МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ
КАФЕДРА ІНФОРМАТИКИ ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

Звіт
до практичної роботи 2
«Метрики розміру. Метрика Lines of Code»
з дисципліни «Економіка ІТ-індустрії та підприємництво»

Викладач:
професор кафедри ІПІ
Сидоров Микола Олександрович

Виконавець:
студент групи ІП-91
Кочев Геннадій Геннадійович
залікова книжка № ІП-9113

Прийняв:
старший викладач кафедри ІПІ
Родіонов Павло Юрійович

Київ - 2022

ЗАВДАННЯ

1. З відкритих джерел вибрати три проекти (відповідно типів наведених в таблиці1). Можна використовувати і власні проекти якщо вони відповідають заданим параметрам.
2. Застосовуючи вимірювачі у відповідних середовищах програмування (наприклад Visual Studio, CodeCounter for Java, CodeCounter, та інші), визначити розмір кожного проекту.
3. Здійснити відповідні економічні розрахунки наведені в роботі.
4. Провести визначення мов програмування C# та Java (або на вибір студента).
5. Підготувати звіт
6. Скласти викладачу

Оберемо три проекти за типами:

- Organic (до 25 тис. рядків коду та невелика команда)

  https://github.com/genndy007/wb_scraper

  Проект для скрапінгу веб-магазину одягу
- Semi-detached (до 75 тис. рядків коду - середній проект)

  https://github.com/unclebob/fitnesse

  Фреймворк для тестування вікі
- Embedded (жорсткі технічні обмеження)

  https://github.com/lvgl/lvgl

  Вбудована графічна бібліотека для маленьких дисплеїв

Виміряємо KLOC для проектів:

```
hennadii@Nitro-AN515-44:~/projects/python/wb_scraper (master)
$ ls
build-n-run.sh       docker-entrypoint.sh  lint-fix.sh  pyproject.toml  src
docker-compose.yaml  Dockerfile            poetry.lock  README.md       test.sh
hennadii@Nitro-AN515-44:~/projects/python/wb_scraper (master)
$ find . -name '*.py' | xargs wc -l
   44 ./src/tests/conftest.py
    0 ./src/tests/__init__.py
    1 ./src/tests/cards/common.py
  141 ./src/tests/cards/test_views.py
   43 ./src/tests/cards/conftest.py
    0 ./src/tests/cards/__init__.py
    1 ./src/tests/users/common.py
   62 ./src/tests/users/test_views.py
    0 ./src/tests/users/__init__.py
   22 ./src/manage.py
   57 ./src/util/stats.py
   54 ./src/util/scrape.py
    0 ./src/util/__init__.py
    6 ./src/cards/apps.py
   14 ./src/cards/serializers.py
   19 ./src/cards/models.py
   27 ./src/cards/tasks.py
    9 ./src/cards/urls.py
    0 ./src/cards/__init__.py
  104 ./src/cards/views.py
   23 ./src/cards/migrations/0002_initial.py
   37 ./src/cards/migrations/0001_initial.py
    0 ./src/cards/migrations/__init__.py
   18 ./src/cards/migrations/0003_alter_card_user_id.py
    6 ./src/users/apps.py
   20 ./src/users/serializers.py
   14 ./src/users/models.py
   44 ./src/users/auth.py
   10 ./src/users/urls.py
    0 ./src/users/__init__.py
  106 ./src/users/views.py
   48 ./src/users/migrations/0001_initial.py
    0 ./src/users/migrations/__init__.py
   23 ./src/main/celery.py
  165 ./src/main/settings.py
   16 ./src/main/asgi.py
   16 ./src/main/wsgi.py
   23 ./src/main/urls.py
    0 ./src/main/__init__.py
 1173 total
hennadii@Nitro-AN515-44:~/projects/python/wb_scraper (master)
$
```

```
hennadii@Nitro-AN515-44:~/projects/misc/fitnesse (master)
$ find ./src -name '*.java' | xargs wc -l
    170 ./src/fitnesse/FitNesseExpediter.java
     49 ./src/fitnesse/logging/LogFormatter.java
     47 ./src/fitnesse/fixtures/SetUp.java
      9 ./src/fitnesse/fixtures/Sleep.java
```

```
    405 ./src/fit/TypeAdapter.java
     10 ./src/fit/FixtureListener.java
     95 ./src/fit/ScientificDouble.java
     83 ./src/fit/FileRunner.java
    242 ./src/fit/Binding.java
     64 ./src/fit/Counts.java
     27 ./src/fit/WikiRunner.java
    229 ./src/fit/FitServer.java
     41 ./src/fit/FixtureClass.java
  55680 total
```

```
hennadii@Nitro-AN515-44:~/projects/misc/lvgl (master)
$ find ./src -name '*.c' | xargs wc -l
    140 ./src/widgets/objx_templ/lv_objx_templ.c
    104 ./src/widgets/spinner/lv_spinner.c
    110 ./src/widgets/win/lv_win.c
    813 ./src/widgets/menu/lv_menu.c
    201 ./src/widgets/line/lv_line.c
```

```
   1181 ./src/themes/default/lv_theme_default.c
    504 ./src/themes/mono/lv_theme_mono.c
    786 ./src/layouts/grid/lv_grid.c
    595 ./src/layouts/flex/lv_flex.c
    104 ./src/hal/lv_hal_tick.c
    195 ./src/hal/lv_hal_indev.c
    589 ./src/hal/lv_hal_disp.c
 244944 total
```

Таким чином, маємо:

- Для першого 1.173 KLOC
- Для другого 55.68 KLOC
- Для третього 244.944 KLOC

Економічні розрахунки

Зусилля (людина/місяць) $Effort = ab * size^{bb}$,
де ab, bb коефіцієнти,
size – розмір продукту в KLOC.

Вартість (грн.) $Cost = Effort * salary$,
де salary – заробітна платня.

Час на розробку $Schedule = cb * Effort^{db}$,
де cb, db –коефіцієнти.

Тип проекту	*ab*	*bb*	*cb*	*db*
Organic	2.4	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

Економічні розрахунки

1) Organic

Size = 1.173 KLOC

Salary = 1300 UAH (стипендія)

Effort = 2.4 * 1.173 ^ 1.05 = 2.838 людина/місяць

Cost = 2.838 * 1300 = 3689,07 UAH

Schedule = 2.5 * 2.838 ^ 0.38 = 3.716 календарних місяців

2) Semi-detached

Size = 55.68 KLOC

Salary = 30000 UAH or around 750 USD

Effort = 3.0 * 55.68 ^ 1.12 = 270.585 людина/місяць

Cost = 270.585 * 30000 = 8 117 550 UAH or around 203 000 USD

Schedule = 2.5 * 270.585 ^ 0.35 = 17.75 календарних місяців

3) Embedded

Size = 244.944 KLOC

Salary = 30000 UAH or around 750 USD

Effort = 3.6 * 244.944 ^ 1.20 = 2649.61 людина/місяць

Cost = 2649.61 * 30000 = 79 488 430 UAH or around 1 987 210 USD

Schedule = 2.5 * 2649.61 ^ 0.32 = 31.14 календарних місяців

Висновок: як можна побачити з результатів обчислень, Organic проекти є найменшими за розміром та відповідно оплачуються менше, зусиль потребують загально менше, та дешевле коштують в цілому. Semi-detached потребують більше робітників для роботи у срок, тобто щоб зробити середній проект за термін 1-2 роки, потрібен доволі великий штат людей - 200-300 чоловік, та по грошах 200 тисяч доларів для того, щоб проект “встав на ноги”. Embedded проекти при тій самій

зарплаті робітникам потребує кардинально більше зусиль для того, щоб вкластися у термін 3-4 роки, і коштує це також значно більше, такі проекти краще починати компаніями з серйозним бюджетом, стартаперам таке менш підходить. Для стартаперів більш реально виглядає проект середній між станами Organic та Semi-detached, коли грошей не так багато, і час відтворення проекту доволі адекватний.

Визначення рівня мови програмування

Для порівняння візьмемо популярні мови програмування - C# та Java

Знайдемо реалізацію одного алгоритму - Дейкстри на графах - на обох мовах, та скомпілюємо їх, порівняємо співвідношення рядків коду до рядків байткоду. Для переводу коду до байткоду скористаємося онлайн-сервісом https://godbolt.org/

Java:

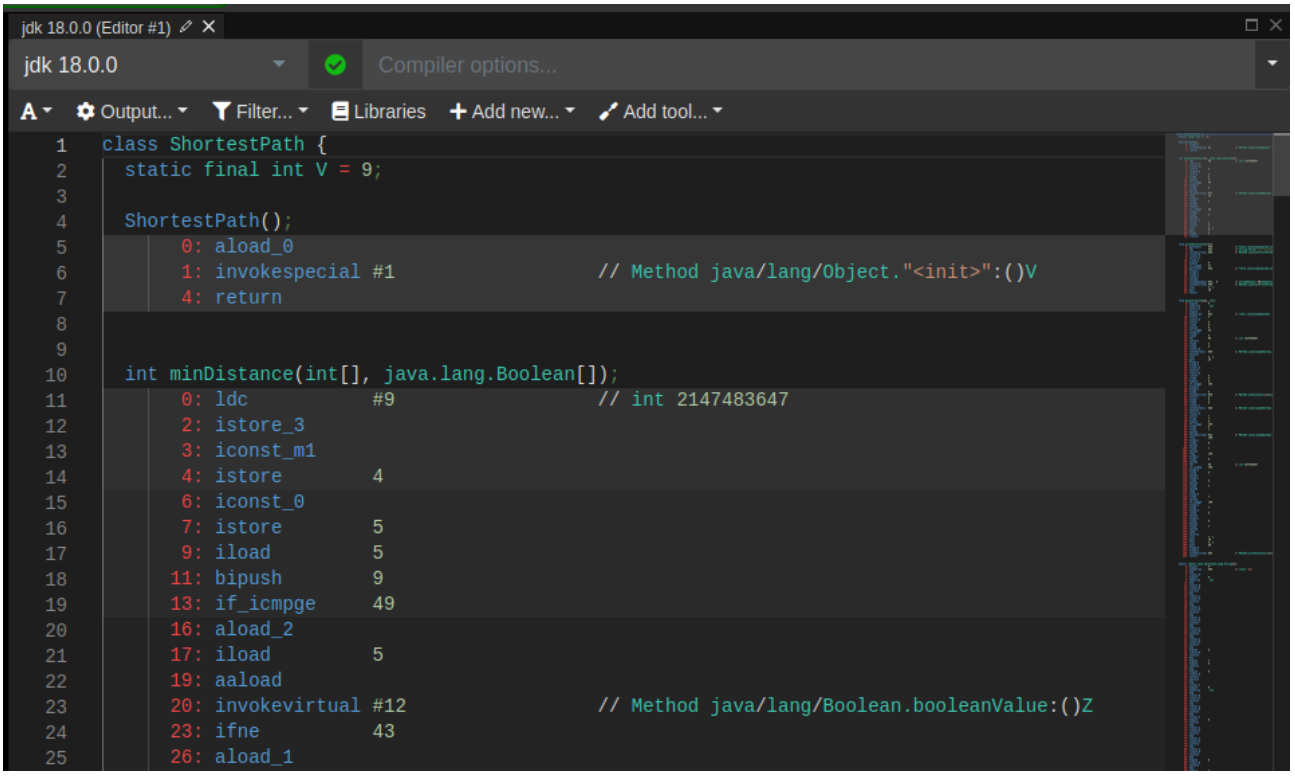
```
import java.io.*;
import java.lang.*;
import java.util.*;

class ShortestPath {
    // A utility function to find the vertex with minimum
    // distance value, from the set of vertices not yet
    // included in shortest path tree
    static final int V = 9;
    int minDistance(int dist[], Boolean sptSet[])
    {
        // Initialize min value
        int min = Integer.MAX_VALUE, min_index = -1;

        for (int v = 0; v < V; v++)
            if (sptSet[v] == false && dist[v] <= min) {
                min = dist[v];
                min_index = v;
            }
```

```
        return min_index;
    }

    // A utility function to print the constructed distance
    // array
    void printSolution(int dist[])
    {
        System.out.println(
            "Vertex \t\t Distance from Source");
        for (int i = 0; i < V; i++)
            System.out.println(i + " \t\t " + dist[i]);
    }

    // Function that implements Dijkstra's single source
    // shortest path algorithm for a graph represented using
    // adjacency matrix representation
    void dijkstra(int graph[][], int src)
    {
        int dist[] = new int[V]; // The output array.
                                 // dist[i] will hold
        // the shortest distance from src to i

        // sptSet[i] will true if vertex i is included in
        // shortest path tree or shortest distance from src
        // to i is finalized
        Boolean sptSet[] = new Boolean[V];

        // Initialize all distances as INFINITE and stpSet[]
        // as false
        for (int i = 0; i < V; i++) {
            dist[i] = Integer.MAX_VALUE;
            sptSet[i] = false;
        }

        // Distance of source vertex from itself is always 0
        dist[src] = 0;

        // Find shortest path for all vertices
```

```
        for (int count = 0; count < V - 1; count++) {
            // Pick the minimum distance vertex from the set
            // of vertices not yet processed. u is always
            // equal to src in first iteration.
            int u = minDistance(dist, sptSet);

            // Mark the picked vertex as processed
            sptSet[u] = true;

            // Update dist value of the adjacent vertices of
            // the picked vertex.
            for (int v = 0; v < V; v++)

                // Update dist[v] only if is not in sptSet,
                // there is an edge from u to v, and total
                // weight of path from src to v through u is
                // smaller than current value of dist[v]
                if (!sptSet[v] && graph[u][v] != 0
                    && dist[u] != Integer.MAX_VALUE
                    && dist[u] + graph[u][v] < dist[v])
                    dist[v] = dist[u] + graph[u][v];
        }

        // print the constructed distance array
        printSolution(dist);
    }

    // Driver's code
    public static void main(String[] args)
    {
        /* Let us create the example graph discussed above
         */
        int graph[][]
            = new int[][] { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },
                            { 4, 0, 8, 0, 0, 0, 0, 11, 0 },
                            { 0, 8, 0, 7, 0, 4, 0, 0, 2 },
                            { 0, 0, 7, 0, 9, 14, 0, 0, 0 },
                            { 0, 0, 0, 9, 0, 10, 0, 0, 0 },
```

```
                                { 0, 0, 4, 14, 10, 0, 2, 0, 0 },
                                { 0, 0, 0, 0, 0, 2, 0, 1, 6 },
                                { 8, 11, 0, 0, 0, 0, 1, 0, 7 },
                                { 0, 0, 2, 0, 0, 0, 6, 7, 0 } };
        ShortestPath t = new ShortestPath();

        // Function call
        t.dijkstra(graph, 0);
    }
}
```

Результуючий байткод має такий вигляд:



.....

```
525      429: bipush        7
526      431: bipush        7
527      433: iastore
528      434: dup
529      435: bipush        8
530      437: iconst_0
531      438: iastore
532      439: aastore
533      440: astore_1
534      441: new           #10                 // class ShortestPath
535      444: dup
536      445: invokespecial #50                 // Method "<init>":()V
537      448: astore_2
538      449: aload_2
539      450: aload_1
540      451: iconst_0
541      452: invokevirtual #51                 // Method dijkstra:([[II)V
542      455: return
543
544  }
```

Таким чином, для Java:

- LOC = 106
- BytecodeLOC = 544

Рівень мови = 544 / 106 = 5.13

C#:

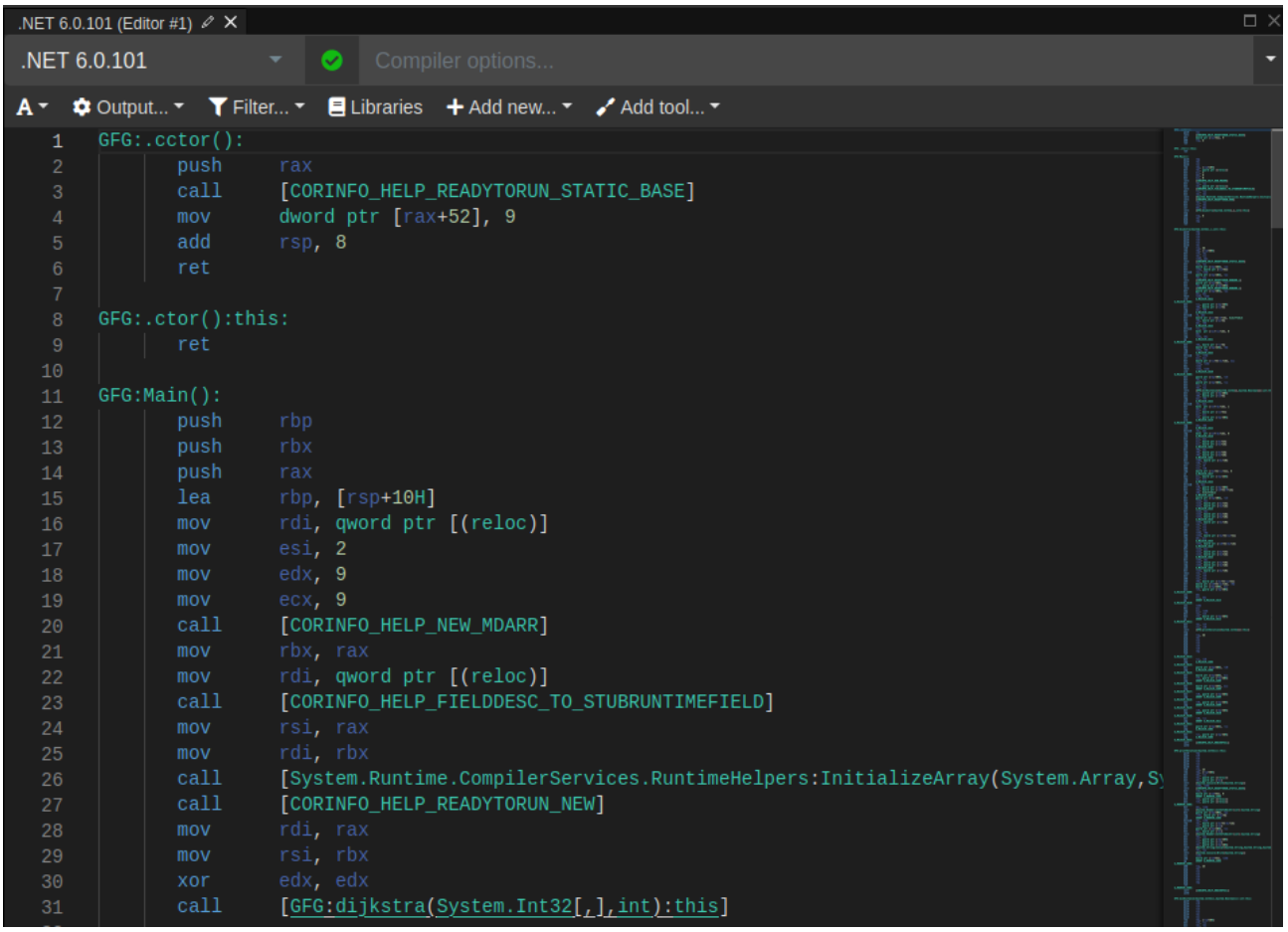
```
using System;

class GFG {
    // A utility function to find the
    // vertex with minimum distance
    // value, from the set of vertices
    // not yet included in shortest
    // path tree
    static int V = 9;
    int minDistance(int[] dist, bool[] sptSet)
    {
        // Initialize min value
        int min = int.MaxValue, min_index = -1;

        for (int v = 0; v < V; v++)
            if (sptSet[v] == false && dist[v] <= min) {
                min = dist[v];
```

```
                min_index = v;
            }

        return min_index;
    }

    // A utility function to print
    // the constructed distance array
    void printSolution(int[] dist)
    {
        Console.Write("Vertex \t\t Distance "
                      + "from Source\n");
        for (int i = 0; i < V; i++)
            Console.Write(i + " \t\t " + dist[i] + "\n");
    }

    // Function that implements Dijkstra's
    // single source shortest path algorithm
    // for a graph represented using adjacency
    // matrix representation
    void dijkstra(int[, ] graph, int src)
    {
        int[] dist
            = new int[V]; // The output array. dist[i]
        // will hold the shortest
        // distance from src to i

        // sptSet[i] will true if vertex
        // i is included in shortest path
        // tree or shortest distance from
        // src to i is finalized
        bool[] sptSet = new bool[V];

        // Initialize all distances as
        // INFINITE and stpSet[] as false
        for (int i = 0; i < V; i++) {
            dist[i] = int.MaxValue;
            sptSet[i] = false;
```

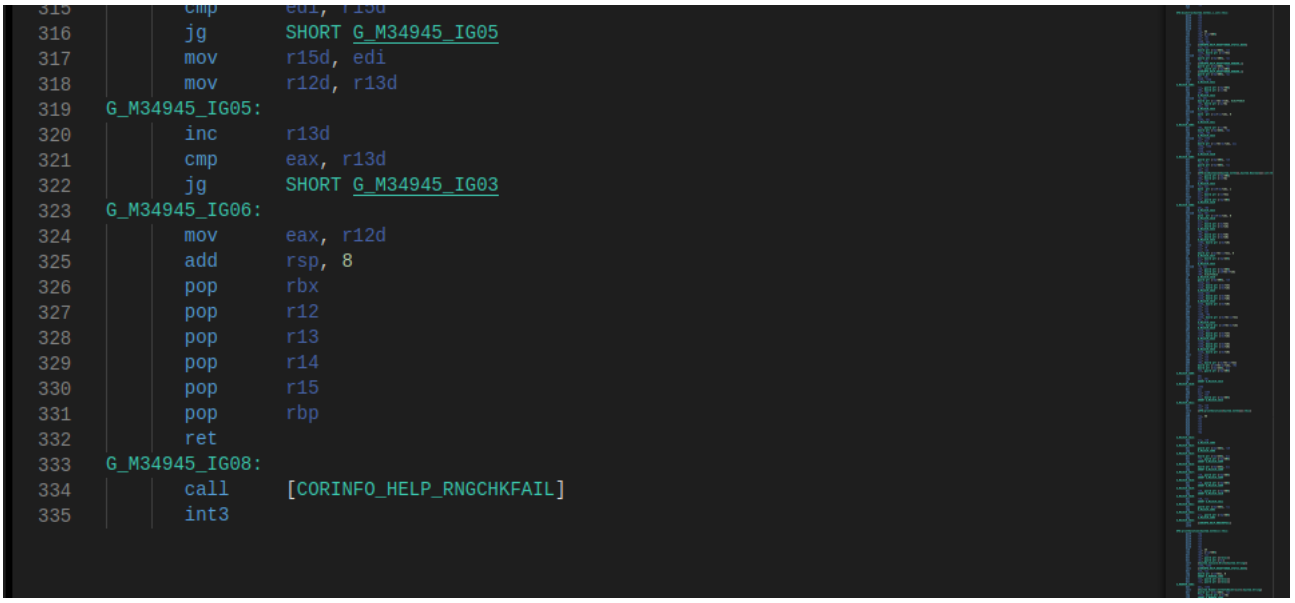
```
        }

        // Distance of source vertex
        // from itself is always 0
        dist[src] = 0;

        // Find shortest path for all vertices
        for (int count = 0; count < V - 1; count++) {
            // Pick the minimum distance vertex
            // from the set of vertices not yet
            // processed. u is always equal to
            // src in first iteration.
            int u = minDistance(dist, sptSet);

            // Mark the picked vertex as processed
            sptSet[u] = true;

            // Update dist value of the adjacent
            // vertices of the picked vertex.
            for (int v = 0; v < V; v++)

                // Update dist[v] only if is not in
                // sptSet, there is an edge from u
                // to v, and total weight of path
                // from src to v through u is smaller
                // than current value of dist[v]
                if (!sptSet[v] && graph[u, v] != 0
                    && dist[u] != int.MaxValue
                    && dist[u] + graph[u, v] < dist[v])
                    dist[v] = dist[u] + graph[u, v];
        }

        // print the constructed distance array
        printSolution(dist);
    }

    // Driver's Code
    public static void Main()
```

```
    {
        /* Let us create the example
graph discussed above */
        int[, ] graph
            = new int[, ] { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },
                            { 4, 0, 8, 0, 0, 0, 0, 11, 0 },
                            { 0, 8, 0, 7, 0, 4, 0, 0, 2 },
                            { 0, 0, 7, 0, 9, 14, 0, 0, 0 },
                            { 0, 0, 0, 9, 0, 10, 0, 0, 0 },
                            { 0, 0, 4, 14, 10, 0, 2, 0, 0 },
                            { 0, 0, 0, 0, 0, 2, 0, 1, 6 },
                            { 8, 11, 0, 0, 0, 0, 1, 0, 7 },
                            { 0, 0, 2, 0, 0, 0, 6, 7, 0 } };
        GFG t = new GFG();

        // Function call
        t.dijkstra(graph, 0);
    }
}
```

Результуючий байткод має такий вигляд:



.....



Таким чином для C#:

- LOC = 112
- BytecodeLOC = 335

Рівень мови = 335 / 112 = 2.99

Висновок: результати порівняння рівнів мов Java та C# такі:

- рівень мови Java 1:5.13, тобто в середньому 1 строка дає 5 строк байткоду
- рівень мови C# 1:2.99, тобто в середньому 1 строка дає 3 строки байткоду

За результатами Java має вищий рівень мови, але обидві мови успішно вирішують задачу алгоритму Дейкстри.