Algorytmy Sortujące

Prezentowane materiały są przeznaczone dla uczniów szkół ponadgimnazjalnych.

Autor artykułu: mgr Jerzy Wałaszek, Wersja 4.1

Sortowanie stogowe
Heap Sort

Podrozdziały

Tematy pokrewne

Algorytm

Jeśli przeczytałeś uważnie poprzednie dwa rozdziały, to zasadę sortowania przez kopcowanie zrozumiesz od razu - w zbiorze tworzymy kopiec, a następnie dokonujemy jego rozbioru. W wyniku wykonania tych dwóch operacji zbiór zostanie posortowany.

Specyfikacja problemu

Dane wejściowe

d[ ] - Zbiór zawierający elementy do posortowania, które są numerowane począwszy od 1.

n - Ilość elementów w zbiorze, *n* Î N

Dane wyjściowe

d[ ] - Zbiór zawierający elementy posortowane rosnąco

Zmienne pomocnicze i funkcje

Twórz_kopiec - procedura budująca kopiec z elementów zbioru d[ ]. Kopiec powstaje w zbiorze d[ ].

Rozbierz_kopiec - procedura dokonująca rozbioru kopca utworzonego w zbiorze d[ ]. Wynik rozbioru trafia z powrotem do zbioru d[ ].

Lista kroków

K01: **Twórz_kopiec**

K02: **Rozbierz_kopiec**

K03: Zakończ algorytm

Ponieważ sortowanie przez kopcowanie składa się z dwóch następujących bezpośrednio po sobie operacji o klasie czasowej złożoności obliczeniowej $O(n \log n)$, to dla całego algorytmu klasa złożoności również będzie wynosić $O(n \log n)$.

Programy

Efekt uruchomienia programu

```
 Sortowanie przez kopcowanie
-----------------------------
   (C)2005  Jerzy Walaszek

Przed sortowaniem:

  48  40  48  29  21  11  12  63  77  89  99  97  80  62  64  32  28   5  16  36

Po sortowaniu:

   5  11  12  16  21  28  29  32  36  40  48  48  62  63  64  77  80  89  97  99
```

DevPascal

```
// Sortowanie przez kopcowanie
//------------------------------------------------------------
// (C)2012 I LO w Tarnowie
// I Liceum Ogólnokształcące
// im. K. Brodzińskiego
// w Tarnowie
//------------------------------------------------------------

program heap_sort;

const N = 20;  // liczebność zbioru

var
  d         : array[1..N] of integer;
  i,j,k,m,x : integer;

begin
  writeln(' Sortowanie przez kopcowanie ');
  writeln('-----------------------------');
  writeln('   (C)2005  Jerzy Walaszek');
  writeln;
  writeln('Przed sortowaniem:'); writeln;

// Wypełniamy tablicę liczbami pseudolosowymi i wyświetlamy je

  randomize;
  for i := 1 to N do
  begin
    d[i] := random(100); write(d[i] : 4);
  end;

// Budujemy kopiec

  for i := 2 to N do
  begin
    j := i; k := j div 2;
    x := d[i];
    while (k > 0) and (d[k] < x) do
    begin
      d[j] := d[k];
      j := k; k := j div 2;
```

```
    end;
    d[j] := x;
  end;

// Rozbieramy kopiec

  for i := N downto 2 do
  begin
    x := d[1]; d[1] := d[i]; d[i] := x;
    j := 1; k := 2;
    while k < i do
    begin
      if (k + 1 < i) and (d[k + 1] > d[k]) then
        m := k + 1
      else
        m := k;
      if d[m] <= d[j] then break;
      x := d[j]; d[j] := d[m]; d[m] := x;
      j := m; k := j + j;
    end;
  end;

// Wyświetlamy wynik sortowania

  writeln('Po sortowaniu:'); writeln;
  for i := 1 to N do write(d[i] : 4);
  writeln;
  writeln('Nacisnij Enter...');
  readln;
end.
```

Code::Blocks

```
// Sortowanie przez kopcowanie
//------------------------------------------------------------
// (C)2012 I LO w Tarnowie
// I Liceum Ogólnokształcące
// im. K. Brodzińskiego
// w Tarnowie
//------------------------------------------------------------

#include <iostream>
#include <iomanip>
#include <cstdlib>
#include <time.h>

using namespace std;

int main()
{
  const int N = 20;  // liczebność zbioru
  int d[N + 1],i,j,k,m,x;

  srand((unsigned)time(NULL));

  cout << " Sortowanie przez kopcowanie \n"
          "-----------------------------\n"
          "   (C)2005  Jerzy Walaszek\n\n"
          "Przed sortowaniem:\n\n";

// Wypełniamy tablicę liczbami pseudolosowymi i wyświetlamy je

  for(i = 1; i <= N; i++)
  {
    d[i] = rand() % 100; cout << setw(4) << d[i];
```

```
  }
  cout << endl;

// Budujemy kopiec

  for(i = 2; i <= N; i++)
  {
    j = i; k = j / 2;
    x = d[i];
    while((k > 0) && (d[k] < x))
    {
      d[j] = d[k];
      j = k; k = j / 2;
    }
    d[j] = x;
  }

// Rozbieramy kopiec

  for(i = N; i > 1; i--)
  {
    swap(d[1], d[i]);
    j = 1; k = 2;
    while(k < i)
    {
      if((k + 1 < i) && (d[k + 1] > d[k]))
        m = k + 1;
      else
        m = k;
      if(d[m] <= d[j]) break;
      swap(d[j], d[m]);
      j = m; k = j + j;
    }
  }

// Wyświetlamy wynik sortowania

  cout << "Po sortowaniu:\n\n";
  for(i = 1; i <= N; i++) cout << setw(4) << d[i];
  cout << endl;
  return 0;
}
```

Free Basic

```
' Sortowanie przez kopcowanie
'------------------------------------------------------------
' (C)2012 I LO w Tarnowie
' I Liceum Ogólnokształcące
' im. K. Brodzińskiego
' w Tarnowie
'------------------------------------------------------------

Const N = 20 ' liczebność zbioru

Dim d(N) As Integer, i As Integer, j As Integer
Dim k As Integer, m As Integer, x As Integer

Print " Sortowanie przez kopcowanie"
Print "----------------------------"
Print "   (C)2005  Jerzy Walaszek"
Print
Print "Przed sortowaniem:": Print

' Wypełniamy tablicę liczbami pseudolosowymi i wyświetlamy je
```

```
Randomize
For i = 1 To N
  d(i) = Int(Rnd * 100): Print Using "####";d(i);
Next
Print

' Budujemy kopiec

For i = 2 To N
  j = i: k = j \ 2
  x = d(i)
  While (k > 0) And (d(k) < x)
    d(j) = d(k)
    j = k: k = j / 2
  Wend
  d(j) = x
Next

' Rozbieramy kopiec

For i = N To 2 Step -1
  Swap d(1), d(i)
  j = 1: k = 2
  While k < i
    If (k + 1 < i) And (d(k + 1) > d(k)) Then
      m = k + 1
    Else
      m = k
    End If
    If d(m) <= d(j) Then Exit While
    Swap d(j), d(m)
    j = m: k = j + j
  Wend
Next

' Wyświetlamy wynik sortowania

Print "Po sortowaniu:": Print
For i = 1 To N: Print Using "####";d(i);: Next
Print
Print "Nacisnij Enter..."
Sleep
End
```

JavaScript

```
<html>
  <head>
  </head>
  <body>
    <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="frmheapsort">
      <h3 style="text-align: center">Sortowanie Przez Kopcowanie</h3>
      <p style="TEXT-ALIGN: center">
        (C)2012 I LO w Tarnowie - I LO w Tarnowie
      </p>
      <hr>
      <p style="TEXT-ALIGN: center">
        <input onclick="main()" type="button" value="Sortuj" name="B1">
      </p>
```

```
      <p id="t_out" style="TEXT-ALIGN: center">...</p>
    </form>

<script language=javascript>

// Sortowanie Przez Kopcowanie
//---------------------------------------------------------
// (C)2012 I LO w Tarnowie
// I Liceum Ogólnokształcące
// im. K. Brodzińskiego
// w Tarnowie
//---------------------------------------------------------

function main()
{
  var N = 20; // liczba elementów
  var d = new Array(N + 1);
  var i,j,k,x,t;

  // Najpierw wypełniamy tablicę d[] liczbami pseudolosowymi

  for(i = 1; i <= N; i++) d[i] = Math.floor(Math.random() * 100);
  t = "Przed sortowaniem:<BR><BR>";
  for(i = 1; i <= N; i++) t += d[i] + " ";
  t += "<BR><BR>";

// Budujemy kopiec

  for(i = 2; i <= N; i++)
  {
    j = i; k = Math.floor(j / 2);
    x = d[i];
    while((k > 0) && (d[k] < x))
    {
      d[j] = d[k];
      j = k; k = Math.floor(j / 2);
    }
    d[j] = x;
  }

// Rozbieramy kopiec

  for(i = N; i > 1; i--)
  {
    x = d[1]; d[1] = d[i]; d[i] = x;
    j = 1; k = 2;
    while(k < i)
    {
      if((k + 1 < i) && (d[k + 1] > d[k]))
        m = k + 1;
      else
        m = k;
      if(d[m] <= d[j]) break;
      x = d[j]; d[j] = d[m]; d[m] = x;
      j = m; k = j + j;
    }
  }

  // Wyświetlamy wynik sortowania

  t += "Po sortowaniu:<BR><BR>";
  for(i = 1; i <= N; i++) t += d[i] + " ";
  document.getElementById("t_out").innerHTML = t;
}
```

```
</script>

  </body>
</html>
```

Tutaj możesz przetestować działanie prezentowanego skryptu:

Sortowanie Przez Kopcowanie

(C)2012 I LO w Tarnowie - I LO w Tarnowie

Sortuj

...

Badania algorytmów sortowania

W celach badawczych testujemy czas wykonania algorytmu sortowania przez kopcowanie w środowisku opisanym we wstępie. Program testujący jest następujący:

DevPascal

```
// Program testujący czas sortowania dla
// danego algorytmu sortującego
//---------------------------------------
// (C)2012 I LO w Tarnowie
// I Liceum Ogólnokształcące
// w Tarnowie
//---------------------------------------

program TestCzasuSortowania;

uses Windows;

const
  NAZWA = 'Sortowanie przez kopcowanie';
  K1    = '-------------------------------------------------------------';
  K2    = '(C)2011/2012 I Liceum Ogolnoksztalcace  w Tarnowie';
  K3    = '------n---------tpo---------tod---------tpp---------tpk---------tnp';
  K4    = '-------------------------------------------------------------------------';
  MAX_LN = 8; // określa ostatnie LN
  LN : array[1..8] of integer = (1000,2000,4000,8000,16000,32000,64000,128000);

var
  d         : array[1..128000] of real; // sortowana tablica
  n         : integer;                  // liczba elementów
  qpf,tqpc  : int64;                    // dane dla pomiaru czasu
  qpc1,qpc2 : int64;

// Tutaj umieszczamy procedurę sortującą tablicę d
//-------------------------------------------------------

procedure HeapSort;
var
  i,j,k,m : integer;
  x       : real;
```

```
begin

// Budujemy kopiec

  for i := 2 to n do
  begin
    j := i; k := j div 2;
    x := d[i];
    while (k > 0) and (d[k] < x) do
    begin
      d[j] := d[k];
      j := k; k := j div 2;
    end;
    d[j] := x;
  end;

// Rozbieramy kopiec

  for i := n downto 2 do
  begin
    x := d[1]; d[1] := d[i]; d[i] := x;
    j := 1; k := 2;
    while k < i do
    begin
      if (k + 1 < i) and (d[k + 1] > d[k]) then
        m := k + 1
      else
        m := k;
      if d[m] <= d[j] then break;
      x := d[j]; d[j] := d[m]; d[m] := x;
      j := m; k := j + j;
    end;
  end;
end;

function Sort : extended;
begin
  QueryPerformanceCounter(addr(qpc1));
  HeapSort;
  QueryPerformanceCounter(addr(qpc2));
  Sort := (qpc2 - qpc1 - tqpc) / qpf;
end;

// Program główny
//---------------
var
  i,j,k                : integer;
  tpo,tod,tpp,tpk,tnp : extended;
  f                    : Text;
begin
  if QueryPerformanceFrequency(addr(qpf)) then
  begin
    QueryPerformanceCounter(addr(qpc1));
    QueryPerformanceCounter(addr(qpc2));
    tqpc := qpc2 - qpc1;

    assignfile(f,'wyniki.txt'); rewrite(f);

// Wydruk na ekran

    writeln('Nazwa: ',NAZWA);
    writeln(K1);
    writeln(K2);
```

```
    writeln;
    writeln(K3);

// Wydruk do pliku

    writeln(f,'Nazwa: ',NAZWA);
    writeln(f,K1);
    writeln(f,K2);
    writeln(f,'');
    writeln(f,K3);
    for i := 1 to MAX_LN do
    begin
      n := LN[i];

// Czas sortowania zbioru posortowanego

      for j := 1 to n do d[j] := j;
      tpo := Sort;

// Czas sortowania zbioru posortowanego odwrotnie

      for j := 1 to n do d[j] := n - j;
      tod := Sort;

// Czas sortowania zbioru posortowanego
// z przypadkowym elementem na początku - średnia z 10 obiegów

      tpp := 0;
      for j := 1 to 10 do
      begin
        for k := 1 to n do d[k] := k;
        d[1] := random * n + 1;
        tpp += Sort;
      end;
      tpp /= 10;

// Czas sortowania zbioru posortowanego
// z przypadkowym elementem na końcu - średnia z 10 obiegów

      tpk := 0;
      for j := 1 to 10 do
      begin
        for k := 1 to n do d[k] := k;
        d[n] := random * n + 1;
        tpk += Sort;
      end;
      tpk /= 10;

// Czas sortowania zbioru nieuporządkowanego - średnia z 10 obiegów

      tnp := 0;
      for j := 1 to 10 do
      begin
        for k := 1 to n do d[k] := random;
        tnp += Sort;
      end;
      tnp /= 10;

      writeln(n:7,tpo:12:6,tod:12:6,tpp:12:6,tpk:12:6,tnp:12:6);
      writeln(f,n:7,tpo:12:6,tod:12:6,tpp:12:6,tpk:12:6,tnp:12:6);
    end;
    writeln(K4);
    writeln(f,K4);
    writeln(f,'Koniec');
```

```
    closefile(f);
    writeln;
    writeln('Koniec. Wyniki w pliku WYNIKI.TXT');
  end
  else writeln('Na tym komputerze program testowy nie pracuje !');
  writeln;
  write('Nacisnij klawisz ENTER...'); readln;
end.
```

Otrzymane wyniki są następujące (dla komputera o innych parametrach wyniki mogą się różnić co do wartości czasów wykonania, dlatego w celach porównawczych proponuję uruchomić podany program na komputerze czytelnika):

Zawartość pliku wygenerowanego przez program

```
Nazwa: Sortowanie przez kopcowanie
------------------------------------------------------------
(C)2011/2012 I Liceum Ogolnoksztalcace  w Tarnowie

------n---------tpo---------tod---------tpp---------tpk---------tnp
   1000    0.000567    0.000541    0.000564    0.000760    0.000500
   2000    0.001245    0.001056    0.001251    0.001294    0.001376
   4000    0.002780    0.002234    0.003499    0.002826    0.002729
   8000    0.006278    0.004912    0.006704    0.006385    0.005474
  16000    0.013232    0.011120    0.013231    0.013665    0.012407
  32000    0.028305    0.024424    0.028688    0.028988    0.028558
  64000    0.060927    0.052423    0.061291    0.062938    0.067073
 128000    0.133429    0.112287    0.131137    0.133940    0.158330
------------------------------------------------------------------
Koniec
```

Objaśnienia oznaczeń (wszystkie czasy podano w sekundach):

- n - ilość elementów w sortowanym zbiorze
- t_{po}- czas sortowania zbioru posortowanego
- t_{od}- czas sortowania zbioru posortowanego malejąco
- t_{pp}- czas sortowania zbioru posortowanego z losowym elementem na początku
- t_{pk}- czas sortowania zbioru posortowanego z losowym elementem na końcu
- t_{np}- czas sortowania zbioru z losowym rozkładem elementów

(Arkusz kalkulacyjny Excel do wyznaczania klasy czasowej złożoności obliczeniowej)

(Arkusz kalkulacyjny Excel do wyznaczania wzrostu prędkości sortowania)

Podsumowanie

Analizując wyniki obliczeń w arkuszu kalkulacyjnym otrzymanych czasów sortowania dla algorytmu sortowania przez kopcowanie wyciągamy następujące wnioski:

Cechy Algorytmu Sortowania Przez Kopcowanie	
klasa złożoności obliczeniowej optymistyczna	$O(n \log n)$
klasa złożoności obliczeniowej typowa	
klasa złożoności obliczeniowej pesymistyczna	
Sortowanie w miejscu	TAK
Stabilność	NIE

Klasy złożoności obliczeniowej szacujemy następująco:

- **optymistyczna** - dla zbiorów uporządkowanych (z niewielką liczbą elementów nie na swoich miejscach) - na podstawie czasów t_{po}, t_{pp}, t_{pk}
- **typowa** - dla zbiorów o losowym rozkładzie elementów - na podstawie czasu t_{np}
- **pesymistyczna** - dla zbiorów posortowanych odwrotnie - na podstawie czasu t_{od}.

Własności algorytmu					
Algorytm	t_{po}	t_{od}	t_{pp}	t_{pk}	t_{np}
Sortowanie przez kopcowanie	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
	$t_{po} \approx \frac{7}{6} t_{od}$		$t_{pp} \approx t_{pk}$		$t_{np} \approx \frac{4}{3} t_{od}$

1. Wszystkie badane czasy są proporcjonalne do $n \log_2 n$, zatem wnioskujemy, iż algorytm sortowania przez kopcowanie posiada klasę czasowej złożoności obliczeniowej równą $O(n \log n)$.
2. Najdłużej trwa sortowanie zbioru nieposortowanego. Otrzymane czasy nie różnią się wiele od siebie, co sugeruje, iż algorytm jest mało czuły na postać danych wejściowych.
3. Ciekawostką jest to, iż czas sortowania zbiorów posortowanych jest dłuższy od sortowania zbioru posortowanego odwrotnie (jest to najkrótszy czas z otrzymanych, zatem możemy przyjąć, iż dla algorytmu sortowania przez kopcowanie przypadkiem optymistycznym jest właśnie sortowanie zbioru posortowanego odwrotnie).

Wzrost prędkości sortowania					
Algorytmy	t_{po}	t_{od}	t_{pp}	t_{pk}	t_{np}
Sortowanie przez scalanie Sortowanie przez kopcowanie	$\approx \frac{1}{2}$ **źle**	$\approx \frac{1}{2}$ **źle**	$\approx \frac{1}{2}$ **źle**	$\approx \frac{1}{2}$ **źle**	$\approx \frac{2}{5}$ **źle**

4. Z porównania czasów sortowania wynika jasno, iż przedstawiony tutaj algorytm sortowania przez kopcowanie jest około dwa razy wolniejszy od wcześniej podanego algorytmu sortowania przez scalanie. Zaletą jest sortowanie w miejscu - poprzedni algorytm wymaga dodatkowej pamięci, co może go dyskwalifikować przy sortowaniu dużych zbiorów danych. Do dalszych porównań czasów sortowania będziemy dalej stosowali algorytm sortowania przez scalanie.

Zadania dla ambitnych

1. Co należy zmienić w podanym algorytmie, aby uzyskać sortowanie malejące?
2. Udowodnij, iż algorytm sortowania przez kopcowanie nie jest stabilny.
3. Uzasadnij, iż czas sortowania zbioru posortowanego odwrotnie jest najkrótszy.
4. Uzasadnij, iż klasa złożoności algorytmu sortowania przez kopcowanie wynosi $O(n \log n)$.

Zobacz również na: Drzewo binarne | Tworzenie kopca | Rozbiór kopca

---

Temat:

Uwaga: ← **tutaj wpisz wyraz ilo , inaczej list zostanie zignorowany**

Poniżej wpisz swoje uwagi lub pytania dotyczące tego rozdziału (max. 2048 znaków).

Liczba znaków do wykorzystania: 2048

Wyślij Kasuj

W związku z dużą liczbą listów do naszego serwisu edukacyjnego nie będziemy udzielać odpowiedzi na prośby rozwiązywania zadań, pisania programów zaliczeniowych, przesyłania materiałów czy też tłumaczenia zagadnień szeroko opisywanych w podręcznikach.

I Liceum Ogólnokształcące
im. Kazimierza Brodzińskiego
w Tarnowie