[学习笔记] 子集卷积相关

作者
UltiMadow

发布时间
2021-08-01 20:43

分类
算法·理论

- 高维前缀和 & sosdp

对于二维前缀和，我们一般有两种计算方法

第一种可以使用容斥，$s_{i,j}=s_{i-1,j}+s_{i,j-1}-s_{i-1,j-1}+a_{i,j}$

第二种，我们可以先选一个维度进行前缀和：$s_{i,j}=s_{i,j-1}+a_{i,j}$，接下来，再对第二个维度进行前缀和：$s_{i,j}\leftarrow s_{i-1,j}+s_{i,j}$

但对于更高维的情况，容斥就不适用了，但是可以用第二种方法，对每个维度依次做前缀和

用高维前缀和的思想，我们可以解决子集 dp 的问题

设 $f_s=\sum\limits_{t\subseteq s}a_t$，则我们可以看做对 s 做了 n 维前缀和，便可以在 $\mathcal{O}(n2^n)$ 的时间内解决这个问题

code:

```
for(int i=1;i<=n;i++)
    for(int s=0;s<(1<<n);s++)
        if(s&(1<<(i-1)))
            f[s]=(f[s]+f[s^(1<<(i-1))])%p;
```

- 快速莫比乌斯变换（FMT）

$\sum\limits_{j or k=i} a_j b_k$

我们对 a_i 做一遍 n 维前缀和，$f_i=\sum\limits_{j\subseteq i}a_j$，得到 a 的 FMT，同样对 b_i 做一遍 n 维

前缀和，得到 b 的 FMT g_i

接下来把 f 和 g 点乘起来，$h_i = f_i g_i$

考虑 h 和 c 的关系：$h_i = f_i g_i = (\sum\limits_{j \subseteq i} a_j)(\sum\limits_{k \subseteq i} b_k) = \sum\limits_{j,k \subseteq i} a_j b_k = \sum\limits_{j \operatorname{or} k \subseteq i} a_j b_k$，这便是 c_i 的 n 维前缀和

最后把 h 做一遍 n 维差分就可以得到 c

时间复杂度 $\mathcal{O}(n2^n)$

对于 and 卷积，我们只要把 n 维前缀和换成 n 维后缀和即可

- 快速沃尔什变换（FWT）

FWT 除了可以解决 or 卷积和 and 卷积之外，还可以解决 xor 卷积

考虑是否存在一种类似 FFT 的变换，能把卷积变成点积

对于 or，考虑一种变换：

$$\operatorname{FWT}(A) = \begin{cases} A & (n = 0) \\ \operatorname{FWT}(A_0), \operatorname{FWT}(A_1 + A_0) & (n > 0) \end{cases}$$

其中，A_0 表示第 n 位为 0 的序列，A_1 表示第 n 位为 1 的序列，$+$ 为两个序列按顺序分别相加

这样，我们就得到了 a 的 FWT f，考虑 f 和 a 的关系

发现序列任何一位如果是 1 都加上了那一位是 0 的值，也就是我们得到了 $f_i = \sum\limits_{j \subseteq i} a_j$，和 FMT 一样

于是我们把 a 的 FWT f，b 的 FWT g 点乘起来，得到的就是 c 的 FWT h

使用其逆变换：

$$\operatorname{IFWT}(A) = \begin{cases} A & (n = 0) \\ \operatorname{IFWT}(A_0), \operatorname{IFWT}(A_1 - A_0) & (n > 0) \end{cases}$$

就可以得到 c 了

时间复杂度 $\mathcal{O}(n2^n)$

code:

```
void FWT(int *A,int sgn){
    for(int mid=1;mid<(1<<n);mid<<=1)
        for(int j=0,R=(mid<<1);j<(1<<n);j+=R)
            for(int k=0;k<mid;k++){
                if(sgn==1)A[j+k+mid]=A[j+k+mid]+A[j+k];
                else A[j+k+mid]=A[j+k+mid]-A[j+k];
            }
}
```

同样的，对于 and 卷积，也有 FWT：

$$\mathrm{FWT}(A)=\begin{cases}A & (n=0)\\ \mathrm{FWT}(A_0+A_1),\mathrm{FWT}(A_0) & (n>0)\end{cases}$$

逆变换：

$$\mathrm{IFWT}(A)=\begin{cases}A & (n=0)\\ \mathrm{IFWT}(A_0-A_1),\mathrm{IFWT}(A_1) & (n>0)\end{cases}$$

code:

```
void FWT(int *A,int sgn){
    for(int mid=1;mid<(1<<n);mid<<=1)
        for(int j=0,R=(mid<<1);j<(1<<n);j+=R)
            for(int k=0;k<mid;k++){
                if(sgn==1)A[j+k]=A[j+k]+A[j+k+mid];
                else A[j+k]=A[j+k]-A[j+k+mid];
            }
}
```

此外，FWT 还可以做 xor 卷积

在 xor-FWT 之前，先有一个结论

记 $a\circ b$ 为 a and b 的二进制表示中 1 的位数（对 2 取模）

有性质 $(a\circ x)\ \mathrm{xor}\ (a\circ y)=a\circ(x\ \mathrm{xor}\ y)$

证明如下：

记 $f(x)$ 表示 x 的二进制表示中 1 的位数

原式等价于 $f(a\ \mathrm{and}\ x)+f(a\ \mathrm{and}\ y)\equiv f(a\ \mathrm{and}\ (x\ \mathrm{xor}\ y))\pmod 2$

接下来，对 a，x，y 的每一位大力分类讨论即可得证

构造一个变换：$f_i=\sum\limits_{i\circ j=0}a_j-\sum\limits_{i\circ j=1}a_j$

记 a 的变换为 f，b 的变换为 g，c 的变换为 h

$$
\begin{aligned}
f_ig_i&=(\sum_{i\circ j=0}a_j-\sum_{i\circ j=1}a_j)(\sum_{i\circ k=0}b_k-\sum_{i\circ k=1}b_k)\\
&=(\sum_{i\circ j=0}a_j\sum_{i\circ k=0}b_k+\sum_{i\circ j=1}a_j\sum_{i\circ k=1}b_k)-(\sum_{i\circ j=0}a_j\sum_{i\circ k=1}b_k+\sum_{i\circ j=1}a_j\sum_{i\circ k=0}\\
&=\sum_{i\circ(j\operatorname{xor}k)=0}a_jb_k-\sum_{i\circ(j\operatorname{xor}k)=1}a_jb_k\\
&=h_i
\end{aligned}
$$

（第 3 行用到了上面那个性质）

考虑变换：

$$
\mathrm{FWT}(A)=\begin{cases}A & (n=0)\\ \mathrm{FWT}(A_0+A_1),\mathrm{FWT}(A_0-A_1) & (n>0)\end{cases}
$$

考虑最开始说的 a 的变换 f

若 f_x 中 x 的第 i 位为 0，则无论 a_y 中 y 的第 i 位为 0 还是 1，$x\circ y$ 都不会改变，所以前面一部分的 a 的 FWT 和 f 是一致的

若 f_x 中 x 的第 i 位为 1，则 a_y 中 y 的第 i 位为 1 时 $x\circ y$ 不会改变，而为 0 时会改变，于是需要把为 0 的部分减掉，而上面式子中 $x\circ y$ 为 1 的系数为 -1，所以后面一部分 a 的 FWT 和 f 是一致的

于是得到 a 和 b 的 FWT f，g 之后就可以直接点乘得到 h 了

逆变换：

$$
\mathrm{IFWT}(A)=\begin{cases}A & (n=0)\\ \mathrm{IFWT}(\frac{A_0+A_1}{2}),\mathrm{IFWT}(\frac{A_0-A_1}{2}) & (n>0)\end{cases}
$$

code:

```
void FWT(int *A,int sgn){
    for(int mid=1;mid<(1<<n);mid<<=1)
```

```
            for(int j=0,R=(mid<<1);j<(1<<n);j+=R)
                for(int k=0;k<mid;k++){
                    int x=A[j+k],y=A[j+k+mid];
                    if(sgn==1)A[j+k]=x+y,A[j+k+mid]=x-y;
                    else A[j+k]=(x+y)/2,A[j+k+mid]=(x-y)/2;
                }
}
```

- 子集卷积

考虑这个式子：$c_i=\sum\limits_{j or k=i,j and k=\emptyset}a_jb_k$

直接枚举子集可以做到 $\mathcal{O}(3^n)$，不够优秀

如果没有 j and $k=\emptyset$ 的限制，则直接 or 卷积即可完成

考虑转换一下限制

设 $f(x)$ 为 x 的二进制表示中 1 的位数

则原限制等价于 j or $k=i$ 且 $f(j)+f(k)=f(j \text{ or } k)$

于是将 a 数组根据 $f(i)$ 拆开，$a'_{i,s}$ 仅当 $f(s)=i$ 时等于 a_s

同样将 b 数组拆开成 b'，c 数组拆开成 c'

则考虑进行如下卷积：$c'_{i,s}=\sum\limits_{j+k=i,p or q=s}a'_{j,p}b'_{k,q}$，得到的 c' 中 $f(s)=i$ 的即为答案

于是对于所有的 a'，b' 都做一遍 or-FWT，之后做卷积得到 c' 的 FWT，再都做一遍 or-IFWT，得到 c'，就可以将答案提取出来了

时间复杂度 $\mathcal{O}(n^2 2^n)$

code:

```
for(int i=0;i<(1<<n);i++)scanf("%d",&a[pcnt[i]][i]);
for(int i=0;i<(1<<n);i++)scanf("%d",&b[pcnt[i]][i]);
for(int i=0;i<=n;i++)FWT(a[i],1),FWT(b[i],1);
for(int i=0;i<=n;i++)
    for(int j=0;j<(1<<n);j++)
        for(int k=0;k<=i;k++)
            c[i][j]=(c[i][j]+1ll*a[k][j]*b[i-k][j]%p)%p;
for(int i=0;i<=n;i++)FWT(c[i],-1);
```

```
for(int i=0;i<(1<<n);i++)printf("%d ",c[pcnt[i]][i]);
//pcnt[i]为f(i)
```

- 例题

P4717 【模板】快速莫比乌斯/沃尔什变换 (FMT/FWT)

P6097 【模板】子集卷积

↑ 板子题 ↑

CF1034E Little C Loves 3 III

> 题意：在模 4 意义下求子集卷积
> $n \le 21$

直接子集卷积是 $\mathcal{O}(n^2 2^n)$ 的，不能通过此题，所以需要进行一些转化

记 $f(x)$ 为 x 的二进制表示中 1 的位数

考虑设 $f_i = a_i \times 4^{f(i)}, g_i = b_i \times 4^{f(i)}$

接下来直接对 f 和 g 做 or 卷积，得到 $h_i = \sum\limits_{j or k} a_j b_k \times 4^{f(j)+f(k)}$

接下来把 h_i 除以 $4^{f(i)}$ 并对 4 取模，就把 $f(j)+f(k) \neq f(i)$ 的不合法情况去掉了（除以 $4^{f(i)}$ 去掉了 $f(j)+f(k) < f(i)$ 的情况，对 4 取模去掉了 $f(j)+f(k) > f(i)$ 的情况，而 $f(j)+f(k) = f(i)$ 的情况正好留下来了并且对 4 取了模）

那么时间复杂度 $\mathcal{O}(n2^n)$

CF1326F2 Wise Men (Hard Version)

> 题意：给定一个 n 个点的无向图，一个 n 个点的排列 p 能生成的 01 串的第 i 位为 p_i 和 p_{i+1} 的连边情况，有边为 0，无边为 1
> 对于每个 01 串，求有多少种排列可以生成它
> $2 \le n \le 18$

考虑对于每个 01 串对排列的限制：第 i 位为 0 要求必须无边，为 1 要求必须有边

显然有边的限制比无边的要好处理一点，于是可以考虑去掉无边的限制

去掉必须无边的限制之后，我们发现答案其实求解的是一个集合 s 的高维后缀和，于是我们得到去掉限制的答案之后直接 and-IFWT/and-IFMT 就可以得到原来的答案了

如果只考虑必须有边，原题就被转化成了原图拆成若干条点不相交的链，对每个 01 串的贡献

发现这就是一个整数拆分问题，可以暴力 dfs 求解

记 $f(x)$ 为 x 的二进制表示中 1 的个数

设 f_s 为 s 集合中的点组成一条链的方案数，可以用 dp 在 $\mathcal{O}(n^2 2^n)$ 的时间内预处理出来

在整数拆分时，我们维护 g_s 为 s 集合中的点组成若干条链，链的长度为当前 dfs 的整数拆分的方案数

我们发现 g 在每次转移的时候要和 f 做一次子集卷积，很慢

我们考虑在预处理时把 f_s 拆成 $f_{f(s),s}$，然后预先对 f_i 跑一遍 or-FWT，这样每次转移 g 的时候只需要把 f_i 点乘进去就行了（这里 i 表示当前 dfs 的整数拆分）

这样计算出的 g 其实是 g 的 or-FWT，所以需要进行一次 or-IFWT，但是这样还是太慢了

发现计算时只需要全集的答案 g_U，而 or-FWT 其实是做了高维前缀和，所以对 g_U 做一次高维差分即可

这里算出的 g 是对于每个整数拆分的答案，我们可以把它用 hashmap 建立一个整数拆分到对应答案的映射

最后对于每个 01 串，都有一个对应的整数拆分序列，于是直接在 hashmap 里查询出答案即可

最后别忘了做一遍 and-IFWT

时间复杂度 $\mathcal{O}(\sum\limits_{i\in p(n)} g(i)\times 2^n)$

其中 $p(n)$ 为 n 的整数拆分集合，$g(i)$ 为当前整数拆分方案拆出来数的个数

作者：UltiMadow　创建时间：2021-08-01 20:43:02

本文章已投稿至 洛谷日报