Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут»

Контрольна робота

з дисципліни «Декларативне програмування»

тема: «пакет операцій над матрицями»

Виконав:
студент групи ІП-32
Ковальчук О. М.
Перевірив:
доцент кафедри АСОІУ
Баклан І. В.

Київ 2015

Теоретичні відомості

Представлення матриць

Матриця — просто прямокутний масив даних, упорядкований по рядках та стовпцях. Більшість мов програмування, таких як C# та Java, мають вбудовану підтримку для двовимірних масивів; у той же час інші мови, такі як Clojure, використовують представлення у вигляді масиву масивів. Матриці представляються за допомогою векторів, елементами яких є інші вектори. Матриці також підтримують декілька арифметичних операцій, таких як додавання або множення. Майже кожна мова програмування має принаймні одну бібліотеку для роботи з лінійною алгеброю. Clojure пішов ще далі, дозволяючи вибрати одну із існуючих бібліотек, які наждають єдиний стандартизований інтерфейс для роботи з матрицями.

У Clojure матриця це просто вектор векторів. Це означає, що вона представлена у вигляді вектора, елементами якого є інші вектори. Вектор — масив елементів, для якого час пошуку елементу є майже константним. Тим не менш, в математичному контексті вектори — це матриці, які складаються з одного рядка або стовпця.

В даній роботі будемо вважати будь-який вектор векторів, елементами якого є числа, матрицею. Для створення матриці із довільної колекції колекцій, що має правильну конфігурацію, скористаємося функцією matrix:

```
(matrix [[1 2 3] [4 5 6]])
=> [[1 2 3] [4 5 6]]
(matrix '((1 2 3) (4 5 6)))
=> [[1 2 3] [4 5 6]]
```

Матриця A складається із елементів $a_{i,j}$, де *i* — індекс рядка, а *j* — індекс стовпця матриці. Математично ми можемопредставити матрицю наступним чином:

$$A_{m\times n}=[a_{i,j}]$$

Для того, щоб перевірити, чи є змінна матрицею, ми використовуємо функцію matrix?:

```
(matrix? [[1 2 3] [4 5 6]])
=> true
(matrix? [[1 2 3] [4 5 ]])
=> false
(matrix? 0)
=> false
```

Важливим атрибутом матриці є її розмір. Визначити кількість рядків матриці ми

можемо за допомогою функції row-count. Вона всього-навсього повертає довжину вектора векторів. Для цих же цілей ми можемо використати стандартну функцію count із пакета clojure.core. Подібним чином column-count повертає кількість стовпців у матриці. Також було додатково визначено метод count, який повертає кількість усіх елементів матриці.

```
(count [[1 2 3] [4 5 6]])
=> 6
(row-count [[1 2 3] [4 5 6]])
=> 2
(clojure.core/count [[1 2 3] [4 5 6]])
=> 2
(column-count [[1 2 3] [4 5 6]])
=> 3
```

Для отримання елементу матриці за індексами, ми можемо скористатися функцією get:

```
(get [[1 2 3] [4 5 6]] 0 1)
=> 2
(get [[1 2 3] [4 5 6]] 1 2)
=> 6
```

Зауважимо, що усі елементи проіндексовані починаючи з 0 у Clojure, що не співпадає з математичною нотацією, де нумерація починається із 1.

Генерація матриць

Матриця називається квадратною, якщо кількість її рядків дорівнює кількості її стовпців. Щоб згенерувати квадратну матрицю, ми можемо скористатися функцією repeat:

```
(defn anxmatrix.core/square-mat [n e]
    (let [repeater #(repeat n %)] (matrix (-> e repeater repeater))))
```

Також у математиці часто використовується такий вид матриць, як одиничні матриці (identity matrices). Одинична матриця — матриця, в якій елементи головної діагоналі дорівнюють 1, а решта елементів — 0. Визначимо функцію для генерації одиничної матриці заданої розмірності:

```
(defn anxmatrix.core/identity-matrix [n]
  (let [row (conj (repeat (dec n) 0) 1)]
```

```
    (vec (for [i (range 1 (inc n))]
           (vec (reduce conj (drop i row) (take i row)))))))
```

Операції над матрицями

Операції над матрицями не підтримуються у Clojure. Для реалізації їх, створимо функції з відповідними іменами:

- matrix-mult — отримати добуток двох матриць
- matrix-add — обчислити суму двох матриць
- matrix-substract — обчислити різницю двох матриць
- matrix-eq — порівняння двох матриць

Дві матриці A і B є рівними тоді, коли виконується наступна тотожність:

$$Let\ A_{m\times n}=[a_{i,j}];B_{p\times q}=[b_{i,j}]$$
$$(A=B)\Leftrightarrow(m=p,n=q,a_{i,j}=b_{i,j}\ \forall\ i,j)$$

Наведемо просте, втім елегентне, рішення:

```
(defn matrix-eq [ma mb]
    (and (= (count ma) (count mb)) (reduce #(and %1 %2) (map = ma mb))))

(mat-eq [[1 2 3][4 5 6]] [[1 2 3][4 5 6]])
=> true
(mat-eq [[1 2 3][4 5 6]] [[1 2 3][4 6 5]])
=> false
```

Щоб додати дві матриці, необхідно, щоб матриці були однакової розмірності. Сума матриць — матриця, елементи якої є сумами відповідних елементів вхідних матриць:

$$[c_{i,j}]=[a_{i,j}+b_{i,j}]$$

```
(defn anxmatrix.core/matrix-add [a b] (mapv (fn [ai bi] (mapv + ai bi)) a b))
```

Множення матриць — це бінарна операція, яка використовуючи дві матриці, утворює нову матрицю, яка називається добутком матриць. Дійсні або комплексні числа множаться відповідно до правил елементарної арифметики. З іншого боку, матриці є масивами чисел, тому існують різні способи визначити добуток матриць. Таким чином, загалом термін «матричне множення» означає різні способи перемноження матриць. Ключовими особливостями будь-якого матричного множення є: кількість рядків і стовпців, в початкових матрицях, і правило, як елементи матриць утворюють

нову матрицю.

$$Let\ A_{m\times n}=[a_{i,j}];B_{p\times q}=[b_{i,j}]$$

$$Then\ \ C_{m\times q}=[c_{i,j}]=[\sum_{r=1}^{n} a_{i,r}b_{r,j}(i=1,2,\ldots,m;\ j=1,2,\ldots,q)]$$

Добуток матриць AB складається з усіх можливих комбінацій скалярних добутків вектор-рядків матриці A і вектор-стовпців матриці B. Елемент матриці AB з індексами i, j є скалярний твір i-го вектор-рядка матриці A і j-го вектор-стовпця матриці B.

Транспонування матриці

Транспонована матриця — матриця A^T , що виникає з матриці A в результаті унарної операції транспонування: заміни її рядків на стовпчики.

```
(defn anxmatrix.core/transpose [s] (apply mapv vector s))

(transpose [[1 2 3] [4 5 6]])
=> [[1 4] [2 5] [3 6]]
```

Обчислення визначника

Визна́чник або детермінáнт — це число; вираз складений за певним законом з n^2 елементів квадратної матриці. Одна з найважливіших характеристик квадратних матриць.

Для квадратної матриці розміру $n\times n$ визна́чник є многочленом степеня n від елементів матриці, і є сумою добутків елементів матриці зі всіма можливими комбінаціями різних номерів рядків і стовпців (в кожному із добутків є рівно по одному елементу з кожного рядка і кожного стовпця). Кожному добутку приписується знак плюс чи мінус, в залежності від парності перестановки номерів.

$$det(A)=|A|=\begin{vmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & & & a_{2n} \\ \vdots & & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{vmatrix}=\sum_{\pi\in S_n} sgn(\pi)(a_{1\pi(1)}\cdot a_{2\pi(2)}\cdot\ldots\cdot a_{n\pi(n)}),$$

$$Where\ \pi\ is\ a\ permutation\ of\ set\ (1,2,\ldots,n)$$

Реалізація

```
(ns anxmatrix.core
  (:require [clojure.core :as cc])
  (:require [clojure.math.combinatorics :as combo])
  (:import java.lang.UnsupportedOperationException))

(defn- anxmatrix.core/inv [left right n]
  (loop [l left r right out [] inv n]
    (if (or (empty? l) (empty? r))
      {:seq (concat out l r) :inv inv}
      (let [x (first l) y (first r)]
        (if (<= x y)
          (recur (rest l) r (conj out x) inv)
          (recur l (rest r) (conj out y) (+ inv (count (filter #(> % y) l)))))))))

(defn- anxmatrix.core/count-inv
  ([coll] (count-inv coll 0))
  ([coll n]
   (if (> (count coll) 1)
     (let [[left right] (split-at (quot (count coll) 2) coll)
           l (count-inv left n)
           r (count-inv right n)]
       (inv (:seq l) (:seq r) (+ (:inv l) (:inv r))))
     {:seq coll :inv n})))

(defn anxmatrix.core/transpose [s] (apply mapv vector s))

(defn- anxmatrix.core/nested-for [f x y] (mapv (fn [a] (mapv (fn [b] (f a b)) y)) x))

(defn anxmatrix.core/matrix-mult [a b] (nested-for (fn [x y] (reduce + (map * x y))) a (transpose b)))

(defn anxmatrix.core/matrix-add [a b] (mapv (fn [ai bi] (mapv + ai bi)) a b))

(defn anxmatrix.core/matrix-substract [a b] (mapv (fn [ai bi] (mapv - ai bi)) a b))

(defn anxmatrix.core/matrix-eq [ma mb] (and (= (count ma) (count mb)) (reduce #(and %1 %2) (map = ma mb))))

(defn anxmatrix.core/matrix? [x]
  (true?
    (and
      (vector? x)                                    ;; vector
      (seq x)                                        ;; not empty
      (every? vector? x)                             ;; consists of vectors
      (every? (fn [xi] (every? number? xi)) x)       ;; which consists of numbers
      (every? seq x)                                 ;; and not empty too
      (apply = (map count x))                        ;; and have equal lengths
      )))

(defn anxmatrix.core/matrix [x]
  (cond
    (matrix? x) x
    (and (coll? x) (seq x) (every? number? x)) (vector (vec x))
    (and (coll? x) (every? coll? x)) (if (matrix? (mapv vec x)) (mapv vec x) nil)
    :else (throw (IllegalArgumentException. "cannot make matrix"))))

(defn anxmatrix.core/count [x]
  (if-not
    (matrix? x)
    (cc/count x)
    (* (cc/count x) (count (get x 0)))))

(defn anxmatrix.core/row-count [x] (if-not (matrix? x) (throw (UnsupportedOperationException. "row-count not
supported on this type")) (cc/count x)))

(defn anxmatrix.core/column-count [x] (if-not (matrix? x) (throw (UnsupportedOperationException. "column-count not
supported on this type")) (cc/count (get x 0))))

(defn anxmatrix.core/square-mat [n e] (let [repeater #(repeat n %)] (matrix (-> e repeater repeater))))
```

```
(defn anxmatrix.core/identity-matrix [n]
  (let [row (conj (repeat (dec n) 0) 1)]
    (vec (for [i (range 1 (inc n))]
           (vec (reduce conj (drop i row) (take i row)))))))

(defn anxmatrix.core/get [m r c]
    (if-not (matrix? m)
        (throw (UnsupportedOperationException. "anxmatrix.core/get not supported on this type"))
        (cc/get (cc/get m r) c)))

(defn anxmatrix.core/set [m r c v]
    (if-not (matrix? m)
        (throw (UnsupportedOperationException. "anxmatrix.core/set not supported on this type"))
        (assoc m r (assoc (cc/get m r) c v))))

(defn anxmatrix.core/det [m]
    (if-not (and (matrix? m) (= (row-count m) (column-count m)))
        (throw (UnsupportedOperationException. "anxmatrix.core/det not supported on this type")))
        (let [o (range (row-count m))]
            (reduce +
                (map (fn [p] (* (Math/pow -1 (cc/get (count-inv p) :inv))
                    (reduce * (map (fn [r c] (get m r c)) o p)))) (combo/permutations o)))))
```

Використання

```
(matrix? 1)
=> false
(matrix? [[1 2 3] [4 5 6]])
=> true
(matrix? [[1 2 3] [4 5 6 7]])
=> false
(matrix? [4 5 6 7])
=> false
(transpose [[1 2 3] [4 5 6]])
=> [[1 4] [2 5] [3 6]]
(matrix-mult [[1 0 0][0 1 0]] [[4 4 4 4][4 4 4 4][4 4 4 4]])
=> [[4 4 4 4] [4 4 4 4]]
(matrix-mult [[1 0 0][0 0 0]] [[4 4 4 4][4 4 4 4][4 4 4 4]])
=> [[4 4 4 4] [0 0 0 0]]
(matrix-add [[1 2 3 4][5 6 7 8]] [[8 7 6 5] [4 3 2 1]])
=> [[9 9 9 9] [9 9 9 9]]
(matrix-substract [[1 2 3 4][5 6 7 8]] [[8 7 6 5] [4 3 2 1]])
=> [[-7 -5 -3 -1] [1 3 5 7]]
(matrix-eq [[1 2 3 4][5 6 7 8]] [[8 7 6 5] [4 3 2 1]])
=> false
(matrix-eq [[1 2 3 4][5 6 7 8]] [[1 2 3 4] [5 6 7 8]])
=> true
(square-mat 4 5)
=> [[5 5 5 5] [5 5 5 5] [5 5 5 5] [5 5 5 5]]
(identity-matrix 4)
=> [[1 0 0 0] [0 1 0 0] [0 0 1 0] [0 0 0 1]]
(det [[1 0 0][0 1 0][0 0 1]])
=> 1.0
(det [[1 0 1][0 1 0][1 0 1]])
=> 0.0
```