תרגיל בית 1: שימוש באלגוריתמי חיפוש מיודעים לתכנון מסלולים

שם:

1) **היתם חוא** **206952749**
2) **דורון בן שושן** **305284366**

חלק א' – מבוא

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

אפליקציות ניווט פופולריות (waze, google maps) משתמשות באלגוריתמים לתכנון המסלול החוקי והמהיר ביותר לנסיעה בכבישים, ונעזרות בנתונים קיימים על דרכים, על היסטורית נסיעה בדרכים, ועל נתונים בזמן-אמת.

גלית, סטודנטית חרוצה, נשאבת לאורח החיים המהיר, ורוצה להגיע ממקום למקום במהירות המרבית. גלית היא נהגת מוכשרת ואחראית שנוסעת בכבישים במהירות המקסימלית המותרת והבטוחה. אולם במקרים של עומסי תנועה היא נוסעת במהירות בזמן-אמת המאפשרת נסיעה בטוחה.

חברים של גלית (אתם!) לוקחים הסמסטר את הקורס "מבוא לבינה מלאכותית". גלית מבקשת מכם לעזור לה לתכנן מראש את הדרך המהירה ביותר להגיע מנקודת מוצא ליעד, תוך התחשבות במהירות מקסימלית לנסיעה בכבישים ובנתוני זמן-אמת על מהירות נסיעה ממוצעת בכבישים עם עומסי תנועה.

פורמאליזם – הגדרת נתוני הבעיה

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

לכל קשת $e \in E_{map}$ נתונים אורך הכביש (e_{length}) והמהירות המקסימלית המותרת לנסיעה עליו $(e_{\max\_speed})$. בנוסף, מסופקים נתוני זמן-אמת על המהירות הנוכחית ברשת הכבישים $(e_{current\_speed})$ אשר תלויה בעומסי תנועה ומקיימת $e_{current\_speed} \le e_{\max\_speed}$. כלומר המהירות המקסימלית על כביש היא חסם עליון למהירות הנסיעה האפשרית עליו בזמן נתון .

שימו לב: לכל כביש על המפה נתונים גם הנתונים הסטטיים (מהירות מקסימלית) וגם הנתונים בזמן-אמת (מהירות זמן-אמת). בתרגיל אנחנו מתייחסים לנתונים בזמן-אמת עבור זמן מסוים ללא שינוי במהלך הנסיעה. במציאות, במערכת אמיתית, נצפה שיהיו נתונים לנו רק הנתונים הסטטיים מבעוד-מועד, כאשר המהירות בזמן-אמת עשויה להשתנות תוך כדי נסיעה.

כמו כן, נתונות נקודת מוצא על רשת הכבישים $v_{src} \in V_{map}$ ונקודת יעד $v_{dst} \in V_{map}$.

חלק ב' – הגדרת מרחב החיפוש במפה (יבש 5 נק')

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.

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

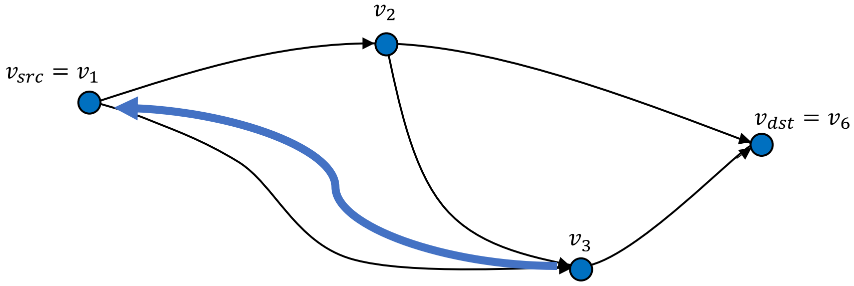
$$SP_{map} \triangleq \langle S_{map}, O_{map}, I_{map}, G_{map} \rangle$$

- **קבוצת המצבים:**
  נרצה לייצג מצב כך שיחזיק את כל המידע שנחוץ לנו עליו במהלך החיפוש במרחב. במקרה המדובר מספיק לשמור את הצומת ברשת הכבישים.
  $$S_{map} \triangleq \{s | s.coordinates \in V_{map}\}$$
  הסבר: לכל מצב s ב- S_{map} יש שדה בודד בשם $coordinates$ שיכול להיות כל נקודה על רשת הכבישים.
- **קבוצת האופרטורים:**
  ניתן לעבור ממצב אחד לעוקב לו בתנאי שקיים כביש מהצומת המיוצג ע"י המצב הראשון לצומת המיוצג ע"י המצב העוקב.
  $$O_{map} \triangleq \left\{o_{map}^{(s_1,s_2)} \middle| s_1, s_2 \in S_{map} \wedge (s_1.coordinates, s_2.coordinates) \in E_{map}\right\}$$
- **עלות אופרטור:**
  נגדיר את פונק' העלות עבור מעבר מצומת דרכים אחד $s_1 \in S_{map}$ לצומת דרכים העוקב שלו $s_2 = o(s_1)$, כאשר $o \in O_{map}$, באופן הבא:
  $$cost_{map}^{time}\left(o_{map}^{(s_1,s_2)}\right) = currentRoadTime\big((s_1.coordinates, s_2.coordinates)\big)$$
- **המצב ההתחלתי:**
  $$I_{map} \triangleq v_{src}$$
- **מצבי המטרה:**
  $$G_{map} \triangleq \{v_{dst}\}$$

תרגילים

לטובת הסעיפים בחלק זה הנח שלאו דווקא קיים פתרון ישיג במרחב.

1. יבש (1 נק'): האם ייתכנו מעגלים במרחב החיפוש שלנו? אם כן תנו דוגמה למעגל כזה, אחרת נמקו. (עד 5 שורות).
   כן, נסתכל על הגודמה מהעיף 4, עם הוספת כביש (קשת) בין v_3 ל v_1
   מעגל: $v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_1$



2. יבש (1 נק'): האם ייתכנו בורות ישיגים מהמצב ההתחלתי שאינם מצבי מטרה במרחב החיפוש? אם כן – איך זה ייתכן? אם לא – למה? (נימוק לכל היותר שורה אחת). תזכורת: בור הינו צומת שאין ממנו קשתות יוצאות.

כן יתכן ויש צומת v ישיג מצומת ההתחלה, ש $d_{out}(v) = 0$, לדוגמא מצב התחלתי שהוא בעצמו בור!

3. יבש (2.5 נק'): הפונקציה $currentRoadTime((s_1.coordinates, s_2.coordinates))$ מחשבת את הזמן המהיר ביותר לנסיעה על הכביש בזמן-אמת לפי נתוני הקשת $(s_1.coordinates, s_2.coordinates)$ המוגדרים בחלק א' של התרגיל.

א. רשמו את הנוסחה לחישוב הפונקציה $currentRoadTime$, המגדירה את עלות האופרטור.

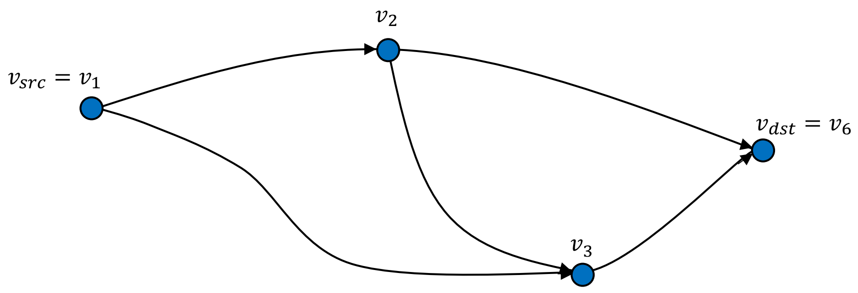
$$e = (s_1.coordinates, s_2.coordinates)$$

$$currentRoadTime = \frac{e_{length}}{e_{current\_speed}}$$

ב. באופן דומה לסעיף א', רשמו נוסחה לחישוב הפונקציה: $scheduledRoadTime((s_1.coordinates, s_2.coordinates))$, אשר מחשבת את זמן הנסיעה המתוכנן אם נוסעים על הכביש במהירות המקסימלית המותרת לנסיעה עליו (max_speed).

$$scheduledRoadTime = \frac{e_{length}}{e_{\max\_speed}}$$

4. יבש (0.5 נק'): נתונה דוגמה לרשת כבישים פשוטה. מלאו את המקום החסר בטבלה החלקית, לפי חישובכם מהסעיף הקודם.



קשת בגרף הכבישים $(s_1.coordinates, s_2.coordinates)$	**אורך הכביש** length $[meter]$	**מהירות נסיעה מקסימלית** max_speed $\left[\frac{meter}{minute}\right]$	**מהירות נסיעה בזמן-אמת** current_speed $\left[\frac{meter}{minute}\right]$	**עלות האופרטור** $cost_{map}^{time}\left(o_{map}^{(s_i,s_j)}\right)$ $[minute]$
(v_{src}, v_2)	100	25	25	4
(v_{src}, v_3)	100	25	20	5

חלק ג' – מתחילים לתכנת (2 נק' יבש)

משימה – הורדת וטעינת קוד התרגיל

5. רטוב: הורידו את *ai_hw1.zip* מהאתר וטענו את התיקייה שבתוכו לסביבת העבודה המועדפת עליכם.
6. רטוב: אם אתם משתמשים ב- IDE לכתיבת והרצת קוד פייתון (אנחנו ממליצים מאוד על VSCode[1] וגם על PyCharm), פתחו פרויקט חדש שתיקיית האם שלו היא התיקייה הראשית של קובץ ה- zip שחולץ (אמור להיות שם קובץ בשם main.py).

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 אוביקט מטיפוס ***StreetsMap***. הטיפוס ***StreetsMap*** יורש מ- *dict*; כלומר ***StreetsMap*** הינו בבסיסו <u>מיפוי</u> ממזהה ייחודי של צומת במפה (מספר שלם) אל אובייקט 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 – המהירות המקסימלית המותרת לנסיעה על הכביש, current_speed – מהירות הנסיעה בזמן-אמת על הכביש.

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 בבדיקות.

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).

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 GraphProblem, GraphProblemState. 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 ממשפחת 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 מממשת את המתודה solve_problem() בהתאם. דוגמאות לאלגוריתמים ממשפחה זו: Uniform Cost Search, 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, בקוד 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

---

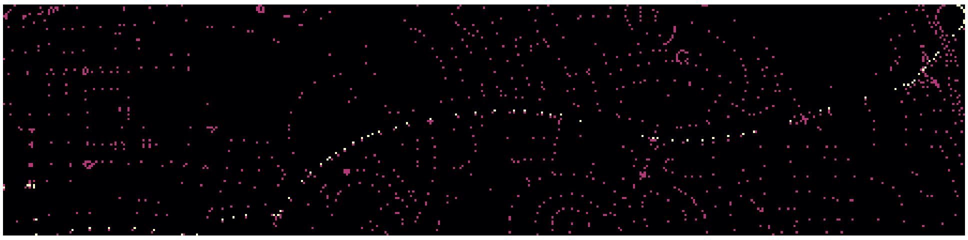
[1] במידה ואתם משתמשים בVSCode, מומלץ להתקין את ההרחבה (extension) בשם "Todo Tree", על מנת לעבור בקלות מסעיף TODO אחד לאחר במהלך התרגיל.

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 בוליאני בשם use_close שקובע האם להשתמש ב- close).

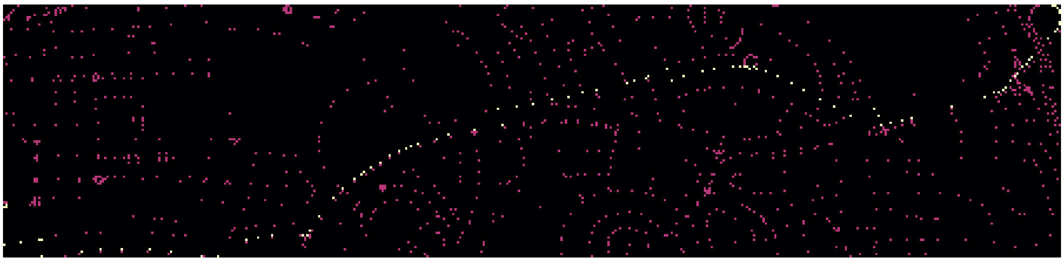
תרגילים

חלק זה של התרגיל נועד על מנת להתחיל להכיר את מבנה הקוד.

7. רטוב: פתחו את הקובץ main.py, קראו את החלק בקוד שמעליו מופיעה הערה המתאימה למספר סעיף זה ומסומנת בTODO. 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. שימו לב כי האלגוריתם מומש עם חלקים חסרים (ואף שגויים), ואתם תשנו את מימושו בהמשך. זאת גם הזדמנות טובה לוודא שהחבילות numpy, scipy, networkx, matplotlib, pandas מותקנות אצלכם כראוי.
8. רטוב: פתחו את הקובץ framework/ways/streets_map.py. השלימו את שתי המשימות בקובץ זה (המסומנות ע"י הערות **TODO** עם סעיף השאלה המתאים, כמו בעוד מקומות רבים לאורך המטלה), אשר מחשבות את זמן הנסיעה לכביש (זמן מתוכנן 'scheduled_time' לפי max_speed וזמן-אמת 'current_time' לפי current_speed), כפי שחישבתם בנוסחה בסעיף 3 בתרגיל זה.
שימו לב כי על מנת להכיר את הקוד נשתמש בכמה פונקציות שונות לחישוב עלות המסלול: על בסיס מרחק 'distance', על בסיס מהירות זמן אמת 'current_time' ועל בסיס המהירות המתוכננת 'scheduled_time'.
<u>הערה:</u> ההערה בTODO עבור הפונקציה compute_scheduled_time שונתה בקבצי הקוד.
9. רטוב: פתחו את הקובץ problems/map_problem.py. בתוכו יש לכם שתי משימות התואמות לסעיף זה. 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.
10. רטוב: עיינו במימוש של המחלקות בקובץ זה. וודאו שאתם מבינים את החלקים השונים. שימו לב שמחלקה זו יורשת מהמחלקה GraphProblem (שתוארה מקודם) וממשת את המתודות האבסטרקטיות הנדרשות.
11. רטוב + יבש (1 נק'): עתה, לאחר תיקון קוד המחלקה MapProblem, הריצו בשנית את main.py, וצרפו את תמונת המסלול שנוצרה בנתיב images/ UCS_path_distance_based.png



12. רטוב: השלימו את המשימה בקובץ main.py, אשר מתאימה לסעיף זה (ומסומנת אף היא בTODO).
13. רטוב + יבש (1 נק'): הריצו שוב את main.py, וצרפו את תמונת המסלול שנוצרה בנתיב images/UCS_path_time_based.png



14. ודאו שעבור ההרצות בשני הסעיפים האחרונים קיבלתם את התוצאות הבאות:

Task num.	|path|	total_g_cost	|space|	#dev
[11]	91	5040.58286	11230	11101
[13]	109	5.99946	11235	11099

חלק ד' – אלגוריתם *A (יבש 14 נק')

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 ג'). 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

15. רטוב: השלימו את המשימות הדרושות תחת הערות ה- **TODO** בקובץ framework/graph_search/astar.py כך שנקבל מימוש תקין לאלגוריתם *Weighted A כפי שלמדנו בקורס, כאשר הצומת הבא נבחר לפי: $f(v) = (1-w) \cdot g(v) + w \cdot h(v)$, כאשר $h(v)$ הוא הערך היוריסטי ו-$g(v)$ הוא ערך עלות המסלול לצומת v. 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 ומהתרגול להבדלים בין אלג' UniformCost לבין אלג' *A.
16. 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 ביוריסטיקה. ה- constructor של המחלקה AStar מקבל את טיפוס היוריסטיקה 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 הכלל). וודאו שהתוצאה המודפסת זהה לזו שקיבלתם בעזרת Uniform Cost.
17. יבש (9 נק'): כפי שראינו בהרצאות ובתרגולים, יוריסטיקה פשוטה לבעיית המפה היא מרחק אווירי לפתרון. בתרגיל שלנו הפרמטר המעניין הוא **זמן** הנסיעה ולא המרחק. שימו לב כי מתכנון נסיעה אחד לאחר מהירות זמן-אמת בכל כביש עשויה להשתנות, אולם המהירות המקסימלית קבועה.

   א. (3 נק') בהינתן חסם תחתון וחסם עליון למהירות המקסימלית המותרת, אשר מחושבים באופן הבא:

$$MAX\_ROADS\_SPEED = \max_{e}(e_{\max\_speed}),\ MIN\_ROADS\_SPEED = \min_{e}(e_{\max\_speed}),\ \ \forall e \in E_{map}$$

   הציעו נוסחה לחישוב יוריסטיקה **קבילה** המתאימה לבעיית זמן הנסיעה, אשר תלויה במרחק האווירי ובאחד החסמים על המהירות המקסימלית.

$$\mathrm{h(state)} = \frac{air\ distance(state)}{MAX\_ROADS\_SPEED}$$

   ב. (4 נק') הוכיחו כי היוריסטיקה שהגדרתם בסעיף א' קבילה.

נניח כי זמן הנסיעה הנכון מ $STATE_X$ ל TARGET הוא $h^*(STATE\_X)$ ונראה כי

$$\forall STATE_X \in map: 0 \le h(STATE_X) \le h^*(STATE_X)$$

נניח בשלילה שזה לא נכון, כלומר קיימת $STATE_Y$ כך ש $h(STATE_Y) > h^*(STATE_Y)$ ($h(STATE_Y) \ge 0$ מאופן ההגדרה מתקיים כי: (h מאופן הגדרת

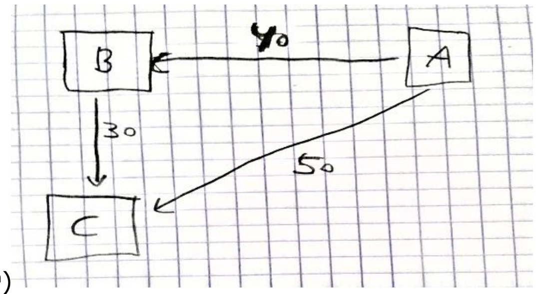
$$\leftarrow average\_speed = \frac{air\ distance(STATE_Y)}{h^*(STATE_Y)} > MAX\_ROADS\_SPEED \leftarrow \frac{air\ distance(STATE_Y)}{MAX\_ROADS\_SPEED} > h^*(STATE_Y)$$

סתירה כי $average_{speed} \le MAX\_ROADS\_SPEED$, ולכן $\forall STATE \in map: 0 \le h(STATE_X) \le h^*(STATE_X)$

ולכן h(state) קבילה

ג. (2 נק׳) האם היוריסטיקה עדיין תהיה קבילה במידה ונשתמש במרחק מנהטן במקום במרחק האווירי בנוסחה שהגדרתם בסעיף א?

לא, דוגמא נגדית: נניח כי לכל קשת (street) המהירות המקסימלית = מהירות מינימלית = 10[km/h]



(יחידות המרחק בKM)

מרחק מנהטן: 30+40 = 70 ← זמן מנהטן: 7 [h] (כאשר $(\mathrm{h(state)} = \frac{manhattan\ distance(state)}{MAX\_ROADS\_SPEED}$

מרחק אופטימלי: 50 (דרך A→C) ← זמן אופטימלי: 5 [h]

$$7 = h(C) > h^*(C) = 5$$

18. רטוב: היכנסו לקובץ problems/map_heuristics.py וממשו את היוריסטיקה שהגדרתם בסעיף הקודם במחלקה TimeBasedAirDistHeuristic (מלאו את המקומות החסרים תחת ההערות שהשארנו לכם שם). כעת הריצו שוב את הבעיה שפתרתם בסעיף הקודם, אך כעת בעזרת היוריסטיקה (מלאו ב- main.py את המשימות שקשורות לסעיף זה).

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.

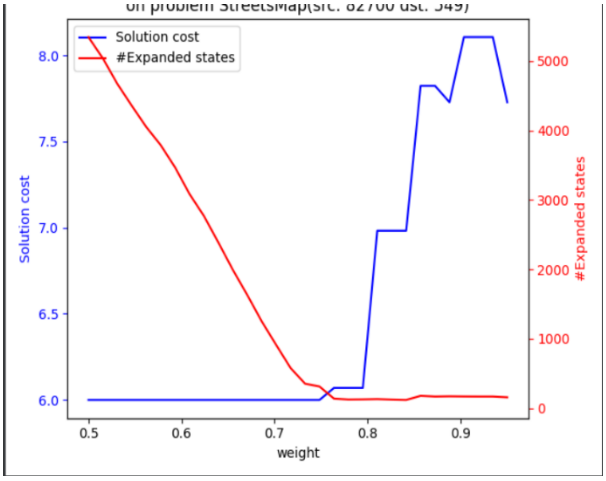
19. יבש (1 נק׳): כתבו בדו״ח את מס׳ פיתוחי המצבים ה**יחסי** שחסכנו בריצה בסעיף קודם לעומת הריצה העיוורת (ההפרש חלקי מס׳ הפיתוחים בריצה **בלי** היוריסטיקה). את מספר פיתוחי המצבים תוכלו לראות ע״י המזהה DEV# בשורה המודפסת המתארת את פתרון בעיית החיפוש, כפי שראיתם בסעיף 7.

$$\#state\ save\ ratio: \frac{11099 - 5343}{11099} = 51.86\%$$

20. רטוב: 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 ביוריסטיקה הנתונה ועם n משקולות שונות בתחום הסגור [0.5,0.95]. 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 (מצוין ע״י TODO בקוד עם מספר הסעיף של שאלה זו) ע״מ ליצור את הגרף המתאים עבור פתרון בעיית המפה תוך שימוש ביוריסטיקה AirDistHeuristic.

21. יבש (4 נק'): a) צרפו לדו"ח את הגרף שנוצר בריצה מהסעיף הקודם. הסבירו את הגרף שהתקבל:

ציינו אילו אזורים בגרף הם יותר כדאיים ואילו פחות – הסבירו למה (עד 2 שורות).



במקרה זה, בתחום $0.5 \le weight \le 0.74$ מקבלים את הפתרון האופטימלי, אך משתמשים בלא מעט זיכרון, ב0.5 משתמשים פי ~14 זיכרון מ0.7 ולכן, יש tradeoff בין הזיכרון לאופטימליות הפתרון, וזמן מציאת הפתרון, ולכן האזור שכדאי לעבוד בו הוא $0.7 \le weight \le 0.8$

b) 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? (תשובה עד 4 שורות).

אם נסתכל על התחום $0.5 \le weight \le 0.8$ ככל ש w קטן אנו מקבלים פתרון יותר איכותי (או אותו פתרון אופטימלי), ומספר הפיתוחים גדל, אך זה לא דווקא נכון, אם נסתכל על התחום $0.85 \le weight \le 0.88$ מתקיים כי אם נקטין את w נקבל פתרון פחות טוב, ומספר הפיתוחים גדל.

חלק ה' – חישוב יוריסטיקה (יבש 18 נק')

בחלק הקודם של התרגיל השתמשנו ביוריסטיקה המבוססת על חסמי מהירות הנסיעה הגלובליים ועל מרחק אווירי בין צמתים. בחלק זה נראה איך אפשר להשתמש בנתונים לוקאליים, נתכנן פונקציות יוריסטיקה נוספות ונראה איך השימוש בהן משפיע על ביצועי אלגוריתם A*.

תרגילים

22. יבש (5 נק'): בסעיף זה נשתמש בגרף המסלולים הקלים ביותר כשלב מקדים לאלגוריתם החיפוש A* וכאמצעי לחישוב יוריסטיקה.
ראשית, נחשב את גרף המסלולים הקלים ביותר כאשר משקל כל קשת מתבסס על נתוני $scheduledRoadTime$ (כפי שמוגדר בסעיף 3). נגדיר את היוריסטיקה באופן הבא:
$$ShortestPathsBasedHeuristic(v_{src}) = \sum_{e \in shortestPath(v_{src}, v_{dst})} scheduledRoadTime(e)$$
הוכח/הפרך: היוריסטיקה שהגדרנו קבילה.

נראה באינדוקציה שעל אורך המסלול הקל ביותר מ v_{src} ל v_{dst} , עבור n=0, מתקיים כי $v_{src} = v_{dst}$ ולכן מתקיים עבורו $0 \le ShortestPathsBasedHeuristic(v_{src}) = 0 \le h^*(v_{src})$

נניח נכונות עבור n ונראה עבור n+1: נסמן המסלול $v_{src}\overrightarrow{e_1}v_2\overrightarrow{e_2}\dots.\overrightarrow{e_{n+1}}v_{dst}$, ואז מתקיים כי:

$$\begin{aligned} ShortestPathsBasedHeuristic(v_{src}) &= \sum_{e \in shortestPath(v_{src}, v_{dst})} scheduledRoadTime(e) \\ &= scheduledRoadTime(e_1) + \sum_{e \in shortestPath(v_1, v_{dst})} scheduledRoadTime(e) \\ &\le currentRoadTime(e_1) + h^*(v_2) = h^*(v_{src}) \end{aligned}$$

- אי שוויון נובע המנחת האינדיקציה, ומכך שכל תת-מסלול של מסלול קל ביותר הוא כל ביותר
- שוויון האחרון נובע מאי שוויון המשולש, ומהגדרת $h^*(v)$

וזה נכון לכל v_{src} ו v_{dst}, ולכן $ShortestPathsBasedHeuristic(v)$ קבילה

23. יבש (2 נק'): איזו יוריסטיקה מיודעת יותר מבין השתיים: TimeBasedAirDistHeuristic או ShortestPathsBasedHeuristic? נמק את תשובתך (מקסימום 2 שורות).

היוריסטיקה $ShortestPathsBasedHeuristic$ מיודעת יותר. זאת כיוון שהיא לוקחת בחשבון את אורכי הכבישים בפועל, בעוד שהשנייה מתייחסת למרחק אווירי. סדר הגדלים בין דרכים במרחק אווירי, יכול להיות שונה מאורכם בפועל.

24. רטוב: מכיוון שחישוב גרף המסלולים הקלים ביותר לכל צומת עשוי לארוך זמן רב, חישבנו עבורכם את מחיר המסלולים הקלים ביותר למספר יעדים מצומצם, כאשר מחיר כל קשת e מתבסס על $scheduledRoadTime(e)$.
מלאו את המשימה הרלוונטית לסעיף זה בקובץ: main.py
היעזרו בפונקציה set_additional_shortest_paths_based_data בקובץ problems/map_problem.py
25. רטוב: כעת נשתמש בנתונים על היסטוריית נסיעות. נניח כי קיים מאגר מידע המכיל את היסטוריית זמני הנסיעה בשעה מסוימת ביממה בארבעת הימים האחרונים, מכל צומת לכל צומת אחר. על בסיס נתונים אלה, נרצה לחשב את הזמן הצפוי לנסיעה היום באותה השעה.
את היוריסטיקה נחשב כממוצע זמני הנסיעה בארבעת הימים האחרונים, באופן הבא:
$$HistoryBasedHeuristic(v_{src}) = \frac{1}{4} \sum_{i \in \{1,2,3,4\}} \sum_{\substack{e \in \\ shortestPath(v_{src}, v_{dst})}} currentRoadTime(e)$$
שימו לב: במקרה זה משקל הקשתות לפיו מחושב המסלול הקל ביותר מתייחס ל$currentRoadTime$ (כפי שמוגדר בסעיף 3).
מלאו את המשימות הרלוונטיות לסעיף זה בשני הקבצים: main.py, problems/map_problem.py
26. יבש (4 נק'): תנו דוגמה למקרה בו היוריסטיקה $HistoryBasedHeuristic$ קבילה ודוגמה למקרה בו היא לא קבילה.

דוגמה קבילה: זמני הנסיעה קבועים לכל שעה ולא משתנים מיום ליום. במקרה כזה, המסלול שהיה הקצר ביותר על פי ממוצע ארבעת הימים האחרונים, יהיה הקצר ביותר גם הלאה ולכן המסלול שימצא יהיה אופטימלי.

דוגמה לא קבילה: נניח שכל ארבעה ימים זמני משתנים כך-עבור כל כביש e וזמן נסיעה t, כל ארבעה ימים הזמן נסיעה הופך ל-t/1. כלומר, סדר הכבישים מתהפך. אם בארבעת הימים האחרונים קיבלנו מסלול מסויים מהיוריסטיקה, הוא יהיה למעשה המסלול הגרוע ביותר לבחירה ולכן היוריסטיקה לא תהיה קבילה.

27. יבש (7 נק'):

<u>הגדרה</u>: יוריסטיקה h נקראת ε-קבילה אם עבור $\varepsilon > 0$ מתקיים כי $h(n) \le h^*(n) + \varepsilon$ לכל מצב n.

א. (יבש 1 נק'): האם אלגוריתם A^* המשתמש ביוריסטיקה ε-קבילה הוא שלם? הסבירו בקצרה, ובמקסימום 5 שורות (אין צורך להוכיח).

שלם (בהנחה שמחיר הקשתות חסום מלמטה ע"י $\delta > 0$)

הסבר: בעצם, גודל ה ε מאפשר "להטעות" את החיפוש, אבל למרות זאת, האלגוריתם לא יתקע בלולאה (בגלל שמחיר הקשתות חסום מלמטה), ולכן הוא יחזור וימצא פתרון. (במקרה הקיצוני בויתר ש $\varepsilon \gg g(v)$ נקבל אלגוריתם UCS שהוא בעצמו שלם)

אם מחיר הקשתות לא חסום, אז האלגוריתם לא שלם, מאותם שיקולים מההרצאה.

ב. (יבש 3 נק'): נסמן ב- C^* את עלות המסלול האופטימלי מהמצב ההתחלתי למצב המטרה. מהו מחיר המסלול הגבוה ביותר (הגרוע ביותר) אשר A^* המשתמש ביוריסטיקה ε-קבילה יכול להחזיר? הוכח את תשובתך.

מחיר המסלול הגרוע ביותר שיכול להיות מוחזר הוא. $C^* + \varepsilon$

הוכחה: נניח בשלילה שהאלגוריתם החזיר מחיר מסלול p שמקיים $h(p) > C* + \varepsilon$. כלומר, פונקציה g של צומת המטרה גדולה גם היא מ-$C* + \varepsilon$ ומכאן גם סכום משקלי הקשתות על המסלול p. בהכרח קיים צומת n במסלול p עבורו $h(n)>C^*$ כיוון שאם לא קיים אזי היוריסטיקה לא e קבילה. עבור צומת זה, בהכרח היה קיים בתור הצמתים צומת s בעל ערך יוריסטי נמוך יותר, זאת כיוון שמהגדרת היוריסטיקה כ-e קבילה ישנו מסלול שקרוב ל-C^* עד כדי e. לפי אופן פעולת A^* היה נבחר צומת s לפיתוח ולא צומת n, בסתירה להנחה.

ג. (יבש 3 נק'): עבור סעיף זה בלבד נניח כי היוריסטיקה $HistoryBasedHeuristic$ היא ε-קבילה, עבור $\varepsilon = 3$. בנוסף, נניח כי לכל מצב n נתון $TimeBasedAirDistHeuristic(n)$ וגם $ShortestPathsBasedHeuristic(n)$. הצע יוריסטיקה קבילה ומיודעת ככל האפשר על בסיס נתונים אלו והיוריסטיקה $HistoryBasedHeuristic$.

לפי סעיף 23 מצאנו ש$ShortestPathsBasedHeuristic$ מיודעת יותר מ- $TimeBasedAirDistHeuristic$. ולכן נשתמש ב $ShortestPathsBasedHeuristic$ בכל מקום שהיה אפשר להשתמש ב $TimeBasedAirDistHeuri$

נגדיר:

$$h(v) = \begin{cases} ShortestPathsBasedHeuristic(v) & if\big(HistoryBasedHeuristic(v) - 3 \le ShortestPathsBasedHeuristic(v)\big) \\ HistoryBasedHeuristic(v) - 3 & else \end{cases}$$

קבילות:

מקרה 1: $h(v) = ShortestPathsBasedHeuristic(v) \le h^*(v)$

מקרה 2: $h(v) = HistoryBasedHeuristic(v) - 3 \le h^*(v) + 3 - 3 = h^*(v)$ (אי שוויון נובע מהגדרה3קבילה)

מיודעת יותר מכולן: נובע באופן ישיר מן ההגדרה

חלק ו' – מימוש האלג' $A^*\varepsilon$ והרצתו (יבש 2 נק')

28. רטוב: ממשו את החלקים החסרים של אלג' $A^*\varepsilon$ בקובץ framework/graph_search/astar_epsilon.py ע"פ ההנחיות המופיעות שם.
29. רטוב: מימשנו יוריסטיקה קבילה ויוריסטיקה לא קבילה אך מיודעת יותר במקרים מסוימים. הבעיה היא שאין לנו אף הבטחה על איכות הפתרון שמניב *A עם יוריסטיקה שאינה קבילה. נרצה לנצל את הבטחת איכות הפתרון של $A^*\varepsilon$ כדי לעשות שימוש מועיל ביוריסטיקה שאינה קבילה במטרה לחסוך במספר הפיתוחים מבלי לפגוע באופן דרסטי באיכות הפתרון. השלימו בקובץ main.py את הקוד תחת ההערה הרלוונטית לסעיף זה.
30. יבש (2 נק'): צרפו לדו"ח את התוצאות שקיבלתם בסעיף הקודם (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 (לא מספיק לטעון ש- $A^*\varepsilon$ גמיש יותר בבחירה של הצומת הבא לפיתוח). נסו להסביר למה בעצם אנחנו מצפים שהגמישות הזאת של $A^*\varepsilon$ אכן תעזור לנו במקרה הזה לבחור מ- open צומת לפיתוח שיקדם אותנו מהר יותר למטרה. מה בעצם הוספנו לאלג' החיפוש? תשובה עד 2 שורות.
time: 1.46 #dev: 2705 |space|: 2606 total_g_cost: 5.99946 |path|: 109

כן, חסכנו $\frac{2755-2705}{2755} = 1.8\%$ בפיתוחים, הסיבה היא שכאן השתמשנו ביוריסטיקה מיודעת יותר, בנוסף לכך שאפשרנו טווח יותר גדול של גמישות, שבתורו עלה על זה שהיוריסטיקה לא קבילה במקרים מסוימים

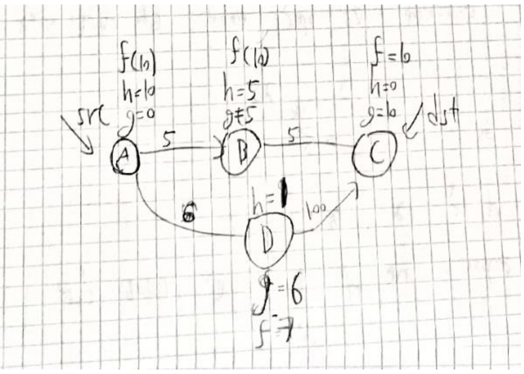
חלק ז׳ – האלג׳ *IDA (יבש 8 נק׳)

ראינו בכיתה את האלגוריתם *IDA אשר משתמש בהעמקה הדרגתית, ומטרתו לשפר את ביצועי אלג׳ *A.

31. יבש (1 נק׳): באיזה אופן אלג׳ *IDA יכול לשפר את ביצועי אלג׳ *A?

אלגוריתם *IDA חוסך בזיכרון (לעומת *A), בכך שהוא מחזיק רשימה מצומצמת(לפי חסם) של צמתי OPEN, ומתקדם כמו DFS

32. יבש (3 נק׳): עבור מרחב המצבים שהגדרנו בתרגיל בית זה, תנו דוגמה לכך שאלג׳ *IDA עדיף על פני *A מבחינת המדד אותו ציינתם בסעיף הקודם. השתמשו ביוריסטיקה TimeBasedAirDistHeuristic שהגדרנו בתרגיל.



עבור *A סדר הפיתוח: A→B→D→C

עבור *IDA סדר הפיתוח כמו DFS עד שנגיע ל f_limit שבמקרה זה הוא מוגדר כ 10 ((h(src)) :

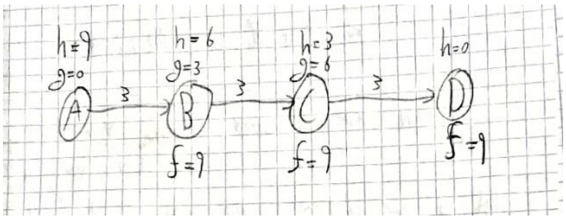
A→B→C

אלגוריתם *IDA חסך בזיכרון לעומת *A

33. יבש (1 נק׳): איזה מדד ביצועים עלול להיפגע ב*IDA לעומת *A?

מדד הזמן, נאלצים לחזור על אותו מסלול חיפוש מספר פעמים עד שנמצא את המסלול הדרוש.

34. יבש (3 נק׳): עבור מרחב המצבים שהגדרנו בתרגיל בית זה, תנו דוגמה בה אלג׳ *IDA **טוב כמו** *A מבחינת המדד אותו ציינתם בסעיף הקודם. השתמשו ביוריסטיקה TimeBasedAirDistHeuristic שהגדרנו בתרגיל.



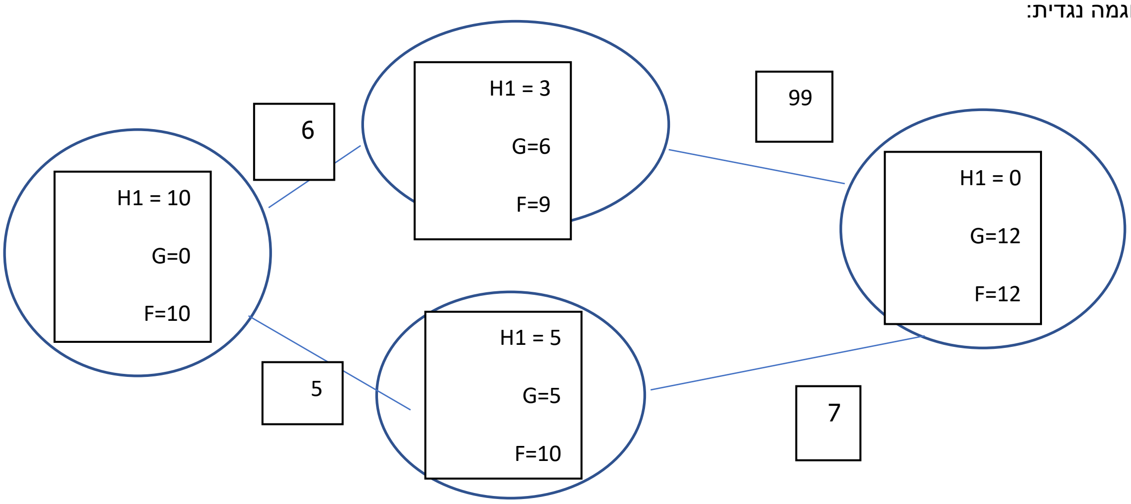
אלגוריתם *IDA יפתח את אותם צמתים ש *A מפתח, ללא חזרה על הצמתים (איטירציות של *IDA),

כי במקרה זה f_limit = h(src) = 9 = מחיר המסלול האופטימלי.

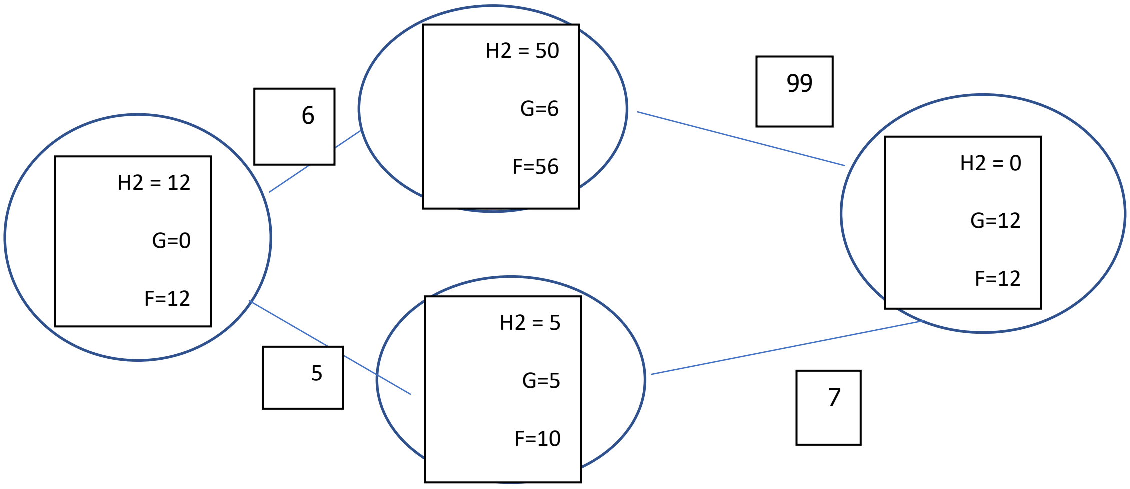
חלק ח' – שאלות נוספות (יבש 11 נק')

35. יבש (3 נק'): נתונות יוריסטיקות קבילות h_1, h_2 עבור בעיית חיפוש כלשהי, אשר מקיימות: $h_1 \leq h_2$.
הוכח/הפרך: כל צומת n אשר מפותח ע"י A^* עם היוריסטיקה h_1 בהכרח יפותח ע"י A^* עם היוריסטיקה h_2.

דוגמה נגדית:



הנתיב שיפותח הוא: מצב התחלה -> צומת עליון->צומת תחתון->צומת מטרה.



יפותח צומת התחלה->צומת תחתון->צומת מטרה.

ברור כי שתי היוריסטיקות קבילות, וH1 =< H2 לכל צומת.

הצומת התחתון יפותח לפי h1 אבל לא לפי h2.

36. יבש (8 נק'): נתונה פונקציה $k(n)$ המוגדרת לכל מצב n במרחב החיפוש ומהווה חסם עליון ליוריסטיקה המושלמת. כלומר: $k(n) \geq h^*(n), \forall n$. בשאלה זו נעסוק באלגוריתם A^* אשר משתמש ביוריסטיקה קבילה h, וכפי שלמדנו מוציא צומת מהתור OPEN לפי סכום מינימלי של $g + h$. בתשובותיכם לשני הסעיפים הבאים התייחסו לפרמטרים g, h, k עבור המצב n, ו/או מצב כלשהו m שיתואר בסעיפים אלו.
בנקודה באלגוריתם בה מכניסים מצב n אל OPEN:

א. יבש (4 נק'): תאר תנאי המשתמש בפונקציה k אשר בודק האם ניתן להסיר מצב מתוך התור OPEN, כך ש-A^* עדיין יחזיר פתרון אופטימלי.

נסיר מהתור צומת n שמקיים: $h(n) > k(n)$ כיוון שנתון שהיוריסטיקה hקבילה, מתקיים h(n)<=h*(n). לכן אם היוריסטיקה גדולה מ-k היא בהכרח גדולה ממש מהפתרון האופטימלי ולכן הצומת בהכרח לא חלק מהמסלול האופטימלי למטרה

ב. יבש (4 נק'): כעת אנו מאפשרים להחזיר פתרונות לא אופטימליים.
נניח כי הדרישה היא להחזיר פתרון שמחירו נמוך מסף $thresh$. כלומר, עבור מצב מטרה כלשהו נדרוש להחזיר מסלול בעל עלות C, כאשר $C \leq thresh$ (אם קיים פתרון).
לאחר הכנסת המצב n לתור OPEN, האם ניתן להסיר מצבים מתוך התור ולהבטיח כי הפתרון המוחזר יהיה לכל היותר $thresh$? נמקו. אם תשובתכם היא כן, ציינו מה התנאי להסרת מצב מהתור.

לכל צומת v בתור open נסיר אותו אם הוא מקיים $h(v) + g(v) > thresh$ כי h קביל, ולכן ערך הפתרון שעובר דרך צומת זה הוא $h(v) + g(v)$ לפחות, שזה ערך לא רצוי

(כי $thresh < g(v) + h(v) \leq g(v) + h^*(v) \leq g(v) + \sum_{e \text{ is in the path from } v \text{ to } dst} cost(e)$ = f(ds)

חלק ח' – הגשת המטלה

- **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 שבתיקייה שקיבלתם – אנא מחקו אותה משם.

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 באדיקות 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.

לא תתאפשרנה הגשות חוזרות, גם לא בגלל טעות טכנית קטנה ככל שתהיה. אחריותכם לוודא טרם ההגשה שהתרגיל רץ בסביבה שהגדרנו ושהקוד עומד בכל הדרישות שפירטנו.

אנא עברו בשנית על ההערות שפורסמו בתחילת מסמך זה. וודאו שאתם עומדים בהם.

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

מקווים שתיהנו מהתרגיל!

בהצלחה!