动态 dp

用于解决一些 dp 递推式简短但需要支持修改的问题，时间复杂度为 $O(nk^3\log n)$，其中 k 为递推式的相关项。

例题一

SP1716

题意

给定长度为 n 的序列，接下来两种操作：

1. 单点修改
2. 查询区间 $[l,r]$ 的所有连续子序列中的元素和的最大值

题解

首先考虑如何进行 dp 递推，设 $f(i)$ 表示所有 $[1,i]$ 的后缀的最大元素和，$g(i)$ 表示所有 $[1,i]$ 的连续子序列的最大元素和。

$$ f(i)=\max(f(i-1)+a_i,a_i),g_i=\max(g(i-1),f(i))=\max(f(i-1)+a_i,g(i-1),a_i) $$

考虑用广义矩阵乘法维护转移，线段树要求广义矩阵乘法需要满足结合律。

$$ \begin{equation}\begin{split} (ABC)_{i,j}&=\sum_{k=1}^n (AB)_{i,k}\oplus C_{k,j} \\ &=\sum_{k=1}^n \left(\left(\sum_{t=1}^nA_{i,t}\oplus B_{t,k}\right)\oplus C_{k,j}\right)\\ &=\sum_{k=1}^n\sum_{t=1}^nA_{i,t}\oplus B_{t,k}\oplus C_{k,j}\\ &=\sum_{t=1}^n\left(A_{i,t}\oplus\left(\sum_{k=1}^nB_{t,k}\oplus C_{k,j}\right)\right)\\ &=\sum_{t=1}^nA_{i,t}\oplus (BC)_{t,j}\\ \end{split}\end{equation} $$ 不难发现，为了使得上式成立，应该有 $a\oplus\sum_{i=1}^n b_i=\sum_{i=1}^n a\oplus b_i$。

普通矩阵乘法中 $\sum$ 表示求和，$\oplus$ 表示乘法。本题可以用 $\max$ 代替 $\sum$，$+$ 代替 $\oplus$。

于是有

$$ \begin{bmatrix}a_i & -\infty & a_i \\a_i & 0 & a_i \\-\infty & -\infty & 0 \\ \end{bmatrix} \begin{bmatrix}f(i-1)\\g(i-1)\\0\end{bmatrix} =\begin{bmatrix}f(i)\\g(i)\\0\end{bmatrix} $$

注意查询的初始值和正常 dp 相同

$$ \begin{bmatrix}f(0)\\g(0)\\0\end{bmatrix} =\begin{bmatrix}0\\-\infty\\0\end{bmatrix} $$

```
const int MAXN=5e4+5,Inf=1e9;
struct Matrix{
```

```
    int val[3][3];
    Matrix(){
        _for(i,0,3)_for(j,0,3)
        val[i][j]=-Inf;
    }
    Matrix(int a){
        val[0][0]=val[0][2]=val[1][0]=val[1][2]=a;
        val[0][1]=val[2][0]=val[2][1]=-Inf;
        val[1][1]=val[2][2]=0;
    }
    Matrix operator * (const Matrix &b)const{
        Matrix c;
        _for(i,0,3)_for(j,0,3)_for(k,0,3)
        c.val[i][j]=max(c.val[i][j],val[i][k]+b.val[k][j]);
        return c;
    }
}s[MAXN<<2];
int a[MAXN],lef[MAXN<<2],rig[MAXN<<2];
void push_up(int k){
    s[k]=s[k<<1]*s[k<<1|1];
}
void build(int k,int L,int R){
    lef[k]=L,rig[k]=R;
    int M=L+R>>1;
    if(L==R){
        s[k]=Matrix(a[M]);
        return;
    }
    build(k<<1,L,M);
    build(k<<1|1,M+1,R);
    push_up(k);
}
void update(int k,int pos,int v){
    if(lef[k]==rig[k]){
        s[k]=Matrix(v);
        return;
    }
    int mid=lef[k]+rig[k]>>1;
    if(mid>=pos)
    update(k<<1,pos,v);
    else
    update(k<<1|1,pos,v);
    push_up(k);
}
Matrix query(int k,int L,int R){
    if(L<=lef[k]&&rig[k]<=R)return s[k];
    int mid=lef[k]+rig[k]>>1;
    if(mid>=R)return query(k<<1,L,R);
    else if(mid<L)return query(k<<1|1,L,R);
    else
```

```
    return query(k<<1,L,R)*query(k<<1|1,L,R);
}
int main()
{
    int n=read_int();
    _rep(i,1,n)a[i]=read_int();
    build(1,1,n);
    int q=read_int();
    while(q--){
        int type=read_int(),x=read_int(),y=read_int();
        if(type==0)
        update(1,x,y);
        else{
            Matrix p=query(1,x,y);
            enter(max(p.val[1][0],p.val[1][2]));
        }
    }
    return 0;
}
```