Solution 4

1、计算FFT算法的数据写入复杂度，FFT算法如下：

$$
\begin{aligned}
&\textbf{Algorithm: Fast Fourier Transform}\\
&\textbf{FFT}([a_0,\cdots,a_{N-1}],\omega,N)\\
&\quad \textbf{if } N=1 \textbf{ then return}[a_0]\\
&\quad F_{even}\leftarrow \textbf{FFT}([a_0,a_2,\cdots,a_{N-2}],\omega^2,N/2)\\
&\quad F_{odd}\leftarrow \textbf{FFT}([a_1,a_3,\cdots,a_{N-1}],\omega^2,N/2)\\
&\quad F\leftarrow \text{ a new vector of length } N\\
&\quad x\leftarrow 1\\
&\quad \textbf{for } j=0 \text{ to } N-1 \textbf{ do}\\
&\quad\quad F[j]\leftarrow F_{even}[j \bmod (N/2)]+xF_{odd}[j \bmod (N/2)]\\
&\quad\quad x\leftarrow x*\omega\\
&\quad \textbf{return } F
\end{aligned}
$$

前$\log\frac{N}{B}$层，每层cache miss为$\frac{N}{B}$，总cache miss$=\frac{N}{B}\log\frac{N}{B}$；后$\log B$层，每一小块大小$<B$，因此每一个小块都是一次cache miss，总cache miss$=\frac{N}{B}\cdot 2+\frac{N}{B}\cdot 4+\cdots+N\approx 2N$。所有层cache miss次数$=O(\frac{N}{B}\log\frac{N}{B}+N)$。

2、你在网上找到了一张很好看的图片并想把它设为自己的桌面，可惜因为拍摄原因，这张图片有一点运动模糊，你希望还原一张未模糊的图片。设未模糊的图片为向量v，经过和向量w的卷积得到了模糊的向量$q=v*w=F^{-1}(F(v)\times F(w))$，假设你知道模糊后的向量$q$和向量$w$，尝试还原向量$v$。（你可以假设计算过程中不会出现除0的问题）

将q和w转化为点表示法，则v的点表示法对应的点=q的点表示法对应的点$/w$的点表示法对应的点。$v=q/w=F^{-1}(F(q)/F(w))$。

3、多项式的减法卷积的定义是：

$$c_n=\sum_{i-j=n}a_ib_j$$

其中，a_i,b_j,c_n分别为多项式A,B,C的系数。与加法卷积唯一的区别就是加法卷积中a_ib_j的乘积会更新c_{i+j}，而减法卷积中a_ib_j，$(i\ge j)$的乘积会更新c_{i-j}。如果把加法卷积理解为把一个取i个物品的方案数为a_i的背包a和另一个取j个物品的方案数为b_j的背包b的合并得到新的一个背包c，那么减法卷积可以理解为从一个取j个物品的方案数为b_j的背包b中取出一些物品放入一个剩余容量是i的方案数为a_i的背包a得到一个新的背包c。减法卷积的求解同样可以使用**FFT**，尝试给出一种算法用$O(n\log n)$的时间求出两个n阶多项式的减法卷积。

令$d_n=b_{N-n-1},e=a*d$，则有$c_n=\sum_{i-j=n}a_ib_j=\sum_{i+j=N+n-1}a_id_j=e_{N+n-1}$。

1、reverse b

2、计算$c=a*b$

3、移除c的前$N-1$项

4、return c

```
vector<ll> mulT(vector<ll> a, vector<ll> b) {
    if (b.empty()) return {};
    int n = b.size();
    reverse(b.begin(), b.end());
    auto res = mul(a, b);
    res.erase(res.begin(), res.begin() + n - 1);
    return res;
}
```

4、给定一个正整数序列$[a_1, a_2, \cdots, a_n]$，对于任意$1 \le x \le \sum_{i=1}^{n} a_i$，求出有多少连续子序列满足区间和$a_l + a_{l+1} + \cdots + a_r = x$。你的算法需要运行在$O(\sum a_i \log \sum a_i)$的时间以内。

对a_i做前缀和得到sum_i，再对sum_i求cnt数组，$cnt[sum_i]$表示大小为sum_i的前缀和出现的次数。接着对cnt自己对自己做减法卷积，卷积的结果f_i就表示区间和为i的区间个数。

证明：区间和为x的区间个数num_x为满足$sum_r - sum_{l-1} = x, l \le r$的$(l, r)$对数。第一部分的限制就是减法卷积的形式，而第二部分限制因为$a_i > 0$，保证$sum_r - sum_{l-1} > 0$时，r一定大于$l-1$，即$l \le r$。因此直接对cnt本身做减法卷积 就能得到区间和大于 0 的所有结果。

5、一天，你在做算法题的时候遇到了一个问题：给定两个字符串S和T，若S的区间$[l, r]$子串与T完全相同，称T在S中出现了，其出现的位置为l。你很快就意识到了这是一道**KMP**算法的模板题，但不幸的是你忘记了怎么写**KMP**算法。好消息是，你发现字符串的长度不是很大，$O(|S| \log |S|)$的算法也可以轻松通过此题，你想起了自己刚刚学过的**FFT**正好是$O(n \log n)$，于是你打算用**FFT**来解决这个问题。

(a) 给出一个$O(|S| \log |S|)$的算法，求出所有T在S中所有出现的位置。

S的第l个位置能匹配T当且仅当$\sum_{i=0}^{|T|-1} (S_{i+l} - T_i)^2 = 0$。把这个式子拆开得到$\sum_{i=0}^{|T|-1} (S_{i+l}^2 + T_i^2 - 2S_{i+l}T_i)$，其中前2项为平方和的求和，可以用$O(|S|)$的时间预处理，$O(1)$查询。最后一项为减法卷积的形式，**FFT** $O(|S| \log |S|)$求解即可。

(b) 你使用**FFT**轻松地解决了这个问题，于是你打算寻求一些挑战。假设字符串S和T中存在一些通配符（通配符可以匹配任何字符），如何用$O(|S| \log |S|)$的时间求出所有T在S中所有出现的位置。

把通配符的位置看作0，则S的第l个位置能匹配T当且仅当$\sum_{i=0}^{|T|-1} S_{i+l} T_i (S_{i+l} - T_i)^2 = 0$，拆开得到$\sum_{i=0}^{|T|-1} (S_{i+l}^3 T_i + S_{i+l} T_i^3 - 2S_{i+l}^2 T_i^2)$。三项都是减法卷积的形式，做三遍**FFT** $O(|S| \log |S|)$求解。

6*（**挑战问题**：可2-3位同学组队回答，但在提交的答案中需写明每位同学的**具体**贡献）、**Cache-Oblivious FFT**算法

(a) **Cache-Oblivious FFT**算法中用到了矩阵转置，课堂上假设矩阵转置的数据搬运复杂度可以做到$O(\frac{nm}{B})$，但并未给出具体做法。

最直接的矩阵转置的数据搬运复杂度是$O(nm)$。假设矩阵是行优先存储，矩阵的形状是$m \times n$，每次取原矩阵的一行写入目的矩阵的一列。取出一行的cache miss次数是$\frac{n}{B}$，一共有m行，总次数$\frac{nm}{B}$，但写入一列时每一行都是不同的cache line，于是每一次写都是一次cache miss，总次数nm。

给出一种Cache-Oblivious（算法中不包含cache大小M和cache line大小B）的矩阵转置算法，保证其数据搬运复杂度为$O(\frac{nm}{B})$。

算法：每次取$\max\{n, m\}$，将矩阵一分为二，分别进行转置。

```
void transpose(int offsetM, int offsetN, int m, int n, int a[M][N], int b[N][M])
{
    if (n == 1 && m == 1) b[offsetN][offsetM] = a[offsetM][offsetN];
    if (n >= m) {
        transpose(offsetM, offsetN, m, n / 2, a, b);
        transpose(offsetM, offsetN + n / 2, m, n / 2, a, b);
    } else {
        transpose(offsetM, offsetN, m / 2, n, a, b);
        transpose(offsetM + m / 2, offsetM, m / 2, n, a, b);
    }
}
```

存在递归到某层使得$m'n' \leq \varepsilon M$，$2m'n' > \varepsilon M$。每一块cache miss次数$\frac{m'n'}{B} \leq \frac{\varepsilon M}{B}$。一共最多$\frac{mn}{m'n'} < \frac{2mn}{\varepsilon M}$块。总cache miss次数$\leq \frac{\varepsilon M}{B} \cdot \frac{2mn}{\varepsilon M} = \frac{2mn}{B} = O(\frac{mn}{B})$。

(b) 算法中设定$X : N_1 \times N_2$；$Y : N_2 \times N_1$，是否可以设定Y同样为$N_1 \times N_2$使得Y和X有相同的layout来避免转置呢？请说明原因。

$$y[k_2N_2 + k_1] = \sum_{n=0}^{N-1} x[n_1N_2 + n_2] \cdot \omega_N^{-(k_2N_2+k_1)(n_1N_2+n_2)}$$

$$\omega_N^{-(k_2N_2+k_1)(n_1N_2+n_2)} = \omega_N^{-k_2n_1N_2^2-k_2n_2N_2-k_1n_1N_2-k_1n_2} = \omega_{N_1}^{-k_2n_1N_2-k_2n_2-k_1n_1} \cdot \omega_N^{-k_1n_2}$$

$$y[k_2 \cdot N_2 + k_1] = \sum_{n_2=0}^{N_2-1}((\sum_{n_1=0}^{N_1-1} x[n_1N_2 + n_2] \cdot \omega_{N_1}^{-k_2n_1N_2-k_1n_1})\omega_N^{-k_1n_2})\omega_{N_1}^{-k_2n_2}$$

外层循环的基不对。

(c) **Cache-Oblivious FFT**的y计算公式是：

$$y[k_2 \cdot N_1 + k_1] = \sum_{n_2=0}^{N_2-1}((\sum_{n_1=0}^{N_1-1} x[n_1N_2 + n_2] \cdot \omega_{N_1}^{-k_1n_1})\omega_N^{-k_1n_2}) \cdot \omega_{N_2}^{-k_2n_2}$$

其中使用了3次转置，带来了很大的开销。是否可以交换求和的顺序来减少转置的次数，即

$$y[k_2 \cdot N_1 + k_1] = \sum_{n_1=0}^{N_1-1}(\sum_{n_2=0}^{N_2-1} x[n_1N_2 + n_2] \cdot \omega_{N_2}^{-k_2n_2} \cdot \omega_N^{-k_1n_2}) \cdot \omega_{N_1}^{-k_1n_1}$$

说明这种做法可能带来的问题。

第一种计算方法，在乘twiddle factor的时候x对应不同k_1是独立的，因此可以每个k_1对应一列乘进去；

第二种计算方法，twiddle factor与n_2有关，不能在第一次卷积计算完成后乘。在做第一次卷积之前，每个k_1无法对应到某个n_1，因此无法计算。