Wprowadzenie

Sieć Petriego to dwudzielny graf skierowany, którego węzłami są **miejsca** (ang. *place*) i **przejścia** (ang. *transition*).

Miejsca i przejścia są połączone **krawędziami** (ang. *arc*).

W klasycznych sieciach Petriego występują **krawędzie wejściowe** (ang. *input arc*), prowadzące z miejsca do przejścia, oraz **krawędzie wyjściowe** (ang. *output arc*), prowadzące z przejścia do miejsca.

Krawędzie wejściowe i wyjściowe mają dodatnie całkowite **wagi** (ang. *weight*).

W sieciach rozszerzonych, które rozważamy w tym zadaniu, z miejsca do przejścia mogą też prowadzić, nie posiadające wagi, **krawędzie zerujące** (ang. *reset arc*) i **krawędzie wzbraniające** (ang. *inhibitor arc*).

Znakowanie (ang. *marking*) sieci przyporządkowuje miejscom nieujemną całkowitą liczbę **żetonów** (ang. *token*).

W danym znakowaniu przejście jest **dozwolone** (ang. *enabled*), jeśli:

- w każdym miejscu, z którym rozważane przejście jest połączone krawędzią wejściową, liczba żetonów jest większa lub równa wadze tej krawędzi, oraz
- w każdym miejscu, z którym rozważane przejście jest połączone krawędzią wzbraniającą, liczba żetonów jest równa zero.

Sieć Petriego ma stan, reprezentowany przez aktualne znakowanie.

Stan sieci ulega zmianie w rezultacie **odpalenia** (ang. *fire*) dozwolonego przejścia.

Odpalenie przejścia to operacja niepodzielna, powodująca wykonanie kolejno trzech kroków:

- usunięcia z każdego miejsca, z którym odpalane przejście jest połączone krawędzią wejściową tylu żetonów, jaka jest waga tej krawędzi,
- usunięcia wszystkich żetonów z każdego miejsca, z którym odpalane przejście jest połączone krawędzią zerującą,
- dodania do każdego miejsca, z którym odpalane przejście jest połączone krawędzią wyjściową tylu żetonów, jaka jest waga tej krawędzi.

Polecenie

- Zaimplementuj sieć Petriego (6 pkt).
- Napisz program przykładowy **Alternator**, demonstrujący zastosowanie sieci Petriego do modelowania systemów współbieżnych i synchronizacji wątków (2 pkt).
- Napisz program przykładowy **Multiplicator**, demonstrujący zastosowanie sieci Petriego do realizacji obliczeń wielowątkowych (2 pkt).

Rozwiązanie ma być zgodne z poniższą specyfikacją.

Specyfikacja implementacji sieci Petriego

W pakiecie `petrinet` jest bezpieczna dla wątków (ang. *thread-safe*) implementacja sieci Petriego.

Zdefiniowane są klasy:

```
package petrinet;

import java.util.Collection;
import java.util.Map;
import java.util.Set;
...

public class PetriNet<T> {

    public PetriNet(Map<T, Integer> initial, boolean fair) {
        ...
    }

    public Set<Map<T, Integer>> reachable(Collection<Transition<T>> transitions) {
        ...
    }

    public Transition<T> fire(Collection<Transition<T>> transitions) throws InterruptedException {
        ...
    }

    ...

}
```

oraz:

```
package petrinet;

import java.util.Collection;
import java.util.Map;
...

public class Transition<T> {

    public Transition(Map<T, Integer> input, Collection<T> reset, Collection<T> inhibitor, Map<T,
Integer> output) {
        ...
    }

    ...

}
```

Konstruktor `Transition<T>(input, reset, inhibitor, output)` tworzy przejście między miejscami typu `T`. Przejście jest połączone:

- krawędziami wejściowymi z miejscami należącymi do `input.keySet()`, przy czym wagą krawędzi prowadzącej z miejsca `x` jest `input.get(x)`,
- krawędziami zerującymi z miejscami należącymi do `reset`,
- krawędziami wzbraniającymi z miejscami należącymi do `inhibitor`,
- krawędziami wyjściowymi z miejscami należącymi do `output.keySet()`, przy czym wagą krawędzi prowadzącej do miejsca `x` jest `output.get(x)`.

Konstruktor `PetriNet<T>(initial, fair)` tworzy sieć Petriego z miejscami typu `T`, w której:

- w stanie początkowym miejsce ma niezerową liczbę żetonów wtedy i tylko wtedy, gdy należy do `initial.keySet()`, przy czym `initial.get(x)` jest liczbą żetonów w miejscu `x`,
- kolejność budzenia wątków oczekujących na wykonanie opisanej poniżej metody `fire(transitions)` jest określona parametrem `fair`.

Metoda `reachable(transitions)` próbuje wyznaczyć zbiór wszystkich znakowań sieci, które są **osiągalne** (ang. *reachable*) z aktualnego jej stanu w rezultacie odpalenia, zero lub więcej razy, przejść z kolekcji `transitions`.

Jeśli zbiór osiągalnych znakowań jest skończony, to jest on wynikiem metody. Mapa `m`, należąca do tego zbioru, reprezentuje znakowanie, w którym miejsce ma niezerową liczbę żetonów wtedy i tylko wtedy, gdy jest elementem `m.keySet()`, przy czym `m.get(x)` jest liczbą żetonów w miejscu `x`.

Jeśli zbiór osiągalnych znakowań jest nieskończony, to wykonanie metody może się zapętlić lub zostać przerwane wyjątkiem.

Można zauważyć, że wywołanie `reachable(transitions)` z pustą kolekcją przejść daje zbiór znakowań sieci, którego jedynym elementem jest znakowanie aktualne.

Metoda `fire(transitions)` dostaje jako argument niepustą kolekcję przejść. Wstrzymuje wątek, jeśli żadne przejście z tej kolekcji nie jest dozwolone. Gdy w kolekcji są przejścia dozwolone, metoda odpala jedno z nich, dowolnie wybrane. Wynikiem metody jest odpalone przez nią przejście.

Jeżeli sieć Petriego została utworzona konstruktorem z argumentem `fair` równym `true`, to spośród tych wątków wstrzymanych metodą `fire(transitions)`, które w danej chwili można wznowić, wybierany jest wątek czekający najdłużej.

W przypadku przerwania wątku, metoda `fire(transitions)` zgłasza wyjątek `InterruptedException`.

Oprócz klas `PetriNet<T>` i `Transition<T>`, w rozwiązaniu mogą być inne definicje. Można je umieścić zarówno w pakiecie `petrinet`, jak i w pakietach pomocniczych.

Specyfikacja programu Alternator

Program uruchamiamy metodą statyczną `main(args)` klasy `Main`, zdefiniowanej w pakiecie `alternator`.

W programie rozważamy rozwiązanie problemu wzajemnego wykluczania dla trzech procesów, z dodatkowym wymaganiem, by w sekcji krytycznej nie mógł się znaleźć, dwa razy z rzędu, ten sam proces.

Program buduje sieć Petriego, będącą modelem tego systemu. Pisze, ile znakowań jest osiągalnych ze stanu początkowego. Sprawdza, czy wszystkie spełniają warunek bezpieczeństwa.

Następnie program rozpoczyna symulację systemu. Uruchamia trzy wątki `A`, `B`, `C`, działające w nieskończonych pętlach. W sekcji krytycznej każdy wątek pisze, dwoma kolejnymi wywołaniami `System.out.print()`, swoją nazwę i kropkę.

Dla każdego wątku, protokołem wstępnym i protokołem końcowym są pojedyncze wywołania metody `fire(transitions)` z odpowiednio dobranymi argumentami.

Jeśli zajdzie taka potrzeba, można uruchomić dodatkowy wątek pomocniczy, odpowiedzialny za odpalanie przejść, których nie odpalają wątki `A`, `B`, `C`.

Po 30 sekundach od rozpoczęcia symulacji wątek główny przerywa wszystkie pozostałe i kończy program.

Specyfikacja programu Multiplicator

Program uruchamiamy metodą statyczną `main(args)` klasy `Main`, zdefiniowanej w pakiecie `multiplicator`.

Program buduje sieć Petriego, mnożącą dwie nieujemne liczby całkowite `A` i `B`, wczytane z wejścia.

W stanie początkowym sieci jest miejsce, w którym jest `A` żetonów i miejsce, w którym jest `B` żetonów.

Iloczyn `A * B` jest liczony przez wielokrotne odpalanie przejść.

Po zakończeniu obliczenia:

- wyróżnione przejście końcowe staje się dozwolone,
- wszystkie pozostałe przejścia nie są dozwolone,
- w pewnym miejscu sieci jest `A * B` żetonów.

Po zbudowaniu sieci, program uruchamia cztery wątki pomocnicze. Każdy, w nieskończonej pętli, wykonuje metodę `fire(transitions)` z kolekcją przejść, w której są wszystkie przejścia z wyjątkiem końcowego.

Wątek główny czeka na zakończenie obliczeń. W tym celu wywołuje `fire(transitions)` z jednoelementową kolekcją przejść, w której jest tylko przejście końcowe.

Po zakończeniu obliczeń, wątek główny wypisuje iloczyn `A * B`, który odczytuje z aktualnego znakowania sieci i przerywa wątki pomocnicze.

Każdy wątek pomocniczy na zakończenie pracy pisze, ile przejść odpalił.

Wymagania techniczne

Program ma być w wersji 11 języka Java. Powinien się kompilować kompilatorem `javac` i działać poprawnie na komputerze `students`.

Wolno korzystać tylko ze standardowych pakietów zainstalowanych na `students`.

Jako rozwiązanie należy wysłać na moodla plik `ab123456.tar.gz`, gdzie `ab123456` to login na `students`.

W wysłanym pliku `.tar.gz` mają być katalogi pakietów z plikami źródłowymi `.java`.