树状数组 & 线段树

许臻佳

上海交通大学

August 7, 2017

Content

问题引入

维护一个数组 $a[1], a[2], \cdots, a[n]$, 有 q 个操作 $(n, q \leq 10^5)$:
修改: $a[x] = a[x] + t$
询问: $a[l] + a[l+1] + \cdots + a[r]$

问题引入

维护一个数组 $a[1], a[2], \cdots, a[n]$，有 q 个操作 $(n, q \le 10^5)$：
修改：$a[x] = a[x] + t$
询问：$a[l] + a[l+1] + \cdots + a[r]$

直接模拟：修改 $O(1)$，询问 $O(n)$
维护前缀和：修改 $O(n)$，询问 $O(1)$

问题引入

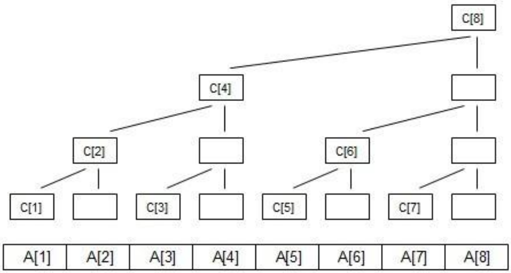
维护一个数组 $a[1], a[2], \cdots, a[n]$, 有 q 个操作 $(n, q \leq 10^5)$:
修改: $a[x] = a[x] + t$
询问: $a[l] + a[l+1] + \cdots + a[r]$

直接模拟: 修改 $O(1)$, 询问 $O(n)$
维护前缀和: 修改 $O(n)$, 询问 $O(1)$
树状数组: 修改 $O(\log n)$, 询问 $O(\log n)$, 总复杂度 $O(n \log n)$

树状数组结构

$$f[x] = \sum_{i=x-lowbit(x)+1}^{x} a[i]$$

$$lowbit(x) = x \ \& \ (-x) = x \ and \ (-x)$$



询问

询问：从低到高，将 1 变成 0

$$13_{10} = 1101_2 = 1 + 4 + 8$$

$$[1, 13] = [13, 13] + [9, 12] + [1, 8]$$

```
//query(x)
//a[1] + a[2] + a[3]... + a[x]
int query(int x){
    sum = 0;
    while(x != 0){
        sum = sum + f[x];
        x = x - lowbit(x);
    }
    return sum;
}
```

修改

```
//modify(x, d)
//a[x] = a[x] + d
void modify(int x, int d{
    while(x <= n){
        f[x] = f[x] + d;
        x = x + lowbit(x);
    }
}
```

问题引入

维护一个数组 $a[1], a[2], \cdots, a[n]$, 有 q 个操作 $(n, q \leq 10^5)$:
修改: $a[x] = a[x] + t$
询问: $max(a[l], a[l+1], \cdots, a[r])$

问题引入

维护一个数组 $a[1], a[2], \cdots, a[n]$, 有 q 个操作 $(n, q \le 10^5)$:
修改: $a[x] = a[x] + t$
询问: $max(a[l], a[l+1], \cdots, a[r])$

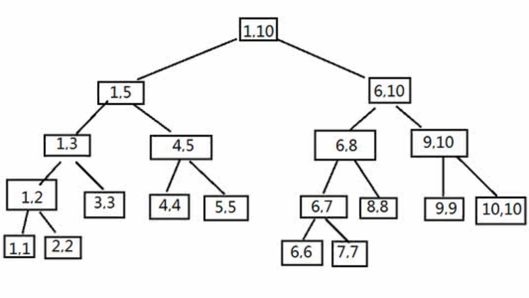
sum 变成了 max
sum(l, r) = sum(1, r) - sum(1, l - 1)
max(l, r) = max(1, r) - max(1, l - 1)???

问题引入

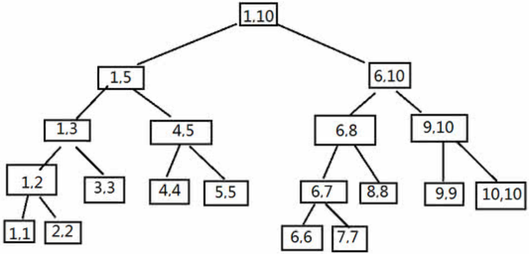
维护一个数组 $a[1], a[2], \cdots, a[n]$, 有 q 个操作 $(n, q \le 10^5)$:
修改: $a[x] = a[x] + t$
询问: $max(a[l], a[l+1], \cdots, a[r])$

sum 变成了 max
sum(l, r) = sum(1, r) - sum(1, l - 1)
max(l, r) = max(1, r) - max(1, l - 1)???
线段树: 修改 $O(\log n)$, 询问 $O(\log n)$, 总复杂度 $O(n \log n)$

线段树结构



线段树结构



- 任何一个区间都可以被表示为 $O(\log N)$ 个不相交区间的并
- 如何存储：存在数组中
  节点 x 的左儿子是 x * 2，右儿子是 x * 2 + 1

建树

```
//build(x, l, r)
void build(int x, int l, int r){
    if(l == r) f[x] = a[l];
    else{
        int mid = (l + r) / 2;
        build(x * 2, l, mid);
        build(x * 2 + 1, mid + 1, r);
        f[x] = max(f[x * 2], f[x * 2 + 1]);
    }
}
```

修改

```
//modify(x, l, r, pos, d)
void modify(int x, int l, int r, int pos, int d){
    if(l == r) f[x] = f[x] + d;
    else{
        int mid = (l + r) / 2;
        if(pos <= mid)
            modify(x * 2, l, mid, pos, d);
        else modify(x * 2 + 1, mid + 1, r, pos, d);
        f[x] = max(f[x * 2], f[x * 2 + 1]);
    }
}
```

询问

```
//query(x, l, r, ql, qr)
int query(int x, int l, int r, int ql, int qr){
    if(l >= ql && r <= qr) return f[x];
    else{
        int mid = (l + r) / 2;
        int s = -INF;
        if(ql <= mid)
            s = max(s,query(x*2, l, mid, ql, qr));
        if(qr >= mid + 1)
            s = max(s,query(x*2+1,mid+1,r,ql,qr));
        return s;
    }
}
```

线段树标记

维护一个数组 $a[1], a[2], \cdots, a[n]$, 有 q 个操作 $(n, q \leq 10^5)$:
修改: $a[x] = a[x] + t, \ \ x \in [l, \ r]$
询问: $max(a[l], a[l+1], \cdots, a[r])$

线段树标记

维护一个数组 $a[1], a[2], \cdots, a[n]$，有 q 个操作 $(n, q \le 10^5)$：
修改： $a[x] = a[x] + t, \ \ x \in [l, \ \ r]$
询问： $max(a[l], a[l+1], \cdots, a[r])$

- 将一个修改变成多个修改，单次最大复杂度达到 $O(n \log n)$，总复杂度 $O(n^2 \log n)$

线段树标记

维护一个数组 $a[1], a[2], \cdots, a[n]$, 有 q 个操作 $(n, q \leq 10^5)$:
修改: $a[x] = a[x] + t,\ \ x \in [l,\ \ r]$
询问: $max(a[l], a[l+1], \cdots, a[r])$

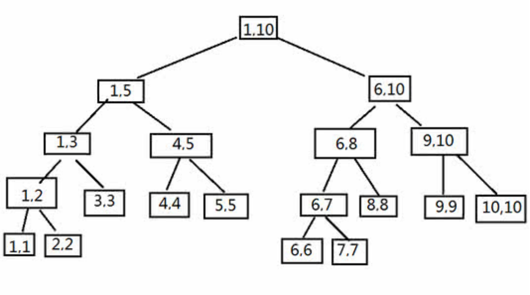
- 将一个修改变成多个修改, 单次最大复杂度达到 $O(n \log n)$, 总复杂度 $O(n^2 \log n)$
- 在每个点上除了维护这段区间的 max, 再多维护一个 tag(标记), 表示这一段区间的修改。
- 之前提到: 一个区间最多被拆成 $O(\log n)$ 个线段树区间, 所以在线段树上修改的时候, 如果修改区间完全包含当前区间, 那就不需要再往下走了, 在这个点打一个标记就行了。

线段树标记

维护一个数组 $a[1], a[2], \cdots, a[n]$，有 q 个操作 $(n, q \le 10^5)$：
修改：$a[x] = a[x] + t,\ x \in [l,\ r]$
询问：$max(a[l], a[l+1], \cdots, a[r])$

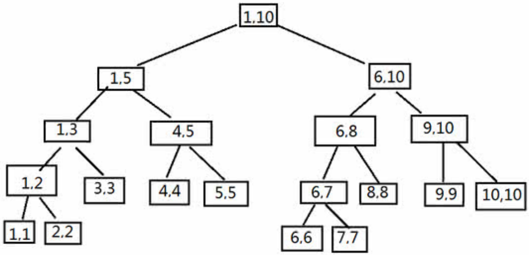
- 将一个修改变成多个修改，单次最大复杂度达到 $O(n \log n)$，总复杂度 $O(n^2 \log n)$
- 在每个点上除了维护这段区间的 max，再多维护一个 tag(标记)，表示这一段区间的修改。
- 之前提到：一个区间最多被拆成 $O(\log n)$ 个线段树区间，所以在线段树上修改的时候，如果修改区间完全包含当前区间，那就不需要再往下走了，在这个点打一个标记就行了。
- 但下次一旦要访问下面的区间，一定要记得下传标记!!!

线段树标记举例



- 比如要修改区间 [4 , 8]， 在 [4, 5], [6, 8] 会放 tag

线段树标记举例



- 比如要修改区间 [4 , 8]， 在 [4, 5], [6, 8] 会放 tag
- 之后如果要询问 [5, 7]， 区间 [4, 5], [6, 8] 的 tag 会下传， 此时区间 [6, 7] 上有 tag。(对于单点，有没有 tag 是一样的)

修改 (带标记)

```
//modify(x, l, r, ll, rr, d)
void modify(int x,int l,int r,int ll,int rr,int d){
    if(l >= ll && r <= rr){
        f[x] = f[x] + d;
        tag[x] = tag[x] + d;
    }
    else{
        downtag(x);
        int mid = (l + r) / 2;
        if(ll <= mid)
            modify(x*2,l,mid,ll,rr,d);
        if(rr >= mid+1)
            modify(x*2+1,mid+1,r,ll,rr,d);
        f[x] = max(f[x * 2], f[x * 2 + 1]);
    }
```

询问

```
//query(x,l, r, ql, qr)
int query(int x, int l, int r, int ql, int qr){
    if(l >= ql && r <= qr) return f[x];
    else{
        downtag(x);
        int mid = (l + r) / 2;
        int s = -INF;
        if(ql <= mid)
            s = max(s,query(x*2, l, mid, ql, qr));
        if(qr >= mid + 1)
            s = max(s,query(x*2+1,mid+1,r,ql,qr));
        return s;
    }
}
```

下传标记

```
//downtag(x)
void downtag(int x){
    if(tag[x] == 0) return;
    tag[x * 2] += tag[x];
    f[x * 2] += tag[x];
    tag[x * 2 + 1] += tag[x];
    f[x * 2 + 1] += tag[x];
    tag[x] = 0;
}
```

下传标记

```
//downtag(x)
void downtag(int x){
    if(tag[x] == 0) return;
    tag[x * 2] += tag[x];
    f[x * 2] += tag[x];
    tag[x * 2 + 1] += tag[x];
    f[x * 2 + 1] += tag[x];
    tag[x] = 0;
}
```

- tag 对自己已经修改了，但是对子树还没有修改!!!
- tag 一旦下传，一定要把自己的 tag 清空!!!

树状数组例题

维护一个数组 $a[1], a[2], \cdots, a[n]$, 有 q 个操作 $(n, q \leq 10^5)$:
修改: $a[x] = a[x] + d,\ x \in [l,\ r]$
询问: $a[x]$

树状数组例题

维护一个数组 $a[1], a[2], \cdots, a[n]$, 有 q 个操作 $(n, q \leq 10^5)$:
修改: $a[x] = a[x] + d,\ \ x \in [l,\ r]$
询问: $a[x]$

之前是单点修改，区间询问；现在改为区间修改，单点查询

树状数组例题

考虑 $b[x] = a[x] - a[x-1]$(b 是 a 的差分序列)

- 区间修改 [l, r] + d 就变成了:b[l] + d; b[r + 1] - d
- 单点询问 a[x] 就变成了: 询问 b 数组 [1, x] 的和
- 所以对 b 维护树状数组即可

树状数组例题（另一种写法）

```
void modify(int x, int d){
    while(x >= 0){
        a[x] = a[x] + d;
        x = x - lowbit(x);
    }
}
int query(int x){
    int sum = 0;
    while(x <= n){
        sum = sum + a[x];
        x = x + lowbit(x);
    }   return sum;
}
```

树状数组例题 (另一种写法)

```
void modify(int x, int d){
    while(x >= 0){
        a[x] = a[x] + d;
        x = x - lowbit(x);
    }
}
int query(int x){
    int sum = 0;
    while(x <= n){
        sum = sum + a[x];
        x = x + lowbit(x);
    }   return sum;
}
```

区间: 不断减 lowbit; 单点: 不断加 lowbit

线段树例题

bzoj 1798

维护一个数组 $a[1], a[2], \cdots, a[n]$，有 q 个操作 $(n, q \leq 10^5)$：
修改 1： $a[x] = a[x] + d,\ \ x \in [l,\ \ r]$
修改 2： $a[x] = a[x] * d,\ \ x \in [l,\ \ r]$
询问： $a[l] + a[l+1] \cdots + a[r]$

线段树例题

bzoj 1798

维护一个数组 $a[1], a[2], \cdots, a[n]$，有 q 个操作 $(n, q \le 10^5)$：
修改 1： $a[x] = a[x] + d,\ \ x \in [l,\ \ r]$
修改 2： $a[x] = a[x] * d,\ \ x \in [l,\ \ r]$
询问： $a[l] + a[l+1] \cdots + a[r]$

之前是区间加，区间求和；现在改成区间加，区间乘，区间求和

线段树例题

bzoj 1798

维护一个数组 $a[1], a[2], \cdots, a[n]$，有 q 个操作 $(n, q \le 10^5)$：
修改 1： $a[x] = a[x] + d,\ x \in [l,\ r]$
修改 2： $a[x] = a[x] * d,\ x \in [l,\ r]$
询问： $a[l] + a[l+1] \cdots + a[r]$

之前是区间加，区间求和；现在改成区间加，区间乘，区间求和
每个点维护两个 tag(tag_k, tag_b)
表示子树内每个数要先乘 tag_k，然后加 tag_b.(先乘后加)

线段树例题

```
//downtag(x)
void downtag(int x, int l, int r){
    if(tag_k[x] == 1 || tag_b[x] == 0) return;
    int mid = (l + r) / 2;
    int len_l = mid - l + 1;
    int len_r = r - mid;
    tag_k[x*2]=tag_k[x*2]*tag_k[x];
    tag_b[x*2]=tag_b[x*2]*tag_k[x]+tag_b[x];
    f[x*2]=f[x*2]*tag_k[x]+tag_b[x]*len_l;
    tag_k[x*2+1]=tag_k[x*2+1]*tag_k[x];
    tag_b[x*2+1]=tag_b[x*2+1]*tag_k[x]+tag_b[x];
    f[x*2+1]=f[x*2+1]*tag_k[x]+tag_b[x]*len_r;
    tag_k[x] = 1;
    tag_b[x] = 0;
}
```

总结

- 树状数组可以处理单点改区间查，区间改单点查。(这里的区间是指前缀，所以所维护的东西必须满足区间减法)
- 线段树可以支持区间改区间查，而且这里的区间只要满足区间加法即可。
- 总的来说，线段树完胜树状数组。
  但树状数组代码简单，常数小。