CSE548, AMS542: Analysis of Algorithms, Spring 2015 **Date:** May 1

Homework #4

(Due: May 10)

Task 1. [100 Points] Copying Files

Suppose you have m files $f_1, f_2, \ldots, f_m$ stored on a storage device $\mathcal{D}_{in}$, and you want to copy as many of them as possible to a hard disk $\mathcal{D}_{out}$ of size n (in bytes). You either copy an entire file or completely skip it. Partial copies are not allowed.

(*a*) [**35 Points**] Figure 1 shows an algorithm Copy-from-Hard-Disk that copies the files from $\mathcal{D}_{in}$ to $\mathcal{D}_{out}$ assuming that $\mathcal{D}_{in}$ allows both reads and writes. Prove that Copy-from-Hard-Disk achieves an approximation ratio of 2, i.e., if an optimal algorithm can copy n_{opt} bytes of files to $\mathcal{D}_{out}$, Copy-from-Hard-Disk copies at least $\frac{1}{2}n_{opt}$ bytes.

(*b*) [**10 Points**] Give an example for which Copy-from-Hard-Disk, indeed, approaches an approximation ratio of 2 as n gets large.

Copy-from-Hard-Disk(f_1, f_2, $\ldots$, f_m, n)

(Inputs are m files $f_1, f_2, \ldots, f_m$ stored on a standard hard disk $\mathcal{D}_{in}$ that allows both reads and writes. This function copies as many of those files as it can to another hard disk $\mathcal{D}_{out}$ of size n (in bytes).)

Sort the given files on $\mathcal{D}_{in}$ in nonincreasing order of size. Let $f'_1, f'_2, \ldots, f'_m$ be the resulting sorted sequence.

$n' \leftarrow n$

for $i \leftarrow 1$ ***to*** m ***do***

    if *size*(f'_i) $\leq n'$ ***then*** {*check file size in bytes*}

        copy f'_i to $\mathcal{D}_{out}$

        $n' \leftarrow n' - size(f'_i)$

Figure 1: Copy files from a disk that allows both reads and writes.

(*c*) [**35 Points**] Figure 2 shows an algorithm Copy-from-CD-ROM that copies the files from $\mathcal{D}_{in}$ to $\mathcal{D}_{out}$ assuming that $\mathcal{D}_{in}$ allows only reads (and no writes). Observe that you can no longer sort the files on $\mathcal{D}_{in}$ as it does not allow writes. Prove that Copy-from-CD-ROM achieves an approximation ratio of 2, i.e., if an optimal algorithm can copy n_{opt} bytes of files to $\mathcal{D}_{out}$, Copy-from-CD-ROM copies at least $\frac{1}{2}n_{opt}$ bytes.

COPY-FROM-CD-ROM($f_1,\ f_2,\ \ldots,\ f_m,\ n$)

(Inputs are m files $f_1, f_2, \ldots, f_m$ stored on a CD drive $\mathcal{D}_{in}$ that allows only reads (and no writes). This function copies as many of those files as it can to another hard disk $\mathcal{D}_{out}$ of size n (in bytes).)

```
n' ← n
for k ← 1 to ⌈log_2 n⌉ do
    for i ← 1 to m do
        if size( f_i ) ≥ n/2^k and size( f_i ) ≤ n' then        {check file size in bytes}
            copy f_i to D_out
            n' ← n' − size( f_i )
```

Figure 2: Copy files from a CD ROM.

(d) [**20 Points**] Argue that the COPY-FROM-CD-ROM algorithm in Figure 2 runs in $\mathcal{O}(m \log n + n)$ time. Devise an algorithm that runs in $\mathcal{O}(m + n)$ time but still achieves the same approximation ratio.

Task 2. [80 Points] Derangement Numbers

Consider the sequence of the first $k \geq 1$ consecutive positive integers: $1, 2, 3, \ldots, k$. The *Derangement Number* D_k is the number ways one can permutate those k integers such that for each $i \in [1, k]$, in none of the resulting permutations integer i appears at location i.

Starting from D_1 the first 10 derangement numbers are as follows: 0, 1, 2, 9, 44, 265, 1854, 14833, 133496 and 1334961. As an example, all $D_4 = 9$ derangements of $\langle 1, 2, 3, 4 \rangle$ are shown below:

$$\begin{bmatrix} 2 & 1 & 4 & 3 \\ 2 & 3 & 4 & 1 \\ 2 & 4 & 1 & 3 \\ 3 & 1 & 4 & 2 \\ 3 & 4 & 1 & 2 \\ 3 & 4 & 2 & 1 \\ 4 & 1 & 2 & 3 \\ 4 & 3 & 1 & 2 \\ 4 & 3 & 2 & 1 \end{bmatrix}$$

Derangement numbers can be computed from the following recurrence:

$$D_k = \begin{cases} 1 & \text{if } k = 0, \\ 0 & \text{if } k = 1, \\ (k-1)\, D_{k-1} + (k-1)\, D_{k-2} & \text{otherwise.} \end{cases}$$

Describe a parallel algorithm that computes the first n derangement numbers in $\mathcal{O}\left(\frac{n}{p} + \log n\right)$ parallel time using $\Theta\left(n\right)$ space, where p is the number of processing elements[1].

[1] Please assume that the span of a ***parallel for*** loop is $\mathcal{O}\left(()\, 1 + k\right)$, where k is the maximum span of a single iteration of the loop.