线段树（基础篇）

——陈宇，张嘉文

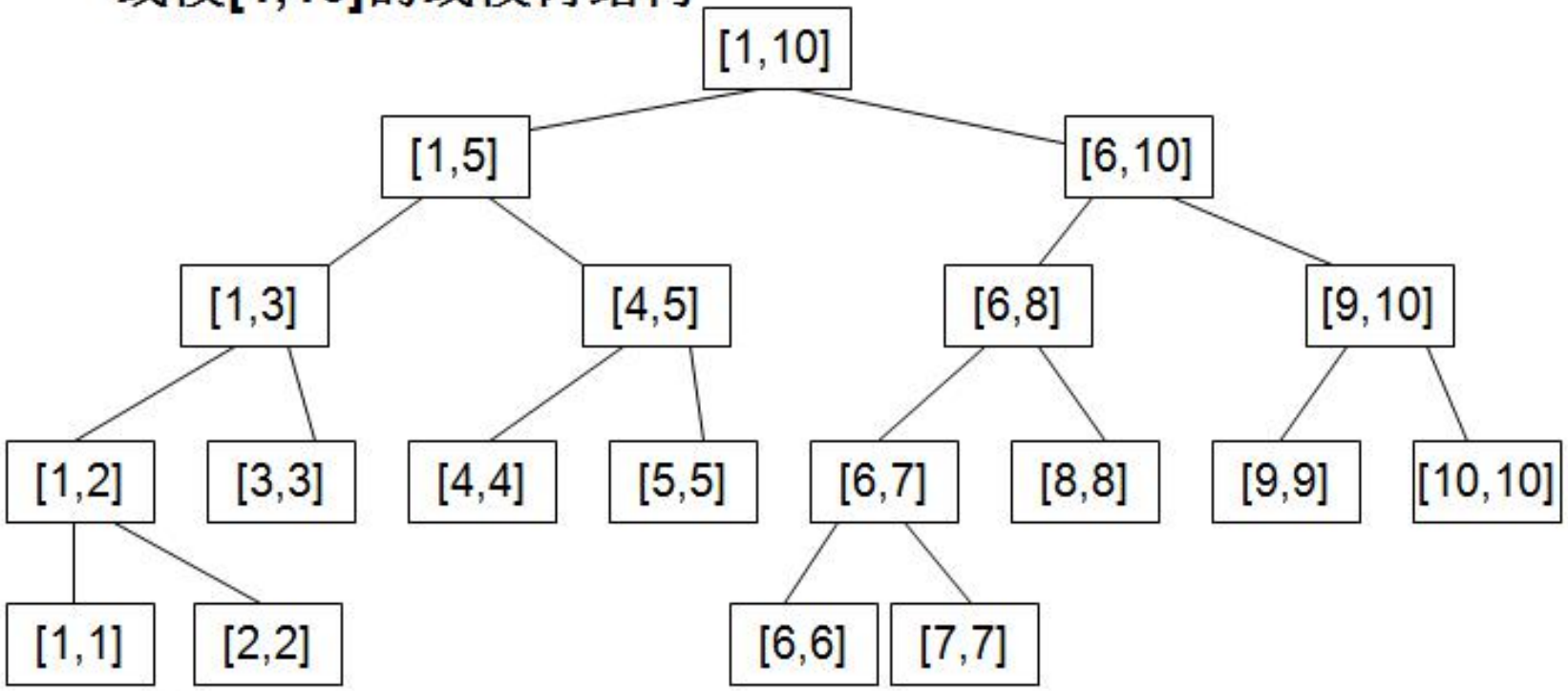
算法介绍

- 线段树是一种二叉搜索树，与区间树相似，它将一个区间划分成一些单元区间，每个单元区间对应线段树中的一个叶结点。
- 使用线段树可以快速的查找某一个节点在若干条线段中出现的次数，时间复杂度为O(logN)。而未优化的空间复杂度为2N，实际应用时一般还要开4N的数组以免越界，因此有时需要离散化让空间压缩。

形象认识

线段树

线段[1,10]的线段树结构



线段树的构造与使用

一般分为三步：

1.Build//构造

2.Update//更新

3.Query//查询

数组的定义

数组类型按需求定义，大小一般开4倍。

```
long long tr[maxn*4];
```

Build建树

用递归来建树，到叶子节点输入该位置的值。

```
void build(int i,int l,int r)
{
    if(l==r)
    {
        scanf("%d",&tr[i]);
        return;
    }
    int mid=(l+r)/2;
    build(2*i,l,mid);
    build(2*i+1,mid+1,r);
    PushUp(i);
}
```

Update更新

此处为单点更新，x为更新的位置。用递归的算法判断x节点是在当前节点的左子树还是右子树，直到找到位置并更新。

```
void update(int i,int l,int r,int x,int c)
{
    if(l==r&&l==x)
    {
        tr[i]=c;
        return;
    }
    int mid=(l+r)/2;
    if(x<=mid) update(2*i,l,mid,x,c);
    else update(2*i+1,mid+1,r,x,c);
    PushUp(i);
}
```

Query查询

此处为区间查询，[x,y]为查询的区间。递归到[l,r]在[x,y]区间内直接返回该节点的信息，不用向下递归。（多此一举）

```
long long query(int i,int l,int r,int x,int y)
{
    long long sum=0;
    if(x<=l&&r<=y)
    {
        sum+=tr[i];
        return sum;
    }
    pushdown(i,r-l+1);
    int mid=(l+r)/2;
    if(x<=mid) sum+=query(2*i,l,mid,x,y);
    if(y>mid) sum+=query(2*i+1,mid+1,r,x,y);
    pushup(i);
    return sum;
}
```

Pushup向上更新

维护某一大区间的值。

```
void pushup(int i)
{
    tr[i]=tr[i<<1]+tr[i<<1|1];
}
```

线段树的区间更新

如果区间更新的时候对区间上的每一个叶子节点都进行更新，时间复杂度很大，还不如用树状数组。那么线段树是怎么做到区间更新的？

Lazy思想

Lazy思想：对整个结点进行的操作，先在结点上做标记，而并非真正执行，直到根据查询操作的需要分成两部分。

根据Lazy思想，我们可以在不代表原线段的结点上增加一个值add，即为对这个结点，留待以后执行的插入操作k值的总和。对整个结点插入时，只更新sum和add值而不向下进行，这样时间复杂度可证明为O(logN）。

对一个add值为0的结点整个进行查询时，直接返回存储在其中的sum值；而若对add不为0的一部分进行查询，则要更新其左右子结点的sum值，然后把add值传递下去，再对这个查询本身，左右子结点分别递归下去。时间复杂度也是O(nlogN）。

代码实现

```
void pushdown(int i,int len)
{
    if(add[i])
    {
        add[i<<1]+=add[i];
        add[i<<1|1]+=add[i];
        tr[i<<1]+=add[i]*(len-len/2);
        tr[i<<1|1]+=add[i]*(len/2);
        add[i]=0;
    }
}
```

完整代码（含Lazy）

```
void build(int i,int l,int r)
{
    if(l==r)
    {
        scanf("%lld",&tr[i]);
        return;
    }
    int mid=(l+r)/2;
    build(i*2,l,mid);
    build(i*2+1,mid+1,r);
    pushup(i);
}
```

```
void update(int i,int l,int r,int x,int y,int c)
{
    if(x<=l&&r<=y)
    {
        add[i]+=c;
        tr[i]=tr[i]+(r-l+1)*c;
        return;
    }
    pushdown(i,r-l+1);
    int mid=(l+r)/2;
    if(x<=mid) update(2*i,l,mid,x,y,c);
    if(y>mid) update(2*i+1,mid+1,r,x,y,c);
    pushup(i);
}
```

```
long long query(int i,int l,int r,int x,int y)
{
    long long sum=0;
    if(x<=l&&r<=y)
    {
        sum+=tr[i];
        return sum;
    }
    pushdown(i,r-l+1);
    int mid=(l+r)/2;
    if(x<=mid) sum+=query(2*i,l,mid,x,y);
    if(y>mid) sum+=query(2*i+1,mid+1,r,x,y);
    pushup(i);
    return sum;
}
```

NEFU1266

区间更新的模板题。

用上面的模板试一试吧~

https://www.cnblogs.com/xenny/p/9801703.html

这个博客也写得挺详细的

奥赛一本通的模板

- 这个模板一直用得挺顺的：先看单点更新

```
const int N=1e5+1;
const int inf=0x3f3f3f3f;
int a[4*N];
int update(int k,int l,int r,int x,int v)
{
    if (x>r||x<l) return 0;
    if (l==r&&l==x)
    {
        a[k]=v;
        return 0;
    }
    int mid=(l+r)/2;
    update(2*k,l,mid,x,v);
    update(2*k+1,mid+1,r,x,v);
    a[k]=min(a[2*k],a[2*k+1]);
  return 0;
}
```

我一直偷懒用这个来建立一棵树，也没慢多少，大家可以试试

区间查询（求和，最大和最小值）

```
int query(int k,int l,int r,int x,int y)
{
    if (r<x||y<l) return inf;
    if (l>=x&&r<=y) return a[k];
    int mid=(l+r)/2;
    int ans1=query(2*k,l,mid,x,y);
    int ans2=query(2*k+1,mid+1,r,x,y);
    return min(ans1,ans2);
}
```

林大OJ 2200 数据区间查询

Problem:2200 Time Limit:1000ms Memory Limit:65535K

Description

有N个正整数，（1<=n<=1e5）;然后有多组询问，或者更新修改一个值，或者计算区间的最小值？

Input

输入一个N和T，(1<=T<=100),T表示询问的组数;
接下来是N个正整数;
然后是T组询问，每组询问有3个数x,y,z,
当 x=1时，表示把第y个值更新为z;
当x=2时，表示查询【y,z】区间的最小值;

Output

输出查询的最小值

```
int main()
{
    ios::sync_with_stdio(0);
     int n,t,w;
    int x,y,z,ans;
    cin>>n>>t;
    for(int i=1;i<=n;i++)
    {
        cin>>w;
        update(1,1,n,i,w);     //注意我的偷懒的建立树的方法
    }
    for(int i=1;i<=t;i++)
    {
       cin>>x>>y>>z;
       if (x==1)
       update(1,1,n,y,z);
       else
       {
          ans=query(1,1,n,y,z);
          cout<<ans<<endl;
       }
    }
    return 0;
}
```

区间修改，单点查询：我的代码
洛谷：P3372

```
const int N=1e5+5;
LL sum[4*N],tag[4*N];
int b[N],c[N];
int add(int k,int l,int r,int v)
{
    tag[k]+=v;
    sum[k]+=(r-l+1)*v;
}

int pushdown(int k,int l,int r,int mid)
{
    if (tag[k]==0) return 0;
    add(2*k,l,mid,tag[k]);
    add(2*k+1,mid+1,r,tag[k]);
    tag[k]=0;

}
```

单点更新

```
int update(int k,int l,int r,int x,int y,int v)
{   if (y<l||x>r) return 0;
    if (l>=x&&r<=y)
    {
        add(k,l,r,v);
        return 0;
    }
    int mid=(l+r)>>1;
    pushdown(k,l,r,mid);
     update(2*k,l,mid,x,y,v);
     update(2*k+1,mid+1,r,x,y,v);
    sum[k]=sum[2*k]+sum[2*k+1];

}
```

查询

```
LL query(int k,int l,int r,int x,int y)
{
    if (l>=x&&r<=y) return sum[k];
    int mid=(l+r)>>1;
    pushdown(k,l,r,mid);
    LL res=0;
    if (x<=mid) res=query(2*k,l,mid,x,y);
    if (y>mid) res+=query(2*k+1,mid+1,r,x,y);
    return res;

}
```

刚开始建树

```
int create(int k,int l,int r)
{ if (l==r)
  {
    sum[k]+=b[l];
    return 0;
  }
  int mid=(l+r)/2;
  create(2*k,l,mid);
  create(2*k+1,mid+1,r);
  sum[k]=sum[2*k]+sum[2*k+1];

}
```

```
int main()
{
    int n,m,k,x,y,p;
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++)
    {
        scanf("%d",&b[i]);
        //c[i]=b[i];
        //update(1,1,n,i,i,b[i]);
    }
    create(1,1,n);
```

```
for(int i=1;i<=m;i++)
   {
     scanf(“%d",&k);
     if (k==1)
     {

scanf("%d%d%d",&x,&y,&p);
         update(1,1,n,x,y,p);
     }
      if (k==2)
      {
          scanf("%d%d",&x,&y);
          LL ans=query(1,1,n,x,y);
          printf("%lld\n",ans);
      }
   }

   return 0;
}
```

线段树的区间合并和扫描线

- 属于线段树的中级算法，后续回学习
- 题目很多：林大OJ 和洛谷都有作业

谢谢聆听！