בסייעתא דשמיא

פרויקט גמר

תשע"ו- 2016

גילה אלקרייף 316957406

צילי שטיינבך 315634980

תוכן עניינים:

מבוא

בבואנו לבחור נושא לפרויקט הגמר, חיפשנו תחום שגם יהיה מעניין ומאתגר וגם יתרום ויביא תועלת.

במהלך חיי היומיום נתקלים רבות בבעיות חיפוש והתמצאות בשטח. על האדם למצוא את דרכו במקום חדש ובלתי מוכר, או למצוא דרך התואמת למגבלות הזמן והתקציב שלו.

לדוגמא: מבקר במוזאון המעוניין להספיק לבקר בכמה שיותר חדרי תצוגה אך זמן הביקור שלו מוגבל, מעוניין למצוא את המסלול המכיל מקסימום תחנות אך ארכו לא עולה על הזמן המוקצב לו. או מטייל בשמורת טבע שהחנה את רכבו בכניסה לאתר ומעוניין לשוב אליו בסיום הטיול, מעוניין למצוא מסלול מעגלי המתחיל ומסתיים בחנייה. גם מבקרים ביריד בו כל כניסה לחדר פעילות כרוכה בתשלום קבוע כלשהוא ותקציבם מוגבל, יהיו מעוניינים למצוא מסלול באורך מינימלי הכולל בתוכו את חדרי הפעילות בהם הם חפצים לבקר.

חיפשנו כלי שיוכל לעזור לנו לפתור בעיות אלו ולהפוך את חוויית הביקור באתרים לנעימה ונוחה יותר, וחשבנו על מבנה הנתונים גרף התואם באופן מושלם לייצוג ומיפוי אתרים ומאפשר פונקציות חיפוש והתמצאות בצורה יעילה וברורה.

באתר שפיתחנו ישנה אפשרות למנהל מרכז מבקרים ליצור את מפת המקום בעזרת הספרייה הממשת את מבנה הנתונים גרף, ולמבקרים בו לערוך עליה חיפושים בהתאם לצרכיהם בעזרת פונקציות החיפוש המוגדרות בספרייה.

תקציר

הפרויקט מכיל מספר חלקים:

הספרייה Graph-

ספרייה גנרית הממשת את מבנה הנתונים גרף.

הספרייה כוללת מחלקות המפשטות את מבנה הנתונים: צומת, קשת ודרך, וכן פונקציות המוגדרות על מבנה נתונים זה: חיפושים שונים, סריקה ועוד.

הספרייה CodeDOM-

ספרייה המספקת פונקציות לבניית מחלקה בזמן ריצה.

הספרייה מכילה פונקציות המוסיפות למחלקה הנבנית את המאפיינים הרצויים והפונקציות הנבחרות, מהדרת את המחלקה ושומרת אותה כקובץ CS וכקובץ DLL.

אנו משתמשים בספרייה זו בפרויקט על מנת לאפשר למנהל אתר לתאר באופן המדויק ביותר את האתר שלו, לנוחות המבקרים בו.

הממשק-

מתוך הבנה שאחד השימושים המתאימים ביותר לנושא הגרפים הוא תיאור של אתרים וקשרים בין צמתים באתר נבנה ממשק המאפשר למנהל אתר לתאר את מבנה האתר שלו ולספק למשתמשים- לקוחותיו סביבה נוחה להתמצאות באתר שלו. הסביבה פותחה בסביבת web ומאפשרת למנהל האתר לקבוע מהם המאפיינים של המקום ואלו פונקציות חיפוש יכולות להיות יעילות ושימושיות למבקרים בו.

האתר מובנה בצורה שמהווה ממשק ידידותי למשתמש, עם הסברים והוראות ברורות לשימוש.

לאחר שהמנהל מסיים את יצירת המפה הוא מקבל לידיו קבצים המתארים אותה- קובץ HTML המכיל את ציור מפת האתר, קובץ נתוני המפה (מידע על כל צומת וכל קשת בגרף) וכן את הספרייה המכילה את פונקציות החיפוש.

הפיתוח-

הספרייה Graph פותחה בסביבת עבודה NET. בשפת # C, שפה מתקדמת ומלאת פונקציונאליות. כמו כן ,הספרייה נעזרת במחלקות ורכיבים שונים שהיה עלינו ללמוד, להכיר ולהתמצא, כגון עבודה עם מחלקות גנריות.

הספר –

המעיין בספר זה ילמד על אופן פעילות המחלקות והפונקציות שמספקות הספריות, אופן השימוש באתר ובניית מפה, האתגרים המיוחדים בהם נתקלנו במהלך הפיתוח ואופן פתרון הבעיות שעלו.

תיאור הפרויקט

מרבית אתרי המבקרים הגדולים אינם מורכבים ממרכז אחד ויחיד, אלא יש בהם מספר תחנות שונות והמבקרים באתר בוחרים באלו הם מעוניינים לבקר. לכל מבקר תהינה העדפות אחרות: אחד יחפש מסלול באורך מוגבל, האחר יעדיף דרך מעגלית- שחוזרת לנקודת המוצא, ועוד קריטריונים רבים המשתנים מאדם לאדם ומאתר לאתר.

<u>בממשק למנהל</u> יכול כל מנהל אתר לתאר את האתר הספציפי שלו ולקבוע איזה מידע הוא מעוניין לספק למבקרים במקום. המידע מחולק לשניים: תיאור תחנה ותאור דרך באתר.

כמו כן המנהל יוצר את מפת המקום באמצעות שימוש בכלי ציור וקובע פרטים אודות כל תחנה ודרך (הפרטים אותם בחר לספק למבקר).

בספרייה ישנן אפשרויות חיפוש רבות ומגוונות העשויות להתאים לסוגי אתרים שונים, ובונה המפה יכול לבחור אלו פונקציות חיפוש הוא מעוניין לאפשר למבקרים באתר שלו.

<u>הממשק למבקר</u> מציג את המפה כפי שנבנתה וכן את פונקציות החיפוש שבחר בונה המפה לאפשר, והמשתמש המעוניין למצוא מסלול בהתאמה להעדפותיו ומגבלותיו בוחר בחיפושים המתאימים לו. אם נמצא/ו מסלול/ים מתאימים הם יסומנו על גבי המפה.

הספרייה Graph מממשת את מבנה הנתונים גרף, שהוא המבנה המתאים ביותר לתיאור ומיפוי אתרים. כל תחנה במפה היא צומת בגרף, וכל דרך בין תחנות במפה היא קשת בגרף.

גם אלגוריתמי הסריקה של גרף והחיפושים בו מתאימים לביצוע חיפושים במפה.

כאשר מנהל גורר תחנה או דרך נוספת לציור המפה, מופעלות הפונקציות המתאימות בספרייה- הוספת צומת/ קשת, וכדומה.

מובן כי לכל אתר מאפיינים שונים וכל מנהל הבונה מפה מעוניין כי היא תכיל מידע ספציפי על כל תחנה וכל דרך, מעבר למידע הנחוץ על מנת למצוא דרך. לשם כך נבנתה הספרייה Graph באופן גנרי, והמחלקות המתארות את האתר הספציפי נבנות בזמן ריצה על ידי המנהל, שקובע מהם מאפייני צומת וקשת באתר שלו. כלומר, מהו המידע אותו הוא מעוניין לספק למבקרים.

כאשר מבקר בוחר בחיפוש מסוים הוא מפעיל פונקציית חיפוש מהספרייה, המבצעת את החיפוש על הגרף שבנה המנהל.

אחד האתגרים העיקריים במהלך הפיתוח היה יצירת המחלקות בזמן ריצה. לשם כך היה עלינו ללמוד רבות אודות האפשרויות הקיימות והפונקציות הבנויות הנכללות במסגרת CodeDom, ולהשתמש בהם בהתאם לצרכינו.

קושי נוסף התעורר בכתיבת וקריאת קובץ הXML שהינו דינאמי והמבנה שלו לא מוכר ולא ידוע מראש, אלא בנוי על פי המחלקות הנבנות בזמן ריצה. נעזרנו במימוש הממשק ISerializable על מנת לפתור את הבעיה.

מפרט טכני

מפרט כללי

סביבת פיתוח

עמדת פיתוח: מחשב Intel

מערכת הפעלה: Windows 7

שפות תכנות: C#, JQuery

עמדת משתמש מינימאלית:

חומרה: מחשב

מערכת הפעלה: Windows 7

חיבור לאינטרנט: נדרש.

למשתמש בספריה:

Microsoft Visual Studio 2013

MVC

כלי התכנה לפיתוח הספרייה והתוכנה

Microsoft Visual Studio 2013

נושאים באחריות המערכת

תחומי הספרייה Graph:

- הספרייה מתאימה לייצוג גרפים קשירים, כלומר: גרפים בהם כל הצמתים מחוברים לקבוצה אחת באמצעות קשתות. (אין מספר קבוצות של קשתות)
- הספרייה מתאימה לייצוג גרפים שאין בהם דרך בעלת משקל שלילי.

תחומי הספרייה CodeDom:

- הספרייה מאפשרת הוספת מאפיינים למחלקה הנבנית (על פי קלט מהמשתמש), הוספת הפונקציות Equals, ToString וכן קומפילציה (הידור) של המחלקה ויצירת קבצי DLL ו CS המכילים אותה.

תחומי האתר:

- האתר מתאים למיפוי אתרים בהם כל התחנות מקושרות באופן כלשהוא (ניתן לייצגם באמצעות גרף קשיר).

מטרות המערכת

עבור מנהל:

- המערכת תאפשר ליצור מחלקה מתארת ייחודית עבור כל אתר.
- המערכת תאפשר לבונה המפה לבחור אילו חיפושים מבין החיפושים המוצעים בספרייה יתאימו למבקרים באתר שלו.
- המערכת תציע פונקציות חיפוש רבות ומגוונות המתאימות לאתרים שונים ולמחפשים שונים, לבחירת המנהל איזו מהם רלוונטית ושימושית לאתר שלו.
- המערכת תספק לבונה המפה קבצים המתארים את המפה שהוא בנה, את המידע על כל פריט במפה ואת הספרייה המאפשרת ביצוע חיפושים עליה.

עבור מבקר:

- המערכת תאפשר למבקרים באתר לחפש מסלול כרצונם באמצעות פונקציות החיפוש המוצעות.
- המערכת תאפשר גישה למפה למטרת עריכת חיפושים דרך קישור מאתר קיים כלשהוא שעבורו נבנתה המפה או דרך טעינת מפה קיימת.

יעדי המערכת

יעדים לספריה Graph

- הספרייה תספק מחלקות ופונקציות לייצוג מבנה הנתונים גרף.
- הספרייה תספק מחלקות חלקיות המשמשות את המחלקה Graph- מחלקה המייצגת דרך ומחלקה המייצגת צומת, וכן מחלקה המייצגת דרך בגרף.
- הספרייה תספק אפשרויות חיפוש מסוגים שונים, כגון הגבלות אורך, הגבלות מחיר, חיפוש המסלול הקצר ביותר ועוד רבות.

יעדים לספרייה CodeDom

- הספרייה תספק פונקציות לבניית מחלקה בזמן ריצה:
- הספרייה תאפשר בניית מחלקה בזמן ריצה על פי פרמטרים מהמשתמש: שם המחלקה ורשימת מאפיינים המכילה צמדים של שם המאפיין וסוגו.
- הספרייה תאפשר עריכת קומפילציה (הידור) למחלקה שנבנתה.
- הספרייה תאפשר יצירת קובץ SC וקובץ DLL המכילים את המחלקה שנבנתה.

כל זאת בעזרת שימוש בפונקציות הכלולות ב:

- System.CodeDom
- System.CodeDom.Compiler

יעדים לממשק המשתמש

<u>ממשק למנהל:</u>

- בנית מרחב לאפיון תחנות ודרכים באתר.
- בנית פונקציות שרות שונות לצורך חיפוש באתר

<u>ממשק למבקר:</u>

- רשימת פונקציות החיפוש השונות העומדות לבחירת המשתמש.
- תצוגה של המסלול שנמצא על פי הגבלות המשתמש (פרמטרים לפונקציות החיפוש)

מבנה המערכת

המערכת מורכבת ממספר חלקים:

הספרייה Graph-

ספריית Graph הינה ספריה המייצגת את מבנה הנתונים גרף ומאפשרת שמירת נתונים המתאימים למבנה זה וניהולם באופן יעיל, ומיועדת לכל מתכנת המעוניין לאחסן ולנהל נתונים במבנה של גרף (בהתאם לתיחום הספרייה).

ייחודה של הספרייה הינו החלק הגנרי המאפשר בקלות להתאים את הגרף לסוגים שונים של אובייקטים, כך שיוכל לייצג מגוון רחב ודינאמי של עצמים, את אלו הבסיסיים והמובנים ואת אלו שיצור או ייבא המשתמש בספריה.

הספרייה כוללת מחלקות המפשטות את מבנה הנתונים:

- Arc- מחלקה המייצגת קשת בגרף.
- Node- מחלקה המייצגת צומת בגרף.
- Way- מחלקה המייצגת דרך בגרף, מורכבת מרשימת הצמתים שבדרך ומרשימת הקשתות שבה, וכן את אורך הדרך.
- Graph- מחלקה המייצגת גרף, מורכבת מרשימת קשתות, רשימת צמתים, פונקציות ניהול לספריה וכן פונקציות חיפוש מסלולים בגרף על פי קריטריונים שונים.

ניתן לומר שהספרייה מאגדת בתוכה מחלקות ופונקציות בנויות על מנת לתאר גרפים ולערוך בהם חיפושים.

הספרייה CodeDOM-

מטרתה לייצר מחלקה בזמן ריצה על פי מספר פרמטרים:

- שם המחלקה.
- רשימת המאפיינים הרצויים במחלקה הנבנית, כאשר עבור כל אחד מהם מוגדר שם השדה וסוגו.

הספרייה מכילה מספר פונקציות הבונות את מרכיבי המחלקה:

- AddProperties- פונקציה שיוצרת במחלקה עבור כל מאפיין משתנה המייצג אותו, וכן פונקציות של Get ו Set , המאפשרות גישה למידע.
- AddMethods- פונקציה המוסיפה למחלקה פונקציות ניהול, כגון ,ToString Equalsועוד.

- GenerateCSharpCode- פונקציה היוצרת קובץ CS המכיל את המחלקה הכתובה בשפת #C.
- CompileCSharpCode- פונקציה המהדרת את המחלקה שנוצרה, ויוצרת ממנה קובץ DLL על מנת שניתן יהיה לייבאה לפרויקטים אחרים, ולקשרה עם הספרייה Graph.

הממשק-

מתוך הבנה שאחד השימושים המתאימים ביותר לנושא הגרפים הוא תיאור של מפות וקשרים בין צמתים במקום נבנה ממשק המאפשר למנהל אתר לתאר את מבנה האתר שלו ולספק למשתמשים- לקוחותיו סביבה נוחה להתמצאות באתר שלו. הסביבה פותחה בסביבת web ומאפשרת למנהל האתר לקבוע מהם המאפיינים של המקום ואלו פונקציות חיפוש יכולות להיות יעילות ושימושיות למבקרים בו.

האתר מובנה בצורה שמהווה ממשק ידידותי למשתמש, עם הסברים והוראות ברורות לשימוש.

לאחר שהמנהל מסיים את יצירת המפה הוא מקבל לידיו את כל הקבצים הנצרכים על מנת לאפשר למבקרים לבצע חיפושים.

האלגוריתמים המרכזיים

1.הספרייה Graph

ספרייה לייצוג מבנה הנתונים גרף.

גרף : מורכב מקבוצה סופית לא ריקה של צמתים ומקבוצת קשתות.

הגדרות בגרף:

1. גרף מכוון- גרף בו כל הקשתות מכוונות (קשת שמובילה מצומת קצה א' לצומת קצה ב', לא מובילה מצומת קצה ב' לצומת קצה א').
2. מעגל בגרף- מסלול בגרף מקדקוד כלשהוא לעצמו.
3. גרף קשיר- גרף בו קיים מסלול בין כל שני קדקודים.
4. קשת מכוונת- קשת המובילה מצומת קצה א' לצומת קצה ב'.
5. קשת שאינה מכוונת- קשת המובילה מצומת קצה א' לצומת קצה ב', וגם מצומת קצה ב' לצומת קצה א'.
6. גרף משוקלל- גרף בו הקשתות מכילות פונקציית משקל.

פונקציית משקל לקשת: במרבית הגרפים ישנו מחיר לכל מעבר בין צמתים. פונקציית המשקל של הקשת העוברת מצומת א' לצומת ב', היא מחיר המעבר מצומת א' לצומת ב'.

לדוגמא: אם המעבר מעיר א' לעיר ב' עולה סכום מסוים (עלות הדלק, כביש אגרה, אורך הנסיעה וכו'), אזי משקל הקשת המובילה מעיר א' לעיר ב' יהיה אותו הסכום.

גרף שאינו משוקלל הוא גרף בו אין פונקציות משקל על הקשתות. כלומר, הקשתות בעלות משקל 0.

חיפושים בגרף:

- חיפושים על פי משקל - שימושיים בעיקר עבור גרף משוקלל
  - מסלול בהגבלת משקל מקסימום
  - מסלול בעל משקל מינימלי
- חיפושים על פי אורך- שימושיים עבור גרף שאינו משוקלל וגם עבור גרף משוקלל
  - מסלולים בהגבלת אורך מקסימום
  - מסלול בעל אורך מינימלי
- חיפוש מעגלים
  - מציאת כל המעגלים בגרף
  - מציאת כל המעגלים מנקודת מוצא מסוימת.

הספרייה שבנינו מכירה את המחלקות הגנריות TArc ו - TNode, זאת על מנת לאפשר למשתמש/ המתכנת לבנות גרף מותאם אישית. כלומר, מאחר ובכל מימוש של גרף יש לשמור תוספת מידע לצמתים וגם לקשתות, מידע שקשור למימוש עצמו, הגרף הוא גנרי ומאפשר לבצע את כל הפעולות על מחלקות שיוגדרו ע"י המשתמש.

<u>לדוגמא</u>: גרף המתאר מפת כבישים צריך לכלול מידע נוסף על המידע ההכרחי לניהול הגרף וביצוע חיפושים בו, כגון שם הכביש, הגבלת מהירות ועוד.

גרף המתאר מפת גן-חיות צריך לכלול מידע אחר, כגון שם החיה, מספר הפרטים שבכלוב, שעות האכלה ועוד רבות.

כמו כן מכילה הספרייה מחלקה המייצגת דרך (מסלול) בגרף.

את הדרך בונה פונקציית החיפוש שנבחרה, והיא הערך המוחזר מהפונקציה.

<u>אופן ייצוג הגרף:</u>

ישנן אפשרויות שונות לייצוג מבנה הנתונים גרף.

בחרנו באופן ייצוג התואם לעקרונות ה OOP, וכן מאפשר בנייה דינאמית של הגרף, שלא כמו ייצוגים רבים אחרים בהם לא ניתן לשנות את גודל הגרף.

הספרייה מורכבת מהמחלקות צומת (Node), קשת (Arc), דרך (Way) וגרף(Graph).

<u>גרף מכיל:</u>

- רשימת צמתים (אובייקטים מסוג צומת)
- רשימת קשתות (אובייקטים מסוג קשת)
- מונה מספר הצמתים בגרף
- מונה מספר הקשתות בגרף

כמו כן הגרף מכיל פונקציות ניהול הספרייה:

הוספה ומחיקה,

חיפושים של צומת/ קשת מסוימת ועוד.

ניתן להוסיף קשתות וצמתים (בתנאי שאינם נמצאים בגרף), וכן למחוק קשתות וצמתים. כאשר נמחקת צומת, נמחקות כל הקשתות שהגיעו אליה/ יצאו ממנה.

צומת מכילה:

- מאפיין מסוג TNode, המכיל את כל המאפיינים הפרטיים של הצומת (כפי שבנה המנהל).
- רשימת הקשתות היוצאות מהצומת (רשימת אובייקטים מסוג קשת).

קשת מכילה:

- מאפיין מסוג TArc, המכיל את כל המאפיינים הפרטיים של הקשת (כפי שבנה המנהל).
- צומת המקור (אובייקט מסוג צומת).
- צומת היעד (אובייקט מסוג צומת).
- כיוון (קשת יכולה להיות מכוונת/ לא מכוונת).
- משקל.

במקרה והקשת אינה מכוונת אין משמעות לאבחנה בין צומת המקור לצומת היעד, אלא שניהם צמתי קצה פשוטים.

דרך מכילה:

- רשימת הצמתים הנמצאים בדרך.
- צומת המקור (ההתחלה) של הדרך.
- צומת היעד (הסיום) של הדרך.
- אורך הדרך (מספר הקשתות).
- המשקל הכללי של הדרך.
- מעגלית (כן/לא).

כל פונקציות החיפוש שניתן להפעיל על הגרף נמצאות במחלקה גרף (Graph).

דיאגרמת המחלקות בספרייה:

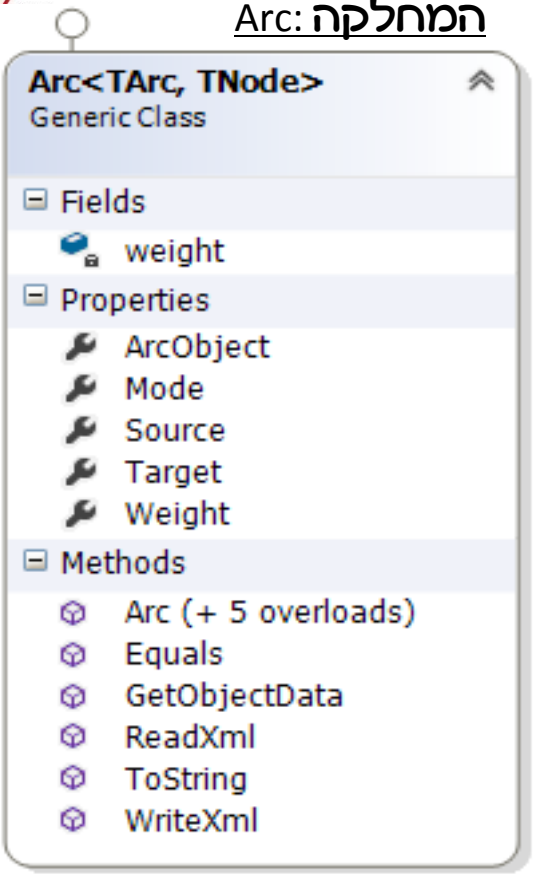
המחלקה Graph:

Graph<TNode, TArc>
Generic Class

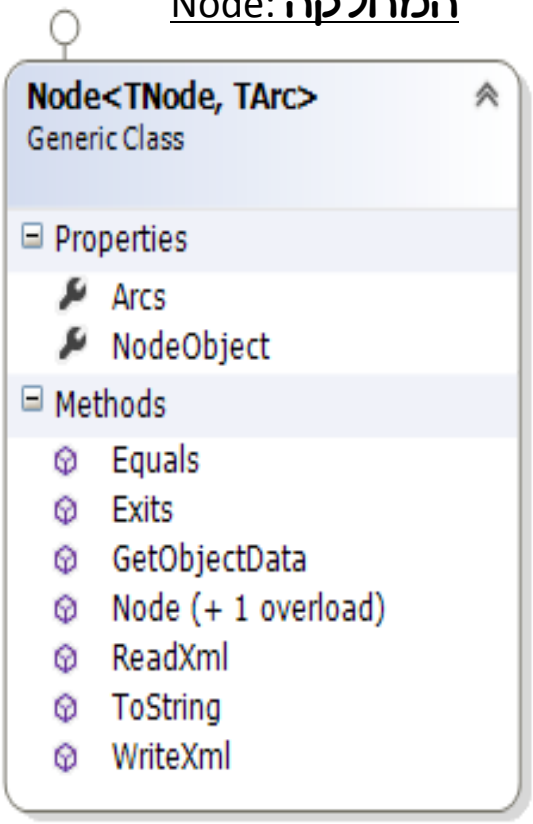
Properties
- AmountArcs
- AmountNodes
- Arcs
- Nodes
- this (+ 2 not shown)

Methods
- AddArc (+ 5 overloads)
- AddNode
- Adjacent
- AllCircleWays
- AllConnected
- BFS_LayerSearch
- Circle
- CircleWay
- Dijkstra
- DoWorkOnArcs
- DoWorkOnNodes
- Equals
- FindArcs
- FindNodes
- GetCheapestWay
- GetCheapestWaysFromSource
- GetCheapestWaysToTarget
- GetShortestWay
- GetShortestWaysFromSource
- GetShortestWaysToTarget
- GetWay
- GetWaysLimitedByWeightFromSource
- Graph
- IsEmpty
- LimitedLengthWayFromSource
- LimitedWeightWayFromSource
- ReadXml
- RemoveArc
- RemoveNode
- ToString
- WaysLimitedByLength
- WaysLimitedByLengthFromSource
- WaysLimitedByLengthToTarget
- WaysLimitedByWeight
- WaysLimitedByWeightToTarget
- WriteXML

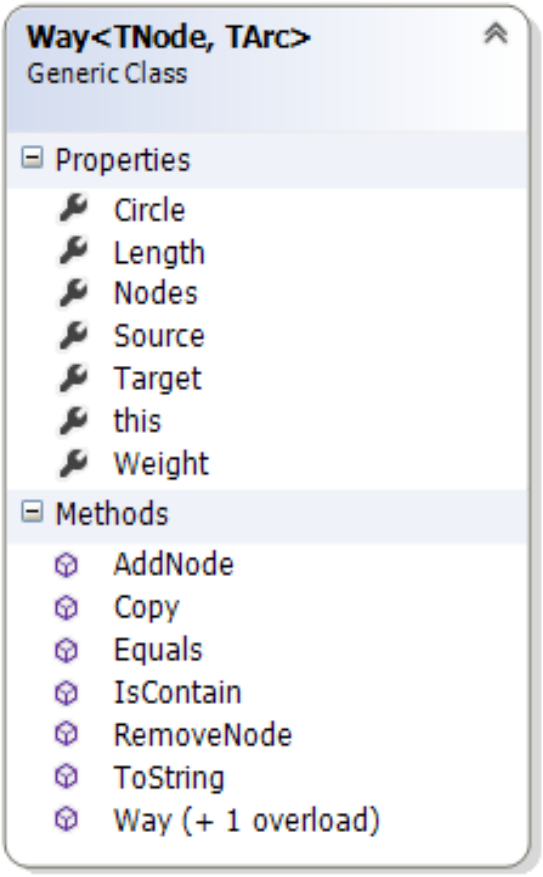
המחלקה: Arc



המחלקה: Node



המחלקה: Way



האלגוריתמים המרכזיים בספרייה Graph

הספרייה שלנו תומכת במספר אלגוריתמים מקובלים של פעולות בגרף:

- חיפוש מסלולים קצרים (הכוללות מספר מינימלי של דרכים) המשתמש בעץ פורש BFS (Breadth-First Search.) .
- חיפוש מסלולים שמשקל הקשתות הכולל שלהם הוא מינימלי, באמצעות שימוש באלגוריתם דיקסטרה (Dikstra).
- מציאת מעגלים (רכיבי קשירות חזקים) בגרף.

כמו כן הספרייה מכילה פונקציות בסיסיות לניהול הגרף, כגון הוספה ומחיקה של צומת/ קשת.

פונקציות החיפוש השונות מהוות מימושים לאלגוריתמי חיפוש בגרף.

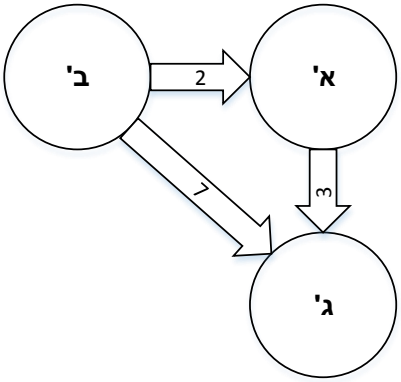
1. *אלגוריתם דיקסטרה (Dikstra):*

אלגוריתם זה מיועד למציאת דרכים בעלות משקל נמוך ביותר (עלות מינימלית), מצומת מקור לשאר הצמתים.

<u>האלגוריתם:</u>

במהלך האלגוריתם נשתמש ב 2 קבוצות של צמתים- קבוצת הקבועים וקבוצת הזמניים, כאשר כל צומת מקבוצת הזמניים שנמצא עבורה מסלול בעל משקל מינימלי <u>סופי</u> עוברת לקבוצת הקבועים.

הערה: לא תמיד קשת ישירה בין שני צמתים מהווה את המסלול בעל המשקל המינימלי ביניהם. לדוגמא: בגרף הבא הדרך בעלת המשקל המינימלי מצומת ב' לצומת ג' אינה הדרך הישירה, אלא זו העוברת דרך צומת א'.



שלב ראשון:

עבור רשימת הצמתים שבגרף, נגדיר:

- רשימת צמתים שתכיל עבור כל צומת את הצומת ההורה שלו במסלול, בהתאמה.
- רשימה שתכיל עבור כל צומת, בהתאמה, את המשקל הזמני של המסלול מצומת המקור עד לצומת זו.

הנחות יסוד:

1. אם קיימת קשת ישירה מצומת א' לצומת ב', משקל המסלול המינימלי הזמני הוא משקל הקשת העוברת מצומת א' לצומת ב'.
   משקל המסלול מקדקוד כלשהוא לעצמו הוא קבוע ושווה לאפס.
2. אם לא קיימת קשת ישירה מצומת א' לצומת ב', לא בהכרח תמצא דרך באורך כלשהוא המגיעה מצומת א' לצומת ב'.
   לכן, משקל המסלול המינימלי הזמני בין צומת א' לצומת ב' הוא המשקל המינימלי הגרוע ביותר: ∞ (אינסוף).

נכנה את משקל המסלול מצומת המקור לצומת x, כמשקל הצומת x.

שלב שני:

עבור צומת המקור נקבע:

- צומת ההורה של צומת המקור הוא NULL.
- המשקל של צומת המקור הוא תמיד אפס.

עבור כל צומת הסמוכה לצומת המקור, נקבע:

- הצומת ההורה במסלול היא צומת המקור.
- משקל הצומת שנבחרה הוא משקל הקשת שבין צומת המקור אליה.

שלב שלישי:

בשלב זה יש לשפר את המסלולים מצומת המקור לכל שאר הצמתים, כך שיהיו המסלולים בעלי המשקל המינימלי.

שלב זה הינו איטרטיבי, כל עוד קבוצת הצמתים הזמניים אינה ריקה.

צעד 1: מתוך קבוצת הצמתים הזמניים נבחר את הצומת שמשקלה מינימלי, אותה נכנה בשם k. נעביר את הצומת k מקבוצת הזמניים לקבוצת הקבועים.

<u>צעד 2:</u> בצעד זה נבדוק האם ניתן לשפר את משקלי שאר הצמתים, בעזרת מסלולים העוברים דרך הצומת k. כלומר, עבור כל צומת x הסמוכה לk, נבדוק מה ייתן את המשקל הנמוך יותר: המשקל העכשווי של צומת x, או המשקל של k+ המשקל של הקשת מ k ל x. אם המשקל הנמוך הוא המשקל של k+ משקל הקשת מk לx, הרי שניתן לבצע הקלה: נעדכן את משקל הצומת x למשקל הנמוך ונקבע את הצומת k לצומת ההורה של הצומת xבמסלול המינימלי הזמני.

כעת ייתכן והשתנו ערכי המשקלים של חלק מהצמתים שבקבוצת הזמניים, ויש לחזור לתחילת השלב השלישי.

<u>שלב רביעי:</u>

כאשר קבוצת הזמניים ריקה, הגענו למסלול האופטימלי מצומת המקור עבור כל צומת בגרף.

בניית המסלול מצומת המקור לצומת x תעשה באמצעות רשימת הצמתים ההורים:

צומת ההורה של הצומת x תהיה הצומת האחרונה במסלול, וצומת ההורה שלה- תהיה הצומת הקודמת לה במסלול, וכן הלאה עד שמגיעים לצומת המקור, לה אין צומת הורה.

<u>מימוש האלגוריתם:</u>

```
פונקציה המוצאת את המסלול הקצר מקדקוד המקור לשאר הקדקודים, על פי אלגוריתם דיקסטרה
public Node<TNode, TArc>[] Dijkstra(Node<TNode, TArc> source)
{
    רשימת הקדקודים שמרחקם מקדקוד המקור הוא זמני כלומר: יתכן ותמצא דרך זולה יותר
                                                                    בהמשך
        List<Node<TNode, TArc>> temporaries = new List<Node<TNode,
        TArc>>();
        temporaries.AddRange(Nodes.FindAll(a => !a.Equals(source)));
    רשימה שמייצגת את המרחקים בין צומת המקור לבין כל צומת אחרת שנמצאת ברשימת
                                                    הזמניים, על פי סדר
        List<double> distance = new List<double>();
    מערך המייצג את הקדקוד הקודם של כל קדקוד במסלול הקצר ביותר, בהקבלה לרשימה
                                                                    Node
        Node<TNode, TArc>[] parent = new Node<TNode,
        TArc>[AmountNodes];
```

```
אתחול המערך במשקל הקשת שבין המקור לצומת במקום האינדקס.במקרה שאין קשת
אתחול בערך גבוה
for (int i = 0; i < temporaries.Count; i++)
{
      Arc<TArc, TNode> arc = this[source, temporaries[i]];
      if (arc == null)
            distance.Add(double.MaxValue);
      else
      {
            distance.Add(arc.Weight);
   קביעה שצומת הורה של הצומת הנוכחית במסלול הזמני היא צומת המקור
            parent[this[temporaries[i]]] = source;
      }
}
//צעד ראשון
while (temporaries.Count >= 1)
{
            מציאת הצומת מרשימת הזמניים שהדרך אליה זולה ביותר
                        לייצוג האינדקס של הצומת שתימצא
      int k = 0;
                  יכיל את המשקל המינימלי עד כה -לצורך החיפוש
      double min = distance[0];
      for (int i = 1; i < distance.Count; i++)
      {
            if (distance[i] < min)
            {
                  min = distance[i];
                  k = i;
            }
      }

                                                צעד שני
בדוק האם ישנה דרך זולה יותר -העוברת דרך הצומת שנמצאה k עבור כל צומת
                                                זמנית:
      for (int i = 0; i < temporaries.Count; i++)
      {
   אם קימת קשת בין הצומת הזמנית אי לצומת קי -אזי:
            if (this[temporaries[k], temporaries[i]] != null)
            {
                  double m = Math.Min(distance[i], min +
                  this[temporaries[k],
         מצא את המינימלית מבין הדרכים
                  temporaries[i]].Weight);
         אם נמצאה דרך זולה יותר– אזי:
                  if (distance[i] > m)
                  {
               שנה את המרחק ועדכן את parent
```

```
                                distance[i] = m;
                                parent[this[temporaries[i]]] =
                                Nodes.Find(a =>
                                a.Equals(temporaries[k]));
                    (לא ניתן לגשת דרך רשימת הזמניים כי הצומת תמחק ממנה)
                            }
                        }
                    }
                    מחיקת הצומת מרשימת הזמניים
                    temporaries.Remove(temporaries[k]);
                    distance.Remove(distance[k]);
                }
                return parent;
            }
```

2. אלגוריתם סריקה לרוחב (BFS):

פעולה חשובה ושכיחה בגרף היא סריקת כל צמתיו. בעת הסריקה, נעבור על פני הגרף כך שנבקר בכל צומת פעם אחת.

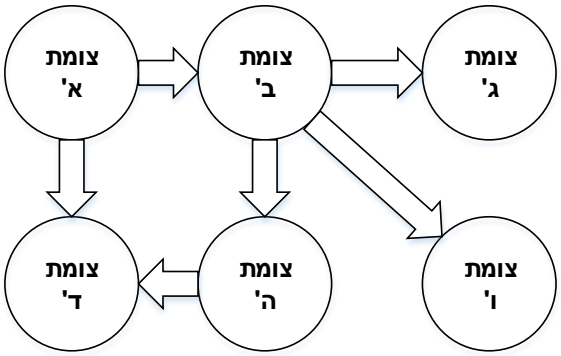
שיטת הסריקה הממומשת להלן היא סריקה לרוחב- לפי שכבות (Breadth-First Search).

בשיטה זו תחל הסריקה מצומת מקור כלשהיא, וצמתי הגרף יתחלקו לשכבות באופן הבא:

בשכבה 1 יהיו הצמתים שניתן להגיע אליהם מצומת המקור דרך מסלול שאורכו 1,
בשכבה 2 יהיו הצמתים שניתן להגיע אליהם מצומת המקור דרך מסלול שאורכו 2, ואינם נמצאים בשכבה הקודמת.

וכן הלאה, עד שכל צמתי הגרף כלולים בשכבה כלשהיא.

לדוגמא:



אם נבחר בצומת א' לצומת ממנה תחל הסריקה, אזי סדר הסריקה יהיה:

בשכבה ראשונה: צומת א'

בשכבה שניה: צומת ב', צומת ד'

בשכבה שלישית: צומת ג', צומת ו', צומת ה'

לא תהיה שכבה רביעית עבור צומת ד' מאחר והיא נסרקה כבר בשכבה שניה.

באלגוריתם נבקר בכל צמתי השכבה הראשונה קודם שניגש לצמתי השכבה השניה.

האלגוריתם:

במהלך האלגוריתם נשתמש במבנה הנתונים תור. התור מממש את גישת First in- First Out, כלומר: האיבר שנכנס ראשון לתור, הוא שיצא ממנו ראשון.

שלב ראשון:

עבור רשימת הצמתים שבגרף, נגדיר:

- רשימת צמתים שתכיל עבור כל צומת את הצומת ההורה שלו בסריקה, בהתאמה.
- מערך עזר שיכיל עבור כל צומת, בהתאמה, ערך בוליאני המסמן האם היא נסרקה או שטרם נסרקה.
- מערך עזר שיכיל עבור כל צומת, בהתאמה, את המרחק מצומת המקור עד אליו (שכבת הצומת).

הנחות יסוד:

1. צומת שכן לצומת X הוא צומת שניתן לגשת אליו מהצומת X דרך מסלול באורך 1.
2. המרחק של צומת X הוא אורך המסלול מצומת המקור עד לצומת X. כלומר, שכבת הצומת X.

שלב שני:

עבור צומת המקור נקבע:

- צומת ההורה של צומת המקור הוא NULL.
- המרחק (שכבה) של צומת המקור הוא תמיד אפס.
- סמן כי הצומת נסרקה.

נכניס את צומת המקור לתור.

שלב שלישי:

שלב זה הינו איטרטיבי. כלומר, מתבצע כל עוד התור לא ריק.

צעד 1: נכנה את הצומת הנמצאת בראש התור בשם V.

צעד 2: עבור כל צומת W שכן של V:

אם w טרם נסרק:

- סמן כי w נסרק.
- קבע את המרחק של w למרחק של v +1.
- קבע את v להורה של w.
- הכנס את w לתור.

הוצא את הצומת v מהתור.

שלב רביעי:

כאשר התור ריק, נסרקו כל צמתי הגרף.

מציאת מסלול הסריקה מצומת המקור לצומת x תעשה באמצעות רשימת הצמתים ההורים:

צומת ההורה של הצומת x תהיה הצומת האחרונה במסלול, וצומת ההורה שלה- תהיה הצומת הקודמת לה במסלול, וכן הלאה עד שמגיעים לצומת המקור, לה אין צומת הורה.

מימוש האלגוריתם:

```
                                                הפונקציה עוברת על כל צמתי הגרף לפי שכבות
public Node<TNode, TArc>[] BFS_LayerSearch(Node<TNode, TArc> source)
{
                                                צעד ראשון-הגדרת משתנים
                                        מערך  השומר עבור כל צומת בגרף האם נסרקה
      Boolean[] used;
                                                מערך המייצג את הצמתים ההורים
      Node<TNode, TArc>[] parent;
      int[] distance;
      Queue<Node<TNode, TArc>> q = new Queue<Node<TNode, TArc>>();
                                                צעד שני-איתחול משתנים
      used = new Boolean[AmountNodes];
      parent = new Node<TNode, TArc>[AmountNodes];
      distance = new int[AmountNodes];

      for (int i = 1; i < AmountNodes; i++)
      {
            distance[i] = int.MaxValue;
      }
      q.Enqueue(source);
      used[this[source]] = true;
      distance[this[source]] = 0;
      while (q.Count > 0)
      {
            List<Node<TNode, TArc>> neighborsNodes =
            q.First().Exits();
                                                עבור  על  כל  הקשתות שיוצאות מהצומת
```

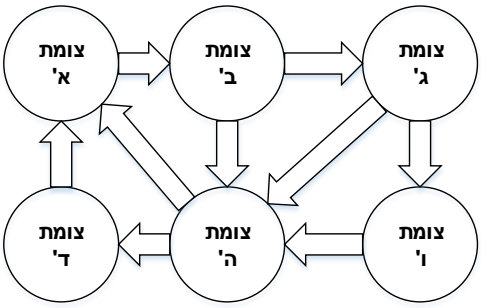
```
            foreach (var item in neighborsNodes)
            {
                מציאת האינדקס של הצומת הבאה אחרי הצומת שבראש התור
                int index = this[item];
                                אם עוד לא סרקנו אותה
                if (used[index] == false)
                {
                                    סמן שנסרקה
                    used[index] = true;
                    המרחק שלה גדול ב 1מהמרחק של הצומת שבראש התור
                    distance[index] = distance[this[q.First()]]+
                    1;
                   קביעת הצומת שבראש התור כצומת האבא
                    parent[index] = q.First();
                    הכנס לתור את הצומת שנסרקה
                    q.Enqueue(item);
                }
            }
            q.Dequeue();
        }
        return parent;
    }
```

3. אלגוריתם למציאת מעגלים (CWS):

אלגוריתם זה משמש למציאת מסלולים מעגליים בגרף. כאמור, מסלול מעגלי הוא מסלול בו ניתן לחזור לנקודת המוצא.

האלגוריתם מוצא את כל המסלולים המעגליים הקיימים בגרף שמתחילים בצומת מסוימת.

לדוגמא:



המסלולים המעגליים שנמצאים בגרף זה, כאשר נקודת המוצא היא צומת א':

1. א' >ב' >ה'
2. א' >ב' >ה' >ד'
3. א' >ב' >ג' >ה'
4. א' >ב' >ג' >ה' >ד'
5. א' >ב' >ג' >ו' >ה' >ד'
6. א' >ב' >ג' >ו' >ה'

האלגוריתם:

מימוש האלגוריתם נעשה בעזרת פונקציה רקורסיבית המתבססת על שיטת ה Back Trackingאשר בונה מסלולים ומוסיפה כל מסלול מעגלי לרשימת המסלולים המעגליים אשר מאוחזרת מהפונקציה.

הפונקציה הרקורסיבית המבצעת את עיקר האלגוריתם נקראת מתוך פונקציה עוטפת, CircleWay. הפונקציה החיצונית מקבלת כפרמטר את הצומת אשר יש למצוא מעגלים המתחילים ומסתיימים בו.

פונקציה זו מגדירה:

- מערך עזר שיכיל עבור כל צומת, בהתאמה, ערך בוליאני המסמן האם היא כלולה במסלול או שאינה כלולה בו.

- משתנה count שיכיל את אורך המעגל, עבור כל קריאה לרקורסיה את אורך המעגל שנשלח לרקורסיה באותה פעם.
- רשימת דרכים, אליה נוספת כל דרך מעגלית חדשה.
- דרך חדשה (שממנה תתפצלנה כל הדרכים).
- צומת שתייצג את הצומת האחרונה במסלול הנוכחי (בקריאה הראשונה צומת זו היא צומת המקור).

הפונקציה הרקורסיבית מקבלת כפרמטרים את:

- צומת המקור (source).
- הצומת האחרונה במסלול הנוכחי (node).
- רשימת הדרכים המעגליות (ways).
- דרך חדשה (way).
- משתנה המייצג את אורך המסלול הנוכחי (count).
- מערך המסמן עבור כל צומת האם היא כלולה במסלול (used).

תנאי העצירה:

אם הצומת הנוכחית (node) זהה לצומת המקור (source) וגם אורך המסלול (count) גדול מאפס- הוסף את הדרך לרשימת הדרכים המעגליות והפסק את בניית המסלול.

אחרת,

אם אורך המסלול (count) שווה למספר הצמתים שבגרף (והמסלול אינו מעגלי)- הפסק את בניית המסלול ללא הוספת הדרך לרשימה.

<u>גוף הפונקציה:</u>

עבור כל צומת v שכן לצומת הנוכחית (node):

אם v לא שותף במסלול:

- סמן כי v משתתף במסלול.
- עדכן את אורך המסלול (count).
- הוסף את v למסלול (way).
- שלח לרקורסיה את הדרך החדשה שנבנתה (שמות הפרמטרים זהים, מאחר וערכיהם שונו).
- סמן כי v לא משתתף במסלול.
- עדכן את אורך המסלול.
- מחק את v מהמסלול.

מימוש האלגוריתם:

```
פונקציה שמחזירה את כל הדרכים המעגליות שמתחילות ומסתיימות בsource
public List<Way<TNode, TArc>> CircleWay(Node<TNode, TArc> source)
{
      List<Way<TNode, TArc>> ways = new List<Way<TNode, TArc>>();
      int[] used = new int[AmountNodes];
      int count = 0;
      CWS(ways, new Way<TNode, TArc>(), source, source, used,
      count);
      return ways;
}
public void CWS(List<Way<TNode, TArc>> ways, Way<TNode, TArc> way,
Node<TNode, TArc> source, Node<TNode, TArc> node, int[] used, int
count)
{
      if (node.Equals(source) && count > 2)
      {
            Way<TNode, TArc> wayToAddToList = way.Copy();
            wayToAddToList.Nodes.Insert(0, source);
            ways.Add(wayToAddToList);
            return;
      }
      if (count == AmountNodes)
      {
            return;
      }
      else
      {
            foreach (var item in node.Exits())
            {
                  if (used[this[item]] == 0)
                  {
                        used[this[item]] = 1;
                        count++;
                        way.AddNode(item);
                        CWS(ways, way, source, item, used, count);
                        used[this[item]] = 0;
                        count--;
                        way.RemoveNode(item);
                  }
            }
      }
}
```

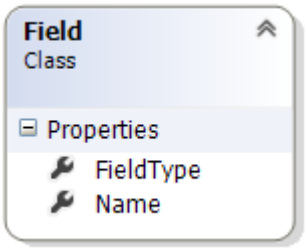
2. הספרייה CodeDom

ספרייה המספקת פונקציות לייצור מחלקה בזמן ריצה.

מאחר וברצוננו לספק מערכת שרות דינאמית ככל האפשר בחרנו לספק למשתמש אפשרות לתאר את הצמתים והקשתות באתר שלו, מטרה זו בצרוף העיקרון לכתוב את הפרויקט כולו בגישה מונחית עצמים הובילה לצורך ליצור מחלקות באופן דינאמי בהתאם למאפיינים שיוכנסו עי מנהל האתר.

<u>הספרייה כוללת:</u>

1. המחלקה Field, המייצגת מאפיין במחלקה הנבנית.



מאפייני המחלקה:

- שם המאפיין
- סוג המאפיין

2. הפונקציות הבאות:

- פונקציה הממירה את מטריצת המאפיינים לרשימה מסוג Field (מורחב בהמשך).
- AddProperties
- AddMethods
- GenerateCSharpCode
- CompileCSharpCode

<u>הספרייה מסתמכת על הפרמטרים הבאים</u>

- שם המחלקה ליצירה.
- מטריצה המכילה את המאפיינים הרצויים באופן הבא: כל שורה במטריצה מכילה צמד פרמטרים עבור מאפיין אחד: שם השדה וסוגו.
- נתיב- המקום בו יש למקם את קובץ הCS שייווצר ואת קובץ הDLL שייווצר.

האלגוריתמים המרכזיים בספרייה CodeDom

פונקציות היצירה השונות, העושות שימוש בפונקציות הבנויות בשפה, מהוות חלק מרכזי מהאלגוריתם.

1. הפונקציה AddProperties

פונקציה זו אחראית על הוספת המאפיינים למחלקה.

המאפיינים מיוצגים באמצעות רשימה של שדות (List<Field>), ועבור כל שדה מהרשימה מוסיפה הפונקציה משתנה פרטי, באופן הבא:

שם המשתנה- נקבע לפי ערך השדה Name.

סוג המשתנה- נקבע לפי ערך השדה FieldType.

```
// הגדרת משתנה עבור כל מאפיין.
for (int i = 0; i < numOfFields; i++)
{
      var field = new CodeMemberField
      {
            Attributes = MemberAttributes.Private,
            Name = char.ToLower(fields[i].Name[0])+
            fields[i].Name.Substring(1,fields[i].Name.Length-1 ),
      };
      //בחירת סוג המאפיין
      switch (fields[i].FieldType)
      {
            case ("int"): field.Type = new
            CodeTypeReference(typeof(System.Int32)); break;
            case ("Double"): field.Type  = new
            CodeTypeReference(typeof(System.Double)); break;
            case ("string"): field.Type = new
            CodeTypeReference(typeof(System.String)); break;
            default:field.Type = new
            CodeTypeReference(typeof(System.Boolean)); break;
      }
      targetClass.Members.Add(field);
```

כאמור, המשתנה שנוסף הוא בהרשאת גישה private, ויש להוסיף עבורו פונקציות גישה- Get & Set.

```
      CodeMemberProperty fieldProperty = new CodeMemberProperty();
```

```
fieldProperty.Attributes = MemberAttributes.Public |
MemberAttributes.Final;
fieldProperty.Name = char.ToUpper(fields[i].Name[0]) +
fields[i].Name.Substring(1, fields[i].Name.Length-1);
//קביעת סוג הערך של פונקציות הגישה- בהתאמה לסוג המאפיין שאליו ניגשים
switch (fields[i].FieldType)
{
      case ("int"): fieldProperty.Type = new
      CodeTypeReference(typeof(System.Int32)); break;
      case ("Double"): fieldProperty.Type = new
      CodeTypeReference(typeof(System.Double)); break;
      case ("string"): fieldProperty.Type = new
      CodeTypeReference(typeof(System.String)); break;
      default: fieldProperty.Type = new
      CodeTypeReference(typeof(System.Boolean)); break;
}
// הוספת הערה שתכתב מעל פונקציות הגישה למאפיין (properties)
fieldProperty.Comments.Add(new CodeCommentStatement("The " +
fields[i].Name + " property for the object."));
//כתיבת גוף פונקציות הגישה
CodeSnippetExpression getSnippet = new
CodeSnippetExpression("return " + field.Name);
CodeSnippetExpression setSnippet = new
CodeSnippetExpression(field.Name + "=value \n");
fieldProperty.GetStatements.Add(getSnippet);
fieldProperty.SetStatements.Add(setSnippet);
// עד פה יצירת properties עבור כל אחד מהשדות
targetClass.Members.Add(fieldProperty);

}
```

2. ***הפונקציה*** *AddMethods*

פונקציה זו אחראית להוסיף למחלקה הנבנית פונקציות הנצרכות לניהול הגרף.

הפונקציות המוספות:

- Constructor
- ToString

- Equals
- GetObjectData

נחיצותן של הפונקציות:

- Constructor- בעת יצירת מחלקה באופן הרגיל נוצר עבורו בנאי ריק כברירת מחדל, אולם כאשר יוצרים מחלקה בעזרת פונקציות הCodeDom יש לכתוב גם בנאי ריק.
- ToString- על מנת שהספרייה Graph (אשר משתמשת במחלקה הנבנית) תוכל להציג את האובייקטים שלה באופן תקין ונוח.
- Equals-על מנת לאפשר את ניהול הספרייה Graph (אשר משתמשת במחלקה הנבנית) באופן תקין, יש לדרוס את הפונקציה Equals הבאה לידי שימוש בחלק מפונקציות המחלקה.
- GetObjectData- נתוני הספרייה Graph נשמרים בקובץ XML. קובץ זה מכיל אף את הנתונים על האובייקטים הנבנים בזמן ריצה. על מנת לאפשר את כתיבת המחלקות הללו, (שאינן ידועות מראש) באופן אוטומטי לקובץ הXML, יש לממש פונקציה זו המוגדרת בממשק ISerializable.

```
public void AddMethod()
{
        // ToString הגדרת הפונקציה
        CodeMemberMethod toStringMethod = new CodeMemberMethod();
        //קביעת הרשאות הגישה לפונקציה
        toStringMethod.Attributes = MemberAttributes.Public |
        MemberAttributes.Override;
        //שם הפונקציה
        toStringMethod.Name = "ToString";
        //סוג הערך המוחזר
        toStringMethod.ReturnType = new
        CodeTypeReference(typeof(System.String));
        CodeFieldReferenceExpression[] list = new
        CodeFieldReferenceExpression[numOfFields];
        //
        for (int i = 0; i < numOfFields; i++)
        {
                CodeFieldReferenceExpression fieldReference =
```

```
        new CodeFieldReferenceExpression( new
        CodeThisReferenceExpression(), fields[i].Name);
        list[i]=fieldReference;
}
CodeFieldReferenceExpression[] arr = list;

// Declaring a return statement for method ToString.
CodeMethodReturnStatement ToStringreturnStatement = new
CodeMethodReturnStatement();
// This statement returns a string representation of the
fields.
string formattedOutput = "The object:";
string s="";
//בניית המשפט המתאר שיוחזר מהפונקציה
for (int i = 0; i < numOfFields; i++)
{
        formattedOutput += "{" + i + "}";
        s+=","+ fields[i].Name;
}
formattedOutput += s;

ToStringreturnStatement.Expression =
new CodeMethodInvokeExpression(
new CodeTypeReferenceExpression("System.String"), "Format",
new CodePrimitiveExpression(formattedOutput),list[0]);
ToStringreturnStatement.Expression.UserData.Add(list[0],
list[1]);
toStringMethod.Statements.Add(ToStringreturnStatement);
 //הוספת הפונקציה למחלקה
targetClass.Members.Add(toStringMethod);

// Equals הגדרת הפונקציה
CodeMemberMethod equalsMethod = new CodeMemberMethod();
//הרשאות גישה
equalsMethod.Attributes = MemberAttributes.Public |
MemberAttributes.Override;
```

```
//שם הפונקציה
equalsMethod.Name = "Equals";
//קביעת סוג הערך המוחזר מהפונקציה
equalsMethod.ReturnType = new
CodeTypeReference(typeof(System.Boolean));
//קביעת סוג הפרמטר המתקבל בפונקציה ושמו
equalsMethod.Parameters.Add(new
CodeParameterDeclarationExpression(new
CodeTypeReference("Object"),"obj"));
CodeMethodReturnStatement EqualsreturnStatement = new
CodeMethodReturnStatement();
string EqualsformattedOutput;
//כתיבת גוף הפונקציה
EqualsformattedOutput=""+fields[0].Name+".Equals("+equalsMetho
d.Parameters[0]+")";
EqualsreturnStatement.Expression=
new CodeMethodInvokeExpression(
new CodeTypeReferenceExpression("System.Boolean"), "Parse",
new CodePrimitiveExpression(EqualsformattedOutput));
equalsMethod.Statements.Add(EqualsreturnStatement);
targetClass.Members.Add(equalsMethod);

//הגדרת בנאי
CodeConstructor constructor = new CodeConstructor();
//הרשאות הגישה
constructor.Attributes = MemberAttributes.Public |
MemberAttributes.Final;
targetClass.Members.Add(constructor);
// GetObjectData הגדרת הפונקציה
CodeMemberMethod getObjectDataMethod = new CodeMemberMethod();
//הרשאות הגישה
getObjectDataMethod.Attributes = MemberAttributes.Public;
//שם הפונקציה
getObjectDataMethod.Name = "GetObjectData";
//שם המשתנה המתקבל וסוגו
```

```
        getObjectDataMethod.Parameters.Add(new
        CodeParameterDeclarationExpression(new
        CodeTypeReference("SerializationInfo"), "info"));
        getObjectDataMethod.Parameters.Add(new
        CodeParameterDeclarationExpression(new
        CodeTypeReference("StreamingContext"), "context"));
        CodeSnippetExpression getDataStatement = new
        CodeSnippetExpression();
        string getdataformattedOutput="";
        //גוף הפונקציה
        for (int i = 0; i < fields.Count; i++)
        {
              getdataformattedOutput +=
              "info.AddValue(\""+fields[i].Name+"\","+fields[i].Name+"
              ,typeof("+ fields[i].FieldType+")); "+
              Environment.NewLine ;
        }
        getDataStatement.Value= getdataformattedOutput;
        getObjectDataMethod.Statements.Add(getDataStatement);
        targetClass.Members.Add(getObjectDataMethod);


}
```

3. הפונקציה GenerateCSharpCode

בפונקציה זו נוצרת מחלקה הכתובה בשפת CSharp, אשר מכילה את המאפיינים והפונקציות שיוצרו בפונקציות דלעיל.

```
public static string GenerateCSharpCode(CodeCompileUnit
classToCompile)
{
        // Generate the code with the C# code provider.
        CSharpCodeProvider provider = new CSharpCodeProvider();
        // Build the output file name.
        string sourceFile;
        if (provider.FileExtension[0] == '.')
        sourceFile = "CodeDomResult" + provider.FileExtension;
        else
        sourceFile = "CodeDomResult." + provider.FileExtension;
        // Create a TextWriter to a StreamWriter to the output file.
        using (StreamWriter sw = new StreamWriter(sourceFile,false))
        {
```

```
            IndentedTextWriter tw = new IndentedTextWriter(sw, "
            ");
            // Generate source code using the code provider.
            provider.GenerateCodeFromCompileUnit(classToCompile, tw,
            new CodeGeneratorOptions());
            // Close the output file.
            tw.Close();
        }
        return sourceFile;
}
```

4. הפונקציה CompileCSharpCode

פונקציה זו מהדרת את התכנית שנוצרה בפונקציה דלעיל, ואם ישנן שגיאות קומפילציה הן נכתבות לחלונית ה Console.

לאחר ההידור הפונקציה יוצרת קובץ DLL המוכן להוספה בכל פרויקט.

```
public static bool CompileCSharpCode(string sourceFile, string
exeFile)
{
        CSharpCodeProvider provider = new CSharpCodeProvider();
        // Build the parameters for source compilation.
        CompilerParameters cp = new CompilerParameters();
        // Add an assembly reference.
        cp.ReferencedAssemblies.Add("System.dll");
        // Generate an executable instead of
        // a class library.
        cp.GenerateExecutable = false;
        // Set the assembly file name to generate.
        cp.OutputAssembly = exeFile;
        // Save the assembly as a physical file.
        cp.GenerateInMemory = false;
        // Invoke compilation.
        CompilerResults cr = provider.CompileAssemblyFromFile(cp,
        sourceFile);
        // Return the results of compilation.
        if (cr.Errors.Count > 0)
        {
        // Display compilation errors.
        Console.WriteLine("Errors building {0} into {1}",  sourceFile,
        cr.PathToAssembly);

        foreach (CompilerError ce in cr.Errors)
        {
        Console.WriteLine("  {0}", ce.ToString());
        Console.WriteLine();
        }
        return false;
        }
        else
        {
        Console.WriteLine("Source {0} built into {1} successfully.",
```

```
        sourceFile, cr.PathToAssembly);
        return true;
        }
        }
    }
```

אנו כמובן נוסיף את קובץ ה DLL למיקום בו תוכל הספרייה Graph להכיר את המחלקה שנוצרה, וניתן יהיה ליצור גרף מסוגה.

<u>דוגמא למחלקה שנוצרה באופן זה:</u>

מאפייני המחלקה (עבור כלוב בגן חיות):

- animalName– מסוג String
- id– מסוג int
- eatingTime– מסוג double

המחלקה הנוצרת תראה כך:

```
//------------------------------------------------------------------------------
-
// <auto-generated>
//     This code was generated by a tool.
//     Runtime Version:4.0.30319.18444
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------
-

namespace CodeDOMSample
{
    using System;
    using System.Runtime.Serialization;


    public class CreatedClass : ISerializable
    {

        private string animalName;

        private int id;

        private double eatingTime;

        public CreatedClass()
        {
        }

        // The AnimalName property for the object.
        public string AnimalName
        {
            get
            {
                return animalName;
```

```
                }
                set
                {
                animalName = value
;
            }
        }

        // The Id property for the object.
        public int Id
        {
            get
            {
                return id;
            }
            set
            {
                id = value
;
            }
        }

        // The eatingTime property for the object.
        public double EatingTime
        {
            get
            {
                return eatingTime;
            }
            set
            {
                eatingTime = value
;
            }
        }

        public override string ToString()
        {
            return string.Format("The
object:{0}{1}{2},AnimalName,Id,eatingTime", this.AnimalName);
        }

        public override bool Equals(Object obj)
        {
            return
bool.Parse("AnimalName.Equals(System.CodeDom.CodeParameterDeclarationExpression
.AnimalName )&" +

"&Id.Equals(System.CodeDom.CodeParameterDeclarationExpression.Id)");
        }

        public virtual void GetObjectData(SerializationInfo info,
StreamingContext context)
        {
            info.AddValue("AnimalName", AnimalName, typeof(string));
            info.AddValue("Id", Id, typeof(int));
            info.AddValue("eatingTime", eatingTime, typeof(Double));
        }
    }
}
```

שמירת הנתונים

לאחר שמנהל האתר / בונה המפה סיים לבנות את הגרף ולבחור את פונקציות החיפוש המתאימות לאתר שלו, ביכולתו לשמור את המפה שעיצב, על מנת שיוכל לאפשר למבקרים באתר לבצע חיפושים כרצונם.

נתוני הגרף נשמרים אצל הלקוח על מנת לחסוך בעומס על הרשת (עבור כל חיפוש קטן יש צורך בכל נתוני הגרף) ועל מנת לאפשר עצמאות לכל בונה מפה.

כמובן, על מנת לאפשר את הפעלת פונקציות החיפוש הלקוח יקבל את הספרייה Graph בקובץ.DLL

לצורך כך, יש צורך בשני קבצים:

- קובץ HTML המכיל את ציור המפה ואת רשימת הפונקציות המאופשרות.
- קובץ XML המכיל את נתוני הגרף.

שני קבצים אלו מתקבלים אצל בונה המפה כאשר הוא מסיים את בנייתה, וביכולתו להוסיף קישור לדף ה HTML מהאתר שלו.

מאחר והנתונים נשמרים אצל הלקוח, העדפנו לשמור אותם בקובץ XML המתאים יותר להתניידות וכן ניתן לקריאה מכל מחשב, בשונה ממסד נתונים שרק מחשב המכיר את סוג מסד הנתונים יכול לקרוא ממנו.

מבנה קובץ שמירת הנתונים- XML:

המבנה הראשי הוא **Graph**, המכיל בתוכו:

- **Nodes** – מכיל רשימה של **Node**
- **Arcs** –מכיל רשימה של **Arc**

Node – מכיל את כל המידע על צומת בגרף:

- **NodeObject** – מתאר מאפיין מסוג המחלקה הגנרית. אינו ידוע מראש, ולכן יירשם כאלמנט אחד, אשר יכול להכיל בתוכו אלמנטים כמספר המאפיינים שיהיו במחלקה הגנרית TNode.
- **Arcs**– רשימת הקשתות היוצאות מצומת זו (עבור כל קשת, יירשם רק המאפיין ArcObject של הקשת, מאחר והוא מזהה אותה).

Arc – מכיל את כל המידע על קשת בגרף:

- **ArcObject** – מתאר מאפיין מסוג המחלקה הגנרית. אינו ידוע מראש, ולכן יירשם כאלמנט אחד, אשר יכול להכיל בתוכו אלמנטים כמספר המאפיינים שיהיו במחלקה הגנרית TArc.
- **Source** – צומת המקור של הקשת (יירשם רק המאפיין NodeObject של הצומת, מאחר והוא מזהה אותה).
- **Target**– צומת היעד של הקשת (יירשם רק המאפיין NodeObject של הצומת, מאחר והוא מזהה אותה).
- **Weight**– משקל הקשת. כלומר, עלות המעבר בה.
- **Mode**– מצב הקשת. כלומר, האם היא מכוונת/ לא מכוונת.

סכמת שמירת הנתונים ב – XML:

```
<Graph>   אלמנט הגרף
  <Nodes>  רשימת צמתי הגרף
    <Node>צומת בגרף
      <ArcObject>מתאר הצומת
      </ArcObject>
      <Arcs> רשימת קשתות הצומת
        <Arc>קשת לצומת
          <ArcObject>מתאר הקשת
          </ArcObject>
        </Arc>
      </Arcs>
    </Node>
  </Nodes>
  <Arcs>      רשימת קשתות הגרף
    <Arc>קשת בגרף
      <ArcObject>מתאר הקשת
      </ArcObject>
      <Source>צומת המקור
        <NodeObject>מתאר הצומת
        </NodeObject>
      </Source>
      <Target>צומת היעד
        <NodeObject>מתאר הצומת
        </NodeObject>
      </Target>
      <Mode></Mode>מצב הקשת – מכוונת/ לא מכוונת
      <Weight></Weight>משקל הקשת- עלות המעבר בה
    </Arc>
  </Arcs>
</Graph>
```

אם נגדיר כי ArcObject **(מסוג** TArc**) מכיל את המאפיינים הבאים:**ID, Name, Max_Speed

(עבור כביש, לדוגמא), אזי האלמנט `<ArcObject>` **יראה כך:**

```
<ArcObject>
     <Id></Id>
     <Name></Name>
     <Max_Speed></Max_Speed>
</ArcObject>
```

ואם נגדיר כי NodeObject **(מסוג** TNode**) מכיל את המאפיינים הבאים:**

Name, crowdedness **(עבור עיר, לדוגמה), אזי האלמנט** `<NodeObject>` **יראה כך:**

```
<NodeObject>
       <Name></Name>
       <crowdedness ></crowdedness>
 </NodeObject>
```

ממשקי המערכת

בפני הנכנס לאתר עומדות מספר אפשרויות:

- טעינת מפה קיימת (ממשק לקוח).
- אפיון אתר + יצירת מפה מתאימה (ממשק מנהל).
- קבלת עזרה ותדרוך לגבי השימוש באתר.

למי מתאימה האפשרות של אפיון אתר ויצירת מפה?

מנהל אתר המעוניין לאפשר למבקרים להתמצא בשטח בקלות ובנוחות. לשם כך עליו ליצור מפה המתארת את האתר שלו, וכוללת את כל המידע שברצונו לספק למבקרים.

אל המפה שיוצר המנהל ניתן יהיה לגשת מהאתר הקיים, בעזרת קישור מתאים אשר יכין מנהל האתר.

למי מתאימה האפשרות של טעינת מפה קיימת?

ישנם מקומות שאינם מחזיקים באתר, מכיוון שאין להם די צורך בו או בשל העלויות הכרוכות באחזקת אתר. גם מקומות אלו יוכלו לאפשר התמצאות קלה ונוחה למבקרים בהם על ידי שימוש באופציה זו של טעינת מפה קיימת: המבקר יקבל את הקבצים הדרושים לעריכת חיפושים ויטען אותם לאתר בדף המתאים.

כאמור, ממשקי המערכת מתחלקים לשני חלקים:

- ממשק מנהל
- ממשק לקוח

ממשק המנהל:

שלב ראשון: אפיון האתר

בשלב זה המנהל קובע מהו המידע שברצונו לכלול עבור תחנה ועבור דרך במפת החיפוש שתוצג למבקרים. על המנהל לספק את כל הפרטים החשובים למבקרים, למטרות התמצאות ולמתן מידע כללי.

הממשק מאפשר:

- עבור כל מאפיין קביעת שמו וסוג הנתונים שהוא יכיל (Type).

- הוספת מאפיין (עבור תחנה/ דרך).
- קביעת מאפייני מפתח - מאפיין אחד/ שניים אשר יבדילו בין תחנה לתחנה או בין דרך לדרך.

שלב שני: מיפוי האתר

בשלב זה המנהל בונה את המפה ומתכננה. בעזרת כלי הציור מסמן המנהל את הצמתים (התחנות) והקשתות (הדרכים) שאותם הוא רוצה לכלול במפה. עבור כל תחנה שמצייר המנהל מופעלת פונקציה להוספת צומת לגרף מתוך הספרייה Graph, ועבור כל דרך שמוסיף המנהל מופעלת פונקציה המוסיפה קשת לגרף, אף היא מהספרייה Graph.

<u>הממשק מאפשר:</u>

- בחירת תמונה שתהווה רקע למפה (תמונת האתר).
- ציור תחנות ודרכים (הוספת צמתים ותחנות לגרף).
- קביעת המאפיינים הפרטיים של כל תחנה/ דרך בחלונית המאפיינים.

שלב שלישי: בחירת פונקציות החיפוש

פונקציות החיפוש הרלוונטיות שונות מאתר לאתר, ועל המנהל לבחור אילו מפונקציות החיפוש שמציעה הספרייה Graph יכולות לבוא לידי שימוש אצל המבקרים באתר שלו. רק פונקציות החיפוש בהן יבחר המנהל תהיינה מאופשרות למבקר ומוצגות לעיניו.

<u>הממשק מאפשר:</u>

- הצגת פונקציות החיפוש שמציעה הספרייה Graph.
- בחירת הפונקציות המתאימות.

שלב רביעי: בחירת מאפיין המשקל

על המנהל לבחור איזה מהמאפיינים שהגדיר עבור הקשת מאפיין את המשקל שלה. כלומר, מכיל את מחיר המעבר בדרך.

ממשק הלקוח:

הלקוח ניגש למפה בנויה ועורך בה חיפושים. ביכולתו לבחור בפונקציית חיפוש מתוך הרשימה המוצגת (רשימת הפונקציות שבחר המנהל), ובמקרה הצורך לקבוע את הפרמטרים המתאימים לבקשת החיפוש שלו. ישנה אפשרות לחפש מסלול העומד במספר קריטריונים. כלומר, להפעיל יותר מפונקציית חיפוש אחת, ולקבל בסופו של דבר את הדרכים העומדות בכל הקריטריונים.

במקרה ונמצאה דרך מתאימה, היא תסומן על גבי המפה.

<u>הממשק מאפשר:</u>

- בחירת פונקציה/ות חיפוש רצויה/ות.
- הצגת המסלול/ים העומדים בהגבלות המשתמש על גבי המפה.

מדריך למשתמש

מדריך למנהל

באם הנך מנהל של אתר / מרכז מבקרים וברצונך לעזור ללקוחותיך להתמצא בשטח, כאן תוכל ליצור עבורם מפת חיפושים.

לשם כך, עליך ליצור מפה המתארת את מרכז המבקרים שלך. ככל שתיאור המקום יהיה מדויק יותר ומותאם למציאות, כך יקל על המבקרים למצוא את מבוקשם.

<u>צעד 1</u>

תחילה, עליך להחליט מהו המידע בו אתה מעוניין לשתף את מבקרי האתר שלך. חשוב על כל המידע הנחוץ לצרכי התמצאות ואף על מידע נוסף שיכול להעניק חוויית ביקור נעימה יותר במקום.

את המידע על האתר עליך לחלק לשניים: מידע על תחנה ומידע על דרך.

מהי תחנה?

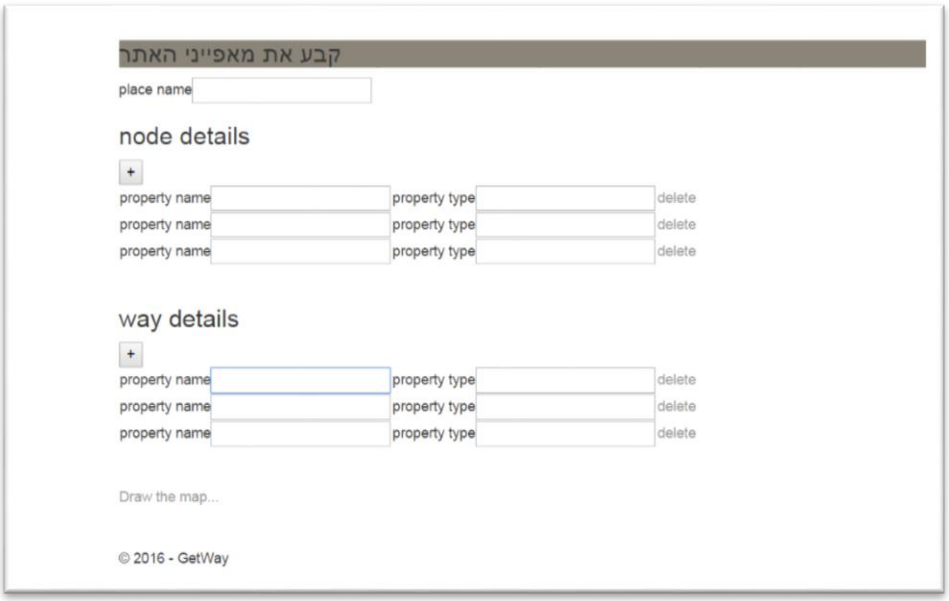
כל יעד לביקור נקרא תחנה. לדוגמה: עיר, חדר במוזיאון, תחנה ביריד, חנות בקניון וכדומה.

מהי דרך?

כל צורת גישה לתחנה נקראת דרך. לדוגמא: כביש, מסדרון, שביל גישה וכדומה.

כעת, עבור כל מאפיין שהחלטת לכלול במפה, יש לחשוב מה יהיה סוג המידע שהוא יכיל. לדוגמא: מספר שלם, מספר עשרוני, כן/לא, טקסט, תאריך/שעה ועוד

את כל זה צריך להכניס לטבלאות הבאות:



ניתן להוסיף שדות כפי הצורך בעזרת הכפתורים המופיעים בראש כל טבלה, ולמחוק שדה בלחיצה על delete.

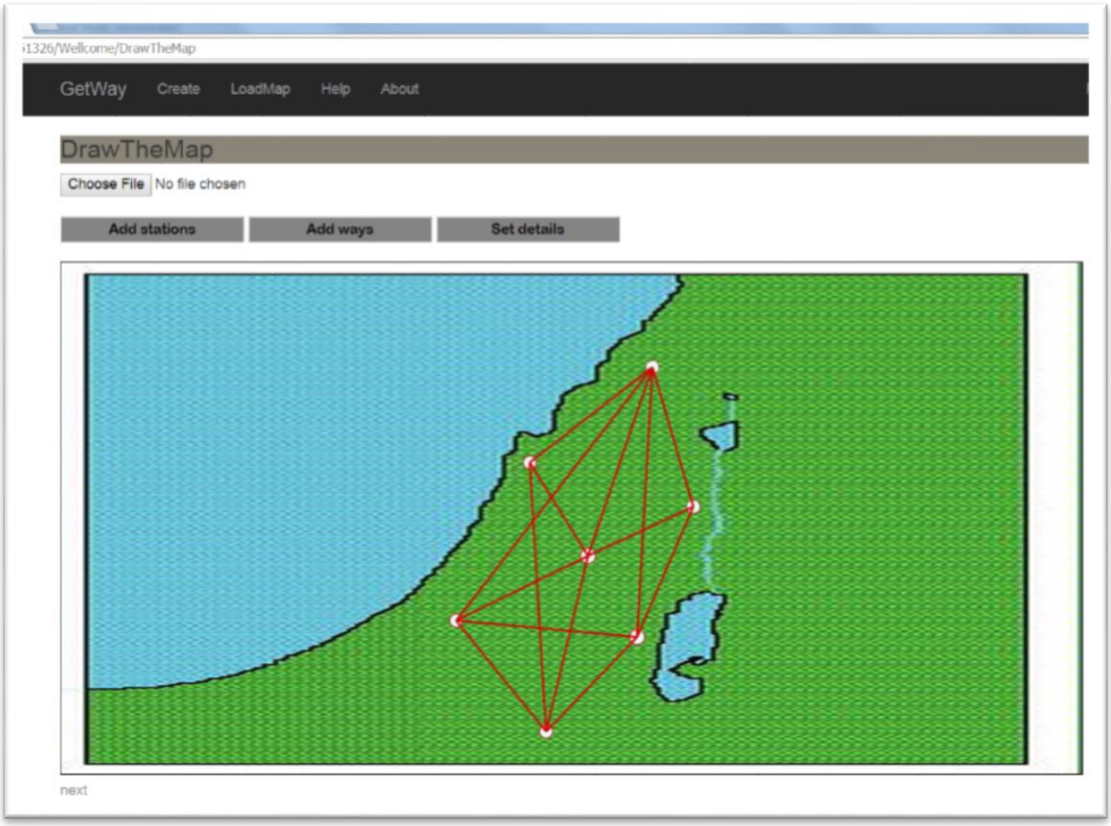
אם אינך מעוניין להגדיר מאפיינים עבור תחנה/ דרך, תוכל להשאיר את הטבלה ריקה.

כעת עליך להגדיר מהו מאפיין המשקל של דרך במפה שלך. לדוגמא: אורך במטרים, זמן הליכה, עלות מעבר, מחיר דלק וכדומה).

כדי להמשיך, יש ללחוץ על *Draw The Map* .

צעד 2

כעת, ניתן לגשת לציור המפה עצמה.



ניתן לבחור תמונת רקע למפה על ידי לחיצה על Choose a background.

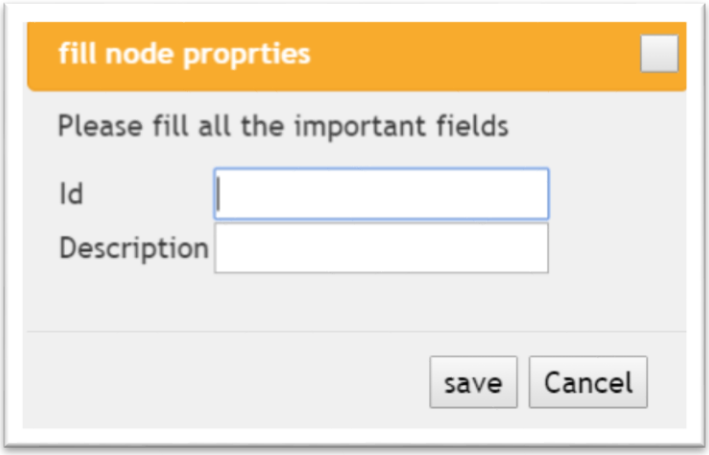
ציור המפה:

- ציור תחנות:

לחץ על הכפתור Add stations המופיע בראש המסך.

לאחר שבחרת באפשרות זו, לחיצה כפולה על המרחב לציור (המלבן הממוסגר) תפתח חלונית בו עליך לקבוע את פרטי התחנה, כפי שהגדרת במסך הקודם.

ליצירת התחנה- לחץ SAVE, לביטול-CANCEL.

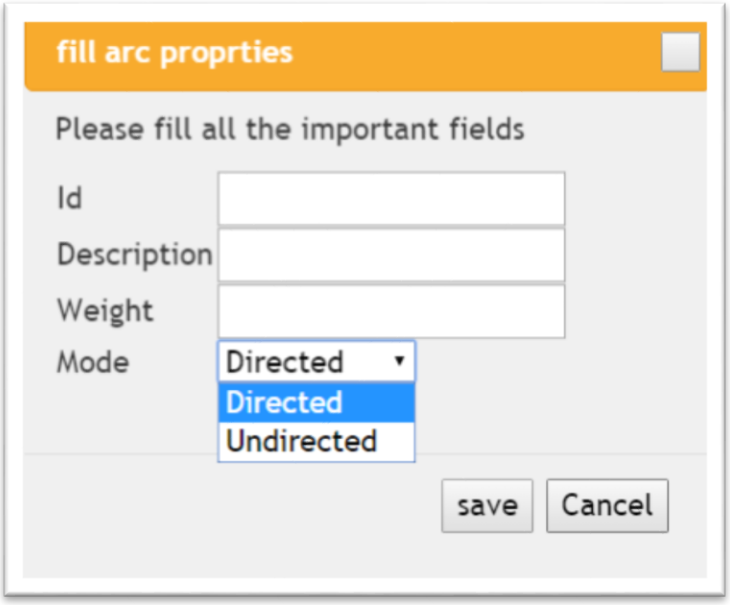


אם לחצת SAVE, יופיע במקום בו לחצת ציור של תחנה. באפשרותך להגדיל/ להקטין את התחנה בהתאם לגודל המפה, כרצונך.

- <u>ציור דרכים</u>:

לחץ על הכפתור **Add ways** המופיע בראש המסך.

לאחר שבחרת באפשרות זו, לחיצה כפולה על תחנה קיימת תוסיף דרך היוצאת מתחנה זו. תפתח חלונית בו עליך לקבוע את פרטי הדרך כפי שהגדרת במסך הקודם, וכן לסמן האם הדרך היא חד כיוונית/ דו כיוונית. ליצירת הדרך- לחץ SAVE, **לביטול**-CANCEL.



כעת, לחיצה על תחנה קיימת תיצור את הדרך מתחנת המקור לתחנה זו.

- <u>שינוי פרטי תחנות/ דרכים:</u>

לחץ על הכפתור Set details המופיע בראש המסך.

לאחר שבחרת באפשרות זו, לחיצה כפולה על תחנה/ דרך תפתח חלונית ובה מופיעים הפרטים שהגדרת עבור התחנה/ הדרך. באפשרותך לשנות ולעדכן את הפרטים לפי הצורך. לשמירת השינויים- לחץ על SAVE, לביטול- לחץ על CANCEL.

- <u>מחיקת תחנה/דרך:</u>

לחץ לחיצה ימנית על התחנה/ הדרך שברצונך למחוק. תפתח חלונית בה עליך לבחור yes אם ברצונך למחוק, ו-cancel אחרת.

בעת מחיקת תחנה יימחקו כל הדרכים היוצאות ממנה והנכנסות אליה.

ניתן להזיז ולסדר את התחנות על גבי המפה, על מנת להתאימה למציאות במידה המקסימלית, לאפשר התמצאות נוחה וקלה.

הערות:

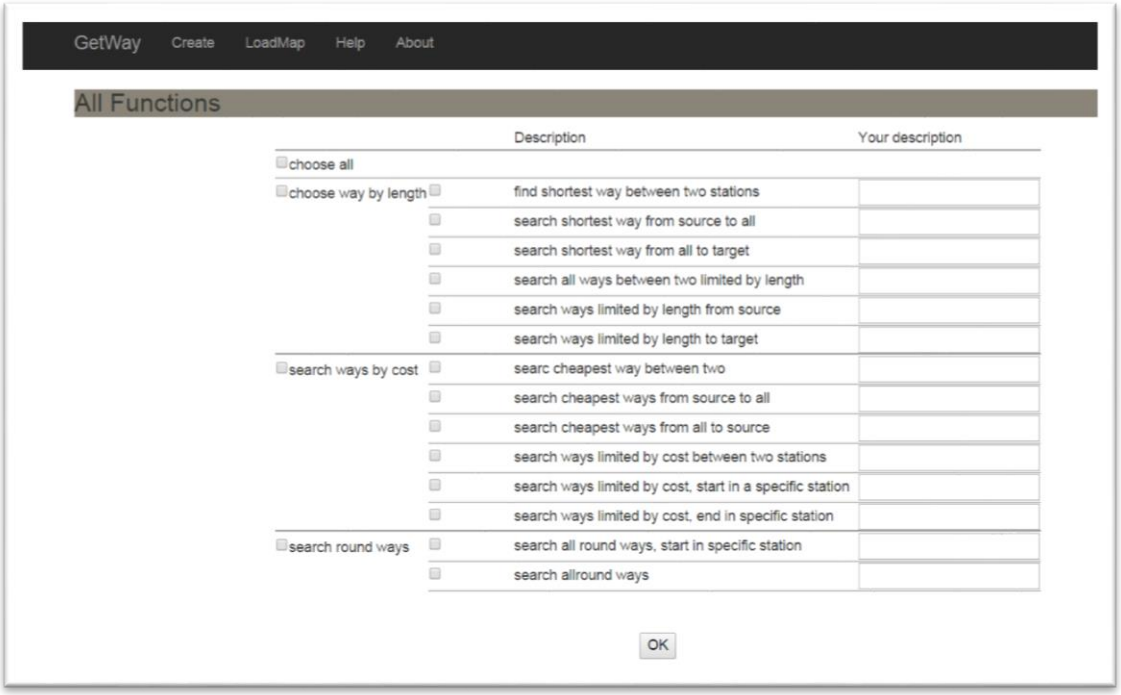
- דרך מחברת בין 2 תחנות בלבד.
- בסיום הציור, על כל התחנות להיות מחוברות ביניהן בדרכים. כלומר, לא תהיינה מספר קבוצות של תחנות.
- כדי להמשיך, לחץ על *next*

<u>צעד 3</u>

השלב הבא הוא בחירת החיפושים המתאימים לאתר שלך. בדף מופיעה רשימת אפשרויות החיפוש המוצעות, ועליך לבחור אלו מהן ברצונך לאפשר למבקרים באתר שלך.

לא לכל אתר מתאימים אותם חיפושים. לדוגמא: אם באתר שלך עלות המעבר בכל הדרכים זהה, לא סביר שהמבקרים בו ירצו לערוך חיפוש לפי עלות כוללת של המסלול, אלא לפי אורך כולל שלו. אם כך, עליך לבחור בחיפושים לפי אורך ולא בחיפושים לפי עלות. כמובן שתוכל לבחור בכל האפשרויות המוצעות.

כל חיפוש בו בחרת עליך לתאר באופן שיהיה ברור ונהיר עבור המבקרים באתר שלך, בעמודה your description. כאשר הם יגשו לבצע חיפוש, יופיעו לפניהם תאורי החיפושים כפי שאתה תכתוב.



המבקרים באתר שלך יוכלו לבצע על המפה שיצרת רק את החיפושים בהם תבחר.

לסיום, לחץ על OK.

סיימת את תהליך בניית המפה, כעת יועברו למחשבך מספר קבצים המתארים אותה.

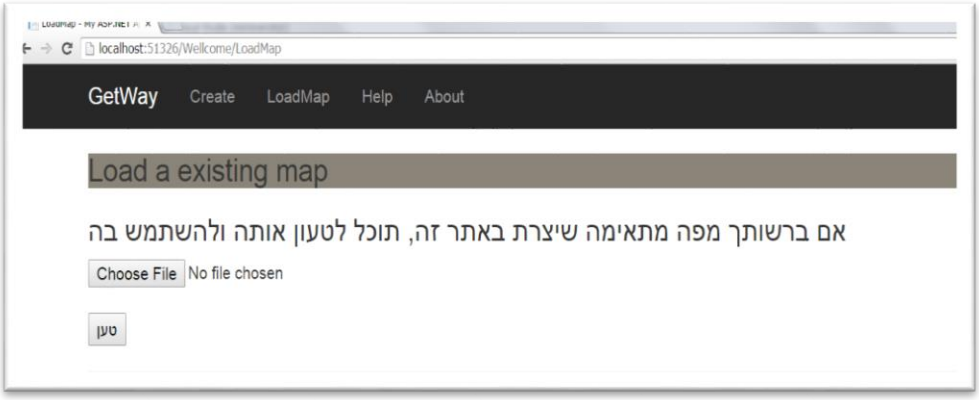
לאחר שיצרת מפה מושלמת, נותר רק לשמור אותה. בלחיצה על הכפתור Save ישמרו עבורך מספר קבצים המכילים את כל המידע להם אתה זקוק על מנת לאפשר ללקוחותיך חוויית התמצאות נוחה ויעילה.

מדריך למבקר

דרך אתר זה תוכל לערוך חיפושים על מפות בנויות.

- אם אתה ניגש למפה דרך אתר של מרכז מבקרים מסוים, תופיע לפניך מפת האתר ובה כל התחנות הנמצאות במקום ודרכי הגישה אליהן.
- אם אתה ניגש לאתר באופן ישיר, וודא כי ברשותך מפה המוכנה לטעינה וכן את הקובץ הנוסף הנדרש על מנת לאפשר לך לבצע חיפושים במפה.

עליך לגשת לדף המתאים *Load a existing map* ולטעון את המפה על ידי לחיצה על *Choose a file*.



באפשרותך לחפש מסלול בעזרת פונקציות החיפוש המוצעות. אלו חיפושים התואמים את האתר בו אתה מבקר, ואשר נבחרו על ידי מנהל האתר.

הנך יכול לבחור במסנן אחד או בכמה מסננים (חיפושים) כדי למצוא את הדרך המתאימה ביותר עבורך. סמן את כל פונקציות החיפוש שברצונך להפעיל, וקבע את הפרמטרים המתאימים לך, כפי הצורך.

לדוגמא: תוכל לבחור בחיפוש מורכב הכולל חיפוש מסלול מתחנת מקור מסוימת לתחנת יעד מסוימת, וחיפוש מסלול באורך 3 לכל היותר. אם קיימים מסלולים באורך 3 לכל היותר מתחנת המקור לתחנת היעד שבחרת, הם יסומנו על המפה.

דוגמא לפרמטרים שתידרש לקבוע: אורך מקסימום למסלול, תחנת התחלה או סיום וכדומה. קבע את הפרמטרים במקומות המתאימים. לבחירת צומת לחץ עליה (פעל בהתאם להוראות).

באם יימצא מסלול העונה על כל הקריטריונים שהצבת, המסלול יסומן על גבי המפה.

כל שנותר לך הוא לתכנן את צעדיך בהתאם למסלול שקיבלת!

מסקנות

הדבר החשוב ביותר שלמדנו הוא שכאשר ניגשים לעבוד על פרויקט בסדר גודל כזה חשוב מאוד להגדיר באופן מדויק ככל האפשר את המטרה ולהציב יעדים מוגדרים.

תכנון טוב ויסודי הוא הבסיס לפרויקט מוצלח, ומביא לפיתוח זורם ולהתקדמות ברורה.

במהלך הפיתוח נתקלנו במספר קשיים אשר היה עלינו להתמודד עמם, ואשר לימדונו רבות:

אתגרים במהלך הפרויקט:

1. אופן ייצוג מבנה הנתונים גרף

על מנת לייצג את ספריית הגרפים באופן היעיל ביותר, סקרנו מספר מבני נתונים אשר באמצעותם ניתן לייצג גרף.

- אחת הדרכים לייצוג גרף היא באמצעות מטריצת סמיכות.

זוהי מטריצה ריבועית בגודל E*E, כאשר E הוא מספר הצמתים בגרף.
המקום ה[j][i] יכיל את משקל הקשת שבין צומת i לצומת j , אם אין קשת כזו יכיל ∞ (אינסוף).

צורה זו נשללה כיוון שהיא מתאימה לגרפים בהם מספר הצמתים הוא סופי וידוע מראש ואילו השימוש המיועד שלנו בספריה הוא עבור גרף שגודלו נקבע בזמן ריצה ויכול להשתנות.

- אפשרות נוספת הייתה לייצג את הגרף באמצעות רשימת סמיכות.

זוהי רשימה באורך E כאשר E הוא מספר הצמתים בגרף. כל מקום ברשימה יכיל רשימה, אף היא באורך E, וכך נוכל לדמות מטריצה ריבועית כבסעיף 1, ולהרוויח את האפשרות שניתן להגדיל את מספר הצמתים על ידי הוספת איבר לרשימה הראשית ולכל אחת מהרשימות שיוצאות ממנה.

אפשרות זו נשללה משום שהיא כרוכה בסרבול רב, ואינה מתאימה באופן מלא לעקרונות ה-OOP.

לבסוף בחרנו לייצג את הגרף בצורה הקרובה ביותר למציאות:

ייצגנו את הגרף על ידי אוסף צמתים ואוסף קשתות, כאשר צומת מכילה את כל המידע אודות הצומת והצבעות לקשתות היוצאות ממנה, וקשת מכילה את כל המידע אודותיה והצבעות לצמתי הקצה שלה.

בצורה זו קל מאד לממש את האלגוריתמים המוגדרים על מבנה הנתונים גרף, וכן ניתן לנהל את הגרף (להוסיף/ למחוק ממנו פריטים) באופן יעיל ונוח, בלי קוד מסובך או פעולות רבות ומיותרות.

מסקירת האפשרויות השונות למדנו רבות על היתרונות והחסרונות במבני הנתונים השונים ונוכחנו לראות כי שיטת ה OOP אכן יעילה ונוחה לשימוש.

2. הצורך לאפשר לבנות מפה מתארת ייחודית

כל מנהל מעוניין למפות את האתר שלו באופן המדויק ביותר על מנת לספק מידע חשוב למבקר. על מנת לאפשר דבר זה יש לתת למנהל להגדיר את המחלקות המתארות את המפה שלו, דבר שכמובן אינו ידוע מראש ומצריך בניית מחלקות אלו בזמן ריצה.

לצורך כך היה עלינו ללמוד אודות הפונקציות והמחלקות הנכללות ב System.CodeDom ולעשות בהם שימוש באופן היעיל והנכון ביותר עבורנו.

החיפוש והלמידה הראו לנו כי ישנה פונקציונליות רבה שכבר כתובה ומוכנה לשימוש, ועלינו רק להשתמש בה על מנת להתקדם ולפתח.

3. ההתייחסות למחלקה שאינה ידועה

אופן שמירת נתוני הגרף הוא באמצעות קובץ XML אשר מחייב להכיר את כל נתוני הגרף.

לא יכולנו להשתמש בפונקציות המאפשרות כתיבה וקריאה אוטומטיות מה XML מאחר ומבנה הנתונים בו בחרנו הינו מעגלי והפונקציות הקיימות לא ידעו להתמודד אתו.

לכן, נאלצנו לכתוב פונקציות אלו בעצמנו.

המחלקות הנבנות בזמן ריצה חייבות להיות מורכבות ממאפיינים פשוטים. כלומר, אין בהן מאפיין שאינו מוכר בשפה. מסיבה זו, כל מחלקה שהיא אשר תבנה בזמן ריצה ניתנת לכתיבה/ קריאה אוטומטית ל XML בעזרת הפונקציות הקיימות.

לכן, חייבנו את הגרף הגנרי לקבל רק מחלקות אשר מממשות את הממשק ISerializable וממילא את הפונקציה GetObjectData המוגדרת בו ומשמשת לכתיבת / קריאת המחלקה, כאמור:

```
public class Graph<TNode, TArc>

where TNode : ISerializable

where TArc : ISerializable
```

כאשר כתבנו /קראנו את מאפייני הגרף באופן ידני, עבור המאפיין שהוא מסוג המחלקה הגנרית הפעלנו את הפונקציה הנ"ל (GetObjectData).

בעת יצירת המחלקה היה עלינו לדאוג להוסיף את מימוש הממשק ISerializable במחלקה הנוצרת, וכן לממש בה את הפונקציה הנדרשת:

```
nameSpaceName.Imports.Add(new
CodeNamespaceImport("System.Runtime.Serialization"));

targetClass.BaseTypes.Add(new CodeTypeReference("ISerializable"));


//declaring GetObjectData method

CodeMemberMethod getObjectDataMethod = new CodeMemberMethod();

getObjectDataMethod.Attributes = MemberAttributes.Public;

getObjectDataMethod.Name = "GetObjectData";

getObjectDataMethod.Parameters.Add(new CodeParameterDeclarationExpression(new
CodeTypeReference("SerializationInfo"), "info"));

getObjectDataMethod.Parameters.Add(new CodeParameterDeclarationExpression(new
CodeTypeReference("StreamingContext"), "context"));

CodeSnippetExpression getDataStatement = new CodeSnippetExpression();

string getdataformattedOutput = "";

for (int i = 0; i < fields.Count; i++)

{

getdataformattedOutput += "info.AddValue(\"" + fields[i].Name + "\"," + fields[i].Name +
",typeof(" + fields[i].FieldType + ")); " + Environment.NewLine;

}

getDataStatement.Value = getdataformattedOutput;

getObjectDataMethod.Statements.Add(getDataStatement);

targetClass.Members.Add(getObjectDataMethod);
```

מאופן הפתרון למדנו כי ניתן להתמודד גם עם סוגי משתנים שאינם ידועים מראש, וכי לכל בעיה ניתן למצוא פתרון.

סיכום

ההשוואה בין התכנון הראשוני למוצר הסופי ,מגלה כי המוצר עומד בדרישות וזהה ברובו לתכנון .הגם שבמהלך הביצוע עלו שאלות רבות לגבי ביצוע טכני ותכנותי בפרויקט . דברים מסוימים שנראו בלתי אפשריים ,התגלו כברי ביצוע על ידי דרכים יצירתיות ומגוונות ובאמצעות שימוש בספריות וחומרים שהתפרסמו בנושא.

במבט לאחור ,הפרויקט היווה עבורנו התנסות בהתמודדות עם עבודה בסדר גודל , מימוש ופיתוח אלגוריתמים מורכבים ושימוש ביישומים וטכנולוגיות מתקדמות.

את הפרויקט ליוו שעות רבות של תכנון ,עמל והשקעה כדי להגיע לרמה תכנותית מקצועית ומדויקת.

ביבליוגרפיה

במהלך כתיבת הפרויקט נעזרנו בין השאר במקורות הבאים:

- Msdn.Microsoft.Com
- www.simonsarris.com
- www.jqueryui.com
- www.w3Schools.com
- ספר 'מבני נתונים ואלגוריתמים' חלק ב', של האוניברסיטה הפתוחה.