Міністерство освіти і науки України

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи №7 з дисципліни
«Алгоритми та структури даних-1.
Основи алгоритмізації»

«Алгоритми сортування послідовностей»

Варіант 13

Виконав студент ІП-43 Дутов Іван Андрійович

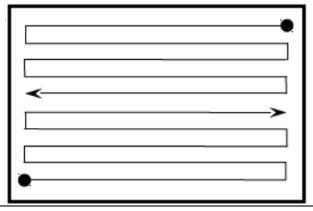
Перевірила Вітковська Ірина Іванівна

Київ 2024

Лабораторна робота №7
АЛГОРИТМИ СОРТУВАННЯ ПОСЛІДОВНОСТЕЙ

Мета – дослідити алгоритми сортування, набути практичних навичок використання цих алгоритмів під час складання програмних специфікацій.

Задача.

№	Розмірність	Тип даних	Спосіб заповнення двовимірного масиву	Обчислення елементів двовимірного масиву
13	6×6	Цілий		Із додатних значень елементів головної діагоналі двовимірного масиву. Відсортувати методом бульбашки за зростанням.

Постановка задачі

Для заповнення двовимірного масиву будемо використовувати одновимірний масив з заготовленими випадковими цілими значеннями від -100 до 100 довжиною 36. З одновимірного масиву ми будемо послідовно заповнювати двовимірний, спочатку верхню половину, згодом нижню, використовуючи арифметичний цикл та оператор альтернативи для визначення необхідного напрямку заповнення (послідовно чи зворотньо). Також створимо підпрограму `fillRow` для заповнення рядка відповідно до його індексу (що працюватиме, адже напрямок заповнення для парних і непарних рядків незмінний як для першої половини, так і для другої).

У отриманому двовимірному масиві визначаємо додатні діагональні елементи, використовуючи інваріант `arr[i][i]` та умовний оператор перевірки додатності елемента. Ці елементи внутрішньо сортуємо із використанням стандартного виду сортування бульбашкою.

Табл. 1: Таблиця імен змінних

Програма			
Змінна	**Тип**	**Ім'я**	**Призначення**
Розмір заповнюваного масива	цілий	`SIZE`	Глобальна константа
Початковий одновимірний масив значень для заповнення	масив цілих	`start`	Константа
Індекс елемента одновимірного масиву	цілий	`next`	Змінна
Індекс рядка у матриці	ціле	`row`	Допоміжна змінна
Матриця для заповнення	двовимірний масив цілих	`filled`	Змінна
Масив діагональних елементів	масив цілих	`arr`	Змінна
Кількість додатніх діагональних елементів	ціле	`diagonalCount`	Змінна
Тимчасова змінна для обміну елементів при сортуванні	ціле	`temp`	Допоміжна змінна
Підпрограма `fillRow`			
Індекс рядка матриці	ціле	`row`	Формальний параметр

Продовження на наступній сторінці...

Підпрограма (продовження)			
Змінна	**Тип**	**Ім'я**	**Призначення**
Одновимірний масив, з якого беруться елементи для заповнення матриці	масив цілих	`from`	Формальний параметр
Матриця, що заповнюється	двовимірний масив цілих	`to`	Формальний параметр
Індекс одновимірного масиву, починаючи з якого заповнювати рядок матриці	ціле	`start`	Формальний параметр
Індекс стовпця матриці	ціле	`col`	Допоміжна змінна
Індекс наступного елемента одновимірного масиву для опрацювання	ціле	`i`	Змінна, що повертається

Крок 1. Визначимо основні дії.

Крок 2. Уточнимо дії ініціалізації констант.

Крок 3. Уточнимо дію заповнення матриці.

Крок 4. Уточнимо дію створення масиву діагональних елементів.

Крок 5. Уточнимо дію сортування масиву діагональних елементів.

Псевдокод програми

Крок 1

1. **Початок**
2. Ініціалізація констант.
3. Заповнення матриці елементами одновимірного масиву.
4. Створення одновимірного масиву діагональних елементів матриці.
5. Сортування масиву діагональних елементів.
6. **Кінець**

Крок 2

1. **Початок**
2. Ініціалізація констант.
   2.1. `SIZE := 6`
   2.2. `start := {98, 67, -60, 64, 90, 1, 4, 18, -72, 17, 86, 29, -72, 10, -67, 11, -2, 77, -55, -43, -48, -81, 30, 18, -10, -28, -11, -89, -67, -50, -47, -10, -52, 71, -75, -56}`
3. Заповнення матриці елементами одновимірного масиву.

4. Створення одновимірного масиву діагональних елементів матриці.
5. Сортування масиву діагональних елементів.
6. **Кінець**

Крок 3

1. **Початок**
2. Ініціалізація констант.
   2.1. `SIZE := 6`
   2.2. `start := {...}`
3. Заповнення матриці елементами одновимірного масиву.
   3.1. Ініціалізація `filled`.
   3.2. `next := 0`
   3.3. **Повторити** для `row` від `0` до `SIZE / 2`
      3.3.1. `next := fillRow(row, start, filled, next)`
      все повторити
   3.4. **Повторити** для `row` від `SIZE - 1` до `SIZE / 2` включно з кроком `-1`
      3.4.1. `next := fillRow(row, start, filled, next)`
      все повторити
4. Створення одновимірного масиву діагональних елементів матриці.
5. Сортування масиву діагональних елементів.
6. **Кінець**

Крок 4

1. **Початок**
2. Ініціалізація констант.
   2.1. `SIZE := 6`
   2.2. `start := {...}`
3. Заповнення матриці елементами одновимірного масиву.
   3.1. Ініціалізація `filled`.
   3.2. `next := 0`
   3.3. **Повторити** для `row` від `0` до `SIZE / 2`
      3.3.1. `next := fillRow(row, start, filled, next)`
      все повторити
   3.4. **Повторити** для `row` від `SIZE - 1` до `SIZE / 2` включно з кроком `-1`
      3.4.1. `next := fillRow(row, start, filled, next)`

все повторити

4. Створення одновимірного масиву діагональних елементів матриці.

   4.1. Ініціалізація `arr`

   4.2. `diagonalCount := 0`

   4.3. **Повторити** для `i` від `0` до `SIZE`

      4.3.1. **Якщо** `filled[i][i] > 0`

      то

      4.3.1.1. `arr[diagonalCount] := filled[i][i]`

      4.3.1.2. `diagonalCount := diagonalCount + 1`

      все якщо

   все повторити

5. Сортування масиву діагональних елементів.

6. **Кінець**

Крок 5

1. **Початок**

2. Ініціалізація констант.

   2.1. `SIZE := 6`

   2.2. `start := {...}`

3. Заповнення матриці елементами одновимірного масиву.

   3.1. Ініціалізація `filled`.

   3.2. `next := 0`

   3.3. **Повторити** для `row` від `0` до `SIZE / 2`

      3.3.1. `next := fillRow(row, start, filled, next)`

   все повторити

   3.4. **Повторити** для `row` від `SIZE - 1` до `SIZE / 2` включно з кроком `-1`

      3.4.1. `next := fillRow(row, start, filled, next)`

   все потворити

4. Створення одновимірного масиву діагональних елементів матриці.

   4.1. Ініціалізація `arr`

   4.2. `diagonalCount := 0`

   4.3. **Повторити** для `i` від `0` до `SIZE`

      4.3.1. **Якщо** `filled[i][i] > 0`

      то

      4.3.1.1. `arr[diagonalCount] := filled[i][i]`

      4.3.1.2. `diagonalCount := diagonalCount + 1`

      все якщо

все повторити

5. Сортування масиву діагональних елементів.

5.1. **Повторити** для `k` від `0` до `diagonalCount`

5.1.1. **Повторити** для `i` від `diagonalCount - 1` до `k` з кроком `-1`

5.1.1.1. **Якщо** `arr[i] < arr[i - 1]`,
то

5.1.1.1.1. `temp := arr[i]`

5.1.1.1.2. `arr[i] := arr[i - 1]`

5.1.1.1.3. `arr[i - 1] := temp`

все якщо

все повторити

все повторити

6. **Кінець**

Псевдокод підпрограми `fillRow`

1. **Початок** `fillRow(row, from, to, start)`

1.1. `i := start`

1.2. **Якщо** `row % 2 == 0`,
то

1.2.1. **Повторити** для `col` від `SIZE - 1` до `0` включно з кроком `-1`

1.2.1.1. `to[row][col] := from[i]`

1.2.1.2. `i := i + 1`

все повторити

інакше

1.2.1. **Повторити** для `col` від `0` до `SIZE`

1.2.1.1. `to[row][col] := from[i]`

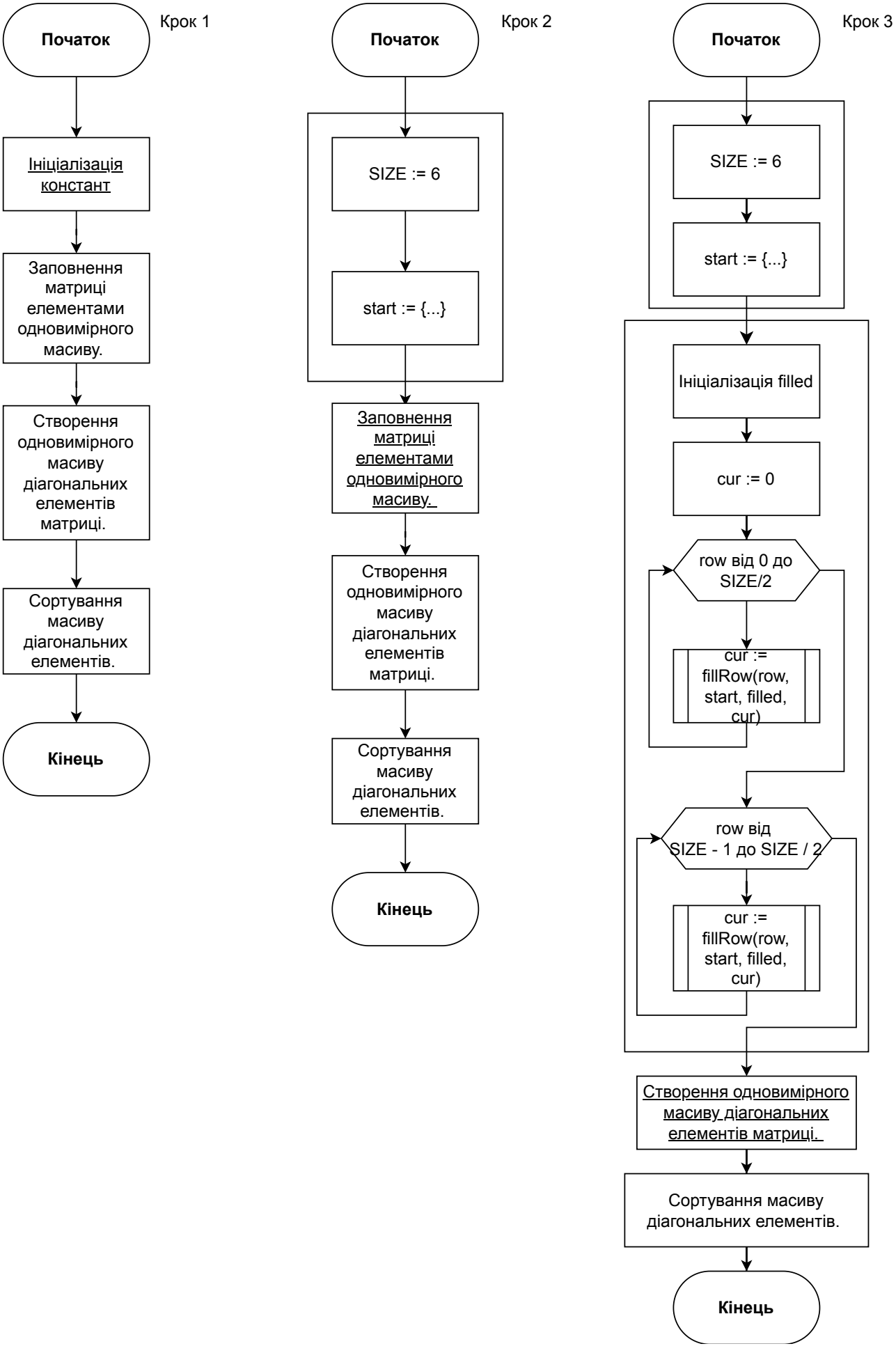
1.2.1.2. `i := i + 1`

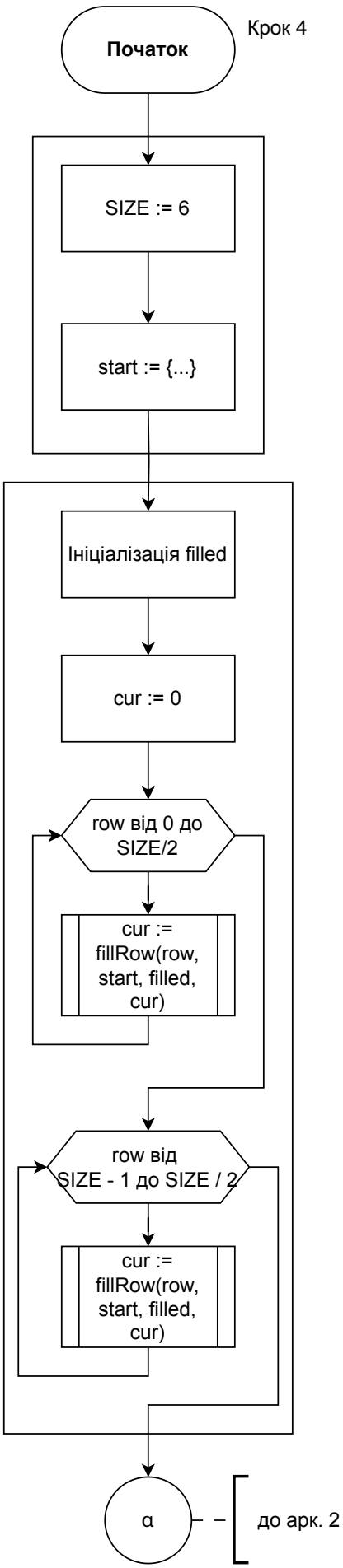
все повторити

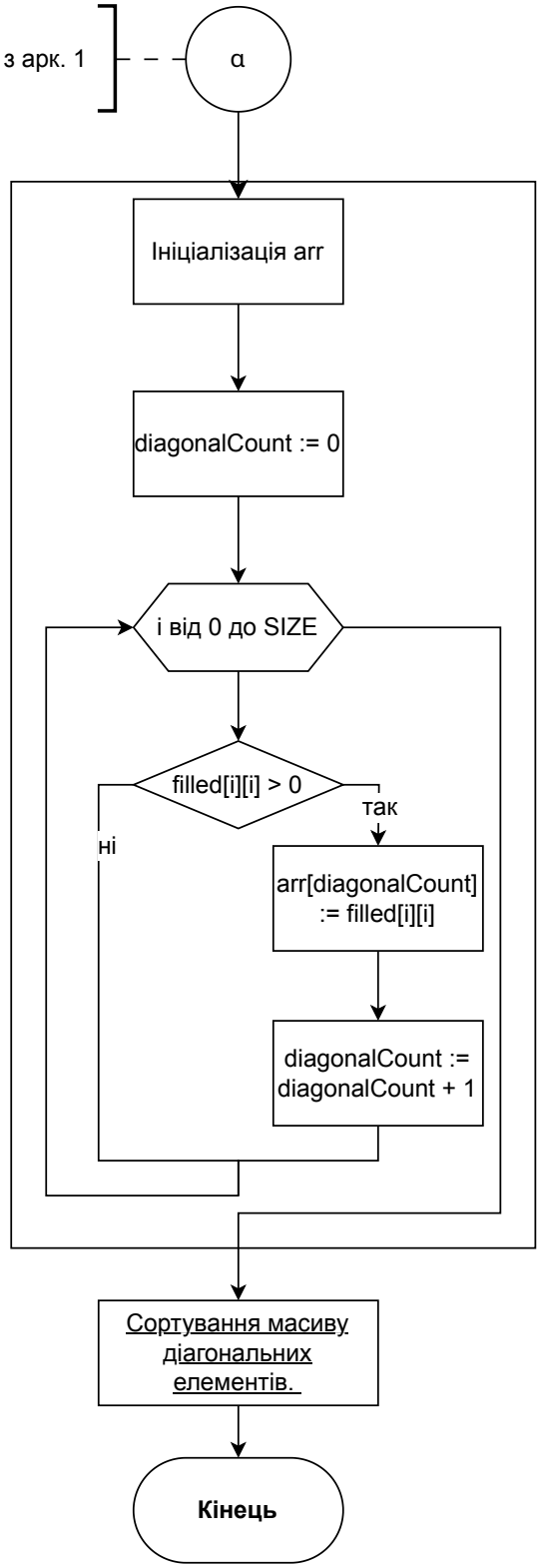
все якщо

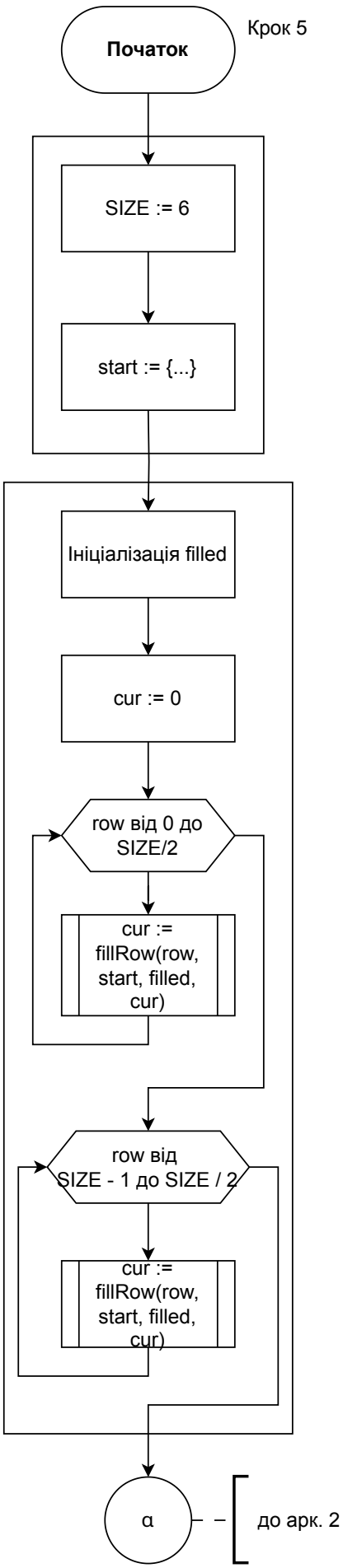
1.3. **Повернути** `i`

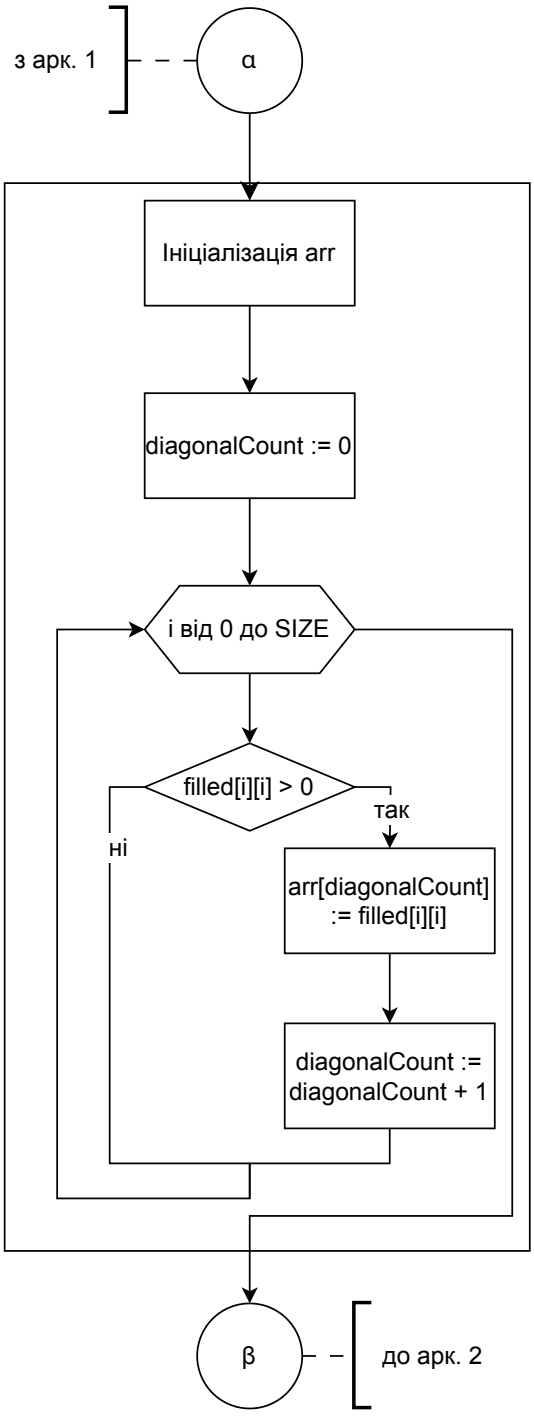
Блок-схема алгоритму

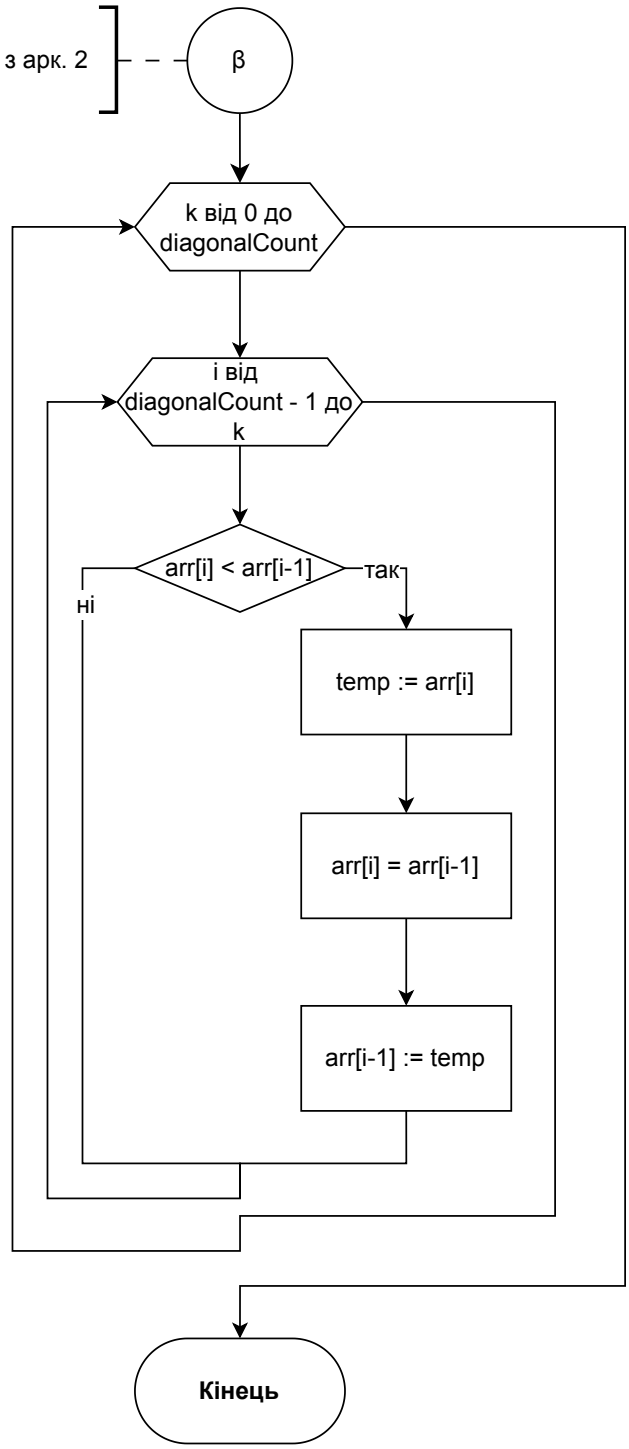

Крок 1
Початок
Ініціалізація констант
Заповнення матриці елементами одновимірного масиву.
Створення одновимірного масиву діагональних елементів матриці.
Сортування масиву діагональних елементів.
Кінець
Крок 2
Початок
SIZE := 6
start := {...}
Заповнення матриці елементами одновимірного масиву.
Створення одновимірного масиву діагональних елементів матриці.
Сортування масиву діагональних елементів.
Кінець
Крок 3
Початок
SIZE := 6
start := {...}
Ініціалізація filled
cur := 0
row від 0 до SIZE/2
cur := fillRow(row, start, filled, cur)
row від SIZE - 1 до SIZE / 2
cur := fillRow(row, start, filled, cur)
Створення одновимірного масиву діагональних елементів матриці.
Сортування масиву діагональних елементів.
Кінець

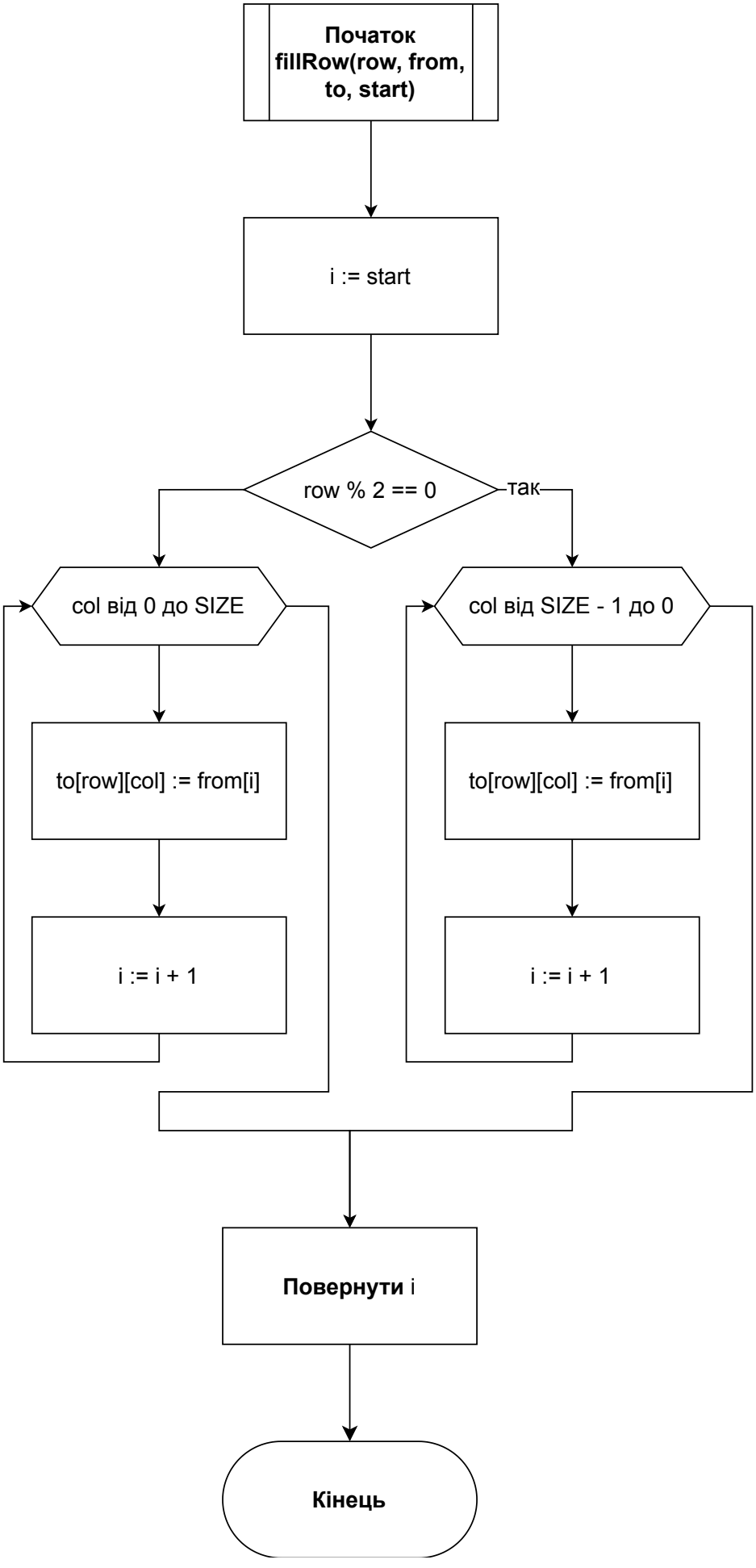

Початок
Крок 4
SIZE := 6
start := {...}
Ініціалізація filled
cur := 0
row від 0 до SIZE/2
cur := fillRow(row, start, filled, cur)
row від SIZE - 1 до SIZE / 2
cur := fillRow(row, start, filled, cur)
α
до арк. 2


з арк. 1
α
Ініціалізація arr
diagonalCount := 0
і від 0 до SIZE
filled[i][i] > 0
так
ні
arr[diagonalCount] := filled[i][i]
diagonalCount := diagonalCount + 1
Сортування масиву діагональних елементів.
Кінець


Початок
Крок 5
SIZE := 6
start := {...}
Ініціалізація filled
cur := 0
row від 0 до SIZE/2
cur := fillRow(row, start, filled, cur)
row від SIZE - 1 до SIZE / 2
cur := fillRow(row, start, filled, cur)
α
до арк. 2


з арк. 1
α
Ініціалізація arr
diagonalCount := 0
і від 0 до SIZE
filled[i][i] > 0
так
ні
arr[diagonalCount] := filled[i][i]
diagonalCount := diagonalCount + 1
β
до арк. 2


з арк. 2
β
k від 0 до diagonalCount
і від diagonalCount - 1 до k
arr[i] < arr[i-1]
так
ні
temp := arr[i]
arr[i] = arr[i-1]
arr[i-1] := temp
Кінець

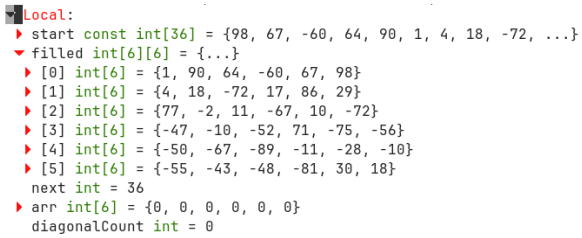

Початок fillRow(row, from, to, start)
i := start
row % 2 == 0
так
col від 0 до SIZE
to[row][col] := from[i]
i := i + 1
col від SIZE - 1 до 0
to[row][col] := from[i]
i := i + 1
Повернути i
Кінець

Код програми мовою C

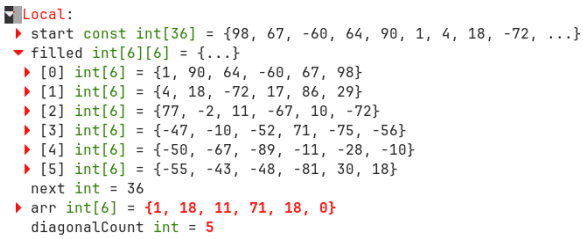
```c
#define SIZE 6

int fillRow(int row, const int *from, int to[SIZE][SIZE], int start);

int main() {
  const int start[SIZE * SIZE] = {98, 67, -60, 64, 90, 1, 4, 18, -72,
                                  17, 86, 29, -72, 10, -67, 11, -2, 77,
                                  -55, -43, -48, -81, 30, 18, -10, -28, -11,
                                  -89, -67, -50, -47, -10, -52, 71, -75, -56};

  int filled[SIZE][SIZE];
  int next = 0;

  // Initializing first half
  for (int row = 0; row < SIZE / 2; row++) {
    next = fillRow(row, start, filled, next);
  }

  // Initializing second half
  for (int row = SIZE - 1; row >= SIZE / 2; row--) {
    next = fillRow(row, start, filled, next);
  }

  int arr[SIZE];
  int diagonalCount = 0;
  for (int i = 0; i < SIZE; i++) {
    if (filled[i][i] > 0) {
      arr[diagonalCount] = filled[i][i];
      diagonalCount++;
    }
  }

  // Bubble sort
  for (int k = 0; k < diagonalCount; k++) {
    for (int i = diagonalCount - 1; i > k; i--) {
      if (arr[i] < arr[i - 1]) {
        int temp = arr[i];
        arr[i] = arr[i - 1];
        arr[i - 1] = temp;
      }
    }
  }

  return 0;
}

int fillRow(int row, const int *from, int to[SIZE][SIZE], int start) {
  int i = start;
  if (row % 2 == 0) { // Odd row
    for (int col = SIZE - 1; col >= 0; col--) {
      to[row][col] = from[i];
      i++;
    }
  } else { // Even row
    for (int col = 0; col < SIZE; col++) {
      to[row][col] = from[i];
      i++;
    }
```

```
59    }
60    return i;
61 }
```

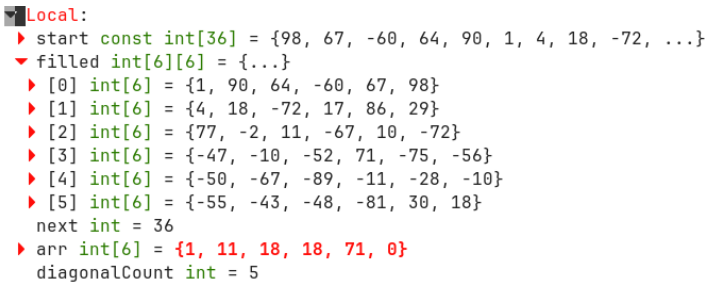
Тестування програми



(a) Матриця після заповнення



(b) Діагональний масив після заповнення



(c) Діагональний масив після сортування

Висновок

Ми дослідили алгоритми сортування, набули практичних навичок використання цих алгоритмів під час складання програмних специфікацій на прикладі програми, що записує елементи до матриці, обирає діагональні для одновимірного масиву та згодом сортує цей одновимірний масив методом бульбашки за зростанням.