2.5. Przeszukiwanie ciągu liczbowego — metody liniowe

2.5.1. Liniowe przeszukiwanie ciągu liczbowego

Analizowany problem polega na znalezieniu w nieuporządkowanym ciągu liczbowym wyrazu spełniającego określone warunki. **Przeszukiwanie liniowe**, którym zajmiemy się w tym rozdziale, to przeszukiwanie tablicy element po elemencie. W metodzie tej należy uwzględnić zarówno sprawdzanie, czy nie wyszliśmy poza zakres tablicy, jak i szukanie elementu spełniającego określone warunki. W punkcie 1.7.1, „Przeszukiwanie binarne ciągu uporządkowanego", przedstawiono również algorytm przeszukiwania uporządkowanego ciągu liczbowego, który wykorzystuje technikę „dzieli i zwyciężaj" — **przeszukiwanie binarne**.

Skonstruujmy algorytm szukający określonego elementu w tablicy zawierającej liczby całkowite, wykorzystujący **liniowe przeszukiwanie ciągu liczbowego**, w postaci schematu blokowego (patrz rysunek 2.10) oraz programów w językach C++ i Pascal. Dodatkowo na płycie CD znajduje się realizacja tego algorytmu wykonana za pomocą arkusza kalkulacyjnego (*arkusz2_6.xls*, *arkusz2_6.ods*).

Specyfikacja:

Dane: Liczba naturalna: $n > 0$ (liczba elementów tablicy T).

Liczba całkowita: *szukana* (wartość elementu szukanego w tablicy T).

n-elementowa tablica jednowymiarowa zawierająca liczby całkowite: $T[0 \ldots n-1]$.

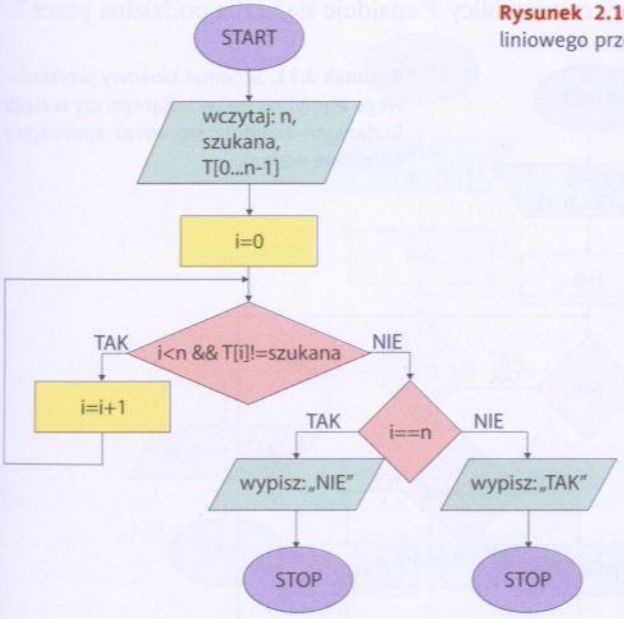
Wynik: Komunikat informujący, czy szukana liczba znajduje się w tablicy T.

Funkcja w języku C++ (prog2_13.cpp):

```
bool szukaj (int T[], int n, int szukana)
{
 int i=0;
 while (i<n && T[i]!=szukana) i++;
 if (i==n) return false;
 return true;
}
```

***Funkcja w języku Pascal** (prog2_13.pas):*

```
function szukaj (T: tablica; n: integer; szukana: integer): boolean;
var i: integer;
begin
 i:=0;
 while (i<n)and(T[i]<>szukana) do i:=i+1;
 if i=n then szukaj:=false else szukaj:=true
end;
```



Rysunek 2.10. Schemat blokowy algorytmu liniowego przeszukiwania ciągu liczbowego

Operacją dominującą w tym algorytmie jest porównanie. W najgorszym przypadku, gdy szukany element nie znajduje się w tablicy, wykonywanych jest n takich działań podczas przeglądania ciągu i jedno przy sprawdzeniu, czy element został znaleziony. W sytuacji gdy pierwszy sprawdzany wyraz w ciągu będzie równy szukanemu, wykonamy tylko dwa porównania. Liczba operacji dominujących może więc wynosić od 2 do n+1, dlatego złożoność czasowa tego algorytmu jest liniowa $O(n)$.

Do omawianego problemu możemy podejść również w inny sposób. Różnica polega na wprowadzeniu warunku wewnętrznego sprawdzającego kolejne elementy tablicy. Wyróżniamy tutaj dwa podejścia:

- sprawdzenie, czy w tablicy istnieje element spełniający określone warunki;
- sprawdzenie, czy wszystkie elementy tablicy spełniają określone warunki.

W obu przypadkach chcemy uzyskać odpowiedź TAK lub NIE. Konstruujemy więc funkcje typu logicznego, których wartość będzie równa **`true`**, gdy warunki będą spełnione, lub **`false`** — w przeciwnym wypadku.

Przykład 2.29.

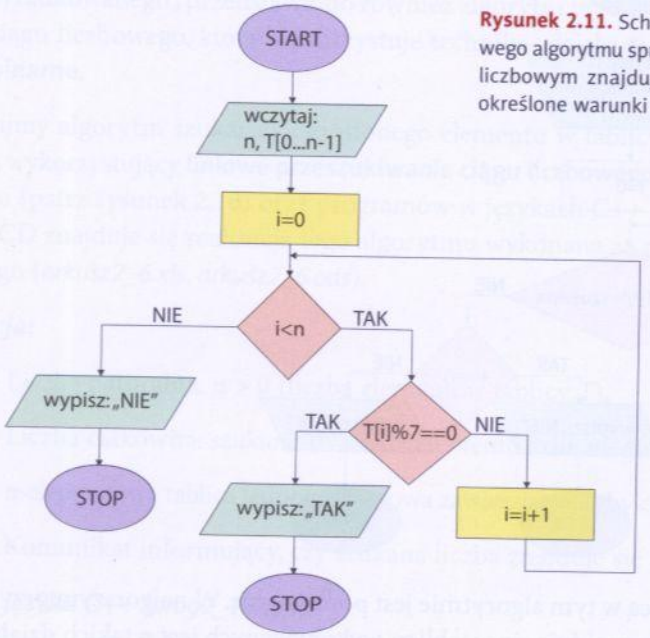
Skonstruujmy algorytm w postaci schematu blokowego (patrz rysunek 2.11) i programu w języku C++ sprawdzający, czy we wczytanym ciągu liczb całkowitych znajduje się liczba podzielna przez 7.

Specyfikacja:

Dane: Liczba naturalna: $n > 0$ (liczba elementów tablicy T).

n-elementowa tablica jednowymiarowa zawierająca liczby całkowite: $T[0 \ldots n-1]$.

Wynik: Komunikat informujący, czy w tablicy T znajduje się liczba podzielna przez 7.



Rysunek 2.11. Schemat blokowy przykładowego algorytmu sprawdzającego, czy w ciągu liczbowym znajduje się wyraz spełniający określone warunki

Funkcja w języku C++ (prog2_14.cpp):

```
bool szukaj (int T[], int n)
{
 for (int i=0;i<n;i++)
  if (T[i]%7==0) return true;
 return false;
}
```

Przykład 2.30.

Skonstruujmy algorytm w postaci schematu blokowego (patrz rysunek 2.12) i programu w języku C++ sprawdzający, czy wszystkie wyrazy wczytanego ciągu liczb rzeczywistych

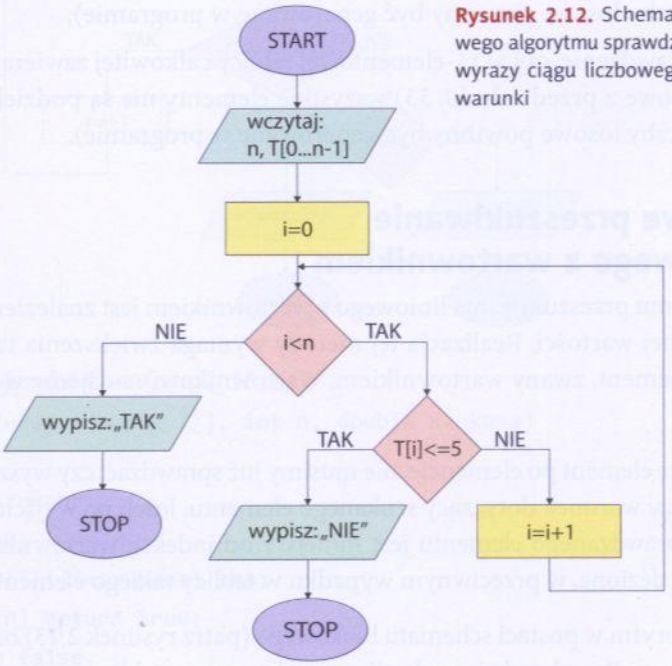
są większe od 5. Rozwiązanie problemu polega tutaj na znalezieniu jednego takiego elementu w tablicy, który nie spełnia podanych warunków.

Specyfikacja:

Dane: Liczba naturalna: $n > 0$ (liczba elementów tablicy T).

n-elementowa tablica jednowymiarowa zawierająca liczby rzeczywiste: $T[0...n-1]$.

Wynik: Komunikat informujący, czy wszystkie elementy tablicy T są większe od 5.



Rysunek 2.12. Schemat blokowy przykładowego algorytmu sprawdzającego, czy wszystkie wyrazy ciągu liczbowego spełniają określone warunki

Funkcja w języku C++ (prog2_15.cpp):

```
bool szukaj (double T[], int n)
{
 for (int i=0;i<n;i++)
  if (T[i]<=5) return false;
 return true;
}
```

Zadanie 2.37. Podaj specyfikacje i skonstruuj algorytmy w postaci programów realizujące następujące operacje:

a) sprawdzenie, czy w n-elementowej tablicy zawierającej wartości rzeczywiste wprowadzone z klawiatury znajduje się liczba różna od 2;

b) sprawdzenie, czy w 10-elementowej tablicy całkowitej zawierającej liczby losowe z przedziału (5, 41] wszystkie elementy są podzielne przez 3 (liczby losowe powinny być generowane w programie);

c) sprawdzenie, czy w 15-elementowej tablicy zawierającej liczby całkowite wprowadzone z klawiatury wszystkie elementy są nie większe od 11;

d) sprawdzenie, czy w n-elementowej tablicy zawierającej wartości rzeczywiste wprowadzone z klawiatury znajduje się liczba nie mniejsza od 20;

e) sprawdzenie, czy w 9-elementowej tablicy rzeczywistej zawierającej liczby losowe z przedziału [–4, 28] wszystkie elementy są nie mniejsze od 9 (liczby losowe powinny być generowane w programie);

f) sprawdzenie, czy w 11-elementowej tablicy całkowitej zawierającej liczby losowe z przedziału [0, 33) wszystkie elementy nie są podzielne przez 4 (liczby losowe powinny być generowane w programie).

2.5.2. Liniowe przeszukiwanie ciągu liczbowego z wartownikiem

Zadaniem algorytmu przeszukiwania liniowego z wartownikiem jest znalezienie w ciągu liczbowym szukanej wartości. Realizacja tej metody wymaga zwiększenia tablicy o jeden dodatkowy element, zwany wartownikiem. Wartownikowi nadajemy wartość szukanego elementu.

Przeglądając tablicę element po elemencie, nie musimy już sprawdzać, czy wyszliśmy poza jej zakres. Wystarczy warunek dotyczący szukanego elementu. Jeżeli po wyjściu z pętli indeks ostatniego sprawdzanego elementu jest mniejszy od indeksu wartownika, szukana wartość została znaleziona, w przeciwnym wypadku w tablicy takiego elementu nie ma.

Skonstruujmy algorytm w postaci schematu blokowego (patrz rysunek 2.13) oraz programów w językach C++ i Pascal szukający określonego elementu w tablicy zawierającej liczby rzeczywiste, stosując **liniowe przeszukiwanie ciągu liczbowego z wartownikiem**.

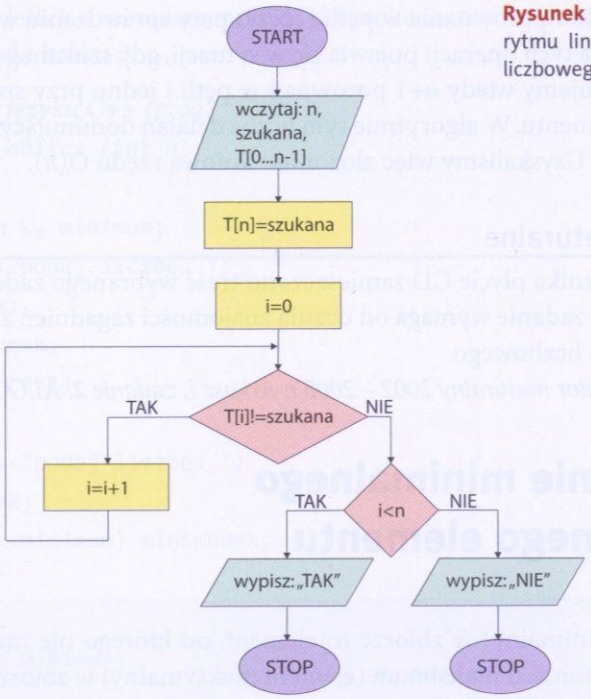
Specyfikacja:

Dane: Liczba naturalna: $n > 0$ (liczba elementów tablicy T).

Liczba rzeczywista: *szukana* (wartość elementu szukanego w tablicy T).

n-elementowa tablica jednowymiarowa zawierająca liczby rzeczywiste: $T[0...n-1]$.

Wynik: Komunikat informujący, czy w tablicy znajduje się szukany element.



Rysunek 2.13. Schemat blokowy algorytmu liniowego przeszukiwania ciągu liczbowego z wartownikiem

Funkcja w języku C++ (prog2_16.cpp):

```
bool szukaj (double T[], int n, double szukana)
{
 T[n]=szukana;
 int i=0;
 while (T[i]!=szukana) i++;
 if (i<n) return true;
 return false;
}
```

Funkcja w języku Pascal (prog2_16.pas):

```
function szukaj (T: tablica; n: integer; szukana: real): boolean;
var i: integer;
begin
 T[n]:=szukana;
 i:=0;
 while T[i]<>szukana do i:=i+1;
 if i<n then szukaj:=true else szukaj:=false
end;
```

Operacją dominującą w tym algorytmie jest porównanie. Minimalna liczba tych działań jest wykonywana, gdy pierwszy sprawdzany wyraz ciągu jest równy szukanej liczbie.

Realizujemy wówczas jedno porównanie w pętli i jedno przy sprawdzaniu wartości indeksu. Maksymalna liczba tych operacji pojawia się w sytuacji, gdy szukanego elementu nie ma w tablicy. Wykonujemy wtedy $n+1$ porównań w pętli i jedno przy sprawdzaniu indeksu znalezionego elementu. W algorytmie tym liczba działań dominujących zawiera się w przedziale $[2, n+2]$. Uzyskaliśmy więc złożoność liniową rzędu $O(n)$.

Wybrane zadania maturalne

Na załączonej do podręcznika płycie CD zamieszczono treść wybranego zadania maturalnego. Zaproponowane zadanie wymaga od ucznia znajomości zagadnień związanych z przeszukiwaniem ciągu liczbowego.

- *matura2.12.pdf* (*Informator maturalny 2002 – 2005 r. Arkusz I, zadanie 2. ALGORYTM*).

2.6. Znajdowanie minimalnego lub maksymalnego elementu

Definicja

Minimum (element minimalny) w zbiorze to element, od którego nie ma w tym zbiorze mniejszego, natomiast **maksimum** (element maksymalny) w zbiorze to element, od którego nie ma w tym zbiorze większego.

Zwykle przy wyznaczaniu elementu maksymalnego lub minimalnego zbiór danych jest określony. Zdarza się jednak, że musimy określić minimum lub maksimum dla liczb wczytywanych na bieżąco.

Skonstruujmy **algorytm wyznaczający element minimalny spośród n liczb rzeczywistych wprowadzanych na bieżąco z klawiatury**.

Specyfikacja:

Dane: Liczba naturalna: $n > 0$ (liczba wczytywanych wartości).

Wynik: Minimum dla liczb rzeczywistych wprowadzonych z klawiatury: *minimum*.

Lista kroków:

Krok 0. Wczytaj n.

Krok 1. Wczytaj pierwszą liczbę k.

Krok 2. Przypisz *minimum* = k.

Krok 3. Jeśli $n = 1$, wypisz *minimum* i zakończ algorytm.

Krok 4. Wczytaj liczbę k.

Krok 5. Zmniejsz n o 1.

Krok 6. Jeśli $k < minimum$, przejdź do kroku 2., w przeciwnym wypadku przejdź do kroku 3.

Funkcja w języku C++ (prog2_17.cpp):

```
double oblicz (int n)
{
 double k, minimum;
 cout<<"podaj liczbę: ";
 cin>>k;
 minimum=k;
 while (n>1)
 {
  cout<<"podaj liczbę: ";
  cin>>k;
  if (k<minimum) minimum=k;
  n--;
 }
 return minimum;
}
```

Funkcja w języku Pascal (prog2_17.pas):

```
function oblicz (n: integer): real;
var k, minimum: real;
begin
 write('podaj liczbę: ');
 readln(k);
 minimum:=k;
 while n>1 do
 begin
  write('podaj liczbę: ');
  readln(k);
  if k<minimum then minimum:=k;
  n:=n-1
 end;
 oblicz:=minimum
end;
```

Zadanie 2.38. Podaj specyfikację zadania i skonstruuj algorytm w postaci schematu blokowego wyznaczający maksimum dla liczb całkowitych wprowadzanych na bieżąco z klawiatury. Wpisywanie liczb powinno się zakończyć po podaniu wartości zero. Zwróć uwagę na to, że nie znamy liczby wczytywanych wartości.

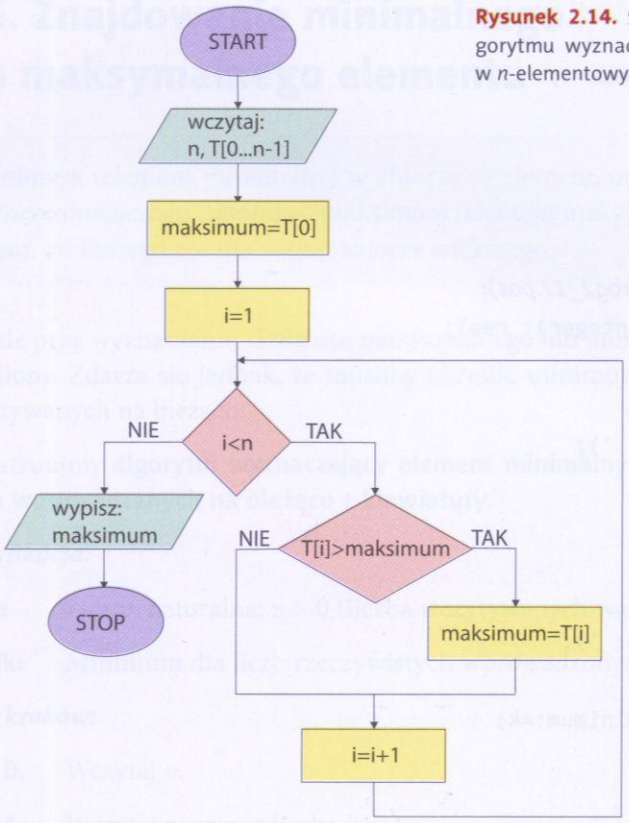
Przeanalizujmy teraz sytuację, w której zbiór liczb jest określony przed rozpoczęciem szukania elementu minimalnego lub maksymalnego. Skonstruujmy algorytm w postaci schematu blokowego (patrz rysunek 2.14) oraz programów w językach C++ i Pascal **wyznaczający maksimum dla *n* liczb całkowitych zapisanych w tablicy jednowymiarowej *T***. Dodatkowo na płycie CD znajduje się realizacja tego algorytmu wykonana za pomocą arkusza kalkulacyjnego (*arkusz2_7.xls*, *arkusz2_7.ods*).

Specyfikacja:

Dane: Liczba naturalna: $n > 0$ (liczba elementów tablicy T).

n-elementowa tablica zawierająca liczby całkowite: $T[0 \ldots n-1]$.

Wynik: Maksymalny element w tablicy T: *maksimum*.



Rysunek 2.14. Schemat blokowy algorytmu wyznaczającego maksimum w n-elementowym zbiorze liczb

Funkcja w języku C++ (prog2_18.cpp):

```
int oblicz (int T[], int n)
{
 int maksimum=T[0];
 for (int i=1;i<n;i++)
```

```
  if (T[i]>maksimum) maksimum=T[i];
 return maksimum;
}
```

Funkcja w języku Pascal (prog2_18.pas):

```
function oblicz (T: tablica; n: integer): integer;
var maksimum, i: integer;
begin
 maksimum:=T[0];
 for i:=1 to n-1 do
  if T[i]>maksimum then maksimum:=T[i];
 oblicz:=maksimum
end;
```

Przeanalizujmy złożoność przedstawionych algorytmów. W obydwu przypadkach operacją dominującą jest porównanie. Liczba tych działań nie zależy od wartości elementów przeszukiwanej tablicy i wynosi $n-1$. Otrzymaliśmy więc złożoność liniową rzędu $O(n)$.

Zadanie 2.39. Podaj specyfikację zadania i skonstruuj algorytm w postaci programu wyznaczający minimum dla n liczb rzeczywistych wygenerowanych losowo z przedziału $[-5, 40]$.

Zadanie 2.40. Podaj specyfikację zadania i skonstruuj algorytm w postaci programu wpisujący do tablicy jednowymiarowej n liczb całkowitych wygenerowanych losowo z przedziału $(-2, 119]$ i wyznaczający maksymalny element w tej tablicy oraz określający liczbę jego wystąpień.

Wybrane zadania maturalne

Na załączonej do podręcznika płycie CD zamieszczono treści wybranych zadań maturalnych. Zaproponowane zadania wymagają od ucznia znajomości zagadnień związanych z wyznaczaniem maksymalnego lub minimalnego elementu.

- *matura2.13.pdf* (*Informator maturalny od 2009 r. Arkusz I, poziom rozszerzony, zadanie 1. SZACHOWNICA*).
- *matura2.14.pdf* (*Egzamin maj 2009 r. Arkusz I, poziom podstawowy, zadanie 1. RZUT OSZCZEPEM*).

2.7. Znajdowanie lidera w zbiorze

Definicja

Liderem nazywamy taki element n-elementowego zbioru, którego liczba wystąpień jest większa niż $n/2$.

Przyjmijmy, że w 6-elementowym zbiorze {2, 3, 1, 1, 4, 2} żaden z jego elementów nie występuje więcej niż 3 razy. Zgodnie z podaną definicją zbiór ten nie zawiera lidera. Natomiast w 8-elementowym zbiorze {3, 1, 3, 4, 3, 3, 0, 3} liczba 3 pojawia się 5 razy, a więc wartość ta jest liderem. Wynika to ze spełnienia warunku $5 > {}^{8}/_{2}$.

Zadaniem algorytmu jest sprawdzenie, czy zbiór zawiera lidera, oraz określenie jego wartości. Działanie algorytmu oparte jest na pewnej własności zbioru, o którym wiemy, że zawiera lidera. **Jeśli ze zbioru mającego lidera usuniemy dwa elementy *a* i *b* o różnych wartościach ($a \neq b$), to uzyskamy nowy zbiór, który również ma lidera.** Załóżmy, że dany jest zbiór {2, 3, 4, 2, 4, 2, 2, 5, 2}, z którego usuwamy wartości 2 i 3. Po redukcji uzyskujemy następujący zbiór: {4, 2, 4, 2, 2, 5, 2}. W obydwu zbiorach liderem jest liczba 2.

Zauważ, że nie można przypisać analogicznej własności zbiorom, które nie mają lidera. Weźmy na przykład zbiór {1, 3, 4, 1, 5, 1}, w którym nie ma lidera. Po usunięciu wartości 3 i 5 uzyskujemy nowy zbiór {1, 4, 1, 1}. Wartość 1 jest w tym zbiorze liderem, ponieważ występuje 3 razy. Wynika to ze spełnienia warunku $3 > {}^{4}/_{2}$.

Redukując w ten sposób liczbę elementów zbioru, dochodzimy do elementu, który jest kandydatem na lidera. Jeżeli w tym zbiorze jest lider, to będzie nim właśnie ten element. Wystarczy sprawdzić, czy wyznaczona wartość występuje w tym zbiorze więcej niż ${}^{n}/_{2}$ razy. Jeśli ten warunek jest spełniony, to lider został znaleziony, w przeciwnym wypadku w zbiorze nie ma lidera.

Specyfikacja:

Dane: Liczba naturalna: $n > 0$ (liczba elementów tablicy T).

n-elementowa tablica jednowymiarowa zawierająca liczby rzeczywiste: $T[0 \ldots n-1]$ (zbiór, w którym szukamy lidera).

Wynik: Lider znaleziony w tablicy $T[0 \ldots n-1]$ lub komunikat informujący, że w tablicy nie ma lidera.

Lista kroków:

Krok 0. Wczytaj wartości danych n, $T[0 \ldots n-1]$.

Krok 1. Przypisz *lider* = $T[0]$ i *pom* = 1.

Krok 2. Dla kolejnych wartości i: $1, 2, \ldots, n-1$, wykonuj kroki 3. – 5., a następnie przejdź do kroku 6.

Krok 3. Jeśli *pom* = 0, wykonaj krok 4., w przeciwnym wypadku przejdź do kroku 5.

Krok 4. Przypisz *lider* = $T[i]$ i *pom* = 1.

Krok 5. Jeśli $T[i]$ = *lider*, zwiększ *pom* o 1, w przeciwnym wypadku zmniejsz *pom* o 1.

Krok 6. Jeśli *pom* = 0, wypisz komunikat „w zbiorze nie ma lidera" i zakończ algorytm, w przeciwnym wypadku przejdź do kroku 7.

Krok 7. Przypisz *ilosc* = 0.

Krok 8. Dla kolejnych wartości *i*: 0, 1, …, *n*–1, wykonuj krok 9., a następnie przejdź do kroku 10.

Krok 9. Jeśli *T*[*i*] = *lider*, zwiększ *ilosc* o 1.

Krok 10. Jeśli $ilosc > \frac{n}{2}$, wypisz wartość zmiennej *lider*, w przeciwnym wypadku wypisz komunikat „w zbiorze nie ma lidera". Zakończ algorytm.

W krokach 3. – 5. następuje łączenie elementów w parzyste grupy, gdzie połowę każdej grupy stanowią wystąpienia „aktualnego lidera", a drugą połowę — równoważące go elementy o innych wartościach.

Funkcja w języku C++ (prog2_19.cpp):

```
void szukaj (double T[], int n)
{
 double lider=T[0];
 int pom=1, ilosc=0;
 for (int i=1;i<n;i++)
  if (pom==0)
  {
   lider=T[i];
   pom=1;
  }
  else if (T[i]==lider) pom++;
       else pom--;
 if (pom==0) cout<<"w zbiorze nie ma lidera"<<endl;
 else
 {
  for (int i=0;i<n;i++)
   if (T[i]==lider) ilosc++;
  if (ilosc>n/2) cout<<"liczba "<<lider<<" jest liderem"<<endl;
  else cout<<"w zbiorze nie ma lidera"<<endl;
 }
}
```

Procedura w języku Pascal (prog2_19.pas):

```
procedure szukaj (T: tablica; n: integer);
var lider: real;
    i, pom, ilosc: integer;
```

```
begin
 lider:=T[0];
 pom:=1;
 ilosc:=0;
 for i:=1 to n-1 do
  if pom=0 then
  begin
   lider:=T[i];
   pom:=1
  end
  else if T[i]=lider then pom:=pom+1
       else pom:=pom-1;
 if pom=0 then writeln('w zbiorze nie ma lidera')
 else
 begin
  for i:=0 to n-1 do
   if T[i]=lider then ilosc:=ilosc+1;
  if ilosc>n div 2 then writeln('liczba ',lider:0:2,' jest liderem')
  else writeln('w zbiorze nie ma lidera')
 end
end;
```

Operacją dominującą w przedstawionym algorytmie jest porównanie. Wyznaczając liczbę wystąpień tych działań, możemy określić klasę złożoności algorytmu. W pierwszej pętli, w której szukamy kandydata na lidera, mamy $n-1$ wykonań pętli, co określa liczbę porównań. Kolejnym miejscem wystąpienia tego działania jest sprawdzenie, czy został znaleziony kandydat na lidera, czyli warunek `pom==0`. W tym miejscu wykonujemy tylko jedno porównanie. Ostatnią częścią algorytmu jest określenie, czy wybrany kandydat rzeczywiście jest liderem. W tej pętli przeglądane są wszystkie elementy zbioru, stąd wykonujemy n porównań. Ostatnie porównanie wykonywane jest przy sprawdzaniu wartości zmiennej *ilosc*, która określa liczbę wystąpień w zbiorze wybranego kandydata na lidera. Łącznie otrzymujemy $(n-1)+1+n+1 = 2n+1$ operacji porównania. Przedstawiona metoda ma więc złożoność czasową rzędu $O(n)$.

Wskazówka

Algorytm znajdujący lidera w zbiorze może być wykorzystany na przykład w sytuacji, gdy liczone są głosy podczas wyborów, w których zwycięzca musi uzyskać więcej niż połowę głosów.

Zadanie 2.41. Określ, czy w podanych zbiorach jest lider. Odpowiedzi uzasadnij.

a) {1, 2, 3, 4, 3, 4, 5, 3, 3, 1},

b) {−3, 0, 0, 3, 0, −2, 0},

c) {9, 6, 7, 6, 2, 2, 2, 2}.

Sprawdź działanie algorytmu przedstawionego w tym punkcie dla podanych zbiorów.

Zadanie 2.42. Skonstruuj schemat blokowy algorytmu znajdującego lidera w zbiorze.

Wybrane zadania maturalne

Na załączonej do podręcznika płycie CD zamieszczono treść wybranego zadania maturalnego oraz niezbędne dane. Zaproponowane zadanie wymaga od ucznia znajomości zagadnień związanych ze znajdowaniem lidera w zbiorze.

- *matura2.15.pdf* (*Egzamin maj 2005 r. Arkusz II, zadanie 5. NAJLEPSZE SUMY, NAJPOPULARNIEJSZE ELEMENTY*).

2.8. Sprawdzanie monotoniczności ciągu liczbowego

Definicja

Dla ciągu liczbowego można określić jego monotoniczność. Mówimy, że **ciąg jest monotoniczny**, jeśli każda para kolejnych wyrazów tego ciągu spełnia określone warunki dotyczące uporządkowania.

Załóżmy, że dany mamy n-wyrazowy ciąg liczbowy $(a_n) = (a_0, a_1, ..., a_{n-1})$. Ciąg ten jest monotoniczny, jeżeli dla $i = 0, 1, \ldots, n-2$ spełniony jest jeden z warunków podanych w tabeli 2.2.

Tabela 2.2. Typy ciągów monotonicznych

Typ ciągu monotonicznego	Warunek	Przykład ciągu liczbowego
rosnący	$a_i < a_{i+1}$ (czyli $a_{i+1} - a_i > 0$)	$(a_6) = (1, 2, 7, 9, 11, 15)$
malejący	$a_i > a_{i+1}$ (czyli $a_{i+1} - a_i < 0$)	$(a_7) = (8, 7, 6, 4, 3, 1, 0)$
nierosnący	$a_i \geq a_{i+1}$ (czyli $a_{i+1} - a_i \leq 0$)	$(a_8) = (11, 7, 7, 6, 5, 4, 4, 2)$
niemalejący	$a_i \leq a_{i+1}$ (czyli $a_{i+1} - a_i \geq 0$)	$(a_9) = (2, 3, 3, 4, 5, 7, 7, 7, 8)$

Ciąg liczbowy, którego nie można zaliczyć do żadnego z podanych typów, jest ciągiem niemonotonicznym. Przykładem takiego ciągu jest $(a_9) = (1, 4, 8, 3, 7, 1, 2, 3, 10)$.