

Wykonał: Radosław Smoter

Grupa: 14

Nr: 27

Numer zadania: 5

Przykład: 62

Prowadzący: Prof. dr hab. inż.
Volodymyr Samotyj

Politechnika Krakowska

Wydział Inżynierii Elektrycznej i Komputerowej

Sprawozdanie: Wstęp do Programowania

Spis treści

Polecenie.....	1
(1) Treść.....	1
(2) Warunki sortowania.....	1
(3) Warunki obliczania ,.....	1
Kod programu.....	2
Wyniki.....	7
Opis programu.....	8
Wnioski.....	9

Polecenie

(1) Treść

Napisać kod programu sortowania macierzy $A = \{a_{i,j}\}$ metodą bąbelkową. Dla posortowanej macierzy obliczyć wartość funkcji $F(f_i(a_{i,j}))$. Algorytm sortowania oraz obliczania funkcji $f_i(a_{i,j})$ zapisać w postaci funkcji. Macierz $A = \{a_{i,j}\}$ odczytać z pliku tekstowego. Wyniki sortowania oraz wartości funkcji $f_i(a_{i,j})$, $F(f_i(a_{i,j}))$ zapisać do pliku tekstowego.

(2) Warunki sortowania

Posortować elementy wierszy od maksymalnego do minimalnego.

(3) Warunki obliczania $f_i(a_{i,j})$, $F(f_i(a_{i,j}))$

$f_i(a_{i,j})$ - minimalny element każdej kolumny pod diagonalą główną

$$F(f_i(a_{i,j})) = \prod f_i(a_{i,j})$$

Kod programu

```
/**
 * @file main.c
 * @author Radoslaw Smoter (radoslaw.smoter@student.pk.edu.pl)
 * @version 0.1
 * @date 2021-12-17
 *
 * @copyright Copyright (c) 2021
 */

/*
 * Read a matrix M from a file.
 * Sort elements of the matrix rows in descending order, denote as M_sort.
 * Sorting algorithm: bubble sort.
 * Select the lowest value of each column of the matrix from under the main diagonal,
denote as f_j(a_ij).
 * Calculate product F of f_j(a_ij) as F(f_j(a_ij)).
 * Save the results of M_sort, f_j(a_ij), F(f_j(a_ij)) to a file.
 */

#include <stdio.h>
#include <math.h>

void importMatrix(double[][10]);
void sortMatrix(double[][10]);
void selectF(double[][10], double[][10]);
void selectLowest(double[][10], double[]);
double product(double[]);
void saveResults(double[][10], double[], double);

int main(void)
{
    /* Space for matrix 10x10 */
    double matrix[10][10];

    importMatrix(matrix);
    sortMatrix(matrix);

    /* Get cols below main diagonal, populate remaining with nan */
    /* Cols below diagonal matrix */
    double diagonalMatrix[10][10];
    selectF(matrix, diagonalMatrix);

    /* Lowest value from each diagonal col */
    double min[10];
    selectLowest(diagonalMatrix, min);

    /* Product of diagonal cols' min */
    double productVal = product(min);

    /* Save resultst to txt file */
    saveResults(matrix, min, productVal);
}
```

```
    return 0;
}
```

```
/* Import matrix from "matrix.txt" file */
void importMatrix(double matrix[][10]) {
    char *filename = "matrix.txt";
    char *filemode = "r";
    FILE *file = fopen(filename, filemode);

    /* Single number from the file */
    double fp;
    /* Is EOF encountered */
    int isEOF;
    /* Row counter */
    int i = 0;
    /* Column counter */
    int j = 0;

    do {
        isEOF = fscanf(file, "%lf", &fp);

        /* Go to a new row */
        if (j >= 10)
        {
            j = 0;
            i++;
        }
        /* Don't assign EOF */
        if (isEOF != EOF)
            matrix[i][j] = fp;

        /* Each new entry */
        j++;
    } while (isEOF != EOF && (i*10 + j) < 100);

    fclose(file);
}
```

```
/* Exchange values of two variables */
void exch(double *a, double *b) {
    double temp = *a;
    *a = *b;
    *b = temp;
}
```

```
/* Sort an array according to monotonicity, ascending order - 1, descending - 0 */
void bubbleSort(double arr[], int n, int monotonicity) {
    int swapCounter = 1;

    /* Do until no more swaps have been encountered */
    while(swapCounter) {
        swapCounter = 0;

        for (int i = 0; i < n-1; i++)
```

```

{
    /* Ascending order */
    if (monotonicity) {
        if (arr[i] < arr[i + 1]) {
            exch(&arr[i], &arr[i + 1]);
            swapCounter++;
        }
        /* Descending order */
    } else {
        if (arr[i] > arr[i + 1]) {
            exch(&arr[i], &arr[i + 1]);
            swapCounter++;
        }
    }
}
}
}
}

```

```

/* Sort all rows of a matrix */
void sortMatrix(double arr[][10]) {
    for (int i = 0; i < 10; i++)
        bubbleSort(arr[i], 10, 1);
}

```

```

/* Select entries from under the main diagonal of the matrix */
void selectF(double arr[][10], double diagonal[][10]) {
    /* Diagonal rows */
    int m = 0;
    /* Diagonal columns */
    int n = 0;
    for (int i = 0; i < 10; i++) {
        for (int j = 0; j < 10; j++) {
            if (i > j) diagonal[m][n] = arr[i][j];
            /* Populate with nans to avoid trash values */
            else diagonal[m][n] = NAN;
            n++;
        }
        m++;
        n = 0;
    }
}

```

```

/* Select a column from a matrix */
void selectCol(double arr[][10], double col[], int n) {
    int j = 0;
    for (int i = 0; i < 10; i++)
        col[j++] = arr[i][n];
}

```

```

/* Select the lowest values in the array arr, return them in the array max */
void selectLowest(double arr[][10], double min[]) {
    /* Placeholder for each column */
    double col[10];
}

```

```

for (int i = 0; i < 10; i++) {
    selectCol(arr, col, i);
    bubbleSort(col, 10, 0);

    int j = 0;
    while (isnan(col[j])) j++;

    /* Select the highest values */
    min[i] = col[j];
    /* Whole column is NaNs; no real values in a column */
    if (j == 10) min[i] = NAN;
}
}

/* Return product of the given array */
double product(double arr[]) {
    /* Product */
    double p = 1;
    for (int i = 0; i < 10; i++)
        /* Ensure none of the entries are NAN */
        if (!isnan(arr[i]))
            p *= arr[i];
    return p;
}

/* Save results of the task to a txt file */
void saveResults(double matrix[][10], double max[], double productVal) {
    char *filename = "results.txt";
    FILE *file = fopen(filename, "w");

    if (file != NULL) {
        /* Sorted matrix entires */
        fprintf(file, "%s", "Sorted matrix:\n");
        for (int i = 0; i < 10; i++) {
            for(int j = 0; j < 10; j++)
                fprintf(file, "%8.1f", matrix[i][j]);
            fprintf(file, "%c", '\n');
        }

        /* Entries from under main diagonal, highest values */
        fprintf(file, "%s", "Under main diagonal, col min:\n");
        for (int i = 0; i < 10; i++)
            fprintf(file, "%8.1f", max[i]);

        /* Product value */
        fprintf(file, "%s", "\nProduct value:\n");
        fprintf(file, "%16.6e", productVal);

        fclose(file);
    }
    /* Error */
    else {
        fprintf(stderr, "Error: File did not open correctly.\n");
        return;
    }
}

```

}
}

Wyniki

Sorted matrix:

194.0	187.0	173.0	159.0	158.0	105.0	77.0	73.0	-150.0	-172.0
118.0	41.0	28.0	26.0	-33.0	-39.0	-86.0	-98.0	-104.0	-126.0
128.0	92.0	46.0	11.0	-46.0	-72.0	-110.0	-119.0	-152.0	-197.0
191.0	147.0	105.0	100.0	5.0	-44.0	-90.0	-100.0	-113.0	-169.0
156.0	156.0	86.0	57.0	30.0	16.0	-98.0	-116.0	-155.0	-182.0
189.0	139.0	-78.0	-83.0	-88.0	-117.0	-129.0	-169.0	-184.0	-198.0
113.0	93.0	90.0	70.0	63.0	51.0	-37.0	-58.0	-168.0	-187.0
181.0	144.0	134.0	83.0	58.0	36.0	-19.0	-27.0	-47.0	-171.0
187.0	175.0	150.0	95.0	29.0	-42.0	-47.0	-146.0	-153.0	-172.0
173.0	146.0	87.0	19.0	-15.0	-47.0	-81.0	-164.0	-167.0	-170.0

Under main diagonal, col min:

113.0	92.0	-78.0	-83.0	-88.0	-47.0	-81.0	-164.0	-167.0	nan
-------	------	-------	-------	-------	-------	-------	--------	--------	-----

Product value:

-6.175396e+17

Opis programu

Główna funkcja programu ma zarezerwowaną tablicę `matrix[][]` na wartości pobierane z pliku „matrix.txt” za pomocą funkcji `importMatrix()`. Następnie `matrix[][]` jest sortowana funkcją `sortMatrix()`, która sortuje ją według wartości wierszy w manierze malejącej. Tablica `diagonalMatrix[][]` przechowuje wartości zwracane przez parametr przez `selectF()` do tej tablicy, które są wartościami spod diagonali głównej `matrix[][]`. Do tablicy `min[]`, funkcja `selectLowest()` zwraca przez parametr wartości najniższe kolumn tablicy `diagonalMatrix[][]`. Następnie do `productVal`, funkcja `product()` zwraca jako wynik iloczyn wartości `min[]`. Wyniki, tj. `matrix[][]` (po posortowaniu), `min[]` oraz `productVal` zostają zapisane do pliku tekstowego „results.txt” przez funkcję `saveResults()`.

Funkcja `importMatrix()` zwraca do tablicy `matrix[][]` wartości liczb typu `double` znajdujące się w pliku tekstowym „matrix.txt”. W tym celu skanuje plik ze źródłem i przepisuje jego wartości do `matrix[][]`. Funkcję kończy znak końca pliku bądź przeskanowanie 100 elementów

Funkcja `sortMatrix()` wywołuje `bubbleSort()` na wszystkich wierszach `matrix[][]`.

Funkcja `bubbleSort()` sortuje tablicę metodą bąbelkową, uwzględniając monotoniczność sortowania. Zmienna `swapCounter` informuje o ilości wykonanych zamian wartości dwóch elementów tablicy `arr[]`. Pętla `while()` wykonuje się dopóki `swapCounter` jest różny od zera. Wewnątrz niej `swapCounter` jest resetowany do zera. Wewnątrz wykonuje się pętla `for()` na wszystkich elementach tablicy `arr[]`. Jeśli dwa przyległe elementy spełniają warunek określony przez monotoniczność żadanego sortowania, to dokonuje się na nich funkcja `exch()`, która je zamienia. Każdej zamianie towarzyszy zwiększenie `swapCounter`.

Funkcja `selectF()` wybiera elementy tablicy `arr[][]` i przepisuje je do tablicy `diagonal[][]`. Puste miejsca (ponad oraz na – diagonal) zapełnia przez `NAN`.

Funkcja `selectLowest()` wybiera najniższe wartości kolumn tablicy `arr[][]` i zapisuje je do tablicy `min[]`. Określa tablicę pomocniczą `col[]`. Iteruje po wszystkich kolumnach `arr[][]`, dla każdej iteracji wybiera i-tą kolumnę funkcją `selectCol()`. Sortuje tę kolumnę `bubbleSort()`. Jeśli wszystkie wartości danej kolumny są równe `NAN`, to zapisuje do `min[]` wartość `NAN`, w przeciwnym razie wybiera wartość najniższą.

Funkcja `product()` oblicza iloczyn wartości podanych w `arr[]`. Pomija przy tym wartości nie będące liczbami (`NAN`).

Funkcja `saveResults()` zapisuje w sposób sformatowany tablicę `matrix[][]`, `min[]` i `productVal`. Otwiera plik „results.txt” w trybie zapisu. Jeśli plik otworzy się poprawnie, dokonuje zapisu podanych elementów, w przeciwnym razie funkcję zakończy błąd.

Wnioski

Wizualna ocena w łatwy sposób pozwala na określenie prawidłowości podanych wyników. Proste wymnożenie wyników z `min[]` pozwala natomiast na weryfikację `productVal`.