<u>תרגיל בית 1: שימוש באלגוריתמי חיפוש היוריסטיים לתכנון מסלולי חלוקה אופטימליים</u>

מטרות התרגיל

- נתמודד עם בעיות פרקטיות ותיאורטיות של חיפוש במרחבי מצבים עצומים.
- נתרגל את הנלמד בהרצאות ובתרגולים.
- נתנסה בתכנות ב- python לפתרון בעיות פרקטיות.

הנחיות כלליות

- **תאריך הגשה:** יום ראשון, 09.12.2020, בשעה 23:59.
- את המטלה יש להגיש **<u>בזוגות בלבד</u>**.
- יש להגיש מטלות מוקלדות בלבד. פתרונות בכתב יד לא ייבדקו.
- ניתן לשלוח שאלות בנוגע לתרגיל לתיבת המייל הקורסית: ai.technion@gmail.com. לפני שליחת שאלה, בדקו האם קיימת לה תשובה כבר בFAQ, שאלות שנענו כבר בFAQ לא יענו שוב במייל.
- המתרגל האחראי על תרגיל זה: אלעד נחמיאס.
- בקשות דחיה **<u>מוצדקות</u>** (מילואים, אשפוז וכו') יש לשלוח למתרגל האחראי (גיא קושלביץ) בלבד.
- במהלך התרגיל ייתכן שנעלה עדכונים, תיקונים והבהרות לדף FAQ ייעודי באתר. העדכונים הינם **<u>מחייבים</u>**, ועליכם להתעדכן דרך עמוד זה.
- שימו לב, התרגיל מהווה כ- 13% מהציון הסופי במקצוע ולכן העתקות תטופלנה בחומרה.
- ציון המטלה יורכב מהגורמים הבאים:
  - **הדו"ח הסופי.** מעבר לתשובות הנכונות, אתם נבחנים גם על הצגת הנתונים והתוצאות בצורה קריאה ומסודרת.
  - **הקוד המוגש.** ראשית, הקוד שלכם ייבדק באופן מקיף ע"י מערכת בדיקות אוטומטיות. המערכת תבדוק את התוצאות שלכם לעומת התוצאות המתקבלות במימוש שלנו. אנו מצפים שתקבלו את אותם הערכים בדיוק. נבדוק את המסלול המתקבל, את עלותו ואת מס' הפיתוחים. לכן עליכם להיצמד להוראות בתרגיל זה. הבדיקות יהיו כמובן מוגבלות בזמן ריצה. ייתנן לכם זמן סביר ביותר להרצת כל טסט. אם תעקבו אחר ההוראות במסמך זה ובקוד אין סיבה שלא תעמדו בזמנים אלו. בנוסף, יש להקפיד על הגשת קוד מסודרת בהתאם להנחיות. יש לכתוב הערות במקומות חשובים בקוד כדי שיהיה קריא וקל לבדיקה ידנית.
- גרסת python איתה אתם נדרשים לעבוד הינה 3.7. גם קבצי המקור שקיבלתם מתאימים לגרסה זו.
- כאמור, הבדיקות האוטומטיות של הקוד שתגישו תהיינה מוגבלות בזמן פר טסט. היו סמוכים ובטוחים שמערכת הבדיקה הינה הוגנת ביותר. מימוש תקין שצמוד להוראות יעמוד במסגרת הזמנים. הסיבה למגבלת הזמן היא פשוטה – לא ניתן להריץ כל טסט אינסוף זמן – אנחנו צריכים לבדוק את כל התרגילים שלכם במסגרת זמן סבירה. בכדי לעמוד במסגרת הזמנים אתם לא מתבקשים לחשוב על אופטימיזציות כאלו או אחרות, אלא רק לעקוב באדיקות אחר ההוראות. הבינו איך משתמשים ב- iterators בפייתון ונסו להשתמש בהם בכל מקום שתוכלו (במקום ליצור רשימות איפה שאין באמת צורך בכך). אנו מכווינים אתכם לעשות כך בחלק מהסעיפים. קשה לפרט דרישת זמנים קשיחה כי לכל אחד יש מחשב בעל מפרט אחר. נפרט כאן הערכה כללית לזמן הריצה הצפוי של מימוש תקין במחשב אישי מודרני סביר, וזאת רק בכדי שתוכלו לקבל סדר גודל ולוודא שאתם לא חורגים מכך באופן דראסטי. אם אתם חורגים מהאמור באופן דרסטי ייתכן שיש לכם טעות במימוש. הריצה הארוכה ביותר (של הקלט הגדול, בסעיף האחרון בתרגיל) אמורה לקחת לכל היותר 4 דקות. בשאר הסעיפים בתרגיל הריצה אמורה להסתיים לכל היותר תוך 2 דקות. בפועל, חלק ניכר מהריצות למעשה מסתיימות תוך פחות מדקה. היעזרו בהערכה גסה זו כדי לוודא/לחשוד בתקינות המימוש שלכם.
- אלא אם נכתב אחרת, אין לשנות פונקציות מוכנות שקיבלתם. בנוסף, אין לשנות את החתימה של פונקציות שהתבקשתם לממש או אחרות. בפרט, אין לשנות תוכן קבצים בהם לא נתבקשתם לבצע שינויים. אין ליצור פונקציות עזר משלכם, אנא השלימו את המימושים אך ורק במקומות המסומנים. בנוסף, אין ליצור קבצים חדשים, אלא לערוך את הקבצים שהתבקשתם במפורש בלבד. ראו הוזהרתם - חריגה מכללים אלו יכולה להוביל לכישלון מידי בבדיקות האוטומטיות. אם יש בעיה נקודתית, ניתן לשלוח מייל לתיבה הקורסית.
- אין להוסיף ו/או לשנות פקודות import בקוד. כל מה שאתם צריכים כבר מיובא במקום הרלוונטי.

- אין לבצע בעצמכם טעינה של קלטים או מפות. אנחנו עשינו זאת עבורכם במקומות הנדרשים. בכל אזור בקוד בו שהתבקשתם להשלים את המימוש יש גישה לכל המבנים להם אתם זקוקים לצורך המימוש. ראו הוזהרתם - חריגה מכללים אלו יכולה להוביל לכישלון מידי בבדיקות האוטומטיות.
- לא יענו שאלות בסגנון: "איך מוצאים את עלות הפתרון שהוחזר?" או "איך ניגשים למפות הכבישים מתוך המימוש של הפונק' ההיא?" או "באיזה שדה שמורה המהירות של הכביש?" או "אילו שדות מצפים לקבל אובייקט מטיפוס frozenset?". בכל מקום בקוד בהם אתם נדרשים להשלים את המימוש (לכתוב קוד כלשהו) השארנו לכם הערות מפורטות שמסבירות כיצד יש לעשות זאת. ברוב המקומות גם הכוונו אתכם במפורש לשמות השדות ולמתודות הרלוונטיות להם תזדקקו. בנוסף, כל הקוד כתוב עם type-annotations (למרות שאין הכרח לציין טיפוסים בפייתון) במטרה להקל עליכם בהתמצאות בקוד ובכדי שתוכלו להבין מה אמור לקבל כל שדה/ארגומנט. אנחנו מצפים מכם להשכיל ולהשתמש ב- IDE (ממליצים על PyCharm) שיוכל לסייע לכם להתמצא בקוד ביתר קלות ויזהה עבורכם שגיאות בצורה סטטית – זה יחסוך לכם הרבה זמן. ה- type-annotations שהוספנו לקוד עוזרות ל- IDEs לעזור לכם – נצלו את זה. בחלק מהמקומות החסרנו חלק מהפרטים בהסבר מתוך כוונה – אנחנו רוצים לעודד אתכם לעיין בקוד ולמצוא פרטים אלו בכוחות עצמכם. הכרת סביבת העבודה שסיפקנו לכם והתמצאות בה הן למעשה חלק מהתרגיל.
- לצורך ההרצות תצטרכו להתקין את החבילות הבאות של python: numpy, scipy, matplotlib, networkx. חלק מחבילות אלו מותקנות כברירת מחדל עם ההתקנה של Anaconda. את אלו שאינן מותקנות אפשר להתקין בעזרת הפקודה `pip install <package name>`.
- בתרגילי הבית בקורס הרצת הניסויים עשויה לקחת זמן רב, ולכן מומלץ מאוד להימנע מדחיית העבודה על התרגיל ו/או כתיבת הדו"ח לרגע האחרון. לא תינתנה דחיות על רקע זה.
- מסמך זה כתוב בלשון זכר מטעמי נוחות בלבד, אך מתייחס לנשים וגברים כאחד.

ייתכן שמסמך זה יתעדכן באתר – הבהרות ועדכונים שנוספים אחרי הפרסום הראשוני יסומנו בצהוב.

חלק א' – מבוא והנחיות

במטלה זו נעסוק בהפעלת אלגוריתמי חיפוש על מרחבי מצבים גדולים במיוחד לבעיות ניווט. מומלץ לחזור על שקפי ההרצאות והתרגולים הרלוונטיים לפני תחילת העבודה על התרגיל.

במהלך התרגיל תתבקשו להריץ מספר ניסויים ולדווח על תוצאותיהם. אתם נדרשים לבצע ניתוח של התוצאות, כפי שיוסבר בהמשך.

מוטיבציה

במקביל ללימודיו בטכניון, שלומי עובד כשליח הובלות. במהלך המשמרת שלו, שלומי מקבל מספר הובלות אותן הוא צריך לאסוף מנקודות איסוף שונות ברחבי העיר ולפזרם בנקודות הורדה. בתחילת המשמרת שלומי מקבל רשימה של x לקוחות (כל לקוח מעוניין לבצע הובלה) ויוצא לדרך במטרה להשלים את כל ההובלות ברשימה.

לכל לקוח, שלומי צריך לעבור בנקודת האיסוף המתאימה ע"מ להעמיס על המשאית שלו את כל החבילות של אותו הלקוח ולאחר מכן להוריד בנקודת ההורדה של אותו הלקוח את אותן החבילות שנאספו. שלומי יכול לבחור באיזה סדר לעבור אצל הלקוחות. בנוסף, שלומי יכול להעמיס על המשאית חבילות ממספר לקוחות ברצף ורק אז לפזרם (בסדר כלשהו), ובלבד שיש לו די מקום פנוי במשאית להעמסת החבילות.

בימים בהם שלומי עמוס בלימודים הוא רוצה לסיים את המשמרת כמה שיותר מהר ולהגיע הביתה כדי לעבוד על ההגשות שלו. בימים בהם אין לשלומי הרבה לימודים הוא מעדיף להרוויח כמה שיותר כסף.

למזלו, חברים של שלומי (זה אתם!) במקרה לוקחים הסמסטר את הקורס "מבוא לבינה מלאכותית". שלומי מבקש מכם לעזור לו לתכנן מראש את הדרך היעילה ביותר לבצע את כל ההובלות.

פורמאליזם – הגדרת הבעיה

נתונה רשת כבישים בצורת גרף $StreetsMap = (V_{map}, E_{map})$ שבו כל צומת מייצג צומת דרכים (junction), והקשתות מייצגות דרך (כביש) המקשרת בין צמתי דרכים (links).

למשאית יש קיבולת מרבית של $TruckCapacity$ חבילות.

נתונה נקודת מוצא על רשת הכבישים $v_0 \in V_{map}$, וכן נתונות $k \in \mathbb{N}$ הובלות שהוזמנו: $D = \{d_1, \dots, d_k\}$, כאשר הובלה i כוללת: נק' העמסה/איסוף $d_i.pick$, נק' הורדה/פריקה $d_i.drop$ ומספר החבילות שיש להעביר $d_i.pkgs \in \{1,2, \dots, TruckCapacity\}$. נקודות ההעמסה וההורדה הן צמתים ברשת הכבישים $d_i.pick, d_i.drop \in V_{map}$.

לצורך פשטות, במהלך כל התרגיל נניח כי כל נקודות העמסה של הובלות, נקודות ההורדה של הובלות ונקודות המוצא הן נקודות זרות במפה. כלומר $\left|\{v_0\} \cup \{d_i.pick\}_{i\in[k]} \cup \{d_i.drop\}_{i\in[k]}\right| \equiv 2k+1$.

<u>*סידור ביקורים אצל לקוחות*</u> הינו פרמוטציה $\pi = w_1, \dots, w_{2k}$ של הנק' $\{d_i.pick\}_{i\in[k]} \cup \{d_i.drop\}_{i\in[k]}$ המקיימת את שני האילוצים הבאים:

1. לכל $i \in [k]$ נק' האיסוף $d_i.pick$ מופיעה בסידור π לפני נק' ההורדה המתאימה $d_i.drop$.
2. נאמר ש- π <u>*אוספת*</u> את משלוח d בביקור i אם $w_i = d.pick$. נאמר ש- π <u>*מורידה*</u> את משלוח d בביקור i אם $w_i = d.drop$. אם π אוספת משלוח d בביקור ה- i, <u>מספר החבילות ש- π אוספת</u> בביקור ה- i הוא $d.pkgs$. אם π אוספת מורידה d בביקור ה- i, <u>מספר החבילות ש- π מורידה</u> בביקור ה- i הוא $d.pkgs$. מספר החבילות על המשאית בביקור ה- $i \in [2k]$ מוגדר להיות סכום מספר החבילות שנאספו בביקורים $\{1, \dots, i\}$ פחות סכום מספר החבילות שנפרקו בביקורים הללו. המסלול מקיים את אילוץ תכולת המשאית: בכל רגע במסלול, מס' החבילות על המשאית לא עולה על הקיבולת המקסימלית של המשאית.

את איכות סידור ביקורים π שיחושב ע"י התוכנית נמדוד לפי מספר מדדים שונים (מרחק, זמן ועלות כספית), כפי שיפורט בהמשך.

הפתרון לבעיה לפי מדד איכות נתון הינו סידור ביקורים אצל לקוחות בעל מחיר מינימלי ע"פ מדד איכות זה.

הבנת קושי הבעיה

בשלב זה אנחנו רוצים לקבל קצת אינטואיציה לגבי הקושי של הבעיה. המטרה היא להשתכנע שאנחנו לא מסוגלים לפתור את הבעיה בעזרת חיפוש brute-force (בגלל מגבלת משאבים). לצורך זאת, ראשית ננסה

להעריך את מספר הסידורים החוקיים השונים אותם יש לבחון במסגרת ריצת brute-force. לשם פשטות החישוב נתעלם ממגבלת קיבולת המשאית (נניח $TruckCapacity = \infty$). נותרנו עם האילוצים הבאים: על הסידור לכלול את כל נק׳ האיסוף ואת כל נק׳ ההורדה וכן לכל הובלה נק׳ האיסוף שלה מופיעה בסידור לפני נק׳ ההורדה שלה.

תרגיל

1. יבש: מצאו ביטוי מתמטי עבור מספר הסידורים החוקיים העונים על האילוצים (המוקלים) שפורטו דלעיל. על הנוסחה להיות תלויה ב- k (מס׳ ההובלות).
2. יבש: מלאו את הטבלה הבאה. הזינו את מספר הפרמוטציות האפשריות (וערכי $\log_2$ שלהן) עבור ערכי k (מספר ההובלות) המופיעים בטבלה. היעזרו בנוסחה שמצאתם בסעיף (1). נניח שמחשב יחיד יכול לבחון 2^{30} סידורים בשנייה. מלאו בעמודה האחרונה כמה זמן ייקח למחשב זה לבדוק כל אחד מהסידורים (לפי היחידות המפורטות).

k	$\#possiblePaths$	$\log_2(\#possiblePaths)$	Calculation time
5			< 1 sec
8			[sec]
9			[hours]
10			[years]
11			[years]
15			[million years]

חלק ב' – הגדרת מרחב החיפוש במפה

כאמור נתונה רשת כבישים בצורת גרף $StreetsMap = (V_{map}, E_{map})$. בעיית המפה עוסקת במציאת מסלול ברשת הכבישים $StreetsMap$ בעל עלות מינימלית (ביחס לפונק' עלות נתונה המוגדרת על כבישים במפה). בחלק זה נייצג את בעיית המפה כמרחב חיפוש. ניצמד להגדרה שלמדנו בכיתה עבור מרחבי חיפוש. אנו מתחילים בבעיית המפה משום שהיא בעיה יחסית פשוטה, הייצוג שלה כמרחב חיפוש הוא אינטואיטיבי ואנו אכן נעשה בה שימוש בחלקים הבאים.

בהינתן רשת הכבישים, נקודת מקור $v_{src} \in V_{map}$ ונקודת יעד $v_{dst} \in V_{map}$, נגדיר מרחב חיפוש עבור מציאת מסלול ביניהן:

$$MapProblem \triangleq (S_{map}, O_{map}, I_{map}, G_{map})$$

- **קבוצת המצבים:**
נרצה לייצג מצב כך שיחזיק את כל המידע שנחוץ לנו עליו במהלך החיפוש במרחב. במקרה המדובר מספיק לשמור את הצומת ברשת הכבישים.
$$S_{map} \triangleq \{(v{:}\,u) \mid u \in V_{map}\}$$
- **קבוצת האופרטורים:**
ניתן לעבור ממצב אחד לעוקבו בתנאי שיש כביש מהצומת המיוצג ע"י המצב הראשון לצומת המיוצג ע"י המצב העוקב.
$$O_{map} \triangleq \{(s_1, s_2) \mid s_1, s_2 \in S_{map} \wedge (s_1.v, s_2.v) \in E_{map}\}$$
- **עלות אופרטור:**
במטלה נגדיר 3 פונק' עלות עבור מעבר מצומת דרכים אחד $s_1 \in S_{map}$ לצומת דרכים עוקב שלו $o \in O_{map}, s_2 = o(s_1)$.
  1. אורך של הכביש ביניהם:
  $$cost_{map}^{dist}((s_1, s_2)) = roadLength((s_1.v, s_2.v))$$
  2. זמן הנסיעה ביניהם:
  $$cost_{map}^{time}((s_1, s_2)) = roadLength((s_1.v, s_2.v)) / roadMaxSpeed((s_1.v, s_2.v))$$
  3. העלות הכספית:
  $$\begin{aligned} cost_{map}^{money}&((s_1, s_2)) \\ &= roadLength((s_1.v, s_2.v)) \\ &\cdot \left[gasCostPerMeter((s_1.v, s_2.v)) + I_{IsTollRoad((s_1.v, s_2.v))} \right. \\ &\left. \cdot FixedTollRoadPricePerMeter\right] \end{aligned}$$

  הניחו שהבאים:
  $roadLength, roadMaxSpeed, gasCostPerMeter, IsTollRoad, FixedTollRoadPricePerMeter$
  נתונים או ניתנים לחישוב (בהינתן כביש במפה $(s_1.v, s_2.v)$). בהמשך התרגיל נפרט עליהם יותר.

  בחלקים הראשונים של התרגיל נשתמש בעלות הפשוטה $cost_{map}^{dist}$ ובחלקים מתקדמים נעשה שימוש גם בשתי עלויות האחרות (זמן וכסף).

- **המצב ההתחלתי:**
$$I_{map} \triangleq (v{:}\,v_{src})$$
- **מצבי המטרה:**
$$G_{map} \triangleq \{(v{:}\,v_{dst})\}$$

חלק ג' – הגדרת מרחב החיפוש של בעיית ההובלות

בהינתן רשת הכבישים, נקודת המוצא ורשימת ההזמנות, נגדיר מרחב חיפוש עבור בעיית המשלוחים:

$$DeliveriesTruckProblem = (S_d, O_d, I_d, G_d)$$

- **קבוצת המצבים:**

$$S_d \triangleq \{(v_0, \emptyset, \emptyset)\} \cup \left\{ \left(\underbrace{curLoc}_{\text{המיקום הנוכחי}}, \underbrace{Loaded}_{\text{קבוצת הובלות על המשאית}}, \underbrace{Dropped}_{\text{קבוצת הובלות שסופקו}} \right) \middle| \begin{array}{c} curLoc \in \{d.pick | d \in Loaded\} \cup \{d.drop | d \in Dropped\} \\ \emptyset \neq Loaded \cup Dropped \subseteq D \\ Loaded \cap Dropped = \emptyset \end{array} \right\}$$

- **קבוצת האופרטורים:**

<u>אופרטורים עבור העמסת הובלה:</u>

ישנם k אופרטורים כאלו. לכל $i \in [k]$ נגדיר את האופרטור $o_{d_i}^{pick}$ להיות האופרטור שבהינתן מצב $s \in S_d$, המצב העוקב $o_{d_i}^{pick}(s)$ (המתקבל מהפעלת האופרטור על המצב s) הינו מצב שבו המיקום הנוכחי הוא נק' ההעמסה $d_i.pick$ וכן ההובלה d_i נמצאת על המשאית ($d_i \in Loaded$).

הפעלת האופרטור $o_{d_i}^{pick}$ על מצב $s \in S_d$ אפשרית אמ"מ ההובלה d_i לא נמצאת כבר על המשאית ($d_i \notin s.Loaded$) ולא סופקה בעבר ($d_i \notin s.Dropped$) וכן יש די מקום פנוי במשאית עבור העמסת כל החבילות של הובלה זו ($d_i.pkgs \leq TruckCapacity - \sum_{d \in s.Loaded} d.pkgs$).

הגדרה פורמלית: לכל $i \in [k]$ נגדיר את האופרטור $o_{d_i}^{pick}$ באופן הבא:

$$\forall_{s \in S_d}: o_{d_i}^{pick}(s) \triangleq \begin{cases} (d_i.pick, s.Loaded \cup \{d_i\}, s.Dropped) & ; \left[\begin{array}{c} d_i \notin s.Loaded \cup s.Dropped \\ \wedge \\ d_i.pkgs \leq TruckCapacity - \sum\limits_{d \in s.Loaded} d.pkgs \end{array} \right] \\ \emptyset & ; \quad otherwise \end{cases}$$

וכן תחום הפעולה של האופרטור $o_{d_i}^{pick}$ מוגדר בהתאם:

$$Domain\left(o_{d_i}^{pick}\right) = \left\{ s \in S_d \middle| o_{d_i}^{pick}(s) \neq \emptyset \right\} = \left\{ s \in S_d \middle| \begin{array}{c} d_i \notin s.Loaded \cup s.Dropped \\ \wedge \\ d_i.pkgs \leq TruckCapacity - \sum_{d \in s.Loaded} d.pkgs \end{array} \right\}$$

<u>אופרטורים עבור פריקת הובלה:</u>

ישנם k אופרטורים כאלו. לכל $i \in [k]$ נגדיר את האופרטור $o_{d_i}^{drop}$ להיות האופרטור שבהינתן מצב $s \in S_d$, המצב העוקב $o_{d_i}^{drop}(s)$ (המתקבל מהפעלת האופרטור על המצב s) הינו מצב שבו המיקום הנוכחי היא נק' הפריקה $d_i.drop$ וכן ההובלה d_i מוגדרת כסופקה ($d_i \in Dropped$) ואינה מצויה כבר על המשאית ($d_i \notin Loaded$).

הפעלת האופרטור $o_{d_i}^{drop}$ על מצב $s \in S_d$ אפשרית רק אם ההובלה d_i נמצאת כבר על המשאית ($d_i \in s.Loaded$).

הגדרה פורמלית: לכל $i \in [k]$ נגדיר את האופרטור $o_{d_i}^{drop}$ באופן הבא:

$$\forall_{s \in S_d}: o_{d_i}^{drop}(s) \triangleq \begin{cases} (d_i.drop, s.Loaded \setminus \{d_i\}, s.Dropped \cup \{d_i\}) & ; \quad d_i \in s.Loaded \\ \emptyset & ; \quad otherwise \end{cases}$$

וכן תחום הפעולה של האופרטור $o_{d_i}^{drop}$ מוגדר בהתאם:

$$Domain\left(o_{d_i}^{drop}\right) = \left\{ s \in S_d \middle| o_{d_i}^{drop}(s) \neq \emptyset \right\} = \{ s \in S_d | d_i \in s.Loaded \}$$

<u>לבסוף, קבוצת כל האופרטורים הינה:</u>

$$O_d \triangleq \left\{ o_{d_i}^{pick} \right\}_{i \in [k]} \cup \left\{ o_{d_i}^{drop} \right\}_{i \in [k]}$$

- **עלות אופרטור:**
במטלה נגדיר 3 פונק' עלות עבור הפעלת אופרטור $o \in O_d$ על מצב $s \in Domain(o)$.
1. אורך המסלול הקצר ביותר על גבי המפה מהנק' בה נמצאת המשאית במצב s לנק' בה מצויה המשאית במצב $s(o)$:
$$cost_d^{dist}(s, o) = optimalDistanceOnStreetsMap(s.curLoc, o(s).curLoc)$$
2. זמן הנסיעה ביניהם:
$$cost_d^{time}(s, o) = optimalDrivingTimeOnStreetsMap(s.curLoc, o(s).curLoc)$$
3. העלות הכספית:
$$cost_d^{money}(s, o) = optimalDrivingMoneyCostOnStreetsMap(s.curLoc, o(s).curLoc)$$
   - כל אחת משלושת פונק' העלויות הללו למעשה מגדירה וריאציה לבעיה. בסופו של דבר כשפותרים בעיה צריך להחליט באיזו פונק' עלות משתמשים.
   - בחלקים הראשונים של התרגיל נשתמש בפונק' העלות הפשוטה $cost_d^{dist}$ ובחלקים מתקדמים נעשה שימוש בשתי עלויות האחרות (זמן וכסף).
   - שימו לב: בהינתן אופרטור $o \in O_d$ ומצב $s \in Domain(o)$, על מנת לחשב כל אחד משלושת פונק' העלות שהוצגו לעיל עבור (o, s), **יש צורך בפתרון של בעיית המפה**.

- **המצב ההתחלתי:**
$$I_d \triangleq (v_0, \emptyset, \emptyset)$$
- **מצבי המטרה:**
$$G_d \triangleq \{(d_i.drop, \emptyset, D) \in S | i \in [k]\}$$

תרגילים

3. יבש: מהם ערכי הקיצון (המקסימלי והמינימלי) האפשריים של דרגת היציאה במרחב החיפוש? נמקו בקצרה.
4. יבש: האם ייתכנו מעגלים במרחב המצבים שלנו? אם כן תנו דוגמה למעגל כזה, אחרת נמקו.
5. יבש: כמה מצבים יש במרחב זה (חשב עבור מקרה בו $TruckCapacity = \infty$)? האם כולם ישיגים? נמקו.
6. יבש: האם ייתכנו בורות ישיגים מהמצב ההתחלתי שאינם מצבי מטרה במרחב המצבים? אם כן – איך זה ייתכן? אם לא – למה?
7. יבש: מהם האורכים האפשריים של מסלולים אל מצב סופי? הערה: זכרו שהנחנו שכל נק' האיסוף והפריקה (וכן נק' ההתחלה) הינן זרות זו מזו.
8. יבש: איך בעצם נראה מרחב המצבים (כמו איזה מבנה מוכר)? הערך את גודלו של מבנה זה (כתוב ביטוי מתמטי שתלוי בפרמטר k). לצורך הקירוב הנח כי דרגת היציאה של כל מצב שווה ל-$(minOutRank + maxOutRank)/2$.
9. יבש: הגדירו פורמלית ובצורה ישירה את פונקציית העוקב $Succ: S \to \mathcal{P}(S)$ המתאימה לבעיה זו (ללא שימוש בקבוצת האופרטורים O).
שימו לב, אנו מצפים לביטוי מהצורה: $Succ(s) = \{(?,?,?)|?\} \cup \{(?,?,?)|?\}$

חלק ד' – מתחילים לתכנת

הורידו את *ai_hw1.zip* מהאתר וטענו את התיקייה שבתוכו לסביבת העבודה המועדפת עליכם.

מבנה מפת הדרכים

בתרגיל נעשה שימוש במפת רשת הכבישים של העיר תל אביב. את המפה אנו טוענים פעם אחת בקובץ main.py למשתנה גלובלי בשם streets_map. המפות מיוצגות ע"י אובייקט מטיפוס ***StreetsMap***. הטיפוס ***StreetsMap*** יורש מ- *dict*; כלומר ***StreetsMap*** הינו בבסיסו מיפוי ממזהה ייחודי של צומת במפה (מספר שלם) לאובייקט מטיפוס ***Junction*** שמייצג את אותו הצומת.

כל צומת הוא כאמור מטיפוס ***Junction***. לצומת יש את השדות הבאים: (1) מספר *index* ייחודי; (2+3) קואורדינטות lat, lon (קווי אורך ורוחב) של המיקום הגיאוגרפי של הצומת במפה; ו- (4) רשימה outgoing_*links* המכילה את כל הקשתות לשכניו. כל קשת כזו מייצגת כביש במפה. קשת היא אובייקט מטיפוס ***Link*** עם מאפיינים source ו- target – המזהים של צמתי המקור והיעד של הקשת, distance – אורך הכביש (במטרים), max_speed – מהירות הנסיעה המקסימלית בכביש (ביחידות של מטר/דקה), וכן is_toll_road – שדה בולאני שמציין האם זהו כביש אגרה.

שימו לב: אין לבצע באף שלב טעינה של מפות. טענו בשבילכם את המפות פעם אחת בתחילת קובץ ה- main.py שסיפקנו לכם. יש לכם גישה למפות בכל מקום בו תזדקקו להן. באופן כללי, טעינות מיותרות בקוד יגרמו להגדלת זמן הפתרון ואולי יובילו לחריגה מהזמן המקסימלי.

הכרת תשתית הקוד הכללית (שסופקה לכם בתרגיל זה) לייצוג ופתרון בעיות גרפים

המחלקות **GraphProblem**, **GraphProblemState** (בקובץ graph_search/graph_problem_interface.py) מגדירות את הממשק (interface) בו נשתמש על מנת לייצג <u>מרחב מצבים</u>. אלו הן מחלקות <u>אבסטרקטיות</u> – כלומר מוגדרות בהן מתודות שאינן ממומשות. לכן, בפרט, לא ניתן ליצור ישירות אובייקט מטיפוסים אלו (ואין לכך שום משמעות). כדי להגדיר מרחב מצבים חדש יש לרשת (inherit) משתי המחלקות הנ"ל. בהמשך התרגיל תראו דוגמא למימוש של מרחב מצבים באופן הנ"ל (שסיפקנו עבורכם) ותממשו מרחב נוסף כזה בעצמכם.

המחלקה **GraphProblemSolver** (באותו הקובץ) מגדירה את הממשק בו נשתמש בכדי לחפש בגרפים. למחלקה יש מתודה אבסטרקטית אחת בשם ()solve_problem שמקבלת כפרמטר בעיה (אובייקט מטיפוס שיורש מ- GraphProblem) ומחזירה את תוצאות החיפוש (אובייקט מטיפוס SearchResult). כל אלג' חיפוש שנממש ישתמש בממשק הנ"ל (ירש ממחלקה זו או ממחלקה שיורשת ממנה).

שימו לב: אלגוריתמי החיפוש אותם נממש לאורך התרגיל יהיו כלליים בכך שלא יניחו כלום על הבעיות אותן יפתרו, פרט לכך שהן תואמות לממשק המוגדר ע"י GraphProblemState, GraphProblem. כלומר, בעתיד תוכלו לקחת את המימוש שלכם מקורס זה כפי שהוא בכדי לפתור בעיות חדשות.

המחלקה **BestFirstSearch** (בקובץ graph_search/best_first_search.py) יורשת מהמחלקה GraphProblemSolver (שתוארה לעיל) ומייצגת אלגוריתמי חיפוש מהמחלקה Best First Search. כפי שנלמד בכיתה, אלו הם אלגוריתמים שמתחזקים תור עדיפויות בשם **open** של צמתים (פתוחים) הממתינים לפיתוח. כל עוד תור זה אינו ריק, האלג' בוחר את הצומת הבא בתור העדיפויות ומפתח אותו. המחלקה ממששת את המתודה ()solve_problem בהתאם. דוגמאות לאלגוריתמים ממשפחה זו: *Uniform Cost, Greedy Best Search, A. כאמור, Best First Search הינה <u>משפחה</u> של אלגוריתמי חיפוש (מכונה גם "אלגוריתם גנרי"), כלומר היא מגדירה שלד כללי של מבנה האלגוריתם, ומשאירה מספר פרטי מימוש חסרים. לכן, המחלקה BestFirstSearch אף היא <u>אבסטרקטית</u>. גם בה מוגדרות מספר מתודות אבסטרקטיות שעל היורש (אלגוריתם החיפוש הקונקרטי) לממש. המתודה האבסטרקטית ()_calc_node_expanding_priority מאפשרת ליורש להגדיר את אופן חישוב ערך ה- f-score של צומת. כזכור, ערך זה משמש כעדיפות של צומת בתור העדיפויות open (בתרגיל זה אנו מכנים ערך זה בשם expanding priority). המתודה האבסטרקטית ()_open_successor_node מאפשרת ליורש להגדיר את אופן הטיפול בצומת חדש שזה עתה נוצר ומייצג מצב עוקב של המצב המיוצג ע"י הצומת שנבחר אחרון לפיתוח (הכנסה ל- open, בדיקה ב- close במידת הצורך). בנוסף, האלגוריתם מאפשר מצב של חיפוש-גרף כפי שנלמד בכיתה, ע"י תחזוק אוסף **סגור / close** של צמתים שכבר פיתחנו במהלך החיפוש (ה- constructor של BestFirstSearch מקבל פרמטר בולאני בשם use_close שקובע האם להשתמש ב- close).

מבנה הקלטים לבעיית ההובלה ואופן טעינתם

המחלקה **DeliveriesTruckProblemInput** (בקובץ deliveries/deliveries_truck_problem_input.py) מייצגת <u>קלט</u> לבעיית ההובלות. מחלקה זו אחראית לטעינה של קלטים שסיפקנו לכם כקבצי טקסט. המחלקה שמייצגת את בעיית ההובלה (נראה בהמשך) תקבל אובייקט מסוג זה. בקובץ הראשי main.py כבר כתובות

שורות הקוד שאחראיות להשתמש במחלקה זו ע"מ לטעון את הקלטים הנדרשים במקומות הנדרשים.
הבהרה: אין לבצע טעינות נוספות של הקלטים. אנו עשינו זאת בשבילכם במקומות הנדרשים.

תרגילים

10. רטוב + יבש: סעיף זה נועד על מנת להתחיל להכיר את מבנה הקוד.
    a. חלצו את תוכן התיקייה ai_hw1.zip.
    b. אם אתם משתמשים ב- IDE לכתיבת והרצת קוד פייתון (אנחנו ממליצים מאוד על PyCharm), פתחו פרויקט חדש שתיקיית האם שלו היא התיקייה הראשית של קובץ ה- zip שחולץ (אמור להיות שם קובץ בשם main.py).
    c. פתחו את הקובץ main.py, קראו את החלק בקוד שמעליו מופיעה הערה המתאימה למספר סעיף זה. שורות קוד אלו מבצעות: יצירת בעיית מפה חדשה, יצירת אובייקט מסוג אלג' חיפוש uniform cost, הרצת אלג' החיפוש על הבעיה ולבסוף הדפסת התשובה שהתקבלה מההרצה. הריצו את הקובץ. וודאו שמודפסת לכם שורה למסך שמתארת את פתרון בעיית החיפוש במפה. זאת גם הזדמנות טובה לוודא שהחבילות numpy, scipy, networkx, matplotlib מותקנות אצלכם כראוי.
    d. פתחו את הקובץ deliveries/map_problem.py. בתוכו יש לכם שתי משימות (המסומנות ע"י הערות **TODO** כמו בעוד מקומות רבים לאורך המטלה). אחת במתודה בשם expand_state_with_costs() והשנייה במתודה בשם is_goal(). בשתי משימות אלו אתם מתבקשים לבצע שינוי בקוד של המחלקה MapProblem כדי לתקן ולהשלים את המימוש שסיפקנו לכם.
    e. זוהי בעיה פשוטה ולכן נוח להתחיל בה כדי להתמצא בקוד שסופק לכם. עיינו במימוש של המחלקות בקובץ זה וודאו שאתם מבינים את החלקים השונים. שימו לב שמחלקה זו יורשת מהמחלקה GraphProblem (שתוארה מקודם) ומממשת את המתודות האבסטרקטיות הנדרשות.
    f. עתה, לאחר תיקון קוד המחלקה MapProblem, הריצו בשנית את main.py. הוסיפו לדו"ח את פלט הריצה המתוקנת.
    g. שאלה יבש: בתכנות לפעמים אנחנו רוצים לכפות על מבני נתונים / טיפוסים מסוימים להיות immutable/frozen. הכוונה היא שאחרי יצירת אובייקט מטיפוס שכזה לא יהיה ניתן לשנותו. הצהרה על טיפוס כ"קפוא" מגבילה אותנו, אך יחד עם זאת היא גם מגינה עלינו. (i) העתק לדו"ח את שורת הקוד הרלוונטית שקובעת שאובייקטים מהטיפוס MapState יהיו בלתי ניתנים לשינוי. (ii) הסבר למה אנחנו רוצים לעשות זאת ספציפית עבור הטיפוס GraphProblemState – תן דוגמא לבאג במתודה expand_state_with_costs במחלקה MapProblem שיוביל לתוצאה שגויה בחלק מהמקרים כאשר נריץ אלג' חיפוש על המרחב הנ"ל.

חלק ה' – אלגוריתם A*

עתה נתחיל במימוש Weighted A*.

עיינו בקובץ framework/graph_search/astar.py. שם מופיע מימוש חלקי למחלקה AStar. שימו לב: המחלקה AStar יורשת מהמחלקה האבסטרקטית BestFirstSearch (הסברנו עליה בחלק ד'). זהו את החלק בהצהרת המחלקה AStar בו הירושה מוגדרת. המחלקה AStar צריכה לממש את המתודות האבסטרקטיות שמוגדרות ע"י BestFirstSearch. הכותרות של מתודות אלו מופיעות כבר במימוש החלקי של המחלקה AStar, אך ללא מימושן. בסעיף זה נרצה להשלים את המימוש של המחלקה AStar ולבחון אותה.

שימו לב: לאורך התרגיל כולו אין לשנות את החתימות של המתודות שסיפקנו לכם. בנוסף, אין לשנות קבצים שלא התבקשתם באופן מפורש.

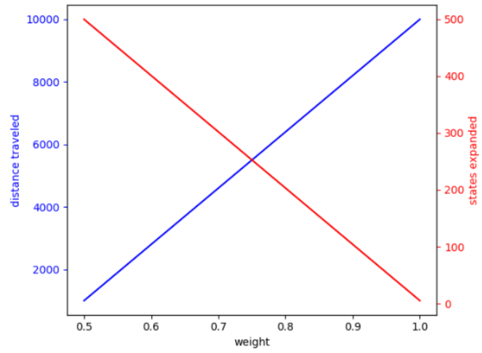
תרגילים

11. רטוב: השלימו את המשימות הדרושות תחת הערות ה- **TODO** בקובץ framework/graph_search/astar.py כך שנקבל מימוש תקין לאלגוריתם Weighted A*, כפי שראיתם בהרצאות. בכדי להבין את מטרת המתודות השונות שעליכם לממש, הביטו במימוש המחלקה BestFirstSearch שעושה בהן שימוש. בנוסף, היעזרו במימוש שסיפקנו לכם ל- UniformCost (בקובץ framework/graph_search/uniform_cost.py). שימו לב בשקפים מההרצאה להבדלים בין אלג' UniformCost לבין אלג' A*.
12. רטוב: בכדי לבחון את האלג' שזה עתה מימשתם, השלימו את המשימות הדרושות תחת הערות ה- **TODO** הרלוונטיות לסעיף זה בקובץ main.py. כידוע, לצורך הרצת A* יש צורך בהיוריסטיקה. ה- constructor של המחלקה AStar מקבל את טיפוס ההיוריסטיקה שמעוניינים להשתמש בה. לצורך בדיקת שפיות, הפעילו את ה- A* על בעיית המפה שפתרתם בסעיף הקודם עם NullHeuristic (מסופקת בקובץ framework/graph_search/graph_problem_interface.py. מחלקה זו כבר מוכרת מ- main.py ללא צורך בביצוע import נוסף. באופן כללי אין לעשות imports בתרגיל זה כלל). וודאו שהתוצאה המודפסת זהה לזו שקבלתם בעזרת Uniform Cost.
13. רטוב + יבש: כפי שראינו בהרצאות ובתרגולים, היוריסטיקה פשוטה לבעיית המפה היא מרחק אווירי לפתרון. היכנסו לקובץ deliveries/map_heuristics.py וממשו את ההיוריסטיקה הזו במחלקה AirDistHeuristic (מלאו את המקומות החסרים תחת ההערות שהשארנו לכם שם). כעת הריצו שוב את הבעיה שפתרתם בסעיף הקודם, אך כעת בעזרת ההיוריסטיקה (מלאו ב- main.py את המשימות שקשורות לסעיף זה). העתיקו לדו"ח את פלט הריצה. כתוב בדו"ח את מס' פיתוחי המצבים היחסי שחסכנו לעומת הריצה העיוורת (ההפרש חלקי מס' הפיתוחים בריצה עם ההיוריסטיקה).

שימו לב: בכדי לחשב מרחק בין זוג Junctions, **אין לחשב את המרחק האווירי ישירות על ידי קווי רוחב ואורך**, אלא יש להשתמש במתודה calc_air_distance_from() של המחלקה Junction.

14. רטוב + יבש: כעת נרצה לבחון את השפעת המשקל w על ריצת wA*. מלאו בקובץ main.py את המשימות הרלוונטיות לסעיף זה. בנוסף, ממשו את הפונק' run_astar_for_weights_in_range() שחתימתה מופיעה בקובץ main.py. פונק' זו מקבלת היוריסטיקה ובעיה לפתרון ומשתמשת באלג' wA* בכדי לפתור את בעיה זו תוך שימוש בהיוריסטיקה הנתונה ועם n משקולות שונות בתחום הסגור [0.5,1]. את התוצאות של ריצות אלו היא אמורה לשמור ברשימות ולאחר מכן היא אמורה לקרוא לפונק' בשם plot_distance_and_expanded_wrt_weight_figure() (שגם בה עליכם להשלים את המימוש באיזורים החסרים). פונק' זו אחראית ליצור גרף שבו מופיעות 2 עקומות: אחת מהעקומות (הכחולה) מתארת את טיב הפתרונות (בציר y) כפונק' של המשקל (אורך המסלול במקרה של בעיית המפה הבסיסית). העקומה השנייה (האדומה) מתארת את מספר המצבים שפותחו כפונק' של המשקל. עתה השתמשו בפונק' run_astar_for_weights_in_range() מהמקום הרלוונטי ב- main.py (מספר סעיף זה מצוין במקום זה) ע"מ ליצור את הגרף המתאים עבור פתרון בעיית המפה תוך שימוש בהיוריסטיקה AirDistHeuristic. צרפו את הגרף שנוצר לדו"ח. הסבירו את הגרף שהתקבל. ציינו באיזה ערך w הייתם בוחרים ולמה. בכיתה למדתם כלל אצבע לפיו "ככל ש- w קטן יותר כך הפתרון איכותי יותר ומס' הפיתוחים גדול יותר". הכלל הנ"ל מצביע על מגמה כללית, אך איננו נכון באופן גורף (כלומר ייתכנו זוג ערכים $w_1 < w_2$ עבורם הפתרון המתקבל עם w_1 פחות טוב מאשר הפתרון המתקבל עם w_2 ו/או מס' הפיתוחים עם w_2 גדול יותר ממס' הפיתוחים עם w_1). כיצד הכלל שהוזכר והדגש הנ"ל באים לידי ביטוי בתרשים שקיבלתם? על

התרשים להראות כמו בדוגמה הזו (צורת העקומות עצמן עשויה להשתנות כמובן):



חלק ו' – מימוש בעיית ההובלות

כעת נרצה לממש את המחלקה שמייצגת את מרחב המצבים של בעיית ההובלות. בבעיה זו נרצה למצוא סדר אופטימאלי להעמסת ופריקת ההובלות תוך התחשבות באילוצי הבעיה כפי שתוארה בחלק ג'.

בשאלות הוכח / הפרך קבילות של היוריסטיקה: אם אתם סבורים שההיוריסטיקה קבילה יש לספק הוכחה לכך. אם אתם סבורים שהיא איננה קבילה יש לספק דוגמא של מרחב חיפוש קטן ככל שתוכלו (ציירו גרף בו הצמתים הם נקודות במפה) עבורו הערך ההיוריסטי על אחד המצבים לפחות גדול ממש מעלות הפתרון האופטימלי למטרה.

15. רטוב: התבוננו בקובץ deliveries/deliveries_truck_problem.py והשלימו את המימושים החסרים במתודות הבאות:
    a. DeliveriesTruckState.__eq__()
    b. DeliveriesTruckState.get_total_nr_packages_loaded()
    c. DeliveriesTruckProblem.get_deliveries_waiting_to_pick()
    d. DeliveriesTruckProblem.expand_state_with_costs()
    e. DeliveriesTruckProblem.is_goal()

    המתודה ()DeliveriesTruckProblem.expand_state_with_costs אמורה להחזיר בין היתר גם את עלות האופרטור שהופעל. כזכור, בחלק ג' ציינו כי בכדי לחשב את עלות האופרטור יש לפתור בעיה על רשת הכבישים. במימוש אנחנו אכן עושים זאת. בהערות בקוד (במתודה *expand_state_with_costs()*) הורנו לכם להשתמש בשדה (של הבעיה) בשם *map_distance_finder* בו שמור אובייקט מטיפוס ***CachedMapDistanceFinder***, שלו יש מתודה בשם *get_map_cost_between()* המחשבת ומחזירה את עלות פתרון אופטימלי על בעיית מפות הכבישים. מאחורי הקלעים המתודה הזו למעשה אמורה ליצור בעיית MapProblem חדשה ולקרוא ל- ()AStar.solve_problem בכדי לפתור אותה. אך לפני זה, לטובת היעילות, היא בודקת האם כבר פתרנו בעיה זו בעבר ואם כן מאתרת את הפתרון שדאגנו לשמור כשפתרנו בעיה זאת לראשונה ומחזירה אותו מיד וללא חישובים נוספים. במובן זה המחלקה CachedMapDistanceFinder שומרת ב- cache הפנימי שלה תוצאות של חישובים קודמים.

    f. ממשו את המתודה ()CachedMapDistanceFinder.get_map_cost_between בקובץ deliveries/cached_map_distance_finder.py בהתאם להוראות המופיעות שם.
16. השלימו את הקוד ב- main.py תחת ההערה הרלוונטית לסעיף זה. הריצו את הקוד הנ"ל (הרצת UniformCost על בעיית ההובלות עם הקלט הקטן). המטרה היא לוודא שהקוד שרשמתם בסעיף הקודם באמת רץ בהצלחה.

עתה, כדי להריץ את *A על הבעיה, יש ראשית להגדיר (ולממש) היוריסטיקות עבור הבעיה.

17. רטוב: השלימו את המימוש עבור המתודה ()DeliveriesTruckProblem.get_all_junctions_in_remaining_truck_path (בקובץ deliveries/deliveries_truck_problem.py). לאחר מכן, השלימו את המימוש עבור ההיוריסטיקה TruckDeliveriesMaxAirDistHeuristic (בקובץ deliveries/deliveries_truck_heuristics.py). היוריסטיקה זו מתבוננת בכל הצמתים (במפת הכבישים) שיש למשאית עוד לעבור בהן (כולל המיקום הנוכחי), ולוקחת את המרחק האווירי הגדול ביותר בין כל זוג מתוך קב' צמתים זו.
18. רטוב: השלימו את הקוד ב- main.py תחת ההערה הרלוונטית לסעיף זה. הריצו את הקוד הנ"ל (הרצת AStar על בעיית ההובלות עם ההיוריסטיקה שמומשה בסעיף הקודם). המטרה היא לוודא שהקוד שרשמתם בסעיף הקודם באמת רץ בהצלחה ולבחון את התוצאות המתקבלות.
19. יבש: הוכח/הפרך: ההיוריסטיקה TruckDeliveriesMaxAirDistHeuristic הינה קבילה (עבור פונק' המחיר $cost_d^{dist}$).
20. רטוב: השלימו את המימוש עבור ההיוריסטיקה TruckDeliveriesSumAirDistHeuristic (בקובץ deliveries/deliveries_truck_heuristics.py). היוריסטיקה זו מתבוננת בכל הצמתים (במפת הכבישים) שיש למשאית עוד לעבור בהן (כולל המיקום הנוכחי), ומחשבת את עלות המסלול הבא: מסלול זה מתחיל בנק' הנוכחית בה נמצאת המשאית. הנקודה ה- $i + 1$ במסלול היא הקרובה ביותר לנק' i במסלול (מבחינת מרחק אווירי) מתוך כל הנק' שנותרו לביקור וטרם נבחרו למסלול זה.
21. רטוב: השלימו את הקוד ב- main.py תחת ההערה הרלוונטית לסעיף זה. הריצו את הקוד הנ"ל (הרצת Astar על בעיית ההובלות עם ההיוריסטיקה שמומשה בסעיף הקודם). המטרה היא לוודא שהקוד שרשמתם בסעיף הקודם באמת רץ בהצלחה ולבחון את התוצאות המתקבלות.

22. יבש: הוכח/הפרך: ההיוריסטיקה TruckDeliveriesSumAirDistHeuristic הינה קבילה (עבור פונק' המחיר $cost_d^{dist}$).
23. רטוב: השלימו את המימוש עבור ההיוריסטיקה TruckDeliveriesMSTAirDistHeuristic (בקובץ deliveries/deliveries_truck_heuristics.py). היוריסטיקה זו מתבוננת בכל הצמתים (במפת הכבישים) הנותרים שעל המשאית לעבור בהן (כולל המיקום הנוכחי של המשאית), ובונה גרף שכולל את כל צמתים אלו וקשת בין כל זוג צמתים שמשקלה מוגדר להיות המרחק האווירי בין זוג צמתים אלו. בשלב זה מחושב עץ פורס מינימלי על הגרף הנ"ל. משקל העץ שחושב הוא הערך ההיוריסטי.
24. רטוב: השלימו את הקוד ב- main.py תחת ההערה הרלוונטית לסעיף זה. הריצו את הקוד הנ"ל (הרצת AStar על בעיית ההובלות עם ההיוריסטיקה שמומשה בסעיף הקודם). המטרה היא לוודא שהקוד שרשמתם בסעיף הקודם באמת רץ בהצלחה ולבחון את התוצאות המתקבלות.
25. יבש: הוכח/הפרך: ההיוריסטיקה TruckDeliveriesMSTAirDistHeuristic הינה קבילה (עבור פונק' המחיר $cost_d^{dist}$).
26. רטוב + יבש: עתה נריץ את wA* עם ערכי w שונים כדי לצייר גרף שמציג את מגמת מחיר הפתרון מגמת מס' הפיתוחים כאשר w משתנה בתחום [0.5,1]. לצורך כך נשתמש בפונק' run_astar_for_weights_in_range() שכבר מימשנו בשלבים מוקדמים. השלימו בקובץ main.py את הקוד תחת ההערה הרלוונטית לסעיף זה. צרפו את הגרף שנוצר לדו"ח. הסבירו את הגרף שהתקבל. ציינו באיזה ערך w הייתם בוחרים ולמה.

שימו לב: הסעיפים האחרונים יכולים לעזור לכם לוודא שהאלגוריתמים שלכם אכן עובדים כשורה. ודאו שהתוצאות שקיבלתם <u>הגיוניות</u>.

חלק ז' – מימוש והשוואת פונק' עלות זמן מול כסף

עד כה לשם פשטות התמקדנו בממד של המרחק על מערכת הכבישים. בפועל, בבעיות מהחיים בדרך כלל אנחנו רוצים למזער ניצול משאבי זמן וכסף. כאמור, בימים בהם שלומי (השליח) עמוס בלימודים הוא מעוניין למצוא פתרון שימזער את זמן העבודה שלו. בימים בהם שלומי אינו עמוס בלימודים הוא מעוניין למזער את עלויות הנסיעה כדי למקסם את הרווח. בחלק זה נרחיב את הבעיה שלנו כדי שתוכל להתאים גם למדדים אלו.

למעשה בחלק ג' הוצגו 3 פונק' עלויות שונות. בסופו של דבר כשפותרים את הבעיה מקבעים פונק' עלות אחת שאיתה עובדים (היא תקבע את עלות האופרטורים והמסלולים). היינו רוצים דרך לקבוע בקוד באיזו פונק' עלות להשתמש עבור בעיית ההובלות. איך זה נעשה? ה- constructor של המחלקה ***DeliveriesTruckProblem*** מקבל פרמטר בשם *optimization_objective* מטיפוס ***OptimizationObjective*** (זהו *enum* שערכיו האפשריים הם *Distance*, *Time*, *Money*). העברת הערך בעת יצירת בעיה מגדיר וריאנט של הבעיה (קובע את פונק' העלות להיות אחת מ- $cost_d^{dist}, cost_d^{time}, cost_d^{money}$).

עד כה בסעיפים הקודמים כאשר יצרנו בעיית משלוחים העברנו לפרמטר *optimization_objective* את הערך *OptimizationObjective.Distance* ובכך הורנו למחלקה *DeliveriesTruckProblem* להשתמש בפונק' העלות $cost_d^{dist}$. ערך זה נשמר באובייקט הבעיה תחת השדה *optimization_objective*. המחלקה מתחשבת בערך הזה כשהיא מחשבת את עלות האופרטור (תכף נראה באיזה אופן זה קורה).

כפי שציינו בחלק ג', בכדי לחשב את ערך כל אחת מפונק' העלויות $cost_d^{dist}, cost_d^{time}, cost_d^{money}$ (כדי לחשב עלות אופרטור המופעל מצב מסוים) צריך לפתור בעיית חיפוש על מפת הכבישים המותאמת לאותה פונק' העלות. ואכן כך עשיתם בסעיף בו מימשתם את המתודה ()expand_state_with_cost של המחלקה DeliveriesTruckProblem – קראתם למתודה ()map_distance_finder.get_map_cost_between בכדי לחשב את עלות האופרטור על מצב נתון. נזכיר כי זאת גם בהתאם לאופן הגדרת פונק' העלויות $cost_d^{dist}, cost_d^{time}, cost_d^{money}$ (כפי שניתנה בחלק ג') – כל אחת מהן הוגדרה להיות עלות של מסלול אופטימלי על מפת הכבישים (בשלב זה חזרו לרגע לחלק ג' והביטו שוב בהגדרות הללו). עבור $cost_d^{dist}$ למשל, עלות הפעלת אופרטור $o \in O_d$ על מצב $s \in S_d$ הוגדרה להיות אורך המסלול הקצר ביותר במפת הכבישים מ- $s.curLoc$ ועד ל- $o(s).curLoc$. לכן פתירת בעיית מציאת המסלול הקצר ביותר במפה נתנה לנו מידית את הערך של $cost_d^{dist}$.

עתה, נצטרך לדאוג לכך שבעיית המפה הפנימית תמצא מסלול שממזער את זמן הנסיעה או את העלות הכספית של הנסיעה (בהתאם לאופן הגד' האופרטורים $cost_d^{time}, cost_d^{money}$). זו תהיה המטרה הבאה שלנו. אבל קודם – נשתכנע שזה אכן יעזור לנו. למשל, פונק' העלות $cost_d^{time}$ על אופרטור $o \in O_d$ ומצב $s \in S_d$ הוגדרה להיות <u>זמן הנסיעה הקצר ביותר</u> מנק' $s.curLoc$ לנק' $o(s).curLoc$. אם נפתור את בעיית המפה כאשר עלות כביש הוא <u>זמן</u> הנסיעה באותו הכביש, אז עלות המסלול האופטימלי (ע"פ מדד זה) במפה הוא בדיוק הערך הרצוי של $cost_d^{time}$. נפלא.

נעבור לפרטים הטכניים. נשים לב שבתוך המימוש של ה- constructor של המחלקה DeliveriesTruckProblem, יש יצירה אובייקט מטיפוס CachedMapDistanceFinder (שאחראי לפתור את בעיות מפת הכבישים) ושם אנחנו מעבירים לו פרמטר road_cost_fn=self._calc_map_road_cost. כלומר אנחנו אומרים לו מהי <u>פונק' העלות</u> שיש להשתמש בעת פתרון <u>בעיות המפה</u>.

כלומר, המתודה ()DeliveriesTruckProblem._calc_map_road_cost היא זו שקובעת את עלותו של כביש בבעיית המפה. זהו – עכשיו רק צריך לגרום למתודה הזאת להחזיר את הערך הנכון (מרחק / זמן / עלות כספית) של כביש, בהתאם לערך של השדה *optimization_objective*. זה מה שיגרום לפונק' העלות של בעיית ההובלות להיות מחושבות כהלכה.

ואכן, המתודה ()DeliveriesTruckProblem._calc_map_road_cost מקבלת אובייקט מטיפוס ***Link*** (כביש) ומחזירה אובייקט מטיפוס DeliveryCost (שמכיל את השדות distance_cost, time_cost, money_cost). כרגע, המתודה מחזירה את הערך הנתון עבור distance_cost, אך מחזירה ערך קבוע של 0 עבור time_cost, money_cost. המשימה הבאה שלכם תהיה לתקן את המתודה הזאת כך שתחזיר את עלויות הזמן והכסף הנכונות עבור הכביש הנתון. נסביר עכשיו איך מחשבים זאת.

לכל כביש במפה (אובייקט מטיפוס ***Link***) יש שדה בשם max_speed. שדה זה מתאר את מהירות הנסיעה המקסימלית המותרת בכביש זה ע"פ חוק (ביחידות של מטר/דקה). בנוסף, חלק מהכבישים הינם כבישי אגרה (השדה is_toll_road מציין האם כביש הינו כביש אגרה). עלות נסיעה בכבישי אגרה הינה אורך הכביש (שניתן במטרים ע"י השדה distance) כפול הקבוע problem.problem.input.toll_road_cost_per_meter.

שלומי יכול לבחור לנסוע בכל מהירות אפשרית כל עוד היא לא חורגת מהמהירות המקסימלית של הכביש. כאשר המטרה שלו היא למזער את זמן הנסיעה – הוא כמובן יבחר תמיד לנסוע במהירות המקסימלית. כאשר המטרה של שלומי היא למזער את העלות הכספית – הוא לעיתים יבחר לנסוע במהירות נמוכה ממהירות הנסיעה האופטימלית. נסביר למה. למשאית של שלומי יש "מהירות שיוט" (מהירות בה צריכת הדלק פר מטר נסיעה הינה מינימלית). אם מהירות השיוט נמוכה מהמהירות

האופטימלית, שלומי יבחר לנסוע במהירות שיוט וצריכת הדלק שלו (וזמן הנסיעה שלו) יהיו בהתאם. למעשה, ישנו חישוב הקובע בהינתן המהירות המקסימלית של הכביש ובהינתן מטרת האופטימיזציה (מרחק / זמן / עלות כספית), מהי המהירות האופטימלית בה כדאי לשלומי לנסוע ומהי צריכת הדלק פר מטר עבור מהירות זאת. החישוב הנ"ל מתחשב באמור לעיל. למתודה שמבצעת את החישוב הנ"ל קוראים problem.problem_input.delivery_truck.calc_optimal_driving_parameters(), היא כבר ממומשת במלואה ואנחנו כבר קראנו לה בשבילכם מתוך המתודה _calc_map_road_cost() אותה אתם מתבקשים לממש. השתמשו בערכים שהיא מחזירה optimal_velocity, gas_cost_per_meter ע"מ לחשב את time_cost ואת money_cost.

27. רטוב: תקנו את המימוש של המתודה _calc_map_road_cost() במחלקה DeliveriesTruckProblem בקובץ deliveries/deliveries_truck_problem.py בהתאם להוראות הרשומות שם ולמבוא הרשום מעלה בחלק זה.
28. רטוב: עתה נוצרה תקלה. בסעיף הקודם דאגנו לכך שעלות המסלולים תהיה במובנים של זמן / כסף כאשר נבחר להשתמש בפונק' עלות של זמן / כסף. אבל ההיוריסטיקות עדיין יודעות להחזיר ערכים רק במובנים של מרחק. בכדי שההיוריסטיקות של בעיית המשלוחים תפעלנה כראוי (תהיינה מיודעות וקבילות), נצטרך להתאים אותן להחזיר ערכים גם במובנים של זמן / כסף (בהתאם לקביעת optimization_objective). כאשר השלמתם את המימוש של ההיוריסטיקות הללו בחלקים קודמים חישבתם חסם / הערכה למרחק הנותר במסלול של המשאית (במטרים). לאחר מכן, הקוד שניתן לכם מראש לקח את הערך הנ"ל שחישבתם והפעיל עליו פונק' בשם get_cost_lower_bound_from_distance_lower_bound(). הפונק' הזו (כשמה כן היא) אמורה לקחת חסם תחתון / הערכה של מרחק (במטרים) ולהחזיר חסם תחתון / הערכה של מרחק או מהירות או עלות כספית (בהתאם ל- optimization_objective שנבחר עבור המטרה). לכן, למזלנו, אין צורך לשנות כלום בשלב זה במימוש של ההיוריסטיקות עצמן.
השלימו את המימוש של המתודה get_cost_lower_bound_from_distance_lower_bound() במחלקה DeliveriesTruckProblem בקובץ deliveries/deliveries_truck_problem.py בהתאם להוראות הרשומות שם.
29. רטוב + יבש: השלימו בקובץ main.py את הקוד תחת ההערה הרלוונטית לסעיף זה. צרפו את התוצאות שקיבלתם לדו"ח (רק את העלויות של שלושת הפתרונות השונים עם ציון של איזו פונק' עלות הייתה בשימוש בכל תוצאה). הדגישו איך רואים בתוצאות שהפתרון המתקבל אכן ממזער את המדד הרלוונטי בהתאם לפונק' העלות שהופעלה.

שילוב בין 2 המדדים (מדד הזמן ומדד העלות הכספית) – יבש בלבד

לעיתים אנו מעוניינים למצוא פתרון שמתחשב ביותר ממדד אחד. במקרה שלנו, היינו רוצים איכשהו להתחשב גם במדד הזמן וגם במדד העלות הכספית. הקושי הוא ששני המדדים הללו הם ביחידות שונות. לא תמיד קל לחשוב על תרגום טבעי בין שני המדדים (בסגנון "1 שעה = 100 שקלים"). לכן הגדרה של מדד משותף היא לא פשוטה (כזה שמשרה יחס סדר מלא על קב' הפתרונות האפשריים תוך התחשבות בשני המדדים). לכן, יותר טבעי לנו לחשוב על מדד בסגנון – מבין כל הפתרונות ה"כמעט" טובים ביותר מבחינה כספית, הבא לי את זה שממזער את הזמן.

נציג הצעה לשילוב בין 2 המדדים: נניח שעלות הפתרון שממזער את מדד הכסף הינו m^*. נקבע ערך $\varepsilon_m > 0$. מבין כל הפתרונות האפשריים שעלותם $(1+\varepsilon)\cdot m^*$, נחפש פתרון שממזער את מדד הזמן (בתוך אותה הקבוצה).

הצגה פורמלית:

$$MoneyEpsOptimal \triangleq \left\{ g \in G_d \middle| cost_d^{money}(g) \le (1+\varepsilon_m) \cdot \min_{g' \in G_d} cost_d^{money}(g') \right\}$$

$$\forall_{g \in G_d}: isOptimal(g) \triangleq \left[g \in MoneyEpsOptimal \wedge cost_d^{time}(g) = \min_{g' \in MoneyEpsOptimal} cost_d^{time}(g') \right]$$

30. יבש: הסטודנט עומר הציע להשתמש באלג' $A^*\varepsilon$ (עם $\varepsilon = \varepsilon_m$) כדי למצוא פתרון אופטימלי (ע"פ המדד שהוגדר) באופן הבא: נפתור את בעיית המשלוחים בגרסתה עם פונק' העלות $cost_d^{money}$, ונשתמש בפונק' העלות $cost_d^{time}$ לצורך מיון בתוך קב' ה- FOCAL. האם הפתרון שהציע עומר אכן מחזיר פתרון אופטימלי כפי שהגדרנו מעלה? אם כן – הוכח; אחרת - הבא דוגמא למרחב חיפוש (קטן) בו הרצת $A^*\varepsilon$ כפי שהציע עומר תוביל לפתרון שגוי.
31. יבש: הסטודנטית רותם טוענה שניתן לשנות את אלג' A^* כך שימצא פתרון אופטימלי ע"פ המדד שהוגדר מעלה. (i) הצע תיקון לאלג' A^* כך שיפתור את הבעיה; (ii) האם קיימים חסרונות בולטים באלג' שהצעת? (מבחינת ביצועים) אם כן, מהם?

חלק ח' – מימוש האלג' $A^*\varepsilon$ והרצתו

32. רטוב: ממשו את החלקים החסרים של אלג' $A^*\varepsilon$ בקובץ framework/graph_search/astar_epsilon.py ע"פ ההנחיות המופיעות שם.
33. רטוב + יבש: מימשנו היוריסטיקה קבילה (MST) והיוריסטיקה לא קבילה אך מיודעת יותר (Sum). הבעיה היא שאין לנו אף הבטחה על איכות הפתרון שמניב A* עם היוריסטיקה שאינה קבילה. נרצה לנצל את הבטחת איכות הפתרון של $A^*\varepsilon$ כדי לעשות שימוש מועיל בהיוריסטיקה שאינה קבילה במטרה לחסוך במספר הפיתוחים מבלי לפגוע באופן דרסטי באיכות הפתרון. השלימו בקובץ main.py את הקוד תחת ההערה הרלוונטית לסעיף זה. צרפו את התוצאות שקיבלתם לדו"ח (אל תצרפו את המסלולים עצמם). האם חסכנו בפיתוחים? אם כן, בכמה? הסבירו למה בכלל ציפינו מראש ש- $A^*\varepsilon$ יוכל לחסוך במס' הפיתוחים בתצורה שבה הרצנו אותו.

חלק ט' – מימוש האלג׳ Anytime A* והרצתו

בסעיף זה נממש ווריאציה של אלג׳ Anytime A*. האלג׳ יפעל בצורה הבאה: נריץ את אלג׳ wA* על הבעיה על ערכי w שונים. בכל הרצה של wA* נגביל אותו למס׳ פיתוחים קבוע מראש (המחלקה BestFirstSearch והאלג׳ היורשים ממנה יודעים לקבל ב- constructor שלהם פרמטר אופציונלי בשם max_nr_states_to_expand שעוצר את החיפוש לאחר חריגה ממספר פיתוחים זה). נבצע "חיפוש בינארי" על ערכי $w \in [0.5,1]$ ונחפש את הפתרון הכי טוב מבין הפתרונות המוגבל במס׳ הפיתוחים כאמור (ושאנו מצליחים למצוא במסגרת שיטה זו). כמו בכל חיפוש בינארי, נתחזק גבול תחתון ועליון במהלך החיפוש. הגבול העליון יאותחל להיות 1 והתחתון יהיה 0.5. לאורך החיפוש תישמר האינווריאנטה הבאה: לא נמצא פתרון עבור ערכי w הקטנים או שווים לגבול התחתון (במסגרת הגבלת מס׳ פיתוחים), אך כן נמצא פתרון כזה עבור ערך w של הגבול העליון. בכל איטרציה של החיפוש נריץ את wA* על הבעיה עם ערך w ששווה למחצית הגבול התחתון והעליון ועם מגבלת מס׳ פיתוחים כאמור. נעדכן את הגבולות (בהתאם לקיום או העדר של פתרון) ע"מ לשמור על האינווריאנטה. בכך בכל איטרציה נצמצם את ההפרש בין הגבולות באופן אקספוננציאלי כיאה לחיפוש בינארי. בכל מקרה, נשמור את הפתרון הטוב ביותר שנמצא עד כה ואת הערך w שהוביל איליו. נמשיך כך עד שערכי הגבולות התחתון והעליון יתקרבו זה לזה מספיק.

שימו לב: בכיתה למדתם כלל אצבע לפיו "ככל ש- w קטן יותר כך הפתרון איכותי יותר ומס׳ הפיתוחים גדול יותר". הכלל הנ"ל מצביע על מגמה כללית, אך ציינו בחלקים הקודמים שכלל זה איננו נכון באופן גורף. לכן כשאנו מעדכנים את הגבול התחתון, אין למעשה הבטחה אמתית שעבור כל ערכי w שקטנים מהגבול החדש לא יימצא פתרון העונה על הדרישות. כלומר האלג׳ שלנו לא באמת מוצא ערך w מינימלי שמקיים את האמור, אלא הוא מנסה לקרב אותו ככל הניתן תוך הנחה על המגמה הכללית של הקשר בין w לבין מס׳ הפיתוחים (כלל האצבע).

הערה: ייתכן שהפתרון האופטימלי לאו דווקא הגיע מערך ה- w הקטן ביותר עבורו הרצנו wA* וקיבלנו פתרון. לכן אנו מעדכנים את המשתנה ששומר את הפתרון הטוב ביותר בזהירות (לאחר בדיקה לקיום שיפור באיכות הפתרון).

34. רטוב: השלימו את המימוש של אלג׳ AnytimeA* בקובץ framework/graph_search/anytime_astar.py ע"פ ההוראות המופיעות שם וע"פ ההערות שכתובות בראש המחלקה.
35. רטוב + יבש: השלימו בקובץ main.py את הקוד תחת ההערה הרלוונטית לסעיף זה. צרפו את התוצאות שקיבלתם לדו"ח (אל תצרפו את המסלולים עצמם). הסבירו איך עזר לנו להריץ את הווריאציה הזו של Anytime A* במקרה זה. מה בעצם קיבלנו? שימו לב לגודל הבעיה אותה פתרנו. חזרו לחלק א׳ והיזכרו כמה זמן ייקח למחשב בודד לעבור על כל הסידורים האפשריים.

חלק י' – הגשת המטלה (5 נק')

מעבר למימוש ולדו"ח, ציונכם מורכב גם מהגשה תקינה של המטלה לפי הכללים הבאים:

- **יש לכתוב קוד ברור:**
  - קטעי קוד מסובכים או לא קריאים יש לתעד עם הערות.
  - לתת שמות משמעותיים למשתנים.
- **הדו"ח:**
  - יש לכתוב בדו"ח את תעודות הזהות של **שני** המגישים.
  - הדו"ח צריך להיות מוקלד במחשב ולא בכתב יד. הדו"ח צריך להיות מוגש בפורמט PDF (לא נקבל דוחות שהוגשו בפורמט וורד או אחרים).
  - יש לשמור על סדר וקריאות גם בתוך הדו"ח.
  - אלא אם נכתב אחרת, תשובות ללא נימוק לא יתקבלו.
  - יש לענות על השאלות לפי הסדר ומספרי הסעיפים שלהם.
- **ההגשה:**
  - יש להעלות לאתר קובץ zip בשם AI1_123456789_987654321.zip (עם תעודות הזהות שלכם במקום המספרים).
  - בתוך ה- zip צריכים להיות זה לצד זה:
    - הדו"ח הסופי בפורמט PDF בשם: AI1_123456789_987654321.pdf.
    - תיקיית הקוד ai_hw1 שקיבלתם בתחילת המטלה, עם כל השינויים הנדרשים. **נא לא להכניס ל- zip את התיקייה db שבתתיקייה שקיבלתם – אנא מחקו אותה משם.**

חריגה מהכללים האלו עלולה לגרור הורדה של כל 5 הנקודות.

קוד לא ברור / לא עובד אף עלול להביא להורדה של נקודות נוספות בפרק בו הוא נכתב.

שימו לב: הקוד שלכם ייבדק ע"י מערכת בדיקות אוטומטיות תחת מגבלות זמני ריצה. במידה וחלק מהבדיקות יכשלו (או לא יעצרו תוך זמן סביר), הניקוד עבורן יורד באופן אוטומטי. לא תינתן הזדמנות להגשות חוזרות. אנא דאגו לעקוב בהדיקות אחר הוראות ההגשה. שימו לב כי במהלך חלק מהבדיקות ייתכן שחלק מהקבצים שלכם יוחלפו במימושים שלנו. אם עקבתם אחר כל הדגשים שפורטו במסמך זה - עניין זה לא אמור להוות בעיה.

שימו לב: **העתקות תטופלנה בחומרה.** אנא הימנעו מאי-נעימויות.

מקווים שתהנו מהתרגיל!

בהצלחה!