Egzamin licencjacki/inżynierski — 11 lutego 2015

Z sześciu poniższych zestawów zadań (Matematyka I, Matematyka II, Programowanie, Matematyka dyskretna, Algorytmy i struktury danych i Metody numeryczne) należy wybrać i przedstawić na osobnych kartkach rozwiązania trzech zestawów. Za brakujące (do trzech) zestawy zostanie wystawiona ocena niedostateczna z urzędu. Egzamin uważa się za zaliczony, jeśli student rozwiąże z oceną dostateczną co najmniej 2 zestawy. Wtedy ocena z egzaminu jest średnią arytmetyczną ocen z trzech wybranych zestawów. Na rozwiązanie 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

Ile jest funkcji $f : \mathbb{N} \to \mathbb{N}$ spełniających następujące warunki?

$$f(1) = 2 \tag{1}$$

$$f(i+j) = f(i) + f(j) - 1 \quad \text{dla wszystkich } i, j \in \mathbb{N} \tag{2}$$

Udowodnij poprawność swojej odpowiedzi.

Matematyka II — Algebra

Za zadania można otrzymać 13 punktów. Aby otrzymać ocenę dostateczną, należy zdobyć 3 punkty, próg dla dst+ to 5p, dla db – 7p, dla db+ 9p, dla bdb – 11p.

Zadanie 1. (6 punktów)
Podać zwartą postać wartości wyznacznika

$$D_n = \begin{vmatrix} 4 & 1 & & & & & \\ 2 & 4 & 1 & & & & \\ & 2 & 4 & 1 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & 2 & 4 & 1 \\ & & & & 2 & 4 \end{vmatrix}.$$

Zadanie 2. (3 punkty)
Obliczyć multiplikatywną odwrotność liczby 23 modulo 71, tzn. znaleźć x taki, że $23\,x \equiv_{71} 1$.

Zadanie 3. (4 punkty)
Sprawdzić czy wektory $w_1 = [1,1,1,0]^T$, $w_2 = [1,1,0,1]^T$, $w_3 = [1,0,1,1]^T$, $w_4 = [0,2,1,0]^T$ są liniowo niezależne
a. nad ciałem $\mathbb{R}$,
b. nad ciałem $\mathbb{Z}_3$.

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 $\{a, b\}$ dana jest za pomocą następującego zbioru produkcji:

$$\{S_1 \to aS_1a,\ S_1 \to bS_1b,\ S \to \varepsilon,\ S \to S_1S\}$$

Gramatyka G_2 z symbolem startowym S nad alfabetem $\{a, b\}$ dana jest za pomocą następującego zbioru produkcji:

$$\{S \to aSa,\ S \to bSb,\ S \to \varepsilon,\ S \to a,\ S \to b,\ \}$$

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 *abbabb* należy do $L(G_1)$? Odpowiedź uzasadnij. **(1)**

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

c) Przedstaw wyrażenie regularne lub gramatykę bezkontekstową generującą zbiór

$$A_1 = \mathcal{L}((a^*b^*a^*) \cap L(G_1)$$

(2)

d) Przedstaw wyrażenie regularne lub gramatykę bezkontekstową generującą zbiór

$$A_2 = \mathcal{L}((a + b)(ab + ba + aa + bb)*) \cap L(G_2)$$

(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_2. Możesz używać języka wybranego z następującej listy: C, C++, Java, C#, Python, Ruby, PHP, AWK, Pascal. **(3)**

Część 2. (**5p**) Napisz w Haskellu funkcję `f1`, która bierze listę liczb i usuwa z początku tej listy powtarzające się elementy, tak żeby `f1([1,1,1,2,3] == [1,2,3])`. Wykorzystaj tę funkcję przy tworzeniu funkcji `f2`, która usuwa powtarzające się elementy z początku i końca listy, tak żeby `f1([1,1,1,2,3,3,3] == [1,2,3])`. Możesz zdefiniować jedną funkcję pomocniczą, nie powinieneś korzystać z żadnej funkcji przez Ciebie niezdefiniowanej. Efektywność nie jest kluczowa, ale złożoność czasowa algorytmu nie powinna być większa niż liniowa.

Część 3. (**5p**) Rozważmy następujący program w Prologu:

```
sel(X,[X|Ys],Ys).
sel(X,[Y|Xs],[Y|Ys]) :- sel(X,Xs,Ys).

p([]).
p(Xs) :-
    sel(X, Xs, Rest),
    sel(X, Rest, Rest2),
    p(Rest2).
```

a) Zaproponuj nazwę dla predykatu `p`, która opisuje, co ten predykat robi (przy założeniu, że jest uruchamiany na listach nie zawierających zmiennych). Jeżeli nazwa nie w pełni wyjaśnia, co ten predykat robi, wyjaśnij to krótko.

b) Rozważmy prostą modyfikację powyższego kodu: zamiast drugiego wystąpienia zmiennej `X` damy inną zmienną `X1`. Wówczas predykat będzie robił co innego, co da się przedstawić za pomocą dużo prostszego kodu. Opisz, co sprawdza ten zmodyfikowany predykat i zaproponuj prostszą definicję (równoważną, przy założeniu, że interesuje nas jedynie wynik na listach stałych, i nie przejmujemy się liczbą nawrotów, generowaną przez predykat). Nie powinieneś korzystać z żadnych predykatów standardowych.

Matematyka dyskretna

Wyznacz zwarty wzór na a_n, gdzie

$$a_n = \sum_{k=1}^{n} (k+1)3^k.$$

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: najdłuższy wspólny zwarty podciąg (5 punktów)

Dane są dwa ciągi: n–elementowy ciąg $A = (a_0, a_1, \ldots, a_{n-1})$ oraz m–elementowy ciąg $B = (b_0, b_1, \ldots, b_{m-1})$. Podaj efektywny algorytm, który obliczy długość najdłuższego wspólnego podciągu zwartego dla zadanych ciągów, czyli wyznaczy największą możliwą wartość k taką, że istnieją identyczne fragmenty obu ciągów o tej długości $(a_i, a_{i+1}, \ldots, a_{i+k-1}) = (b_j, b_{j+1}, \ldots, b_{j+k-1})$ dla pewnych $0 \le i \le n-k$ oraz $0 \le j \le m-k$.

- Opisz ideę algorytm, który rozwiązuje ten problem.
- Krótko uzasadnij poprawność jego działania.
- Zapisz ten algorytm w pseudokodzie.
- Przeanalizuj złożoność obliczeniową (czasową i pamięciową) opisanej metody w zależności od parametrów n i m.

Zadanie 2: łagodniejsze zrównoważenie w drzewie AVL (4 punkty)

Drzewo AVL jest zrównoważonym drzewem BST. Definicja tego zrównoważenia mówi nam, że różnica wysokości poddrzew w każdym węźle jest nie większa niż 1. Chcielibyśmy złagodzić ten warunek w taki sposób, aby różnica wysokości poddrzew w każdym węźle była nie większa niż 2. Czy taki warunek zrównoważenia zagwarantuje nam w dalszym ciągu logarytmiczną wysokość drzwa?

- Krótko ale precyzyjnie opisz strukturę drzewa AVL.

- Wykaż, że po modyfikacji zrównoważenia w drzewie AVL jego wysokość będzie nie większa niż $2\log n$, gdzie n to liczba węzów w drzewie.
- Czy rotacje przywracające zrównoważenie w tradycyjnym drzewie AVL będą dalej poprawnie działały po modyfikacji warunku zrównoważenia? Odpowiedź uzasadnij.

Metody numeryczne

Przyjmijmy, że

$$\begin{cases} Q_0(x) = 1, \qquad Q_1(x) = x - c_1, \\ Q_k(x) = (x - c_k)Q_{k-1}(x) - d_k Q_{k-2}(x) \qquad (k = 2, 3, \ldots), \end{cases}$$

gdzie c_k, d_k są dane. Niech będzie $q(x) := \sum_{k=0}^{n} a_k Q_k(x)$ (a_k – dane). Udowodnij następujący *algorytm Clenshawa* obliczania wartości wielomianu q w punkcie $x \in \mathbb{R}$:

```
B[n+2]:=0; B[n+1]:=0;

  for k from n downto 0
    B[k]:=a[k]+(x-c[k+1])*B[k+1]-d[k+2]*B[k+2]

Return(B[0])
```

gdzie `a[k]`:= a_k, `c[k]`:= c_k, `d[k]`:= d_k oraz `x`:= x. Jakie zastosowanie w aproksymacji średniokwadratowej znajduje ten algorytm?