**אלגוריתם לבדיקה האם מילה שייכת לשפה הנוצרת ע"י דקדוק חסר הקשר**

**מגיש: רביב קומם**

**ת"ז: 316217751**

**מחלקה: הנדסת תכנה**

**מסגרת ההגשה: עבודה במסגרת קורס אוטומטים מסלול מצוינות**

**שנה"ל: 2020**

**סמסטר: א'**

**תאריך הגשה: 20.02.2020**

**בהנחיית: ד"ר יואב רודה**

**תוכן עניינים:**

**חלק א'**

[**רקע כללי**](#חלק_א_רקע_אלגוריתם) **.............................. עמ' xx**

[**רקע תאורטי**](#חלק_א_רקע_תאורתי) **.............................. עמ' xx**

[**הכנות**](#חלק_א_מימוש_אלגוריתם) **לניסוי ............................ עמ' 3-4**

[**תיאור**](#חלק_א_אלגוריתם_קוד)**יה של הניסוי ................................. עמ' 5**

[**המחשת האלגוריתם**](#חלק_א_המחשת_אלגוריתם) **.......................... עמ' 6-7**

[**הרצת האלגוריתם**](#חלק_א_הרצת_אלגוריתם) **............................ עמ' 8-15**

[**ניתוח תוצאות ומסקנות**](#חלק_א_ניתוח_תוצאות) **.................. עמ' 16-17**

**נספחים**

[**רשימת מקורות**](#רשימת_מקורות) **................................... עמ' XX-XX**

**חלק א' – בדיקה האם מילה שייכת לשפה הנוצרת ע"י דקדוק חסר הקשר**

**רקע כללי:**

דקדוקים חסרי הקשר היו בשימוש נרחב על מנת לתאר את התחביר של שפות תכנות ושל שפות טבעיות. לעיתים קרובות הם נקראים דקדוקי () כ*אשר משתמשים בהם עבור שפות תכנות. האנליזה של תחביר של תוכנות או משפטים הוא חלק קריטי ונכבד במימוש של מהדרים () ומתורגמנים (interpreters) עבור שפות תכנה וכמובן עבור כל תוכנה באופן כללי אשר בעלת "הבנה" של שפות טבעיות או יכולת תרגום שלהן. לכן לפיתוח של מנתחי תחבירים (parsing) עבור דקדוקים חסרי הקשר יש חשיבות בתחומים הללו בעולם מדעי המחשב.*

האלגוריתם מהווה את הבסיס של אפליקציית הניווט המוכרת Waze אשר מאפשרת למשתמשים למצוא את המסלול הקצר ביותר בין שתי נקודות במפה. כמו כן הוא נמצא בשימוש נרחב בתחום פיתוח המשחקים על מנת לאפשר מימוש של "מציאת מסלול" (pathfinding) עבור השחקן או האויבים.

מציאת מסלול קצר ביותר זוהי בעיה הנדסית הלקוחה מהעולם האמיתי, כיצד למצוא את המרחק הקצר ביותר בין שתי נקודות. אם היינו מדברים על מרחק אווירי אזי הבעיה הייתה פשוטה, פשוט ניתן לטוס בקו ישר בין שתי נקודות ולדאוג שלא נתנגש במטוס אחר. אך במציאת מסלול על הקרקע יש מכשולים רבים בדרך ולכן הבעיה אינה טריוויאלית.

האלגוריתם פותח כחלק מפרויקט לפיתוח רובוט נייד ("Shakey the robot") אשר יוכל לתכנן את פעולותיו.

האלגוריתם פועל בצורה דומה לאלגוריתם דייקסטרה (פרי יצירתו של אסחר דייקסטרה) ואף נחשב להרחבה שלו. אלגוריתם דייקסטרה יכול למצוא את המסלול הקצר ביותר עבור כל הנקודות בעולם (אשר מיוצגות על ידי גרף) מנקודת מוצא אחת, ובפרט את המסלול הקצר ביותר בין שתי נקודות נתונות.

אלגוריתם משפר את הביצועים והיעילות ביחס לאלגוריתם דייקסטרה כיוון שהוא ממוקד מטרה במציאת מסלול קצר ביותר בין הנקודות הנתונות. הוא מבצע זאת על ידי התקדמות בצורה "מושכלת" על ידי פונקציית אומדן הנקראת גם היוריסטיקה (heuristic) אשר מאפשרת לאלגוריתם למצוא את המרחק הקל ביותר בין שתי נקודות ללא צורך לחקור את כל הגרף.

**רקע תאורטי:**

אלגוריתם הינו אלגוריתם מסוג חיפוש לרוחב, כאשר הוא מנוסח במונחים של גרפים ממושקלים. האלגוריתם מתחיל מצומת ספציפית בגרף ומטרתו היא למצוא את הדרך הקלה ביותר לצומת המטרה (דרך קלה ביותר משתנה בהתאם לצרכים לדוגמא המרחק הקצר ביותר, הזמן הקצר ביותר וכו'...). הוא עושה זאת על ידי שמירת עץ כל הדרכים אשר מתחילות בצומת ההתחלתית והארכת הדרכים כל פעם בצומת אחת, עד אשר תנאי הסיום מתרחש.

בכל איטרציה של הלולאה, האלגוריתם נדרש לקבוע איזו דרך מהדרכים הוא ירחיב, הוא מבצע זאת בהסתמכות על המשקל של הדרך והערכה של המשקל שיידרש לדרך כדי להגיע עד צומת היעד.

באופן ספציפי אלגוריתם בוחר בצומת אשר לה יש את התוצאה הנמוכה ביותר של הפונקציה:

כאשר הביטויים במשוואה מייצגים:

זוהי הצומת הבאה בדרך

– המשקל של הדרך מצומת ההתחלה ועד צומת

– פונקציית היוריסטיקה אשר מחשבת אומדן של המשקל הכי קל מצומת שעליה אנו מסתכלים ועד צומת המטרה.

האלגוריתם נעצר כאשר הדרך שהוא בחר להרחיב היא דרך מצומת המוצא עד צומת היעד, או אם אין יותר דרכים שניתן להרחיב.

פונקציית היוריסטיקה היא ספציפית לבעיה, כלומר עבור בעיות שונות ייתכן ונדרש להיוריסטיקות שונות (לדוגמא מסלול קצר ביותר מבחינת מרחק לעומת מסלול בו זמן ההגעה הוא המהיר ביותר).

אם פונקציית ההיוריסטיקה היא קבילה ("admissible"), משמע שהיא לעולם לא מאריכה יתר על המידה את העלות האמיתית להגיע אל המטרה, אזי מובטח שהאלגוריתם יחזיר לנו את הדרך הקלה ביותר מנקודת המוצא אל המטרה.

**הכנות לניסוי:**

1. מימוש האלגוריתם בקוד יבוצע באמצעות:

* סביבת עבודה– Eclipse
* שפת תוכנה– Java
* כאשר אני נשען במימוש שלי על הספרייה org.graphstream שפותחה ומתוחזקת על ידי הצוות: Stefan Balev, Antoine Dutot, Yoann Pigné, Guilhelm Savin.

1. באמצעות הספרייה org.graphstream ניתן לבצע את הפעולות הבאות:

* יצירת אובייקט מסוג גרף.
* יצירת אובייקט מסוג צומת.
* יצירת אובייקט מסוג קשת.
* יצירת אובייקט מסוג sprite.
* הוספת תכונות (attributes) לאובייקטים של הגרף.
* הצגת הגרף בצורה ויזואלית.
* עדכון דינאמי של תצוגת הגרף.
* בניית הגרף על ידי קריאת קובץ dgs (dynamic graph source).
* המרת נתונים גאוגרפים לקובץ dgs.
* סגנון אובייקטים (גודל, צבע וכו'..).

פעולות אלה מאפשרות גמישות רבה במעבר על הגרף ומימוש גרפים גדולים אשר מייצגים מידע אמיתי על העולם. בנוסף לכך ישנה אפשרות של הוספת תכונות רבות לכל אובייקט שמקלה על מימוש האלגוריתם. כמו כן אציין כי ריבוי מידע עבור הגרף מאפשר הרצת אלגוריתמי חיפוש שונים ביחס לכל אלמנט מידע, לדוגמא מרחקים הכי קצרים וגם זמנים הכי מהירים.

1. כתבתי בעצמי את המחלקות הבאות:

* **MyGraphUtil** – זוהי מחלקה אבסטרקטית אשר כל המתודות שלו הן סטטיות ובאות לאפשר לי פעולות בסיסיות על הגרף, כמו משקול קשתות הגרף ומתן תכונות תצוגה וסגנון שיאפשרו הצגה ברורה ואיכותית של המידע.
* **Interface Heuristic** – כל היוריסטיקה שכתבתי מממשת את ממשק זה ובכך אני יכול לפעול באמצעות אותו אלגוריתם חיפוש תוך כדי שימוש בהיוריסטיקות שונות עליהן ארחיב בהמשך.
* **SearchAlgorithm** – זוהי המחלקה המרכזית אשר גם היא אבסטרקטית והיא מכילה בתוכה מתודה סטטית של סריקת הגרף וחיפוש המסלולים הקלים ביותר כאשר היא מסתמכת בכך על תכונות המשקל – "weight" של כל קשת. אציין כי במימוש שלי בחרתי לצבוע כל צומת וקשת בסריקה בכחול בכדי להמחיש את היקף החיפוש ובנוסף לכך אני מתחזק מונה של מס' הצמתים אשר ביקרתי בהן בכדי שיהיה לי מדד להשוואות.
* במחלקה זו ישנה מתודה נוספת אשר משחזרת את המסלול הקצר ביותר וצובעת אותו באדום בכדי להדגיש את המסלול הקל ביותר.

**תאוריה של הניסוי:**

**הסבר מבני הנתונים:**

1. לכל צומת אתחזק שלושה תכונות (attributes) חדשות:

* – זוהי תכונה של המרחק האמיתי הנמדד מצומת המוצא ועד הצומת הנוכחית. חישוב ערך של תכונה זו עבור כל צומת יתבצע על ידי כך שנחבר בין הg\_score של הצומת שממנה יש קשת אל הצומת שאנו בודקים לבין משקל הקשת.
* – תכונה זו היא חיבור של הg\_score לבין התוצאה של פונקציית האומדן (heuristic) ודרך תכונה זו נבצע את ההשוואות של הצמתים בכל שלב בכדי לבחור את הצומת הכי קרובה.
* – תכונה זו בעצם מכילה את צומת ה"אב" של הצומת הנוכחית, תכונה זו חשובה בכדי לשחזר את המסלול.
* תכונות אלו יכולות להתעדכן בזמן ריצה אם נמצאה צומת שדרכה המסלול קל יותר.

1. אתחזק מבנה נתונים של קבוצת הצמתים בהן האלגוריתם ביקר וחקר את כל שכניהן וקבע אומדן למרחקים .

אממש מבנה נתונים זה על ידי אובייקט מסוג HashSet שיאפשר לי חיפוש של האם צומת נמצאת בקבוצה הזאת באופן מיטבי כאשר בממוצע אצפה לסיבוכיות זמן של .

1. אתחזק מבנה נתונים של קבוצת הצמתים אליהן הגעתי במהלך הסיור בגרף וטרם חקרתי בצורה מלאה – .

אממש מבנה זה על ידי אובייקט מסוג תור קדימויות (( כאשר ההשוואה בין הצמתים תעשה על ידי השוואה של גודל שדה הf\_score בין הצמתים.

**תיאור מילולי של האלגוריתם:**

קלט האלגוריתם הינו צומת המוצא - , צומת היעד - , פונקציית ההיוריסטיקה - ואובייקט מסוג sprite (אשר ישמש כמונה צמתים ואינו נחוץ לנכונות האלגוריתם).

תחילה אאתחל את מבני הנתונים ו, ואציב בשדות של נקודת המוצא:

ואכניס את צומת ההתחלה לתור הקדימויות.

כעת כל עוד תור הקדימות מכיל איברים ועוד לא הגעתי לצומת היעד אני אחקור את הגרף באופן הבא:

אוציא את הצומת הנוכחית מתור הקדימויות (כיוון שזהו תור קדימויות לפי ערך ה אז זוהי בהכרח תהיה הצומת עם הסבירות הכי גבוהה לקדם אותנו לעבר נקודת היעד) ואכניס אותה לרשימת הצמתים אותן חקרתי אם צומת זו היא צומת היעד הרי שהאלגוריתם הגיע אל המטרה וניתן לסיים, אחרת עבור כל שכן של צומת זו נחשב את הערכים הבאים:

* משקל הצומת מהצומת הנוכחית אל השכן
* ערך זמני שהוא חיבור אל ערך ה הנוכחי עם משקל הצומת
* ערך פונקציית ההיוריסטיקה בין השכן לבין נקודת היעד
* ערך ה זמני שהוא חיבור של

ויתפרש מצומת המוצא כלפי חוץ כאשר בכל שלב הוא יחפש להוסיף צומת ולגרום לכך שיהיה לו מסלול קל ביותר, כך עד שהוא מגיע לצומת היעד או שלא נותרו לו יותר צמתים לחקור.

**שרטוט הניסוי:**

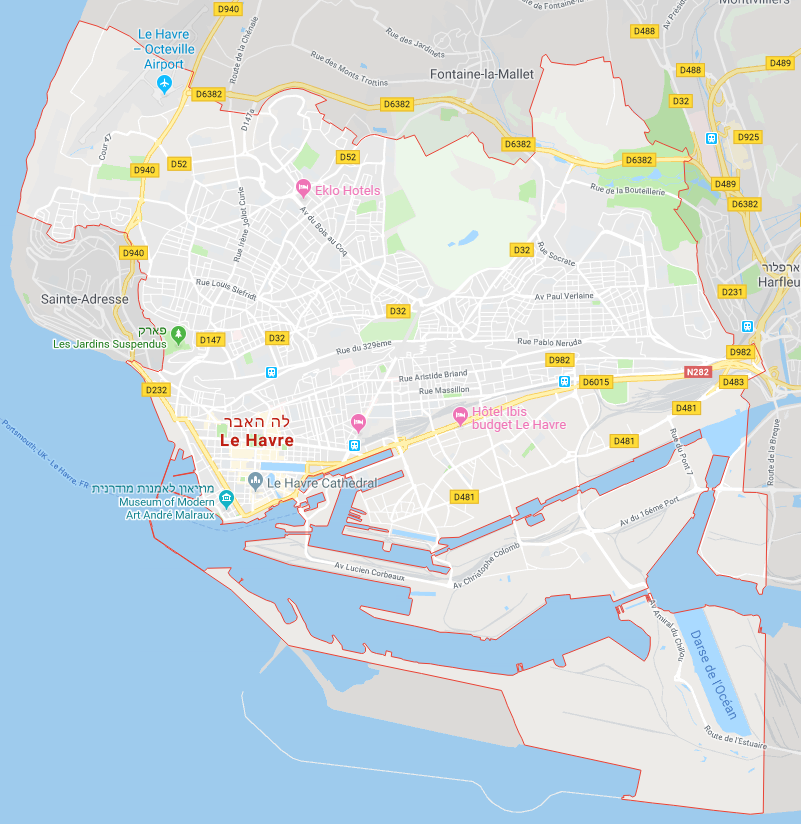
לצורך ההמחשה של האלגוריתם לקחתי מאגר נתונים גאוגרפיים של העיר לה האבר ("Le Havre") בצרפת אשר הומר לקובץ "dgs" שהספרייה יודעת לקרוא ולתרגם לגרף.

הdata base מכיל מידע נחוץ להרצת האלגוריתם על הגרף:

1. עבור כל צומת יש לנו קואורדינטות x,y (כלומר מידע שישמש אותנו בחישוב ההיוריסטיקה)
2. לכל קשת יש מידע של אורך הקשת במטרים, זמן ממוצע שלוקח לעבור את המקטע וכמו כן גם מידע האם הקשת מייצגת כביש אגרה. (מידע שישמש אותנו במשקול קשתות הגרף).

נסתכל על תמונה מGoogle Maps של העיר ונשווה אותו אל הגרף שהתקבל בכדי להראות שאכן מדובר בגרף שמייצג מידע גאוגרפי אמיתי על העולם:

תמונה א' – העיר לה האבר מGoogle Maps



תמונה ב' – העיר לה האבר בייצוג הגרף לאחר קריאת הנתונים



אציין כי הגרף עצמו גדול מאוד, הוא מורכב מ11734 צמתים אשר מחוברות על ידי 15135 קשתות ולכן בחרתי להתמקד באזורה הדרום מערבי של העיר, אשר ניתן לראות שהוא גם יותר צפוף מבחינת הגרף.

תמונה ג' – אזור ההתמקדות של הגרף



**הרצת האלגוריתם**

על אף גודלו של הגרף האלגוריתם מתבצע כמעט באופן מידי ולכן בכדי לאפשר תחושה של התקדמות החיפוש בזמן אמת בחרתי להשתמש בקריאת מערכת ההפעלה Thread.sleep כדי לגרום להשהיה יזומה בין ביקור בכל צומת.

כעת אדגים את ריצת האלגוריתם במספר תנאים שונים, כאשר בחרתי נקודת מוצא ונקודת יעד בעיר באופן שרירותי בקצוות שונים של האזור בו בחרתי להתמקד.

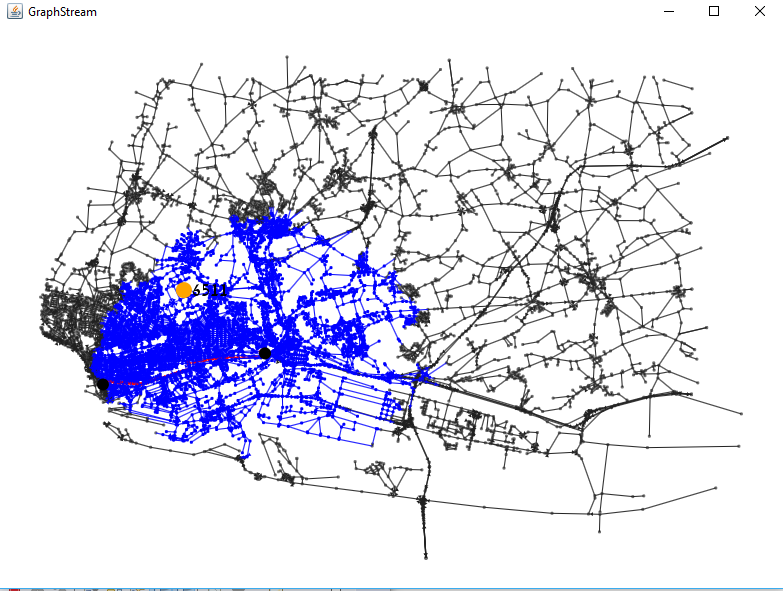
בחלק הבא של הפרויקט אנתח את תוצאות ההרצה ואבצע הסקת מסקנות אשר נובעות מהרצת האלגוריתם.

**הרצה מס' 1**:

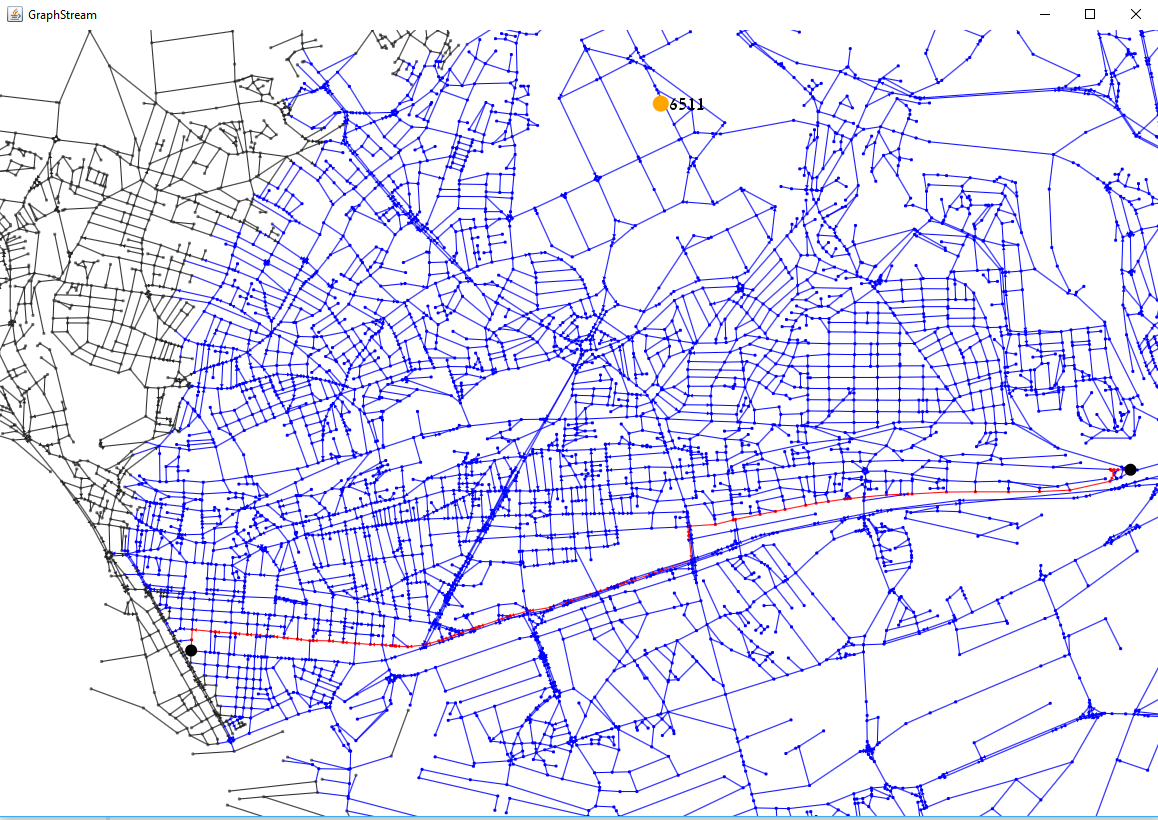
**תנאים מקדימים:**

1. משקל כל קשת נקבע על ידי אורך המסלול (length) במטרים.
2. היוריסטיקת האלגוריתם תהיה " NoDistanceHeuristic" אשר תמיד מחזירה את הערך 0.0, אציין כי בהרצת אלגוריתם עם ההיוריסטיקה הנבחרת בעצם אני מבצע הרצת אלגוריתם חיפוש דייקסטרה.

**תמונת מצב החיפוש:**



ובהתמקדות באזור החיפוש נראה את הגרף באופן הבא:



**תוצאות החיפוש:**

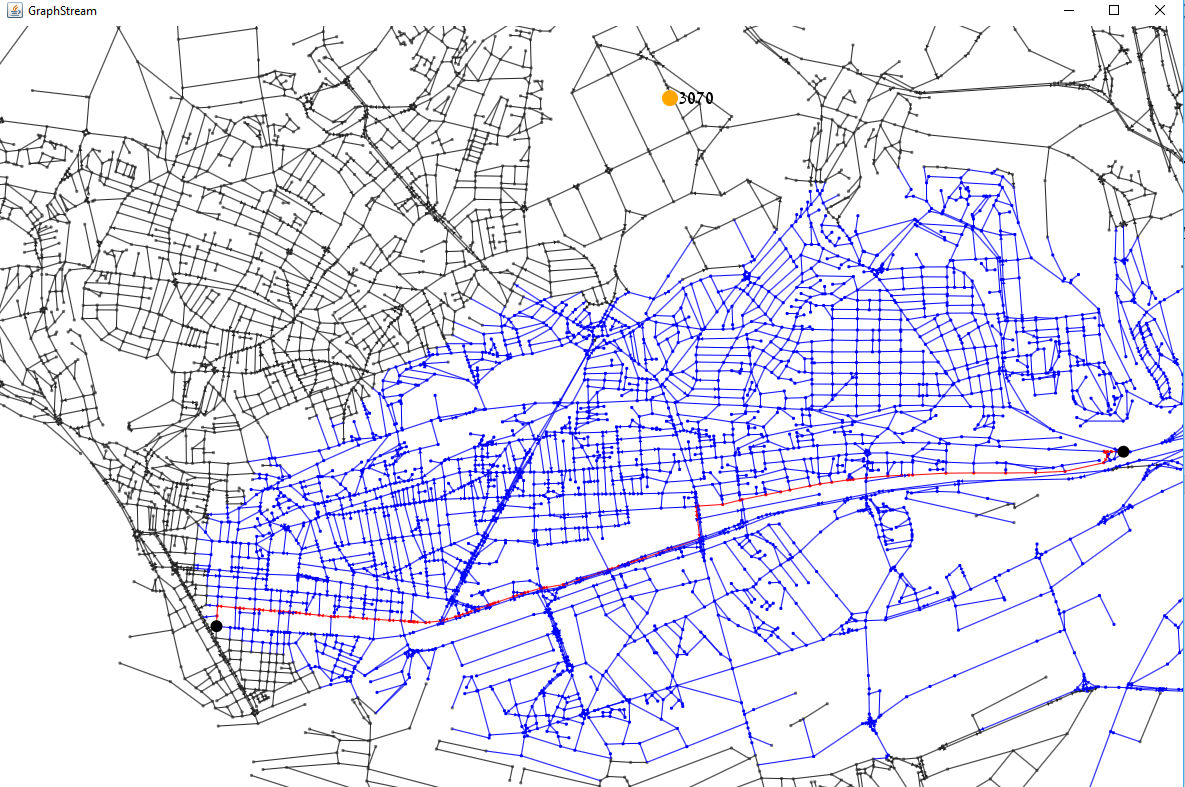
* כמות צמתים שהאלגוריתם חקר והוסיף לדרכים – 6521
* כמות צמתים במסלול הקל ביותר – 85
* מרחק כולל של המסלול – 6.765 קילומטרים
* זמן ממוצע שייקח לעבור את המסלול – 7 דק ו55 שניות

**הרצה מס' 2**:

**תנאים מקדימים:**

1. משקל כל קשת נקבע על ידי אורך המסלול (length) במטרים.
2. היוריסטיקת האלגוריתם תהיה " ManhattanDistanceHeuristic" אשר מחזירה את מרחק מנהטן בין הצומת הנוכחית לבין צומת המטרה, חישוב על ידי הנוסחה

**תמונת מצב החיפוש:**



**תוצאות החיפוש:**

* כמות צמתים שהאלגוריתם חקר והוסיף לדרכים – 3070
* כמות צמתים במסלול הקל ביותר – 85
* מרחק כולל של המסלול – 6.765 קילומטרים
* זמן ממוצע שייקח לעבור את המסלול – 7 דק ו55 שניות

**הרצה מס' 3**:

**תנאים מקדימים:**

1. משקל כל קשת נקבע על ידי אורך המסלול (length) במטרים.
2. היוריסטיקת האלגוריתם תהיה " DiagonalDistanceHeuristic" אשר מחזירה ערך מרחק המחושב על ידי הנוסחה

**תמונת מצב החיפוש:**



**תוצאות החיפוש:**

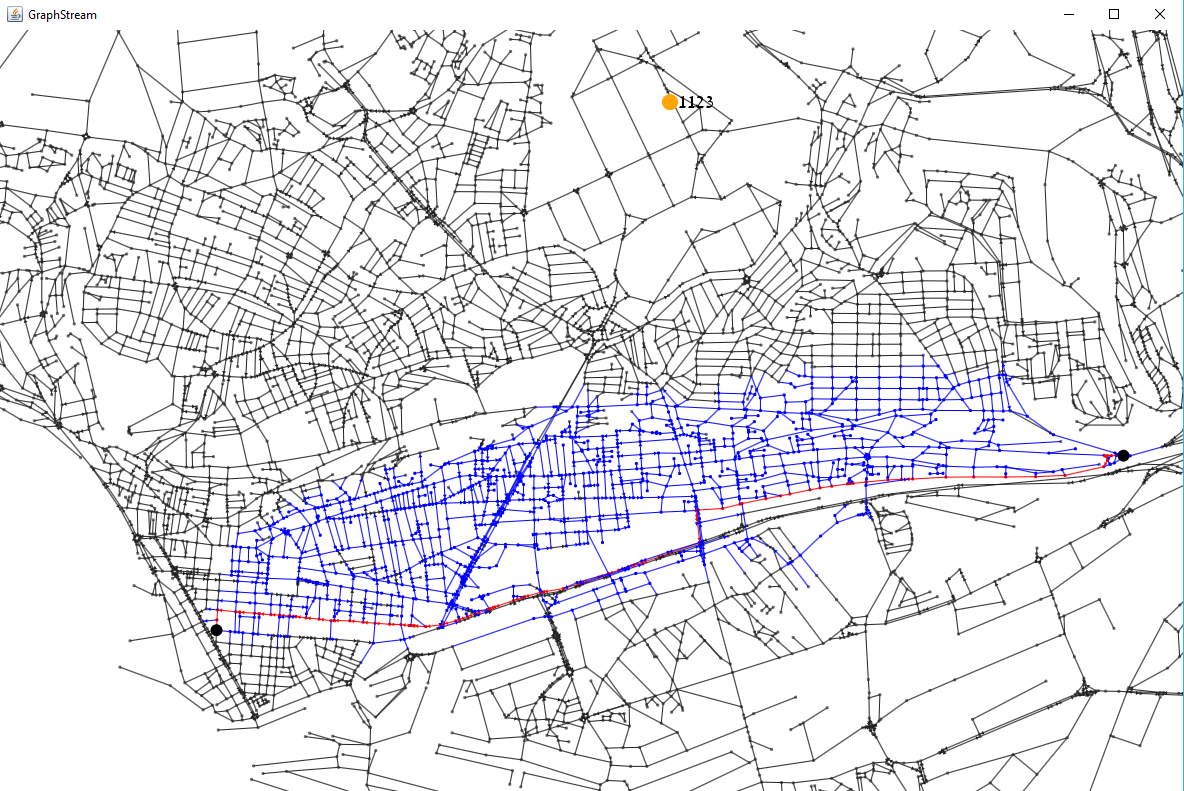
* כמות צמתים שהאלגוריתם חקר והוסיף לדרכים – 1637
* כמות צמתים במסלול הקל ביותר – 85
* מרחק כולל של המסלול – 6.765 קילומטרים
* זמן ממוצע שייקח לעבור את המסלול – 7 דק ו55 שניות

**הרצה מס' 4**:

**תנאים מקדימים:**

1. משקל כל קשת נקבע על ידי אורך המסלול (length) במטרים.
2. היוריסטיקת האלגוריתם תהיה " EuclidianDistanceHeuristic" אשר מחזירה את המרחק האוקלידי בין הצומת הנוכחית לבין צומת המטרה, חישוב על ידי הנוסחה

**תמונת מצב החיפוש:**



**תוצאות החיפוש:**

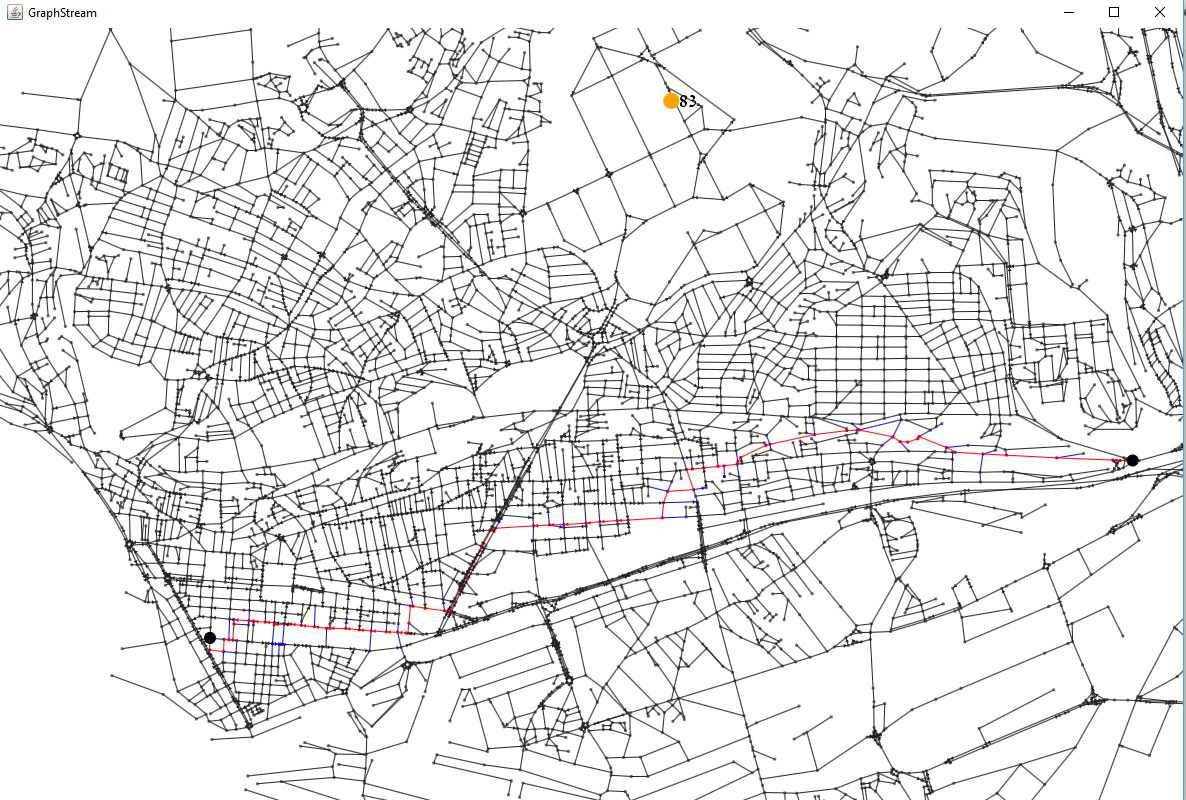
* כמות צמתים שהאלגוריתם חקר והוסיף לדרכים – 1123
* כמות צמתים במסלול הקל ביותר – 85
* מרחק כולל של המסלול – 6.765 קילומטרים
* זמן ממוצע שייקח לעבור את המסלול – 7 דק ו55 שניות

**הרצה מס' 5**:

**תנאים מקדימים:**

1. משקל כל קשת נקבע על ידי אורך המסלול (length) במטרים.
2. היוריסטיקת האלגוריתם תהיה " ManhattanDistanceHeuristic".
3. ביצעתי מניפולציות בקוד הכתוב כך ש- כלומר כעת אלגוריתם זה מסתכל אך ורק על פונקציית האומדן ללא התחשבות במשקל שהצטבר עד כה. בכך אני הופך את אלגוריתם החיפוש לאלגוריתם חמדן אשר בכל שלב מחפש את הצומת לה הוא חושב יהיה את המסלול הכי קל אל עבר היעד.

**תמונת מצב החיפוש:**



**תוצאות החיפוש:**

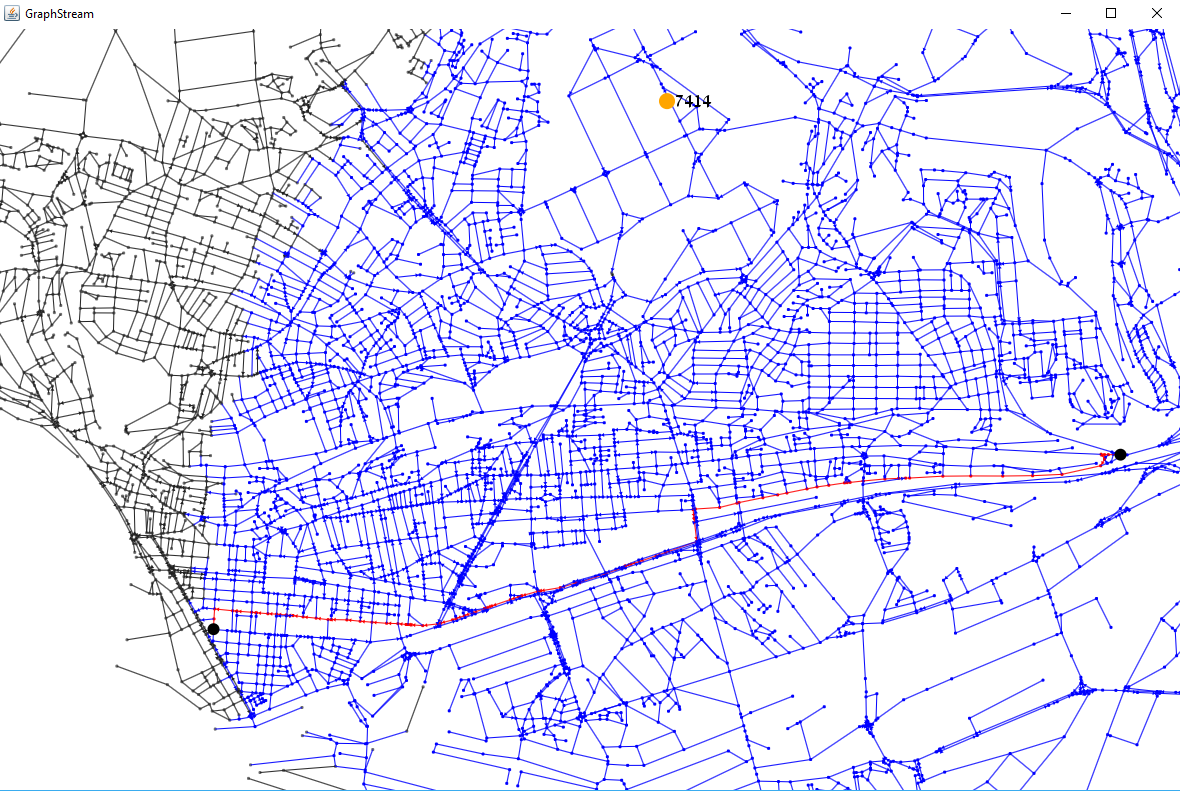
* כמות צמתים שהאלגוריתם חקר והוסיף לדרכים – 83
* כמות צמתים במסלול הקל ביותר – 78
* מרחק כולל של המסלול – 7.48 קילומטרים
* זמן ממוצע שייקח לעבור את המסלול – 10 דק ו52 שניות

**הרצה מס' 6**:

**תנאים מקדימים:**

1. משקל כל קשת נקבע על ידי אורך זמן ממוצע שלוקח לעבור אותה ביחידות מידה של דקות.
2. היוריסטיקת האלגוריתם תהיה " NoDistanceHeuristic".

**תמונת מצב החיפוש:**



**תוצאות החיפוש:**

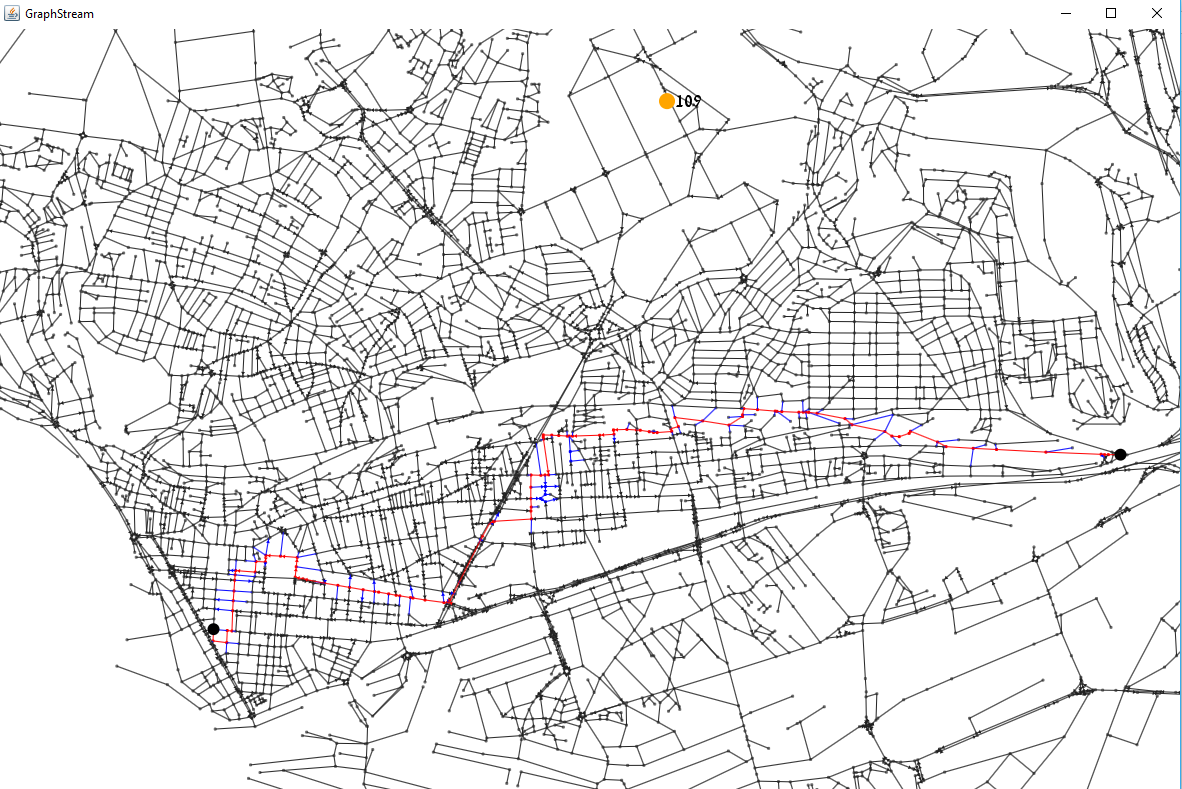
* כמות צמתים שהאלגוריתם חקר והוסיף לדרכים – 7414
* כמות צמתים במסלול הקל ביותר – 84
* מרחק כולל של המסלול – 6.766 קילומטרים
* זמן ממוצע שייקח לעבור את המסלול – 7 דק ו42 שניות

**הרצה מס' 7**:

**תנאים מקדימים:**

1. משקל כל קשת נקבע על ידי אורך זמן ממוצע שלוקח לעבור אותה ביחידות מידה של דקות.
2. היוריסטיקת האלגוריתם תהיה " EuclidianDistanceHeuristic".

**תמונת מצב החיפוש:**



**תוצאות החיפוש:**

* כמות צמתים שהאלגוריתם חקר והוסיף לדרכים – 109
* כמות צמתים במסלול הקל ביותר – 102
* מרחק כולל של המסלול – 8.30 קילומטרים
* זמן ממוצע שייקח לעבור את המסלול – 12 דק ו16 שניות

**ניתוח תוצאות ומסקנות**

**הרצה 1** – זוהי הייתה הרצה של אלגוריתם דייקסטרה (כיוון שאין שימוש בהיוריסטיקה) ניתן לראות כי אכן האלגוריתם ביצע חיפוש והתפרשות לכל הכיוונים, כמו כן האלגוריתם נדרש לבדוק מספר רב של צמתים בכדי לקבוע מהו המסלול הקצר ביותר. כיוון שהרצה זו בעצם מייצגת את אלגוריתם דייקסטרה אשר ידועה נכונותו נתייחס לתוצאות אלה בתור קבוצת בדיקה, אשר תוצאותיה נכונות בוודאות אבל כמות הצמתים שלקחה להגיע אל תוצאות אלה גדולה באופן יחסי.

**הרצה 2 –** נציין כי פונקציית אומדן של "מרחק מנהטן" טובה לתזוזה ב4 כיוונים (צפון, דרום, מזרח ומערב) ושמה נובע מתזוזה ברחובות מנהטן אשר מחולקים לבלוקים. בעקבות שימוש בפונקציית האומדן אלגוריתם החיפוש נעשה יעיל יותר וניתן לראות כי הוא צמצם את טווח החיפוש ביחס לקבוצת הבדיקה כאשר עדיין הצליח לספק את הפתרון האידאלי לבעיה.

**הרצה 3** - פונקציית האומדן של "מרחק אלכסוני" טובה לתזוזה ב8 כיוונים (שושנת הרוחות) לכן נצפה שהיא תספק לנו אומדן יותר טוב מאשר פונקציית מרחק מנהטן כיוון שתזוזה זאת תהיה דומה יותר לתזוזה במציאות בין הצמתים. ניתן לראות כי אכן שימוש בהיוריסטיקה זאת מהווה שיפור ביחס לקבוצת הבדיקה ואף ביחס לשימוש ב"מרחק מנהטן" אך עדיין פונקציה זו קבילה כיוון שקיבלנו את הפתרון האידאלי.

**הרצה 4 –** היוריסטיקה של "מרחק אוקלידי" מהווה את המרחק הקצר ביותר בין שתי נקודות כאשר ניתן לזוז לכל הכיוונים. נצפה שפונקציית אומדן זו תספק את התוצאות הכי טובות כיוון שהיא מדמה בצורה הקרובה ביותר את התנועה האפשרית על גבי הגרף.

ניתן לראות כי שימוש בהיוריסטיקה זאת אפשר לנו לקבל את המסלול הקל ביותר בין שתי הצמתים תוך כדי חיסכון של בכמות הצמתים בהם הוא ביקר בכדי להגיע לתוצאה.

**הרצה 5 –** ניתן לראות כי במקרה זה לא השתמשתי כלל בערך , כלומר במשקל שהצטבר עד לצומת זו. לכן זהו בעצם אלגוריתם חמדן אשר לא יגיע בהכרח לפתרון האידאלי אך הוא יפצה על כך בזה שהוא יגיע לפתרון מספיק קרוב לפתרון האידאלי אבל תוך כדי חסכון עצום בכמות הצמתים שהוא בדק.

באמת התוצאות מאשרות זאת ואלגוריתם זה לא הצליח למצוא את המסלול הקצר ביותר, אלא סיפק תוצאה אשר בינה לבין המסלול הקל ביותר יש הפרש של כ-715 מטרים. כלומר סטייה של זוהי סטייה משמעותית בדרך שהאלגוריתם בחר לעומת קבוצת הבדיקה אבל החסכון בכמות הצמתים שהוא בדק בדרך היא משמעותית ביותר של 99%, דבר אשר יגרור במערכות בעולם האמיתי חיסכון עצום בזמן ריצה.

**הרצה 6** – בהרצה זו השתמשתי במשקל שונה על הקשתות כאשר כעת אני מחפש את המסלול בעל הזמן הקצר ביותר, מסלול אשר כמו במציאות לא בהכרח חייב להיות חופף למסלול הקצר ביותר. הרצת אלגוריתם זה ללא היוריסטיקה היא בעצם הרצת אלגוריתם דייקסטרה אשר בהכרח יחזיר תוצאה אידאלית שלגביה נשתמש בתור קבוצת בדיקה.

ניתן לראות כי אכן המסלול שהתקבל הוא שונה מהמסלול בהרצות 1-4 שהיוו את המסלול הקצר ביותר.

**הרצה 7 –** בהרצה זו השתמשתי בפונקציית אומדן של מרחק מנהטן, בהרצה מספר 2 פונקציית אומדן זו סייעה לנו לצמצם את טווח החיפוש תוך כדי שמירה על מציאת הפתרון האידאלי.

ניתן לראות כי המסלול שהתקבל אינו המסלול הקצר ביותר. כאשר התקבל מסלול עם הפרש של 4:34 דקות ביחס לפתרון האידאלי כלומר סטייה של . כמו כן ניתן לראות כי הגרף נראה דומה מאוד להרצה מס' 5 בה השתמשתי באלגוריתם חמדן שאינו מתייחס למשקל עד כה.

הסבר לתופעה שהתקבלה היא שבחרתי בהיוריסטיקה לא מתאימה למציאת הזמן הקצר ביותר. התוצאה של חישוב מרחק מנהטן במטרים תהיה גדולה בצורה משמעותית ביחס למדידה של זמן בדקות. לכן מבחינת האלגוריתם הזמן שהצטבר עד כה הוא זניח ביחס לאומדני המרחק. כלומר הוא יתנהג בצורה דומה לאלגוריתם החמדן.

פתרון אפשרי לבעיה הוא שנבחר בהיוריסטיקה שלוקחת בחשבון את יחס הפרופורציה בין הגדלים ומחלקת את המרחק של האומדן ביחס זה בכדי שהיא אכן תהיה קבילה ולא יתקבל אומדן גדול מידי.

**חלק ב' – מציאת תחנת דלק הקרובה ביותר**

**תיאור הבעיה:**

**נתון:** גרף מכוון עם פונקציית משקל אי שלילית אשר מייצג מערכת כבישים. כמו כן נתונים קבוצה של צמתים אשר מייצגת קבוצה של תחנות דלק.

**צריך למצוא:** אלגוריתם יעיל ככל האפשר למציאת המסלול הקצר ביותר מכל צומת אל עבר תחנת דלק.

**הסקת אבחנות:**

אבחנה מס' 1: יש את אלגוריתם דייקסטרה אשר הוכחה לו הנכונות, למציאת המרחק הקצר ביותר מצומת מוצא לעבר כל הנקודות האחרות בגרף. אנסה לתמרן את הגרף כך שהוא יספק את התנאים עבור כל צומת.

אבחנה מס' 2: המרחק בין שתי צמתים זהה למרחק , כלומר ניתן להסתכל על המרחק מכל תחנת דלק אל כל צומת בגרף.

אבחנה מס' 3:אם אסתכל על הגרף המשוחלף כך ש-

(כלומר החלפתי את הכיוון של כל קשת). אזי הבעיה נעשית פשוטה יותר, למצוא את המרחק הקצר ביותר מכל תחנת דלק לעבר כל צומת אחרת בגרף.

אבחנה מס' 4: ניתן לנסח פתרון "נאיבי" עבור הבעיה הפשוטה יותר כך שנעבור על כל צומת בקבוצה של תחנות הדלק ונריץ ממנה אלגוריתם דייקסטרה, ולכל צומת נבחר את הערך המינימלי מכל ההרצות. בסיבוכיות זמן לא יעילה במיוחד של

אבחנה מס' 5: נציין כי ניתן לחסוך הרצות מיותרות על ידי כך שנוסיף לגרף המשוחלף צומת חדשה אשר אותה נחבר לכל צומת ונמשקל את כל הקשתות החדשות במשקל 0. כעת אם נריץ אלגוריתם חיפוש דייקסטרה הרי שכל תחנת דלק תקבל עדיפות זהה מבחינת מרחק והתוצאה הסופית תהיה שלכל צומת נקבל את המשקל המינימלי מ כלומר בעצם המשקל המינימלי מכל תחנת דלק.

**בניית אלגוריתם:**

כעת בעזרת האבחנות יש בידי מספיק כלים בכדי לנסח אלגוריתם שיפתור את הבעיה בצורה יעילה:

1. שחלף את הגרף
2. הוסף לגרף החדש צומת
3. הוסף קשתות מצומת לעבר כל צומת
4. הגדר משקל 0 לכל הקשתות החדשות
5. תריץ אלגוריתם דייקסטרה מצומת

**ניתוח סיבוכיות:**

אעבור שלב שלב על האלגוריתם ואנתח את הסיבוכיות שלו, לאחר מכן אסכום את כל הסיבוכיות בכדי להגיע לסיבוכיות הכוללת של האלגוריתם:

1. סיבוכיות שחלוף הגרף תלויה בדרך המימוש שלו, בהנחה שבחרנו לממש את הגרף באמצעות רשימת שכנויות אזי הסיבוכיות של השחלוף תהיה – מעבר על כל הרשימה ובניית רשימה חדשה.
2. הוספת צומת חדשה תעשה בסיבוכיות זמן
3. הוספת קשתות תהיה כגודל הקבוצה של תחנות הדלק, כלומר בסיבוכיות זמן של (כיוון ש)
4. ניתן לבצע את משקול הקשתות ביחד עם שלב מס' 3 ומשקול כל קשת יעשה ב לכן לא נוספה סיבוכיות זמן.
5. הרצת אלגוריתם דייקסטרה יעשה בסיבוכיות של

**סיכום:** ניתן לראות כי הסיבוכיות הכוללת של האלגוריתם תהיה סיבוכיות של אלגוריתם דייקסטרה כלומר

**רשימת מקורות**

* Hart, P. E.; Nilsson, N. J.; Raphael, B. (1972). "Correction to "A Formal Basis for the Heuristic Determination of Minimum Cost Paths"". SIGART Newsletter.
* <http://graphstream-project.org/>
* [Efficient Point-to-Point Shortest Path Algorithms"](http://www.cs.princeton.edu/courses/archive/spr06/cos423/Handouts/EPP%20shortest%20path%20algorithms.pdf) from Princeton University
* Dechter, Rina; Judea Pearl (1985). "Generalized best-first search strategies and the optimality of A\*". Journal of the ACM. **32** (3): 505–536