Algoritmy pro práci s vektory a s maticemi

Jitka Kreslíková, Aleš Smrčka

2023

Fakulta informačních technologií
Vysoké učení technické v Brně

IZP – Základy programování

Algoritmy pro práci s vektory a s maticemi

- Algoritmy pro práci s vektory
- Algoritmy pro práci s maticemi

Algoritmy pro práci s vektory

Uspořádanou n-tici reálných čísel $a = (a_1,a_2,a_3, \ldots,a_n)$ nazýváme n-rozměrným reálným (aritmetickým) vektorem.

Pořadí	1.	2.	3.	4.	. . .	n.
Hodnota	a_1	a_2	a_3	a_4	. . .	a_n
Index	0	1	2	3	. . .	n-1

Hodnota a_i, i= 1,2,3,…,n se nazývá i-tá složka vektoru **a**.

Algoritmy pro práci s vektory

Sémantický význam v programování:

- **Datový typ pole**
  - N položek stejného typu opakovaně uložených za sebou v paměti.
  - Na jednotlivé prvky se lze odkazovat pořadovým indexem.
  - V jazyce C vždy indexy začínají od nuly.
  - U pole délky N tedy bude rozsah použitelných indexů <0,N-1>.

Algoritmy pro práci s vektory

- **Vztah polí a vektorů**
  - Datový typ pole má v programovacích jazycích obecnější použití nežli jen ve významu vektoru.
  - Skutečný význam použitého pole závisí na povaze úlohy a interpretaci.
  - Pole znaků takto například můžeme považovat za textový řetězec.
  - Pole čísel můžeme interpretovat jako:
    - obecnou posloupnost,
    - vektor popisující vztahy v N-rozměrném prostoru.

Operace nad polem

- Operace nad polem:
  - přiřazení hodnoty určitému prvku, který je zadán indexem,
  - přiřazení stejné hodnoty všem prvkům,
  - vyhledání prvku s určitou hodnotou,
  - seřazení prvků podle relace uspořádání (sestupně, vzestupně) ,
  - operace pro jednotlivé prvky pole odpovídající typu prvku,
  - vložení prvku(ů) mezi jiné prvky, vyjmutí prvku + odpovídající posun prvků.

Operace nad vektory

- Operace nad vektory:
  - násobení vektoru konstantou,
  - skalární součin vektorů,
  - vektorový součin vektorů,
  - součet vektorů,
  - rozdíl vektorů atd.
- Implementační poznámka:
  - pole se v jazyku C předávají automaticky odkazem,
  - identifikátor pole má stejný význam, jako konstantní ukazatel na první prvek pole,
  - při volání se před parametr typu pole **nepíše** referenční operátor (&).

Operace nad vektory

- Násobení vektoru konstantou.

Pro násobení vektoru konstantou platí vztah: $K\vec{a} = \vec{b}$

Hodnoty jednotlivých prvků nového vektoru pak vypočítáme podle vztahu:

$b_i = K.a_i$, $i = 1, 2, \ldots, n$

Příklad: varianta s vektorem obecné délky. Výsledkem je modifikovaný vektor.

```
void multConst(double v[], int n, double k)
{
  for(int i = 0; i < n; i++)
  {
    v[i] *= k;
  }
}
```

Operace nad vektory

Příklad: varianta s vektorem obecné délky, který je uložen ve struktuře a alokován dynamicky. Výsledkem bude nově alokovaný vektor.

```
typedef struct tvector
{
  int n;
  double *v;
} TVector;
TVector multConst(const TVector *v, double c)
{
  TVector w = allocVect(v->n); // alokuje potřebnou paměť
  for(int i = 0; i < w.n; i++)
  {
    w.v[i] = v->v[i] * c;
  }
  return w;
}
```

Operace nad vektory

Poznámka: v dalších příkladech budeme pro jednoduchost používat staticky alokované pole o velikosti N, pokud nebude výslovně uvedeno něco jiného.

```
#define N 100
double pole[N];
```

- Součet dvou vektorů.

Pro součet dvou vektorů o stejném rozměru platí:

$$\vec{a} + \vec{b} = \vec{c}$$

Hodnoty prvků tohoto nového vektoru se pak počítají podle vztahu:

$c_i = a_i + b_i \quad i = 1, 2, \ldots n,$

Operace nad vektory

Příklad: součet dvou vektorů uložených v poli **a** a **b,** výsledek bude uložen do pole **c**.

```
void addVect(double a[], double b[], double c[])
{
  for(int i = 0; i < N; i++)
  {
    c[i] = a[i] + b[i];
  }
}
```

Operace nad vektory

- Skalární součin dvou vektorů.

Pro skalární součin dvou vektorů o stejném rozměru platí:

$$\vec{a} \cdot \vec{b} = c$$

Hodnota skalárního součinu se počítá podle vztahu:

$$c = \sum_{i=1}^{n} a_i b_i$$

Operace nad vektory

Příklad: funkce vrací skalární součin dvou vektorů o stejném rozměru.

```
double multScalar(double a[], double b[])
{
  double scalar = 0;
  for(int i = 0; i < N; i++)
  {
    scalar += a[i]*b[i];
  }
  return scalar;
}
```

Operace nad vektory

- Vektorový součin.

Vektorovým součinem (**a** × **b**) vektorů **a**(a1,a2,a3), **b**(b1,b2,b3), nazýváme vektor:

$$\vec{w} = \vec{a} \times \vec{b}$$

$$\vec{w} = \left(\begin{vmatrix} a_2, & a_3 \\ b_2, & b_3 \end{vmatrix}, \begin{vmatrix} a_3, & a_1 \\ b_3, & b_1 \end{vmatrix}, \begin{vmatrix} a_1, & a_2 \\ b_1, & b_2 \end{vmatrix} \right)$$

součin prvků

- +

Lit.: Bartsch: Matematické vzorce, 2006, studentské vydání
Vektorový součin, str. 269

Operace nad vektory

Příklad: funkce pro výpočet vektorového součinu dvou vektorů. Výsledek bude uložen do pole w.

```
#define N 3
...
void multVect(double a[], double b[], double w[])
{
   w[0] = a[1]*b[2] - a[2]*b[1];
   w[1] = a[2]*b[0] - a[0]*b[2];
   w[2] = a[0]*b[1] - a[1]*b[0];
}
```

Operace nad poli

- Hledání prvočísel <2,N>
  - Postupně se berou všechna čísla X počínaje 2 a konče N a tato se dělí všemi čísly od 2 do odmocniny z X a zjišťují se zbytky po dělení.
  - Je- li některý zbytek roven nule, pak číslo **není** prvočíslo v opačném případě je prvočíslo.
  - Mohou se použít různé optimalizace (např. nedělíme číslem 2, případně dělíme jen prvočísly do odmocniny z X).
  - Efektivnější způsob ➔ **Eratostenovo síto**

Operace nad poli

- **Eratostenovo síto**

Pomocí pole:

1. vytvoříme (bitové) pole tak, že pro každé přirozené číslo od 2 do N vyhradíme jeden prvek pole (bit), N je horní hranice výpisu prvočísel (N-1).
2. index pole bude uvádět číslo, hodnota prvku pole rovna 1 bude znamenat, že číslo je prvočíslo, 0, že není prvočíslo.
3. na začátku pole inicializujeme tak, jako by všechna čísla byla prvočísly (nastavíme na 1).

Operace nad poli

Pole po inicializaci 1 - značí je prvočíslo, 0 - značí není prvočíslo

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

4. procházíme prvky pole a vyhledáváme první nenulový bit (v prvním průchodu to bude u indexu 2, dále 3, 5, 7 atd.).
5. nalezené číslo (P) je prvočíslo, proto na jeho místě necháme v poli jedničku.

Operace nad poli

6. nyní bereme všechny prvočíselné násobky P počínaje P (P$\times$P, P$\times$(první prvočíslo za P z předchozího kroku), P$\times$(druhé prvočíslo za P z předchozího kroku), atd.) a na místa těchto čísel ukládáme nuly (zcela jistě to nejsou prvočísla).

Najdeme číslo 2, označíme násobky dvou (2$\times$2, 3$\times$2, 4$\times$2, ...)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
		1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

0 – není prvočíslo 1 – je prvočíslo

Operace nad poli

7. Postupujeme tak dlouho, dokud hodnota násobku nepřesáhne zadané N.
8. vrátíme se k bodu 4) a cyklus opakujeme od současného P dotud, dokud není P větší než odmocnina z N.

Najdeme číslo 3, označíme násobky tří (3×3, 5×3, 7×3, ...)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
		1	1	0	1	0	1	0	0	0	1	0	1	0	0	0	1	0	1

0, 0 – není prvočíslo   1 – je prvočíslo

Operace nad poli

Příklad: hledání prvočísel.

```
#include <stdio.h>
#include <stdbool.h>
#define N 20
int main(void)
{
  bool a[N];
  for(int i=2; i<N; i++)
    a[i] = true;
  for(int i=2; i<N; i++)
  { // hledani prvocisel
    if (a[i])
      for(int j=i*i; j<N; j+=i)
        a[j] = false;
  }
```

Operace nad poli

Příklad: hledání prvočísel - pokračování.

```
for(int i=2; i<N; i++)
  { // vypis prvocisel
    if (a[i])
      printf("%d ", i);
  }
  printf("\nkonec\n");
  return 0;
}
```

Složitost algoritmu: N+N/2+N/3+N/5 + .. ~ N ln N (linearitmická).

Úkol: uvedený program neodpovídá zcela přesně výše uvedenému algoritmu. Analyzujte program a upravte ho tak, aby odpovídal uvedenému algoritmu.

Operace nad poli

- Algoritmus výpočtu prvočísel (Eratostenovo síto)

Pomocí množiny:

1. zvolíme množinu zvanou síto a naplníme ji celými čísly z intervalu <2, n>. Prvky množiny reprezentují lichá čísla (prvek 2 reprezentuje číslo 3, prvek 3 číslo 5, obecně prvek i číslo 2i-1).
2. vybereme nejmenší číslo ze síta - je to prvočíslo.
3. Vybrané číslo zahrneme do množiny prvočísel.
4. ze síta odstraníme toto číslo a také všechny jeho násobky.
5. pokud síto není prázdné, opakujeme body 2 až 5.

Operace nad poli

- Reverze řetězce

0	1	2	3	4	5	6	7	8	9
'D'	'o'	'b	'r'	'y'	' '	'd'	'e'	'n'	'\0'

0	1	2	3	4	5	6	7	8	9
'n'	'e'	'd	' '	'y'	'r '	'b'	'o'	'D'	'\0'

Operace nad poli

Příklad: reverze řetězce, rekurzívně - neefektivní.

```
int _revert(char *s, int n)
{
  char c = s[n];
  if (c == '\0')
    return n;
  int length = _revert(s, n+1);
  s[length-(n+1)] = c;
  return length;
}
void revert(char *s)
{
  _revert(s, 0);
}
```

Operace nad poli

Příklad: reverze řetězce - záměnou prvního a posledního znaku v řetězci.

```
#include <string.h>
inline void swap(char *a, char *b)
{// nonekvivalence a přiřazení - XOR
   *a ^= *b;
   *b ^= *a;
   *a ^= *b;
}
void revert(char *s)
{
   int length = strlen(s);
   for(int i = 0; i < length/2; i++)
      swap(&s[i], &s[length-i-1]);
}
```

a	0	0	1	0	0	0	1	0
b	0	1	0	1	0	1	1	0
a	0	1	1	1	0	1	0	0
b	0	0	1	0	0	0	1	0
a	0	1	0	1	0	1	1	0

Poznámka:

inline - funkční specifikátor paměťové třídy,

Specifikátor inline není příkaz, ale požadavek pro překladač, aby přeložená funkce byla co nejrychlejší.

Algoritmy pro práci s maticemi

- Soubor čísel uspořádaných do *m* řádků a *n* sloupců nazýváme *maticí typu (m, n)*.

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} = (a_{ij})_{\substack{i=1,2,\dots,m \\ j=1,2,\dots,n}}$$

  - Čísla a_{ij} se nazývají prvky matice.
  - Matice ukládáme do dvourozměrného pole.

Algoritmy pro práci s maticemi

- Matice je:
  - čtvercová řádu n typ (n, n)
  - obdélníková typ (m, n), $m \neq n$
  - řádková typ $(1, n)$
  - sloupcová typ $(m, 1)$
  - nulová (značíme $\boldsymbol{O}$) všechny prvky jsou rovny nule,
  - jednotková (značíme $\boldsymbol{I}$) - čtvercová matice, prvky hlavní diagonály jsou rovny jedné, všechny ostatní jsou rovny nule

Algoritmy pro práci s maticemi

- transponovaná k matici ***A,*** když zaměníme řádky matice ***A*** za sloupce (značíme ***A***^{T}),
- symetrická - čtvercová matice ***A***, pro kterou platí ***A*** = ***A***^{T},
- trojúhelníkový tvar matice - všechny prvky hlavní diagonály jsou nenulové a všechny prvky nad[***dolní, levá***]/pod[***horní, pravá***] hlavní diagonálou jsou rovny nule.

1	0	0	0
2	7	0	0
5	9	3	0
6	8	2	4

1	3	6	8
0	7	6	5
0	0	3	2
0	0	0	4

Algoritmy pro práci s maticemi

Přístup k prvkům na hlavní diagonále matice ***A***(*n*,*n*).

Indexy prvků na hlavní diagonále čtvercové matice:

A[5][5]

0,0	0,1	0,2	0,3	0,4
1,0	**1,1**	1,2	1,3	1,4
2,0	2,1	**2,2**	2,3	2,4
3,0	3,1	3,2	**3,3**	3,4
4,0	4,1	4,2	4,3	**4,4**

A [i][i]

Operace nad poli

Poznámka: nadále budeme pracovat se statickými dvojrozměrnými poli.

```
#define R 10
#define S 20
int matice1[R][S]; // obdelnikova matice
int matice2[R][R]; // ctvercova matice
```

Příklad: tisk prvků na hlavní diagonále.

```
void printDiagonal(int a[R][R])
{
  for(int i = 0; i < R; i++)
    printf("%d ", a[i][i]);
  printf("\n");
}
```

Algoritmy pro práci s maticemi

Přístup k prvkům na vedlejší diagonále matice ***A***(*n*,*n*).

Indexy prvků na vedlejší diagonále čtvercové matice:

A[5][5]

0,0	0,1	0,2	0,3	0,4
1,0	1,1	1,2	1,3	1,4
2,0	2,1	2,2	2,3	2,4
3,0	3,1	3,2	3,3	3,4
4,0	4,1	4,2	4,3	4,4

A [i] [n-(i+1)]

Operace nad poli

Příklad: tisk prvků na vedlejší diagonále.

```
void printSDiagonal(int a[R][R])
{
  for(int i = 0; i < R; i++)
    printf("%d ", a[i][R-(i+1)]);

  printf("\n");
}
```

Algoritmy pro práci s maticemi

Přístup k prvkům trojúhelníkové matice ***A***(*n*,*n*).

Indexy prvků **horní** trojúhelníkové matice:

A[5][5]

0,0	0,1	0,2	0,3	0,4
1,0	1,1	1,2	1,3	1,4
2,0	2,1	2,2	2,3	2,4
3,0	3,1	3,2	3,3	3,4
4,0	4,1	4,2	4,3	4,4

Operace nad poli

Příklad: tisk prvků horní trojúhelníkové matice.

```
void printUpTriangle(int a[R][R])
{
  for(int i = 0; i < R; i++)
  {
    for(int j = i; j < R; j++)
      printf("%d ", a[i][j]);
    printf("\n");
  }
}
```

Úkol: Definujte funkci pro tisk prvků **dolní** trojúhelníkové matice.

Operace s maticemi

- Rovnost matic.

  matice ***A*, *B*** stejného typu (m, n)
  $$\mathbf{A} = \mathbf{B} \Leftrightarrow \underset{\substack{i=1,2,\ldots,m \\ j=1,2,\ldots,n}}{\forall} \left(a_{ij} = b_{ij}\right)$$

Příklad: jsou si matice rovné?

```
bool areEqual(double a[R][S], double b[R][S])
{
  for(int r = 0; r < R; r++)
    for(int s = 0; s < S; s++)
      if (a[r][s] != b[r][s])
        return false;

  return true;
}
```

Operace s maticemi

- Součet matic.

  matice ***A*, *B*** stejného typu (*m*,*n*) $\mathbf{A}+\mathbf{B}=\left(a_{ij}+b_{ij}\right)$

  platí: $\mathbf{A}+\mathbf{B}=\mathbf{B}+\mathbf{A}$

$$\mathbf{A}+(\mathbf{B}+\mathbf{C})=(\mathbf{A}+\mathbf{B})+\mathbf{C}=\mathbf{A}+\mathbf{B}+\mathbf{C}$$

Příklad: sečte dvě matice a výsledek uloží do pole a.

```
void addMatrix(double a[R][S], double b[R][S])
{
  for(int r = 0; r < R; r++)
    for(int s = 0; s < S; s++)
      a[r][s] += b[r][s];
}
```

Operace s maticemi

- Násobení matice skalárem.

$$k\boldsymbol{A} = k\left(a_{ij}\right) = \left(ka_{ij}\right)$$

platí: $$\left(k_1 + k_2\right)\boldsymbol{A} = k_1\boldsymbol{A} + k_2\boldsymbol{A}$$

$$k\left(\boldsymbol{A} + \boldsymbol{B}\right) = k\boldsymbol{A} + k\boldsymbol{B}$$

Příklad: vynásobení matice skalárem.

```
void multConst(double a[R][S], double k)
{
  for(int r = 0; r < R; r++)
    for(int s = 0; s < S; s++)
      a[r][s] *= k;
}
```

Operace s maticemi

- Násobení matic.

  matice ***A*** typu (*m*, *p*), matice ***B*** typu (*p*, *n*)

$$\mathbf{AB} = \mathbf{C} \qquad \left(c_{ij}\right) = \left(\sum_{k=1}^{p} a_{ik} b_{kj}\right)$$

platí: $\boldsymbol{AB} \neq \boldsymbol{BA}$ - obecně neplatí komutativní zákon,

jestliže $\boldsymbol{AB} = \boldsymbol{BA}$ pak matice ***A*** a ***B*** jsou zaměnitelné,

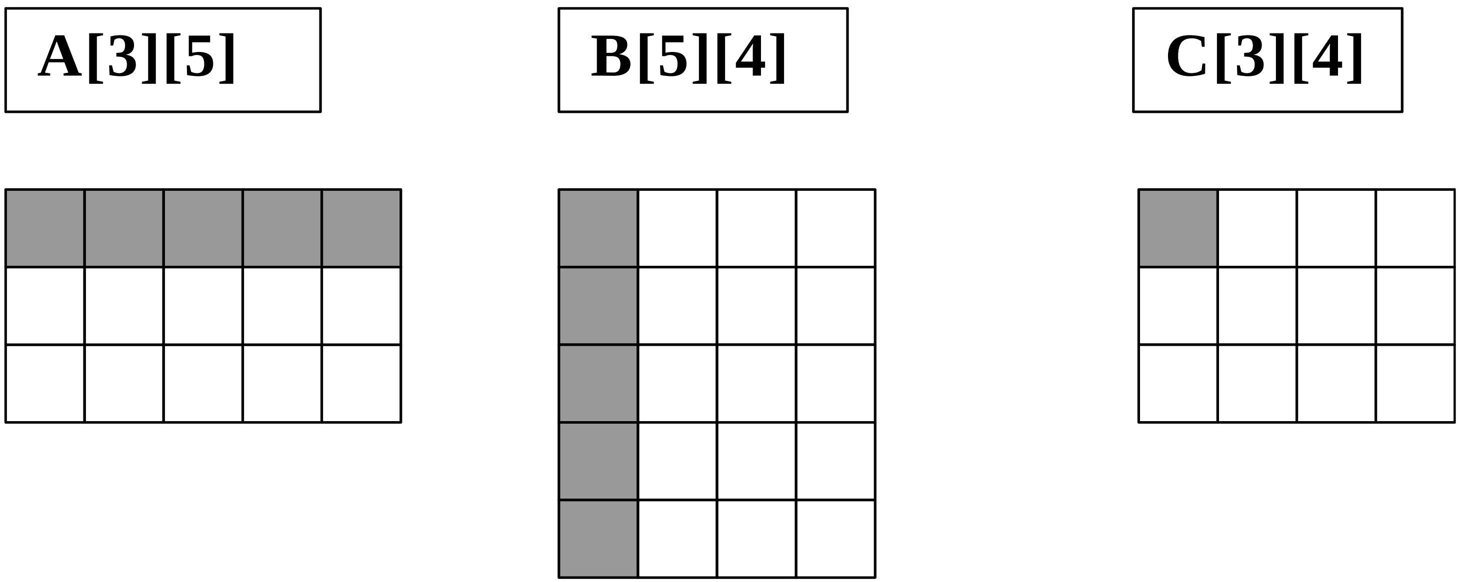
$$\boldsymbol{A}(\boldsymbol{BC}) = (\boldsymbol{AB})\boldsymbol{C} = \boldsymbol{ABC}$$ - asociativní zákon

$$\boldsymbol{A}(\boldsymbol{B} + \boldsymbol{C}) = \boldsymbol{AB} + \boldsymbol{AC}$$ - levý distributivní zákon

$$(\boldsymbol{A} + \boldsymbol{B})(\boldsymbol{C} + \boldsymbol{D}) = \boldsymbol{A}(\boldsymbol{C} + \boldsymbol{D}) + \boldsymbol{B}(\boldsymbol{C} + \boldsymbol{D})$$

Operace s maticemi

A (m,p), ***B*** (p,n), ***C*** (m,n)



Operace s maticemi

Příklad: součin matic.

```
void multMatrix(double a[M][P], double b[P][N],
                double c[M][N])
{
  for (int i = 0; i < M; i++)
  {
    for (int j = 0; j < N; j++)
    {
      c[i][j] = 0;
      for (int k = 0; k < P; k++)
        c[i][j] += a[i][k]*b[k][j];
    }
  }
}
```

Operace s maticemi

- Osmisměrka

K	A	L	T	J	S	H	O	D	A
L	L	P	U	K	L	T	O	A	T
A	K	T	A	A	K	A	A	R	R
S	A	A	N	L	A	K	P	E	A
A	R	P	O	V	P	T	O	K	K
R	H	O	M	O	L	I	C	E	A
K	O	L	S	P	E	K	E	S	R
O	R	A	O	C	A	A	L	T	P
S	P	O	K	V	S	T	I	A	A
M	A	T	K	A	F	T	K	A	T
A	I	A	K	O	S	T	K	A	Y

Operace s maticemi

Hledaná slova:

ALKA HORA JUTA KAPLE KARPATY KARTA KASA KAVKA KLAS KOSMONAUT KOST KROK LAPKA MATKA OKRASA OPAT OSMA PAKT PATKA PIETA POCEL POVLAK PROHRA SEKERA SHODA SOPKA TAKT TAKTIKA TLAK VOLHA

Operace s maticemi

A[7][6]

0,0	0,1	0,2	0,3	0,4	0,5
1,0	1,1	1,2	1,3	1,4	1,5
2,0	2,1	2,2	2,3	2,4	2,5
3,0	3,1	3,2	3,3	3,4	3,5
4,0	4,1	4,2	4,3	4,4	4,5
5,0	5,1	5,2	5,3	5,4	5,5
6,0	6,1	6,2	6,3	6,4	6,5

A [R][S]	**přírůstek**	**přírůstek**	
směr	**r index**	**s index**	**podmínka**
vlevo	0	-1	s-index=0
vpravo	0	+1	s-index=S-1
nahoru	-1	0	r-index=0
dolu	+1	0	r-index=R-1
vpravo-n	-1	+1	
vpravo-d	+1	+1	
vlevo-n	-1	-1	
vlevo-d	+1	-1	

Algoritmy pro práci s vektory a s maticemi

Kontrolní otázky

1. Jaký je zásadní rozdíl mezi vektorem a polem?
2. Jaký je rozdíl mezi skalárním a vektorovým součinem dvou vektorů? Lze sčítat vektory různých délek?
3. Jaká omezení klademe na vektory a matice při násobení konstantou?
4. K čemu slouží algoritmus nazývaný Eratostenovo síto? Vysvětlete stručně jeho princip.
5. Jaká omezení klademe na matice při jejich sčítání a násobení?
6. Která z verzí algoritmu pro reverzi řetězce (nebo pole) je efektivnější? Proč?

Úkoly k procvičení

1. Vytvořte program, který vypočítá, zda se na šachovnici lze dostat koněm ze zadané pozice na jinou zadanou pozici.
2. Vytvořte algoritmus pro transpozici matice obecných rozměrů.
3. Vytvořte funkci pro výměnu dvou zadaných řádků matice.
4. Vytvořte funkci pro výměnu dvou zadaných sloupců matice.
5. Vytvořte funkci pro přičtení zadaného řádku matice k jinému zadanému řádku stejné matice.
6. Vytvořte funkci pro přičtení zadaného sloupce matice k jinému zadanému sloupci stejné matice.
7. Vytvořte program pro řešení soustavy rovnic Gaussovou eliminační metodou.