Projekt č. 3 do předmětu IZP

Maticové operace

17. prosince 2006

Kamil Dudka, xdudka00@stud.fit.vutbr.cz
Fakulta informačních technologií
Vysoké Učení Technické v Brně

Obsah

1 Úvod

Tato dokumentace popisuje návrh a implementaci programu, který provádí různé operace s maticemi. V kapitole 2 se nachází stručná charakteristika rešeného problému, jeho analýza a z ní vyplývající možná řešení. Zvolené řešení je představeno v kapitole 3.

Výsledky provedených testů jsou zachyceny v kapitole 4. Implementace (kapitola 5) přímo vyplývá z provedené analýzy a návrhu řešení. Celkové řešení programu je zhodnoceno v závěru (kapitola 6). Dokumentace je vysázena systémem LaTeX.

2 Analýza problému

2.1 Zadání problému

Bylo požadováno vytvoření programu v jazyce C, který bude provádět s maticemi následující operace:

- Sčítání matic
- Součin matic
- Test monotónosti
- „Spirála“ – vypsání prvků matice po spirále
- „Terč“ – spočítání aritmetického průměru prvků na „kružnicích terče“
- „Žížala“ – cyklické posunutí prvků matice daným způsobem

Program musí být schopen načíst zdrojové matice ze souboru v zadaném formátu (pracuje se s maticemi obecných rozměrů). Výstupní data mají být směrovány na `stdout`.

2.2 Vstupní a výstupní data

Ze zadání vyplývá, že program musí být schopen provádět různé operace. Tyto operace je vhodné si rozdělit z hlediska vstupních dat na dvě skupiny:

- Binární operace s maticemi – sčítání a násobení matic
- Operace s jednou maticí – test monotónosti, spirála, terč, žížala

Vzhledem k tomu, že pro binární operace s maticemi je potřeba načíst ze souboru postupně dvě matice, je užitečné vytvořit si obecný algoritmus pro načítání matice ze souboru do paměti.

Podobně je potřeba vytvořit obecný algoritmus pro tisk matice na `stdout`. Dále je vhodné vytvořit si podprogramy pro tisk chybových hlášek a výpis nápovědy. Vzhledem k požadované efektivitě se budou vlastní operace s maticemi provádět pouze v paměti.

2.3 Nároky na paměť

Pracuje se s maticemi obecných rozměrů, zadané matice mohou být obrovské. Je tedy důležité zvolit algoritmy, které potřebují pro svou činnost minimum paměti. Dynamická alokace paměti pro matice je samozřejmostí.

Ideální je případ, kdy podprogram alokuje paměť pouze pro vstupní matice. V tomto paměťovém prostoru se pak provádí veškeré elementární operace s prvky matice. Na konci podprogramu zůstane v tomto prostoru pouze výsledek operace.

Tento algoritmus se dá bohužel jednoduše použít jen pro operace, jejichž výsledkem je matice stejných rozměrů jako matice zdrojová.

2.4 Datové struktury

Z předchozí kapitoly vyplývá, že je potřeba vytvořit obraz matice v paměti. Ukládání matice do paměti v podobě textového řetězce by bylo značně neefektivní. Jako datový typ pro uložení prvků matice bylo zvoleno **jednorozměrné pole typu int**. Druhou variantou je dvojrozměrné pole typu int, které se při zápise programu v jazyce C přehledněji indexuje, ale má jisté nevýhody:

- Náročnější algoritmy pro operace se sekvenčním přístupem (načítání prvků matice ze souboru, výpis prvků matice a sčítání,...)
- Náročnější algoritmy alokace a uvolňování paměti
- Problémy s fragmentací alokované paměti[1]

Bylo tedy zvoleno pole jendorozměrné, které se indexuje pomocí ukazatelové aritmetiky. Pro binární operace (viz. 2.2) je potřeba uložit do paměti více matic najednoou. Je tedy vhodné vytvořit datový typ, který bude matici

[1] Tyto problémy se projeví zejména, pokud je paměť alokována na stránkovacím souboru na pevném disku.

charakterizovat. Každá matice je v tomto případě určena rozměrem a adresou v paměti. Tyto položky jsem vložil do struktury. A pro každou matici jsem vytvořil jednu instanci této struktury.

3 Návrh řešení problému

V této kapitole je vysvětleno řešení daných operací na úrovni algoritmu. Matematické definice a jejich vysvětlení se vymyká rozsahu a zaměření této dokumentace.

3.1 Sčítání matic

Pro **sčítání matic** je potřeba načíst ze souboru dvě matice. Tyto matice musí mít stejný rozměr, jinak není operace sčítání definována. Algoritmus bude procházet prvky matice 1 a ke každému prvku bude postupně přičítat odpovídající prvek z matice 2. Po skončení podprogramu bude tedy výsledek operace v paměťovém prostoru matice 1, čímž jsou eliminovány nároky na paměť.

3.2 Násobení matic

Pro **násobení matic** je nutno opět načíst dvě matice. Aby byla operace násobení matic definována, musí mít první matice stejný počet sloupců jako druhá řádků. Výsledná matice bude mít stejný počet řádků jako matice 1 a stejný počet sloupců jako matice 2. To znamená, že výsledná matice bude mít stejné rozměry pouze v případě čtvercových matic. Je tedy nutné alokovat prostor pro výslednou matici.

Podprogram nejprve (na základě rozměrů zdrojových matic) alokuje paměť pro výslednou matici. Potom prochází prvky výsledné matice a ukládá do nich odpovídající čísla. Každý prvek výsledné matice je vypočítán jako **skalární součin** řádkového vektoru matice 1 a sloupcového vektoru matice 2.

3.3 Test monotónosti

Test monotónosti naopak vyžaduje načtení pouze jedné matice a je definován pro jakoukoliv matici. Podprogram nealokuje žádnou paměť pro matice ani vstupní matici nijak nemění. Prochází nejprve její řádky a potom sloupce. Porovnává vždy dva sousední prvky, přičemž si nastavuje příznak, je-li posloupnost **neklesající**, **nerostoucí**, popř. obojí.

Jakmile program narazí na první neshodu, vyhodnotí matici jako **nemonotóní** a další prvky matice už neprochází, čímž je eliminována časová složitost algoritmu. Naopak, projde-li cyklus až na konec a přitom na neshodu nenarazí, prohlásí matici za **monotónní**. Časová složitost algoritmu je tedy nejvyšší pro matice, které jsou monotónní.

3.4 Spirála

Požadovaný výsledek operace **spirála** je zřejmý ze zadání úlohy. Algoritmus je založen na principu **obecného průchodu maticí**. K aktuální pozici (dané řádkem a sloupcem aktuálního prvku) je přičítán vektor udávající směr pohybu maticí. V tomto případě se jedná o vektor jednotkový a jeho směr se periodicky mění.

V prvním závitu spirály je změna směru podmíněna dosažením okraje matice. S dalšími závity se tento okraj postupně „ořezává“ o prvky, které již byly vypsány. Tím se poloměr otáčení stále zmenšuje a aktuální pozice se přibližuje ke středu spirály. Jakmile jsou vypsány všechny prvky matice, cyklus se přeruší.[2]

3.5 Terč

Algoritmus **terče** je podobný jako u spirály. Opět využívá principu **obecného průchodu maticí**. Rozdíl je v tom, že jako první se načte prvek v levém horním rohu matice. Po dosažení „okraje“ se směr periodicky mění. Jakmile jsou načteny všechny prvky „kružnice“ daného poloměru, spočítá se jejich průměr a vypíše se na `stdout`.

Při tomto průchodu je první prvek načten dvakrát a je tedy nutné jej z množiny načtených prvků vyřadit. Načtené prvky se ve skutečnosti nikam neukládají, ale zaznamenává se pouze jejich suma a počet. Pro matici „lichého“ rozměru skončí cyklus dříve, než se vypíše průměr naposledy načtených prvků. Tento průměr je tedy potřeba vypsat dodatečně po skončení cyklu.

3.6 Žížala

Operace **žížala** je mírně komplikovanější než předcházející operace. Počet posunutí a jejich směr závisí na hodnotě prvního prvku zdrojové matice. To znamená, že tyto hodnoty nejsou známy v době překladu, což zvyšuje nároky

[2] Samotná podmínka je samozřejmě složitější – to jedáno tím, že program si nikam neukládá, které prvky už vypsal a které ne, resp. ani nepočítá vypsané prvky.

na obecnost algoritmu. Tetno komplexní problém jsem si rozdělil na dva podproblémy, resp. podprogramy.

První podprogram zajišťuje výpočet potřebného počtu posunutí a jeho směr. Dále obsahuje jeden cyklus typu `FOR`, který opakovaně volá druhý podprogram, tolikrát, kolikrát je potřeba elementární posunutí provést. Při posunutí se musí projít všechny prvky matice, což je časově náročná operace. Je tedy zřejmé, že počet těchto posunů musí být omezen na minimum. Toho se dosáhne tím, že se od počtu posunutí odečte celý počet násobků počtu prvků matice.[3]

Posunutí ve tvaru žížaly o jeden prvek daným směrem provádí druhý podprogram. Je využit podobný algoritmus jako pro spirálu nebo terč. Při průchodu se vždy vymnění obsah aktuálního prvku s obsahem pomocné proměnné `tmp`. Tělo cyklu, provádějícího posunutí, je stejné pro oba směry. Liší se pouze inicializace tohoto cyklu. Pro směr vpřed se prvky prochází shora dolů. Pro směr vzad se prvky prochází zdola nahoru.

4 Specifikace testů

Vzhledem k rozsahu této dokumentace jsou uvedeny pro každou operaci pouze jedny testovací data. Ve skutečnosti bylo testů provedeno mnohem více. Rozpoznání parametrů, předaných z příkazové řádky, bylo v tomto programu triviální. Testy rozpoznávání parametrů proto rovněž nejsou zahrnuty do dokumentace.

4.1 Sčítání matic

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 1 & 2 & 3 & 4 \\ 4 & 5 & 1 & 2 & 3 \\ 3 & 4 & 5 & 1 & 2 \end{pmatrix} + \begin{pmatrix} 5 & 4 & 3 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ -2 & -2 & -2 & -2 & -2 \end{pmatrix} = \begin{pmatrix} 6 & 6 & 6 & 6 & 6 \\ 5 & 1 & 2 & 3 & 4 \\ 5 & 6 & 2 & 3 & 4 \\ 1 & 2 & 3 & -1 & 0 \end{pmatrix}$$

Test proběhl bez problémů.

[3] Ikdyž zní princip optimalizace složitě, jedná se o triviální operaci.

4.2 Násobení matic

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 2 & 1 \\ -1 & -1 & -1 & -1 \\ -2 & -1 & 0 & 1 \\ -1 & 0 & 1 & 2 \end{pmatrix} \cdot \begin{pmatrix} -1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix} = \begin{pmatrix} 1 & 5 & -1 \\ -1 & 5 & 6 \\ 0 & -2 & 1 \\ 1 & -1 & 4 \\ 1 & 1 & -3 \end{pmatrix}$$

Test proběhl bez problémů.

4.3 Test monotónosti

Uvedu příklady dvou matic – jedna je monotóní (A) a druhá nemonotóní(B):

$$A = \begin{pmatrix} 5 & 4 & 3 & 2 & 1 \\ 6 & 5 & 4 & 3 & 2 \\ 7 & 6 & 5 & 4 & 3 \\ 7 & 7 & 6 & 5 & 4 \end{pmatrix}, B = \begin{pmatrix} 0 & 1 & 3 & 4 & 5 \\ 0 & 1 & 3 & 2 & 3 \\ 2 & 2 & 2 & 2 & 2 \\ 3 & 2 & 2 & 1 & 0 \end{pmatrix}$$

Test proběhl bez problémů.

4.4 Spirála

$$\begin{pmatrix} 10 & 9 & 8 & 7 \\ 11 & 16 & 15 & 6 \\ 12 & 13 & 14 & 5 \\ 1 & 2 & 3 & 4 \end{pmatrix} \longmapsto 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15\ 16$$

Test proběhl bez problémů.

4.5 Terč

$$\begin{pmatrix} 1 & 18 & 17 & 16 & 15 \\ 2 & 1 & 2 & 3 & 14 \\ 3 & 10 & 1 & 4 & 13 \\ 4 & 9 & 2 & 5 & 12 \\ 5 & 8 & 7 & 6 & 11 \\ 6 & 7 & 8 & 9 & 10 \end{pmatrix} \longmapsto 9.50000\ 5.50000\ 1.50000$$

Test proběhl bez problémů.

4.6 Žížala

$$\begin{pmatrix} 2 & 0 & 1 & 2 \\ 3 & 4 & 5 & 6 \\ 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 \end{pmatrix} \longmapsto \begin{pmatrix} 12 & 11 & 2 & 0 \\ 5 & 6 & 2 & 1 \\ 4 & 3 & 7 & 8 \\ 13 & 14 & 10 & 9 \end{pmatrix}$$

Test proběhl bez problémů.

5 Popis řešení

5.1 Ovládání programu

Program je řešen jako konzolová aplikace, má tedy pouze textové ovládání. Je volán s jedním parametrem, který specifikuje prováděnou operaci. Za tímto parametrem následují názvy souborů matic. Kolik názvů je nutné zadat, záleží na konkrétní operaci (viz. 2.2). Varianty prvního parametru jsou uvedeny v tabulce 1.

Parametr	Provedená operace
`-h`	Vypíše nápovědu
`-add`	Sečte matice
`-mult`	Vynásobí matice (v pořadí, jak jsou zadány)
`-mono`	Provede test monotónosti
`-spiral`	Vypíše prvky matice ve tvaru spirály
`-dartboard`	Spočítá aritmetický průměr prvků na „kružnicích“ terče
`-worm`	Provede „žížalí“ transformaci matice

Tabulka 1: Varianty prvního parametru

Názvy souborů matic musí být platné názvy souborů podle pravidel operačního systému, ve kterém je program spuštěn. Matice budou uloženy v textových souborech v přesně zadaném formátu. Na prvním řádku je vždy dvojice čísel, udávající rozměr matice (v pořadí řádky, sloupce). Následují vlastní prvky matice, které jsou od sebe odděleny bílými znaky. Program nijak nerozlišuje, jak jsou čísla rozdělena na řádcích souboru.[4]

Rozměry matice jsou načítány jako `unsigned int`. Rozsah hodnot tohoto typu záleží na platformě. Aby se však matice úspěšně načetla, musí být k dispozici dostatek paměti pro všechny její prvky. Vlastní prvky matice jsou

[4]Pro ruční vytváření souboru je samozřejmě vhodné zapisovat řádky matice na samostatné řádky souboru. Zápis je potom mnohem názornější.

typu `int`, hodnoty prvků mohou být tedy i záporné. Za platný je považován soubor, který obsahuje nejméně tolik prvků, kolik udává rozměr matice.

Pokud se vyskytne chyba při rozpoznávání parametrů, nebo během načítání matice, je vypsána chybová hláška na `stderr`. Během výpočtu nejsou vypisovány žádné informace, ani chybové hlášky. Pokud operace není pro zadané matice definována vytiskne se `#` na `stdout`.[5]

Pokud je výsledkem operace matice, je vypsána na `stdout` ve stejném tvaru, jako je požadován pro vstupní matice. Při **testu monotónosti** se vytiskne `1`, pokud je matice monotónní nebo `#`, pokud matice není monotónní. Výsledkem operací **spirála** a **terč** je posloupnost čísel, která je rovněž vypsána na `stdout`.

5.2 Vlastní implementace

Aby byl hlavní program přehlednější, uložil jsem funkce pro výpis nápovědy a chybových hlášek do samostatného souboru `proj3.h`. Rozpoznání prvního parametru, kontrola počtu parametrů a obsluha chybových stavů jsou implementovány ve funkci `main`, ze které jsou přímo nebo nepřímo volány všechny ostatní funkce. Na konci funkce main (po zveřejnění výsledku operace) se provede dealokace dynamicky alokované paměti pro matice.[6]

Funkce `ctiMatici` zajišťuje načtení matice ze souboru do paměti, přičemž si sama alokuje potřebný prostor. Funkce `tiskniMatici` tiskne matici uloženou v paměti na `stdout` v požadovaném tvaru. Ostatní funkce zajišťují požadované operace s maticemi. Vzhledem k tomu, že během výpočtu nemají být vypisovány žádné hlášky, není při výpočtech nijak kontrolováno přetečení datového typu `int`.

6 Závěr

Program byl otestován se všemi navrženými testovacími hodnotami – všechny testy proběhly bez problémů. Požadavky na formát vstupních a výstupních dat byly přesně dodrženy, takže může být program bezproblémově používán spolu s dalšími programy ve skriptech nebo jiných programech. Hlavním cílem tohoto programu bylo, naučit se pracovat s maticemi v jazyce C. Přesto nejsou implementované oprarace s maticemi zcela samoúčelné – naleznou upaltnění např. v kartografii nebo počítačové grafice.

[5] Křížek se také vytiskne, pokud se při násobení matic nepodaří alokovat paměť pro výslednou matici.

[6] Pokud dojde k chybě během načítání vstupních dat, tak se dealokace provede okamžitě a program se ukončí.

A Metriky kódu

Počet souborů:	2 soubory
Počet řádků zdrdojového textu:	686 + 60 řádků
Velikost statických dat:	6603 bytů
Velikost spustitelného souboru:	13780 bytů (Linux; bez ladících informací)