ¸  Ŀ
```

```
28    for(int i=1;i<=m;i++){
29      ll f,x,y,z;
30      read(f);
31      if(f==1){
32        read(x);read(y);read(z);
33        add(x,y,z);
34      }
35      else{
36        read(x);
37        printf("%lld\n",v[x]+a[b[x]]);//ɪ±jɪ  +±
38      }
39    }
40    return 0;
41  }
```

code/A2121.cpp

分块算法求解问题主要从**以下几个方面去思考：**

1、不完整块的处理

2、中间完整块的处理

3、预处理数据（复杂度一般在 $n\sqrt{n}$ 内）

1.1.2 区间修改，区间查询

问题描述：给出一个长为 n 的数列，以及 n 个操作，操作涉及区间加法，询问区间内小于某个值 x 的元素个数。

解析：

首先，如果考虑只有询问

不完整的块枚举统计即可；而要在每个整块内寻找小于一个值的元素数，于是我们不得不要求块内元素是有序的，这样就能使用二分法对块内查询，需要预处理时每块做一遍排序，复杂度 O(nlogn)，每次查询在 $\sqrt{n}$ 个块内二分，以及暴力 $\sqrt{n}$ 个元素，总复杂度 $O(nlogn+n\sqrt{n}log\sqrt{n})$。

然后，考虑区间修改问题

套用第一题的方法，维护一个加法标记，略有区别的地方在于，不完整的块修改后可能会使得该块内数字乱序，所以头尾两个不完整块需要重新排序，复杂度分析略。

在加法标记下的询问操作，块外还是暴力，查询小于（x -加法标记）的元素个数，块内用（x -加法标记）作为二分的值即可。

参考代码：

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  #define N 100010
4  int lazy[N]={0},v[N],idx[N];
5  //v      養a      æ¬ idx¼¼ÿ¸  Ŀ
6  int n,m,k;
7  vector<int>ve[1001];
```

```cpp
inline void read(int &x){
    x=0;int f=1;char ch=getchar();
     while(ch<'0'||ch>'9'){if(ch=='-')f=-1;ch=getchar();}
     while(ch>='0'&&ch<='9'){x=x*10+ch-'0';ch=getchar();}
     x*=f;
}
void resort(int id){
  ve[id].clear();
  for(int i=(id-1)*k+1;i<=min(id*k,n);i++)
    ve[id].push_back(v[i]);
  sort(ve[id].begin(),ve[id].end());
}
void add(int x,int y,int z){
  for(int i=x;i<=min(idx[x]*k,y);i++)
    v[i]+=z;
  resort(idx[x]); //
  if(idx[x]!=idx[y]){
    for(int i=(idx[y]-1)*k+1;i<=y;i++)
      v[i]+=z;
    resort(idx[y]) ;
  }
  for(int i=idx[x]+1;i<=idx[y]-1;i++)
    lazy[i]+=z;
}
int query(int x,int y,int z){
  int ans=0;
  for(int i=x;i<=min(idx[x]*k,y);i++)
    if(v[i]+lazy[idx[x]]<z)ans++;
  if(idx[x]!=idx[y])
    for(int i=(idx[y]-1)*k+1;i<=y;i++)
      if(v[i]+lazy[idx[y]]<z)ans++;
  for(int i=idx[x]+1;i<=idx[y]-1;i++) {
    int temp=z - lazy[i];//¼
    ans+=lower_bound(ve[i].begin(),ve[i].end() , temp) - ve[i].begin();
  }
  return ans;
}
int main(){
  read(n); k=sqrt(n);
  for(int i=1;i<=n;i++){
    read(v[i]);
    idx[i]=(i-1)/k+1;// •
    ve[idx[i]].push_back(v[i]);// •
  }
  for(int i=1;i<=idx[n];i++)//¿
    sort(ve[i].begin(),ve[i].end()) ;
   for(int i=1;i<=n;i++){
    int op,x,y,z;
    read(op);read(x);read(y);read(z);
    if(op==0)add(x,y,z);
    else
      printf("%d\n",query(x,y,z*z));
  }
  return 0;
}
```

code/loj6278.cpp

1.1.3 区间修改，询问 x 前驱

问题描述：给出一个长为 n 的数列，以及 m 个操作，操作涉及区间加法，询问区间内小于某个值 x 的前驱（比其小的最大元素）。

解析：

依据第二题的解法，其实只要把块内查询的二分稍作修改即可。

可以在块内维护其它结构使其更具有拓展性，比如放一个 set ，这样如果还有插入、删除元素的操作，会更加的方便。

分块的**调试检测技巧：**

如果无法自己分析复杂度, 可以生成一些大数据，然后用两份分块大小不同的代码来对拍，还可以根据运行时间尝试调整分块大小，减小常数。

参考代码：

```
#include <bits/stdc++.h>
#define N 100010
using namespace std;
typedef long long ll;

vector <ll> v[N];

int n,m,k;
ll a[N],lazy[N];
int idx[N];

void resort(int pos){
  v[pos].clear();
  for(int i=(pos-1)*k+1;i<=min(pos*k,n);i++)
    v[pos].push_back(a[i]);
  sort(v[pos].begin(),v[pos].end());
}

void update(int l,int r,ll val){
  for(int i=l;i<=min(idx[l]*k,r);i++)
    a[i]+=val;
  resort(idx[l]);
  if(idx[l]!=idx[r]){
    for(int i=(idx[r]-1)*k+1;i<=r;i++)
      a[i]+=val;
    resort(idx[r]);
  }
  for(int i=idx[l]+1;i<=idx[r]-1;i++)
    lazy[i]+=val;
}

ll query(int l,int r,ll val){
  ll ans=-1;
  for(int i=l;i<=min(idx[l]*k,r);i++)
    if(a[i]+lazy[idx[l]]<val)
      ans=max(ans,a[i]+lazy[idx[i]]);
  if(idx[l]!=idx[r])
    for(int i=(idx[r]-1)*k+1;i<=r;i++)
      if(a[i]+lazy[idx[r]]<val)
```

```
        ans=max(ans,a[i]+lazy[idx[i]]);
    for(int i=idx[l]+1;i<=idx[r]-1;i++){
      int temp=lower_bound(v[i].begin(),v[i].end(),val-lazy[i])-v[i].begin();
      if(!temp) continue;
      ans=max(ans,v[i][temp-1]+lazy[i]);
    }
    return ans;
}

int main(){
    int op,x,y;ll z;
    scanf("%d",&n  );
    k=sqrt(n);
    for(int i=1;i<=n;i++){
      scanf("%lld",&a[i]);
      idx[i]=(i-1)/k+1;
      v[idx[i]].push_back(a[i]);
    }
    for(int i=1;i<=idx[n];i++)
      sort(v[i].begin(),v[i].end());
    for(int i=1;i<=n;i++){
      scanf("%d%d%d%lld",&op,&x,&y,&z);
      if(op==0) update(x,y,z);
      else  printf("%lld\n",query(x,y,z));
    }
    return 0;
}
```

code/loj6279.cpp

1.1.4 区间开方修改，区间求和

问题描述：给出一个长为 n 的数列，以及 m 个操作，操作涉及区间开方，区间求和。

解析：

稍作思考可以发现，开方操作比较棘手，主要是对于整块开方时，必须要知道每一个元素，才能知道他们开方后的和，也就是说，难以快速对一个块信息进行更新。

看来我们要另辟蹊径。不难发现，这题的修改就只有下取整开方，而一个数经过几次开方之后，它的值就会变成 0 或者 1。

如果每次区间开方只不涉及完整的块，意味着不超过 $2\sqrt{n}$ 个元素，直接暴力即可。

如果涉及了一些完整的块，这些块经过几次操作以后就会都变成 0 / 1，于是我们采取一种分块优化的暴力做法，只要每个整块暴力开方后，记录一下元素是否都变成了 0 / 1，区间修改时跳过那些全为 0 / 1 的块即可。

这样每个元素至多被开方不超过 6-7 次，复杂度略高。

参考代码：

```
#include<bits/stdc++.h>
using namespace std;
```

```
#define N 100010
#define LL long long
LL flag[N]={0},v[N],idx[N],sum[N]={0};
//v     賽lazy     æ¬ idx¼¼ÿ¸ Ŀ  ,sum  ÿ¿
LL n,m,k;
inline void read(LL &x){
    x=0;int f=1;char ch=getchar();
     while(ch<'0'||ch>'9'){if(ch=='-')f=-1;ch=getchar();}
     while(ch>='0'&&ch<='9'){x=x*10+ch-'0';ch=getchar();}
     x*=f;
}
void solve_sqrt(int id){
   if(flag[id])return ;
   flag[id]=1;
   sum[id]=0;
   for(int i=(id-1)*k+1 ; i<=id*k ; i++){//   ¼
      v[i]=sqrt(v[i]);
      sum[id]+=v[i];
      if(v[i]>1)flag[id]=0;
   }
}
void add(LL x,LL y){
   for(LL i=x;i<=min(idx[x]*k,y);i++){//sqrt(n)
      sum[idx[i]]-=v[i];
      v[i]=sqrt(v[i]);
      sum[idx[i]]+=v[i];
   }
   if(idx[x]!=idx[y]){
      for(LL i=(idx[y]-1)*k+1;i<=y;i++){
         sum[idx[i]]-=v[i];
         v[i]=sqrt(v[i]);
         sum[idx[i]]+=v[i];
      }
   }
   for(LL i=idx[x]+1;i<=idx[y]-1;i++) //sqrt(n)
      solve_sqrt(i);//sqrt(n)
}
LL query(LL x,LL y){
   LL ans=0;
   for(LL i=x;i<=min(idx[x]*k,y);i++)
      ans+=v[i];
   if(idx[x]!=idx[y])
      for(LL i=(idx[y]-1)*k+1;i<=y;i++)
         ans+=v[i];

   for(LL i=idx[x]+1;i<=idx[y]-1;i++) //sqrt(n)
      ans+=sum[i];
   return ans;
}
int main(){
   read(n);k=sqrt(n);
   for(LL i=1;i<=n;i++)  read(v[i]);
   for(LL i=1;i<=n;i++){
      idx[i]=(i-1)/k+1;// •
      sum[idx[i]]+=v[i];
   }

   for(int i=1;i<=n;i++){//n
```

```
61      LL op,x,y,z;
62      read(op);read(x);read(y);read(z);
63      if(op==0)
64        add(x,y);
65      else
66        printf("%lld\n",query(x,y));
67    }
68    return 0;
69  }
```

code/loj6281.cpp

1.1.5 单点插入，单点询问

问题描述：给出一个长为 n 的数列，以及 m 个操作，操作涉及单点插入，单点询问，数据随机生成。

解析：

先说随机数据的情况, 之前提到过，如果我们块内用数组以外的数据结构，能够支持其它不一样的操作，比如此题每块内可以放一个动态的数组，每次插入时先找到位置所在的块，再暴力插入，把块内的其它元素直接向后移动一位，当然用链表也是可以的。

查询的时候类似，复杂度分析略。

但是这样做有个问题，如果数据不随机怎么办？

如果先在一个块有大量单点插入，这个块的大小会大大超过 $\sqrt{n}$，那块内的暴力就没有复杂度保证了。

还需要引入一个操作：重新分块（重构）

每根号 n 次插入后，重新把数列平均分一下块，重构需要的复杂度为 $O(n)$，重构的次数为 $\sqrt{n}$, 所以重构的复杂度没有问题，而且保证了每个块的大小相对均衡。当然，也可以当某个块过大时重构，或者只把这个块分成两半。

参考代码：

```
#include<bits/stdc++.h>
using namespace std;
#define N 100010
#define LL long long
typedef pair<int,int> P;
int v[N],temp[N];
int  n,m,k,last;
vector<int>ve[1010];
inline void read(int &x){
   x=0;int f=1;char ch=getchar();
    while(ch<'0'||ch>'9'){if(ch=='-')f=-1;ch=getchar();}
    while(ch>='0'&&ch<='9'){x=x*10+ch-'0';ch=getchar();}
    x*=f;
}
P query(int x){//    xµL  ţ¬º
  int id=1;
  for(;x >  ve[id].size();id++)
```

```
18      x-=ve[id].size();
19    return make_pair(id,x-1);
20  }
21  void rebuild(){
22    int top=0;
23    for(int i=1;i<=last;i++){
24      for(int j=0;j<ve[i].size();j++)
25        temp[++top]=ve[i][j];
26      ve[i].clear();
27    }
28    int kk=sqrt(top);
29    for(int i=1;i<=top;i++){
30      int id=(i-1)/kk+1;
31      ve[id].push_back(temp[i]);
32    }
33    last=(top-1)/kk+1;
34  }
35  void insert(int x,int v){
36    P t= query(x);
37    ve[t.first].insert(ve[t.first].begin() + t.second , v);
38    if(ve[t.first].size()>20*k)
39      rebuild();
40  }
41  int main(){
42    read(n);k=sqrt(n);
43    for(int i=1;i<=n;i++)read(v[i]);
44    for(int i=1;i<=n;i++){
45      int id=(i-1)/k+1;
46      ve[id].push_back(v[i]);
47    }
48    last=(n-1)/k+1;
49    for(int i=1;i<=n;i++){
50      int op,x,y,z;
51      read(op);
52      if(op==0){
53        read(x);read(y);read(z);
54        insert(x,y);
55      }
56      else{
57        read(x);read(y);read(z);
58        P t=query(y);
59        printf("%d\n",ve[t.first][t.second]);
60      }
61    }
62    return 0;
63  }
```

code/loj6282.cpp