Egzamin licencjacki/inżynierski — 12 luty 2013

Z zestawu sześciu zadań (Matematyka I, Matematyka II, Programowanie, Matematyka dyskretna, Algorytmy i struktury danych i Metody numeryczne) poniżej należy wybrać i przedstawić na osobnych kartkach rozwiązania trzech zadań. Za brakujące (do trzech) zadania zostanie wystawiona ocena nieostateczna z urzędu. Egzamin uważa się za zaliczony, jeśli student rozwiąże z oceną dostateczną co najmniej 2 zadania. Wtedy ocena z egzaminu jest średnią arytmetyczną ocen z trzech wybranych zadań. Na rozwiązanie zadań przeznacza się czas 3x40=120 minut. Po wyjściu z sali egzaminacyjnej w czasie egzaminu nie ma możliwości powrotu do tej sali i kontynuowania pisania egzaminu.

Matematyka I — Logika dla informatyków

Dla liczb naturalnych k niech $\underline{k}$ oznacza zbiór $\{n \in \mathbb{N} \mid n \leq k\}$. Rozważmy dwie funkcje $F : \underline{2013}^{\underline{2013}} \times \mathbb{N} \to \underline{2013}^{\underline{2013}}$ i $G : \underline{2013}^{\underline{2013}} \times \mathbb{N} \to \underline{2013}^{\underline{2013}}$ zadane wzorami

$$\begin{aligned}
F(f, 0) &= id_{\underline{2013}} \\
F(f, n+1) &= f \cdot F(f, n) \\
G(f, 0) &= id_{\underline{2013}} \\
G(f, n+1) &= G(f, n) \cdot f
\end{aligned}$$

gdzie A^B oznacza zbiór funkcji z B w A, $id_{\underline{2013}} : \underline{2013} \to \underline{2013}$ jest funkcją identycznościową zbioru $\underline{2013}$, a $\cdot$ oznacza standardową operację składania funkcji. Udowodnij, że $F = G$.

Matematyka II — Algebra

Zadanie 1.
Używając jedynie dodawań i porównań podać algorytm obliczenia najmniejszej wspólnej wielokrotności naturalnych a, b.

Zadanie 2.
W pierścieniu Z_{11} obliczyć wartość wyrażenia $\dfrac{2}{3} + \dfrac{4}{5}$.

Zadanie 3.
Dana jest grupa G i ustalony jej element: a. Sprawdzić czy przekształcenie $f : G \to G$, określone wzorem $f(x) = axa^{-1}$ jest izomorfizmem tej grupy z sobą.

Programowanie

Za zadanie można otrzymać 20 punktów. Aby otrzymac ocenę dostateczną, należy zdobyć 7 punktów, próg dla dst+ to 9p, dla db – 11p, dla db+ 13p, dla bdb – 15p.

Część 1. Gramatyka G_1 z symbolem startowym S nad alfabetem $\{a, b, c, d, e, f\}$ dana jest za pomocą następującego zbioru produkcji:

$$\{S \to \varepsilon,\ S \to aSd,\ S \to aSe,\ S \to aSf,\ S \to bSd,\ S \to bSe,\ S \to bSf,\ S \to cSd,\ S \to cSe,\ S \to cSf\}$$

Dla gramatyki G przez $L(G)$ rozumieć będziemy język generowany przez G. Dla wyrażenia regularnego r przez $\mathcal{L}(r)$ rozumiemy język opisany przez wyrażenie r.

a) Czy $abcdef$ należy do $L(G_1)$? Odpowiedź uzasadnij. **(1)**

b) Czy gramatyka G_1 jest jednoznaczna? Odpowiedź krótko uzasadnij. **(2)**

c) Zdefiniuj gramatykę G_2, taką że $L(G_1) = L(G_2)$, i gramatyka G_2 ma mniej produkcji niż G_1 (choć może mieć więcej symboli nieterminalnych). **(2)**

d) Przedstaw wyrażenie regularne lub gramatykę bezkontekstową generującą zbiór $A_1 = \mathcal{L}(a^*b^*f^*) \cap L(G_1)$ **(2)**

e) Napisz w języku imperatywnym funkcję, która bierze jako wejście napis i zwraca wartość logiczną, równą True wtedy i tylko wtedy, gdy ten napis należy do zbioru A_1. Możesz używać języka wybranego z następującej listy: C, C++, Java, C#, Python, Ruby, PHP, AWK, Pascal. **(3)**

Część 2. Liczbę nazwiemy *ciekawą*, jeżeli dzieli się przez sumę i przez iloczyn cyfr. Przykładowe ciekawe liczby to 36 albo 102. Twoim zadaniem będzie napisanie funkcji (predykatu) znajdującego wszystkie ciekawe liczby z zadanego zakresu. W zadaniu ważne jest (i również będzie oceniane) to, w jaki sposób podzielisz je na podproblemy (czyli jakie zdefiniujesz funkcje/predykaty pomocnicze). Oczywiście powinieneś opisać znaczenie każdego predykatu/funkcji którą definiujesz. Nie wolno korzystać z funkcji i predykatów standardowych (choć oczywiście możesz je sobie samemu zdefiniować, jeżeli uznasz to za celowe).

Wariant funkcjonalny. Haskell.

Napisz funkcję `ciekawe :: Int -> Int -> [Int]` zwracającą wszystkie ciekawe liczby z zadanego przedziału (zakładamy, że granice przedziału należą do przedziału).

Wariant logiczny. Prolog.

Napisz predykat `ciekawe(+A,+B,-L)` unifikujący `L` z listą wszystkich ciekawych liczb między `A` a `B` (zakładamy, że granice przedziału należą do przedziału).

Matematyka dyskretna

Oblicz sumę $\sum_{n=0}^{\infty} F_n 3^{-n}$, gdzie F_n jest n-tą liczbą Fibonacciego.

Algorytmy i struktury danych

Za rozwiązanie obydwu zadań z tej części można otrzymać w sumie do 9 punktów. Skala ocen: poniżej 3 punktów — ocena niedostateczna (egzamin niezdany), 3 punkty dają ocenę dostateczną, 4 — dostateczną z plusem, 5 — dobrą, 6 — dobrą z plusem, 7 albo więcej punktów daje ocenę bardzo dobrą.

Zadanie 1: stabilny trójpodział (4 punkty)

W sortowaniu szybkim *quick–sort* korzysta się z procedury *partition*, która dzieli dane względem piwota p (element dzielący) na dwie części: w pierwszej mają się znaleźć elementy $\leq p$ a w drugiej elementy $\geq p$. Napisz własną wersję procedury podziału *partition* $(A[0...n{-}1], p)$, która będzie stabilna, i która na danych umieszczonych w n–elementowej tablicy dokona trójpodziału, czyli podzieli dane na trzy części: w pierwszej znajdą się elementy $< p$, w drugiej $= p$ a w trzeciej $> p$.

1. Opracuj metodę, która efektywnie rozwiązuje to zadanie; zapisz swój algorytm w pseudokodzie i opisz go krótko.
2. Napisz co to znaczy, że algorytm jest stabilny. Uzasadnij, że Twoja procedura dokonuje podziału w sposób stabilny.
3. Oszacuj złożoność obliczeniową (czasową i pamięciową) opisanego algorytmu.

Wskazówka! Skorzystaj z techniki "dziel i zwyciężaj".

Zadanie 2: najmniejszy zbiór w bieżącym podziale (5 punktów)
Zaprojektuj efektywną strukturę danych do wykonywania ciągów podanych operacji na dynamicznie zmieniającym się podziale zbioru $\{1, 2, \ldots, n\}$:

- `init()` — utworzenie podziału $\{\{1\}, \{2\}, \ldots, \{n\}\}$;
- `union(A,B)` — połączenie zbiorów A i B bieżącego podziału w jeden zbiór;
- `find(x)` — wyznaczenie zbioru, do którego należy element x;
- `find-smallest()` — wyznaczenie zbioru w bieżącym podziale, który ma najmniej elementów.

1. Opracuj strukturę, która będzie efektywnie realizować wymienione operacje — w swoim rozwiązaniu wykorzystaj zbiory rozłączne w postaci drzewiastej.
2. Krótko ale precyzyjnie opisz zastosowaną w algorytmie strukturę danych reprezentującą zbiory rozłączne — napisz procedury *union* i *find* w pseudokodzie, oszacuj ich złożoność obliczeniową.
3. Opisz dokładnie, jak będzie działać procedura *find-smallest*. Jaki wpływ ma ta funkcjonalność w strukturze na działanie operacji *union* i *find* (opisz zrobione modyfikacje)? Jeśli do realizacji *find-smallest* będziesz wykorzystywał jakieś dodatkowe struktury danych to napisz jakie i uzasadnij w jakim celu są one stosowane.
4. Rozważ ciąg opracji, z których pierwsza to *init* a pozostałe to *union*, *find* i *find-smallest*. Jaki będzie koszt czasowy wykonania ciągu takich operacji? Odpowiedź uzasadnij.

Uwaga! Pesymistyczna złożoność pojedynczej operacji `find-smallest()` powinna być rzędu $O(\log n)$.

Metody numeryczne

1. Niech dana będzie nieosobliwa macierz $A \in \mathbb{R}^{n\times n}$ postaci

$$A := \begin{bmatrix} b_1 & c_1 & & & & & c_n \\ a_2 & b_2 & c_2 & & & & \\ & a_3 & b_3 & c_3 & & & \\ & & a_4 & b_4 & c_4 & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & a_{n-1} & b_{n-1} & c_{n-1} \\ a_1 & & & & & a_n & b_n \end{bmatrix}.$$

Zaproponuj algorytm znajdowania macierzy odwrotnej A^{-1} i podaj jego złożoność.