Egzamin licencjacki/inżynierski

10 września 2021

Informacja dla zdających egzamin na kierunku informatyka

Z sześciu poniższych zestawów zadań (Matematyka I, Programowanie, Matematyka dyskretna, Algorytmy i struktury danych, Metody numeryczne, Matematyka II) należy wybrać i przedstawić na osobnych kartkach rozwiązania trzech zestawów.

Informacja dla zdających egzamin na kierunku ISIM

Z sześciu poniższych zestawów zadań (Matematyka I, Programowanie, Matematyka dyskretna, Algorytmy i struktury danych, Metody numeryczne, Języki formalne i złożoność obliczeniowa) należy wybrać i przedstawić na osobnych kartkach rozwiązania trzech zestawów.

Informacja dla wszystkich zdających

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

Wprowadzenie. Zbiór $L(X)$ wszystkich skończonych list nad danym zbiorem X jest zdefiniowany indukcyjnie w następujący sposób:

- nil jest skończoną listą nad zbiorem X;
- jeśli x jest elementem zbioru X oraz xs jest skończoną listą nad zbiorem X to $x : xs$ jest skończoną listą nad zbiorem X.

Dla wszystkich zbiorów X operacja konkatenacji list $+\!\!+ : L(X) \times L(X) \to L(X)$ jest zdefiniowana w następujący sposób. Dla wszystkich elementów $x \in X$ oraz wszystkich list $xs, ys \in L(X)$ przyjmujemy

$$\begin{aligned} \mathsf{nil} +\!\!+ ys &= ys, \\ (x : xs) +\!\!+ ys &= x : (xs +\!\!+ ys). \end{aligned}$$

Dla wszystkich zbiorów X definiujemy funkcję $\mathit{length} : L(X) \to \mathbb{N}$ w następujący sposób. Dla dowolnego elementu $x \in X$ oraz dowolnej listy $xs \in L(X)$ przyjmujemy

$$\begin{aligned} \mathit{length}(\mathsf{nil}) &= 0, \\ \mathit{length}(x : xs) &= \mathit{length}(xs) + 1. \end{aligned}$$

Dla wszystkich zbiorów X definiujemy funkcję $\mathit{take} : \mathbb{N} \times L(X) \to L(X)$ w następujący sposób. Dla dowolnej liczby naturalnej n, dowolnego elementu $x \in X$ oraz dowolnej listy $xs \in L(X)$ przyjmujemy

$$\begin{aligned} \mathit{take}(0, xs) &= \mathsf{nil}, \\ \mathit{take}(n+1, \mathsf{nil}) &= \mathsf{nil}, \\ \mathit{take}(n+1, x : xs) &= x : \mathit{take}(n, xs). \end{aligned}$$

Właściwe zadanie.

1. Sformułuj zasadę indukcji strukturalnej dla list (w takiej postaci, żeby można było jej użyć w dowodzie w punkcie 2).

2. Korzystając z zasady indukcji sformułowanej w punkcie 1 udowodnij indukcyjnie, że dla dowolnego zbioru X, dowolnej liczby naturalnej n oraz dowolnych list $xs, ys \in L(X)$, jeśli $\mathit{length}(xs) \geq n$ to zachodzi równość

$$\mathit{take}(n, xs +\!\!+ ys) = \mathit{take}(n, xs).$$

Uwaga: Zadanie konceptualnie jest bardzo proste: łatwo zauważyć, że jeśli lista ma długość co najmniej n, to doklejenie na jej końcu czegokolwiek nie ma wpływu na wynik wyjęcia z niej pierwszych n elementów. Celem tego zadania jest sprawdzenie jak sobie radzisz ze zmiennymi wolnymi i związanymi, czy panujesz nad kwantyfikatorami, założeniami i celami do udowodnienia. Podczas oceniania zwrócimy uwagę na poprawność i klarowność rozumowania, w szczególności sprawdzimy czy

- w sformułowanej zasadzie indukcji występują zmienne wolne pierwszego rzędu (nie powinno ich być),
- w rozumowaniu zmienne drugiego rzędu (np. zbiory lub predykaty użyte w sformułowaniu zasady indukcji) są właściwie zainicjowane,

- założenie indukcyjne jest poprawnie sformułowane (powinno być jawnie sformułowane – będziemy w nim np. liczyć zmienne wolne),
- założenie indukcyjne jest poprawnie użyte (miejsce jego użycia powinno być dokładnie wskazane).

Matematyka II — Algebra

Zadanie 1. (5 punktów) Wartości własne macierzy $A \in \mathbb{R}^{3\times 3}$ to 2, 3, 5. Jaka jest wartość det(A)? (Odpowiedź z uzasadnieniem.)

Zadanie 2. (5 punktów) Znaleźć $x, y \in \mathbb{Z}$ takie, że $23x + 17y = 1$.

Zadanie 3. (4 punkty)

$$J = \begin{bmatrix} 5 & 2 & 0 & 0 \\ 2 & 2 & 0 & 0 \\ 0 & 0 & 7 & 8 \\ 0 & 0 & 2 & 1 \end{bmatrix}.$$

Podać wartości własne tej macierzy.

Progi punktowe: dst – 4, dst+ – 6, db – 8, db+ – 10, bdb – 12 punktów.

Programowanie – wariant A

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

Wygodną formą reprezentacji (skończonych) zbiorów elementów typu danych, dla którego mamy operację porównania, są drzewa przeszukiwań. W ramach przypomnienia, są to drzewa ukorzenione z elementami zbioru w wierzchołkach wewnętrznych, w których elementy te stanowią ograniczenia (górne lub dolne) dla elementów znajdujących się w odpowiednich poddrzewach. Jeśli takie drzewo jest dodatkowo zbalansowane (tj., wszystkie ścieżki od korzenia do liścia są tej samej długości), to jest ono efektywne. Niestety, w pełni zbalansowane drzewa binarne mają rozmiar postaci $2^n - 1$, co oznacza że większości zbiorów nie da się w ten sposób zareprezentować (podobnie dla dowolnej arności większej od 1).

Aby pokonać ten problem możemy pozwolić aby wierzchołki wewnętrzne drzewa miały *zmienną liczbę poddrzew*. W najprostszym przypadku, który tu rozważymy, wierzchołki wewnętrzne będą dwóch rodzajów:

- wierzchołki *binarne*, składające się z etykiety i dwóch poddrzew, ze standardową własnością drzewa przeszukiwań binarnych, i
- wierzchołki *ternarne*, składające się z dwóch etykiet i trzech poddrzew, przy czym własność drzewa przeszukiwania uogólnia się naturalnie: etykiety rozdzielają elementy trzech poddrzew względem porządku na etykietach.

Oprócz wierzchołków wewnętrznych mamy również liście, które nie zawierają żadnej informacji. Zbalansowane drzewa przeszukiwań o powyższej strukturze nazwiemy *2-3-drzewami*, przy czym przyjmujemy, że etykiety nie powtarzają się.

Zadanie 1. (5 p.) Zaproponuj reprezentację 2-3-drzew opisanych powyżej w wybranym języku funkcyjnym (Racket, OCaml, Haskell). Zadbaj o to, żeby wyrazić zarówno strukturę drzewa, jak i niezbędne niezmienniki używając właściwych dla wybranego języka form wyrazu, w szczególności zdefiniuj procedurę rozstrzygającą czy jej argument jest poprawnie sformowanym 2-3-drzewem.[1]

Zadanie 2. (5 p.) Zaproponuj reprezentację 2-3-drzew opisanych powyżej w wybranym języku imperatywnym (C, Python, Java itp.). Zadbaj o to, żeby wyrazić strukturę drzewa i niezmienniki używając właściwych dla danego języka form wyrazu i zdefiniuj procedurę sprawdzającą czy jej argument jest poprawnie sformowanym 2-3-drzewem.[2]

Zadanie 3. (3 p.) Dla reprezentacji z wybranego z powyższych zadań zaimplementuj dwuargumentową procedurę member sprawdzającą czy dany element należy do zbioru reprezentowanego przez dane drzewo. Następnie wyjaśnij, które z niezmienników Twojej struktury danych są niezbędne dla poprawności implementacji, a które nie.

Zadanie 4. (7 p.) Dla reprezentacji z wybranego z dwóch pierwszych zadań zaimplementuj dwuargumentową procedurę insert, która wstawia dany element do danego zbioru (reprezentowanego oczywiście jako drzewo). W zależności od reprezentacji, procedura może zwracać reprezentację powiększonego zbioru lub modyfikować przekazaną do niej strukturę. Uzasadnij, że procedura zachowuje niezmienniki opisane w definicji Twojej struktury danych.

[1] Jeśli używasz OCamla/Haskella, używaj systemu typów, ale nie nadużywaj go. To że jakąś własność da się wyrazić nie oznacza, że koniecznie należy to robić w warunkach egzaminu.

[2] Nie ma powodu żeby używać nietrywialnych bibliotecznych struktur danych: nie rób tego.

Wskazówka: *Zanim zaczniesz implementować* procedurę wstawiania zastanów się które niezmienniki mogą zostać naruszone w procesie wstawiania elementu do drzewa i w jaki sposób. Może być wygodnie zdefiniować pomocniczy typ danych opisujący drzewa, które naruszają niezmiennik w określony sposób.

Uwaga! Jakość kodu, zwłaszcza struktura definicji, ma znaczenie i będzie punktowana.

Programowanie – wariant B

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

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

$$\{S \to aSb,\ S \to c, S \to SxS\}$$

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

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

c) Przedstaw wyrażenie regularne lub gramatykę bezkontekstową generującą zbiór

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

. Odpowiedź uzasadnij. **(3p)**

d) 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 $L(G_1)$. Możesz używać języka wybranego z następującej listy: C, C++, Java, C#, Python, Ruby, Go, Rust. **(4p)**

Zadanie 2. (**6p**) Napisz w Haskellu dwuargumentową funkcję `nth`, która dla listy i liczby n zwraca n-ty element listy wejściowej (licząc od zera). Napisz również funkcję `f2`, która nie wywołuje funkcji `nth` i zwraca te same wartości, co funkcja `f` zdefiniowana poniżej

```
f k xs = nth k (qsort xs)
```

Zadanie 3. (**4p**) Formułę logiczną będziemy zapisywać używając operatorów `*`, `+`, `-` (odpowiednio jako koniunkcję, alternatywę i negację) oraz zmiennych prologowych. Przykładowo, formułę $x \vee (y \wedge \neg z)$ zapiszemy jako `X + Y * (-Z)`. Napisz predykat `sat/1`, który sprawdza, czy formuła jest spełnialna i w kolejnych nawrotach generuje wszystkie spełniające ją podstawienia. Możesz zdefiniować więcej niż 1 predykat, nie powinieneś korzystać z negacji.

Matematyka dyskretna

Zbiorem niezależnym w grafie nazywamy zbiór wierzchołków grafu, z których żadne dwa nie są połączone krawędzią. Pokryciem wierzchołkowym nazywamy zbiór takich wierzchołków, że każda krawędź grafu ma któryś z końców w tym zbiorze. Pokaż, że jeśli największy zbiór niezależny w G ma moc k, to najmniejsze pokrycie wierzchołkowe ma moc $n - k$.

Metody numeryczne

Za rozwiązanie zadań można otrzymać łącznie 12 punktów. Otrzymanie 4 pkt. gwarantuje ocenę dostateczną, próg dla dst+ to 5.5 pkt., dla db – 7 pkt., dla db+ 8.5 pkt., a dla bdb – 10 pkt.

1. **4 punkty** Zaproponuj efektywny algorytm obliczania z dużą dokładnością wartości $\sqrt[5]{a}$ $(a > 0)$ wykorzystując **jedynie** operacje arytmetyczne $(+, -, \cdot, /)$. **Uzasadnij** skuteczność przedstawionej metody.

2. **4 punkty** Wstęp. Niech dane będą wektory liczb rzeczywistych `x`:=$[x_0, x_1, \ldots, x_n]$ $(x_0 < x_1 < \cdots < x_n)$, `y`:=$[y_0, y_1, \ldots, y_n]$ i `z`:=$[z_0, z_1, \ldots, z_m]$ $(m, n \in \mathbb{N})$. Niech s_n oznacza naturalną interpolacyjną funkcję sklejaną trzeciego stopnia (*w skrócie*: NIFS3) spełniającą warunki $s_n(x_k) = y_k$ $(0 \leq k \leq n)$. W języku `PWO++` procedura `NSpline3(x,y,z)` wyznacza wektor $[s_n(z_0), s_n(z_1), \ldots, s_n(z_m)]$.

   Zadanie. Wiadomo, że NIFS3 odpowiadająca danym $(x_k, f(x_k))$ $(0 \leq k \leq 100)$ bardzo dobrze przybliża funkcję f. Można więc przypuszczać, że

   $$S_n := \int_{x_0}^{x_n} s_n(x)\,\mathrm{d}x$$

   jest bardzo dobrym przybliżeniem wartości całki $I := \int_{x_0}^{x_n} f(x)\,\mathrm{d}x$. Stosując procedurę `NSpline3` **tylko raz**, zaproponuj szkic **efektywnego algorytmu** wyznaczania wielkości S_n. Zadbaj więc m.in. o to, aby liczba współrzędnych wektora `z` (czyli wartość $m + 1$) **była możliwie jak najmniejsza**.

3. **4 punkty** Zaproponuj i opisz **szczegółowo** algorytm obliczania wyznacznika macierzy stopnia n. Wyznacz jego złożoność obliczeniową i pamięciową.

   Uwaga. Metody o złożoności obliczeniowej powyżej $O(n^3)$ nie wchodzą w grę.

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.

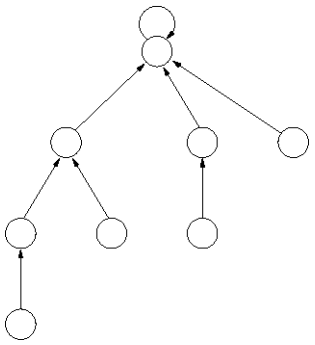
Zadanie 1: graf dwukolorowalny (4 punkty)

Kolorowanie grafu polega na takim przyporządkowaniu kolorów wierzchołkom grafu, aby żadne sąsiadujące ze sobą wierzchołki nie były pokolorowane tak samo. Skonstruuj algorytm, który sprawdzi, czy graf prosty $G = (V, E)$ da się pokolorować dwoma kolorami.

Precyzyjnie opisz swój algorytm i oszacuj jego złożoność obliczeniową. Uzasadnij poprawność opisanej metody.

Zadanie 2: drzewiasta reprezentacja zbiorów rozłącznych (5 punktów)

Rozważmy drzewiastą implementację struktury danych dla zbiorów rozłącznych. Każdy zbiór jest reprezentowany jako drzewo swoich elementów. Reprezentantem zbioru jest korzeń drzewa. Każdy węzeł ma bezpośredni wskaźnik do poprzednika (reprezentant zbioru, czyli korzeń, ma ten wskaźnik ustawiony na siebie). Operacja *Union* to połączenie zbiorów. Jest ona implementowana poprzez podłączenie korzenia jednego drzwa do korzenia drugiego drzewa; jeśli wskażemy reprezentantów dwóch zbiorów, to operacja ta wykona się w czasie stałym $O(1)$ (sprowadza się bowiem do zmiany wskaźnika w korzeniu jednego drzewa na korzeń drugiego drzewa). Operacja *Init* jest również bardzo prosta i polega na utworzeniu n jednoelementowych drzew, co zajmuje czas liniowy $O(n)$, gdzie n to liczba wszystkich elementów.



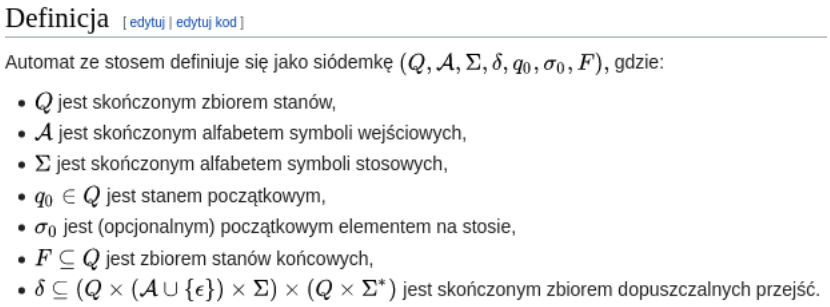
Operacja *Find*, to wskazanie reprezentanta zbioru, do którego należy dany element. Dzięki wskaźnikom koszt operacji *Find* jest proporcjonalny do głębokości węzła reprezentującego dany element w drzewie. Koszt tej operacji może być jednak duży, nawet rzędu $O(n)$, gdy powstałe w wyniku ciągu operacji *Union* drzewo będzie bardzo niezrównoważone (drzewo może przypominać listę i mieć wysokość równą nawet n).

Jaką technikę należy zastosować (dokładnie ją opisz), aby wysokość drzewa była co najwyżej logarytmiczna w stosunku do liczby elementów w drzewie? Jaki będzie wówczas koszt jednej operacji *Find*? Przeanalizuj jeszcze złożoność czasową wykonania ciągu m operacji *Union* i *Find*, przy czym w ciągu tym znajduje się $n - 1$ operacji *Union*. Wykaż, że zamortyzowany koszt wykonania na początku operacji *Init* a potem m operacji *Union* i *Find* wynosi $O(m + n \log n)$.

Druga część zadania polega na zastosowaniu struktury dla zbiorów rozłącznych do sprawdzenia, czy dwa wierzchołki w grafie prostym nieskierowanym należą do tej samej składowej spójności. Opisz ideę algorytmu a potem zapisz go w pseudokodzie (wraz z niezbędnymi komentarzami). Uzasadnij, że opisany algorytm działa poprawnie oraz oszacuj jego złożoność czasową.

Języki formalne i złożoność obliczeniowa

To zadanie dotyczy automatów ze stosem. Dla ustalenia uwagi, przyjmiemy definicję takiego automatu z Wikipedii:



Definicja [edytuj | edytuj kod]

Automat ze stosem definiuje się jako siódemkę $(Q, \mathcal{A}, \Sigma, \delta, q_0, \sigma_0, F)$, gdzie:

- Q jest skończonym zbiorem stanów,
- $\mathcal{A}$ jest skończonym alfabetem symboli wejściowych,
- Σ jest skończonym alfabetem symboli stosowych,
- $q_0 \in Q$ jest stanem początkowym,
- σ_0 jest (opcjonalnym) początkowym elementem na stosie,
- $F \subseteq Q$ jest zbiorem stanów końcowych,
- $\delta \subseteq (Q \times (\mathcal{A} \cup \{\epsilon\}) \times \Sigma) \times (Q \times \Sigma^*)$ jest skończonym zbiorem dopuszczalnych przejść.

Konfiguracja automatu to para (q, s), gdzie $q \in Q$ jest stanem, a $s \in \Sigma^*$ jest zawartością stosu. Przez $\to$ będziemy oznaczać taką relację ternarną, że $(q, s) \to_w (q', s')$ oznacza, że z konfiguracji (q, s) po słowie w automat może przejść do konfiguracji (q', s'). Taki automat akceptuje słowo w, jeśli istnieje konfiguracja (q, s) taka, że $q \in F$ oraz $(q_0, \epsilon) \to_w (q, s)$.

Automat z popsutym stosem działa podobnie, jak zwykły automat ze stosem. Jedyny problem z takim automatem polega na tym, że raz, w czasie swojego obliczenia, może zgubić stos, tzn. wszystko ze stosu może zniknąć. Język rozpoznawany przez taki automat to zbiór wszystkich słów, które ten automat może zaakceptować, a słowo w jest akceptowane przez taki automat, jeśli istnieją konfiguracje (q, s), (q', s') oraz słowa v, v' takie, że $w = vv'$ oraz $(q_0, \epsilon) \to_v (q, s)$, $(q, \epsilon) \to_{v'} (q', s')$ i $q' \in F$.

- (1 punkt) Czy każdy język akceptowany przez automat z popsutym stosem jest regularny?
- (2 punkty) Czy każdy język akceptowany przez automat z popsutym stosem jest bezkontekstowy?
- (2 punkty) Czy każdy język bezkontekstowy jest akceptowany przez jakiś automat z popsutym stosem?

Każdą odpowiedź należy udowodnić. Ocena to liczba zdobytych punktów.