8. homework assignment; JAVA, Academic year 2016/2017; FER

Nastavite raditi u Maven-projektu koji ste napravili za 7. domaću zadaću. Time će direktorij projekta na disku ostati `hw07-0000000000` (gdje su nule zamijenjene Vašim JMBAG-om), a koordinate projekta su i dalje `hr.fer.zemris.java.jmbag0000000000:hw07-0000000000` (gdje su nule zamijenjene Vašim JMBAG-om).

Problem 1.

Dopunite razred `hr.fer.zemris.bf.utils.Util` sljedećom metodom:

```
public static byte[] indexToByteArray(int x, int n);
```

Zadaća metode je cijeli broj `x` pretvoriti u binarni zapis širine `n` bita, gdje se za pohranu znamenaka koristi polje okteta. Pri tome se na poziciju 0 u polju pohranjuje bit najveće težine. Primjerice,

`indexToByteArray(3, 2)` vraća `new byte[] {1, 1}`
`indexToByteArray(3, 4)` vraća `new byte[] {0, 0, 1, 1}`
`indexToByteArray(3, 6)` vraća `new byte[] {0, 0, 0, 0, 1, 1}`

Metodu riješite izravno: nemojte koristiti pomoćne metode koje rade pretvorbe u druge objekte tipa stringove. Potom za metodu napišite junit testove i provjerite njezin rad. Kao jednu od provjera ispitajte radi li Vaša metoda korektno za `indexToByteArray(-2, 32)` odnosno `indexToByteArray(-2, 16)`.

Napomena: metoda u polje koje vraća smješta n najmanje značajnih bitova. Ako korisnik dade premali `n` za neki konkretan cijeli broj koji koristi više bitova u prikazu, doći će do gubitka informacije odnosno na temelju vraćenog polja taj se broj više neće moći rekonstruirati – to je OK. Primjerice, očekujemo da:

`indexToByteArray(19, 4)` vraća `new byte[] {0, 0, 1, 1}`.

Problem 2.

Napravite razred `hr.fer.zemris.bf.qmc.Mask`. Primjerci ovog razreda predstavljaju specifikaciju (moguće nepotpunih) produkata. Primjerice, neka radimo s Booleovom funkcijom definiranom nad varijablama `A`, `B`, `C`, `D` (tim poretkom). Produkt `ABC'D` (gdje `C'` označava komplement varijable `C`) kraće ćemo zapisati maskom `1101`, produkt `A'B'CD` maskom `0011` i slično. Komplementirane varijable u maski su reprezentirane znamenkom `0`, nekomplementirane znamenkom `1`. Produkt `AC'D` ćemo zapisati maskom `1201`, gdje znamenka `2` označava da se varijabla na čijoj je poziciji znamenka `2` ne pojavljuje u produktu. Evo još par primjera: produkt `A'D` imat će masku `0221`, "produkt" `B'` imat će masku `2022`. Kako je svaki produkt zapravo oblika `1*varijable`, tautologija u našem primjeru ima zapis `2222`. Primijetite također da je produkt `A'B'CD` za našu funkciju zapravo minterm 3, a produkt `AC'D` istovremeno predstavlja skup minterma 9 i 13. Konačno, za neku funkciju, specificirani produkt može ukazivati na mjesta na kojima funkcija poprima vrijednost 1, ili može ukazivati na mjesta na kojima vrijednost funkcije nije bitna (sjetite se *don't care* oznake i nepotpuno specificiranih Booleovih funkcija). Primjerci razreda `Mask` omogućit će pohranu svih triju informacija: maske (polje okteta), skupa indeksa minterma koje produkt predstavlja te je li to "normalan" produkt ili *don't care* (booleova zastavica).

Razred treba imati sljedeće javne konstruktore:

```
public Mask(int index, int numberOfVariables, boolean dontCare);
public Mask(byte[] values, Set<Integer> indexes, boolean dontCare);
```

Prvi konstruktor prima redni broj minterma, broj varijabli funkcije te je li minterm *don't care* i na temelju toga računa i pamti masku, jednočlani skup indeksa i zastavicu. Drugi konstruktor eksplicitno prima masku, skup indeksa i zastavicu (pamti kopije, kako korisnik izvana ne bi mogao mijenjati zapamćene podatke). Ovaj drugi konstruktor ne mora provjeravati jesu li predano polje i predani skup međusobno konzistentni u smislu da set sadrži baš one indekse koje maska stvarno predstavlja; to će biti odgovornost pozivatelja. U slučaju da se predaju podatci koji nisu korektni (npr. `null` za masku, predan broj varijabli je manji od 1, `null` za set ili je set prazan), treba baciti `IllegalArgumentException`. Skup indeksa treba biti takav da iterator indekse dohvaća od najmanjeg prema najvećem.

S obzirom da će primjerci razreda `Mask` prema van biti (skoro pa) nepromjenjivi (neće dozvoljavati promjene zapamćene maske, skupa odnosno zastavice *don't care*), Vaši konstruktori moraju izračunati sažetak maske i zapamtiti ga (pogledajte `Arrays#hashCode`) pa nadjačati vlastitu metodu `hashCode` tako da pri pozivu ništa ne računa već vraća u konstruktoru unaprijed izračunatu vrijednost: u primjeni koju imamo, ključna će biti maksimalna efikasnost izvođenja pa se zato služimo ovakvim rješenjem. Nadjačajte i metodu `equals` tako da kaže da su dva primjerka razreda `Mask` jednaka ako su im jednake maske koje pamte (dakle gledate samo sadržaj zapamćenog polja; pogledajte `Arrays#equals`). Pri tome na usporedbu polja krenite samo ako je to nužno: prije toga usporedite sažetke, pa ako na temelju toga možete donijeti odluku, donesite je.

Razred treba imati *read-write* svojstvo `combined`: booleova zastavica koja govori je li produkt iskorišten u kombinaciji s nekim drugim produktom za stvaranje trećeg produkta. Potrebne metode su:

```
public boolean isCombined();
public void setCombined(boolean combined);
```

Trebamo i *read-only* svojstvo za zastavicu *don't care*: te getter za neizmjenjiv skup indeksa (neizmjenjivost treba osigurati konstruktor):

```
public boolean isDontCare();
public Set<Integer> getIndexes();
```

Dodajte u razred metodu čijim ćete pozivom dobiti informaciju na koliko je mjesta u maski prisutna vrijednost `1` (npr. za `2012` će vratiti `1`):

```
public int countOfOnes();
```

Nadjačajte metodu `toString` tako da generira ispis u kojem je vidljiva maska, uključeni indeksi, je li produkt *don't care* (veliko D) te je li produkt označen kao kombiniran (*). Evo nekoliko ispisa koji ilustriraju željeno formatiranje.

```
1010 . * [10]
0110 D * [6]
01-0 D   [4, 6]
1-1- .   [10, 11, 14, 15]
```

Umjesto znamenke 2, kod maske ćemo ispisivati znak minus.

Dodajte i sljedeće dvije metode:

```
public int size();
public byte getValueAt(int position);
```

Prva metoda treba vratiti broj varijabli koje čine masku (veličinu internog polja), a druga metoda treba vratiti vrijednost koja je pridružena traženoj varijabli (valjane pozicije idu od `0` do `size()-1`); vrijednost može biti `0`, `1` ili `2`.
Konačno, dodajte u razred metodu:

```
public Optional<Mask> combineWith(Mask other);
```

Ova metoda provjerava je li trenutni produkt moguće kombinirati s predanim produktom uporabom teorema simplifikacije, i ako je, vraća novi produkt koji je kombinacija (no trenutni produkt i predani produkt ne modificira *ni na koji način*). Ako kombiniranje nije moguće, vraća informaciju da "nema rezultata" (ako se još niste upoznali s razredom `Optional`, sad je pravi trenutak). Ako se kao argument preda `null` ili ako maske nisu jednake širine, metoda baca `IllegalArgumentException`.

Napišite ovu metodu tako da memoriju alocira samo ako je to potrebno: nemojte stvarati pomoćne objekte samo da biste na pola puta utvrdili da se stvar ne da kombinirati i onda u memoriji ostavili smeće. Ako se maske mogu kombinirati, nemojte zaboraviti na indekse!

Problem 3.

U okviru razvoja algoritma u nastavku, htjet ćemo omogućiti dobivanje ispisa kroz "log", pri čemu ćemo pri pokretanju programa moći odrediti zanimaju li nas ispisi ili ne. U takve svrhe Java platforma nudi uslugu rada s logovima, a danas je razvijeno i nekoliko biblioteka otvorenog koda. Za pregled funkcionalnosti koja je dostupna u okviru Java platforme pročitajte:

https://docs.oracle.com/javase/8/docs/technotes/guides/logging/overview.html

Poruke koje se šalju kroz logger imaju pridruženu razinu. Razine su:
https://docs.oracle.com/javase/8/docs/api/java/util/logging/Level.html

Obradu poruka obrađuju tzv. *handleri* (ajmo ih zvati *ispisivači*). Moguće je definirati da se poruka poslana kroz logger pošalje na više ispisivača (primjerice, jedan je ispiše na ekran a drugi istovremeno zapiše u log datoteku).

Napravite u direktoriju projekta datoteku `logging.properties` sljedećeg sadržaja:

```
# Log poruke salji samo na konzolu:
handlers= java.util.logging.ConsoleHandler

# Defaultni log-level postavi na INFO
.level = INFO

# Konfiguracija konzolnog ispisivaca
# ----------------------------------
# ispisuj samo poruke koje su razine INFO ili vaznije
java.util.logging.ConsoleHandler.level = FINEST
# koristi jednostavno formatiranje poruka
java.util.logging.ConsoleHandler.formatter = java.util.logging.SimpleFormatter
# Format koji koristi SimpleFormatter kojeg koristi prethodni ConsoleHandler:
java.util.logging.SimpleFormatter.format=[%1$tF %1$tT] %4$s: %5$s%n

# Evo konfiguracije za logger koji koristimo u paketu demo2
demo2.level = INFO
```

Što smo ovime definirali? Podesili smo da je pretpostavljena razina koju loggeri zaprimaju `INFO` ili više. Podesili smo da se koristi jedan ispisivač, koji poruke ispisuje u konzoli, i taj ispisivač ispisuje sve primljene poruke koje su razine `FINEST` ili više, poruke prilikom ispisa formatira uporabom razreda SimpleFormatter, a taj pak koristi `[%1$tF %1$tT] %4$s: %5$s%n` kao formatnu specifikaciju (to je isti format koji koristi `String#format` pa tamo možete pogledati detalje). Konačno, definirali smo da logger imena demo2 (ili bilo koji hijerarhijski ispod, budući da za takve ništa nismo navodili) prihvaća logiranje poruka koje su razine `INFO` ili više – ostale odbacuje, odnosno ne šalje ih podešenom ispisivaču. Uočite da će konačna odluka što će biti ispisano gdje ovisiti i o razini podešenoj za *logger*, i o razini podešenoj za ispisivač. Zašto je to tako, evo jednostavnog primjera. Mogli bismo imati *logger* koji prihvaća poruke koje su barem razine `INFO` i šalje ih na dva ispisivača: jednog koji ih zapisuje u datoteku ali samo ako su razine `SEVERE` te drugog koji ih ispisuje na ekran ako su bilo koje razine.

U paket `demo2` dodajte program prikazan u nastavku.

```
package demo2;

import java.util.logging.Level;
import java.util.logging.Logger;

public class Logiranje {

	private static final Logger LOG = Logger.getLogger("demo2");

	public static void main(String[] args) {

		Level[] levels = new Level[] {
			Level.SEVERE,
			Level.WARNING,
			Level.INFO,
			Level.CONFIG,
			Level.FINE,
			Level.FINER,
			Level.FINEST
		};
		for(Level l : levels) {
			LOG.log(l, "Ovo je poruka "+l+" razine.");
		}
	}
}
```

Pokrenite jednom program da biste u Eclipseu stvorili *Run* konfiguraciju. Potom otiđite u tu *Run* konfiguraciju i na mjestu gdje možete zadati parametre virtualnom stroju (kartica *Arguments*, polje *VM arguments*, ne *Program arguments*) unesite:

```
-Djava.util.logging.config.file=./logging.properties
```

Ovime virtualnom stroju definiramo tzv. system property imena `java.util.logging.config.file` i vrijednosti `./logging.properties`. Prilikom inicijalizacije podsustava za logiranje, razred LogManager čita to svojstvo i učitava konfiguracijsku datoteku na koje svojstvo pokazuje.

https://docs.oracle.com/javase/8/docs/api/java/util/logging/LogManager.html

Važno: ako datoteka ne postoji, čitav podsustav logiranja će se ponašati kao "mrtav" ali neće biti nikakvog upozorenja ili iznimke. Ako eksplicitno ne podesite konfiguracijsku datoteku, koristi se defaultna koja ima podešen ispis na zaslon i `INFO` kao razinu.

Otvorite sada konfiguracijsku datoteku. Ako `demo2.level` postavite na `INFO` i pokrenete program, dobit ćete ispis poput:

```
[2017-04-22 21:14:20] SEVERE: Ovo je poruka SEVERE razine.
[2017-04-22 21:14:20] WARNING: Ovo je poruka WARNING razine.
[2017-04-22 21:14:20] INFO: Ovo je poruka INFO razine.
```

Ako `demo2.level` postavite na `FINEST`, dobit ćete ispis poput:

```
[2017-04-22 21:14:56] SEVERE: Ovo je poruka SEVERE razine.
[2017-04-22 21:14:56] WARNING: Ovo je poruka WARNING razine.
[2017-04-22 21:14:56] INFO: Ovo je poruka INFO razine.
[2017-04-22 21:14:56] CONFIG: Ovo je poruka CONFIG razine.
[2017-04-22 21:14:56] FINE: Ovo je poruka FINE razine.
[2017-04-22 21:14:56] FINER: Ovo je poruka FINER razine.
[2017-04-22 21:14:56] FINEST: Ovo je poruka FINEST razine.
```

Uočite također kako smo u programu došli do objekta kroz koji šaljemo poruke (loggera): definirali smo privatnu statičku konstantu na razini razreda, i potom u tom razredima na svim mjestima gdje smo to trebali, pozivali smo nad tim objektom prikladne `log` metode.

Problem 4.

Sada krećemo na izradu minimizacijskog postupka Quine-McCluskey s Pyne-McCluskey pristupom. Napravite razred `hr.fer.zemris.bf.qmc.Minimizer`. Razred predstavlja implementaciju navedenog minimizacijskog postupka za jednu Booleovu funkciju (ne radimo minimizaciju višeizlazne funkcije). Razred mora imati javni konstruktor:

```
public Minimizer(
      Set<Integer> mintermSet,
      Set<Integer> dontCareSet,
      List<String> variables);
```

kroz koji prihvaća skup indeksa minterma, skup indeksa *don't careova* te listu varijabli nad kojima je funkcija definirana.

Zadaća konstruktora je provjeriti da ova dva skupa nemaju preklapanja (inače `IllegalArgumentException`) te da su predani indeksi legalni za predane varijable (funkcija nad {A,B,C} ne može imati minterm 42).

U konfiguracijskoj datoteci logera podesite razinu za logger imena `hr.fer.zemris.bf.qmc`.

```
# Evo konfiguracije za logger koji koristimo u paketu hr.fer.zemris.bf.qmc
hr.fer.zemris.bf.qmc.level = INFO
```

Deklarirajte u razredu `Minimizer` logger tog imena, i nemojte zaboraviti prilikom pokretanja demonstracija koje koriste ovaj minimizator virtualnom stroju poslati *system property* sa stazom do konfiguracijske datoteke za podsustav logiranja.

Gruba struktura konstruktora je sljedeća (kažem "struktura" jer prije/nakon slobodno dodajte što Vam još zatreba):

```
checkNonOverlapping(mintermSet, dontCareSet);
Set<Mask> primCover = findPrimaryImplicants();
List<Set<Mask>> minimalForms = chooseMinimalCover(primCover);
```

Konstruktor započinje provjerom jesu li predani skupovi disjunktni. Potom poziva metodu `findPrimaryImplicants` čija je zadaća provesti prvi korak minimizacijskog postupka QMC i vratiti skup primarnih implikanata (sjetite se koji su ti!). Taj se skup potom šalje metodi `chooseMinimalCover` čija je zadaća napraviti minimalni izbor primarnih implikanata, odnosno pronaći minimalni skup primarnih implikanata koji pokrivaju sve minterme funkcije. Kako je moguće da postoji više minimalnih načina da se to učini, metoda vraća listu skupova koji predstavljaju jednako vrijedne minimalne oblike.

Razmotrimo sada metodu `findPrimaryImplicants`. Njezina je zadaća pronalazak primarnih implikanata funkcije. Metoda kreće tako da izgradi prvi stupac tablice koji se sastoji od slijeda grupa. U prvu grupu dolaze svi mintermi i *don't careovi* koji imaju 0 jedinica u maski, u sljedeću grupu dolaze svi mintermi i don't careovi koji imaju jednu jedinicu u maski, i tako dalje. Za izgradnju ovog stupca poslužit ćemo se pomoćnom metodom koju ćemo ovdje samo pozvati:

```
private List<Set<Mask>> createFirstColumn();
```

Svaka grupa modelirana je kao skup, pa metoda vraća listu skupova. Skupovi pri tome moraju biti takvi da čuvaju poredak kojim su im elementi dodani. Metoda `createFirstColumn` najprije procesira sve minterme, potom sve *don't careove* i konačni rezultat vraća pozivatelju.

Metoda `findPrimaryImplicants` ne mora u memoriji pamtiti kompletnu tablicu. Metodu ćemo izvesti tako da iterativno, na temelju trenutnog stupca (a kreće s prvim koji je stvoren na prethodno opisani način) izgenerira sljedeći stupac. Nakon što je generiranje novog stupca gotovo, metoda može provjeriti kakvo je stanje u trenutnom stupcu i ako ima primarnih implikanata, nadoda ih u pomoćni skup koji gradi. Zatim odbaci trenutni stupac, kao trenutni stupac postavi prethodno novoizgrađeni i postupak iterativno ponavlja tako dugo dok po ulasku u iteracu trenutni stupac ne bude prazan. Na kraju svake iteracije ako je logiranje razine `FINER` omogućeno, metoda ispisuje trenutni stupac uz tu razinu (pogledajte ispise u nastavku). Ako je omogućeno logiranje razine `FINEST`, na toj razini se logiraju informacije o dodavanju primarnih implikanata u pomoćni skup koji se gradi kroz ovaj iterativni postupak. Ako je omogućeno logiranje `FINE`, na kraju izvođenja metode `findPrimaryImplicants` logira se skup pronađenih primarnih implikanata.

Pročitajte na početku dokumentacije razreda `Logger`:
https://docs.oracle.com/javase/8/docs/api/java/util/logging/Logger.html
diskusiju vezanu uz metode koje prihvaćaju `msgSupplier`. Radi se o tome da ne želite zvati metodu `log` s argumentom kojeg je skupo stvoriti ako niste sigurni da će se to stvarno i logirati, a ne odbaciti. Iskoristite to (ili kombinaciju s metodom `Log#isLoggable`) kako biste u "skupe" stvari krenuli tek ako to ima smisla.

Prilikom kombiniranja dvije maske, ako ih uspijete iskombinirati u novu, nemojte ih zaboraviti označiti kao kombinirana (postoji takvo svojstvo i njega smo jedinog ostavili kao promjenjivog u razredu `Mask`). Uz pretpostavku da ste konstruktor pozvali na sljedeći način:

```
Set<Integer> minterms = new HashSet<>(Arrays.asList(0,1,3,10,11,14,15));
Set<Integer> dontcares = new HashSet<>(Arrays.asList(4,6));

Minimizer m = new Minimizer(minterms, dontcares, Arrays.asList("A","B","C","D"));
```

te ako su dozvoljena logiranja svih poruka, ispis koji nastaje tijekom izvođenja metode `findPrimaryImplicants` bi morao biti poput ovoga:

```
[2017-04-22 22:05:03] FINER: Stupac tablice:
[2017-04-22 22:05:03] FINER: ================================
[2017-04-22 22:05:03] FINER: 0000 . * [0]
[2017-04-22 22:05:03] FINER: ------------------------------
[2017-04-22 22:05:03] FINER: 0001 . * [1]
[2017-04-22 22:05:03] FINER: 0100 D * [4]
```

```
[2017-04-22 22:05:03] FINER: ------------------------------
[2017-04-22 22:05:03] FINER: 0011 . * [3]
[2017-04-22 22:05:03] FINER: 1010 . * [10]
[2017-04-22 22:05:03] FINER: 0110 D * [6]
[2017-04-22 22:05:03] FINER: ------------------------------
[2017-04-22 22:05:03] FINER: 1011 . * [11]
[2017-04-22 22:05:03] FINER: 1110 . * [14]
[2017-04-22 22:05:03] FINER: ------------------------------
[2017-04-22 22:05:03] FINER: 1111 . * [15]
[2017-04-22 22:05:03] FINER:
[2017-04-22 22:05:03] FINER: Stupac tablice:
[2017-04-22 22:05:03] FINER: ================================
[2017-04-22 22:05:03] FINER: 000- .   [0, 1]
[2017-04-22 22:05:03] FINER: 0-00 .   [0, 4]
[2017-04-22 22:05:03] FINER: ------------------------------
[2017-04-22 22:05:03] FINER: 00-1 .   [1, 3]
[2017-04-22 22:05:03] FINER: 01-0 D   [4, 6]
[2017-04-22 22:05:03] FINER: ------------------------------
[2017-04-22 22:05:03] FINER: -011 .   [3, 11]
[2017-04-22 22:05:03] FINER: 101- . * [10, 11]
[2017-04-22 22:05:03] FINER: 1-10 . * [10, 14]
[2017-04-22 22:05:03] FINER: -110 .   [6, 14]
[2017-04-22 22:05:03] FINER: ------------------------------
[2017-04-22 22:05:03] FINER: 1-11 . * [11, 15]
[2017-04-22 22:05:03] FINER: 111- . * [14, 15]
[2017-04-22 22:05:03] FINER:
[2017-04-22 22:05:03] FINEST: Pronašao primarni implikant: 000- .   [0, 1]
[2017-04-22 22:05:03] FINEST: Pronašao primarni implikant: 0-00 .   [0, 4]
[2017-04-22 22:05:03] FINEST: Pronašao primarni implikant: 00-1 .   [1, 3]
[2017-04-22 22:05:03] FINEST: Pronašao primarni implikant: -011 .   [3, 11]
[2017-04-22 22:05:03] FINEST: Pronašao primarni implikant: -110 .   [6, 14]
[2017-04-22 22:05:03] FINEST:
[2017-04-22 22:05:03] FINER: Stupac tablice:
[2017-04-22 22:05:03] FINER: ================================
[2017-04-22 22:05:03] FINER: 1-1- .   [10, 11, 14, 15]
[2017-04-22 22:05:03] FINER:
[2017-04-22 22:05:03] FINEST: Pronašao primarni implikant: 1-1- .   [10, 11, 14, 15]
[2017-04-22 22:05:03] FINEST:
[2017-04-22 22:05:03] FINE:
[2017-04-22 22:05:03] FINE: Svi primarni implikanti:
[2017-04-22 22:05:03] FINE: 000- .   [0, 1]
[2017-04-22 22:05:03] FINE: 0-00 .   [0, 4]
[2017-04-22 22:05:03] FINE: 00-1 .   [1, 3]
[2017-04-22 22:05:03] FINE: -011 .   [3, 11]
[2017-04-22 22:05:03] FINE: -110 .   [6, 14]
[2017-04-22 22:05:03] FINE: 1-1- .   [10, 11, 14, 15]
```

Pogledajmo sada detaljnije metodu `chooseMinimalCover`. Ona enkapsulira sljedeća dva koraka algoritma: prvi korak stvara tablicu pokrivenosti minterma primarnim implikantima i temeljem toga traži bitne primarne implikante: one koje sigurno mora uzeti u minimalni oblik. Potom, ako preostanu nepokriveni mintermi, gradi funkciju pokrivenosti p u obliku produkta suma, svodi taj oblik u oblik sume produkata te vraća skup produkata koji imaju minimalan broj varijabli. Konačno, na temelju kardinaliteta tog skupa metoda gradi jedan ili više minimalnih zapisa tako što u svaki minimalni zapis uzme sve bitne primarne implikatne i tom popisu nadoda popis primarnih implikanata iz vraćenog produkta.

Gruba struktura ove metode, uz izbačene dijelove koji se bave logiranjem, prikazana je u nastavku.

```
private List<Set<Mask>> chooseMinimalCover(Set<Mask> primCover) {
	// Izgradi polja implikanata i minterma (rub tablice):
	Mask[] implicants = primCover.toArray(new Mask[primCover.size()]);
	Integer[] minterms = mintermSet.toArray(new Integer[mintermSet.size()]);

	// Mapiraj minterm u stupac u kojem se nalazi:
	Map<Integer,Integer> mintermToColumnMap = new HashMap<>();
	for(int i = 0; i < minterms.length; i++) {
		Integer index = minterms[i];
		mintermToColumnMap.put(index, i);
	}

	// Napravi praznu tablicu pokrivenosti:
	boolean[][] table = buildCoverTable(implicants, minterms, mintermToColumnMap);

	// Donji redak tablice: koje sam minterme pokrio?
	boolean[] coveredMinterms = new boolean[minterms.length];

	// Pronađi primarne implikante...
	Set<Mask> importantSet = selectImportantPrimaryImplicants(
			implicants, mintermToColumnMap, table, coveredMinterms);

	// Izgradi funkciju pokrivenosti:
	List<Set<BitSet>> pFunction = buildPFunction(table, coveredMinterms);

	// Pronađi minimalne dopune:
	Set<BitSet> minset = findMinimalSet(pFunction);

	// Izgradi minimalne zapise funkcije:
	List<Set<Mask>> minimalForms = new ArrayList<>();
	for(BitSet bs : minset) {
		Set<Mask> set = new LinkedHashSet<>(importantSet);
		bs.stream().forEach(i -> set.add(implicants[i]));
		minimalForms.add(set);
	}

	return minimalForms;
}
```

Metoda `buildCoverTable` vraća tablicu u kojoj retci odgovaraju primarnim implikantima, stupci mintermima funkcije a element na poziciji (i,j) je true ako i-ti primarni implikant pokriva minterm koji se nalazi u j-tom stupcu tablice.

Metoda `selectImportantPrimaryImplicants` vraća skup primarnih implikanata. Traži one primarne implikante koji prema tablici jedini pokrivaju neki minterm i njih dodaje u skup koji vraća. Za svaki primarni implikant koji doda u skup automatski u "retku" ispod tablice (`coveredMinterms`) označi da su pokriveni svi mintermi koje taj implikant pokriva.

Metoda `buildPFunction` zadužena je za izgradnju funkcije pokrivenosti koja će se koristiti u sljedećem koraku minimizacije. Pogledajmo primjer Booleove funkcije koju smo već dali. Analiza bitnih primarnih implikanata pronaći će da je bitan samo implikant `1-1-` koji pokriva minterme 10, 11, 14 i 15. Time nam ostaju nepokriveni mintermi 0, 1 i 3. Popis pronađenih primarnih implikanata je:

```
P0: 000- .   [0, 1]
P1: 0-00 .   [0, 4]
P2: 00-1 .   [1, 3]
P3: -011 .   [3, 11]
P4: -110 .   [6, 14]
P5: 1-1- .   [10, 11, 14, 15]
```

Primarnim implikantima dana su imena oblika *P* pa redni broj.

Minterm 0 možemo pokriti tako da uzmemo implikant *P0* ili implikant *P1*.
Minterm 1 možemo pokriti tako da uzmemo implikant *P0* ili implikant *P2*.
Minterm 3 možemo pokriti tako da uzmemo implikant *P2* ili implikant *P3*.

Kako moramo pronaći takav odabir primarnih implikanata koji pokrivaju I minterm 0 I minterm 1 I minterm 3, a iste redom možemo pokriti tako da uzmemo *P0* ILI *P1*, *P0* ILI *P2* te *P2* ILI *P3*, gradimo funkciju pokrivenosti $p(P0, ..., P7) = (P0 + P1) * (P0 + P2) * (P2 + P3)$. Metoda `buildPFunction` vraća ovaj oblik funkcije pokrivenosti.

Uočite: za funkciju pokrivenosti *p*, *P0* do *P7* tretiramo kao novi skup Booleovih varijabli sa sljedećom interpretacijom: ako je varijabla *Pi* istinita (tj. 1), tada ću primarni implikant *Pi* dodati pronađenom skupu bitnih primarnih implikanata pri izgradnji minimalnog zapisa funkcije.

Da bismo pronašli minimalni broj primarnih implikanata koje trebamo nadodati bitnim primarnim inplikantima, funkciju *p* trebamo svesti na oblik koji je suma produkata, i potom izdvojiti one produkte koji imaju najmanje članova *P*; ako takvih ima više, funkcija ima više minimalnih oblika. Krenimo ovo izmnožiti. Najprije množimo prvu i drugu zagradu:

$$\begin{aligned} p(P0, ..., P7) &= (P0 + P1) * (P0 + P2) * (P2 + P3) \\ &= (P0 + P0P2 + P0P1 + P1P2) * (P2 + P3) \\ &= P0P2 + P0P2 + P0P1P2 + P1P2 + P0P3 + P0P2P3 + P0P1P3 + P1P2P3 \end{aligned}$$

Primijetite sljedeće.

1. Množenjem *P0* i *P0* opet dobivamo *P0* – jedan od teorema Booleove algebre. Ne želimo pamtiti *P0P0* već samo *P0*.
2. Množenjem *P1* i *P0* dobivamo isto što i množenjem *P0* i *P1*; ne želimo razlikovati *P0P1* od *P1P0*.
3. Opća struktura izraza: imamo više zagrada koje se množe; unutar zagrade imamo sumu; u sumi se nalaze produkti.
4. Ne želimo razlikovati sumu *P0P1 + P0P2* od suma *P0P2 + P0P1*.

Uzevši u obzir navedena zapažanja, točke (1)+(2) sugeriraju da bi produkte bilo dobro pamtiti kao skup indeksa primarnih implikanata. U tom slučaju, množenje dvaju produkta odgovara uniji indeksa što rješava i problem idempotencije i poretka / komutativnosti. Stoga za pamćenje produkata koristite primjerke razreda `BitSet` i u njima postavite indekse produkata koje predstavljaju.

Uzevši u obzir točke (3)+(4), jednu zagradu modelirajte skupom objekata tipa `BitSet`, pri čemu skup mora biti takav da mu iterator obilazi elemente redosljedom dodavanja.

Čitavu početnu funkciju *p* modelirajte kao listu "zagrada", odnosno listu skupova čiji su elementi objekti tipa `BitSet`. Metoda `findMinimalSet` prima ovaj oblik, množi sve i vraća skup minimalno velikih produkata.

Dodajte u metodu `chooseMinimalCover` kod za logiranje koji će osigurati da će se njenim pozivom dobiti sljedeći ispis (obratite pažnju na razine):

```
[2017-04-22 22:40:12] FINE:
[2017-04-22 22:40:12] FINE: Bitni primarni implikanti su:
[2017-04-22 22:40:12] FINE: 1-1- .   [10, 11, 14, 15]
[2017-04-22 22:40:12] FINER:
[2017-04-22 22:40:12] FINER: p funkcija je:
[2017-04-22 22:40:12] FINER: [[{0}, {1}], [{0}, {2}], [{2}, {3}]]
[2017-04-22 22:40:12] FINER:
[2017-04-22 22:40:12] FINER: Nakon prevorbe p-funkcije u sumu produkata:
```

```
[2017-04-22 22:40:12] FINER: [[{0, 2}, {0, 3}, {0, 2, 3}, {0, 1, 2}, {0, 1, 3}, {1, 2}, {1, 2, 3}]]
[2017-04-22 22:40:12] FINER:
[2017-04-22 22:40:12] FINER: Minimalna pokrivanja još trebaju:
[2017-04-22 22:40:12] FINER: [{0, 2}, {0, 3}, {1, 2}]
[2017-04-22 22:40:12] FINE:
[2017-04-22 22:40:12] FINE: Minimalni oblici funkcije su:
[2017-04-22 22:40:12] FINE: 1. [1-1- .   [10, 11, 14, 15], 000- .   [0, 1], 00-1 .   [1, 3]]
[2017-04-22 22:40:12] FINE: 2. [1-1- .   [10, 11, 14, 15], 000- .   [0, 1], -011 .   [3, 11]]
[2017-04-22 22:40:12] FINE: 3. [1-1- .   [10, 11, 14, 15], 0-00 .   [0, 4], 00-1 .   [1, 3]]
```

Napomena: dio ovih ispisa su doslovno ispisi koje dobijete kad nad kolekcijom pozovete `toString()` - provjerite pa nemojte sami ići raditi nepotreban kod.

Dodajte razredu `Minimizer` getter metodu:

```
public List<Set<Mask>> getMinimalForms();
```

koja vraća pronađene minimalne oblike.

Razmislite što se događa ako je funkcija koju minizirate tautologija a što ako je kontradikcija? Vaš kod se ne smije raspasti u takvim situacijama – provjerite.

Problem 5.

Dodajte razredu `Minimizer` metodu:

```
public List<Node> getMinimalFormsAsExpressions();
```

Metoda na poziv mora stvoriti i vratiti listu izraza (jedan po minimalnom obliku). Pazite pri tome na "rubne" uvjete. Primjerice, ako je neki produkt minimizacijm sveden na varijablu, onda nema čvora AND iznad nje. Ako je čitava funkcija svedena na jedan produkt, onda nema čvora OR.

Problem 6.

Dodajte razredu `Minimizer` metodu:

```
public List<String> getMinimalFormsAsString();
```

Metoda na poziv mora stvoriti i vratiti listu stringova (jedan po minimalnom obliku) čijim bi se parsiranjem parserom koji ste razvili ponovno dobila funkcija koja ima jednak skup minterma. Pazite pri tome na "rubne" uvjete: ovdje kod ispisa problem će stvarati zagrade koje će biti sintaksno ispravne ali ih ne želimo. Primjerice, ne želimo izraze poput:

```
(A) OR (B AND C)
(B AND NOT C)
A OR (B AND C)
```

i slično. Zapravo, ako malo razmislimo, trebamo li ikada uopće zagrade kod ispisa minimalnih oblika?

Problem 7.

Dodajte glavni program `demo.QMC`. Program nakon pokretanja čita funkcije od korisnika, pokreće minimizaciju i ispisuje rezultat. Program prestaje s radom nakon što korisnik zada naredbu `quit`. Definicija čitave funkcije se zadaje u jednom retku (ovisno kako čitate, `BufferedReader` / `Scanner`, pročitajte jedan čitav redak pa ga analizirajte). Evo primjera interakcije korisnika i programa. Crveno je ispis programa, crno je unos korisnika.

```
> f(A,B,C,D) = NOT A AND NOT B AND (NOT C OR D) OR A AND C | NOT A AND B AND NOT D
1. A AND C OR NOT A AND NOT B AND NOT C OR NOT A AND NOT B AND D
2. A AND C OR NOT A AND NOT B AND NOT C OR NOT B AND C AND D
3. A AND C OR NOT A AND NOT C AND NOT D OR NOT A AND NOT B AND D
> f(A,B,C,D) = NOT A AND NOT B AND (NOT C OR D) OR A AND C | [4,6]
1. A AND C OR NOT A AND NOT B AND NOT C OR NOT A AND NOT B AND D
2. A AND C OR NOT A AND NOT B AND NOT C OR NOT B AND C AND D
3. A AND C OR NOT A AND NOT C AND NOT D OR NOT A AND NOT B AND D
> f(A,B,C,D) = NOT A AND NOT B AND (NOT C OR D) OR A AND C | NOT A AND NOT D
Pogreška: skup minterma i don't careova nije disjunktan.
> f(A,B,C,D) NOT A AND NOT B AND (NOT C OR D) OR A AND C | NOT A AND NOT D
Pogreška: funkcija nije ispravno zadana.
> quit
```

Redak sa specifikacijom funkcije navodi ime funkcije, u zagradama popis varijabli, znak jednako, izraz koji koji definira minterme, opcionalni znak `|` i ako je on prisutan, nakon njega mora doći izraz koji specificira *don't careove*. Na oba mjesta mogu doći izrazi u dva formata: ili u uglatim zagradama dolazi popis indeksa, ili je izraz u obliku koji se može isparsirati parserom koji ste razvili.

Prijedlog za obradu retka sa specifikacijom funkcije: nađite poziciju `=` te opcionalnu poziciju `|` i rascjepkajte string po njima.

- Lijevi dio: ime funkcije počinje sa slovom i dalje slijede slova ili brojke; imena varijabli isto tako; praznine su dopuštene posvuda (u smislu između imena i zagrade, između zagrade i varijable, ...) i zanemaruju se.
- Središnji dio i opcionalni desni dio: `trim`, pogledajte počinje li sa `[`; ako da, jedan način obrade, ako ne, drugi.

Ako su zadani izrazi koje šaljete parseru, oni mogu imati manje varijabli od popisa koji je formalno deklariran u funkciji, ali ne mogu imati više (ili različite) – to provjerite.

Za isprobavanje kako ste implementirali minimizator, isprobajte ga na sljedećim funkcijama.

- f(A,B) = (A:+:B)*(NOT A * NOT B + A * B)
- f(A,B) = (A:+:B)+(NOT A * NOT B + A * B)
- f(A,B,C) = !A*!B*!C + A*C + A*B*!C | !A*!B*C

Zatim ga isprobajte na funkcijama f(A,B,C,D,E) prikazanim u nastavku. Sve su funkcije potpuno specificirane (skup *don't careova* je prazan) pa navodim samo sumu minterma.

- 0, 1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 15, 16, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30
- 0, 1, 2, 3, 4, 5, 6, 8, 10, 11, 12, 13, 14, 15, 16, 17, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31
- 0, 2, 3, 4, 5, 6, 7, 8, 9, 11, 13, 14, 15, 16, 18, 19, 20, 21, 22, 24, 25, 26, 27, 28, 29, 30, 31
- 0, 1, 3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 15, 16, 17, 18, 21, 22, 23, 24, 25, 26, 27, 29, 30, 31
- 0, 1, 2, 4, 5, 7, 8, 9, 10, 11, 12, 14, 15, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 28, 29, 30, 31
- 0, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 31
- 0, 1, 2, 3, 4, 5, 7, 9, 10, 11, 13, 14, 16, 18, 19, 20, 21, 22, 23, 24, 25, 27, 28, 29, 30, 31
- 0, 1, 2, 3, 4, 6, 7, 9, 10, 12, 13, 14, 15, 16, 18, 19, 20, 21, 23, 24, 26, 27, 28, 30, 31

Prilikom isprobavanja ovog drugog niza primjera možda ćete poželjeti prikladnije podesiti razinu logiranja.

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 for this homework (whether it is yours old code or someones else). You can use Java Collection Framework and other parts of Java covered by lectures; if unsure – e-mail me. Document your code!

If you need any help, I have reserved a slot for consultations every day from Monday to Friday at 1 PM. Feel free to drop by my office.

All source files must be written using UTF-8 encoding. All classes, methods and fields (public, private or otherwise) must have appropriate javadoc.

You are expected to write tests for problems 1 and 2.
You are encouraged to write tests for other problems.

When your **complete** homework is done, pack it in zip archive with name `hw08-0000000000.zip` (replace zeros with your JMBAG, and yes, it is hw08 ZIP for hw07 project). Upload this archive to Ferko before the deadline. Do not forget to lock your upload or upload will not be accepted. Deadline is April 29th 2017. at 07:00 AM.