16. W poszukiwaniu lidera i idola

Słowa „lider" i „idol" są dziś bardzo popularne. Często mówi się o liderach zespołów, partii, wyścigów, klasyfikacji oraz o idolach muzycznych, sportowych, filmowych. Lider to ktoś, kto przewodzi grupie ludzi lub – jak np. w zawodach sportowych czy sondażach wyborczych – wygrywa, ma przewagę nad innymi. Z kolei jedną z głównych cech idola jest to, że jest rozpoznawalny, czyli znany przez wiele osób, których sam najczęściej nie zna. Wymienione cechy lidera i idola sprawiły, że pojęcia te przeniesiono na grunt algorytmiki. W tym temacie zajmiemy się właśnie poszukiwaniem lidera i idola w zbiorze.

Cele lekcji

- Znajdziesz lidera w zbiorze danych.
- Poszukasz idola w grupie osób.
- Poznasz algorytmy znajdowania lidera i idola różniące się złożonością czasową.
- Wykorzystasz funkcję `sort` z biblioteki STL.
- Poznasz tablice dwuwymiarowe i zastosujesz je podczas znajdowania idola.

16.1. Jak znaleźć lidera w zbiorze?

Lider

W ujęciu algorytmicznym **liderem** nazywamy taki element zbioru n-elementowego, który występuje w tym zbiorze więcej niż $n/2$ razy.

Sformułujmy specyfikację problemu i zastanówmy się nad algorytmami go rozwiązującymi.

Warto wiedzieć

W wyborach przeprowadzanych zgodnie z ordynacją większościową wygrywa kandydat, który uzyskał większość wszystkich oddanych głosów. To znaczy, że na n oddanych głosów zwycięzca musi zdobyć przynajmniej $n/2 + 1$ głosów.

Specyfikacja

Dane: `A[0..n-1]` – tablica n liczb całkowitych nieujemnych.

Wynik: `lider` – liczba, która występuje w tablicy `A` więcej niż `n div 2` razy, lub –1, gdy takiej liczby nie ma.

Zauważ, że lider w zbiorze n-elementowym może być co najwyżej jeden. Gdyby liderami były dwa elementy, musielibyśmy mieć co najmniej $2 \cdot (n/2 + 1) = n + 2$ elementów, czyli więcej, niż liczy zbiór.

Najprostszym sposobem na znalezienie lidera wydaje się sprawdzanie kolejnych elementów tablicy. Wystarczy policzyć, ile razy w tablicy powtarza się pierwszy element. Jeśli występuje więcej niż $n/2$ razy, to jest on liderem. Jeśli nie, to powtarzamy czynności dla drugiego elementu tablicy itd. Jeśli żaden element nie będzie występował odpowiednio wiele razy, to lidera nie ma, czyli wynikiem będzie wartość –1.

Oto zapis tego algorytmu w pseudokodzie:

```
funkcja SzukajLidera(A[])
    dla i ← 0, 1, ..., n - 1 wykonuj
        ile ← 0
        dla j ← 0, 1, ..., n - 1 wykonuj
            jeśli A[j] = A[i] to ile ← ile + 1
        jeśli ile > n div 2 to zwróć A[i] i zakończ
    zwróć -1 i zakończ
```

Ćwiczenie 1

Zastanów się, jak można by nieco usprawnić powyższy algorytm. Czy trzeba sprawdzać wszystkie elementy tablicy? Kiedy można stwierdzić, że lider w zbiorze nie występuje? Zapisz poprawiony algorytm w pseudokodzie.

Zastanówmy się, czy posortowanie elementów tablicy ułatwiłoby znalezienie lidera. Ponieważ musi on występować więcej niż $n/2$ razy, to jedynym kandydatem na lidera w ciągu uporządkowanym jest element środkowy. Wystarczy więc policzyć, ile razy występuje w tablicy element znajdujący się na pozycji n **div** 2.

Oto algorytm poszukiwania lidera z wykorzystaniem sortowania zapisany w pseudokodzie:

```
funkcja SzukajLidera(A[])
    Sortuj(A)
    ile ← 0
    kandydat ← A[n div 2]
    dla i ← 0, 1, ..., n - 1 wykonuj
        jeśli A[i] = kandydat to ile ← ile + 1
    jeśli ile > n div 2 to zwróć kandydat i zakończ
    w przeciwnym przypadku zwróć -1 i zakończ
```

W powyższym zapisie zastosowaliśmy funkcję o nazwie Sortuj, która sortuje liczby wybraną metodą.

Warto wiedzieć

Element środkowy w tablicy n-elementowej, indeksowanej od 0, ma indeks n **div** 2. Dla n nieparzystego jest to rzeczywiście element środkowy. Dla n parzystego w tablicy występują dwa elementy środkowe – można wówczas wybrać dowolny z nich. Indeks n **div** 2 wskaże element środkowy znajdujący się bliżej końca tablicy.

Ćwiczenie 2

Porównaj dwa powyższe algorytmy przy założeniu, że w drugim zastosujesz jedną z poznanych wcześniej **metod sortowania prostego**. Oceń złożoność czasową tych algorytmów.

Metody sortowania prostego, s. 170–179

Funkcja losująca dane

Zanim napiszemy funkcję poszukującą lidera, musimy w odpowiedni sposób przygotować dane wejściowe, czyli tablicę z losowymi liczbami. Nie wystarczy w tym przypadku zwykłe wylosowanie tablicy, ponieważ prawdopodobieństwo, że w tak utworzonym ciągu jakiś element wystąpi ponad $n/2$ razy (czyli będzie liderem), jest bliskie zeru.

Dobra rada

Zwykłe losowanie tablicy możesz zastosować w przypadku, gdyby tablica miała się składać wyłącznie z zer i jedynek.

Dobra rada

Dane możesz przygotować ręcznie, np. w pliku tekstowym. Program powinien na początku wczytać je do tablicy.

Wylosujmy liczby w taki sposób, żeby istniała realna szansa, że wśród nich będzie lider. W tym celu najpierw wylosujemy jedną liczbę i zapamiętamy ją w zmiennej `x`. Następnie dla każdego elementu tablicy będziemy losować wartość zero-jedynkową, która będzie decydować o tym, czy dany element tablicy wypełnić wartością losową czy wartością zmiennej `x`.

Oto kod źródłowy funkcji losującej tablicę:

Kod źródłowy funkcji losującej tablicę potencjalnie zawierającą lidera

```
1.  void Losuj(int A[])
2.  {
3.      int x=rand()%100;
4.      for (int i=0;i<N;i++)
5.          if (rand()%2==0) A[i]=rand()%100;
6.          else A[i]=x;
7.  }
```

W linii 3 zmiennej `x` przypisana jest losowa wartość od 0 do 99. Będzie to wartość potencjalnego lidera w zbiorze. W pętli (linie 4–6) podejmowana jest losowa decyzja (z prawdopodobieństwem 50%), czy w aktualnym polu tablicy ma się pojawić wartość `x` czy losowa liczba z przedziału [0; 99].

Ćwiczenie 3

Napisz program realizujący pierwszy algorytm znajdowania lidera (sprawdzanie każdego elementu oddzielnie). Algorytm zaimplementuj w postaci funkcji. Wykorzystaj powyższą funkcję `Losuj`, uzupełnij także program o funkcję wypisującą tablicę.

Dobra rada

Kiedy losujesz tablicę, pamiętaj, aby dodać do programu biblioteki `ctime` oraz `cstdlib`.

Funkcja szukająca lidera

Zaimplementujemy teraz funkcję poszukującą lidera w wylosowanej tablicy zgodnie z drugim algorytmem (z uporządkowaną tablicą). Do posortowania elementów tablicy możemy wykorzystać jeden z poznanych wcześniej algorytmów sortowania.

Biblioteka STL

Możemy też skorzystać z **biblioteki STL** (ang. *Standard Template Library* – standardowa biblioteka szablonów), która zawiera zbiór szablonów wielu algorytmów i struktur danych, w tym **funkcji sortującej `sort`**. Funkcję tę wywołujemy w następujący sposób:

Funkcja `sort`

```
sort(tablica, tablica + liczba elementów do sortowania)
```

Warto wiedzieć

Za pomocą funkcji `sort` można posortować fragment tablicy oraz zdefiniować własny porządek sortowania (domyślnie stosowana jest relacja <, zapewniająca porządek rosnący).

Pierwszym parametrem funkcji `sort` jest nazwa tablicy, a drugim – nazwa tablicy, znak + oraz liczba elementów do posortowania. Na przykład sortowanie `N` elementów tablicy `T` z wykorzystaniem funkcji `sort` zapisujemy następująco: `sort(T,T+N)`. Aby skorzystać z funkcji `sort` w języku C++, należy na początku kodu programu dołączyć bibliotekę `algorithm`.

Oto zapis funkcji SzukajLidera, realizującej drugi z przedstawionych wcześniej algorytmów:

Kod źródłowy funkcji poszukującej lidera w zbiorze uporządkowanym

```
1.  int SzukajLidera(int A[])
2.  {
3.      int i, kandydat, ile=0;
4.      sort(A,A+N);
5.      kandydat=A[N/2];
6.      for (i=0;i<N;i++)
7.          if (A[i]==kandydat) ile++;
8.      if (ile>N/2) return kandydat;
9.      else return -1;
10. }
```

W linii 4 wywoływana jest funkcja **sort**. Stała N określa rozmiar tablicy, dlatego drugim parametrem jest A+N. W linii 5 wskazywany jest kandydat na lidera – element środkowy tablicy A.

Ćwiczenie 4

Napisz program wykorzystujący powyższą funkcję szukającą lidera, w której używa się funkcji **sort** z biblioteki STL.

16.2. Efektywny algorytm poszukiwania lidera

Omówimy jeszcze jeden algorytm, który najpierw wyłoni kandydata na lidera bez porządkowania zbioru, a potem sprawdzi, czy rzeczywiście występuje on odpowiednią liczbę razy.

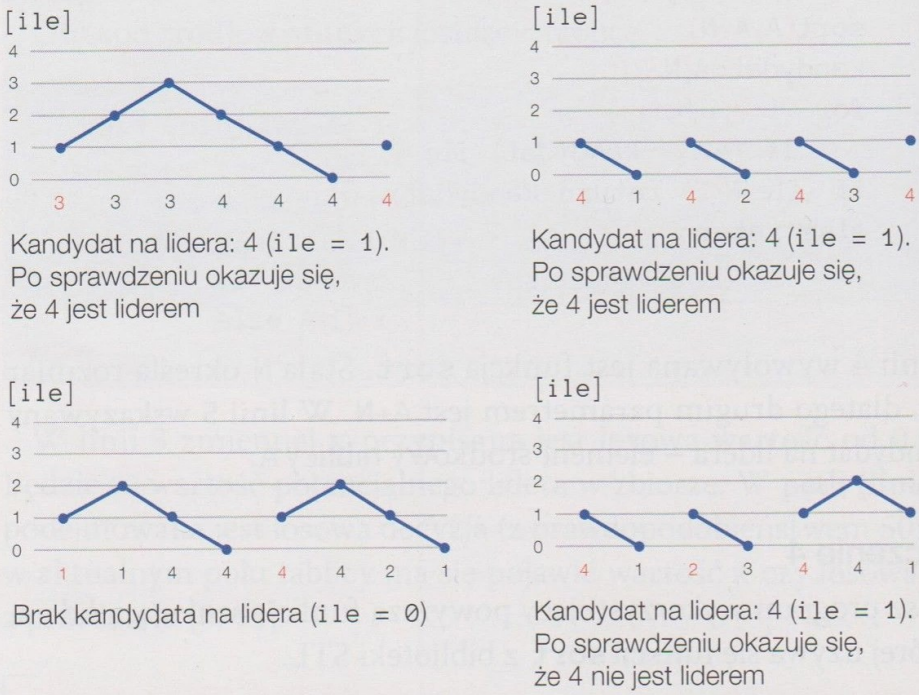
Zauważ, że jeżeli ze zbioru mającego lidera usuniemy dwie różne liczby, to lider się nie zmienia. Dzieje się tak, ponieważ:

- jeśli żadna z usuwanych liczb nie jest liderem, to liczba wystąpień lidera się nie zmienia, a zbiór liczy o dwa elementy mniej,
- jeśli jedna z usuwanych liczb jest liderem, to liczba wystąpień lidera zmniejszy się o jeden, jednocześnie liczebność zbioru zmniejszy się o dwa elementy, więc nadal lider występuje więcej niż $n/2$ razy.

Nie będziemy usuwać liczb z tablicy, ale zliczać wystąpienia kandydata na lidera. Najpierw za kandydata przyjmiemy pierwszy element i ustawimy licznik jego wystąpień na 1. Jeśli kolejny element jest taki sam, to zwiększamy licznik wystąpień, w przeciwnym przypadku go zmniejszamy. Gdy licznik osiągnie wartość 0, kolejna liczba staje się kandydatem.

Jeśli po przejrzeniu całego zbioru licznik będzie równy 0, to lider nie występuje w zbiorze. Dodatnia wartość licznika nie oznacza jeszcze, że wyłoniony kandydat jest liderem. Należy to sprawdzić, czyli policzyć jego wystąpienia.

Działanie algorytmu poszukiwania lidera dla różnych zbiorów danych ilustruje rysunek 16.1. Widać na nim zmianę wartości zmiennej `ile`. Ta wartość wskazuje, czy brana pod uwagę liczba jest kandydatem na lidera. Liczby zaznaczone na czerwono są rozpatrywane jako kandydaci na lidera w zbiorze.



Rys. 16.1. Schematy objaśniające, jak się zlicza wystąpienia kandydatów na lidera w zbiorze

Zauważ, że jeśli wartość zmiennej `ile` po rozpatrzeniu ostatniego elementu zbioru jest większa od zera (na rys. 16.1 występują trzy takie sytuacje), to został ustalony kandydat na lidera. Jest nim ostatnia liczba zaznaczona na czerwono. Kandydata na lidera należy jeszcze sprawdzić, czyli zliczyć jego wystąpienia w zbiorze.

Oto zapis algorytmu w pseudokodzie:

Warto wiedzieć

W pseudokodach funkcji należy podawać parametry, na których operują. Robi się to podobnie jak w językach programowania. Jeśli parametrem jest tablica `T`, to zapisuje się ją jako `T[]`.

```
funkcja SzukajLidera(A[])
    ile ← 0
    dla i ← 0, 1, ..., n - 1 wykonuj
        jeśli ile = 0 to
            kandydat ← A[i]
            ile ← 1
        w przeciwnym przypadku
            jeśli A[i] = kandydat to ile ← ile + 1
            w przeciwnym przypadku ile ← ile - 1
    jeśli ile = 0 to zwróć -1 i zakończ
    ile ← 0
    dla i ← 0, 1, ..., n - 1 wykonuj
        jeśli A[i] = kandydat to ile ← ile + 1
    jeśli ile > n div 2 to zwróć kandydat i zakończ
    w przeciwnym przypadku zwróć -1 i zakończ
```

Kod źródłowy funkcji SzukajLidera, realizującej powyższy algorytm, może być następujący:

Kod źródłowy funkcji poszukującej lidera w zbiorze

```
1.  int SzukajLidera(int A[])
2.  {
3.      int i, kandydat, ile=0;
4.      for (i=0;i<N;i++)
5.          if (ile==0)
6.          {
7.              ile=1;
8.              kandydat=A[i];
9.          }
10.         else if (A[i]==kandydat) ile++;
11.              else ile--;
12.     if (ile==0) return -1;
13.     ile=0;
14.     for (i=0;i<N;i++)
15.         if (A[i]==kandydat) ile++;
16.     if (ile>N/2) return kandydat;
17.     else return -1;
18. }
```

Pętla w liniach 4–11 wyznacza kandydata na lidera. Jeśli go nie znajdzie (wartość zmiennej ile będzie równa 0), to wynikiem funkcji jest –1 (linia 12). W pętli w liniach 14–15 następuje sprawdzenie, czy wyłoniony kandydat rzeczywiście jest liderem.

Rysunek 16.2 pokazuje przykładowe wywołanie programu dla zbioru 10-elementowego.

```
58 32 32 11 32 32 10 32 32 32
Lider = 32
```

Rys. 16.2. Przykładowe wywołanie programu poszukującego lidera w zbiorze

Ćwiczenie 5

Napisz program wykorzystujący powyższą funkcję i sprawdź jego działanie.

Zapamiętaj

Biblioteka STL dostarcza wielu użytecznych szablonów struktur danych i algorytmów, np. funkcję **sort**, umożliwiającą porządkowanie danych. Żeby korzystać z określonej struktury danych lub algorytmów, należy dołączyć do programu odpowiednią bibliotekę – dla funkcji **sort** jest to biblioteka algorithm.

16.3. Porównanie złożoności czasowej algorytmów poszukiwania lidera

Warto wiedzieć

Pierwszy algorytm można nieznacznie poprawić przez ograniczenie poszukiwania lidera tylko do połowy tablicy. Algorytm wykona wtedy dwa razy mniej operacji, ale jego złożoność czasowa nadal będzie wynosić $O(n^2)$.

W pierwszym z omówionych algorytmów jeśli nie ma lidera, to dla każdego elementu należy przejrzeć całą tablicę, a więc wykonać n^2 operacji porównania elementów tablicy.

Złożoność czasowa drugiego algorytmu jest uzależniona od zastosowanej metody sortowania. Po posortowaniu tablica jest przeglądana jeden raz, a więc wykonywanych jest dodatkowo n operacji. Jeśli zastosujemy jedną z poznanych wcześniej metod sortowania prostego o złożoności czasowej $O(n^2)$, nie uzyskamy poprawy względem pierwszego algorytmu. Użyta w przykładzie funkcja `sort` z biblioteki STL działa znacznie szybciej niż metody sortowania prostego – jej złożoność czasowa wynosi $O(n \cdot \log n)$.

Trzeci algorytm wykonuje o wiele mniej operacji. Przegląda on tablicę dwa razy: pierwszy raz w celu wyznaczenia kandydata na lidera, a drugi raz, żeby sprawdzić, czy kandydat rzeczywiście jest liderem. Złożoność czasowa tego algorytmu jest liniowa – wynosi $O(n)$.

Liczba elementów zbioru (n)	Przybliżona liczba operacji dominujących		
	Algorytm 1 $O(n^2)$	Algorytm 2 $O(n \cdot \log n)$	Algorytm 3 $O(n)$, dokładniej $2 \cdot n$
100	10 000	700	200
1 000	1 000 000	10 000	2 000
10 000	100 000 000	132 000	20 000

Tabela 16.1. Porównanie złożoności czasowej algorytmów znajdowania lidera

16.4. Jak znaleźć idola w zbiorze?

Idol

Z algorytmicznego punktu widzenia **idolem** w grupie osób nazywamy taką osobę, którą znają wszyscy w tej grupie, ale ona nie zna nikogo. Zastanówmy się, jaka struktura danych może reprezentować informację o znajomości dwóch osób.

Załóżmy, że grupa liczy n osób. Ile par można utworzyć w obrębie takiej grupy? Pierwsza osoba może być w parze ze wszystkimi pozostałymi, czyli $n - 1$ osobami, druga z $n - 2$ (bo policzyliśmy już parę z pierwszą osobą) itd. Łączna liczba par wynosi:

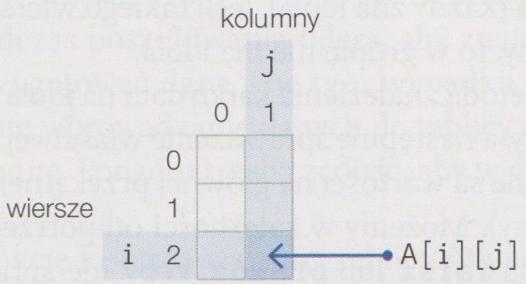
$$(n-1)+(n-2)+\ldots+2+1=\frac{n\cdot(n-1)}{2}$$

Dla każdej pary należy pamiętać dwie informacje, ponieważ relacja znajomości między dwoma osobami nie jest symetryczna, tzn. jedna osoba może znać drugą, ale druga nie musi znać pierwszej. Należy więc przechować $n \cdot (n - 1)$ informacji o znajomości poszczególnych par.

Informacje te zapamiętamy w **tablicy dwuwymiarowej** o n wierszach i n kolumnach. W takiej tablicy mamy n^2 elementów, czyli możemy zapisać wszystkie informacje o znajomościach w grupie n osób.

● Tablica dwuwymiarowa

Liczba wierszy i kolumn w tablicy dwuwymiarowej nie musi być równa. Można deklarować tablicę o n wierszach i m kolumnach. Rysunek 16.3 przedstawia tablicę A o 3 wierszach i 2 kolumnach.



Rys. 16.3. Przykład tablicy dwuwymiarowej 3 na 2 elementy

Do poszczególnych elementów odwołujemy się, podając dwa indeksy – numer wiersza i numer kolumny, np. `A[i][j]`.

Informacja o tym, czy osoba i zna osobę j, jest informacją logiczną, więc w tablicy będą przechowywane wartości **`prawda`** lub **`fałsz`**. Jeśli element `A[i][j]` przyjmuje wartość **`prawda`**, oznacza to, że osoba i zna osobę j. W przeciwnym razie (gdy `A[i][j]` ma wartość **`fałsz`**) osoba i nie zna osoby j. Oto specyfikacja problemu znajdowania idola:

Warto wiedzieć

Można także deklarować tablice wielowymiarowe (o więcej niż dwóch wymiarach), ale w praktyce stosuje się je bardzo rzadko.

Specyfikacja

Dane: `n` – liczba całkowita większa od 1 określająca liczebność grupy, `A[0..n-1][0..n-1]` – tablica `n` na `n` elementów zawierająca wartości logiczne określające znajomości w grupie. Jeśli `A[i][j]` przyjmuje wartość **`prawda`**, to osoba `i` zna osobę `j`, $i \neq j, 0 \leq i < n, 0 \leq j < n$. Osoba `i` nie zna osoby `j`, gdy `A[i][j]` przyjmuje wartość **`fałsz`**.

Wynik: `idol` – liczba, która określa numer idola w grupie, lub –1, gdy idola nie ma w zbiorze.

Rysunek 16.4 przedstawia przykładową tablicę znajomości dla grupy 5 osób. Wartości na głównej przekątnej (znajomość osoby samej ze sobą) nie mają znaczenia, dlatego pola te są szare.

Osoba o numerze 0 zna osoby 2 i 3, a nie zna osób 1 i 4. Osoba 1 zna 0 i 3, a nie zna 2 i 4 itd. W powyższej grupie znajduje się idol – jest to osoba o numerze 3, którą znają wszyscy, a ona nie zna nikogo.

	0	1	2	3	4
0		fałsz	prawda	prawda	fałsz
1	prawda		fałsz	prawda	fałsz
2	prawda	fałsz		prawda	prawda
3	fałsz	fałsz	fałsz		fałsz
4	fałsz	fałsz	prawda	prawda	

Rys. 16.4. Przykładowa tablica znajomości dla 5 osób

Warto wiedzieć

Elementy na głównej przekątnej oznaczają, że dana osoba zna samą siebie.

Algorytm poszukiwania idola

W grupie osób nie może być dwóch idoli. Pierwszy musiałby znać drugiego, a drugi pierwszego, podczas gdy idol nie zna nikogo. Szukanie idola sprowadza się do znalezienia wiersza złożonego z samych wartości **fałsz** (kandydat na idola nie zna nikogo) i sprawdzenia, czy kolumna o indeksie będącym numerem kandydata składa się z samych wartości **prawda** (każdy zna idola). Jeśli takiego wiersza i takiej kolumny nie znajdziemy, to w grupie nie ma idola.

Najprostszą metodą znalezienia kandydata na idola jest przeglądanie kolejnych wierszy, a następnie sprawdzenie właściwej kolumny. Pamiętajmy, że nieważne są wartości na głównej przekątnej (taki sam indeks wiersza i kolumny). Możemy w zależności od potrzeb wpisać do tych komórek wartość **fałsz** lub **prawda**, żeby nie sprawdzać dodatkowo, czy indeksy są różne. Algorytm znajdowania idola można zapisać w pseudokodzie następująco:

```
funkcja SzukajIdola(A[][])
    kandydat ← 0
    jest_kandydat ← fałsz
    dopóki kandydat<n oraz nie jest_kandydat wykonuj
        i ← 0
        A[kandydat][kandydat] ← fałsz
        dopóki i<n oraz nie A[kandydat][i] wykonuj
            i ← i + 1
        jeśli i = n to jest_kandydat ← prawda
        w przeciwnym przypadku kandydat ← kandydat + 1
    jeśli nie jest_kandydat to zwróć -1 i zakończ
    // sprawdzenie kolumny o indeksie kandydat
    i ← 0
    A[kandydat][kandydat] ← prawda
    dopóki i<n oraz A[i][kandydat] wykonuj i ← i + 1
    jeśli i = n to zwróć kandydat i zakończ
    w przeciwnym przypadku zwróć -1 i zakończ
```

Dobra rada

Jeśli przynajmniej dwie osoby nie znają nikogo, to w grupie nie ma idola. Gdyby w grupie był idol, to musiałby być znany przez wszystkich, także przez te osoby. Dlatego wystarczy, że sprawdzisz tylko pierwszy znaleziony wiersz wypełniony wartościami **fałsz**.

Dobra rada

Elementy na przekątnej nie mają znaczenia przy naszych rozważaniach, dlatego możesz nadawać im wartości wygodne do sprawdzania kluczowych warunków w algorytmie.

Dobra rada

Jeśli po wykonaniu wewnętrznej pętli zmienna `i` ma wartość `n`, to przejrzane zostały wszystkie elementy wiersza i wszystkie miały wartość **fałsz**. Oznacza to, że znaleźliśmy kandydata.

A to ciekawe

Czy algorytmy wybierają nam muzycznych idoli?

Historie dojścia gwiazd muzyki na szczyty sławy często są barwne i niepowtarzalne. Jednak dzięki rosnącej popularności serwisów strumieniowych oraz inteligentnych głośników to wbudowane w nie algorytmy mogą promować nowych idoli. Technologie te na podstawie metadanych często samodzielnie decydują, które utwory zagrać. Jak donosi firma BuzzAngle, zajmująca się analizą rynku muzycznego, w 2017 r. za 99% wszystkich odtworzeń w serwisach strumieniowych odpowiadało tylko 10% utworów.

Ćwiczenie 6

Zapisz algorytm znajdowania idola tak, żeby najpierw poszukiwał kolumny złożonej z wartości **prawda**, a potem sprawdzał odpowiedni wiersz.

Implementacja algorytmu poszukiwania idola

Podobnie jak podczas poszukiwania lidera, aby znaleźć idola, należy odpowiednio przygotować dane – w tym wypadku tablicę wartości logicznych. Ręczne wprowadzanie danych do tablicy dwuwymiarowej byłoby pracochłonne. Ponadto trzeba zadbać, aby w grupie mógł znajdować się idol.

Stworzymy funkcję losującą wartości tablicy. Wypełnimy ją liczbami 0 i 1, reprezentującymi wartości **fałsz** i **prawda**. Prawdopodobieństwo, że w tak wylosowanej tablicy znajdzie się idol, jest bliskie zeru. Dlatego po wypełnieniu tablicy wylosujemy jeszcze informację, czy w grupie ma się znajdować idol. Jeśli tak, to wylosujemy numer idola i ponownie wypełnimy odpowiednie wiersz i kolumnę tablicy znajomości. Oto kod źródłowy funkcji losującej oraz wypisującej na ekranie utworzoną tablicę:

Dobra rada

Dane możesz przygotować ręcznie, np. w pliku tekstowym. Program powinien na początku wczytać je do tablicy.

Kod źródłowy funkcji losującej tablicę oraz wypisującej jej wartości na ekranie

```
1.  const int N=5;
2.
3.  void Losuj(bool A[][N])
4.  {
5.      int i, j, idol;
6.      for (i=0;i<N;i++)
7.          for (j=0;j<N;j++) A[i][j]=rand()%2;
8.      if (rand()%2==1)
9.      {
10.         idol=rand()%N;
11.         for (i=0;i<N;i++) A[i][idol]=true;
12.         for (j=0;j<N;j++) A[idol][j]=false;
13.     }
14. }
15.
16. void Wypisz(bool A[][N])
17. {
18.     for (int i=0;i<N;i++)
19.     {
20.         for (int j=0;j<N;j++)
21.             if (i!=j) cout<<A[i][j]<<" ";
22.             else cout<<"  ";
23.         cout<<endl;
24.     }
25. }
```

Zwróć uwagę na nagłówki funkcji `Losuj` i `Wypisz` (wiersze 3 i 16). W języku C++, gdy parametrem funkcji jest tablica dwuwymiarowa, nie można pominąć w nawiasach jej drugiego rozmiaru (liczby kolumn). W liniach 6 i 7 losowana jest tablica wartości logicznych (0 reprezentuje fałsz, 1 – prawdę). W wierszu 8 losujemy, czy do tablicy znajomości wpisać idola. Jeśli tak, to jest losowany numer idola (linia 10), a następnie są wypełniane określony wiersz i określona kolumna (linie 11–12).

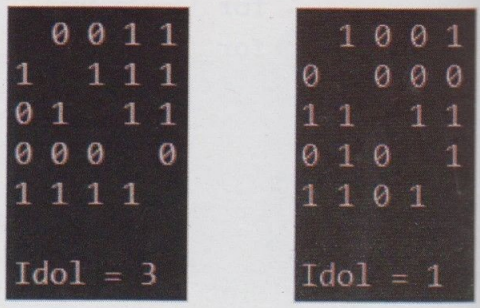
W funkcji `Wypisz` nie są wypisywane elementy leżące na głównej przekątnej, zamiast nich wypisywane są spacje (linie 21–22).

Kod funkcji `SzukajIdola` może być następujący:

Kod źródłowy funkcji poszukującej idola w zbiorze

```
1.  int SzukajIdola(bool A[][N])
2.  {
3.      int i, kandydat=0;
4.      bool jest_kandydat=false;
5.      while (kandydat<N && !jest_kandydat)
6.      {
7.          i=0;
8.          A[kandydat][kandydat]=false;
9.          while (i<N && !A[kandydat][i]) i++;
10.         if (i==N) jest_kandydat=true;
11.         else kandydat++;
12.     }
13.     if (!jest_kandydat) return -1;
14.     i=0; A[kandydat][kandydat]=true;
15.     while (i<N && A[i][kandydat]) i++;
16.     if (i==N) return kandydat;
17.     else return -1;
18. }
```

Pętla w liniach 5–12 poszukuje kandydata na idola, czyli wiersza złożonego z samych wartości **fałsz**. Pętla w linii 15 sprawdza, czy znaleziony kandydat rzeczywiście jest idolem (kolumna złożona z samych wartości **prawda**). Rysunek 16.5 przedstawia przykładowe wywołania programu.



Rys. 16.5. Dwa przykładowe wywołania programu szukającego idola

Ćwiczenie 7

Napisz program znajdujący idola za pomocą omówionego algorytmu z wykorzystaniem funkcji `Losuj`, `Wypisz`, `SzukajIdola`.

16.5. Efektywny algorytm poszukiwania idola

Poprzedni algorytm w najgorszym wypadku musiał przejrzeć raz całą kwadratową tablicę. Poszukajmy efektywniejszego algorytmu.

Zauważ, że przy każdym porównaniu dwóch osób jedną z nich można wykluczyć, ponieważ tylko jedna ma szasnę być idolem. Wystarczy więc wykonać $n - 1$ porównań. Przeanalizujmy znajdowanie kandydata na idola dla tablicy znajomości z rysunku 16.6.

1. Porównujemy osoby o numerach 0 i 1. Kandydatem zostaje 0, 1 nie może być idolem.
2. Porównujemy osoby o numerach 0 i 2. Kandydatem zostaje 2, 0 nie może być idolem.
3. Porównujemy osoby o numerach 2 i 3. Kandydatem zostaje 3, 2 nie może być idolem.
4. Porównujemy osoby o numerach 3 i 4. Kandydatem pozostaje 3, 4 nie może być idolem.

	0	1	2	3	4
0		fałsz	prawda	prawda	fałsz
1	prawda		fałsz	prawda	fałsz
2	prawda	fałsz		prawda	prawda
3	fałsz	fałsz	fałsz		fałsz
4	fałsz	fałsz	prawda	prawda	

Rys. 16.6. Tablica znajomości

Dobra rada

Porównanie znajomości osób *i* oraz *j* możesz rozumieć jako sprawdzenie wartości w *i*-tym wierszu i *j*-tej kolumnie. Jeśli wartością tą jest **fałsz**, to odrzucamy *j*-tą osobę, jeśli **prawda** – odrzucamy *i*-tą osobę.

Następnie należy sprawdzić jeszcze kandydata – zarówno jego wiersz, jak i kolumnę. Oto zapis algorytmu w pseudokodzie:

```
funkcja SzukajIdola(A[][])
    kandydat ← 0
    i ← 1
    // pętla znajdująca kandydata
    dopóki i < n wykonuj
        jeśli A[kandydat][i] to kandydat ← i
        i ← i + 1
    i ← 0
    A[kandydat][kandydat] ← fałsz
    // sprawdzenie wiersza
    dopóki i < n oraz nie A[kandydat][i] wykonuj i ← i + 1
    jeśli i < n to zwróć -1 i zakończ
    i ← 0
    A[kandydat][kandydat] ← prawda
    // sprawdzenie kolumny
    dopóki i < n oraz A[i][kandydat] wykonuj i ← i + 1
    jeśli i = n to zwróć kandydat i zakończ
    w przeciwnym przypadku zwróć -1 i zakończ
```

W zmiennych `kandydat` oraz `i` pamiętane są aktualne numery porównywanych osób. Jeśli element tablicy `A[kandydat][i]` przyjmuje wartość **prawda**, to kandydat zna osobę `i`, więc nie może być idolem. Kandydatem staje się osoba o numerze `i`. Następną osobą do porównania jest osoba o numerze `i + 1`, niezależnie od tego, czy nastąpiła zmiana kandydata.

Oto kod źródłowy funkcji poszukującej idola:

Kod źródłowy funkcji poszukującej idola w zbiorze zgodnie z drugim algorytmem

```
1.  int SzukajIdola(bool A[][N])
2.  {
3.      int kandydat=0, i=1;
4.      while (i<N)
5.      {
6.          if (A[kandydat][i]) kandydat=i;
7.          i++;
8.      }
9.      i=0; A[kandydat][kandydat]=false;
10.     while (i<N && !A[kandydat][i]) i++;
11.     if (i<N) return -1;
12.     i=0; A[kandydat][kandydat]=true;
13.     while (i<N && A[i][kandydat]) i++;
14.     if (i==N) return kandydat;
15.     else return -1;
16. }
```

Pętla w liniach 4–8 wyznacza kandydata na idola. Dwie kolejne pętle (linie 10 i 13) sprawdzają, czy kandydat rzeczywiście jest idolem, czyli czy odpowiedni wiersz (o indeksie określonym wartością zmiennej `kandydat`) jest złożony z samych wartości **`fałsz`**, a odpowiednia kolumna – z samych wartości **`prawda`**.

Ćwiczenie 8

Napisz program poszukujący idola, korzystający z powyższej funkcji.

16.6. Porównanie złożoności algorytmów poszukiwania idola

W przypadku pierwszego z omówionych algorytmów jeśli idol ma numer 0, sprawdzamy tylko jeden wiersz i jedną kolumnę, czyli wykonujemy $2 \cdot n$ operacji. Jeśli w zbiorze nie ma idola, przejrzymy wszystkie elementy tablicy, których jest n^2. Jeśli w grupie jest idol, możemy oczekiwać, że prawdopodobnie należy przejrzeć połowę wierszy i dodatkowo jedną kolumnę, a więc wykonać około $n^2/2 + n$ operacji. Złożoność czasowa tego algorytmu, zarówno pesymistyczna, jak i oczekiwana, jest więc kwadratowa: $O(n^2)$.

W drugim algorytmie aby wyłonić kandydata, należy wykonać $n - 1$ porównań, a następnie przejrzeć jeden wiersz i jedną kolumnę. Ulepszony algorytm wykona więc około $3 \cdot n$ operacji. Jego złożoność czasowa jest zatem liniowa: $O(n)$. Dlatego algorytm można uznać za efektywny.

Podsumowanie

- Lider w n-elementowym zbiorze danych to element, który występuje więcej niż $n/2$ razy.
- Idol w grupie osób to osoba, która nie zna w tej grupie nikogo, ale wszyscy inni ją znają.
- Istnieją szybkie algorytmy o złożoności liniowej, które znajdują lidera i idola w zbiorze danych.
- STL to standardowa biblioteka szablonów wielu algorytmów i struktur danych. Jest uniwersalna, tzn. algorytmy można wykonywać dla różnych typów danych, a w strukturach danych mogą być pamiętane dane różnych typów.
- Jedną z często używanych funkcji z biblioteki STL jest funkcja `sort`. Porządkuje ona dane i ma złożoność czasową $O(n \cdot \log n)$. Aby z niej skorzystać, należy w kodzie źródłowym dołączyć bibliotekę `algorithm`.
- Oprócz tablic jednowymiarowych możemy także korzystać z tablic wielowymiarowych. W praktyce mają zastosowanie głównie tablice dwuwymiarowe.

Zadania

* **1** Napisz program, który wylosuje i wypisze tablicę dwuwymiarową liczb całkowitych (o n wierszach i n kolumnach), a następnie znajdzie w niej maksimum.

* **2** Napisz program, który wylosuje i wypisze tablicę dwuwymiarową liczb całkowitych, następnie posortuje każdy wiersz tablicy i ponownie wypisze tablicę.

* **3** Napisz program, który wylosuje i wypisze tablicę dwuwymiarową liczb całkowitych (o n wierszach i n kolumnach), następnie posortuje każdą kolumnę tablicy i ponownie wypisze tablicę.

** **4** Napisz program, który wylosuje i wypisze tablicę dwuwymiarową liczb całkowitych (o n wierszach i n kolumnach), a następnie sprawdzi, czy w tablicy znajdują się wiersz i kolumna o tej samej sumie liczb. Program powinien wypisać numer wiersza i kolumny lub słowo „BRAK".

** **5** Utwórz n-elementową tablicę liczb z zakresu od 0 do $n - 1$. Przygotuj ją tak, żeby z prawdopodobieństwem 50% żaden element tablicy się nie powtarzał. Napisz program, który sprawdzi, czy w tablicy wszystkie elementy są różne. Program powinien wypisać odpowiedni komunikat słowny.

*** **6** Napisz program sprawdzający, czy w tablicy n-elementowej znajduje się liczba, która występuje więcej niż $n/4$ razy. Przygotuj tablicę tak, aby była szansa wystąpienia w niej takiej liczby. Przyjmij, że n jest wielokrotnością 4. Program powinien wykonywać jak najmniej operacji.