Znajdowanie określonego elementu w zbiorze

Źródło: Alex Block, domena publiczna.

> W tym e-materiale powtarzamy wiadomości ze szkoły podstawowej.

Zastanów się, jak często (nawet nieświadomie!) wykorzystujesz w codziennym życiu algorytmy wyszukiwania określonego elementu. Najczęściej korzystasz z konkretnych kryteriów – szukasz butów w danym rozmiarze czy książek konkretnej autorki. To samo zadanie wykonują ze ciebie wyszukiwarki internetowe, choć one korzystają z innych algorytmów.

Implementację algorytmu wyszukiwania określonego elementu w zbiorze przedstawiamy w e-materiałach:

- Znajdowanie określonego elementu w zbiorze w języku C++,
- Znajdowanie określonego elementu w zbiorze w języku Java,
- Znajdowanie określonego elementu w zbiorze w języku Python.

Więcej zadań? Znajdziesz je w e-materiale Znajdowanie określonego elementu w zbiorze – zadania maturalne

> **Twoje cele**
>
> - Zapoznasz się z iteracyjną wersją algorytmu wyszukiwania binarnego.
> - Wyjaśnisz, czym są lider oraz idol.

- Przeanalizujesz algorytmy wyboru lidera oraz wyszukiwania idola w zbiorze.
- Rozwiążesz zadania wymagające wykorzystania algorytmów wyszukiwania określonego elementu w zbiorze.

Przeczytaj

Wyszukiwanie liniowe

Przypomnijmy sobie najważniejsze informacje dotyczące algorytmu wyszukiwania liniowego.

Polega on na sprawdzaniu, czy kolejne elementy zbioru są szukaną liczbą. Możemy wyróżnić najprostszą wersję algorytmu oraz jego modyfikację – algorytm wyszukiwania liniowego **z wartownikiem.**

Specyfikacja problemu:

Dane:

- `n` – liczba naturalna; liczba elementów w tablicy `tablica`
- `tablica` – `n`-elementowa tablica liczb naturalnych
- `szukana` – liczba naturalna

Wynik:

- komunikat wskazujący indeks, pod którym znajduje się szukana liczba lub informujący, że w tablicy nie ma szukanej liczby

Wyszukiwanie liniowe bez wartownika

Zapiszmy nagłówek funkcji. Przyjmuje ona trzy argumenty – tablicę liczb, szukany element oraz liczbę elementów tablicy.

Na początku przypisujemy dwóm zmiennym pomocniczym odpowiednie wartości. Zmiennej `znaleziono` przypisujemy wartość `fałsz`, a zmiennej `i` wartość 0.

W kolejnym kroku zapisujemy pętlę `dopóki`. Będzie się ona wykonywać tak długo, jak długo będą spełnione dwa warunki – zmienna `znaleziono` będzie przechowywać wartość logiczną `fałsz` oraz wartość zmiennej `i` będzie mniejsza od długości tablicy, czyli wartości zmiennej `n`.

W pętli zapisujemy warunek logiczny `jeżeli`. Będziemy sprawdzać, czy wartość `i`-tego elementu tablicy jest równa szukanej liczbie – jeżeli tak, wypiszemy odpowiedni komunikat, przypiszemy zmiennej `znaleziono` wartość `prawda` i zakończymy działanie algorytmu; jeżeli nie, zwiększymy wartość zmiennej `i` o 1.

W sytuacji, gdy po wyjściu z pętli `dopóki` wartością zmiennej `znaleziono` będzie `fałsz`, wypiszemy komunikat, że szukana liczba nie została znaleziona.

```
funkcja wyszukiwanie_liniowe(tablica, szukana, n)
    znaleziono ← fałsz
    i ← 0
    dopóki znaleziono != prawda oraz i < n wykonuj:
        jeżeli tablica[i] = szukana
            wypisz "Znaleziono liczbę na pozycji " + i
            znaleziono ← prawda
            zakończ działanie pętli
        i ← i + 1

    jeśli znaleziono = fałsz
        wypisz "Nie znaleziono szukanej liczby"
```

Operacjami, które najbardziej wpływają na złożoność tego algorytmu, są operacje porównania (wykonujące się w pętli `dopóki`). Dla `n` liczb może się ich wykonać, zależnie od ułożenia danych, od 1 do `n`, ponieważ w najgorszym wypadku musimy sprawdzić wszystkie liczby. Zatem złożoność czasowa tego algorytmu to $O(n)$.

Wyszukiwanie liniowe z wartownikiem

Wartownik dodajemy na początku lub na końcu tablicy. Zależnie od implementacji może to być zarówno liczba, która nie występuje w tablicy, jak i wartość szukana.

Przedstawimy wersję, w której wartownik jest szukaną liczbą. Umieścimy go na końcu tablicy.

Zapiszmy nagłówek funkcji. Ponownie przyjmuje ona trzy argumenty – tablicę liczb, szukany element oraz liczbę elementów tablicy.

Na początku umieszczamy wartownik na końcu tablicy, a zmiennej pomocniczej `i` przypisujemy wartość 0.

W kolejnym kroku zapisujemy pętlę `dopóki`. Będzie się wykonywać tak długo, jak długo wartość `i`-tego elementu tablicy będzie różna od szukanej liczby.

W każdej iteracji pętli będziemy zwiększać wartość zmiennej pomocniczej o 1.

Zapisujemy warunek logiczny `jeżeli`. Będziemy sprawdzać, czy wartość zmiennej `i` jest mniejsza od różnicy długości tablicy oraz liczby 1. Jeśli tak, wypiszemy odpowiedni komunikat mówiący o tym, że szukana liczba została znaleziona pod indeksem równym `i`.

W przeciwnym wypadku wypiszemy komunikat mówiący o tym, że szukana liczba nie została znaleziona.

```
funkcja wyszukiwanie_liniowe_wartownik(tablica, szukana, n)
    tablica[n] ← szukana
    i ← 0

    dopóki tablica[i] != szukana wykonuj:
        i ← i + 1

    jeśli i < n - 1:
        wypisz "Znaleziono liczbę na pozycji " + i
    w przeciwnym przypadku:
        wypisz "Nie znaleziono szukanej liczby"
```

Ponownie operacjami, które najbardziej wpływają na złożoność tego algorytmu, są operacje porównania (wykonujące się w pętli `dopóki`). Może się ich wykonać, zależnie od ułożenia danych, od 1 do `n`. Zatem złożoność czasowa tego algorytmu to również $O(n)$. Zwróćmy jednak na to, że poprzez dodanie wartownika wyeliminowaliśmy konieczność sprawdzania przy każdym kolejnym elemencie, czy nie przekroczyliśmy rozmiaru tablicy – zredukowaliśmy zatem liczbę instrukcji.

> **Polecenie 1**
>
> Porównaj ze sobą dwie implementacje. Jakie dostrzegasz różnice? Czy potrafisz wskazać korzyści związane z wykorzystaniem wartownika?

Wyszukiwanie binarne

Specyfikacja problemu:

Dane:

- `n` – liczba naturalna; liczba elementów tablicy `tablica`
- `tablica` – `n`-elementowa tablica liczb naturalnych
- `szukana` – liczba naturalna

Wynik:

- komunikat wskazujący indeks, pod którym znajduje się szukana liczba, lub informujący, że w tablicy nie ma szukanej liczby

Algorytm wyszukiwania binarnego wyszukuje wybrany element w posortowanym w zbiorze. W e-materiale Rozwiązywanie problemów informatycznych – strategia „dziel i zwyciężaj" przedstawiliśmy jego wersję rekurencyjną.

W tym e-materiale skupimy się na jego iteracyjnej wersji.

Zasada działania algorytmu nie zmienia się. Algorytm nie wywołuje się rekurencyjnie, wykorzystujemy natomiast pętlę `dopóki`.

Zapisujemy nagłówek funkcji `wyszukiwanie_binarne_iter`. Przyjmie ona cztery argumenty – tablicę liczb, szukaną wartość, indeks początku oraz indeks końca.

W kolejnym kroku określimy środek analizowanego przedziału, przypisując zmiennej `środek` wartość dzielenia całkowitego sumy wartości zmiennych `pocz` oraz `kon` przez liczbę 2.

Następnie zapiszemy instrukcję warunkową `jeżeli`. Sprawdzamy, czy szukana liczba jest równa wartości elementu tablicy o indeksie równym wartości zmiennej `środek`. Jeśli tak, wypisujemy odpowiedni komunikat.

Jeśli element nie jest równy elementowi przechowywanemu pod indeksem `środek`, sprawdzamy, czy jest on od niego mniejszy. Jeśli tak, zmieniamy wartość zmiennej `pocz`, przypisując jej wartość sumy zmiennej `środek` oraz liczby 1.

Jeżeli element jest mniejszy, przypisujemy zmiennej `kon` wartość różnicy zmiennej `środek` oraz liczby 1.

Jeśli po wyjściu z pętli `dopóki` element nie został znaleziony, wypisujemy odpowiedni komunikat.

```
funkcja wyszukiwanie_binarne_iter(tablica, szukana, pocz, kon):
    dopóki pocz <= kon:
        środek ← (pocz + kon) div 2
        jeżeli tablica[środek] = szukana:
            wypisz "Znaleziono liczbę na pozycji " + środek
            zwróć środek
        w przeciwnym wypadku jeżeli tablica[środek] < szukana:
            pocz = środek + 1
        w przeciwnym wypadku:
            kon = środek - 1

    wypisz "Nie znaleziono szukanej liczby"
```

> **Ważne!**
>
> W rozwiązaniu wykorzystujemy operator `div`. Operator `div` oznacza dzielenie całkowitoliczbowe.

Algorytm wyszukiwania binarnego działa przez podział zakresu wyszukiwania na dwie równe części i wybór tej, która zawiera szukaną wartość (lub może ją zawierać). Dzięki temu zakres wyszukiwania jest dzielony na pół w każdej iteracji. W konsekwencji każdy podział sprowadzi problem do problemu o połowę mniejszego.

W związku z tym, jeżeli mamy tablicę o długości n, to w najgorszym wypadku potrzebujemy $O(\log n)$ podziałów, aby zawęzić zakres wyszukiwania do jednego elementu. Stąd złożoność czasowa wynosi $O(\log n)$.

Lider w zbiorze

Czym jest lider? Określamy tak liczbę, która występuje w zbiorze `n`-elementowym więcej niż $\frac{n}{2}$ razy. Ponieważ jedna liczba nie może występować w tym samym zbiorze liczbowym więcej niż $\frac{n}{2}$ razy, lider jest tylko jeden.

Na przykład w zbiorze `{1, 4, 6, 1, 1}` lider to liczba 1, ponieważ występuje w nim aż 3 razy.

$$\frac{n}{2} = \frac{5}{2} = 2,5 \Leftrightarrow 3 > 2,5$$

Natomiast w zbiorze `{8, 6, 4, 5, 2, 2}` żadna z sześciu liczb nie jest liderem, choć liczba 2 występuje więcej razy niż pozostałe.

Zbiory, które mają lidera, posiadają bardzo ważną właściwość: jeżeli z takiego zbioru usuniemy parę różnych liczb, zbiór ten nadal będzie miał tego samego lidera.

Rozważmy dwa przypadki.

Pierwszy dotyczy usunięcia pary liczb, z której żadna nie jest liderem. Ze zbioru `{1, 4, 6, 1, 1}` usuwamy liczby 4 oraz 6 i zostajemy z podzbiorem `{1, 1, 1}`. Ponieważ zbiór ten składa się tylko z jedynek, bardzo łatwo stwierdzić, że lider tego zbioru nie zmienił się i nadal jest nim liczba 1.

W drugim przypadku usuniemy jedną liczbę, która nie jest liderem, i jedną, która nim jest – czyli usuniemy liczby 4 oraz 1. Zostajemy wtedy ze zbiorem `{1, 6, 1}`, którego liderem nadal jest liczba 1. Ponieważ para, która zostaje wyeliminowana, nie może składać się z tych samych liczb, przypadek, w którym usunięte zostaną dwie liczby lidera, nie zaistnieje.

> **Ciekawostka**

> Liderem w zbiorze będzie zwycięzca pierwszej tury polskich wyborów prezydenckich. Więcej na temat sposobu wybierania prezydenta znajdziesz w e-materiałach z **Wiedzy o społeczeństwie**, np. w e-materiale Głowa państwa.

Algorytm wyboru lidera

Specyfikacja problemu:

Dane:

- `n` – liczba naturalna; liczba elementów zbioru `zbiór`
- `zbior` – zbiór liczb naturalnych

Wynik:

- komunikat wskazujący lidera zbioru lub informujący, że zbiór nie ma lidera

Przykład będzie bazował na zbiorze trzyelementowym, ponieważ łatwiej będzie na nim wskazać pewne zależności.

Wykorzystamy poznaną zależność.

Załóżmy, że pierwsza liczba ze zbioru jest liderem, oraz stwórzmy zmienną, która będzie liczyć wystąpienia lidera. Nazwijmy ją `licznik`. Skoro zakładamy, że pierwsza liczba jest liderem, zmienną `licznik` inicjalizujemy wartością 1, ponieważ jedno wystąpienie potencjalnego lidera już nastąpiło. Dodajemy również zmienną, która będzie przechowywała liczbę elementów zbioru.

W pseudokodzie będzie to wyglądało następująco:

```
zbior ← {1, 3, 1}
lider ← zbior[0]
licznik ← 1
n ← 3
```

Następnym krokiem będzie iteracja przez cały zbiór. Ponieważ zakładamy, że pierwszy element zbioru jest liderem, i został on już uwzględniony w zmiennej `licznik`, możemy go pominąć w pętli i zainicjować zmienną sterującą wartością 1.


```
zbior ← {1, 3, 1}
lider ← zbior[0]
licznik ← 1
n ← 3
```

```
5
6 dla i = 1, 2,... n - 1 wykonuj
7
```

Naszym celem jest eliminacja par liczb, które nie są takie same. W konsekwencji, jeżeli kolejny element zbioru nie jest taki sam jak dotychczasowy lider (czyli poprzedni element), zmniejszymy wartość zmiennej `licznik` o jeden. W kolejnej iteracji sprawdzimy, czy zmienna `licznik` ma wartość 0. Jeżeli warunek jest prawdziwy, przesuniemy lidera na miejsce sprawdzanego elementu, ponieważ oznacza to, że para, która została sprawdzona, zawierała dwie różne liczby – może zatem zostać pominięta. Zobaczmy, jak będzie to wyglądać w pseudokodzie, a następnie przetestujmy działanie tej części algorytmu na przykładzie:

```
1  zbior ← {1, 3, 1}
2  lider ← zbior[0]
3  licznik ← 1
4  n ← 3
5
6  dla i = 1, 2,... n - 1 wykonuj
7      jeżeli licznik > 0 to
8          jeżeli lider ← zbior[i] to
9              licznik ← licznik + 1
10         w przeciwnym wypadku
11             licznik ← licznik - 1
12     w przeciwnym wypadku
13         lider ← zbior[i]
14         licznik ← 1
```

Dla przykładu rozważmy zbiór `{1, 3, 1, 3, 1, 3, 1}`, w którym liderem jest 1, i przeanalizujmy działanie algorytmu.

```
1  zbior ← {1, 3, 1, 3, 1, 3, 1}
2  lider ← 1
3  licznik ← 1
4  n ← 3
```

Wchodzimy w pętlę `for`, zmienna sterująca `i` otrzymuje wartość 1. Zmienna `licznik` ma wartość 1, więc jest większa niż 0.

Czy `lider` jest równy `zbior[i]`? Podstawmy wartości i sprawdźmy.

Po podstawieniu wartości wiemy, że wartości tych zmiennych nie są sobie równe.

Odejmujemy od zmiennej sterującej 1 i przechodzimy do kolejnej iteracji. Zmienna `licznik` jest równa 0, więc przenosimy lidera do `zbior[2]` czyli 1,
a zmiennej `licznik` przypisujemy wartość 1. Dzięki temu pierwsza para (1, 3) zostaje usunięta, ponieważ liczby w tej parze nie są takie same.

Co się jednak dzieje, gdy para zawiera te same liczby? Ponieważ liczba powtórzyła się, zostaje kandydatem na lidera. Zwiększamy więc wartość zmiennej `licznik`, która liczy liczbę wystąpień potencjalnego lidera.

W rezultacie działania pętli otrzymamy potencjalnego lidera oraz zmienną `licznik`. Jeżeli wartość zmiennej `licznik` wynosi 0, badany zbiór nie ma lidera. Aby mieć pewność, że potencjalny lider jest faktycznie liderem, policzymy liczbę jego wystąpień w zbiorze
i sprawdzimy, czy liczba ta jest większa niż długość zbioru podzielona na 2. Dodajmy te zmiany do algorytmu:

```
zbior ← {1, 3, 1}
lider ← zbior[0]
licznik ← 1
n ← 3

dla i = 1, 2,...n - 1 wykonuj
    jeżeli licznik > 0 to
        jeżeli lider = zbior[i] to
            licznik ← licznik + 1
        w przeciwnym wypadku
            licznik ← licznik - 1
    w przeciwnym wypadku
        lider ← zbior[i]
        licznik ← 1

jeżeli licznik = 0 to
    wypisz "Zbiór nie ma lidera"
    zakończ działanie programu

liczba_wystapien_lidera ← 0
dla i = 0, 1,...n - 1 wykonuj
    jeżeli zbior[i] = lider to
        liczba_wystapien_lidera = liczba_wystapien_lidera + 1
```

```
24
25 jeżeli liczba_wystapien_lidera > n / 2 to
26     wypisz "Liderem zbioru jest " + lider
27 w przeciwnym wypadku
28     wypisz "Zbiór nie ma lidera"
```

Dominującymi operacjami w omawianym rozwiązaniu są porównania w pętlach wykonujących się dla każdego elementu zbioru wejściowego, a zatem złożoność czasowa algorytmu wynosi $O(n)$.

Kim jest idol w zbiorze?

Specyfikacja problemu:

Dane:

- `n` – liczba naturalna; liczba osób
- `znajomosci` – tablica o wymiarach `n` × `n`; przechowuje liczby naturalne 0 oraz 1; tablica informacji na temat tego, które osoby się znają

Wynik:

- komunikat wskazujący lidera zbioru lub informujący, że zbiór nie ma idola

Na przyjęciu znajduje się `n` osób. Osoby oznaczone są numerami od 0 do $n-1$. Ponieważ jest to duża impreza, nie wszyscy goście się znają. Osoba A może znać osobę B, co niekoniecznie musi oznaczać, że osoba B zna osobę A. Może się okazać, że na przyjęciu pojawiła się osoba na tyle popularna, że jest znana przez wszystkich. Jeżeli ponadto okaże się, że osoba ta nie zna nikogo spośród pozostałych gości, możemy nazwać ją **idolem**.

Poniżej zaprezentowano dwuwymiarową tablicę typu logicznego, która określa stan znajomości wśród osób znajdujących się na przyjęciu. Jeżeli wartość `znajomość[A][B]` wynosi 1 (zauważmy, że to tablica liczb całkowitych, nie tablica boolowska), to przyjmujemy, że osoba A zna osobę B. Dla porządku przyjmujemy, że dana osoba nie zna siebie samej.

```
1 n ← 5
2 znajomość[n][n] ← [
3     [ 0, 1, 0, 1, 1 ],
4     [ 0, 0, 1, 0, 1 ],
5     [ 0, 1, 0, 1, 1 ],
6     [ 1, 1, 1, 0, 1 ],
```

```
    [ 0, 0, 0, 0, 0 ]
]
```

> **Polecenie 2**
>
> Czy na podstawie powyższej tablicy jesteś w stanie wskazać, która osoba jest idolem?

Zastanówmy się, ile pytań „Czy osoba A zna osobę B?” musimy zadać, aby sprawdzić, czy na przyjęciu pojawił się idol oraz kto nim jest.

Spróbujmy dokładnie przeanalizować informację, która wynika z odpowiedzi na pojedyncze pytanie: „Czy osoba A zna osobę B?” w kontekście wyszukiwania idola. Ponieważ idol jest osobą, która musi być znana przez każdego oraz sama nie może znać nikogo, wymienione wyżej pytanie pozwala nam stwierdzić, która z osób A lub B na pewno nie jest idolem. Jeżeli odpowiedź na nie brzmi twierdząco, wiemy, że osoba A nie może być idolem, ponieważ zna przynajmniej jedną osobę. Jeśli odpowiedź jest przecząca, to z kolei osoba B nie może być uznana za idola, ponieważ istnieje przynajmniej jedna osoba, która jej nie zna.

Tym sposobem możemy wyznaczyć idealnego kandydata na idola, zadając tylko 0 do $n - 1$ pytań. Zapiszmy ten proces w postaci pseudokodu:

```
n ← 5
znajomość[n][n] ← [
    [ 0, 1, 0, 1, 1 ],
    [ 0, 0, 1, 0, 1 ],
    [ 0, 1, 0, 1, 1 ],
    [ 1, 1, 1, 0, 1 ],
    [ 0, 0, 0, 0, 0 ]
]
kandydat ← 0

dla i = 1, 2, ..., n - 1 wykonuj
    jeżeli znajomość[kandydat][i] to
        kandydat = i
```

Następnie sprawdzimy, czy wyznaczony kandydat może być idolem. W tym celu musimy przepytać wszystkich gości, czy znają wyznaczonego kandydata, a także przepytać kandydata, czy nie zna nikogo. Każdy z tych procesów będzie składał się z zadania $n - 1$ pytań.

Do wcześniejszego pseudokodu dopiszemy linijki odpowiadające za sprawdzenie, czy wyznaczony kandydat jest idolem:

```
n ← 5
znajomość[n][n] ← [
    [ 0, 1, 0, 1, 1 ],
    [ 0, 0, 1, 0, 1 ],
    [ 0, 1, 0, 1, 1 ],
    [ 1, 1, 1, 0, 1 ],
    [ 0, 0, 0, 0, 0 ]
]

kandydat ← 0

dla i = 1, 2, ..., n - 1 wykonuj
    jeżeli znajomość[kandydat][i] to
        kandydat ← i
    // w przeciwnym wypadku kandydat dalej może być idolem

// przepytaj gości, czy znają kandydata
dla i = 0, 1, ..., n - 1 wykonuj
    // sprawdź czy osoba i nie jest kandydatem
    jeżeli kandydat != i to
        // sprawdź, czy osoba i zna kandydata
        jeżeli !znajomość[i][kandydat] to
            // osoba i nie zna kandydata
            // kandydat nie jest idolem
            // idol nie istnieje
            wypisz "Idol nie istnieje"
            zakończ działanie pętli

// przepytaj kandydata, czy nie zna gości
dla i = 0, 1, ..., n - 1 wykonuj
    // sprawdź, czy kandydat nie jest osobą i
    jeżeli kandydat != i to
        // sprawdź, czy kandydat nie zna osoby i
        jeżeli znajomość[kandydat][i] to
            // kandydat zna osobę i
            // kandydat nie jest idolem
            // idol nie istnieje
            wypisz "Idol nie istnieje"
```

```
            zakończ wykonywanie programu

wypisz "Idolem jest osoba numer " + kandydat
```

Powyższy kod można skrócić, wykorzystując podobieństwo dwóch pętli:

```
n ← 5
znajomość[n][n] ← [
    [ 0, 1, 0, 1, 1 ],
    [ 0, 0, 1, 0, 1 ],
    [ 0, 1, 0, 1, 1 ],
    [ 1, 1, 1, 0, 1 ],
    [ 0, 0, 0, 0, 0 ]
]

kandydat ← 0

dla i = 1, 2, ..., n - 1 wykonuj
    jeżeli znajomość[kandydat][i] to
        kandydat ← i
    // w przeciwnym wypadku kandydat dalej może być idolem

// sprawdź, czy kandydat jest idolem
dla i = 0, 1, ..., n - 1 wykonuj
    // sprawdź czy nie pytamy kandydata o samego siebie
    // oraz czy czy osoba i zna kandydata
    // oraz czy kandydat nie zna osoby i
    jeżeli kandydat != i
        oraz !znajomość[i][kandydat]
        oraz znajomość[kandydat][i] to
        // osoba i nie zna kandydata
        // lub kandydat zna osobę i
        // kandydat nie jest idolem
        // idol nie istnieje
        wypisz "Idol nie istnieje"
        zakończ wykonywanie programu

wypisz "Idolem jest osoba numer " + kandydat
```

Przedstawiony algorytm pozwala nam na znalezienie idola po zadaniu $3(n-1)$ pytań.

By to uzasadnić, przypomnijmy jak działa omówiony algorytm. Najpierw sprawdzamy, czy każda kolejna osoba (poza pierwszą) zna obecnego kandydata (czyli pierwszą osobę). Daje to $n-1$ pytań.

Następny krok wymaga sprawdzana, czy kolejne osoby z grupy znają kandydata (czyli znowu zadajemy $n-1$ pytań) oraz czy kandydat zna te kolejne osoby (kolejne $n-1$ pytań).

By znaleźć idola zadajemy $3(n-1)$ pytań. To oznacza, że złożoność czasowa omówionego algorytmu wynosi $O(n)$.

Słownik

idol

taka osoba, którą znają wszyscy, a która nie zna nikogo

lider

wartość w zbiorze n-elementowym, która powtarza się więcej niż $\frac{n}{2}$ razy

wartownik

element umieszczany w zbiorze danych, które mają zostać przeszukane z wykorzystaniem algorytmu wyszukiwania liniowego

wyszukiwanie binarne

algorytm odnajdowania elementów w uporządkowanych zbiorach; polega na wielokrotnym dzieleniu przeszukiwanego zbioru na dwie równe części i porównywaniu wartości szukanej z elementem znajdującym się w środku dzielonego obszaru

wyszukiwanie liniowe

algorytm wyszukiwania elementu w zbiorze danych polegający na porównywaniu poszukiwanego elementu z elementami zbioru danych

złożoność czasowa

ilość czasu potrzebna do wykonania zadania, wyrażona jako funkcja ilości danych

złożoność obliczeniowa

ilość zasobów komputerowych potrzebnych do wykonania zadania; złożoność obliczeniowa obejmuje złożoność pamięciową i czasową

złożoność pamięciowa

ilość pamięci potrzebnej do wykonania zadania, wyrażona jako funkcja ilości danych

Schemat interaktywny

Polecenie 1

Zapisz algorytm, który w podanym n-elementowym zbiorze liczb znajdzie takie wartości, które należą do przedziału $\langle a, b \rangle$. Wykorzystaj **schemat interaktywny**. Przetestuj działanie programu dla zbioru liczb `liczby = {26, 4, 24, 27, 5, 23}` oraz przedziału $\langle 2, 4 \rangle$.

Specyfikacja problemu:

Dane:

- n – liczba naturalna; liczba elementów tablicy `liczby`
- `liczby` – tablica n liczb naturalnych
- a – liczba naturalna; górna granica przedziału
- b – liczba naturalna; dolna granica przedziału

Wynik:

- liczby należące do przedziału $\langle a, b \rangle$

Polecenie 2

Zapisz pseudokod wybranego innego algorytmu wyszukiwania liczby `szukana` w nieposortowanym n-elementowym zbiorze liczb `liczby`. Zastanów się, jak na złożoność czasową algorytmu wpłynie zmiana zbioru nieposortowanego na posortowany.

```
1
```

Polecenie 3

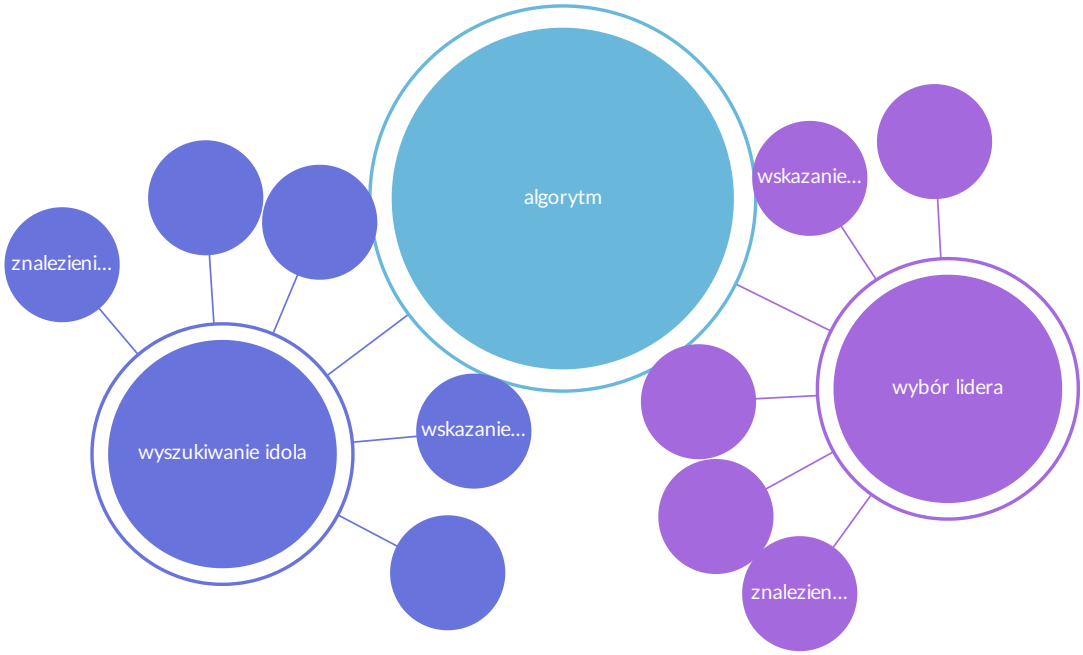
Zmodyfikuj program z **polecenia 1** tak, by liczył wykonywane pętle. Przetestuj jego działanie dla zbioru posortowanego oraz nieposortowanego.

Sprawdź się

Pokaż ćwiczenia:

Ćwiczenie 1

Zapoznaj się z mapą myśli przedstawiającą przykładowe zastosowania algorytmów wyszukiwania **lidera** oraz **idola** w zbiorze. Dopisz własne przykłady wykorzystania wymienionych algorytmów.



Stwórz listę wypunktowanych haseł w odniesieniu do zaproponowanych w powyższej mapie myśli pojęć.

Ćwiczenie 2

Wskaż poprawne zdania opisujące wartownika w algorytmie wyszukiwania liniowego.

- [ ] Wartownik może mieć wartość równą poszukiwanej liczbie.
- [ ] Wartownik może być umieszczony w dowolnym miejscu zbioru.
- [ ] Wartownik może zostać dodany na początku zbioru.
- [ ] Nie możemy wykorzystać wartownika w algorytmie wyszukiwania liniowego.
- [ ] Wartownik musi zostać dodany na początku zbioru.
- [ ] Wartownik zawsze musi być liczbą dodatnią.

Ćwiczenie 3

Zdecyduj, czy zdanie jest prawdziwe:
Algorytm wyszukiwania lidera pozwala na znalezienie w zbiorze elementu, który występuje w nim najczęściej.

- ◯ fałsz
- ◯ prawda

Ćwiczenie 4

Zdecyduj, którego algorytmu użyjesz dla danego zadania.

algorytm wyszukiwania lidera

algorytm wyszukiwania idola

wskazanie kandydata, który na podstawie sondażów *exit poll* ma największe szanse na zwycięstwo w wyborach

określenie, który dworzanin króla ma największe wpływy na podstawie tego, z iloma synami króla wyruszył na polowanie

znalezienie w katalogu sklepu internetowego najpopularniejszych produktów na podstawie liczby sprzedanych egzemplarzy

wskazanie, które miasto ma najwięcej węzłów połączeń na podstawie mapy połączeń

Ćwiczenie 5

Medianą nazywamy taki środkowy element uporządkowanego zbioru składającego się z nieparzystej liczby liczb. Jeśli liczba liczb jest parzysta, mediana jest równa średniej arytmetycznej dwóch środkowych liczb.

Zapisz za pomocą pseudokodu propozycję algorytmu obliczającego wartość mediany dla zbioru, który nie jest posortowany.

Specyfikacja problemu:

Dane:

- `n` – liczba naturalna; liczba elementów zbioru
- `zbiór` – zbiór n liczb całkowitych

Wynik:

- `mediana` – mediana zbioru

```
1
```

Ćwiczenie 6

Dominantą nazywamy wartość, która najczęściej występuje w zbiorze danych.

Zapisz za pomocą pseudokodu propozycję algorytmu obliczającego wartość dominanty zbioru, który jest posortowany.

Specyfikacja problemu:

Dane:

- n – liczba naturalna; liczba elementów zbioru
- `zbiór` – zbiór n liczb całkowitych

Wynik:

- `dominanta` – dominanta zbioru

```
1
```

Ćwiczenie 7

Zapoznaj się z algorytmem wyszukiwania liniowego przeszukującego zbiór n-elementowy zaczynającego od prawej strony (oznacza to, iż wyszukiwanie zaczyna się od elementu o indeksie o jeden mniejszym niż liczba elementów całego zbioru, a kończy na elemencie o indeksie równym 0). Uzupełnij miejsca, w których kod został skasowany i na jego miejscu postawiono symbol X.

Specyfikacja problemu:

Dane:

- n – liczba naturalna; liczba elementów zbioru
- `zbiór` – n-elementowy zbiór liczb całkowitych
- `szukana` – liczba całkowita

Wynik:

- informacja, czy i na jakiej pozycji została odnaleziona liczba `szukana`

```
funkcja wyszukiwanie_liniowe_od_prawej(zbiór, szukana, n):
    znaleziono ← fałsz
    X
    dopóki znaleziono != prawda X wykonuj:
        jeżeli X szukana
            wypisz X
            znaleziono ← prawda
            zakończ działanie algorytmu
        i ← X
    X
        X
```

```
1
```

Ćwiczenie 8

Za pomocą schematu interaktywnego zapisz algorytm równoczesnego wyszukiwania wartości minimalnej i maksymalnej w zbiorze liczb.

Działanie programu przetestuj dla zbioru {27, 5, 23, 26, 4, 24}.

Specyfikacja problemu:

Dane:

- `n` – liczba naturalna; liczba elementów zbioru
- `zbior` – `n`-elementowy zbiór liczb całkowitych

Wynik:

- wartości `min` oraz `max`

Dla nauczyciela

Autor: zespół autorski Contentplus.pl sp. z o.o.
Przedmiot: Informatyka

Temat: Znajdowanie określonego elementu w zbiorze

Grupa docelowa:

Liceum ogólnokształcące i technikum, liceum ogólnokształcące, technikum, zakres podstawowy i rozszerzony

Podstawa programowa:

Cele kształcenia – wymagania ogólne

I. Rozumienie, analizowanie i rozwiązywanie problemów na bazie logicznego i abstrakcyjnego myślenia, myślenia algorytmicznego i sposobów reprezentowania informacji.

II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera oraz innych urządzeń cyfrowych: układanie i programowanie algorytmów, organizowanie, wyszukiwanie i udostępnianie informacji, posługiwanie się aplikacjami komputerowymi.

Treści nauczania – wymagania szczegółowe

I. Rozumienie, analizowanie i rozwiązywanie problemów.

Zakres podstawowy. Uczeń:

1) planuje kolejne kroki rozwiązywania problemu, z uwzględnieniem podstawowych etapów myślenia komputacyjnego (określenie problemu, definicja modeli i pojęć, znalezienie rozwiązania, zaprogramowanie i testowanie rozwiązania).

3) wyróżnia w problemie podproblemy i charakteryzuje: metodę połowienia, stosuje podejście zachłanne i rekurencję;

4) porównuje działanie różnych algorytmów dla wybranego problemu, analizuje algorytmy na podstawie ich gotowych implementacji;

I + II. Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

1) zapisuje za pomocą listy kroków, schematu blokowego lub pseudokodu, i implementuje w wybranym języku programowania, algorytmy poznane na

wcześniejszych etapach oraz algorytmy:

b) znajdowania określonego elementu w zbiorze: lidera, idola, elementu w zbiorze uporządkowanym metodą binarnego wyszukiwania,

3) objaśnia, a także porównuje podstawowe metody i techniki algorytmiczne oraz struktury danych, wykorzystując przy tym przykłady problemów i algorytmów, w szczególności:

a) wyszukiwanie elementów liniowe i przez połowienie (do znajdowania elementów w zbiorze, sortowania przez wstawianie, przybliżonego rozwiązywania równań, sprawdzania przynależności punktu do wielokąta wypukłego),

Kształtowane kompetencje kluczowe:

- kompetencje cyfrowe;
- kompetencje osobiste, społeczne i w zakresie umiejętności uczenia się;
- kompetencje matematyczne oraz kompetencje w zakresie nauk przyrodniczych, technologii i inżynierii.

Cele operacyjne (językiem ucznia):

- Zapoznasz się z iteracyjną wersją algorytmu wyszukiwania binarnego.
- Wyjaśnisz, czym są lider oraz idol.
- Przeanalizujesz algorytmy wyboru lidera oraz wyszukiwania idola w zbiorze.
- Rozwiążesz zadania wymagające wykorzystania algorytmów wyszukiwania określonego elementu w zbiorze.

Strategie nauczania:

- konstruktywizm;
- konektywizm.

Metody i techniki nauczania:

- dyskusja;
- rozmowa nauczająca z wykorzystaniem multimedium i ćwiczeń interaktywnych;
- burza mózgów;
- ćwiczenia praktyczne.

Formy pracy:

- praca indywidualna;
- praca w parach;
- praca w grupach;
- praca całego zespołu klasowego.

Środki dydaktyczne:

- komputery z głośnikami, słuchawkami i dostępem do internetu;
- zasoby multimedialne zawarte w e-materiale;
- tablica interaktywna/tablica, pisak/kreda.

Przebieg lekcji

Przed lekcją:

1. **Przygotowanie do zajęć.** Nauczyciel loguje się na platformie i udostępnia e-materiał: „Znajdowanie określonego elementu w zbiorze". Nauczyciel prosi uczniów o zapoznanie się z treściami w sekcji „Przeczytaj".

Faza wstępna:

1. Wyświetlenie przez nauczyciela tematu i celów zajęć, przejście do wspólnego ustalenia kryteriów sukcesu.
2. **Rozpoznanie wiedzy uczniów.** Nauczyciel prosi wybranego ucznia lub uczniów o przedstawienie sytuacji problemowej związanej z tematem lekcji.

Faza realizacyjna:

1. **Praca z tekstem.** Uczniowie przystępują do cichego czytania tekstu e-materiału. Indywidualnie zapoznają się z treścią w sekcji „Przeczytaj". W razie potrzeby nauczyciel odpowiada na pytania uczniów i rozwiewa ich wątpliwości.
2. **Praca z multimedium.** Nauczyciel wyświetla zawartość sekcji „Schemat interaktywny". Uczniowie wspólnie wykonują polecenie nr 1 oraz 2.
3. **Ćwiczenie umiejętności.** Nauczyciel wyświetla zawartość sekcji „Sprawdź się". Metodą burzy mózgów uczniowie na forum klasy rozwiązują ćwiczenie 1 i szukają innych przykładów wykorzystania algorytmów wyboru lidera i wyszukiwania idola. Chętna lub wybrana osoba uzupełnia mapę myśli na tablicy.
4. Uczniowie w parach wykonują ćwiczenia 2–5 z sekcji „Sprawdź się". W razie potrzeby nauczyciel odpowiada na ewentualne pytania.

Faza podsumowująca:

1. Wybrany uczeń podsumowuje zajęcia, zwracając uwagę na nabyte umiejętności.
2. Nauczyciel wyświetla na tablicy temat lekcji i cele zawarte w sekcji „Wprowadzenie". W kontekście ich realizacji podsumowuje przebieg zajęć, a także wskazuje mocne i słabe strony pracy uczniów.

Praca domowa:

1. Uczniowie wykonują ćwiczenia nr 6–8 z sekcji „Sprawdź się".
2. Uczniowie wykonują polecenie nr 3 z sekcji „Schemat interaktywny".

Wskazówki metodyczne:

- Treści w sekcji „Przeczytaj” można wykorzystać jako podsumowanie i utrwalenie wiedzy uczniów.