MALACHOVIACUS
INFORMATICUS

Płock, 16 kwietnia 2015 r.

Instrukcje dla zawodników

Arkusze otwieramy na wyraźne polecenie komisji. Wszystkie poniższe instrukcje zostaną odczytane i wyjaśnione.

1. Arkusz składa się z **3 zadań**.
2. Każde zadanie składa się z **wprowadzenia** oraz kilku **pytań**.
3. Liczba punktów możliwa do uzyskania za każde pytanie podana jest przy jego treści. Suma tych punktów w każdym zadaniu wynosi 20.
4. Przed udzieleniem odpowiedzi na pytania **przeczytaj dokładnie** wprowadzenie oraz treści poleceń.
5. Swoje odpowiedzi zapisz **czytelnie** na przeznaczonych do tego arkuszach.
6. Do zapisu odpowiedzi używaj wyłącznie **długopisu lub pióra** z **czarnym lub niebieskim** tuszem. Do wykonywania rysunków możesz użyć ołówka.
7. Każdy arkusz odpowiedzi powinien zawierać odpowiedź, lub jej część, na **tylko jedno** zadanie.
8. Na pierwszej stronie każdego arkusza odpowiedzi, w prawym górnym rogu, zapisz czytelnie **swój kod** oraz **numer zadania**.
9. Czas na rozwiązanie zadań wynosi **120 minut**.

Powodzenia!

ZADANIE 1

Średnia arytmetyczna

Średnia arytmetyczna zestawu liczb równa jest sumie tych liczb podzielonej przez liczbę liczb w zestawie. Sumę danych liczb możemy policzyć używając następującej procedury:

```
1: procedura SUMA(T)
2:     suma ← 0
3:     dla każdego t ∈ T wykonaj
4:         suma ← suma + t
5:     koniec
6:     zwróć suma
7: koniec
```

W podobny sposób możemy zapisać procedurę DŁUGOŚĆ(T), która znajdzie liczbę elementów zestawu. Używając tych dwóch procedur możemy napisać:

```
1: procedura ŚREDNIA(T)
2:     zwróć SUMA(T) / DŁUGOŚĆ(T)
3: koniec
```

Obliczanie średniej arytmetycznej w ten sposób ma jednak dwie istotne wady. Po pierwsze, wymaga to przejrzenia całego zestawu liczb dwukrotnie, co dwukrotnie zwiększy czas wykonania algorytmu. Dużo lepiej byłoby napisać jedną procedurę, która w jednej pętli znajdzie zarówno długość jak i sumę elementów listy.

Po drugie, nawet jeśli każda z liczb jest dość mała, a co za tym idzie – średnia też jest dość niewielka, lecz liczb jest kilka milionów lub miliardów, ich suma będzie również duża. Ze względów technicznych, dokładne obliczenia na liczbach większych niż 3-4 miliardy mogą być dość problematyczne.

Istnieje jednak inny sposób liczenia średniej. Oznaczmy najpierw wszystkie nasze liczby przez $a_1, a_2, \ldots, a_{n-1}, a_n$. **Częściową średnią** będziemy nazywać średnią arytmetyczną pierwszych k liczb i oznaczać będziemy przez S_k. Tak na przykład $S_1 = a_1$, $S_2 = \frac{a_1+a_2}{2}$, $S_n = \frac{a_1+a_2+\cdots+a_n}{n}$.

Między kolejnymi wartościami S_k zachodzi następująca zależność:

$$S_k = \frac{(k-1)S_{k-1} + a_k}{k} \tag{1}$$

Możemy tę zależność przekształcić, aby pozbyć się działań na dużych liczbach, konkretniej iloczynu $(k-1)S_{k-1}$:

$$S_k = \left(1 - \frac{1}{k}\right) \cdot S_{k-1} + \frac{1}{k} \cdot a_k \tag{2}$$

Dodatkowo pamiętając, że $S_1 = a_1$, możemy dla każdego k od 1 do n policzyć kolejne wartości S_k. Kiedy dojdziemy do S_n, uzyskamy naszą pożądaną średnią arytmetyczną.

Pytania

Pytanie 1 Wzorując się na procedurze Suma(T) z treści zadania, napisz własną procedurę Długość(T), która policzy i zwróci liczbę elementów znajdujących się w T. [4 punkty]

Pytanie 2 Zmodyfikuj procedurę Suma(T) z treści zadania tak, aby otrzymać procedurę Średnia(T), która policzy średnią arytmetyczną liczb z T. Twoja procedura powinna jednocześnie sumować elementy zestawu i zliczać ich liczbę. Nie odwołuj się do innych procedur. [5 punktów]

Pytanie 3 Udowodnij poprawność zależności 2 między kolejnymi wartościami S_k przedstawionej w treści zadania. [5 punktów]

Pytanie 4 Napisz procedurę LepszaŚrednia(T), która policzy średnią liczb w T używając zależności 2 z treści zadania. [6 punktów]

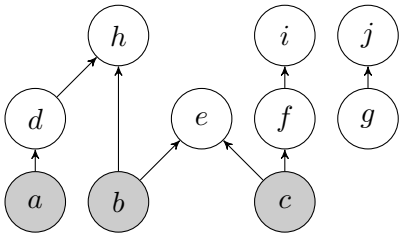
ZADANIE 2

Odśmiecanie pamięci

Programy napisane w językach programowania takich jak Java, Python czy PHP operują na tzw. **obiektach**. Na potrzeby tego zadania nie musisz wiedzieć dokładnie czym są obiekty. Musisz jednak wiedzieć, że:

- w czasie działania programu obiekty mogą być tworzone i usuwane,
- w czasie swojego istnienia obiekty zajmują miejsce w pamięci komputera, która jest ograniczona i należy ją oszczędzać,
- każdy obiekt może posiadać **odnośniki** do innych obiektów,
- niektóre z obiektów są dla programu **dostępne bezpośrednio**,
- obiekty są dla programu dostępne **pośrednio**, jeżeli istnieje ciąg odnośników do tego obiektu z jakiegoś obiektu dostępnego bezpośrednio.

Zależności między obiektami w pamięci możemy przedstawić za pomocą grafu skierowanego. Wierzchołki reprezentować będą obiekty, krawędzie zaś odnośniki. Niektóre z wierzchołków oznaczymy na szaro – są to obiekty dostępne dla programu bezpośrednio.



Rysunek 1: Przykładowy graf reprezentujący zależności między obiektami w pamięci. Obiekty a, b i c są dostępne bezpośrednio. Obiekty d, e, f, h oraz i dostępne są pośrednio.

Z naszych założeń o obiektach wynika, że mogą istnieć obiekty, które nie są dostępne dla programu. Przez to nie mogą być wykorzystywane. Na przykład, na rysunku 1 mamy dwa takie obiekty: g oraz j. Ponieważ niedostępne obiekty zajmują one miejsce w pamięci, a nie są już dłużej przydatne, chcemy je znaleźć i usunąć, aby zwolnić miejsce na nowe obiekty. Proces ten nazywamy **odśmiecaniem pamięci**.

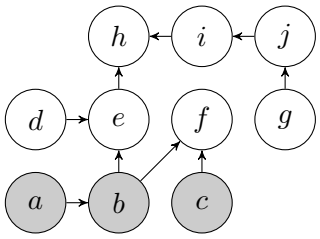
Jednym ze sposobów odśmiecania jest algorytm **zliczania odnośników**. Dla każdego obiektu w pamięci, który nie jest dostępny bezpośrednio, zliczamy liczbę obiektów, które się do niego odnoszą. Jeżeli jest ona równa 0, obiekt nie jest dłużej dostępny i może zostać usunięty z pamięci, razem ze swoimi odnośnikami. Na rysunku 1 przykładem takiego obiektu jest g.

Może się okazać, że w ten sposób liczba odniesień do innego obiektu spadnie do zera. Dlatego po usunięciu każdego obiektu powtarzamy cały proces dla wszystkich obiektów, do których się on odnosił itd. Przestajemy dopiero w momencie, gdy

w pamięci nie ma już obiektów, do których nie odnosi się żaden inny obiekt. Na rysunku 1, po usunięciu obiektu g, do obiektu j nie odnosi się żaden inny obiekt, zatem on również zostanie usunięty.

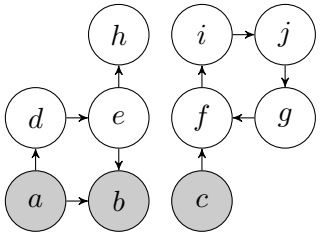
Pytania

Pytanie 1 Które z obiektów na rysunku poniżej nie są już dłużej dostępne i powinny zostać usunięte z pamięci? [2 punkty]



Pytanie 2 Na rysunku z poprzedniego pytania, ile obiektów odnosi się do każdego z obiektów: c, e, f, g, h i j? [3 punkty]

Pytanie 3 Spójrz na rysunek poniżej. Chcemy usunąć obiekt a. Które obiekty zostaną usunięte przy odśmiecaniu, jeżeli stosowane jest zliczanie odniesień? [4 punkty]



Pytanie 4 Spójrz na rysunek z poprzedniego pytania. Tym razem chcemy usunąć obiekt c. Które obiekty zostaną usunięte przy odśmiecaniu, jeżeli stosowane jest zliczanie odniesień? Które z obiektów nie zostaną usunięte, mimo tego że przestaną być dostępne? Dlaczego? [5 punktów]

Pytanie 5 Zapisz, używając dowolnego sposobu, algorytm przeszukiwania grafu zaczynając od jednego wierzchołka. Które wierzchołki zostaną odwiedzone, jeżeli przeszukamy graf obiektów zaczynając od każdego obiektu dostępnego bezpośrednio? Które pozostaną nieodwiedzone?

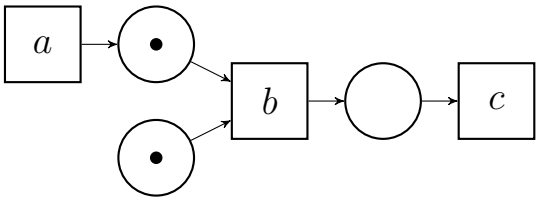
Bazując na tych obserwacjach, krótko opisz algorytm odśmiecania pamięci, który będzie wykorzystywał przeszukiwanie grafu do znalezienia niedostępnych już obiektów. Czy poradzi on sobie lepiej od zliczania odnośników z odśmiecaniem w sytuacji opisanej w poprzednim pytaniu? [6 punktów]

Zadanie 3

Proste sieci Petriego

Sieci Petriego to teoretyczny model, który służy do przedstawiania obliczeń równoległych. Prosta sieć Petriego to graf skierowany, zawierający dwa różne rodzaje wierzchołków: **przejścia** i **miejsca**. Każda z krawędzi w tym grafie prowadzi z przejścia do miejsca lub z miejsca do przejścia. Nie może istnieć krawędź łącząca dwa przejścia lub dwa miejsca.

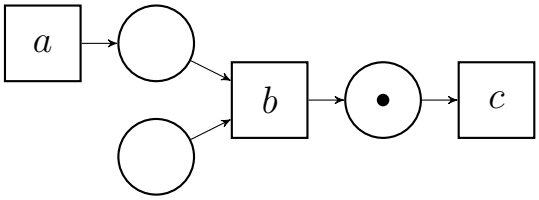
Dodatkowo, każde z miejsc może być w danym momencie **pełne** (zawierające **żeton**) lub **puste**. Umiejscowienie żetonów w sieci oraz ich liczba mogą się zmieniać. Pojedynczy rozkład żetonów nazywamy **konfiguracją**.



Rysunek 2: Przykładowa prosta sieć Petriego. Przejścia oznaczone są kwadratami, miejsca zaś kołami. Miejsca pełne posiadają w środku czarne kropki – żetony.

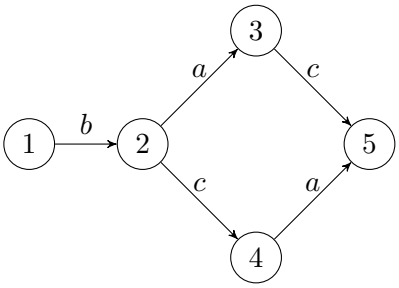
Każde przejście posiada pewną liczbę (może być 0) miejsc **wejściowych**, to jest takich, które posiadają krawędź kierującą do tego wydarzenia, oraz miejsc **wyjściowych**, czyli takich, do których prowadzi krawędź wychodząca z danego przejścia.

Dane przejście jest **aktywne**, jeżeli wszystkie jego miejsca wejściowe są pełne, a wszystkie jego miejsca wyjściowe są puste. Na przykład na rysunku 2 aktywne jest jedynie przejście b. Każde z aktywnych przejść możemy **odpalić**, co spowoduje opróżnienie wszystkich jego miejsc wejściowych i zapełnienie wszystkich miejsc wyjściowych.



Rysunek 3: Sieć z rysunku 2 po odpaleniu przejścia b. Przejście b nie jest już aktywne, za to aktywne są przejścia a i c.

Dla każdej sieci Petriego możemy narysować **graf przejść**, czyli graf skierowany, w którym każdy z wierzchołków reprezentuje konfigurację sieci, natomiast krawędzie przedstawiają odpalenia przejść. Każda z krawędzi oznaczona jest literą odpowiadającą przejściu, którego odpaleniu odpowiada.



Rysunek 4: Graf przejść dla sieci na rysunkach powyżej. Konfiguracje oznaczone zostały kolejnymi liczbami naturalnymi. Konfiguracja 1 widoczna jest na rysunku 2, zaś konfiguracja 2 na rysunku 3.

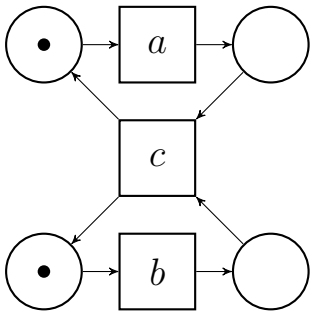
Pytania

Pytanie 1 Spójrz na rysunek 5 poniżej. Wypisz wszystkie aktywne przejścia znajdujące się w sieci na nim przedstawionej. [2 punkty]

Pytanie 2 Wybierz jedno z przejść wypisanych w poprzednim pytaniu. Jak będzie wyglądać sieć po odpaleniu wybranego przejścia? Narysuj ją. [2 punkty]

Pytanie 3 Narysuj konfiguracje 3, 4 i 5 dla przykładowej sieci z treści zadania, według oznaczeń z grafu na rysunku 4. Narysuj również jedną konfigurację tej sieci, która nie znajduje się w jej grafie przejść. [8 punktów]

Pytanie 4 Narysuj graf przejść dla sieci Petriego z rysunku poniżej. Narysuj wszystkie konfiguracje. [8 punktów]



Rysunek 5: Rysunek do pytań 1, 2 i 4.