Cvičení Programování I

Cvičící: **Pavel Surynek, KTIML,**
pavel.surynek@seznam.cz
Semestr: **Zima 2005/2006**
Kroužek: **Matematika/59**
Rozvrh: **Pátek 10:40-12:10 (učebna K2)**

Stručné poznámky ke cvičení ze 11.11.2005

1. Organizační záležitosti. 18.11. proběhne od 8:00 v počítačové laboratoři v Karlíně další **pascalovská schůzka**.

Jakékoli **dotazy** ke cvičení lze posílat na uvedenou e-mailovou adresu. Osobní konzultace ke cvičení lze dohodnout e-mailem (alespoň den předem).

2. Vyhledávání s setříděné posloupnosti. Je dána setříděná posloupnost N přirozených čísel (například: 5, 20, 27, 35, 60, 66, 91). Napište program v Pascalu, který dostane jako vstup přirozené číslo x (například: 26) a zmiňovanou posloupnost a jehož úkolem bude rozhodnout, zda se zadané přirozené číslo nachází v zadané posloupnosti (v uvedeném příkladě bude odpověď algoritmu ne). Algoritmus může provádět operace porovnání a v každém kroku se „může podívat“ na jedno číslo na libovolné pozici v posloupnosti. Snažte se minimalizovat počet kroků algoritmu.

Řešení. První varianta programu. Program prochází setříděnou posloupnost postupně od začátku a prohlíží jednotlivé její prvky. Když zjistí, že je prohlížený prvek roven hledanému, oznámí to a ukončí procházení posloupnosti. Když zjistí, že prohlížený prvek je větší než hledaný, oznámí, že hledání bylo neúspěšné a ukončí procházení posloupnosti. Když nenastane ani jeden z těchto případů, program pokračuje v procházení posloupnosti.

```
program linearni_vyhledavani;

const N = 7; { delka vstupni posloupnosti }
const posl: array[1..10] of integer = (5,20,27,35,60,66,91);
  { vstupni setridena posloupnost }
var x: integer; { hledane cislo }
var i: integer; { pomocna promenna }

begin
   read(x);
   i := 1;
   while i <= N do begin
      if posl[i] = x then begin
           writeln('Nalezeno.');
           break;
      end
      else begin
           if posl[i] > x then begin
                writeln('Nenalezeno.');
                break;
           end;
      end;
      i := i + 1;
   end;
end.
```

V nejhorším případě provede algoritmus cN kroků, kde c je zhruba 3 (v každé otáčce cyklu se provedou asi tři porovnání). Asymptotická časová složitost algoritmu je tedy $O(N)$. Promyslete jak snížit konstantu c (nápověda: vyhledávání se zarážkou).

Druhá varianta programu. Program bude zadané číslo hledat pomocí půlení intervalu. V každém kroku algoritmus vyhledává zadané číslo mezi d-tým a h-tým prvkem vstupní setříděné posloupnosti (na začátku je $d = 1$ a $h = N$). Interval $[d, h]$ posloupnosti rozdělí na dvě poloviny a podle hodnoty prostředního prvku tohoto intervalu rozhodne, ve které polovině intervalu se hledané číslo nachází. Potom algoritmus stejným způsobem pokračuje v hledání, tentokrát ale na intervalu poloviční velikosti. Toto půlení intervalu algoritmus provádí tak dlouho, dokud se velikost intervalu nezmenší na 1 (0), pak už lze snadno rozhodnout.

```
program binarni_vyhledavani;

const N = 7; { delka vstupni posloupnosti }
const posl: array[1..10] of integer = (5,20,27,35,60,66,91);
   { vstupni setridena posloupnost }
var x: integer; { hledane cislo }
var s: integer; { prostredni index intervalu }

procedure vyhledej(d,h,x: integer);
   { d - dolni mez, h - horni mez, x - hledani cislo }
begin
   if h-d = 0 then begin
      if posl[h] = x then begin
            writeln('Nalezeno.');
      end
      else begin
            writeln('Nenalezeno.');
      end;
   end
   else begin
      if h-d>0 then begin
            s := (h+d) div 2;
            if x >= posl[s] then begin
                  vyhledej(s, h, x);
            end
            else begin
                  vyhledej(d, s-1, x);
            end;
      end
      else begin
            writeln('Nenalezeno.');
      end;
   end;
end;

begin
   read(x);
   vyhledej(1,N,x);
end.
```

V nejhorším případě provede algoritmus $c \log_2 N$. Asymptotická časová složitost algoritmu je tedy $O(\log_2 N)$.

3. Souvislost grafu maticově. Vsuvka: Násobení matic. Nechť A je matice typu $m \times n$ nad přirozenými čísly

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}, \text{ kde } a_{ij} \in N \text{ pro } i \in \{1,2,...,m\}, j \in \{1,2,...,n\}$$

nechť B je matice typu $n \times k$ nad přirozenými čísly

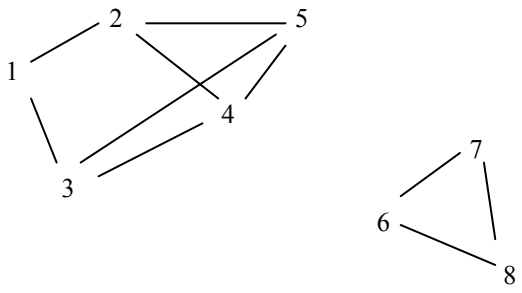
$$B = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1k} \\ b_{21} & b_{22} & \dots & b_{2k} \\ \vdots & \vdots & & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nk} \end{pmatrix}, \text{ kde } b_{ij} \in N \text{ pro } i \in \{1,2,...,n\}, j \in \{1,2,...,k\}.$$

Součinem matic A a B je matice C typu $m \times k$ nad přirozenými čísly

$$A \times B = C = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1k} \\ c_{21} & c_{22} & \dots & c_{2k} \\ \vdots & \vdots & & \vdots \\ c_{m1} & c_{m2} & \dots & c_{mk} \end{pmatrix}, \text{ kde } c_{ij} = \sum_{l=1}^{n} a_{il} b_{lj} \text{ pro } i \in \{1,2,...,m\}, \ j \in \{1,2,...,k\}.$$

Násobení matic lze snadno definovat též pro Boolovské matice, tj. matice nad $\{0,1\}$, resp. $\{pravda, nepravda\}$. Stačí nahradit operaci součtu operací logického součtu a operaci součinu operací logického součinu.

Je dán neorientovaný graf $G = (V, E)$, kde $V = \{1,2,...,n\}$, pro n přirozené číslo, a $E \subseteq \binom{V}{2}$. V je množina vrcholů a E je množina hran. Graf lze znázornit diagramem, jako například na následujícím obrázku (vrcholy jsou vrcholy a hrany spojnice mezi vrcholy):



Uvedený graf lze jednoznačně reprezentovat pomocí Boolovské matice M typu $n \times n$, kde $m_{ij} = 1$, právě když $\{i, j\} \in E$ a $m_{ij} = 0$ jinak, a toto platí pro každé $i, j \in \{1,2,...,n\}$. Matice M se nazývá **matice sousednosti** grafu G. V matici $M' = M \times M$ je $m'_{ij} = 1$, právě když mezi vrcholy i a j vede cesta délky 2, $m'_{ij} = 0$ jinak, a toto platí pro každé $i, j \in \{1,2,...,n\}$.

Navrhněte co nejefektivnější algoritmus a zformulujte jej jako program v Pascalu, který odpoví na dotaz (nebo více dotazů), zda mezi vrcholy i a j vede cesta. Využijte přitom násobení matic.