C-0 — Cryptography

Objectives:
- Introduce cryptography (symmetric and asymmetric) and cryptanalysis.
- Introduce substitution, shift, and affine ciphers and explain their mathematical foundations ($\mathbb{Z}_m$, Euclidean algorithm).
- Explain brute force attacks and the letter frequency analysis attack on substitution, shift, and affine ciphers.

Comments:
- It is crucial that students understand the importance of Kerckhoffs' principle.

Outcomes:
- 3.1.2 Explain substitution, shift, and affine ciphers.
- 3.1.3 Understand mathematical foundations.
- 3.1.4 Determine plaintext from given ciphertext using letter frequency attack.

C-23 — Cryptanalysis – Letter Frequency Analysis

Example:
Ciphertext: "iq ifcc vqqr fb rdq vfllcq na rdq cfjwhwz hr bnnb hcc hwwbsqvbre hwq vhqlq"

Table of letter counts in our ciphertext:

<table>
<thead>
<tr>
<th>Letter</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
<th>h</th>
<th>i</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>1</td>
<td>5</td>
<td>6</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>7</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Letter</th>
<th>j</th>
<th>k</th>
<th>l</th>
<th>m</th>
<th>n</th>
<th>o</th>
<th>p</th>
<th>q</th>
<th>r</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Letter</th>
<th>s</th>
<th>t</th>
<th>u</th>
<th>v</th>
<th>w</th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Frequencies help to determine likely substitution candidates.
Frequency analysis can be generalized to by looking at pairs, triples (and so on) of ciphertext symbols.
Frequently used short words ("AND""," THE", . . .) can be spotted easily.
The analysis of the ciphertext reveals the plaintext:
"WE WILL MEET IN THE MIDDLE OF THE LIBRARY AT NOON ALL ARRANGEMENTS ARE MADE"

M-1 — Multithreaded Algorithms – 1st Attempt

1st attempt for a parallel letter frequency algorithm.
Divide the input into two parts of equal size $\frac{n}{2}$.

Algorithm (LETTER-FREQ(t,n))

1. Let $c[0..25]$ and $f[0..25]$ be a new arrays
2. for ($0 \leq i \leq 25$) do
   3. $c[i] = 0$
4. /* Compute in parallel */
5. /* thread 1 */
6. for ($1 \leq i \leq \lfloor \frac{n}{2} \rfloor$) do
   7. $c[t[i]] = c[t[i]] + 1$
8. /* thread 2 */
9. for ($\lceil \frac{n}{2} \rceil + 1 \leq i \leq n$) do
   10. $c[t[i]] = c[t[i]] + 1$
11. barrier.n.wait
12. return $f$

M-2 — Multithreaded Algorithms – 1st Attempt

Synchronize threads:
- We use a barrier to synchronize a given number of threads.
- A barrier that synchronizes $n$ threads is denoted with barrier$n$.
- The instruction barrier$n$.wait lets a thread wait until the given number of $n$ threads has reached the "waiting point".
- All threads continue as soon as the $n$-th thread reached this "waiting point".
- Different barriers are denoted by barrier$i$.

Correctness:
- Counters are initialized before parallel computation start.
- Frequencies are computed after the threads have been synchronized.
- Problem: counter updates in line (8) could cause problems.
Letter frequency for 2 threads (4th attempt):

```cpp
boost::mutex lf_lock[26];
boost::barrier lf_barrier(2);

void letter_freq_p(char* text, long* freq, long s, long e)
{
    int j;
    for (long i = s; i < e; i++)
    {
        j = text[i] - 'a';
        lf_lock[j].lock();
        freq[j]++;
        lf_lock[j].unlock();
    }
    lf_barrier.wait();
}

void letter_freq(char* text, long length, long* freq)
{
    boost::thread* thrds[2];
    for (int i = 0; i < 26; i++)
        freq[i] = 0;
    thrds[0] = new boost::thread(boost::bind(&letter_freq_p, text, freq, 0, length/2));
    thrds[1] = new boost::thread(boost::bind(&letter_freq_p, text, freq, length/2, length));
    thrds[0]->join();
    thrds[1]->join();
    delete thrds[0];
    delete thrds[1];
}
```

Letter frequency for 2 threads:

```cpp
void letter_freq_p(char* text, long* freq, long s, long e)
{
    /* reset frequency array */
    for (long i = 0; i < 26; i++)
        freq[i] = 0;
    /* count characters */
    for (long i = s; i < e; i++)
        freq[(text[i] - 'a')]++;
}

void letter_freq(char* text, long length, long* freq)
{
    boost::thread* thrds[2];
    long freq_t1[26];
    long freq_t2[26];
    /* create two threads and bind them to letter_freq_p */
    thrds[0] = new boost::thread(boost::bind(&letter_freq_p, text, freq_t1, 0, length/2));
    thrds[1] = new boost::thread(boost::bind(&letter_freq_p, text, freq_t2, length/2, length));
    /* wait for threads to finish */
    thrds[0]->join();
    thrds[1]->join();
    /* add up the frequencies */
    for (int i = 0; i < 26; i++)
        freq[i] = freq_t1[i] + freq_t2[i];
    delete thrds[0];
    delete thrds[1];
}
```

P-11 — Programming – Performance Analysis

**Speedup for letter frequency algorithms:**

- Final variant of the letter frequency algorithm

![Graph showing speedup vs number of threads]

- Attempt 3 and 4 of the letter frequency algorithm

![Graph showing speedup vs number of threads]

R-1 — References

