Parallel Programming
Exercise Session 3

10.03.2014 – 14.03.2014
Lab Overview

- Feedback from the last exercise
- Discussion of exercise 3
  - Long Multiplication
  - Sieve of Eratosthenes
  - Java Tutorial
Feedback: Exercise 2

• Pipelining
• Amdahl’s Law
• Task Graphs
• Identify potential parallelism
Lab Overview

- Feedback from the last exercise
- Discussion of exercise 3
  - Long Multiplication
  - Sieve of Eratosthenes
  - Java Tutorial
Exercise 3: Goals

What is an *algorithm*?

- Formal description
  Input, output, operations
- Correctness and exceptions
  Does it work? If not, in which cases?
- Complexity analysis
  Is it asymptotically better than X? If so when?

What is a *program*?

- Familiarize yourself with key concepts and practices of Java
- Syntax
  How do I write the instructions down?
- Topics: types, variables, control flow, loops and arrays
Lab Overview

- Feedback from the last exercise
- Discussion of exercise 3
  - Long Multiplication
  - Sieve of Eratosthenes
  - Java Tutorial
Multiplication

**Input:** $x, y \in \mathbb{N}_{>0}$

**Output:** $x \times y$ (multiplication)

**Approaches:**
- Russian Peasant method (Lecture last week)
- Long multiplication (Primary school)
- Karatsuba multiplication (Data Structures and Algorithms course)
- ...
Long Multiplication

Input: $x, y \in \mathbb{N}_{>0}$

Output: $x \times y$ (multiplication)

Example:

\[
\begin{array}{c}
334589 \\
\times \quad 2170
\end{array}
\]

\[
\begin{array}{cccc}
0 & 0 & 0 & 0 & 0 \\
(= \quad 334589 \times \quad 0 ) \\
2342123 & \quad (= \quad 334589 \times \quad 70 ) \\
334589 & \quad (= \quad 334589 \times \quad 100 ) \\
669178 & \quad (= \quad 334589 \times \quad 2000 ) \\
\hline
726058130
\end{array}
\]
Long Multiplication: Implementation

• We provide a skeleton class and (failing) unit tests

• Solution should support arbitrary precision
  Helper routines to convert strings to their digit representation and back

• Use of the multiplication operator (*) is allowed
  but only for multiplication by constants or single digit values
Russian Peasant Method: Efficiency of the Algorithm

Question: How long does it take to multiply $a$ and $b$?

Efficiency metric: Total number of basic operations
- Multiplication, division, additions, test for even/odd, tests equal to “1”
- Per call to $f(a, b)$ not more than 5 basic operations
Efficiency estimate

More precise analysis needs to take costs of individual operations into account

- Multiplication (by two) is cheaper than adding
- Large values are more expensive than small values
- We assume that all operations are equally expensive

Number of recursions is decisive:
- This only depends on $b$ not on $a$

How many recursions do we need?
- Conservatively estimated via rounding:
  - $\frac{b}{2^x} \leq 1 \Rightarrow x \geq \log_2 b$

We only need $5 \log_2 b$ operations or less

```
static int f(int a, int b){
  if (b == 1) return a;
  if (b%2 == 0) return f(a+a, b/2);
  else return a + f(a+a, b/2);
}
```
Comparison with Long multiplication

Why do we learn long multiplication in school?

Is this better (than the old-egyptian method)?

\[
\begin{array}{cccc}
3 & 1 & 2 & \times \\
\hline
1 & 2 & 4 & 8 \\
3 & 1 & 2 & + \\
1 & 5 & 6 & 0 = \\
\hline
1 & 2 & 9 & 4 & 8 & 0
\end{array}
\]
Challenge: Karatsuba multiplication

- Analyzed at Data Structures and Algorithms course
- At the time of its discovery (1960), first algorithm known to be asymptotically faster than long multiplication
- Applies divide-and-conquer approach and requires only three rather than the usual four multiplications
Lab Overview

- Feedback from the last exercise
- Discussion of exercise 3
  - Long Multiplication
  - Sieve of Eratosthenes
  - Java Tutorial
Algorithm

Input: \( max \in \mathbb{N}_{>0} \)

Output: all the prime numbers less than or equal to \( max \)

Operations:

for each ... do

end
Prime Numbers

*Prime*: integer \( p > 1 \) with no divisors other than 1 and \( p \) itself

Example:

- 13 is prime since its only divisors are 1 and 13
- 24 is not prime. It has divisors 1, 2, 3, 4, 6, 8, 12, and 24. It can be factorized as \( 24 = 2^3 \cdot 3 \)

*Prime sieve*: algorithm for finding all primes up to a given limit
Sieve of Eratosthenes: Example

• Demonstrate the algorithm on the blackboard.

• This week: sequential, next week: parallelise.
Sieve of Eratosthenes: Example
Sieve of Eratosthenes: Example
Sieve of Eratosthenes: Example
Sieve of Eratosthenes: Example
Sieve of Eratosthenes: Example
Sieve of Eratosthenes: Example
Sieve of Eratosthenes: Example

... and so on
Challenge: Iterative Prime Sieve

- Alternative formulation of the sieve in lazy functional style
- Allows to generate prime numbers indefinitely
- Implement in Java using iterators
- For details and example code see paper: “The Genuine Sieve of Eratosthenes” by Melissa E. O’Neill
Lab Overview

- Feedback from the last exercise
- Discussion of exercise 3
  - Long Multiplication
  - Sieve of Eratosthenes
  - Java Tutorial
Java Programing

“Java in a Nutshell for Eiffel Programmers”, Stephanie Balzer, February 2010

http://www.lst.inf.ethz.ch/teaching/lectures/ss10/24/slides/2010-02-25-code_ex.zip

[@TAs: discuss the slides referenced above]
Eclipse: importing a project (1)

- Download ZIP from course page
- Menu: File -> Import
Eclipse: importing a project (2)

- Enter ZIP file path
- Hit <Enter> to confirm
- Make sure *assignment3* project is selected
- Click “Finish”
- Repeat the steps to add project to SVN
Eclipse: running JUnit tests (1)
Eclipse: running JUnit tests (2)
JavaDoc

• API Specification for Java
  http://docs.oracle.com/javase/7/docs/api/
• Also available directly within Eclipse / IntelliJ / ...
• Explain: how to read a method signature
Java™ Platform, Standard Edition 7
API Specification

This document is the API specification for the Java™ Platform, Standard Edition.

See: Description

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.applet</td>
<td>Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.</td>
</tr>
<tr>
<td>java.awt</td>
<td>Contains all of the classes for creating user interfaces and for painting graphics and images.</td>
</tr>
<tr>
<td>java.awt.color</td>
<td>Provides classes for color spaces.</td>
</tr>
<tr>
<td>java.awt.datatransfer</td>
<td>Provides interfaces and classes for transferring data between and within applications.</td>
</tr>
<tr>
<td>java.awt.dnd</td>
<td>Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the GUI.</td>
</tr>
<tr>
<td>java.awt.event</td>
<td>Provides interfaces and classes for dealing with different types of events fired by AWT components.</td>
</tr>
<tr>
<td>java.awt.font</td>
<td>Provides classes and interface relating to fonts.</td>
</tr>
<tr>
<td>java.awt.geom</td>
<td>Provides the Java 2D classes for defining and performing operations on objects related to two-dimensional geometry.</td>
</tr>
<tr>
<td>java.awt.im</td>
<td>Provides classes and interfaces for the input method framework.</td>
</tr>
<tr>
<td>java.awt.im.spi</td>
<td>Provides interfaces that enable the development of input methods that can be used with any Java runtime environment.</td>
</tr>
<tr>
<td>java.awt.image</td>
<td>Provides classes for creating and modifying images.</td>
</tr>
<tr>
<td>java.awt.image.renderable</td>
<td>Provides classes and interfaces for producing rendering-independent images.</td>
</tr>
<tr>
<td>java.awt.print</td>
<td>Provides classes and interfaces for a general printing API.</td>
</tr>
<tr>
<td>java.beans</td>
<td>Contains classes related to developing beans – components based on the JavaBeans™ architecture.</td>
</tr>
<tr>
<td>java.beans.beancontext</td>
<td>Provides classes and interfaces relating to bean context.</td>
</tr>
<tr>
<td>java.io</td>
<td>Provides for system input and output through data streams, serialization and the file system.</td>
</tr>
<tr>
<td>java.lang</td>
<td>Provides classes that are fundamental to the design of the Java programming language.</td>
</tr>
<tr>
<td>java.lang.annotation</td>
<td>Provides library support for the Java programming language annotation facility.</td>
</tr>
<tr>
<td>java.lang.instrument</td>
<td>Provides services that allow Java programming language agents to instrument programs running on the JVM.</td>
</tr>
</tbody>
</table>
public interface List<E>
extends Collection<E>

An ordered collection (also known as a sequence). The user of this interface has precise control over where in the list each element is inserted. The user can access elements by their integer index (position in the list), and search for elements (in the list).

Unlike sets, lists typically allow duplicate elements. More formally, lists typically allow pairs of elements e1 and e2 such that e1.equals(e2), and they typically allow multiple null elements if they allow null elements at all. It is not inconceivable that someone might wish to implement a list that prohibits duplicates, by throwing runtime exceptions when the user attempts to insert them, but we expect this usage to be rare.

The List interface places additional stipulations, beyond those specified in the Collection interface, on the contracts of the iterator, add, remove, equals, and hashCode methods. Declarations for other inherited methods are also included here for convenience.

The List interface provides four methods for positional (indexed) access to list elements. Lists (like Java arrays) are zero based. Note that these operations may execute in time proportional to the index value for some implementations (the LinkedList class, for example). Thus, iterating over the elements in a list is typically preferable to indexing through if the caller does not know the implementation.

The List interface provides a special iterator, called a ListIterator, that allows element insertion and replacement, and bidirectional access in addition to the normal operations that the Iterator interface provides. A method is provided to obtain a list iterator that starts at a specified position in the list.

The List interface provides two methods to search for a specified object. From a performance standpoint, these methods should be used with caution. In many implementations they will perform costly linear searches.

The List interface provides two methods to efficiently insert and remove multiple elements at an arbitrary point in the list.

Note: While it is permissible for lists to contain themselves as elements, extreme caution is advised; the equals and hashCode methods are no longer well-defined on such a list.
Detailed Documentation:

- **Class Description**
- **Inheritance Hierarchy**
- **Method Summary**
add

void add(int index,
   E element)

Inserts the specified element at the specified position in this list (optional operation). Shifts the element currently at that position (if any) and any subsequent elements to the right (adds one to their indices).

Parameters:
   index - Index at which the specified element is to be inserted
   element - element to be inserted

Throws:
   UnsupportedOperationException - if the add operation is not supported by this list
   ClassCastException - if the class of the specified element prevents it from being added to this list
   NullPointerException - if the specified element is null and this list does not permit null elements
   IllegalArgumentException - if some property of the specified element prevents it from being added to this list
   IndexOutOfBoundsException - if the index is out of range (index < 0 || index >= size())

remove

E remove(int index)

Removes the element at the specified position in this list (optional operation). Shifts any subsequent elements to the left (subtracts one from their indices). Returns the element that was removed from the list.

Parameters:
   index - the index of the element to be removed

Returns:
   the element previously at the specified position

Throws:
   UnsupportedOperationException - if the remove operation is not supported by this list
   IndexOutOfBoundsException - if the index is out of range (index < 0 || index >= size())
More Resources

Stanford CS 108: Object Oriented Systems Design

Handouts on Java, Eclipse, Debugging, Generics, Threading, ...

http://www.stanford.edu/class/cs108/
BACKUP
Sieve of Eratosthenes: Algorithm

**Input:** a non-negative integer $max$

**Output:** all the prime numbers less than or equal to $max$
Sieve of Eratosthenes: Algorithm

**Input:** a non-negative integer \( \text{max} \)

**Output:** all the prime numbers less than or equal to \( \text{max} \)

**Algorithm:**

- Create a list of natural numbers 2, 3, 4, 5, ..., \( \text{max} \). (None of which is marked.)
- Set \( k \) to 2, the first unmarked number on the list.
- Repeat until \( k^2 > \text{max} \):
  - Mark all multiples of \( k \) between \( k^2 \) and \( \text{max} \).
  - Find the smallest number greater than \( k \) that is unmarked. Set \( k \) to this new value.
- All the unmarked numbers are primes.