ALGORYTMY, CO TO JEST ALGORYTM ?

Sebastian Majewski

14 grudnia 2014

Spis treści

1 Gdzie możemy je spotkać? 3

2 Definicja 3

3 Etapy rozwiązywania problemów 3

4 Działania niealgorytmiczne 4

5 Budowa algorytmów 4

6 Algorytmy z życia codziennego 4

7 Przykłady algorytmów liniowych 4
7.1 Przykład 1 4
7.2 Przykład 2 5

8 Przykłady algorytmów warunkowych 5
8.1 Przykład 1 6

9 Przykłady algorytmów iteracyjnych 6
9.1 Przykład 1 7
9.2 Przykład 2 7

10 Cechy algorytmów 8

11 Elementy schematów blokowych 9

12 Pole kwadratu 9

13 Czy liczba jest parzysta? 10

14 Pętle iteracyjne i warunkowe 10

15 Suma N Liczb 11

16 Średnia arytmetyczna N Liczb 11

17 Suma dowolnie długiego ciągu liczb 12

18 Wszystkie dzielniki liczby 13

19 Sortowanie bąbelkowe 14

20 Sortowanie szybkie 15

21 Sortowanie przez scalanie 16

22 Sortowanie przez wybieranie 17

23 Sortowanie przez wstawianie 18

1 Gdzie możemy je spotkać?

Na lekcjach matematyki czy fizyki często słyszymy zdanie "rozwiąż zadanie". Większość tych zadań rozwiązujemy według pewnych schematów. Najpierw wypisujemy dane i zastanawiamy się do jakiego celu dążymy, a więc jaki ma być wynik. Następnie wypisujemy wzory łączące dane z szukanymi bądź twierdzenia, które można zastosować. Przed koniecznością rozwiązywania zadań stajemy również poza szkołą. Robimy to w każdej innej dziedzinie naszego życia, na przykład:

1. telefonując do koleżanki i zapraszając ją na ciasto
2. gotując jajko na twardo
3. kupując piłkę w sklepie
4. znajdując najniższego ucznia w klasie
5. pisząc wypracowanie

Również przy takim typie zadań musimy określić dane i warunki, które muszą one spełniać. formułujemy także wynik, który pragniemy uzyskać.

2 Definicja

Algorytm to przepis rozwiązania zadania, zawierający opis danych wraz z opisem czynności, które należy wykonać z tymi danymi, aby osiągnąć zamierzony cel.

3 Etapy rozwiązywania problemów

W procesie rozwiązywania każdego zadania możemy wyróżnić pewne etapy, które nas do niego prowadzą:

1. Sformułowanie zadania
2. Określenie danych wejściowych
3. Określenie celu, czyli wyniku
4. Poszukiwanie metody rozwiązania, czyli algorytmu
5. Przedstawienie algorytmu w postaci:
    - opisu słownego
    - listy kroków
    - schematu blokowego
    - języka programowania
6. Analiza poprawności rozwiązania
7. Testowanie rozwiązania dla różnych danych. Ocena efektywności przyjętej metody.

4 Działania niealgorytmiczne

Czy wszystkie działania są algorytmiczne ? Czy dla każdego zadania można skonstruować algorytm? Czy rozwiązanie każdego zadania polega na wykonywaniu jednoznacznie opisanych, ściśle określonych czynności? Oczywiście, że nie. Istnieją zadania, których realizacji nie można ująć w ramy jakiegoś planu działania. Taki charakter ma np. każda twórczość artystyczna. Konieczna jest do tego wyobraźnia i twórcze działanie, a na to nie ma przepisu.

5 Budowa algorytmów

Algorytmy powininny być tak przedstawiane, aby było możliwe ich jednoznaczne odczytanie i zastosowanie. Niektóre algorytmy można opisać w języku potocznym, zwłaszcza wtedy, gdy jego wykonawcą ma być człowiek (w informatyce zajmujemy się opracowywaniem algorytmów, których wykonanie powierzamy komputerom). Z czego składa się algorytm? Zawiera on opis danych, opis wyników oraz plan działania, czyli przetworzenia danych. Plan ten można przedstawić w postaci ciągu czynności, które muszą być wykonane w określonej kolejności. Opis czynności występujących w algorytmie nazywamy instrukcjami.

6 Algorytmy z życia codziennego

Zastanów się jak ugotować jajko na miękko. Na początku opracowywania algorytmu przyjmijmy założenie, że używamy kuchenki gazowej, posiadamy garnek i wodę. Oczywiście niezbędne jest też samo jajko. Zakładamy również, że nic nie utrudni samej czynności, to znaczy np. w trakcie gotowania nie zostaniemy pozbawieni dopływu gazu, czy też osoba nie wie co to garnek.

7 Przykłady algorytmów liniowych

7.1 Przykład 1

Przykład algorytmu z życia codziennego - gotowanie jaja na miękko.

Algorytm ten ma postać:

1. Wlać do garnka zimną wodę.
2. Zapalić gaz.
3. Gotować wodę do wrzenia.
4. Włożyć jajko.
5. Odczekać trzy minuty.
6. Zgasić gaz.
7. Wyjąć jajko

7.2 Przykład 2

Opracuj algorytm telefonicznego zaproszenia koleżanki na ciasto. Numer telefonu : 666-22-88. W tym przypadku zrobimy założenie, że nic nie utrudni połączenia (a więc już za pierwszym razem zostaniemy połączeni z wybraną osobą), a wykonawca czynności zna cyfry i wie jak wybiera się je z tarczy telefonu. Oto jak wygląda algorytm (wersja 1):

1. Podnieś słuchawkę.
2. Wybierz cyfrę 6.
3. Wybierz cyfrę 1.
4. Wybierz cyfrę 6.
5. Wybierz cyfrę 2.
6. Wybierz cyfrę 2.
7. Wybierz cyfrę 2.
8. Wybierz cyfrę 2.
9. Przekaż informacje.
10. Odłóż słuchawkę.

Algorytmy liniowe mają opisy składające się z kroków, które nie zależą od żadnych warunków i są wykonywane w zapisanej kolejności. Istnieją jednak sytuacje, w których dalsze postępowanie w algorytmie zależy od spełnienia, bądź nie, określonych warunków. Czasami musimy powtórzyć pewne kroki algorytmu kilka razy.

8 Przykłady algorytmów warunkowych

W rzeczywistości, większość algorytmów ma bardziej rozbudowaną strukturę. Często występują w nich instrukcje, których wykonanie uzależnione jest od spełnienia pewnego warunku lub też spełnienie pewnego warunku powoduje wykonanie jednej instrukcji, a niespełnienie go -innej. Taką instrukcję nazywamy instrukcją warunkową. Działa ona według jednego z dwóch przedstawionych schematów:

- Jeśli spełniony jest warunek W, wykonaj instrukcję A.
- Jeśli spełniony jest warunek W, to wykonaj instrukcję A; w przeciwnym razie wykonaj instrukcję B.

Instrukcja A i B opisuje jedną instrukcję lub instrukcję składającą się z ciągu instrukcji wykonywanych sekwencyjnie. Instrukcja warunkowa pozwala dokonać wyboru jednej z dwóch dalszych dróg wykonania algorytmu.

8.1 Przykład 1

Opracowany wcześniej algorytm tak, nie uwzględniał sytuacji, gdy po wybraniu numeru jest on zajęty lub połączenie okazało się błędne. Kiedy słychać sygnał zajętości numeru, a więc nie udało się uzyskać połączenia trzeba Odłożyć słuchawkę. Tak samo postępujemy, gdy nawiązane zostało połączenie z innym Numerem. Aby zrealizować taką sytuację zastosujemy instrukcję warunkową. Zrobimy to po to, aby opisać czynności, powinny być wykonane wtedy kiedy zostało nawiązane poprawne połączenie, jak również nie zostało nawiązane. Zauważ, że wtedy wykonawca znajdzie się w punkcie wyjścia, czyli jakby w ogóle nie podjął próby telefonowania. Algorytm może mieć postać taką (wersja 2):

1. Podnieś słuchawkę.
2. Wybierz cyfrę 6.
3. Wybierz cyfrę 1.
4. Wybierz cyfrę 6.
5. Wybierz cyfrę 2.
6. Wybierz cyfrę 2.
7. Wybierz cyfrę 2.
8. Wybierz cyfrę 2.
9. Czy połączyłeś się z koleżanką ?
10. Jeśli TAK, to przejdź do kroku 10.
11. Jeśli NIE, to przejdź do kroku 11.
12. Zaproś koleżankę.
13. Odłóż słuchawkę.

9 Przykłady algorytmów iteracyjnych

Instrukcja iteracyjna - ze znaną ilością powtórzeń Przyjrzyj się uważnie algorytmowi. Zauważyłeś, że istnieją tu powtarzające się instrukcje, aż czterokrotnie występuje "Wybierz cyfrę 2". Takie wielokrotne powtarzanie niektórych instrukcji jest cechą charakterystyczną wielu algorytmów, jednak nie zawsze (tak jak w tym algorytmie) możemy określić dokładnie liczbę powtórzeń. Może ona zależeć od spełnienia pewnych warunków. Wielokrotne powtarzanie instrukcji umożliwiają instrukcje iteracyjne (pętle) . Działa ona według schematu: Wykonuj instrukcję A dokładnie n razy.

9.1 Przykład 1

Popraw opracowany wcześniej algorytm tak, aby sekwencję jednakowych czynności zastąpić pętlą. Oto algorytm (wersja 3):

1. Podnieś słuchawkę.
2. Wybierz cyfrę 6.
3. Wybierz cyfrę 1.
4. Wybierz cyfrę 6. Wykonaj czynność cztery razy
5. Wybierz cyfrę 2. Czy połączyłeś się z koleżanką ?
6. Jeśli tak, to przejdź do kroku 7.
7. Jeśli nie, to przejdź do kroku 8.
8. Zaproś koleżankę.
9. Odłóż słuchawkę.

Działanie algorytmu może zakończyć się na dwa różne sposoby, albo uzyskamy połączenie z koleżanką i zaprosimy ją na ciasto, albo nie połączymy się i wykonanie algorytmu ograniczy się tylko do wykręcenia numeru. Jak widzisz zastosowanie pętli zmienia sposób zapisu algorytmu, nie zmienia się jednak jego działanie.

9.2 Przykład 2

Uwzględnij w opracowanym wcześniej algorytmie przypadek braku połączenia lub nawiązanie nieprawidłowego połączenia. W poprzednim algorytmie w przypadku uzyskania nieprawidłowego połączenia bądź jego braku, przechodziliśmy do ostatniego kroku , w którym odkładaliśmy słuchawkę. Kończyło się Działanie algorytmu. W tej sytuacji powinniśmy rozpocząć raz jeszcze jego wykonywanie, nie zostało to jednak opisane w konstrukcji. Rozbudujemy teraz algorytm, tak by powtarzano wybieranie numeru aż do uzyskania połączenia. Dopiszemy w tym celu polecenie będące drugim rodzajem instrukcji iteracyjnej: Powtarzaj wykonywanie instrukcji A aż do spełnienia warunku W. W naszym algorytmie sprawdzamy warunek uzyskania połączenia z koleżanką (krok 4). Czasami zdarza się jednak, że już po podniesieniu słuchawki słychać, że linia jest zajęta. Należałoby wtedy odłożyć słuchawkę i ponownie ją podnieść. Jeśli okazałoby się, że nadal słychać w niej sygnał zajętości linii czynność należałoby powtórzyć. Musielibyśmy wykonywać te czynności dopóki linia nie byłaby "czysta".

Algorytm (w wersji 5) wygląda tak :

1. Czy słuchawka jest odłożona ?
   - Jeśli tak, to przejdź do kroku 2.
   - Jeśli nie, to odłóż słuchawkę.
2. Podnieś słuchawkę.
3. Czy linia jest zajęta ?
   - Jeśli Tak, to:
   - Odłóż słuchawkę.
   - Podnieś słuchawkę.
   - Przejdź do kroku 3.
4. Jeśli Nie, to przejdź do kroku 4.
5. Wybierz cyfrę 6.
6. Wybierz cyfrę 1.
7. Wybierz cyfrę 6.
8. Wykonaj czynność cztery razy Wybierz cyfrę 2.
9. Czy połączyłeś się z koleżanką ?
   - Jeśli tak, to przejdź do kroku 9.
   - Jeśli nie, to przejdź do kroku 1.
10. Zaproś koleżankę.
11. Odłóż słuchawkę.

W kroku 3 występuje pętla, w której sprawdzamy najpierw warunek, a dopiero potem wykonywana jest instrukcja opisana przez kroki a, b. Jeśli warunek nie jest spełniony, to instrukcja nie zostanie wykonana ani razu.

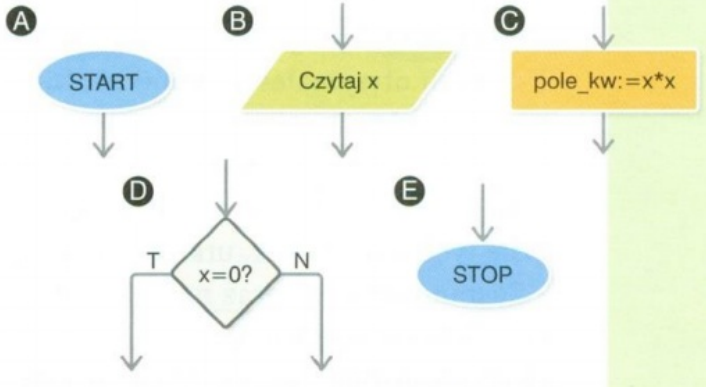
10 Cechy algorytmów

1. Poprawność (algorytm daje dobre wyniki, odzwierciedlające rzeczywistość)
2. Jednoznaczność (brak rozbieżności wyników przy takich samych danych, jednoznaczne opisanie każdego kroku)
3. Skończoność (wykonuje się w skończonej ilości kroków)
4. Sprawność (czasowa - szybkość działania i pamięciowa - "zasobożerność")
5. Prostota wykonania (operacje powinny być jak najprostsze)

Aby nasz algorytm rozwiązywał poprawnie pewne zagadnienia niezbędne są dane początkowe i nadanie ograniczeń np. warunki brzegowe, jednym słowem należy przedstawić rzeczywistość poprzez:

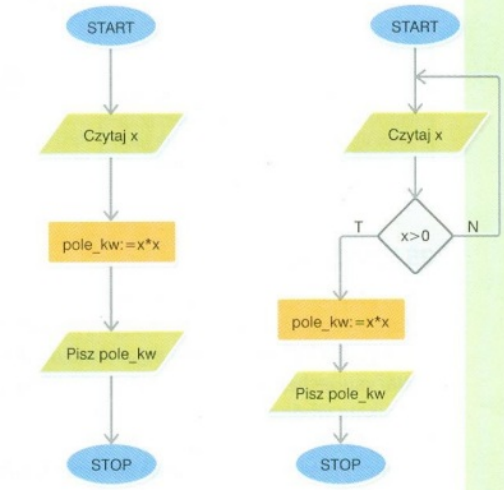
- zdefiniowanie zadania
- wprowadzenie założeń i ograniczeń
- algorytm rozwiązania

11 Elementy schematów blokowych



Ryc. 1. Elementy schematu blokowego: A) początek algorytmu, B) blok wejścia/wyjścia, C) blok obliczeniowy, D) blok warunkowy, E) zakończenie algorytmu

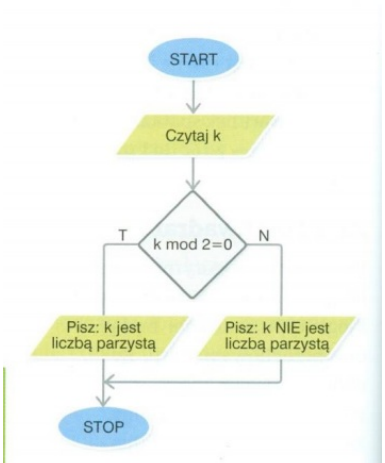
12 Pole kwadratu



Ryc. 2. Schemat blokowy do obliczania pola kwadratu

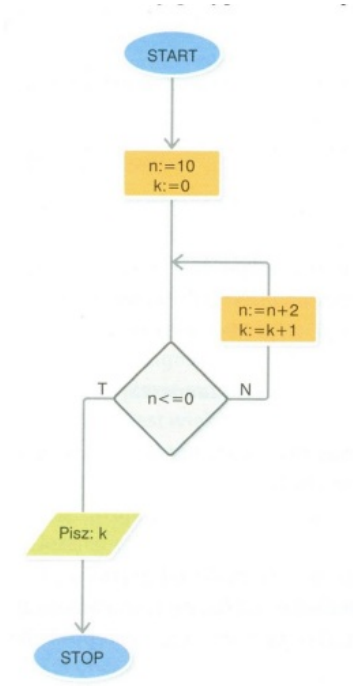
Ryc. 3. Schemat blokowy do obliczania pola kwadratu uniemożliwiający wprowadzenie wartości niedodatnich

13 Czy liczba jest parzysta?



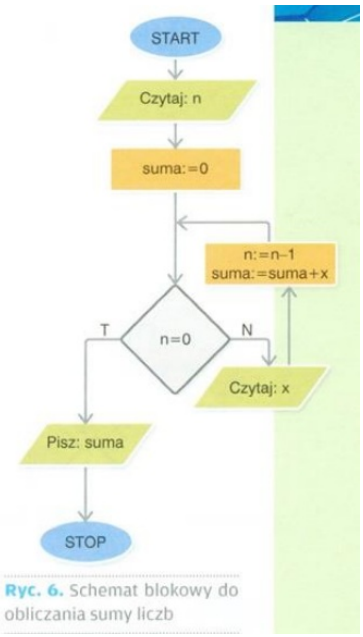
Ryc. 4. Schemat blokowy sprawdzający, czy dana liczba jest parzysta

14 Pętle iteracyjne i warunkowe



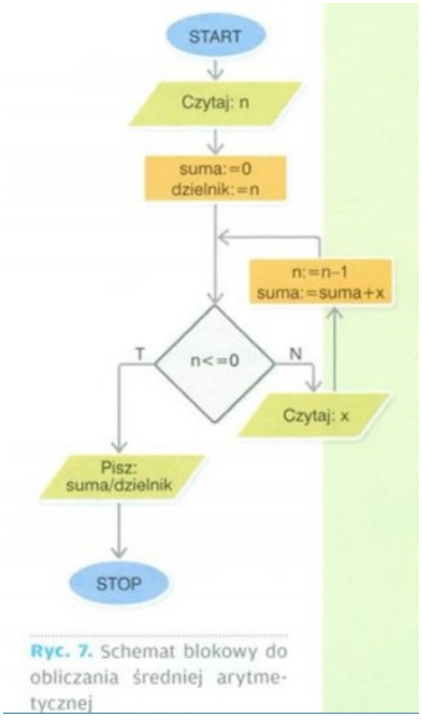
Ryc. 5. Przykład pętli nieskończonej

15 Suma N Liczb

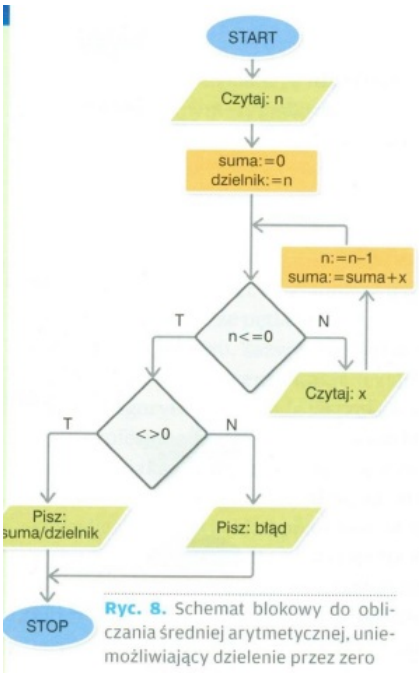


Ryc. 6. Schemat blokowy do obliczania sumy liczb

16 Średnia arytmetyczna N Liczb

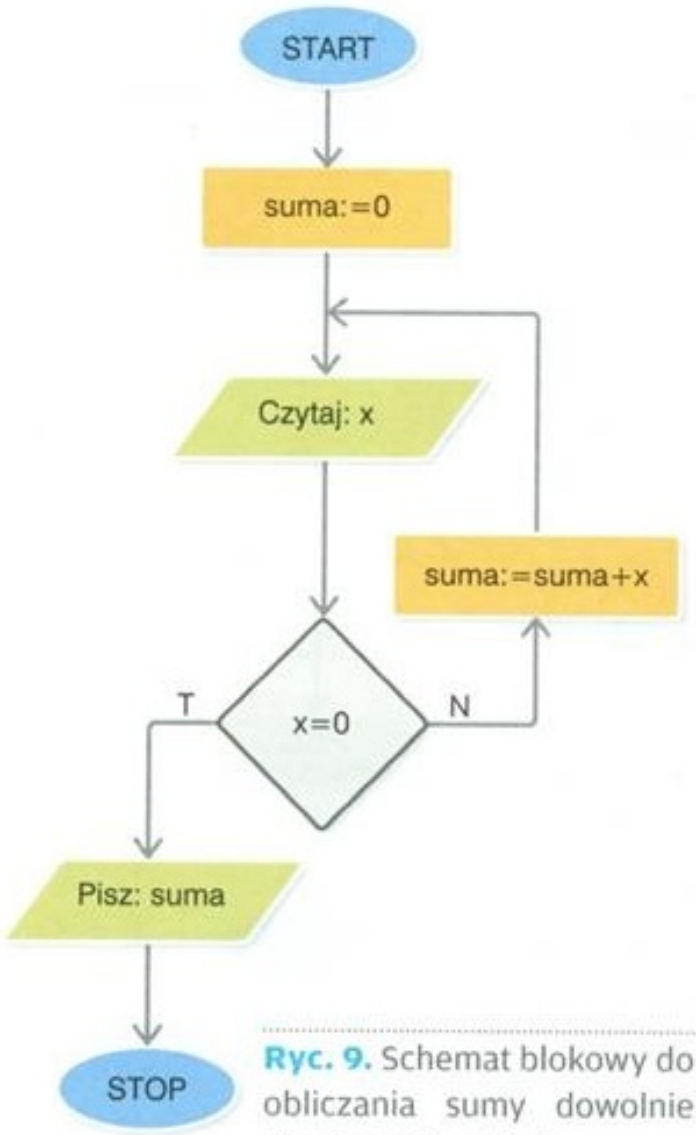


Ryc. 7. Schemat blokowy do obliczania średniej arytmetycznej



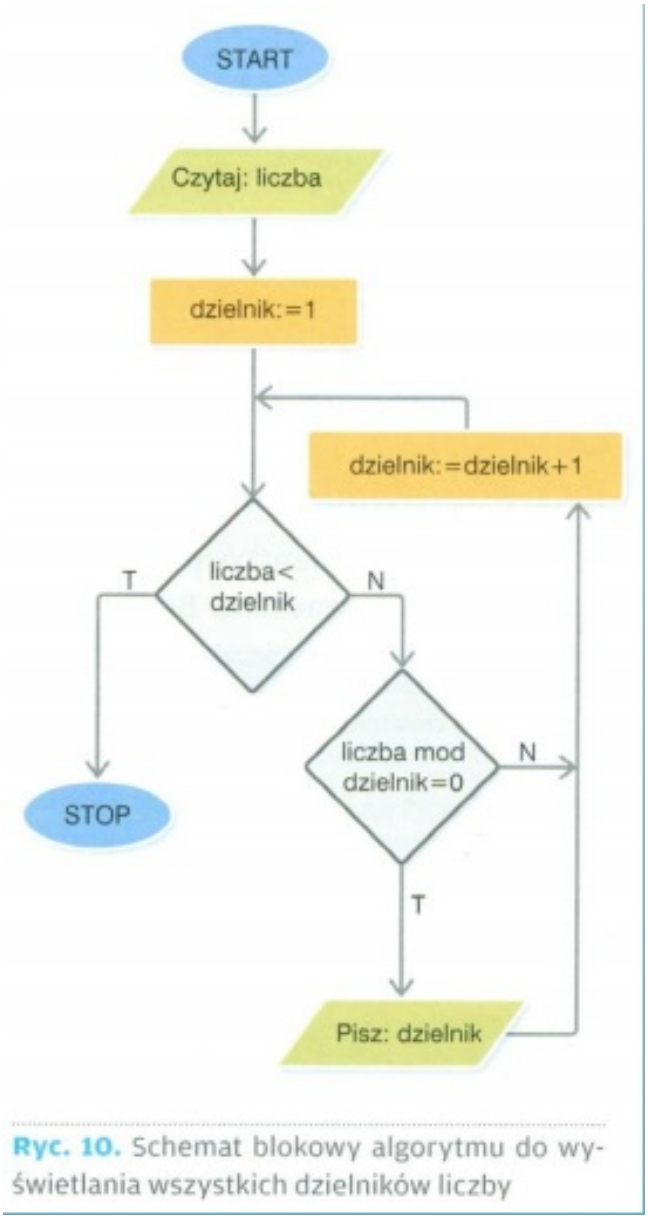
Ryc. 8. Schemat blokowy do obliczania średniej arytmetycznej, uniemożliwiający dzielenie przez zero

17 Suma dowolnie długiego ciągu liczb



Ryc. 9. Schemat blokowy do obliczania sumy dowolnie długiego ciągu liczb

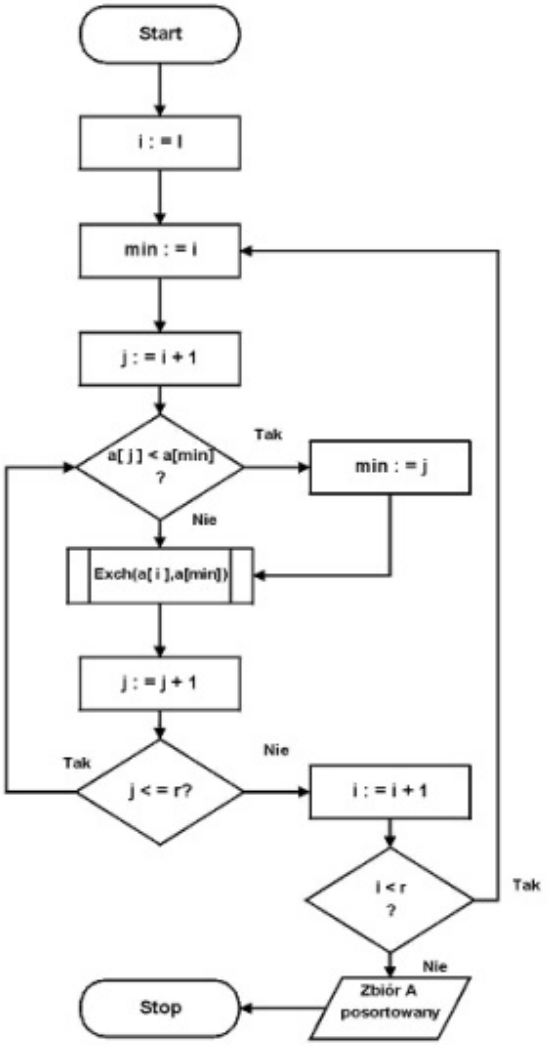
18 Wszystkie dzielniki liczby



Ryc. 10. Schemat blokowy algorytmu do wyświetlania wszystkich dzielników liczby

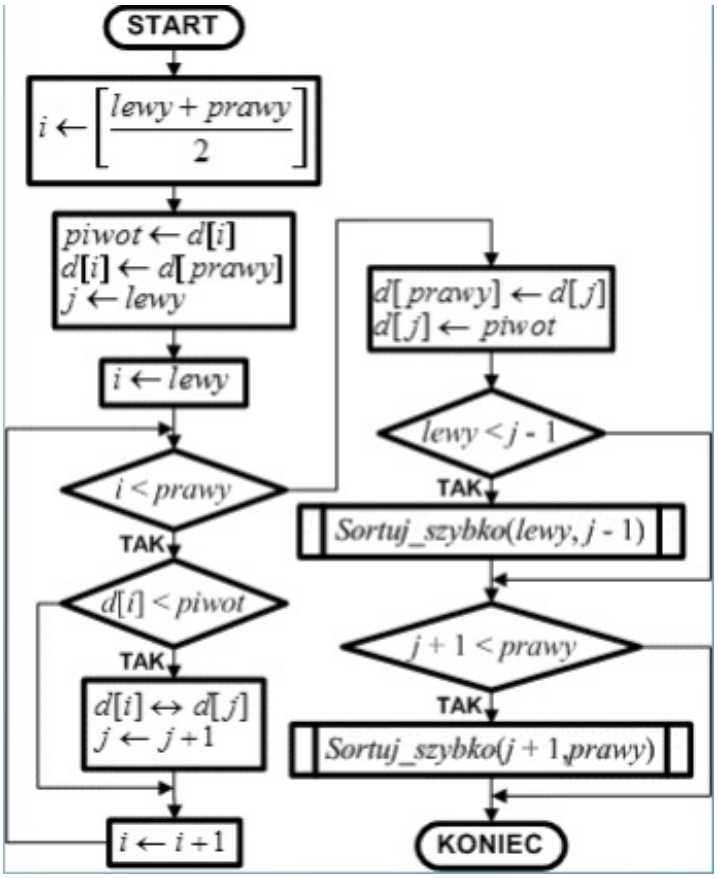
19 Sortowanie bąbelkowe

Sortowanie bąbelkowe jest jednym z prostszych w implementacji algorytmów sortowania. Swoją nazwę zawdzięcza temu, że w przypadku pionowego przedstawienia zbioru danych, element najmniejszy(o najmniejszej masie) wypływa do góry. Działanie tego algorytmu opiera się na porównywaniu każdego elementu z elementem następującym po nim i w przypadku stwierdzenia nieprawidłowej relacji pomiędzy tymi elementami następuje zamiana ich kolejności. Wynika to z założenia, że nieposortowany ciąg zawiera, co najmniej dwa elementy znajdujące się na nieodpowiednich miejscach. Kolejnym krokiem jest sprawdzenie, czy poczyniona przez algorytm zmiana kolejności dwóch elementów nie wpłynęła na prawidłowość relacji pozostałych elementów. Jeżeli zaburzyła tą prawidłowość, zbiór danych jest ponownie przeszukiwany w tym samym kierunku. Algorytm kończy swoje działanie w przypadku stwierdzenia, że wszystkie elementy znajdują się w prawidłowej relacji, czyli że nie została wykonana jakakolwiek zamiana kolejności elementów. Pesymistyczny koszt takiego algorytmu wynosi n^2, gdzie n oznacza ilość elementów do posortowania. Algorytm ten jest bardzo wydajny, jeżeli używamy go do zbioru danych o bardzo małej ilości elementów, lub też zbiór danych jest prawie posortowany (wymaga bardzo małej ilości zmian). W przypadku sortowania tablicy posortowanej malejąco, algorytm wykona n2 kroków, jest to wariant najbardziej pesymistyczny.



20 Sortowanie szybkie

Jest to kolejny z algorytmów sortowania opierający się na zasadzie "dziel i zwyciężaj". W algorytmie tym wprowadzono pewną dozę prawdopodobieństwa, mianowicie. Na początku wybieramy pewien element tablicy, tzw. element dzielący, po czym na początek tablicy przenosimy wszystkie elementy mniejsze od niego, na koniec wszystkie większe, a w powstałe między tymi obszarami puste miejsce trafia wybrany element. Potem sortujemy osobno początkową i końcową część tablicy. Rekursja kończy się, gdy kolejny fragment uzyskany z podziału zawiera pojedynczy element, jako że jednoelementowa podtablica jest oczywiście posortowana. Spójrzmy na jedna rzecz, mianowicie na wagę wyboru elementu dzielącego, gdy będziemy wybierać zawsze skrajny przypadek, możemy spowodować, że posortowanie już posortowanej tablicy będzie zadaniem trudnym. Tak więc odrzucimy taki sposób wyboru elementu dzielącego. Drugi sposób wykorzystuję już wspomniany przez nas element prawdopodobieństwa. Sposób ten opiera się na losowaniu elementu dzielącego, jako jednego z trzech elementów środkowych co w sposób zasadniczy sprowadza prawdopodobieństwo zajścia najgorszego przypadku do wartości zaniedbywalnie małych. Kolejny sposób polega na wstępnym wyborze trzech elementów z rozpatrywanego fragmentu tablicy, i użyciu jako elementu dzielącego tego z trzech, którego wartość leży pomiędzy wartościami pozostałych dwu. Można również uzupełnić algorytm o poszukiwanie przybliżonej mediany.

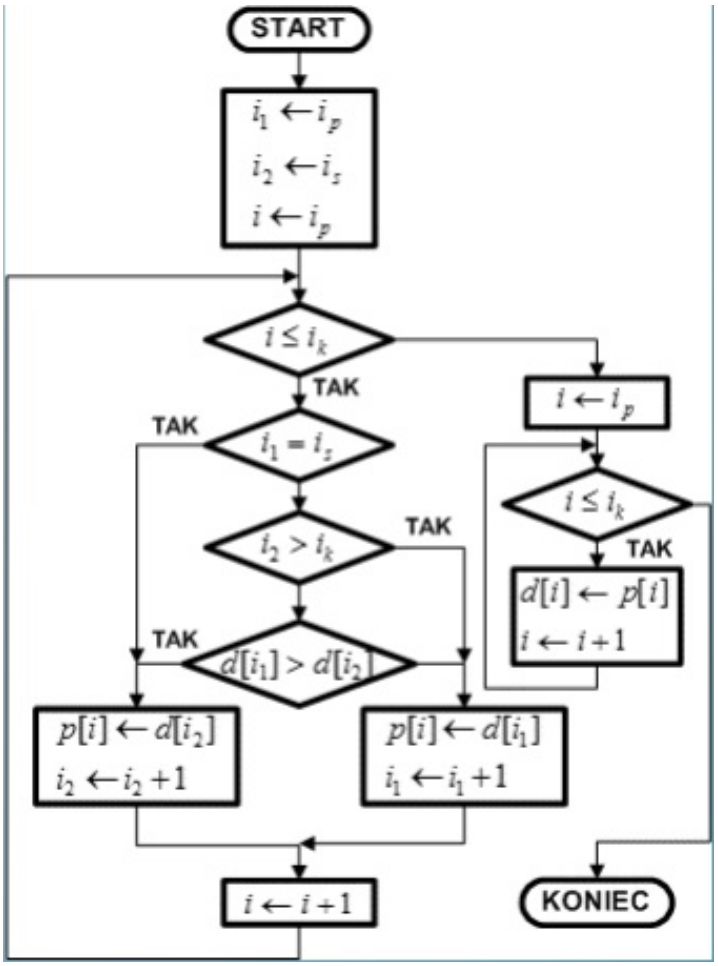


21 Sortowanie przez scalanie

W algorytmie sortowania przez scalanie jest wykorzystywana strategia "dziel i zwyciężaj". Strategia polega na podzieleniu większego problemu na mniejsze takie same problemy a następnie rozwiązywaniu mniejszych i scalaniu wyników poszczególnych problemów.

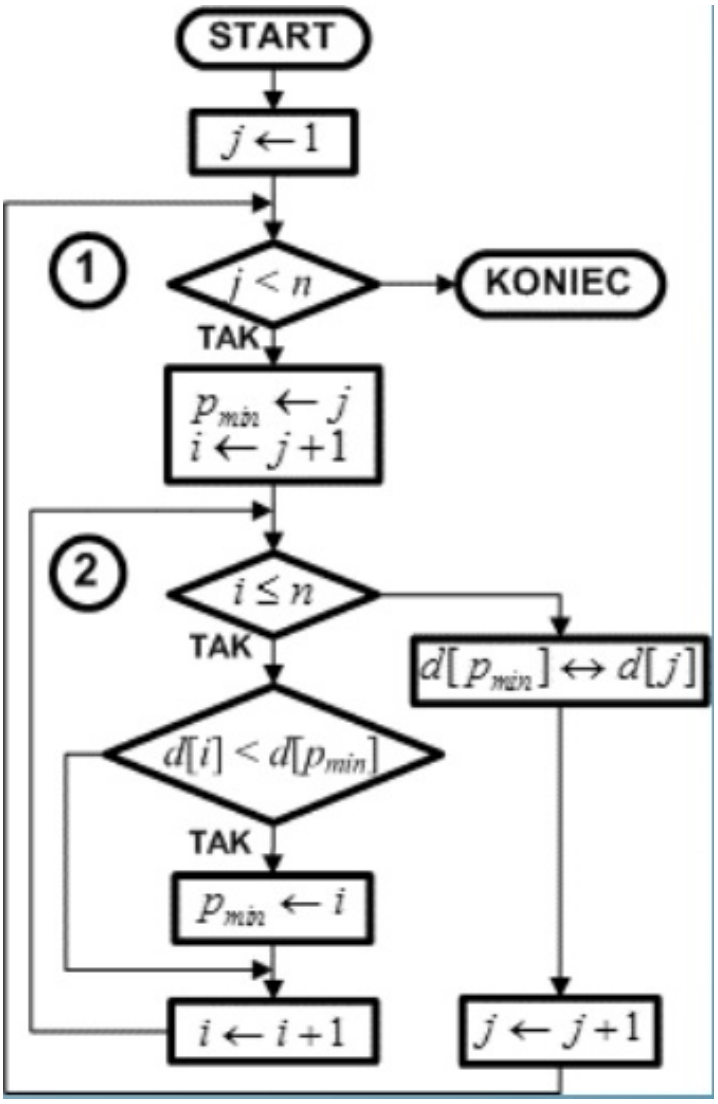
Algorytm sortowania przez scalanie przedstawia się następująco:

1. Jeśli ciąg zawiera więcej niż jeden element, to podziel go na dwie równe części (lub prawie równe,
2. jeśli ciąg ma nieparzystą liczbę elementów) posortuj pierwszą część stosując ten sam algorytm
3. Posortuj drugą część stosując ten sam algorytm
4. Połącz dwa uporządkowane ciągi w jeden ciąg uporządkowany



22 Sortowanie przez wybieranie

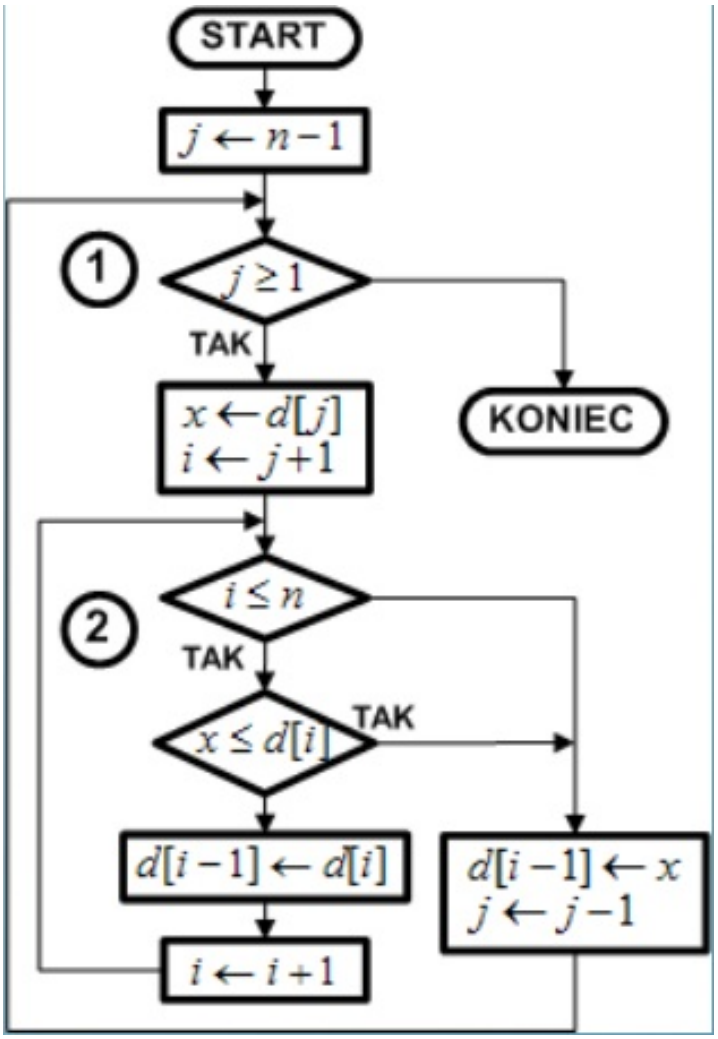
Sortowanie przez wybieranie jest już bardziej wydajnym algorytmem sortowania, A niżeli sortowanie bąbelkowe. Idea jego działania polega na wybieraniu z podzbioru danych zbioru elementu najmniejszego (znajdowania minimum) i zamianie jego położenia z początkowym elementem podzbioru. Następnie zakres poszukiwania zostaje zawężony do podzbioru danych znajdujących się po posortowanych już elementach (co krok zbiór będzie mniejszy o 1). W pierwszym przeszukiwaniu tym podzbiorem jest naturalnie cały zbiór. Koszt tego algorytmu jest zauważalnie mniejszy od algorytmów bąbelkowego i sortowania przez wstawianie. Wynosi on w pesymistycznym wypadku $\frac{n^2-n}{2}$, gdzie n oznacza ilość elementów do posortowania. Zaletami algorytmu sortowania przez wybieranie jest optymalna ilość przestawień (n-1); prostota implementacji oraz zadowalająca szybkość dla małych wartości n. Algorytm przedstawia się następująco:



1. wyszukaj minimalną wartość z tablicy spośród elementów od i+1 do końca tablicy
2. zamień wartość minimalną, z elementem na pozycji i
3. przesuń początek zbioru o 1 w przód

23 Sortowanie przez wstawianie

Algorytm ten jest przydatny przy sortowaniu danych sukcesywnie napływających. Podobnie jak w przypadku algorytmów bąbelkowego i sortowania przez wybieranie pesymistyczny koszt działania tego algorytmu wynosi $\frac{n^2-n}{2}$, gdzie n oznacza ilość elementów do posortowania. Schemat działania algorytmu:



1. Utwórz zbiór elementów posortowanych i przenieś do niego dowolny element ze zbioru nieposortowanego.
2. Weź dowolny element ze zbioru nieposortowanego.
3. Wyciągnięty element porównuj z kolejnymi elementami zbioru posortowanego póki nie napotkasz elementu równego lub elementu większego (jeśli chcemy otrzymać ciąg niemalejący) lub nie znajdziemy się na początku/końcu zbioru uporządkowanego.
4. Wyciągnięty element wstaw w miejsce gdzie skończyłeś porównywać.
5. Jeśli zbiór elementów nieuporządkowanych jest niepusty wróć do punkt 2.