Algorytmy wyszukujące - wyszukiwanie binarne

Algorytm

Jeśli zbiór jest posortowany (np. rosnąco), to problem wyszukiwania elementu da się rozwiązać znacznie efektywniej stosując poniższą metodę zwaną wyszukiwaniem binarnym:

> Jeśli zbiór jest pusty, to kończymy algorytm z wynikiem negatywnym. W przeciwnym razie wyznaczamy element leżący w środku zbioru. Porównujemy poszukiwany element z elementem środkowym. Jeśli są sobie równe, to zadanie wyszukania elementu jest wypełnione i kończymy algorytm. W przeciwnym razie element środkowy dzieli zbiór na dwie partycje - lewą z elementami mniejszymi od środkowego oraz prawą z elementami większymi. Jeśli porównywany element jest mniejszy od środkowego elementu zbioru, to za nowy zbiór poszukiwań przyjmujemy lewą partycję. W przeciwnym razie za nowy zbiór przyjmujemy prawą partycję. W obu przypadkach rozpoczynamy poszukiwania od początku, ale w nowo wyznaczonym zbiorze.

Przykład

Dla zobrazowania algorytmu wyszukiwania binarnego znajdziemy element 2 w zbiorze:

{ 1 1 2 4 4 5 6 6 6 7 7 8 8 9 9 }

Lp.	Operacja	Opis
1.	1 1 2 4 4 5 6 **6** 6 7 7 8 8 9 9	Znajdujemy środkowy element zbioru.
2.	> > > > > > > 2 1 1 2 4 4 5 6 **6** 6 7 7 8 8 9 9	Element środkowy porównujemy z poszukiwanym elementem. Nie ma równości. Za nowy zbiór poszukiwań wybieramy lewą partycję.
3.	1 1 2 **4** 4 5 6 6 6 7 7 8 8 9 9	Znajdujemy środkowy element zbioru.

4.		Element środkowy porównujemy z poszukiwanym. Znów nie ma równości. Wybieramy lewą partycję.
5.	1 **1** 2 4 4 5 6 6 6 7 7 8 8 9 9	Znajdujemy element środkowy.
6.		Porównujemy element środkowy z poszukiwanym. Brak równości. Wybieramy prawą partycję - należy do niej tylko jeden element.
7.	1 1 **2** 4 4 5 6 6 6 7 7 8 8 9 9	Znajdujemy element środkowy.
8.		Porównujemy elementy - jest zgodność. Poszukiwany element został znaleziony w zborze.

Każde porównanie sprowadza nam problem do zbioru o połowę mniejszego. W najgorszym przypadku wykonamy $\log_2 n + 1$ porównań. Zatem przedstawiony sposób wyszukiwania elementu zbioru posiada logarytmiczną klasę złożoności obliczeniowej O(log n). Czy to dużo, czy mało? Bardzo mało, logarytm rośnie wolno. Na przykład w zbiorze zawierającym 1.000.000.000 elementów algorytm wyszukiwania binarnego wykona maksymalnie 30 porównań, podczas gdy algorytm wyszukiwania liniowego będzie średnio musiał wykonać 500.000.000 porównań. Wnioski wyciąg sam.

> Gdyby w punkcie 8 powyższego przykładu porównanie dało wynik negatywny, to jednoelementowego zbioru nie można już podzielić na dalsze partycje. Innymi słowy otrzymalibyśmy partycje puste. Zatem zgodnie z opisem algorytmu skończylibyśmy jego wykonanie z wynikiem negatywnym - poszukiwanego elementu nie byłoby w zbiorze
>
> ---
>
> Zwróć uwagę, iż algorytm wyszukiwania binarnego nie gwarantuje znalezienia pierwszego wystąpienia elementu w zbiorze, jeśli element poszukiwany występuje w nim kilkakrotnie.

Pozostaje wyjaśnienie wyznaczenia elementu środkowego oraz podziału na partycje. Otóż załóżmy, iż zmienna i_p zawiera początkowy indeks elementu w

partycji, a i_k zawiera odpowiednio indeks końcowy. Na początku pracy algorytmu i_p ustawiamy na 1, a i_k na n. W ten sposób obejmiemy cały zbiór.

d_1	d_2	d_3	...	d_{n-1}	**d_n**
i_p					**i_k**

Element środkowy wyznaczamy jako całkowitą średnią arytmetyczną indeksów pierwszego i ostatniego elementu w tablicy:

$$i_{sr} = \left[\frac{i_p + i_k}{2} \right]$$

d_1	d_2	d_3	...	d_{sr-1}	**d_{sr}**	d_{sr+1}	...	d_{n-1}	**d_n**
i_p					**i_{sr}**				**i_k**

Jeśli zbiór jest uporządkowany, to jego elementy spełniają warunek:

$$d_1 \leq d_2 \leq d_3 \leq \ldots \leq d_{sr-1} \leq d_{sr} \leq d_{sr+1} \leq \ldots \leq d_{n-1} \leq d_n$$

W lewej partycji są wszystkie elementy o wartościach nie większych od elementu środkowego d_{sr}. Lewa partycja obejmuje zatem elementy o indeksach od i_p do i_{s-1}.

W prawej partycji są wszystkie elementy o wartościach nie mniejszych od elementu środkowego d_{sr}. Prawa partycja obejmuje elementy o indeksach od i_{sr+1} do i_k.

Umówmy się, iż partycja będzie pusta, gdy indeks jej początku jest większy od indeksu końca:

$$i_p > i_k$$

Tak określona partycja nie może zawierać żadnego elementu.

Specyfikacja problemu

Dane wejściowe

n - ilość elementów w przeszukiwanym zbiorze. $n \in \mathbf{N}$
d[ ] - posortowany rosnąco zbiór danych. Indeksy elementów rozpoczynają się od 1.
w - wartość poszukiwanego elementu. Typ elementu taki sam jak typ elementów zbioru.

Dane wyjściowe

p - pozycja elementu w zbiorze, $p \in \mathbf{C}$, $1 \leq p \leq n$ - element znaleziony, $p = 0$ - element nie znaleziony

Zmienne pomocnicze

i_p - zawiera indeks pierwszego elementu w partycji. $i_p \in \mathbf{C}$

i_k - zawiera indeks ostatniego elementu w partycji, $i_k \in \mathbf{C}$

i_{sr} - zawiera indeks środkowego elementu, $i_{sr} \in \mathbf{C}$

Lista kroków

krok 1: $i_p \leftarrow 1$; $i_k \leftarrow n$; $p \leftarrow 0$

krok 2: Dopóki $i_p \leq i_k$: wykonuj kroki 3...5

krok 3: $i_{sr} = \left[\frac{i_p + i_k}{2}\right]$

krok 4: Jeśli $w = d[i_{sr}]$, to $p \leftarrow i_{sr}$ i zakończ algorytm

krok 5: Jeśli $w > d[i_{sr}]$, to $i_p \leftarrow i_{sr} + 1$. Inaczej $i_k \leftarrow i_{sr} - 1$

krok 6: Zakończ algorytm

Schemat blokowy

Na początku pracy algorytmu ustawiamy zmienne robocze. Zmienna i_p przechowuje pierwszy element partycji, w której wykonujemy oszukiwanie elementu w. Zmienne i_k przechowuje ostatni indeks elementu w tej partycji. Początkowo partycja obejmuje cały zbiór.
W zmiennej p algorytm umieści indeks elementu zbioru równego poszukiwanemu elementowi. Wstępnie ustawiamy ten indeks na 0 w celu zaznaczenia, iż element nie został jeszcze znaleziony.
Rozpoczynamy pętlę warunkową. Warunkiem kontynuacji jest niepusta partycja, w której poszukuje się elementu w. Pierwszą czynnością w pętli jest wyliczenie indeksu elementu środkowego i_{sr}. Następnie sprawdzamy, czy element poszukiwany jest równy elementowi środkowemu. Jeśli tak, to w zmiennej p umieszczamy indeks elementu środkowego i algorytm kończymy.
W przeciwnym razie poszukiwany element w może leżeć w lewej partycji jeśli jest mniejszy od elementu środkowego lub w prawej partycji w przypadku

przeciwnym. Ustawiamy zatem odpowiednio zmienne i_p oraz i_k:

w > d[i_{sr}] - prawa partycja, i_p przesuwamy na i_s + 1, i_k pozostaje bez zmian

w $\leq$ d[i_{sr}] - lewa partycja, i_p pozostaje bez zmian, i_k przesuwamy na i_{sr} - 1.

Po tej operacji wracamy na początek pętli.

Gdy pętla warunkowa się zakończy w sposób naturalny (w wyniku ostatniego podziału otrzymujemy pustą partycję), zmienna p zawiera 0. Oznacza to, iż element w nie występuje w zbiorze. Jest to przypadek pesymistyczny i wymaga wykonania $\log_2 n + 1$ obiegów pętli.

Programy

Poniższe, przykładowe programy są praktyczną realizacją omawianego w tym rozdziale algorytmu. Zapewne można je napisać bardziej efektywnie. To już twoje zadanie. Dokładny opis stosowanych środowisk programowania znajdziesz we wstępie. Programy przed opublikowaniem w serwisie edukacyjnym zostały dokładnie przetestowane. Jeśli jednak znajdziesz jakąś usterkę (co zawsze może się zdarzyć), to prześlij o niej informację do autora. Pozwoli to ulepszyć nasze artykuły. Będziemy Ci za to wdzięczni.

Program generuje pseudolosowy zbiór uporządkowany o 100 elementach z zakresu od 0 do 99. Następnie wybiera losowo wartość z tego samego zakresu i poszukuje jej w zbiorze za pomocą algorytmu wyszukiwania binarnego.

Efekt uruchomienia programu

```
   Wyszukiwanie binarne
--------------------------
(C)2006 mgr Jerzy Wałaszek

  0   1   2   3   4   5   5   5   6   7   7   8   9   9  10  10  11  12  12  12
 13  16  18  20  23  24  24  25  26  26  29  30  30  32  33  33  34  34  37  38
 38  41  44  45  49  50  50  53  54  54  55  57  58  58  59  59  60  61  61  63
 63  65  66  67  67  68  68  69  72 (73) 74  74  75  75  75  75  76  76  76  76
 77  77  77  79  79  80  83  83  85  85  86  88  89  90  90  90  95  95  95  97


Element w = 73 jest na pozycji nr 70

KONIEC. Naciśnij dowolny klawisz...
```

Program w Borland Delphi

```
//    Wyszukiwanie binarne
//----------------------------
// (C)2006 mgr Jerzy Wałaszek
//      I LO w Tarnowie
//----------------------------

program binsearch;

{$APPTYPE CONSOLE}

const
  N = 100;  // Liczba elementów w zbiorze
  M =  99;  // Maksymalna wartość elementu

var
  d               : array[1..N] of integer;
  w,i,ip,ik,isr,p : integer;

begin
  writeln('   Wyszukiwanie binarne   ');
  writeln('--------------------------');
  writeln('(C)2006 mgr Jerzy Walaszek');
  writeln;

// Generujemy zbiór pseudolosowy

  randomize;
  for i := 1 to N do d[i] := random(M + 1);

// Zbiór sortujemy rosnąco

  for i := N - 1 downto 1 do
  begin
    w := d[i]; ip := i + 1;
    while (ip <= N) and (w > d[ip]) do
    begin
      d[ip - 1] := d[ip]; inc(ip);
    end;
```

```
    d[ip - 1] := w;
  end;

// Generujemy poszukiwany element

  w := random(M + 1);

// Szukamy elementu w

  ip := 1; ik := N; p := 0;
  while ip <= ik do
  begin
    isr := (ip + ik) div 2;
    if w = d[isr] then
    begin
      p := isr; break;
    end
    else if w > d[isr] then
      ip := isr + 1
    else
      ik := isr - 1;
  end;

// Prezentujemy wyniki

  for i := 1 to N do
    if i = p then write('(',d[i]:2,')') else write(d[i]:3,' ');
  writeln; writeln;
  write('Element w = ',w);
  if p > 0 then
    writeln(' jest na pozycji nr ',p)
  else
    writeln(' w zbiorze nie wystepuje.');
  writeln;

// Gotowe

  writeln('Nacisnij klawisz Enter...');
  readln;
end.
```

```
//    Wyszukiwanie binarne
//----------------------------
// (C)2006 mgr Jerzy Wałaszek
//      I LO w Tarnowie
//----------------------------

#include <iostream>
#include <iomanip>

using namespace std;

const int N = 100;  // Liczba elementów w zbiorze
const int M =  99;  // Maksymalna wartość elementu

int main(void)
{
  int d[N + 1],w,i,ip,ik,isr,p;

  cout << "   Wyszukiwanie binarne   \n"
          "--------------------------\n"
          "(C)2006 mgr Jerzy Walaszek\n\n";

// Generujemy zbiór pseudolosowy

  srand((unsigned)time(NULL));
  for(i = 1;i <= N; i++) d[i] = rand() % (M + 1);

// Zbiór sortujemy rosnąco

  for(i = N - 1; i >= 1; i--)
  {
    w = d[i]; ip = i + 1;
    while((ip <= N) && (w > d[ip]))
    {
      d[ip - 1] = d[ip]; ip++;
    }
    d[ip - 1] = w;
  }
```

```
// Generujemy poszukiwany element

  w = rand() % (M + 1);

// Szukamy elementu w

  ip = 1; ik = N; p = 0;
  while(ip <= ik)
  {
    isr = (ip + ik) / 2;
    if(w == d[isr])
    {
      p = isr; break;
    }
    else if(w > d[isr])
      ip = isr + 1;
    else
      ik = isr - 1;
  }

// Prezentujemy wyniki

  for(i = 1; i <= N; i++)
    if(i == p)
      cout << "(" << setw(2) << d[i] << ")";
    else
      cout << setw(3) << d[i] << " ";
  cout << "\nElement w = " << w;
  if(p)
    cout << " jest na pozycji nr " << p << endl;
  else
    cout << " w zbiorze nie wystepuje.\n";
  cout << endl;

// Gotowe

  system("pause"); return 0;
}
```

```
'    Wyszukiwanie binarne
'----------------------------
' (C)2006 mgr Jerzy Wałaszek
'      I LO w Tarnowie
'----------------------------

Module Module1

  Sub Main()
    Const N = 100  ' Liczba elementów w zbiorze
    Const M = 99  ' Maksymalna wartość elementu

    Dim d(N), w, i, ip, ik, isr, p As Integer

    Console.WriteLine("   Wyszukiwanie binarne   ")
    Console.WriteLine("--------------------------")
    Console.WriteLine("(C)2006 mgr Jerzy Wałaszek")
    Console.WriteLine()

    ' Generujemy zbiór pseudolosowy

    Randomize()
    For i = 1 To N : d(i) = Int(Rnd(1) * (M + 1)) : Next

    ' Zbiór sortujemy rosnąco

    For i = N - 1 To 1 Step -1
      w = d(i) : ip = i + 1
      While (ip <= N) AndAlso (w > d(ip))
        d(ip - 1) = d(ip) : ip += 1
      End While
      d(ip - 1) = w
    Next

    ' Generujemy poszukiwany element

    w = Int(Rnd(1) * (M + 1))

    ' Szukamy elementu w

    ip = 1 : ik = N : p = 0
    While ip <= ik
```

```
      isr = (ip + ik) \ 2
      If w = d(isr) Then
        p = isr : Exit While
      ElseIf w > d(isr) Then
        ip = isr + 1
      Else
        ik = isr - 1
      End If
    End While

    ' Prezentujemy wyniki

    For i = 1 To N
      If i = p Then
        Console.Write("({0,2})", d(i))
      Else
        Console.Write(" {0,2} ", d(i))
      End If
    Next
    Console.WriteLine()
    Console.WriteLine()
    Console.Write("Element w = {0}", w)
    If p > 0 Then
      Console.WriteLine(" jest na pozycji nr {0}", p)
    Else
      Console.WriteLine(" w zbiorze nie występuje.")
    End If

    ' Gotowe

    Console.WriteLine()
    Console.WriteLine("KONIEC. Naciśnij dowolny klawisz...")
    Console.ReadLine()

  End Sub

End Module
```

```
# -*- coding: cp1250 -*-
#    Wyszukiwanie binarne
#----------------------------
# (C)2006 mgr Jerzy Wałaszek
#      I LO w Tarnowie
#----------------------------

import random

N = 100  # Liczba elementów w zbiorze
M =  99  # Maksymalna wartość elementu

d = []

print "   Wyszukiwanie binarne   "
print "--------------------------"
print "(C)2006 mgr Jerzy Walaszek"
print

# Generujemy zbiór pseudolosowy

for i in range(N): d.append(random.randint(0, M))

# Zbiór sortujemy rosnąco

d.sort()

# Generujemy poszukiwany element

w = random.randint(0, M)

# Szukamy elementu w

ip, ik, p = 0, N - 1,  -1
while ip <= ik:
  isr = (ip + ik) / 2
  if w == d[isr]:
    p = isr; break
  elif w > d[isr]:
    ip = isr + 1
  else:
    ik = isr - 1
```

```
# Prezentujemy wyniki

for i in range(N):
  if i == p:
    print "(%2d)" % d[i],
  else:
    print " %2d " % d[i],
print
print
print "Element w =", w,
if p >= 0:
  print "jest na pozycji nr", p + 1
else:
  print "w zbiorze nie wystepuje."
print

# Gotowe

raw_input("Nacisnij klawisz Enter...")
```


```
<html>
  <head>
  </head>
  <body>
    <div align="center">
      <form style="BORDER-RIGHT: #ff9933 1px outset; PADDING-RIGHT: 4px;
                   BORDER-TOP: #ff9933 1px outset; PADDING-LEFT: 4px;
                   PADDING-BOTTOM: 1px; BORDER-LEFT: #ff9933 1px outset;
                   PADDING-TOP: 1px; BORDER-BOTTOM: #ff9933 1px outset;
                   BACKGROUND-COLOR: #ffcc66" name="frmBinSearch">
        <h4 id="out_t0" style="text-align: center">Wyszukiwanie binarne</h4>
        <p style="TEXT-ALIGN: center">
          (C)2006 mgr Jerzy Wałaszek<br>
          I LO w Tarnowie
        </p>
        <hr>
        <p style="TEXT-ALIGN: center">
          <input type="button" value="Szukaj" name="B1" onclick="main();">
        </p>
        <p style="TEXT-ALIGN: center" id="t_out">...</p>
      </form>
```

```
<script language=javascript>

// Wyszukiwanie binarne
//----------------------------
// (C)2006 mgr Jerzy Wałaszek
// I LO w Tarnowie
//----------------------------

function main()
{
  var N = 100; // Liczba elementów w zbiorze
  var M =  99; // Maksymalna wartość elementu
  var d = new Array(N + 1);
  var w,i,ip,ik,isr,p,l,t;

// Generujemy zbiór pseudolosowy

  for(i = 1; i <= N; i++) d[i] = Math.floor(Math.random() * (M + 1));

// Zbiór sortujemy rosnąco

  for(i = N - 1; i >= 1; i--)
  {
    w = d[i]; ip = i + 1;
    while((ip <= N) && (w > d[ip]))
    {
      d[ip - 1] = d[ip]; ip++;
    }
    d[ip - 1] = w;
  }

// Generujemy poszukiwany element

  w = Math.floor(Math.random() * (M + 1));

// Szukamy elementu w

  ip = 1; ik = N; p = l = 0;
  while(ip <= ik)
  {
    isr = Math.floor((ip + ik) / 2);
    l++;
    if(w == d[isr])
    {
      p = isr; break;
    }
    else if(w > d[isr])
      ip = isr + 1;
```

```
    else
    ik = isr - 1;
  }

// Prezentujemy wyniki

  t = "";
  for(i = 1; i <= N; i++)
  if(i == p)
    t += " <b><font color=red>(" + d[i] + ")</font></b>";
  else
    t += "  " + d[i] + " ";
  t += "<br><br>Ilość prób = " + l + "<br><br>Element w = " + w;
  if(p > 0)
    t += " jest na pozycji nr " + p;
  else
    t += " w zbiorze nie wystepuje";

// Gotowe

  document.getElementById("t_out").innerHTML = t;
}

</script>

    </div>
  </body>
</html>
```


Źródło: [mgr Jerzy Wałaszek]