Wykonał: Radosław Smoter

Grupa: 14

Nr: 27

Numer zadania: 6

Przykład: 62

Prowadzący: Prof. dr hab. inż. Volodymyr Samotyy

Politechnika Krakowska

Wydział Inżynierii Elektrycznej i Komputerowej

Sprawozdanie: Wstęp do Programowania

Kraków 2021

Spis treści

Polecenie

(1) Treść

Napisać kod programu sortowania macierzy $A=\{a_{i,j}\}$ metodą Shella. 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 macierzy od maksymalnego do minimalnego.

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

$f_i(a_{i,j})$ - suma elementów każdego wiersza pod diagonalą główną

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

Kod programu

```c
/**
 * @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 in descending order, denote as M_sort.
  * Sorting algorithm: shell sort.
  * Calculate sums of each row from under the main diagonal, denote as f_j(m_ij).
  * Calculate product F of f_j(m_ij) as F(f_j(m_ij)).
  * Save the results of M_sort, f_j(m_ij), F(f_j(m_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 sumRows(double [][10], double []);
double product(double []);
void saveResults(double [][10], double [], double);


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

  importMatrix(matrix);
  sortMatrix(matrix);


  double diagonal[10][10];
  selectF(matrix, diagonal);

  double rowsSum[10];
  sumRows(diagonal, rowsSum);

  double prod = product(rowsSum);

  saveResults(matrix, rowsSum, prod);

  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);

  if (file != NULL) {
    /* 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);
  }
  else {
    fprintf(stderr, "Error: File did not open successfully.\n");
    return;
  }
}


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

/* Sort an array according to mode, ascending order  - 1, descending - 0 */
void shellSort(double v[], int n, int mode) {
  int gap, i, j;

  for (gap = n/2; gap > 0; gap /= 2)
    for (i = gap; i < n; i++)
    {
      if(mode)
        /* Compare elements that are gap apart from each other */
        for (j = i-gap; j >= 0 && v[j] > v[j+gap]; j -= gap)
```

```
          exch(&v[j], &v[j+gap]);
      else
        for (j = i-gap; j >= 0 && v[j] < v[j+gap]; j -= gap)
          exch(&v[j], &v[j+gap]);
    }
}


/* Convert array 10x10 to array 100 */
void straightenMatrix(double array[], double matrix[][10]) {
  /* Index of array */
  int k = 0;
  for (int i = 0; i < 10; i++)
    for (int j = 0; j < 10; j++)
      array[k++] = matrix[i][j];
}


/* Convert array 100 to matrix 10x10 */
void makeMatrix(double matrix[][10], double array[]) {
  int k = 0;
  for (int i = 0; i < 10; i++)
    for (int j = 0; j < 10; j++)
      matrix[i][j] = array[k++];
}


/* Sort elements of the matrix in descending order */
void sortMatrix(double matrix[][10]) {
  double array[100];
  straightenMatrix(array, matrix);
  shellSort(array, 100, 0);
  makeMatrix(matrix, array);
}


/* Select entries from under the main diagonal of arr */
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;
  }
}


/* Sum elements of rows from under the main diagonal */
void sumRows(double arr[][10], double sums[]) {
```

```
  for (int i = 0; i < 10; i++) {
    double sum = 0;
    int j;
    for (j = 0; !isnan(arr[i][j]); j++) sum += arr[i][j];
    /* Don't include NANs */
    if (j > 0) sums[i] = sum;
    /* Row contains only NANs */
    else sums[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 array[], double product) {
  char *filename = "results.txt";
  char *filemode = "w";
  FILE *file = fopen(filename, filemode);

  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, rows sums:\n");
    for (int i = 0; i < 10; i++)
      fprintf(file, "%8.1f", array[i]);

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

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

}
```

Wyniki

```
Sorted matrix:
   178.0   162.0   162.0   158.0   157.0   151.0   150.0   146.0   145.0   141.0
   137.0   132.0   126.0   122.0   119.0   119.0   118.0   113.0   113.0   109.0
   106.0   105.0   100.0   100.0    99.0    97.0    96.0    95.0    91.0    88.0
    84.0    82.0    65.0    61.0    48.0    41.0    40.0    40.0    37.0    36.0
    35.0    34.0    33.0    32.0    31.0    30.0    28.0    25.0    21.0    19.0
    16.0    14.0    14.0     1.0    -2.0   -13.0   -15.0   -17.0   -23.0   -26.0
   -29.0   -34.0   -41.0   -42.0   -42.0   -44.0   -44.0   -45.0   -49.0   -57.0
   -59.0   -59.0   -64.0   -66.0   -66.0   -66.0   -75.0   -79.0   -80.0   -91.0
   -96.0   -96.0   -98.0  -103.0  -116.0  -117.0  -118.0  -129.0  -129.0  -130.0
  -141.0  -143.0  -145.0  -152.0  -153.0  -156.0  -157.0  -165.0  -176.0  -178.0
Under main diagonal, rows sums:
     nan   137.0   211.0   231.0   134.0    43.0  -232.0  -455.0  -873.0 -1388.0
Product value:
    4.921430e+21
```

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 sortuję ją według wartości elementów w manierze malejącej. Tablica diagonal[][] przechowuje wartości zwracane przez parametr przez selectF() do tej tablicy, które są wartościami spod diagonali głównej matrix[][]. Do tablicy rowsSum[], funkcja sumRows() zwraca przez parametr wartości sum elementów wierszy tablicy diagonal[][]. Następnie do prod, funkcja product() zwraca jako wynik iloczyn wartości rowsSum[]. Wyniki, tj. matrix[][] (po posortowaniu), rowsSum[] oraz prod 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() do array[] „rozprostowuje” tablicę matrix[][] funkcją straightenMatrix(), następnie sortuje array[] za pomocą shellSort(), a ostatecznie przywraca ją do postaci dwuwymiarowej przez makeMatrix().

Funkcja shellSort() sortuje tablicę metodą Shella, uwzględniając monotoniczność sortowania (mode). W funkcji działają trzy pętle. Pierwsza określa odległość elementów, które będą porównywane. Druga – iteruje po wszystkich elementach. Trzecia – określa pozycje porównywanych elementów (ich indeksy). Jeżeli zostanie spełniony warunek określony w mode, następuje zamiana elementów o pozycjach określanych przez te trzy pętle.

Działanie funkcji straightenMatrix() i makeMatrix() jest komplementarne. Pierwsza rozkłada tablicę matrix[][] do tablicy array[], druga – na odwrót. Obie działają poprzez prostą iterację po wszystkich elementach określonych tablic.

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

Funkcja sumRows() iteruje po elementach tablicy arr[][] i przypisuje sumę każdego wiersza do zmiennej automatycznej sum. Jeżeli sum nie zostanie zmieniony, to do odpowiedniego indeksu sums[] przypisywana jest wartość NAN, w innym wypadku jest to wartość określona w sum.

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

Funkcja saveResults() zapisuje w sposób sformatowany tablicę matrix[][], max[] 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 rowsSum[] pozwala natomiast na weryfikację prod.