Digitální učební materiál

Číslo projektu:	CZ.1.07/1.5.00/34.0548
Název školy:	Gymnázium, Trutnov, Jiráskovo náměstí 325
Název materiálu:	VY_32_INOVACE_150_IVT
Autor:	Ing. Pavel Bezděk
Tematický okruh:	Algoritmy
Datum tvorby:	srpen 2013
Ročník:	4. ročník a oktáva
Anotace:	Algoritmus X. – Třídění dat II. - Přihrádky : Bucket Sort, Radix Sort
Metodický pokyn:	Při výuce nutno postupovat individuálně. Části DUM – „ Pro hloubavé“ jsou určeny pro zájemce o studium na technických a matematicko-fyzikálních oborech vysokých škol.

esf evropský sociální fond v ČR
EVROPSKÁ UNIE
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY
OP Vzdělávání pro konkurenceschopnost
INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Autor	**Ing. Pavel Bezděk**		
Vytvořeno dne	**10. 8. 2013**		
Odpilotováno dne	**20. 2. 2014**	ve třídě	**4.A**
Vzdělávací oblast	**Informatika a informační a komunikační technologie**		
Vzdělávací obor	**Informatika a výpočetní technika**		
Tematický okruh	**Algoritmus**		
Téma	**Algoritmus X. - Třídění dat II. - Přihrádky : Bucket Sort, Radix Sort**		
Klíčová slova	**Algoritmus, přihrádkové třídění dat, Radix Sort**		

Přihrádkové třídění dat

Bucket Sort a Radix Sort

Pro hloubavé

Bucket Sort

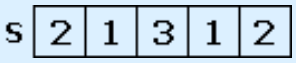
Přihrádkové algoritmy - algoritmy řadící nikoliv na základě porovnávání, ale využívající vlastností prvků řazené sekvence k provádění distribucí (roztřídění).

V rámci algoritmu **Bucket Sort** se využívá předpoklad, že prvky řazené sekvence jsou z prostoru, jehož velikost je malá, daná nějakou konstantou (např. **každý prvek řazené sekvence je z množiny {1,2,3,...,m}**).

Bucket Sort využívá ***m*** počítadel. ***I-té* počítadlo uvádí počet výskytů *i-tého* elementu v sekvenci**. Jediným průchodem sekvencí lze určit hodnoty počítadel a získat seřazenou sekvenci.

Bucket Sort

Příklad: Mějme prvky **m=3 {1,2,3}** a řazenou sekvenci **S.**



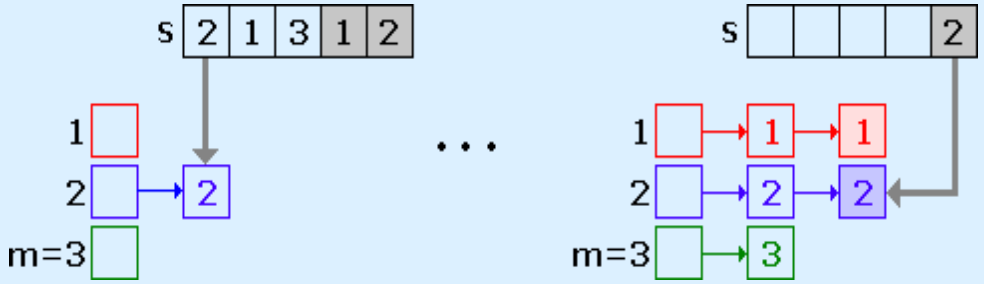
Sekvence S - prvek "2" a prvek "1" jsou v sekvenci obsaženy dvakrát

Budeme tedy postupovat následujícím způsobem:

1) vytvoříme ***m* počítadel** (věder - buckets)

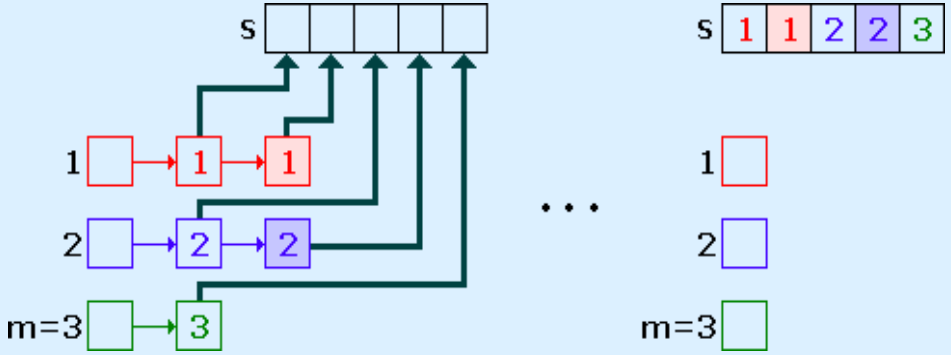
Vytvořená tři počítadla

2) každý prvek ze sekvence **S** je vložen do jednoho z ***m*** počítadel



Každý prvek je ve správném počítadle

3) přetáhneme prvky z jednotlivých počítadel do původní sekvence **S**



Časová složitost Bucket Sort

Algoritmus Bucket Sort má

asymptotickou časovou složitost O(m + n),

kde **n** je **počet prvků** řazené sekvence a **m** je **počet počitadel**.

Bucket Sort je algoritmus s lineární časovou složitostí.

Přihrádkové třídění Radix Sort

velmi jednoduché třídění

Data na vstupu
(v řádcích)

853	322	636	7	739	89	320	231	259	48
385	801	540	270	932	691	547	958	500	235
483	415	206	357	24	122	11	564		

Přihrádky: **Čísla rozdělím do přihrádek, nejprve podle jednotek, pak beru postupně přihrádky (s rozdělenými čísly podle jednotek) od 0 do 9 a obsah každé přihrádky rozdělím do dalších přihrádek, tentokrát podle desítek, vybírám ze přihrádky podle pořadí, jak jsem do přihrádky ukládal, když skončím, pak beru postupně přihrádky (podle desítek) od 0 do 9 a obsah každé přihrádky rozdělím do nových přihrádek, nyní podle stovek. Dodržuji výběr z každé přihrádky, podle pořadí, jak bylo do ní ukládáno.**

Jednotky

0	1	2	3	4	5	6	7	8	9
320	231	322	853	24	385	636	7	48	739
540	801	932	483	564	235	206	547	958	89
270	691	122			415		357		259
500	11								

Desítky

0	1	2	3	4	5	6	7	8	9
500	11	320	231	540	853	564	270	483	691
801	415	322	932	547	357			385	
206		122	235	48	958			89	
7		24	636		259				
			739						

Stovky

0	1	2	3	4	5	6	7	8	9
7	122	206	320	415	500	636	739	801	932
11		231	322	483	540	691		853	958
24		235	357		547				
48		259	385		564				
89		270							

Výstup

7	11	24	48	89	122	206	231	235	259
270	320	322	357	385	415	483	500	540	547
564	636	691	739	801	853	932	958		

Horní prvek ve sloupci

- začátek pole

Dolní prvek ve sloupci

- konec pole

Pole přihrádek seřadím za sebou a mám setříděné vstupní pole

Popis průběhu třídění

Toto třídění je zvláštní v tom, že se třídí podle cifer (nebo jiné sekvence). Zopakujeme si tedy postup pro trojciferná čísla.

Rozdělíme je do přihrádek 0, 1, 2 … 9, a to podle jejich poslední cifry.

Poté vezmeme přihrádku 0, ze které budeme vybírat čísla (vybíráme podle pořadí, jak jsme čísla do přihrádky ukládali) a přiřazovat je podle druhé cifry do přihrádek 0, 1, 2 … 9. Nebudou to ovšem ty samé přihrádky, ale přihrádky pro druhý řád.

Tak budeme pokračovat i s čísly z přihrádek 1 – 9 prvního řádu. Teď postupně budeme brát čísla z přihrádek 2. řádu.

Začneme opět u 0 a umístíme je do přihrádky 0 – 9 třetího řádu, a to právě podle třetí cifry. Čísla z přihrádek se berou v takovém pořadí, v jakém jsme je vložili. Použijeme tedy frontu (data, která jdou první dovnitř, jdou první ven, na rozdíl od zásobníku, kde data, která jsme uložili naposledy, čteme jako první).

Jako příklad uvedu cestu čísla 473. V prvním rozmisťování se číslo dostane do přihrádky číslo 3. Z té je při následujícím přemístění přesunuto do přihrádky druhého řádu s číslem 7 a na závěr je přesunuto do přihrádky třetího řádu s číslem 4.

Princip Radix Sort

Třídící algoritmus Radix Sort je, stejně jako Bucket Sort, založen na řazení pomocí distribucí (rozdělování).

Radix Sort lze využít v případě, že prvky řazené sekvence lze chápat jako sekvence cifer, popřípadě písmen nebo jiných symbolů.

Radix Sort je postaven na algoritmu Bucket Sort, ale je použitelný pro mnohem rozsáhlejší univerzální množiny, z nichž vybíráme prvky řazené posloupnosti.

1. řazení dle nejméně významné cifry (jednotky)
2. řazení dle významnější cifry (desítky)
3. řazení dle nejvýznamnější cifry (stovky)

Každý z těchto kroků je implementován jako Bucket Sort. V každém kroku budeme potřebovat **m** počítadel.

V našem příkladě jsme pracovali s dekadickými ciframi, a proto potřebujeme v každém kroku **10 počítadel** (buckets) v intervalu **<0,9>.**
Obecně lze **Radix Sort užít v případě, že řazené prvky jsou P-ciferná přirozená čísla soustavě o základu R.**

Každý prvek je tedy v intervalu <0,R-1>.

Radix Sort má v tomto případě P-kroků. V každém kroku se užívá přesně R počítadel.

Časová složitost algoritmu Radix Sort

Předpokládejme, že jednotlivé kroky algoritmu Radix Sortu lze provádět vždy jedním průchodem sekvence, tedy v O(n). Potom celková časová složitost algoritmu bude součtem těchto kroků. Počet těchto kroků je dán **počtem cifer P**, z nichž jsou složeny tříděné prvky sekvence.

Algoritmus Radix Sort má tedy dobu běhu **O(nP)**, kde **n** je **počet prvků řazené sekvence**.

Tato rychlost je ale vyvážena tím, že je algoritmus omezen pouze na sekvence cifer, popřípadě písmen nebo jiných symbolů a není tedy tak univerzální jako ostatní třídící algoritmy.

Přihrádkové třídění
Pascal

Pro hloubavé

```
program Prihradkove_Trideni
const N = 10; {pocet zaznamu}
D = 1; {dolni mez klice} H = 999; {horni mez klice}
type Pole = array[1..N] of record    klic: integer;  {cosi dalsiho ...} end;
var i:integer;    A:Pole;
procedure PrihradkoveTrideni(var A: Pole);
  var B: Pole;              {setridene zaznamy}
      C: array[D..H] of integer;  {hranice prihradek}
      I, K: integer;
  begin  {Velikosti prihradek:}
   for I:= D to H do C[I]:= 0;
   for I:= 1 to N do begin K := A[I].klic; C[K] := C[K] + 1 end;{Konce prihradek:}
   for I:= D+1 to H do    C[I] := C[I] + C[I-1];{Zarazeni zaznamu do prihradek:}
   for I:= N downto 1 do
                  begin      K:= A[I].klic; B[C[K]]:= A[I]; C[K]:= C[K] -1      end;
   A := B {Predani setrideneho pole ven:}
  end; { konec procedury  PrihradkoveTrideni}
begin
 writeln('Zadej ',N,' tridenych cisel z rozmezi od ',D,' do ',H,' :');
 for i:=1 to N do read(A[i].klic); PrihradkoveTrideni(A);
 writeln('Setrideno:'); for i:=1 to N do write(A[i].klic:5); writeln
end.
```

Přihrádkové třídění
Pascal

Pro hloubavé

```
Program SortRadix;
Uses Crt, Dos;
Type
   AType = Array [1..400] of Integer;
   Ptr   = ^Node;
   Node  = Record
        Info : Integer;
        Link : Ptr;
      end;
   LType = Array [0..9] of Ptr;
Var
  Ran     : AType;
  MaxData : Integer;
```

```
Procedure ReadData (Var A : AType; Var MaxData : Integer);
Var I : Integer;
Begin       MaxData := 10;      For I := 1 to MaxData do read(A [I]); end;

Procedure WriteArray (A : AType; MaxData : Integer);
Var I : Integer;
Begin     For I := 1 to MaxData do       Write (A [I] : 7);   Writeln;     end;

Procedure Insert (Var L : LType; Number, LN : Integer);
Var
  P, Q : Ptr;
Begin      New (P);    P^.Info := Number;    P^.Link := Nil;   Q := L [LN];
  if Q = Nil then         L [LN] := P
                else   begin
                            While Q^.Link <> Nil do       Q := Q^.Link;     Q^.Link := P;
                       end;
end;
```

```
Procedure Refill (Var A : AType; Var L : LType);
Var
  I, J : Integer;
  P    : Ptr;
begin
  J := 1;     For I := 0 to 9 do
          begin
                P := L [I];      While P <> Nil do
                                    begin
                                        A [J] := P^.Info;     P := P^.Link;       J := J + 1;
                                    end;
          end;
  For I := 0 to 9 do       L [I] := Nil;
end;
```

```
Procedure RadixSort (Var A : AType; MaxData : Integer);
Var
  L       : LType;
  I,  divisor,  ListNo,  Number   : Integer;
begin
  For I := 0 to 9 do L [I] := Nil;
  divisor := 1;
  While divisor <= 1000 do
      begin      I := 1;      While I <= MaxData do
                          begin
                            Number := A [I];       ListNo := Number div divisor MOD 10;
                             Insert (L, Number, ListNo);        I := I + 1;
                          end;
                 Refill (A, L);
                 divisor := 10 * divisor;
      end;
end;
```

```
begin
   clrscr;
   writeln;       writeln('    Put the numbers to sort: ');
   ReadData (Ran, MaxData);

   writeln ('   Unsorted : ');
   writeArray (Ran, MaxData);
   writeln;       writeln('   Start sorting! ');
   writeln;
    Delay(500);

   RadixSort (Ran, MaxData);
   writeln ('   Sorted   : ');
   writeArray (Ran, MaxData);
   writeln;        writeln;

   writeln('   Press the spacebar to exit the program ');
   repeat until keypressed;
end.
```

Přihrádkové třídění C++

Pro hloubavé

```
#include <iostream>
#include <vector>
#include <math.h>
using namespace std;
void printSorted(int x[], int length);
vector < vector <int> > buckets;
void radixSort(int x[],int length) //Begin Radix Sort
 {    int temp;       int m=0;
     for(int i=0;i<7;i++)          //Determine which bucket each element should enter
          {  for(int j=0;j<length;j++)
               { temp=(int)((x[j])/pow(10,i))%10;
                    buckets[temp].push_back((x[j]));
                }        //Transfer results of buckets back into main array
                   for(int k=0;k<10;k++)
                     {  for(int l=0;l<buckets[k].size();l++)
                          {  x[m]=buckets[k][l];  m++;     }
          buckets[k].clear();                    //Clear previous bucket
                     }          m=0;
          }       buckets.clear();      printSorted(x,length);
}
```

```
void printSorted(int x[], int length)
  {
   for(int i=0;i<length;i++)
       cout<<x[i]<<endl;
  }
int main(){
   int testcases;
   cout<<"How many numbers do you want to sort? "<<endl;      cin>>testcases;
   int input[testcases];
   buckets.resize(10);
   int number;
   cout<<"Put the numbers: "<<endl;
   for(int i=0;i<testcases;i++)
     { cout<<i<<". number: ";     cin>>number;          input[i]=number;  }
      cout<<endl;
   cout<<"I print sorted numbers:"<<endl;

   radixSort(input,testcases);
   return 0;
}
```

Doporučené video

Radix Sort:

http://www.youtube.com/watch?v=ibtN8rY7V5k

Použité zdroje

Algoritmy a datové struktury [online]. 20. 07. 2002 [cit. 2013-08-10]. Dostupné z: http://www.alg.webzdarma.cz/diplomka/kap5/bucket.html

BÖHM, Martin. *Programátorská kuchařka: Recepty z programátorské kuchařky* [online]. Praha: KSP MFF UK Praha, 2011/2012 [cit. 2013-08-10]. KSP, Korespondenční seminář z programování: Programátorské kuchařky, 24. ročník KSP. Dostupné z: http://ksp.mff.cuni.cz/tasks/24/cook1.html