Zadania na pracownię nr 9

Uwaga: W tym tygodniu dwa zadania! Termin oddania rozwiązań to poniedziałek, 7 maja 2018, godz 6:00.

Przy rozwiązanie tego zadanie **nie** wolno używać wbudowanych w Racketa procedur do generowania liczb pseudolosowych. Należy użyć generatora podobnego do tego, który opisany był na liście ćwiczeniowej nr 9 i jest dołączony w szablonie! Oba zadania mają wspólny szablon i należy je oddać jako jeden plik.

Zadanie A: Probabilistyczny test Fermata na pierwszość

Pierwszym zadaniem jest napisanie programu w języku WHILE, który bada pierwszość liczb za pomocą probabilistycznej wersji testu Fermata. Oczywiście, tak jak w przypadku poprzedniego języka, WHILE nie ma składni konkretnej – program istnieje jako Racketowa reprezentacja składni abstrakcyjnej (tak jak silnia na wykładzie).

Na SKOSie dostępny jest szablon, w którym język WHILE rozbudowujemy o kilka konstrukcji w wyrażeniach arytmetyczno-boolowskich: stałe `true` i `false`; operatory `and`, `or` i `not`; operację `mod` (reszta z dzielenia) i dość słabej jakości generator liczb pseudolosowych, którego można używać dzięki operatorowi `rand`, który przyjmuje jeden argument `max` i generuje losową liczbę z zakresu `0 ... max`. Na przykład, program, który zmiennej `x` przypisuje losową liczbę z zakresu `2 ... 10` reprezentujemy w następujący sposób:

```
'(x := (+ 2 (rand 8)))
```

Sprawdź eksperymentalnie, jak kiepskiej jakości jest generator liczb pseudoloswych wbudowanych w interpreter WHILE-a dostępny w szablonie![1]

Probabilistyczny test Fermata służy do tego, by z pewnym prawdopodobieństwem stwierdzić czy dana liczba n jest pierwsza. Jeśli test powie, że n jest liczbą złożoną, to na pewno taka właśnie ona jest. Jeśli test powie, że n jest liczbą pierwszą, to n jest liczbą pierwszą **z pewnym prawdopodobieństwem** –

[1] To nie jest część zadania, ale ta wiedza może się przydać podczas testowania rozwiązania!

na tyle dużym, że test bywa w wielu sytuacjach użyteczny. Jednym z zastosowań jest wydajne szukanie kandydatów na duże liczby pierwsze, użyteczne np. w szyfrowaniu RSA.

Sam test wygląda następująco. Jest on parametryzowany dwoma liczbami naturalnymi: n i k. Liczba n to liczba, którą testujemy, czy jest pierwsza, a k to liczba iteracji: im większe k, tym algorytm rozpoznaje liczby pierwsze z większym prawdopodobieństwem, ale za cenę dłuższego czasu działania. Wykonanie testu można opisać następującym pseudokodem:

Powtórz k razy:

1. Wybierz losowo wartość a z zakresu $2 \ldots (n-2)$
2. Jeśli $a^{n-1} \not\equiv 1 \pmod{n}$, to zakończ algorytm z odpowiedzią „**złożona**".

Zakończ algorytm z odpowiedzią „**pierwsza**".

W programie załóż, że przy interpretowaniu programu w pamięci znajdują się zmienne `n` i `k`, których wartości wyznaczają parametry testu. Wynik powinien być zapisany w zmiennej `composite`. Jeśli przyjmuje ona wartość `true`, oznacza to, że test mówi, że liczba jest złożona. Jeśli przyjmuje ona wartość `false`, oznacza to, że test mówi, że liczba jest pierwsza. Przykładowe zastosowanie, które powinno rozwiać wszelkie wątpliwości na ten temat, znajduje się w procedurze `probably-prime?` zamieszczonej w szablonie.

Rozwiązanie zadania A polega na rozbudowaniu szablonu dostępnego na SKOSie poprzez zastąpienie tymczasowej definicji wartości `fermat-test`. Jeśli czujesz taką potrzebę, możesz najpierw rozbudować język WHILE o elementy z ćwiczeń, takie jak operatory `++` i `--` czy pętlę `for`.

Pamiętaj przetestować rozwiązanie! Uważaj na błędy wynikające ze słabości generatora liczb losowych. Słabość tę można zniwelować, rozwiązując zadanie B.

Zadanie B: Rozszerzenie interpretera o generator liczb pseudolosowych

W szablonie dostępnym na SKOSie znajdują się:

- Prosty generator liczb pseudolosowych
- Składnia abstrakcyjna języka WHILE rozbudowana o konstrukcję `rand` w wyrażeniach

- Interpreter języka WHILE, który jednak nie wspiera w pełni konstrukcji `rand`.

Zadanie polega na rozbudowaniu interpretera o konstrukcję `rand` działającą przy użyciu podanego generatora. (W szczególności **nie wolno** użyć Racketowego generatora!)

Wskazówka: Zauważ, że całe zadanie opiera się na odpowiednim przekazywaniu stanu (początkowego ziarna), który zmienia się jedynie w momencie generowania nowej liczby pseudolosowej. Ponieważ `rand` jest konstrukcją w **wyrażeniach**, a nie instrukcją (typu `while` czy przypisanie), obliczenie wartości wyrażenia nie tylko zależy od stanu, ale także może go zmienić.

Uwaga: Jak ktoś nie chce, to nie musi stosować rozwiązań typu `st-app`. Rozwiązanie, które trzyma początkowe ziarno jako zmienną o określonej nazwie w pamięci jest akceptowane.

Przesyłanie rozwiązań

Rozwiązania należy **porządnie** przetestować i zamieścić testy w przesłanym pliku. Rozwiązania powinny być rozszerzeniem szablonu dostępnego na SKOS-ie.

Rozwiązanie każdego zadania należy przesłać jako plik o nazwie w formacie `nazwisko-imie.rkt`, jak zwykle bez spacji i polskich znaków.