Materiały dla finalistów

Malachoviacus Informaticus 2015

7 kwietnia 2015

Wprowadzenie

Poniższy dokument zawiera opisy zagadnień, które będą niezbędne do rozwiązania zadań w drugim etapie konkursu. Polecamy dokładnie zapoznać się z dostarczonymi wyjaśnieniami. Przydatne może być również wyszukanie dodatkowych informacji na poniższe tematy na własną rękę.

Zapoznanie się i zrozumienie informacji zawartych poniżej jest częścią konkursu. Rozwiązywanie zadań na końcu każdej z sekcji **nie** jest wymagane, jednak może pomóc w lepszym zrozumieniu materiału.

1 Algorytmy i pseudokod

Formalnie, **algorytm** to skończony ciąg jasno określonych czynności, koniecznych do wykonania pewnego rodzaju zadań. Mniej formalnie, algorytm to po prostu pewien sposób wykonania czegoś. Przykładem algorytmu z życia codziennego jest na przykład przepis kucharski. W informatyce zajmujemy się raczej algorytmami operującymi na liczbach.

Istnieją różne sposoby zapisu algorytmów. Najbardziej naturalnym dla człowieka jest zapis w postaci **listy kroków**. Jednakże dla bardziej skomplikowanych algorytmów może się on stać bardzo nieporęczny.

Listing 1 Algorytm znajdowania minimum w formie listy kroków.

1. Zapamiętaj pierwszą liczbę zestawu jako dotychczasowe minimum.
2. Przejdź do kolejnej liczby.
3. Jeżeli liczba ta jest mniejsza od dotychczasowego minimum, zapamiętaj ją jako nowe minimum.
4. Jeżeli zostały jeszcze jakieś liczby, wróć do kroku 2.

Z drugiej strony, zapisem najlepiej zrozumiałym dla komputera jest zapis w **języku programowania**. Zrozumienie takiego zapisu może być jednak trudne dla osoby nieobytej z danym językiem, czy programowaniem w ogóle.

Listing 2 Funkcja znajdująca minimum tablicy liczb, zapisana w C++.

```
int minimum(int *T, int n) {
  int m = T[0];
  for (int i = 0; i < n; i++) {
    if (T[i] < m) {
      m = T[i];
    }
  }

  return m;
}
```

Trzecia metoda zapisu, **pseudokod**, stanowi połączenie dwóch pozostałych. Pseudokod opiera się na uproszczonej składni opartej na popularnych językach programowania, jednak możemy w nim używać także sformułowań języka naturalnego.

Listing 3 Procedura znajdująca minimum, zapisana w pseudokodzie.

```
1: procedura MINIMUM(T)
2:     min ← ∞
3:     dla każdego t ∈ T wykonaj:
4:         jeżeli t < min to wykonaj:
5:             min ← t
6:         koniec
7:     koniec
8:     zwróć min
9: koniec
```

Przyjrzyjmy się poszczególnym elementom powyższej wersji pseudokodu.

1.1 Procedury

Podstawą zapisu algorytmów w pseudokodzie są **procedury**. Każda procedura posiada swoją nazwę, przyjmuje jakieś dane wejściowe i wykonuje obliczenia stanowiące część lub całość algorytmu. Procedura może dodatkowo zwracać jakąś wartość, będącą wynikiem jej działania. Przykład na listingu 4. Możemy także wykorzystywać opisane wcześniej procedury wewnątrz innych procedur, jak na listingu 5.

Listing 4 Szkielet procedury.

```
1: procedura NAZWA(dane)
2:     działania na danych
3:     zwróć wynik
4: koniec
```

Listing 5 Procedura obliczająca amplitudę (różnicę maksimum i minimum) zestawu liczb. Procedura MINIMUM została zdefiniowana wyżej, procedura MAKSIMUM w podobny sposób znajduje największą wartość zestawu.

```
1: procedura AMPLITUDA(T)
2:     zwróć MAKSIMUM(T) - MINIMUM(T)
3: koniec
```

1.2 Zmienne

Często skomplikowane obliczenia potrzebujemy rozbić na kilka mniejszych części. Musimy być w stanie zapamiętać i odwołać się później do wyniku poprzednich działań. Służą do tego **zmienne**. Możemy napisać na przykład $n \leftarrow 5$, co oznaczać będzie, że od teraz pisząc n mamy na myśli 5. Przykład na listingu 6.

Listing 6 Procedura obliczająca wartość wyrażenia $(a + b)^2$ dla danego a i b. Tworzy i wykorzystuje zmienną s przechowującą sumę danych liczb.

```
1: procedura KWADRATSUMY(a, b)
2:     s ← a + b
3:     zwróć s · s
4: koniec
```

1.3 Wyrażenia warunkowe

Możemy również sprawdzać czy pewne warunki są prawdziwe i od tego uzależnić który fragment kodu będzie wykonany jako następny. Używamy do tego **wyrażeń warunkowych**, tak jak na listingu 7.

1.4 Listy

Na zestawach liczb działamy używając **list**. Listy, podobnie jak liczby, zapisywać możemy w zmiennych. Listy zapisujemy w nawiasach kwadratowych, oddzielając kolejne liczby przecinkami, np. $[8, 15, 16, 23]$. Aby odwołać się do pojedynczego elementu listy wewnątrz zmiennej, zapisujemy najpierw nazwę zmiennej, a następnie w nawiasach kwadratowych numer miejsca na liście, do którego chcemy się odwołać. Przykład na listingu 8. **Uwaga!** Numeracja zaczyna się od zera!

Listing 7 Procedura obliczająca wartość bezwzględną danej liczby.

```
1: procedura WartośćBezwzględna(n)
2:     jeżeli n > 0 to wykonaj:
3:         w ← n
4:     w przeciwnym razie wykonaj:
5:         w ← −n
6:     koniec
7:     zwróć w
8: koniec
```

Listing 8 Procedura pokazująca jak działać na listach. Zwraca jako wynik liczbę 13. Czy wiesz dlaczego?

```
1: procedura NicKonkretnego
2:     T ← [1, 1, 2, 3, 0, 0]
3:     T[4] ← T[2] + T[3]
4:     T[5] ← T[3] + T[4]
5:     zwróć T[4] + T[5]
6: koniec
```

1.5 Pętle

Możemy również wykonać jakiś fragment kodu dla każdego elementu danej listy, używając **pętli „dla każdego”**. Przykład na listingu 9. Innym rodzajem pętli jest **pętla „dopóki”**, która powtarza pewien fragment kodu tak długo jak zadany warunek jest prawdziwy. Warunek sprawdzany jest przed każdym wykonaniem kodu znajdującego się w pętli. Przykład na listingu 10.

Listing 9 Procedura znajdująca sumę liczb danej listy T. Wykorzystuje pętlę „dla każdego”.

```
1: procedura Suma(T)
2:     s ← 0
3:     dla każdego t ∈ T wykonaj:
4:         s ← s + t
5:     koniec
6: koniec
```

1.6 Zadania

Zadanie konkursowe będzie kłaść nacisk na rozumienie prostych algorytmów zapisanych przy pomocy pseudokodu i ich modyfikację według zaleceń. Zadania nie będą wymagać pisania pełnych procedur od podstaw.

1. Upewnij się, że rozumiesz jak działa każda z procedur opisanych powyżej. Używając kartki i ołówka prześledź instrukcje w każdej z

Listing 10 Procedura obliczająca wartość n^k dla danego n i k. Wykorzystuje pętlę „dopóki".

```
1: procedura Potęga(n, k)
2:     w ← 1
3:     dopóki k > 0 wykonaj:
4:         w ← w · n
5:         k ← k − 1
6:     koniec
7:     zwróć w
8: koniec
```

nich, zapisując wartości jakie przyjmować będą zmienne po wykonaniu kolejnych linii pseudokodu. Sprawdź, na przykład, że wynikiem:

(a) Minimum$([5, 2, 3, 1, 5])$ jest 1,

(b) KwadratSumy$(2, 3)$ jest 25,

(c) WartośćBezwzględna(-15) jest 15,

(d) Suma$([1, 2, 3])$ jest 6,

(e) Potęga$(2, 5)$ jest 32.

2. Bazując na procedurze Minimum napisz procedurę Maksimum, która będzie znajdować i zwracać największą wartość danego zestawu liczb.

3. Bazując na procedurze WartośćBezwzględna napisz procedurę Parzysta, która zwróci 1, jeżeli dana liczba jest parzysta, 0 w przeciwnym wypadku. Możesz użyć zapisu $n \mid k$ oznaczającego „n jest dzielnikiem k".

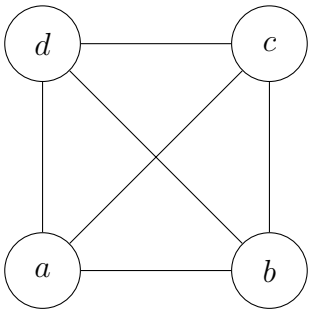
4. Bazując na procedurze Suma napisz procedurę Długość, która policzy ile liczb znajduje się w zadanym zestawie. Używając tych procedur i bazując na procedurze Amplituda napisz procedurę Średnia, która policzy średnią arytmetyczną zadanego zestawu liczb.

5. Bazując na procedurze Potęga napisz procedurę SumaN, która policzy sumę liczb naturalnych od 1 do pewnego zadanego n włącznie. Następnie napisz procedurę Silnia, która policzy silnię zadanej liczby. Silnia z n (zapisywane $n!$) to iloczyn kolejnych liczb naturalnych od 1 do n włącznie. Na przykład $4! = 1 \cdot 2 \cdot 3 \cdot 4 = 24$.

2 Wprowadzenie do teorii grafów

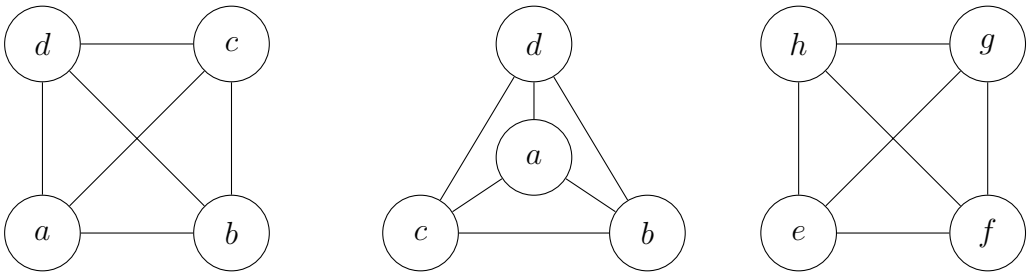
Graf to matematyczna konstrukcja składająca się z **wierzchołków** oraz łączących je **krawędzi**. Wierzchołki będziemy oznaczać małymi literami

alfabetu (a, b, c, ...), a krawędzie jako pary wierzchołków, np. (a, c), (b, b) itp. Dwa wierzchołki połączone krawędzią nazywamy **sąsiadami**.



Rysunek 1: Przykład grafu posiadającego wierzchołki a, b, c i d oraz krawędzie (a, b), (a, c), (a, d), (b, c), (b, d) i (c, d).

Diagramy takie jak na rysunku 1 stanowią jedynie graficzną reprezentację grafów. Położenie wierzchołków ani kształt krawędzi nie są w rzeczywistości częścią grafu. Również oznaczenia literowe wierzchołków istnieją jedynie dla rozróżnienia wierzchołków wewnątrz grafu. Możemy je dowolnie zmieniać, nie wpływając w żaden sposób na graf.

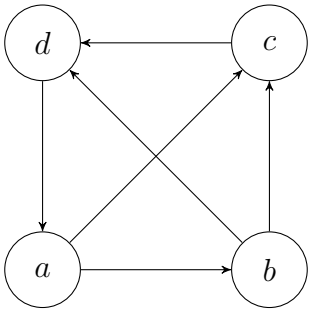


Rysunek 2: Powyższe grafy, mimo różnic w wyglądzie i oznaczeniach wierzchołków, w rzeczywistości są identyczne.

2.1 Grafy skierowane i nieskierowane

Graf skierowany to taki, w którym każda z krawędzi ma przypisany kierunek. W takim grafie krawędź (u, v) różni się od krawędzi (v, u). Na diagramach oznaczamy to przy użyciu strzałki na końcu każdej z krawędzi.

W **grafie nieskierowanym** krawędzie nie posiadają kierunku. W takim grafie krawędzie (u, v) i (v, u) są identyczne. Na diagramach krawędzie będą po prostu ciągłymi liniami, tak jak na rysunkach 1 i 2.



Rysunek 3: Przykład grafu skierowanego.

2.2 Ścieżka w grafie

Ścieżką w grafie nazywamy ciąg krawędzi taki, że każda kolejna krawędź zaczyna się tam, gdzie poprzednia się skończyła. Przykładem ścieżki w grafie na rysunku 1 będzie $(a, c), (c, b), (b, d)$. Dla skrócenia zapisu możemy podawać jedynie kolejne wierzchołki (np. a, c, b, d), jednak należy pamiętać, że ścieżka jest ciągiem krawędzi, nie wierzchołków.

Zwróć uwagę, że mimo posiadania tych samych wierzchołków i niemal tych samych krawędzi, graf z rysunku 3 nie posiada ścieżki a, c, b, d, ponieważ nie istnieje w nim krawędź (b, c), a jedynie (c, b).

Cykl to taka ścieżka w grafie, która zaczyna się i kończy w tym samym wierzchołku. W grafie na rysunku 1 mamy kilka cykli, na przykład a, b, d, a. Graf, który nie posiada cykli nazywamy **acyklicznym**.

2.3 Przeszukanie grafu

Mając dany graf, chcemy być w stanie odpowiedzieć na pytanie „Czy istnieje w tym grafie ścieżka zaczynająca się w wierzchołku u, a kończąca w v?" lub znaleźć wszystkie takie wierzchołki v, że istnieje ścieżka z u do v. Pomocny będzie w tym algorytm przeszukania grafu.

Każdy z wierzchołków grafu w tym algorytmie może być oznaczony jako „nieodwiedzony", „do odwiedzenia" lub „odwiedzony". Czasem używa się także kolorów, odpowiednio: biały, szary i czarny.

Początkowo wszystkie wierzchołki oznaczamy jako nieodwiedzone, ponieważ nasz algorytm nie dotarł jeszcze do żadnego z nich. Chcemy zacząć od wierzchołka s, zatem oznaczamy go jako „do odwiedzenia". Następnie zaczynamy właściwe przeszukanie: wybieramy jeden z wierzchołków do odwiedzenia (pierwszy zawsze będzie s), oznaczamy go jako odwiedzonego. Każdy wierzchołek połączony krawędzią z właśnie odwiedzonym wierzchołkiem oznaczamy jako „do odwiedzenia", ale tylko, jeżeli nie był uprzednio odwiedzony.

Jeżeli nie ma już w grafie wierzchołków „do odwiedzenia", kończymy działanie algorytmu. Jeżeli jakiś wierzchołek został oznaczony jako „odwiedzony",

Listing 11 Bazowy algorytm przeszukania grafu G, zaczynając od wierzchołka s. Po zakończeniu działania tej procedury, wierzchołki do których można dotrzeć z wierzchołka s będą oznaczone jako „odwiedzone".

```
 1: procedura PrzeszukanieGrafu(G, s)
 2:     dla każdego wierzchołka u w grafie G wykonaj:
 3:         oznacz u jako „nieodwiedzony"
 4:     koniec
 5:     oznacz s jako „do odwiedzenia"
 6:     dopóki istnieją wierzchołki oznaczone „do odwiedzenia" wykonaj:
 7:         wybierz jeden z nich; nazwijmy go u
 8:         oznacz u jako „odwiedzony"
 9:         dla każdego wierzchołka v takiego że (u, v) jest krawędzią w G
    i v nie jest „odwiedzony" wykonaj:
10:             oznacz v jako „do odwiedzenia"
11:         koniec
12:     koniec
13: koniec
```

to znaczy, że istnieje ścieżka z s do tego wierzchołka.

Zależnie od tego w jakiej kolejności będziemy wybierać kolejne wierzchołki oznaczone „do odwiedzenia", otrzymamy przeszukiwanie wszersz lub przeszukiwanie w głąb. Jeżeli chcesz, możesz przeczytać poniższe artykuły na Wikipedii:

- `http://pl.wikipedia.org/w/index.php?title=Przeszukiwanie_wszerz&stableid=42027240`
- `http://pl.wikipedia.org/w/index.php?title=Przeszukiwanie_w_g%C5%82%C4%85b&stableid=42027233`

Nie jest to wymagane, jednak może pomóc Ci w lepszym zrozumieniu idei przeszukiwania grafu.

2.4 Zadania

Dla obycia się z ideą grafów możesz wykonać następujące zadania:

1. I edycja, I etap, zadanie 2: „Problem mostów Królewieckich"
2. arkusz przykładowy, zadanie 1: „Drzewa przeszukiwań binarnych"
3. I edycja, II etap, zadanie 3: „Obliczenia równoległe po raz drugi"
4. Spróbuj narysować taki graf skierowany, który po wykonaniu na nim algorytmu przeszukania będzie posiadać wierzchołki oznaczone jako „nieodwiedzone". Odpowiedz na poniższe pytania:

(a) Czy istnieje w tym grafie wierzchołek, od którego moglibyśmy zacząć przeszukanie, aby wszystkie wierzchołki zostały odwiedzone?

(b) Czy zmiana wszystkich krawędzi tego grafu na nieskierowane sprawi, że wszystkie wierzchołki po przeszukaniu oznaczone będą jako „odwiedzone”?

Spróbuj narysować inne takie grafy, aby odpowiedzi na powyższe pytania były inne niż dla narysowanego pierwotnie grafu.

5. Wypisz wszystkie cykle znajdujące się w grafie na rysunku 1. Które z tych cykli nie znajdują się w grafie z rysunku 3?