פרויקט – תרגיל 1

פרטי המגיש:

שם: איתמר איילון

ת.ז: 206024796

מייל: [itamaraa@mta.ac.il](mailto:itamaraa@mta.ac.il)

**בונוסים שמומשו:**

**בונוס 1, 2, 3, 4, 5 .**

הסבר (בסיסי) על המערכת:

כשהתוכנית עולה מופיע תפריט אותו ניתן לתפעל רק לאחר טעינת קובץ תקין למערכת.

באפשרות 1 ניתן לטעון קובץ xml עם פורמט תקין (במידה ולא נקבל שגיאה שתכוון אותנו היכן הבעיה).

לאחר טעינת הקובץ למערכת, באופציה 2 בתפריט יופיעו כל הפרמטרים השונים שנטענו למערכת ואיתם האלגוריתם האבולוציוני יעבוד וינסה למצוא פתרון אופטימלי.

באופציה 3 מפעילים את האלגוריתם כאשר מזינים את תנאי הסיום הרצוי, ואת המרווח של הדורות בהם אנו מעוניינים להתעדכן. במידה וכבר קיימים תוצאות במערכת, היא תודיע על כך למשתמש.

באופציה 4 ניתן לראות את הפתרון הטוב ביותר, המערכת תבקש כיצד להציג את הפלט ובהתאם תדפיס למשתמש איך המערכת נראת עם הסבר על מה שמוצג.

באופציה 5 ניתן לראות את התקדמות האלגוריתם למציאת הפתרון.

באופציה 6 ניתן לשמור את המצב הנתון של המערכת.

באפשרות 7 ניתן לטעון קובץ שנשמר ע"י המערכת בשלב כלשהו.

\*\* אפשרות 8 מקנה למשתמש את האופציה לבחור האם המערכת תפעל כ multi-threaded או לא.

(המצב התחלתי של האפליקציה הוא ב multi-threaded).

ו-9 יציאה מהמערכת.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | כמות דורות | | |
| **1,000** | **10,000** | **100,000** |
| גודל אוכלוסיה | **200** | 10 seconds | 2:10 minutes | 23 minutes |
| **500** | 35 seconds | 6:40 minutes | 1:07 hours |
| **1,000** | 2:05 minutes | 21:20 minutes | 3:35 hours |

\* באוכלוסיה של 1,000 זה תמיד מוצא לי פתרון בsmall.xml. אז הכפלתי כמעט לכל כיתה את כמות השעות (ולכן מן הסתם מחזיק יותר "חמישיות" שפוגעות ביעילות).

\* עבור יותר מ-3 דקות, מדדתי כמה זמן לוקח לבצע 500 דורות וחישבתי את הזמן הכולל עבור 10k\100k.

\* מבחינת זיכרון לא הייתה שום בעיה (רץ בערך מיליון דורות בלילה עם אוכלוסיה בגודל 500)

הערות:

\* לאחר כל סיום פעולה בתפריט, וסיום מתן הפלט, המשתמש מתבקש להקיש אנטר לחזור לתפריט הראשי (זה נועד לשם נוחות לראות ישירות את הפלט (במידה ויש) לאחר בחירת האופציה בתפריט).

\* הגבלות שנראו לי הגיוניות למרות שלא נאמר כלום:

גודל האוכלוסיה, מספר הימים וכן השעות חייבים להיות אי-שליליים.

\* בחישוב החוקים של הפתרון הטוב ביותר הוספתי העלאה בחזקה מסויימת (2 או 3) על מנת לתת לשינויים טובים שהערך השינוי שלהם קטן - יותר משמעות.

\* החוקים מחושבים בסקלה של 0 עד 100. כנ"ל לגבי ה fitness של הפתרון.

\* כאשר האלגוריתם עובד במצב multi-threaded, באופציה 3 ניתן להפסיק את האלגוריתם באמצע ולקבל תוצאות עד לעצירה. וכן אופציות 2,4,5 עובדות כמבוקש.

\* במצב multi-threaded, אם האלגוריתם רץ ויוצאים מהמערכת, מפסיקים אותו ומקבלים את תוצאת הסיום רגע לפני היציאה.

\* כל עוד האלגוריתם אינו עובד, ניתן לדרוס את הנתונים שנטענו למערכת (כנדרש) ע"י טעינת קובץ נוסף באפשרויות 1 או 7. (המערכת תתריע על עך לפני כן אם קיים מידע חיוני)

\* האלגוריתם מודיע כאשר הוא מגיע לתנאי העצירה.

\*ההיסטוריה (שניתן לראות באופציה 5) נשמרת ע"י זוגות המכילים דור, ואת ה fitness הגבוהה באותו הדור.

\* בהצגת ההיסטוריה, דור 0 מתייחס לדור ההתחלתי שנוצר עוד לפני הפעלת האלגוריתם.

- בעזרת כמה קונפיגורציות שונות ומגוונות, כן הגעתי בכל הפעמים (האפשריות) לפתרון אופטימלי בקובץ הגדול או הקטן.

הסבר על המחלקות העיקריות:

**במודול ui** קיימת כל התקשורת של המשתמש אל מול האפליקציה.

בו קיימת חבילה menu שדרכה ניתן לייצר תפריט קונסול בצורה נוחה אותו ניתן להציג למשתמש.

ב ApplicationMenus נתונות האפשרויות של התפריט הראשי בתוכנית.

ישנה מחלקה Application, אשר היא האפליקציה הראשית שמופעלת ע"י run. היא זאת שמציגה את התפריטים השונים במערכת ומקבלת את כל הקלט מהמשתמש ואחראית לפלט חזרה. בנוסף היא זאת שמתפעלת את כל המנוע והאלגוריתם האבולוציוני כשהמשתמש מבקש זאת, וגם האחראי לכל טיפול השגיאות במערכת והצגת הפלט בהתאם לשגיאה.

מחלקות EvoInfoOutput, TimeTableInfoOutput נועדו לקריאות ולסייע לApplication להוציא פלט מתאים עבור אפשרויות 4, 2 במערכת.

**במודול evolutionAlgorithm** קיים כל מנוע האלגוריתם הכללי-גנרי בעזרתו ניתן לממש פתרון לבעיה כלשהית בעזרת האלגוריתם אבולוציוני. בו יש אינטרפייסים Crossover, Selection, Solution, Problem ומחלקות אבסטרקטיות Mutation, Population ו-EvolutionEngine.

ב EvolutionEngine מתבצע האלגוריתם עצמו. הוא מכיל שדה population כדי לתאר את האוכלוסיה. שדה selection, crossover ורשימה של mutation כדי להפעיל על האוכלוסיה ולמצוא את הדור הבא.

כמו כן הוא שומר את ההיסטוריה של הפתרונות הטובים ביותר לפי המרווח שניתן לו בהפעלת האלגוריתם. בנוסף מכיל (Supplier<Boolean>) תנאי עצירה, ורשימה של מאזינים למי שמעוניין לדעת מתי האלגוריתם מגיע לדורות לפי המרווח שביקשנו ממנו, וכן מתי האלגוריתם מסיים.

**במודול logic** מממשים את כל הטיפוסים הנדרשים לאלגוריתם האבולוציוני שפותר את בעיית מערכת השעות. ולכן, מכיל את המחלקות השונות ליצירת מערכת שעות. בו לא מתבצע שום פלט, ואינו עובד לבד ללא גורם נוסף שיתפעל אותו.

תחת החבילה timetable ניתן למצוא את כל הדברים שמתקשרים לבעיה אותה אנו מנסים לפתור – בעיית מערכת השעות. שם קיימת המחלקה TimeTable - הפתרון עצמו. המכיל רשימה של שיעורים(מחלקת שיעור מכיל יום, שעה, מורה, כיתה, קורס). בנוסף שם קיימים כל החוקים האפשריים שחלים על מערכת השעות.

תחת החבילה schema נמצא הקוד המג'ונרט מקובץ xsd (הנמצא התיקייה resources בלוגיקה). בנוסף אליו קיימת מחלקה XMLExtractor שבה מטודות בכדי לחלץ חלקים שונים מתוך קובץ הxml ולייצר עבורם את האובייקטים עם הפרמטרים הדרושים בקובץ. במידה ומשהו נכשל, נזרקת שגיאה XMLExtractException עם פירוט היכן הייתה הבעיה. המחלקה TTEvoEngineCreator יוצר אובייקט XMLExtractor וקורת למטודות השונות בכדי ליצור את מנוע האלגוריתם האבולוציוני דרך קובץ הxml.

חבילה timeTableEvolution מספקת את המחלקות הקונקרטיות עבור הבעיה של מערכת השעות - מחלקות המממשות את crossover, mutation, selection, proble... כמו כן האוכלוסיה הספציפית, וכן הבעיה הספציפית. ואליהם קיימים factories שונים בכדי ליצור את האובייקטים השונים לפי הדרישה.

בלוקיגה גם קיימת מחלקה evoEngineSettingsWrapper שסך הכל מעבירה פרמטרים ממנוע האלגוריתם (ולא נותן לבצע השמות אלא רק קריאות של נתונים)

ומחלקת Engine שמנהלת את כל הלוגיקה. היא מחזיקה את מנוע האלגוריתם, ומציגה דרך evoEngineSettingsWrapper את הנתונים הקיימים במנוע. בנוסף דרכה מתבצע רישום להאזנה, הצבת תנאי העצירה, יצירת האלגוריתם מקובץ, הפעלתו והפסקתו בכל בקשה.