9. homework assignment; JAVA, Academic year 2013/2014; FER

Tekst ove zadaće dan je na hrvatskom.

Zadatak 1.

Proučite algoritam sortiranja *umetanjem* te algoritam *quicksort*. Kao osnovni "predložak" koda koristite:

http://www.brpreiss.com/books/opus4/html/page488.html

te

http://www.brpreiss.com/books/opus4/html/page495.html

Navedeni kodovi napisani su u jeziku C++; zanemarite napisane razrede: proučite konkretne algoritme koji su priloženi te izvedbu odabira pivota metodom medijan-od-tri:

http://www.brpreiss.com/books/opus4/html/page500.html.

Napišite implementaciju algoritma QuickSort koja podposlove obavlja u novim dretvama (ako su dovoljno veliki) odnosno klasično slijedno ako je veličina posla manja ili jednaka određenom pragu. Kako je stvaranje i pokretanje nove dretve relativno zahtjevno, pokazuje se da posao koji se želi paralelizirati mora biti dovoljno velik da bi se taj trošak isplatio. U našem rješenju ovo regulira prag `P_THRESHOLD`. Jednom kada Vam rješenje proradi, ispitajte što se događa s ukupnim vremenom potrebnim za sortiranje ako vrijednost ovog praga dovoljno smanjite odnosno ako ga dovoljno povećate (primjerice, da je veći od veličine predanog posla).

Za paralelno izvođenje novih poslova smijete isključivo direktno koristiti razred `Thread` (uporaba `java.util.concurrent` paketa i njegovih podpaketa nije dozvoljena).

Kostur koda koji samo trebate nadopuniti dan je u nastavku. Najprije evo glavnog programa.

```
package hr.fer.zemris.java.sorters;

import java.util.Random;

public class Glavni {

	public static void main(String[] args) {
		final int SIZE = 150000;

		Random rand = new Random();
		int[] data = new int[SIZE];
		for(int i = 0; i < data.length; i++) {
			data[i] = rand.nextInt();
		}

		long t0 = System.currentTimeMillis();
		QSortParallel.sort(data);
		long t1 = System.currentTimeMillis();

		System.out.println("Sortirano: " + QSortParallel.isSorted(data));
```

```
			System.out.println("Vrijeme: " + (t1-t0)+" ms");
		}

}
```

Sljedeće je prikazan glavni dio rješenja ovog zadatka.

```
package hr.fer.zemris.java.sorters;

public class QSortParallel {

		/**
		 * Prag koji govori koliko elemenata u podpolju minimalno
		 * mora biti da bi se sortiranje nastavilo paralelno;
		 * ako elemenata ima manje, algoritam prelazi na klasično
		 * rekurzivno (slijedno) sortiranje.
		 */
		private final static int P_THRESHOLD = 6000;
		/**
		 * Prag za prekid rekurzije. Ako elemenata ima više od
		 * ovoga, quicksort nastavlja rekurzivnu dekompoziciju.
		 * U suprotnom ostatak sortira algoritmom umetanja.
		 */
		private final static int CUT_OFF = 5;

		/**
		 * Sučelje prema klijentu: prima polje i vraća se
		 * tek kada je polje sortirano. Primjenjujući gornje
		 * pragove najprije posao paralelizira a kada posao
		 * postane dovoljno mali, rješava ga slijedno.
		 *
		 * @param array polje koje treba sortirati
		 */
		public static void sort(int[] array) {
			new QSortJob(array, 0, array.length-1).run();
		}

		/**
		 * Model posla sortiranja podpolja čiji su elementi na pozicijama
		 * koje su veće ili jednake <code>startIndex</code> i manje
		 * ili jednake <code>endIndex</code>.
		 */
		static class QSortJob implements Runnable {
			private int[] array;
			private int startIndex;
			private int endIndex;

			public QSortJob(int[] array, int startIndex, int endIndex) {
				super();
				this.array = array;
				this.startIndex = startIndex;
				this.endIndex = endIndex;
			}

			@Override
			public void run() {
				if(endIndex-startIndex+1 > CUT_OFF) {
					boolean doInParallel = endIndex-startIndex+1 > P_THRESHOLD;

					// pronađi pivot, razdijeli polje u lijevi i desni dio
					// …
```

```
                        // Neka je ovdje 'i' konačna lokacija na kojoj je završio
                        // pivot

                        Thread t1 = null;
                        if(startIndex < i) {
                                QSortJob job = … lijevi dio
                                t1 = executeJob(doInParallel, job);
                        }
                        Thread t2 = null;
                        if(endIndex > i) {
                                QSortJob job = … desni dio
                                t2 = executeJob(doInParallel, job);
                        }
                        // ako su t1/t2 pokrenuti, dočekaj ih da završe
                        // ...
                } else {
                        // Obavi sortiranje umetanjem
                        // ...
                }
        }

        /**
         * Direktno izvodi zadani posao pozivom run() i tada vraća <code>null</code>
         * ili pak stvara novu dretvu, njoj daje taj posao i pokreće je te vraća
         * referencu na stvorenu dretvu (u tom slučaju ne čeka da posao završi).
         * @param doInParallel treba li posao pokrenuti u novoj dretvi
         * @param job posao
         * @return <code>null</code> ili referencu na pokrenutu dretvu
         */
        private Thread executeJob(boolean doInParallel, QSortJob job) {
                // ...
        }

        /**
         * Odabir pivota metodom medijan-od-tri u dijelu polja
         * <code>array</code> koje je ograđeno indeksima
         * <code>startIndex</code> i <code>endIndex</code>
         * (oba uključena).
         *
         * @return vraća indeks na kojem se nalazi odabrani pivot
         */
        public int selectPivot() {
                // ...
        }

        /**
         * U predanom polju <code>array</code> zamjenjuje
         * elemente na pozicijama <code>i</code> i <code>j</code>.
         * @param array polje u kojem treba zamijeniti elemente
         * @param i prvi indeks
         * @param j drugi indeks
         */
        public static void swap(int[] array, int i, int j) {
                // ...
        }
}

/**
 * Pomoćna metoda koja provjerava je li predano polje
 * doista sortirano uzlazno.
 *
 * @param array polje
```

```
         * @return <code>true</code> ako je, <code>false</code> inače
         */
        public static boolean isSorted(int[] array) {
                    // ...
        }
}
```

Zadatak 2.

Ovdje radimo nadogradnju domaće zadaće u kojoj ste radili model Booleovih funkcija. Proširite razred `MaskBasedBF` sa sljedećim metodama.

```
public List<Mask> getMasks() { ... }
public List<Mask> getDontCareMasks() { ... }
public boolean areMasksProducts() { ... }
```

Prva metoda vraća listu predanih maski koje definiraju lokacije na kojima funkcija poprima vrijednost 1 ili 0 (ovisno o zastavici predanoj u konstruktoru), druga vraća popis maski koje definiraju *don't care* lokacije a treća vraća jesu li maske produkti (`true`) ili sume (`false`).

Potom napravite podpaket `hr.fer.zemris.bool.qmc` i u njemu razred `QMCMinimizer` koji omogućava provedbu minimizacije booleove funkcije. Rezultat minimizacije je polje Booleovih funkcija predstavljenih razredom `MaskBasedBF`; pri tome polje ima onoliko elemenata koliko funkcija ima minimalnih oblika. Željeni primjer uporabe je sljedeći.

```
BooleanFunction bf = ... / na bilo koji legalni način definirana funkcija (imali smo tri takva)
boolean želimoDobitiProdukte = true;
MaskBasedBF[] fje =  QMCMinimizer.minimize(bf, želimoDobitiProdukte);
System.out.println("Minimalnih oblika ima: " + fje.length);
for(MaskBasedBF f : fje) {
     System.out.println("Mogući minimalni oblik:");
     for(Mask m : f.getMasks()) {
          System.out.println("  " + m);
     }
}
```

Da biste otkrili sve minimalne oblike nužno je da nakon prvog koraka provedete Pyne-McCluskey postupak minimizacije.

Napomena: drugi argument metode `minimize` je booleova zastavica koja govori radi li se minimizacija minterma ili maksterma; u prvom slučaju u prvom koraku QMC postupka indekse grupirate po broju jedinica; u drugom slučaju po broju nula! Za svako kombiniranje morate gledati što ste kombinirali (i je li to i dalje *don't care* ili nije). Stoga preporučam da proširite razred koji definira masku svojstvom *dontCare* koje je samo za čitanje i koje se postavlja u konstruktoru; gdje je još potrebno, provedite potrebne modifikacije.

Napomena 2: jasno je da u funkcijama koje ste stvorili minimizacijom i koje vraćate u polju nema *don't care* maski.

Napomena 3: ovaj zadatak nije vezan uz paralelizaciju (ako se netko pita što tu treba paralelizirati).

Napomena 4: Implementacija mora biti napisana tako da radi za funkcije proizvoljnog broja varijabli (drugim riječima: što god date kao ulaz, neće se srušiti, osim, naravno, ako usfali memorije).

Vaša implementacija bi morala biti takva da proizvoljnu booleovu funkciju od do-uključivo 4 varijable minimizira u vremenu koje je kraće od jedne sekunde na prosječnom jednojezgrenom procesoru takta od 2 Ghz).

Evo nekoliko funkcija od četiri varijable koje su zadane kao sume minterma (indeksi su dani u zagradama) s kojima biste na kraju trebali probati.

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
[0, 1, 2, 3, 4, 6, 7, 8, 9, 11, 12, 13, 14, 15]
[0, 1, 2, 3, 4, 5, 6, 9, 10, 11, 12, 13, 14]
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15]
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
[1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]

Vaše početne implementacije testirajte na sljedećim funkcijama (dane su sume minterma).

[0,1,4,5,11,15]
  ==> ima 1 minimalni oblik, maske su (0x0x, 1x11)

[0,1,4,5,9,11,15]
  ==> ima 2 minimalna oblika, maske su (0x0x, 1x11, 10x1) odnosno (0x0x, 1x11, x001)

[0,1,4,5,9,15] + dontcare[11]
  ==> ima 2 minimalna oblika, maske su (0x0x, 1x11, 10x1) odnosno (0x0x, 1x11, x001)

[4,5,6,7,8,9,10,11,13,14]
  ==> ima 4 minimalna oblika

[4,5,7,9,10,11,13,14] + dontcare[6,8]
  ==> ima 4 minimalna oblika

[0,1,4,5,11,15] + dontcare[2,6,10]
  ==> ima 1 minimalni oblik, maske su (0x0x, 1x11)

Napomena 5: ako bilo tko ima problema (ne sjeća se, ne zna, nije nikada naučio) QMC minimizaciju, stojim na raspolaganju u ponedjeljak u terminu od 15:00 te u utorak u terminu od 10:00, pa svratite.

Important notes

Solve all of the problems in a single Eclipse project. Configure Eclipse to use two source directories: `src/main/java` for your source files and `src/test/java` for sources files of unit tests.

You are required to write the adequate number of unit tests for this project.

You must equip your project with `build.xml` script so that the project can be build from the command line. In that script, you must integrate all of the quality-checking tools which I have described in the book (by integrate, I mean have a new target for each tool so that you can call each tool separately). It should be possible to run at least the following targets: `init`, `compile`, `compile-tests`, `jar`, `run-tests`, `quality`, `reports`, `clean`.

Target `quality` must run unit tests and all of the quality checks (i.e. *checkstyle*, *pmd*, *findbugs*): make it just a target which depends on all tool-targets. Unit tests must be run with the code coverage analysis. Target `reports` is a wrapper that will run all of the unit tests, the quality checks and the javadoc generation.

All of the classes should have appropriate javadoc.

Please note. You can consult with your peers and exchange ideas about this homework *before* you start actual coding. Once you open you IDE and start coding, consultations with others (except with me) will be regarded as cheating. You can not use any of preexisting code or libraries which is not part of Java standard edition (Java SE) unless explicitly provided by me. This means that from this point on, you can use Java Collection Framework classes or its derivatives (moreover, I recommend it). Document your code!

In order to solve this homework, create a blank Eclipse Java Project and write your code inside. You must name your project's main directory (which is usually also the project name) `HW09-`*yourJMBAG*; for example, if your JMBAG is 0012345678, the project name and the directory name must be `HW09-0012345678`. Once you are done, export the project as a ZIP archive and upload this archive to Ferko before the deadline. Do not forget to lock your upload or upload will not be accepted. Deadline is May 13th 2014. at 23:59.