Wykonawcy i współbieżne struktury danych

1. Dana jest tablica `int [] L` wypełniona losowymi, nieujemnymi liczbami całkowitymi. Należy napisać program, który zliczy ile z tych liczb jest pierwszych. Sprawdzanie liczb pierwszych należy podzielić na podzadania realizowane współbieżnie za pomocą Wykonawcy (np. Executors.newFixedThreadPool).

   - Podzadania powinny implementować interfejs Callable i odpowiadać za zliczanie liczb pierwszych w wybranym fragmencie tablicy `L`.
   - Po zakończeniu wykonywania poszczególnych zadań wątek główny powinien zsumować otrzymane wyniki z poszczególnych podzadań (za pomocą obiektów Future).
   - Program przetestować dla różnych rozmiarów tablicy `L`.

2. Napisać program wyznaczający maksymalny element w tablicy liczb typu `int`. Wyznaczanie elementu maksymalnego powinno być wykonywane w sposób współbieżny z zastosowaniem modelu Fork / Join obsługiwanego przez wykonawcę `ForkJoinPool`.

   Pseudokod algorytmu jest następujący:

   a) Jeżeli podany zakres indeksów tablicy jest mniejszy niż zadany próg, to wyznacz maksimum w sposób sekwencyjny

   b) Jeżeli podany zakres indeksów jest większy niż próg (np. 1000), to podziel zakres na dwa mniejsze i wyznacz maksimum w każdej z części w sposób rekurencyjny. Następnie zwróć jako wynik większą z dwóch otrzymanych liczb.

   Program należy testować na tablicy liczb o rozmiarze N wypełnionej losowymi wartościami. Porównać czas działania wersji równoległej z wersją sekwencyjną dla różnych wartości N, tj. $N \in 10^3, 10^6, 10^7$.

3. Napisać program realizujący równoległe mnożenie dwóch macierzy A i B liczb całkowitych. Wynikiem powinna być macierz C. Obliczenia powinny być realizowane rekurencyjnie na blokach macierzy:

   - jeżeli rozmiar bloku jest poniżej progu P, to mnożenie wykonywane jest bezpośrednio; dla uproszczenia można przyjąć, że najmniejszą „jednostką” (blokiem) jest pojedynczy wiersz macierzy C

- jeżeli rozmiar bloku nie jest mniejszy, niż próg P, to zadanie dzielone jest na podzadania,
- podzadania powinny dziedziczyć po klasie `RecursiveAction` i do ich obliczenia należy zastosować model Fork / Join obsługiwany przez wykonawcę ForkJoinPool.

5.1 GUI w programach wielowątkowych

1. Napisać program w Javie znajdujący wszystkie pliki w zadanym katalogu i jego podkatalogach, których rozmiar jest większy od 1 MB.

   Program powinien oferować interfejs użytkownika (Swing) zawierający nast. elementy:

   - przycisk „Start” uruchamiający wyszukiwanie w wątku roboczym;
   - listę nazw plików spełniających kryteria (można użyć `JList`);
   - wynikową etykietę z liczbą sprawdzonych plików.

   Aktualizcja listy plików powinna być wykonywana na bieżąco, tj. w trakcie procesu wyszukiwania. Etykieta wynikowa powinna zostać zaktualizowana po zakończeniu przeszukiwania.

   Szablon rozwiązania z zastosowaniem klasy `SwingWorker` do implementacji wątku roboczego przedstawiono poniżej. Uwaga – do poprawnego działania należy pobrać bibliotekę MigLayout (core oraz swing) ze strony:
   `https://oss.sonatype.org/content/repositories/snapshots/com/miglayout/`

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;
import java.nio.file.*;
import java.nio.file.attribute.BasicFileAttributes;
import java.util.List;
import net.miginfocom.swing.MigLayout;

public class Main {
    public static void main(String[] args) throws IOException {
        final MainWindow mainWindow = new MainWindow();
    }
}

class MainWindow implements ActionListener {
    private final JLabel resultLabel = new JLabel("Wynik");
    private DefaultListModel filesListModel = new DefaultListModel();
    private BigFilesFinder finder = null;

    public MainWindow() {
        JFrame frame = new JFrame();
        frame.setTitle("Test Frame");
        frame.setSize(600, 400);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JButton startBtn = new JButton("Start");
```

```
        startBtn.setActionCommand("start");
        startBtn.addActionListener(this);

        JList<String> filesList = new JList<>(filesListModel);
        JScrollPane listScroller = new JScrollPane(filesList);
        listScroller.setPreferredSize(new Dimension(4000, 4000));

        JPanel mainPanel = new JPanel(new MigLayout());
        mainPanel.add(startBtn, "wrap");
        mainPanel.add(listScroller, "span, wrap");
        mainPanel.add(resultLabel);

        frame.getContentPane().add(mainPanel);
        frame.setVisible(true);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        if ( "start".equals(e.getActionCommand()) ) {
            if (finder == null || finder.isDone()) {
                final long MB = 1024 * 1024;
                finder = new BigFilesFinder("c:\\Dokumenty", 1 * MB);
            }
            filesListModel.clear();
            finder.execute();
        }
    }

    class BigFilesFinder extends SwingWorker<Long, Path> {
        private final long fileSizeThreshold;
        private final String searchPath;

        BigFilesFinder(String searchPath, long fileSizeThreshold) {
            this.fileSizeThreshold = fileSizeThreshold;
            this.searchPath = searchPath;
        }

        @Override
        protected Long doInBackground() throws Exception {
            // TODO
            return 0;
        }

        @Override
        protected void done() {
            // TODO
        }

        @Override
        protected void process(List<Path> paths) {
            // TODO
        }
    }
}
```