

# תרגיל בית 1

## מנהלות

### הנחיות כלליות

**שימו לב! חריגה מהנהלים ללא אישור מיוחד תגרור ציון 0 בתרגיל.**

- תאריך הגשה: 16.12.21. מותר לאחר בהגשת התרגיל בתמורה לקנס, ע"פ נוהל איחורים שמפורט באתר הקורס.
- יש להגיש את פתרון התרגיל בזוגות בלבד. ניתן לבקש אישור להגשה ביחידים.
- פתרון התרגיל חייב להיות מוקלד, ובעברית בלבד. ניתן לבקש אישור להגשה באנגלית.
- ניתן לשלוח שאלות בנוגע לתרגיל לטל סויסה ([talswisa@campus.technion.ac.il](mailto:talswisa@campus.technion.ac.il)) עם הכותרת AI\_HW1, רק לאחר שבדקתם את ה-FAQ שבאתר. אין לשלוח שאלות לתיבת המייל הקורסית.
- ניתן לשלוח בקשות דחיה **מוצדקות** לטל סויסה.
- ייתכן שיהיו שינויים במהלך התרגיל. כל העדכונים מחייבים - יש להתעדכן ב-FAQ שבאתר הקורס.
- העתקות בחלק הרטוב והיבש של התרגיל ייבדקו, ויטופלו בחומרה.
- הציון בתרגיל יורכב משאלות יבשות וחלקים רטובים:
- **דגשים לחלק היבש:** מעבר לתשובות הנכונות, אתם נבחנים גם על הצגת הנתונים והתוצאות בצורה קריאה ומסודרת במקומות בהם התבקשתם לכך.
- **דגשים לחלק הרטוב:** 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 בין היתר את המסלול המתקבל, את עלותו ואת מס' הפיתוחים. הבדיקות יהיו כמובן מוגבלות בזמן ריצה. יינתן לכם זמן סביר ביותר להרצת כל טסט. אם תעקבו אחר ההוראות במסמך זה ובקוד אין סיבה שלא תעמדו בזמנים אלו.

### הנחיות טכניות

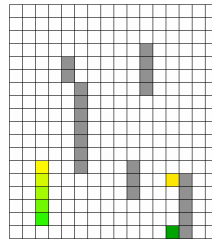
- גרסת python איתה אתם נדרשים לעבוד הינה 3.8. גם קבצי המקור שקיבלתם מתאימים לגרסה זו. לא לאחר הגשת התרגיל לא יתאפשר לתקן טעויות הנובעות משימוש בגרסת פייתון אחרת.
- אין לשנות קוד שלא התבקשתם לשנות.
- יש לכתוב את המימוש אך ורק במקומות המסומנים.
- אין ליצור קבצי קוד חדשים.
- אין לשנות פקודות import.
- לצורך התרגיל תצטרכו להתקין את החבילות, numpy, pandas, matplotlib ו-heapdict (לא HeapDict).
- התשובות לסעיפים בהם מופיע הסימון 📌 צריכים להופיע בדוח.
- בכל השאלות היבשות עליכם להסביר את תשובותיכם.
- כל השינויים בתרגיל מאז הגרסה הראשונה שפורסמה מסומנים עם **מרקר כתום**.

## 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.

### תיאור הבעיה

הבעיה שנתמחד איתה בתרגיל הינה מציאת מסלול במבוך מנקודת התחלה לנקודת יעד המוגדרות במבוך, עבור רובוט בצורת מלבן. בתמונה הבאה מתוארת בעיה לדוגמה:



הרובוט מופיע משמאל למטה, בצבעי צהוב-ירוק בהירים. הקצה הימני של הרובוט הוא ראש הרובוט והקצה הצהוב הוא זנב הרובוט. המטרה היא למצוא מסלול שיביא את זנב הרובוט לנקודה הצהובה שמופיעה מימין (נקודת היעד של זנב הרובוט), ואת הראש לנקודה הימנית (נקודת היעד של ראש הרובוט). הרובוט יכול להתקדם בכיוון הראש שלו, ולפנות ימינה או שמאלה, כאשר ציר הסיבוב הוא מרכז הרובוט. הנקודות האפורות על המפה הן קירות. במהלך התנועה של הרובוט במבוך, אסור לו להתנגש בקיר. בהמשך התרגיל נקרא לבעיות מסוג זה "בעיות מבוך".

### פירמול הבעיה

#### ייצוג מצב

מצב מיוצג על ידי:

1. מטריצה שמייצגת את מפת המבוך.
2. מיקומו הנוכחי של הרובוט על המפה.

במטריצה, משבצת ריקה תסומן ב- 0 ומשבצת שהיא קיר תסומן ב- 1. למשל המטריצה הבאה יכולה לייצג מפת מבוך:

	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	0	0	-1	0
3	0	0	0	0
4	0	0	0	0

מיקומו של הרובוט מיוצג על ידי מיקום הראש ומיקום הזנב של הרובוט - אינדקסים של שתי כניסות במטריצה. למשל, במפת המבוך שבדוגמה, נוכל להגדיר את מיקום הראש של הרובוט להיות בכניסה ה- (3, 2), ואת מיקום הזנב להיות בכניסה ה- (3, 0).

בעיית מבוך ניתנת להגדרה על ידי המצב ההתחלתי שלה ועל ידי מיקומי היעד של ראש וזנב הרובוט. לשם נוחות נוכל לתאר לעצמנו את בעיית המבוך על ידי מטריצה שבה במיקום בו נמצא זנב הרובוט יש 1, במיקום של הראש יש 2, במיקום היעד של הזנב 3 ובמיקום היעד של הראש 4. אם נגדיר את מיקום היעד של הזנב והראש להיות (1, 1) ו- (1, 3) בהתאמה, נקבל את המטריצה הבאה:

	0	1	2	3
0	0	0	0	0
1	3	0	4	0
2	0	0	-1	0
3	1	0	2	0
4	0	0	0	0

נסמן את הבעיה המיוצגת ע"י המטריצה הזו ב-  $M_{example}$ .

שימו לב שלא כל בעיות המבוך ניתנות לתיאור בדרך זו, מכיוון שיייתכן למשל שמיקומו ההתחלתי של ראש הרובוט זהה למיקום היעד של זנב הרובוט. לכן ייצוג זה של בעיית המבוך אינו טוב מספיק, ואנחנו נשתמש בו רק לצרכי הסברת הבעיה (ויצרת מפות מבוך, בהמשך).

לכניסות המטריצה שבין הראש לזנב נקרא גוף הרובוט. בכל מצב תקין גוף הרובוט נמצא על משבצות ריקות (שערכן 0). לכל אורך התרגיל הרובוט יהיה בצורת מלבן בעובי 1, ובאורך אי זוגי, 3 לפחות.

### אפורטורים

התקדמות - הראש והזנב של הרובוט יתקדמו משבצת אחת לכיוון שאליו הרובוט "מסתכל" (בהמשך לקו מהזנב לראש של הרובוט). הפעולה אפשרית רק אם היא לא מוציאה את הרובוט מחוץ למבוך, ואם הראש של הרובוט לא עולה על משבצת שיש בה קיר.

לאחר התקדמות קדימה מהמצב המיוצג על ידי  $M_{example}$ , יתקבל המצב:

	0	1	2	3
0	0	0	0	0
1	3	0	4	0
2	0	0	-1	0
3	0	1	0	2
4	0	0	0	0

סיבוב ימינה - הרובוט יסתובב ימינה ביחס לכיוון אליו הוא מסתכל, כאשר הסיבוב יתבצע סביב נקודת האמצע של הרובוט. הפעולה אפשרית רק אם במהלך הסיבוב הרובוט לא מתנגש עם ראשו, זנבו או גופו בקירות, ולא יוצא מהמבוך. למשל, כדי שהרובוט יוכל לבצע סיבוב ימינה מהמצב המיוצג על ידי  $M_{example}$ , התאים הוורודים צריכים להיות פנויים.

	0	1	2	3
0	0	0	0	0
1	3	0	4	0
2	0	0	-1	0
3	1	0	2	0
4	0	0	0	0

לאחר סיבוב ימינה יתקבל המצב הבא:

	0	1	2	3
0	0	0	0	0
1	3	0	4	0
2	0	1	-1	0
3	0	0	0	0
4	0	2	0	0

סיבוב שמאלה - כמו סיבוב ימינה, רק שמאלה. לא ניתן לבצע סיבוב שמאלה מהמצב המיוצג על ידי הטבלה הראשונה, בגלל שהקיר מתנגש עם ראש הרובוט במהלך הסיבוב.

משימה 1 (2 נק')

👉 האם קיים פתרון לבעיית המבוך המיוצגת ע"י  $M_{example}$ ?  
אם כן, רשמו את סדרת פעולות של פתרון אופטימלי, ואת מחיר הפתרון. אם לא, הסבירו מדוע.

משימה 2 (2 נק')

👉 האם ייתכנו מעגלים בגרף המצבים של בעיית מבוך כלשהי?  
אם כן, תנו דוגמה למצב ורצף אופרטורים שיוצרים מעגל. אם לא, הסבירו מדוע.

משימה 3 (4 נק')

👉 האם בכל מרחב חיפוש ניתן להגיע לבור?  
אם כן, הוכיחו זאת. אם לא, תנו דוגמה למצב ממנו לא ניתן להגיע לבור.  
תזכורת: בור הוא מצב שלא ניתן להפעיל עליו אף אופרטור (או לפי ההגדרה בתרגול, מצב שהפעלת כל אופרטור עליו מחזירה  $\emptyset$ ).

## חלק ב' - היכרות עם הקוד

מומלץ להבין היטב את ההסברים בחלק זה.

### ייצוג הבעיה בקוד (קובץ MazeProblem.py)

#### המחלקה MazeProblem

מחלקה המייצגת בעיית מבוכ. כדי לייצר בעיית מבוכ יש צורך במטריצה שמייצגת את המפה (מכילה אפסים ואחדים), במיקומים ההתחלתיים של הראש והזנב, במיקומי היעד שלהם, ובעלות האופרטורים. המחלקה מממשת את הפונקציות הבאות:

`expand_state`

פונקציה המקבלת מצב ומחזירה בזה אחר זה את המצבים העוקבים האפשריים, עם מחיר הפעלת האופרטורים המתאימים. אין צורך לבדוק שהמצבים המוחזרים מפונקציה זו הם תקינים. שימו לב, מספר הצמתים

המפותחים בזמן הרצתו של אלגוריתם חיפוש הוא מספר הפעמים שנקראה הפונקציה `expand_state`.

`is_goal`

פונקציה המקבלת מצב ומחזירה האם הוא מצב מטר.

#### המחלקה MazeState

מחלקה זו מייצגת מצב במרחב החיפוש. כדי לייצר מצב, יש צורך בבעיית מבוכ (אובייקט מטיפוס `MazeProblem`), ובמיקומי הראש והזנב של הרובוט. המחלקה מממשת את הפונקציות הבאות:

`robot_direction`

פונקציה המחזירה מערך המייצג את הכיוון אליו הרובוט מסתכל (כפי שהוגדר באופרטור "התקדמות").

למשל, עבור בעיית המבוכ ע"י  $M_{example}$ , כיוון הרובוט הוא  $[0, 1]$ . האיבר הראשון הוא 0 כי בציר הראשון

שמתאים לשורות המטריצה, מיקומם של ראש וזנב הרובוט זהה. האיבר השני הוא 1 כי בציר השני, ראש הרובוט נצמא באינדקס גדול יותר מזנב הרובוט. אם הרובוט היה מסתכל למעלה, כיוון הרובוט היה  $[-1, 0]$ .

בנוסף בקובץ `MazeProblem.py` ממומשת הפונקציה `compute_robot_direction` שמחשבת את הכיוון אליו הרובוט מסתכל עבור מיקומי ראש וזנב כלשהם.

### כלים לפתרון בעיות חיפוש בגרף (קובץ GraphSearch.py)

#### המחלקה Node

מחלקה זו מייצגת צומת בעץ החיפוש. כדי לייצר צומת צריך את המצב שהצומת מייצג, את האב של הצומת בעץ החיפוש (אם הצומת הוא השורש) ואת ערך ה- $g$  של הצומת. המחלקה מממשת את הפונקציות הבאות:

`get_path`

פונקציה המחזירה את המסלול בעץ החיפוש עד הצומת.

#### המחלקה GraphSearchSolution

מחלקה זו מייצגת פתרון של בעיית חיפוש בגרף. כדי לייצר פתרון צריך את הצומת הסופי, את הזמן שלקח למציאת הפתרון, ואת מספר הצמתים שפותחו במהלך החיפוש. במידה ולא נמצא פתרון, יש לייצר אובייקט מהמחלקה עם `final_node=None`, ועם הסיבה שלא נמצא פתרון (נגמר הזמן, או שאין פתרון).

#### המחלקה NodeCollection

מחלקה זו מממשת אוסף של מצבים המשוויכים לצמתים בעץ החיפוש. אנו נשתמש במחלקה זו בתור `close` באלגוריתמי החיפוש.

המחלקה מממשת את הפונקציות הבאות:

`add`

הוספה של צומת לאוסף. המחלקה מניחה שהמצב המיוצג ע"י הצומת לא נמצא באוסף (כפי שמתקיים באלגוריתמי החיפוש כאשר מוסיפים צומת ל-`close`). אם הוא נמצא תתקבל שגיאה.

`remove_node`

מחיקה של צומת מהאוסף.

`get_node`

מחזירה את הצומת המתאים למצב.

בנוסף ניתן לבדוק אם מצב נמצא באוסף על ידי `in` אופרטור `(s in close)`.

### המחלקה `NodesPriorityQueue`

מחלקה זו מממשת תור עדיפויות של צמתים. אנו נשתמש במחלקה זו בתור `open` באלגוריתמי החיפוש.

`:add`

הוספה של צומת לתור, עם עדיפות מסוימת. המחלקה מניחה שהמצב המיוצג ע"י הצומת לא נמצא באוסף (כפי שמתקיים באלגוריתמי החיפוש כאשר מוסיפים צומת ל-`open`). אם הוא נמצא תתקבל שגיאה.

`:pop`

מוציאה מהתור ומחזירה את הצומת עם העדיפות הנמוכה ביותר. אם התור ריק יוחזר `None`.

`:get_node`

מחזירה את הצומת המתאים למצב.

`:remove_node`

מסירה את הצומת מהתור.

בנוסף ניתן לבדוק אם מצב נמצא בתור על ידי אופרטור `in` `(s in open)`, ולבדוק את מספר הצמתים בתור על ידי הפונקציה `len`, `(len(open))`.

### המחלקה `Queue`

המחלקה ממשת תור.

`:add`

הוספה של איבר לתור.

`:pop`

מחזירה ומוציאה מהתור איבר לפי סדר FIFO. אם התור ריק יוחזר `None`.

בנוסף ניתן לבדוק אם מצב נמצא בתור על ידי אופרטור `in` `(s in q)`, ולבדוק את מספר הצמתים בתור על ידי הפונקציה `len`, `(len(q))`.

## אלגוריתמי חיפוש (קובץ `Robot.py`)

### המחלקה `Robot`

מחלקה זו היא מחלקה אבסטרקטית עבור אלגוריתם חיפוש בגרף מצבים. מחלקות המממשות את המחלקה הזו צריכות לממש את הפונקציות הבאות:

`:solve`

מחלקה המקבלת בעיה מטיפוס `MazeProblem` ומחזירה פתרון מטיפוס `GraphSearchSolution`.

### המחלקה `BreadthFirstSearchRobot`

מחלקה זו יורשת מהמחלקה `Robot`, ומממשת `breadth first search`.

### המחלקה `BestFirstSearchRobot`

מחלקה זו היא מחלקה אבסטרקטית היורשת מהמחלקה `Robot`, ומממשת `best first search`. מלבד לפונקציה `solve`, מחלקות המממשות את מחלקה זו צריכות לממש את הפונקציה:

`:_calc_node_priority`

פונקציה המחשבת את העדיפות (ערך ה-`f`) עבור צומת בעץ החיפוש.

### המחלקות `WAStartRobot` ו-`UniformCostSearch`

מחלקות היורשות מ-`BestFirstSearchRobot`, ומממשות את אלגוריתמי החיפוש `WA*` ו-`UCS` בהתאמה.

## חלק ג' - חיפוש לא מיועד

בחלק זה נתחיל לכתוב קוד. הקובץ `main.py` שתגישו לא ייבדק, לכן אתם יכולים להרגיש חופשיים לשנות את ה-`main` כרצונכם בשביל לבצע את משימות התרגיל.

## משימה 4 (6 נק')

- השלימו את המימוש של המחלקה `BreadthFirstSearchRobot` בקובץ `Robot.py`.  
על אף שאלגוריתם `breadth first search` לא מתייחס למחירי האופרטורים, על הפתרון המוחזר להכיל את עלות המסלול (ולא את אורך המסלול מבחינת מספר האופרטורים בו).
- בקובץ `Utilities.py` ממומשת הפונקציה `test_robot` שמקבלת טיפוס לא אבסטרקטי היורש מהמחלקה `Robot`, אינדקסים של מבוכים, וגם פרמטרים לאתחול הרובוט אם צריך. הפונקציה טוענת את המבוכים מקבצי `csv`, מריצה את הרובוט על המבוכים ומדפיסה את התוצאות.  
הוסיפו קריאה ל-`test_robot` עם אובייקט מסוג `BreadthFirstSearchRobot` ועם מבוכים 0 עד 5:

```
if __name__ == "__main__":
    test_robot(BreadthFirstSearchRobot, [0, 1, 2, 3, 4, 5])
```

אתם צריכים לקבל את ההדפסה הבאה:

```
breadth first search robot solved maze_0 in 0.05 seconds. solution cost = 36, expanded 57 nodes.
breadth first search robot solved maze_1 in 0.28 seconds. solution cost = 51, expanded 362 nodes.
breadth first search robot solved maze_2 in 1.79 seconds. solution cost = 216, expanded 1212 nodes.
breadth first search robot solved maze_3 in 5.79 seconds. solution cost = 88, expanded 2430 nodes.
breadth first search robot solved maze_4 in 2.23 seconds. solution cost = 123, expanded 1209 nodes.
breadth first search robot solved maze_5 in 8.88 seconds. solution cost = 376, expanded 5299 nodes.
```

כמובן שזמן הריצה יכול להשתנות, אך מחיר הפתרון ומספר הצמתים שפותחו צריכים להיות זהים. אם אתם מקבלים תוצאות אחרות, עליכם לתקן את המימוש שלכם.

- אתם יכולים לצפות במבוכים ובפתרונות שנמצאו על ידי הרובוטים שממישתם על ידי קריאה לפונקציה `solve_and_display`. הפונקציה מקבלת טיפוס לא אבסטרקטי היורש מהמחלקה `Robot`, אינדקס של מבוכ, וגם פרמטרים לאתחול הרובוט אם צריך, ומציגה את הפתרון של בעיית המבוכ שנמצא ע"י הרובוט באנימציה. נסו זאת כעת עם `BreadthFirstSearchRobot` והמבוכים 0 עד 5:

```
if __name__ == "__main__":
    a = solve_and_display(BreadthFirstSearchRobot, 1)
```

יכול להיות שתתקלו בבעיות בהרצת האנימציה. באתר הקורס (איפה שהתרגיל) מופיע מסמך עזר להרצת האנימציה. אם ניסיתם את כל מה שנאמר במסמך ועדיין לא הצלחתם להריץ את האנימציה, תצרו בבקשה קשר עם טל כדי שנוכל לטפל בבעיה ולהוסיף לנספח את הפתרון. אפשר לפתור את התרגיל גם בלי האנימציה.

## משימה 5 (8 נק')

- השלימו את המימוש של המחלקה `BestFirstSearchRobot` בקובץ `Robot.py`.  
שימו לב, כמו שמוסבר בתרגול, אלגוריתם  $A^*$  הוא מקרה פרטי של `best first search` עבורו מתקיים  $f = g + h$ . הפסאודו קוד של `best first search` הוא זהה לזה של  $A^*$  למעט זה שפונקציית ה-`priority` היא כללית (נקבעת על ידי המימוש של `_calc_node_priority`).
- השלימו את המימוש של המחלקה `UniformCostSearchRobot` על ידי מימוש הפונקציה `_calc_node_priority`.
- קראו לפונקציה `test_robot` עם `UniformCostSearchRobot` ומבוכים 0 עד 5.  
אתם אמורים לקבל את ההדפסה הבאה:

```
uniform cost search robot solved maze_0 in 0.06 seconds. solution cost = 36, expanded 51 nodes.
uniform cost search robot solved maze_1 in 0.41 seconds. solution cost = 47, expanded 312 nodes.
uniform cost search robot solved maze_2 in 2.12 seconds. solution cost = 216, expanded 1212 nodes.
uniform cost search robot solved maze_3 in 6.82 seconds. solution cost = 84, expanded 2447 nodes.
uniform cost search robot solved maze_4 in 3.13 seconds. solution cost = 123, expanded 1205 nodes.
uniform cost search robot solved maze_5 in 12.0 seconds. solution cost = 376, expanded 5299 nodes.
```

## משימה 6 (5 נק')

1. מחיר המסלול שהתקבל מהרצת UniformCostSearchRobot על מבוכים 1 ו-3 קטן מזה המתקבל עם BreadthFirstSearchRobot. הסבירו מדוע זה קורה.
2. כתבו תנאי פשוט ככל הניתן על הפתרונות של בעיית חיפוש כללית כלשהי, כך שמחיר המסלול המוחזר מכל הרצה של breadth first search יהיה זהה למחיר המסלול המוחזר מכל הרצה של UCS.

## חלק ד' - חיפוש מיועד

### משימה 7 (5 נק')

1. השלימו את המימוש של המחלקה WStartRobot על ידי מימוש הפונקציה `_calc_node_priority`. שימו לב, אנו מגדירים  $f = (1 - w) \cdot g + w \cdot h$ .
2. כדי שנוכל להשתמש ב-WStartRobot, עלינו להגדיר יוריסטיקה. נגדיר את יוריסטיקת מנהטן להיות מרחק מנהטן בין זנב הרובוט למיקום היעד של הזנב כפול מחיר אופרטור ההתקדמות. ממשו יוריסטיקה זו בפונקציה `tail_manhattan_heuristic` שבקובץ `Heuristics.py`.
3. קראו לפונקציה `test_robot` עם WStartRobot המאותחל עם `tail_manhattan_heuristic`, ועם המפות 0 עד 5.

```
if __name__ == "__main__":
    test_robot(WStartRobot, [0, 1, 2, 3, 4, 5], heuristic=tail_manhattan_heuristic)
```

אתם אמורים לקבל את ההדפסה הבאה:

```
wA* [0.5, tail_manhattan_heuristic] solved maze_0 in 0.07 seconds. solution cost = 36, expanded 38 nodes.
wA* [0.5, tail_manhattan_heuristic] solved maze_1 in 0.36 seconds. solution cost = 47, expanded 193 nodes.
wA* [0.5, tail_manhattan_heuristic] solved maze_2 in 3.2 seconds. solution cost = 216, expanded 1196 nodes.
wA* [0.5, tail_manhattan_heuristic] solved maze_3 in 6.18 seconds. solution cost = 84, expanded 2312 nodes.
wA* [0.5, tail_manhattan_heuristic] solved maze_4 in 1.76 seconds. solution cost = 123, expanded 732 nodes.
wA* [0.5, tail_manhattan_heuristic] solved maze_5 in 10.69 seconds. solution cost = 376, expanded 5289 nodes.
```

### משימה 8 (12 נק')

- היוריסטיקה `tail_manhattan_heuristic` לא קבילה.
1. הסבירו מדוע, והגדירו תנאי הכרחי ומספיק על המחיר של אופרטורי הסיבוב כך שהיוריסטיקה תהיה קבילה בכל מפה.
  2. צרו קובץ `csv` בשם `maze_99` בתיקיה `Mazes` המייצג בעיית מבוכ עבודה הפתרון המוחזר על ידי WStartRobot המשתמש ביוריסטיקה זו עם  $w=0.5$ . אינו אופטימלי. משבצות ריקות במבוכ צריכות להכיל את הערך 0, משבצות קיר את הערך -1, במיקומים ההתחלתיים של זנב וראש הרובוט צריכים להופיע את הערכים 1 ו-2 בהתאמה, ובמיקומי המטרה של זנב וראש הרובוט צריכים להופיע הערכים 3 ו-4 בהתאמה. שימו לב, המבוכ צריך להיטען בהצלחה בקריאה לפונקציה `test_robot`.

### משימה 9 (5 נק')

- תיקון פשוט ליוריסטיקה הוא חישוב מרחק מנהטן בין נקודת מרכז הרובוט הנוכחית לבין נקודת מרכז הרובוט במיקום המטרה שלו.
1. הוכיחו כי יוריסטיקה זו קבילה.
  2. ממשו את יוריסטיקה זו בפונקציה `center_manhattan_heuristic`.

### משימה 10 (5 נק')

- הפרמטר  $w$  של אלגוריתם  $wA^*$  משפיע על רמת הגרידיות של אלגוריתם החיפוש. נרצה לבחון את ההשפעה של  $w$  על זמן הריצה ואיכות הפתרון של אלגוריתם  $wA^*$  על המבוכים 0 עד 5 בשימוש עם היוריסטיקה `center_manhattan_heuristic`.



1. השלימו את מימוש הפונקציה `w_experiment` שמקבלת אינדקס של מבוכ ושומרת בתיקה `plots` שני גרפים - גרף אחד של זמן הריצה של  $wA^*$  כפונקציה של  $w$ , וגרף שני של מחיר הפתרון שמצא  $wA^*$  כפונקציה של  $w$ , כאשר בשני הגרפים  $w$  יהיה בציר ה- $x$ , וערכיו יהיו בין 0 ל-1 בקפיצות של 0.1.
  2. הריצו את הפונקציה עם המפות 0 עד 2, צרפו את הגרפים לדוח והסבירו את התוצאות.
- \* מסתבר שאי אפשר לייצר קבצים עם כוכבית בשם שלהם ב- Windows, אז מי שמשתמש ב- Windows צריך להסיר את הכוכבית מהשמות של הגרפים שנשמרים בפונקציה, כלומר לשנות את  $wA^*$  ל-  $wA$  בקריאות לפונקציה `plt.savefig`.

## חלק ה' - יוריסטיקת מתוחכמת

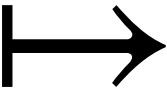
סימונים עבור חלק זה

- עבור זוג מצבים  $s, s'$  במרחב החיפוש, נסמן ב-  $d(s, s')$  את מחיר המסלול הקל ביותר בין המצבים. אם אין מסלול בין המצבים נגדיר  $d(s, s') = \infty$ .
- בהינתן בעיית מבוכ, נסמן ב-  $s^*$  את מצב המטרה וב-  $s^i$  את המצב ההתחלתי.

אחד הדברים שמקשים על התנועה של רובוט במבוכ הוא האורך שלו - רובוטים קצרים יותר יכולים לבצע פניות במקומות רבים יותר וכך להגיע למיקום היעד בדרכים זולות יותר. נוכל לנצל תכונה זו של הבעיה כדי להגדיר את היוריסטיקה  $h_k(s)$ :  $h_k(s)$  הוא מחיר המסלול הקצר ביותר מהמצב  $s$  עבור רובוט קטן ב-  $k$  מהרובוט המקורי (עבור  $k$  כלשהו קבוע, זוגי, לכל היותר אורך הרובוט פחות 3).

בהינתן מצב  $s$ , נגדיר את  $s_k$  להיות מצב הזהה ל-  $s$  מלבד לכך שהרובוט התקצר ב-  $k$  משבצות. ליתר דיוק, ב-  $s_k$  מיקומי הראש והזנב של הרובוט קרובים למרכזו ב-  $k/2$  משבצות. למשל אם המצב השמאלי בדוגמה הבאה הוא  $s$ , אז  $s_k$  עבור  $k = 2$  הוא המצב מימין.

	1	2	3	4	5	6
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	0	-1	0	0
3	1	0	0	0	2	0
4	0	0	0	0	0	0



	1	2	3	4	5	6
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	0	-1	0	0
3	0	1	0	2	0	0
4	0	0	0	0	0	0

כמו כן, עבור  $s^*$  מצב המטרה של בעיית המבוכ, נגדיר את  $s_k^*$  להיות זהה ל-  $s^*$ , מלבד לכך שמיקומי הראש והזנב קרובים יותר באותו האופן. כעת נוכל לרשום  $h_k(s) = d(s_k, s_k^*)$ .

נשים לב שבמימוש נאיבי, שימוש ב-  $h_k$  יגזול הרבה זמן - עבור כל מצב  $s$  שנוצר במהלך החיפוש, נצטרך לחשב את  $d(s_k, s_k^*)$  על ידי פתרון בעיית המבוכ המתאימה. במקום זאת, נבצע חישוב מקדים (לפני החיפוש) ממנו נוכל להסיק את  $h_k(s)$  לכל מצב  $s$ . עבור מצב  $s$  נגדיר את  $\bar{s}$  להיות מצב בו מיקומי הראש והזנב של הרובוט התחלפו.

משימה 11 (5 נק')

הסבירו מדוע לכל זוג מצבים  $s, s'$  מתקיים  $d(s, s') = d(\bar{s}', \bar{s})$ .

לפיכך מתקיים  $h_k(s) = d(s_k, s_k^*) = d(\bar{s}_k^*, \bar{s}_k) = d(\bar{s}_k^*, \bar{s}_k)$  לכל מצב  $s$ , וכך נוכל לחסוך זמן יקר במהלך החיפוש.

כפי שלמדנו (או תלמדו) בקורס אלגוריתמים 1, ניתן לחשב ביעילות את מחיר המסלול הקל ביותר מצומת מסויים לכל שאר הצמתים בגרף בעזרת אלגוריתם דייקסטרה. למעשה, כבר מימשתם את אלגוריתם זה בקוד כאשר

מימשתם את UniformCostSearchRobot. קריאה לפונקציה solve עם `compute_all_dists=True` תמנע מהחיפוש להסתיים עד ש- `open` יתרוקן, ותחזיר את `close`, שיכיל את כל המצבים הישיגים מהצומת ההתחלתי, ואת ערכי ה- `g` שלהם, שמובטחים להיות מחירי המסלולים הקצרים ביותר מהמצב ההתחלתי.

### משימה 12 (8 נק')

נניח לשם משימה זו בלבד שבשלב החישוב המקדים, במקום לחשב את המסלול **הזול** ביותר מ-  $\bar{s}_k^*$  לכל המצבים, מחשבים את המסלול **הזול** ביותר מ-  $\bar{s}_k^*$  ל-  $\bar{s}_k^i$  בלבד, עבור  $s^i$  המצב ההתחלתי של הבעיה המקורית, בעזרת אלגוריתם  $A^*$  ושימוש ביוריסטיקה  $h_{pre}$  כלשהי. בעת הרצת החיפוש עבור פתרון הבעיה המקורית, משתמשים גם כן באלגוריתם  $A^*$ , אבל עם היוריסטיקה  $h(s) = g_{\bar{s}_k^*}(\bar{s}_k)$  כאשר  $g_{\bar{s}_k^*}(\bar{s}_k)$  הינו ערך ה- `g` המינימלי שנשמר למצב  $\bar{s}_k$  הנמצא ב- `close` במהלך הריצה בשלב החישוב המקדים. במידה והמצב  $\bar{s}_k$  לא הוכנס ל- `close` במהלך החישוב המקדים נגדיר  $g_{\bar{s}_k^*}(\bar{s}_k) = 0$ .

1. האם היוריסטיקה  $h(s) = g_{\bar{s}_k^*}(\bar{s}_k)$  קבילה בהכרח? אם כן, הוכיחו. אם לא, הגדירו תנאי על  $h_{pre}$  כך שאם הוא מתקיים אז היוריסטיקה  $h(s) = g_{\bar{s}_k^*}(\bar{s}_k)$  היא קבילה בהכרח.
  2. האם **באופן כללי** שימוש ביוריסטיקה לא קבילה מבטיח קבלת פתרון לא אופטימלי? הסבירו.
- \* המשימה הזו השתנתה לאחר פרסום התרגיל, תשובות עבור הגרסה הקודמת יתקבלו גם כן.

### משימה 13 (9 נק')

- נממש את היוריסטיקה  $h_k$  במחלקה `ShorterRobotHeuristic`. שימו לב, בניגוד למה שתואר במשימה 12, בחישוב המקדים אנחנו נחשב מרחק קצר ביותר לכל המצבים הנגישים מ-  $\bar{s}_k^*$ , כפי שתואר לאחר משימה 11. החישוב של ערכי  $d(\bar{s}_k^*, \bar{s}_k)$  יתבצע בעת אתחול המחלקה, על ידי יצירת בעיית המבוך המתאימה וחישוב כל המרחקים מהמצב ההתחלתי שלה על ידי הרצת אלגוריתם `UniformCostSearch`.
1. ממשו את הפונקציה `_compute_shorter_head_and_tails` אשר מקבלת מיקומי ראש וזנב של רובוט ומחזירה את מיקומי הראש והזנב לאחר קיצורו של הרובוט ב- `k` יחידות (שימו לב, `k` נשמר כשדה במחלקה).
  2. השלימו את המימוש של הפונקציה `__init__` במחלקה `ShorterRobotHeuristic` לפי השלבים הבאים:
    1. צרו את בעיית המבוך בעזרתה נחשב את ערכי  $d(\bar{s}_k^*, \bar{s}_k)$ . שימו לב, מכיוון שברצוננו לחשב את המרחקים מהמצב ההתחלתי לכל המצבים, ולמעשה להתעלם ממצב המטרה של הבעיה, לכן אין חשיבות להגדרת מיקום הראש והזנב של הרובוט במצב המטרה.
  2. שמרו את ה- `close` של `UCS` שמוחזר מקריאה לפונקציה `solve` עם פרמטר `compute_all_dists=True`.
  3. ממשו את הפונקציה `__call__` של המחלקה `ShorterRobotHeuristic`, אשר תקרא בעת חישוב ערך יוריסטיקה של מצב.

### משימה 15 (8 נק')

- בעוד שחסכנו זמן רב בעזרת מימוש יעיל של היוריסטיקה, היא עדיין דורשת חישוב מקדים כבד. נרצה לנתח את היוריסטיקה החדשה ולהבין מתי כדאי להשתמש בה.
- נניח כי קיים מסלול יחיד בין  $\bar{s}_k^*$  ל-  $\bar{s}_k^i$ .
1. הוכיחו כי אם קיים פתרון לבעיה המקורית, אז הוא יחיד.
  2. הוכיחו או הפריכו:  $h_k = h^*$ .
  3. בהנחה שקיים פתרון לבעיה המקורית, הוכיחו כי בזמן פתרון הבעיה המקורית עם  $h_k$ ,  $A^*$  מפתח אך ורק צמתים על מסלול הפתרון האופטימלי.

נסמן ב-  $n_{manhattan}$  את מספר הצמתים שפותחו בזמן ריצת  $A^*$  עם `center_manhattan_heuristic`, ב-  $m_k$  את מספר הצמתים שפותחו בזמן ריצת החישוב המקדים עם אלגוריתם `UCS`, וב-  $n_k$  את מספר הצמתים שפותחו בזמן ריצת  $A^*$  עם היוריסטיקה  $h_k$  (לא כולל  $m_k$  הצמתים שפותחו בזמן החישוב המקדים). נניח שזמן הריצה של החישוב המקדים הוא  $c \cdot m_k$ , זמן הריצה של אלגוריתם  $A^*$  עם היוריסטיקה  $h_k$  הוא  $c \cdot n_k$ .

זמן הריצה של אלגוריתם  $A^*$  עם יוריסטיקת מנהטן הוא  $c \cdot n_{manhattan}$  עבור  $c$  כלשהו. לפיכך עדיף (מבחינת זמן) להשתמש ביוריסטיקה החדשה אם ורק אם מתקיים:

$$n_k + m_k < n_{manhattan}$$

4. נסמן ב- $a$  את מספר המצבים הנגישים מ- $s_k^*$ , ונניח כי קיים מסלול יחיד בין  $\bar{s}_k^*$  ל- $s_k^*$  וכי קיים פתרון לבעיה המקורית שמספר האופרטורים בו הוא  $r$ . תנו תנאי הכרחי ומספיק לכך שזמן הריצה של  $A^*$  עם היוריסטיקה  $h_k$  (כולל חישוב מקדים) קטן מזמן הריצה של  $A^*$  עם יוריסטיקת  $center\_manhattan\_heuristic$ . השתמשו אך ורק במשתנים  $r, a$  ו- $n_{manhattan}$ .

5. **שאלת בונוס!** (5 נק' לציון בתרגיל, והרבה הערכה) תארו מפה כללית בה זמן הריצה של  $A^*$  עם  $center\_manhattan\_heuristic$  גדול כרצוננו מזמן הריצה של  $A^*$  עם  $h_k$  (כולל זמן החישוב המקדים). פתרונות שמכילים רק את הרעיון הכללי ולא מפרטים איך נראית המפה ממש יקבלו 2 נק'.

### משימה 16 (6 נק')

נרצה לבדוק האם השימוש ביוריסטיקה החדשה משתלם.

1. קראו לפונקציה `test_robot` עם `WStartRobot` המאותחל עם  $w=0.5$  ועם `ShorterRobotHeuristic`, עם ערכי  $k$  משתנים, ומפות שונות. האם השימוש ביוריסטיקה החדשה משפר את הביצועים לעומת השימוש ב-`center\_manhattan\_heuristic`? עבור הקריאות הבאות ל-`test_robot`:

```
if __name__ == "__main__":
    for k in [2, 4, 6, 8]:
        test_robot(WStartRobot, [3, 4], heuristic=ShorterRobotHeuristic, k=k)
```

אתם אומרים לקבל את ההדפסות הבאות:

```
WA* [0.5, ShorterRobotHeuristic, {'k': 2}] solved maze_3 in 8.54 seconds. solution cost = 84, expanded 340 nodes.
WA* [0.5, ShorterRobotHeuristic, {'k': 2}] solved maze_4 in 0.21 seconds. solution cost = 123, expanded 107 nodes.
WA* [0.5, ShorterRobotHeuristic, {'k': 4}] solved maze_3 in 10.93 seconds. solution cost = 84, expanded 1159 nodes.
WA* [0.5, ShorterRobotHeuristic, {'k': 4}] solved maze_4 in 0.25 seconds. solution cost = 123, expanded 107 nodes.
WA* [0.5, ShorterRobotHeuristic, {'k': 6}] solved maze_3 in 10.64 seconds. solution cost = 84, expanded 1383 nodes.
WA* [0.5, ShorterRobotHeuristic, {'k': 6}] solved maze_4 in 4.74 seconds. solution cost = 123, expanded 137 nodes.
WA* [0.5, ShorterRobotHeuristic, {'k': 8}] solved maze_3 in 11.41 seconds. solution cost = 84, expanded 1606 nodes.
WA* [0.5, ShorterRobotHeuristic, {'k': 8}] solved maze_4 in 5.31 seconds. solution cost = 123, expanded 137 nodes.
```

2. הסבירו באופן כללי כיצד הערך  $k$  משפיע על זמני הריצה.



3. נרצה לבחון את השפעת הערך  $k$  על זמני הפתרון של המפות השונות. השלימו את המימוש של הפונקציה `shorter_robot_heuristic_experiment`, אשר פותרת את בעיית המבוך בעזרת הרצת אלגוריתם  $wA^*$  עם `ShorterRobotHeuristic` וערכי  $k$  שונים, ומייצרת גרף המציג את זמן הריצה הכולל וזמן אתחול היוריסטיקה עבור כל אחד מהערכים השונים של  $k$ .

4. הריצו את הפונקציה `shorter_robot_heuristic_experiment` עם המבוכים 2 עד 5, צרפו את הגרפים לדוח והסבירו את התוצאות. (ההרצה חלק מהמפות צפויה לקחת כמה דקות, בעיקר על מפה 5).

## חלק ו' - שתי שאלות על $IDA^*$

משימה 17 (10 נק')

במהלך השאלה הניחו שמשתמשים ביוריסטיקות קבילות בלבד.

1.  במהלך ריצת  $IDA^*$ , בסיום איטרציה כלשהי (כלומר לאחר החזרה מקריאה ל- $DFS - f$ ), הוחלט להגדיל את אורך הרובוט ב-2 יחידות. האם מובטח שנקבל פתרון אופטימלי עבור הרובוט הגדול אם נשתמש בערך  $new-limit$  שהתקבל באיטרציה האחרונה? הסבירו.
2.  רוצים לפתור מספר בעיות מבין מפת המבוך היא זהה, ומיקומי ההתחלה והיעד של הרובוטים זהים, למעט אורכי הרובוטים (בדומה להגדרת  $s_k$ ). הוחלט להשתמש באלגוריתם  $IDA^*$ . כיצד ניתן לקצר את זמן הריצה הכולל של פתרון הבעיות עבור כל הרובוטים, על ידי אתחולים חכמים של ערכי  $f-limit$ ? הסבירו.

## הוראות הגשה

- יש לכתוב בדוח את מספרי תעודות הזהות של שני המגישים.
- הדוח צריך להיות מוקלד ובפורמט PDF. **דוחות בפורמט word יקבלו קנס.**
- הדוח צריך להיות קריא ומסודר.
- התשובות בדוח צריכות להופיע לפי הסדר.
- יש להגיש קובץ zip עם השם AI\_HW1\_123456789\_987654321 (עם מספרי תעודות הזהות שלכם במקום המספרים).
- ב- zip צריכים להיות:
  - הדוח בפורמט PDF.
  - כל קבצי הקוד והתיקיות שקיבלתם, עם כל הקבצים שהיו בהן, ועם הקובץ maze\_99 מסוג csv (לא excel) בתיקיה Mazes.

# בהצלחה!