

# 动态 dp

用于解决一些  $\text{dp}$  递推式简单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

CF1380F

### 题意

给定一个长度为  $n$  的十进制数  $D$

设  $a_i$  表示  $A$  从低到高的第  $i$  位，其他定义类同  $A+B$  表示字符串  $\{a_n+b_n \dots a_2+b_2, a_1+b_1\}$

例如  $3248+908=\{3+0, 2+9, 4+0, 8+8\}=\{3, 11, 4, 16\}=311416$ 。

接下来  $q$  次修改，每次修改  $D$  的某一位，问每次修改后有多少组  $(A, B)$  满足  $A+B=D$

### 题解

从低位到高位考虑  $\text{dp}$  设  $\text{dp}(i)$  表示  $A+B=D[1, i]$  的方案数。

若  $d_i$  不是通过进位得到的，则有  $\text{dp}(i) = \text{dp}(i-1)$

若  $d_i$  是通过进位得到的，则有  $\text{dp}(i) = \sum_{10 \leq d_{i-1} \leq 18} (19 - d_{i-1}) \text{dp}(i-2)$

于是有状态转移方程

$$\begin{bmatrix} d_{i+1} & [10 \leq d_{i-1} \leq 18] (19 - d_{i-1}) \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \text{dp}(i-1) \\ \text{dp}(i-2) \end{bmatrix} = \begin{bmatrix} \text{dp}(i) \\ \text{dp}(i-1) \end{bmatrix}$$

注意每次更新需要更新  $i, i+1$  两个位置的矩阵，另外把  $d_0$  设置成  $19$  防止影响位置  $1$  的矩阵。另外初始值为

$$\begin{bmatrix} \text{dp}(0) \\ \text{dp}(-1) \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

最后还有一点：矩阵乘法不满足交换律，而我们是从左往右  $\text{dp}$  所以线段树  $\text{push\_up}$  的时候需要用右区间矩阵乘上左区间矩阵。

总时间复杂度  $O(q \log n)$

```
const int MAXN=5e5+5, Mod=998244353;
```

```
struct Matrix{
    int val[2][2];
    Matrix(){
        _for(i,0,2)_for(j,0,2)
            val[i][j]=0;
    }
    Matrix(int a,int b){
        val[0][0]=a+1;
        val[0][1]=(10<=b&&b<=18)?19-b:0;
        val[1][0]=1;
        val[1][1]=0;
    }
    Matrix operator * (const Matrix &b) const{
        Matrix c;
        _for(i,0,2)_for(j,0,2)_for(k,0,2)
            c.val[i][j]=(c.val[i][j]+1LL*val[i][k]*b.val[k][j])%Mod;
        return c;
    }
}s[MAXN<<2];
int a[MAXN],lef[MAXN<<2],rig[MAXN<<2];
char buf[MAXN];
void push_up(int k){
    s[k]=s[k<<1|1]*s[k<<1];//注意别写反了
}
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],a[M]*10+a[M-1]);
        return;
    }
    build(k<<1,L,M);
    build(k<<1|1,M+1,R);
    push_up(k);
}
void update(int k,int pos){
    if(lef[k]==rig[k]){
        s[k]=Matrix(a[pos],a[pos]*10+a[pos-1]);
        return;
    }
    int mid=lef[k]+rig[k]>>1;
    if(mid>=pos)
        update(k<<1,pos);
    else
        update(k<<1|1,pos);
    push_up(k);
}
int main()
{
    int n=read_int(),q=read_int();
```

```

scanf("%s",buf+1);
a[0]=19;
_rep(i,1,n)a[i]=buf[n+1-i]-'0';
build(1,1,n);
while(q--){
    int pos=n+1-read_int(),d=read_int();
    a[pos]=d;
    update(1,pos);
    if(pos<n)update(1,pos+1);
    enter((s[1].val[0][0]+s[1].val[0][1])%Mod);
}
return 0;
}

```

## 例题二

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  $\text{dp}$  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。

$$\begin{aligned} (ABC)_{i,j} &= \sum_{k=1}^n (AB)_{i,k} \oplus C_{k,j} \\ &= \sum_{k=1}^n \left( \left( \sum_{t=1}^n A_{i,t} \oplus B_{t,k} \right) \oplus C_{k,j} \right) \\ &= \sum_{k=1}^n \sum_{t=1}^n A_{i,t} \oplus B_{t,k} \oplus C_{k,j} \\ &= \sum_{t=1}^n \left( A_{i,t} \oplus \left( \sum_{k=1}^n B_{t,k} \oplus C_{k,j} \right) \right) \\ &= \sum_{t=1}^n A_{i,t} \oplus (BC)_{t,j} \end{aligned}$$

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  $\text{dp}$  相同

$\begin{matrix} f(0) \\ g(0) \end{matrix} = \begin{matrix} 0 \\ -\infty \end{matrix}$   
 $\text{或} \begin{matrix} -\infty \\ -\infty \end{matrix}$

ps. 这题从左往右  $\text{dp}$  和从右往左  $\text{dp}$  状态转移完全相同，所以  $\text{push\_up}$  写反也不影响正确性，但要注意  $\text{query}$  和  $\text{push\_up}$  的一致性。

```
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|1]*s[k<<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|1,L,R)*query(k<<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;
}

```

From:  
<https://wiki.cvbbacm.com/> - CVBB ACM Team

Permanent link:  
[https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:legal\\_string:jxm2001:%E5%8A%A8%E6%80%81dp&rev=1613701374](https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:legal_string:jxm2001:%E5%8A%A8%E6%80%81dp&rev=1613701374)

Last update: 2021/02/19 10:22