מבוא לבינה מלאכותית תרגיל ראשון - חלק יבש

איל רביד, ראובן טימסיט

שאלה 1.

2. עבור (S,O,I,G) מתקיים ש S זה כל המספרים השלמים בין אפס ל 63 (כולל 0 ו 63). כלומר:

$$S = \{i|0 \leq i \leq 63\}$$

כאשר i שלם. כל מצב מייצג את המשבצת הנוכחית עליה איימי נמצאת בקמפוס.

מתקיים ש O זה הקבוצה {0,1,2,3} כאשר 0 זה הליכה של צעד למטה, 1 זה הליכה צעד ימינה, 2 זה צעד למעלה ו 3 זה צעד שמאלה.

מתקיים ש I זו הקבוצה {0} (מצב התחלתי יחיד). ו G זה הקבוצה {63} (מצב סופי יחיד).

3. לפי הסעיף הקודם, גודל מרחב המצבים זה 64 מכיוון המפה המדוברת היא בגודל של 64 משבצות (8 על 8) וכל מצב מייצג את מיקומה הנוכחי של איימי במפה.

4. הדומיין של האופרטור DOWN זה כל המצבים שאינם מצבים סופיים. כלומר מספרים שמייצגים משבצת במפה שאינה מצב המטרה או בור. כל המצבים שמייצגים משבצת בשורה התחתונה של המפה נמצאים ב דומיין כי ניתן להפעיל עליהם את האופרטור DOWN ולהשאר באותו מצב.

5. הפונקציה Succ על המצב ההתחלתי תחזיר את:

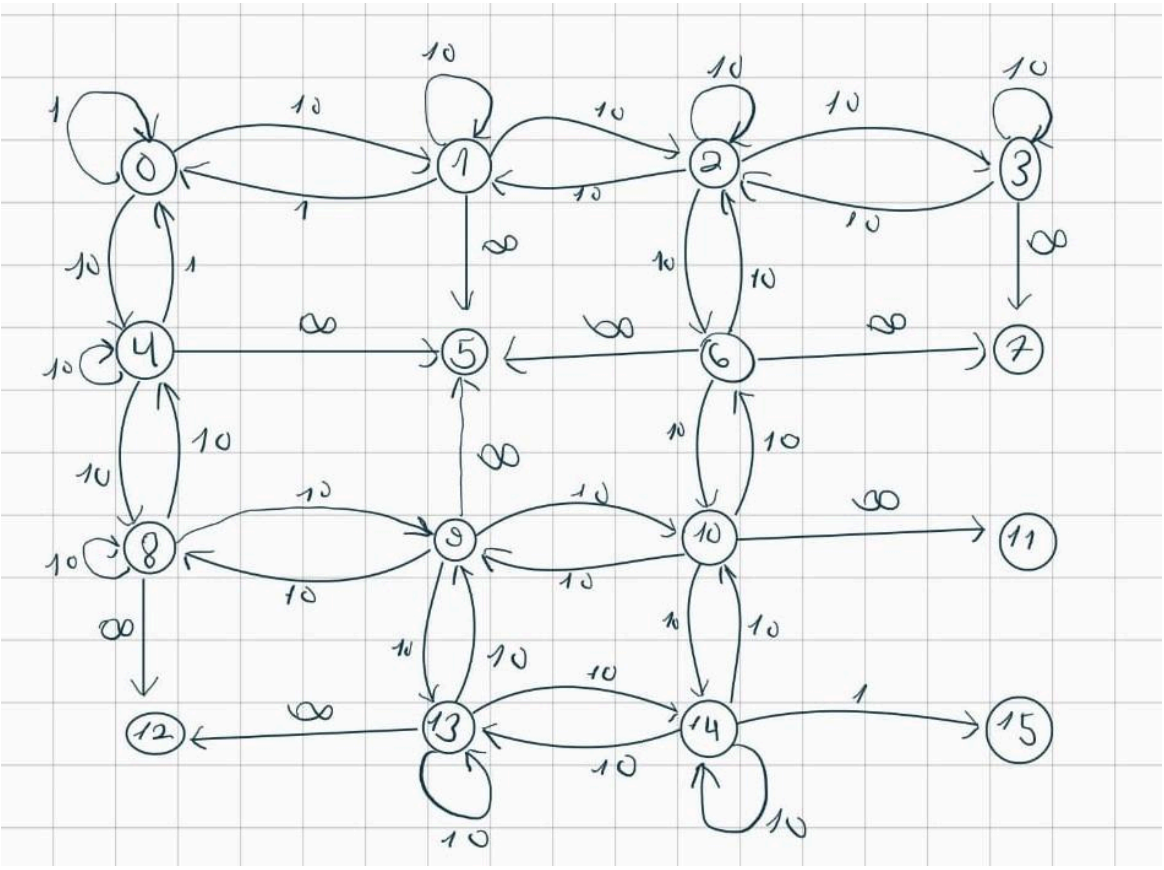
$$\{0, 1, 8\}$$

אלה המצבים שנגיע אליהם מהליכה למטה (מצב 8), ימינה (מצב 1) ולמעלה או שמאלה (מצב 0)

6. קיימים מעגלים במרחב החיפוש כי ניתן ללכת במעגלים במפה. לדוגמא ממצב 0 ללכת למטה למצב 8 ואז ממצב 8 ללכת למעלה למצב 0.

7. מקדם הסיעוף בבעיית הניווט בקמפוס הוא 4 כי מכל מצב אפשר להגיע לכל היותר ל 4 מצבים שונים (על ידי הפעלת ארבעת הפעולות האפשריות).

8. סרטוט של הגרף (המספרים על הקשתות זה מחיר המעבר והמספרים בצמתים זה שמות המצבים):



9. במקרה הגרוע ביותר, ידרשו לסוכן הרנדומלי אינסוף מצבים להגיע למצב הסופי (לדוגמא אם הוא הוא הוא ילך למעלה ואז למטה ואז שוב למעלה...). במקרה הטוב הוא יוכל לעשות את זה ב 14 צעדים (אם הוא ילך שמאלה 7 צעדים ואז עוד 7 צעדים למטה).

10. זוהי טענה לא נכונה. נסתכל על המפה הבאה:

S	L	L	G2
F	H	H	H
G1	H	H	H
H	H	H	H

במקרה זה אורך המסלול הקצר ביותר מהמצב ההתחלתי למצב סופי זה המסלול מ S ל G1 שהולך 2 צעדים למטה אבל מחירו הוא 11 (1+10). לעומת זאת המסלול הזול ביותר במפה זה המסלול ל G2 (הליכה 3 צעדים שמאלה מהמצב ההתחלתי). מחירו של מסלול זה הוא 3 (1+1+1) ובכך הראינו שיכול להיות שהמסלול הקצר ביותר לא שווה למסלול הזול ביותר במפה כללית.

שאלה 2.

1. אלגוריתם BFS שלם מכיוון שלגרף יש מקדם סיעוף סופי ולכן כפי שלמדנו בקורס, האלגוריתם שלם (אם קיים פתרון הוא ימצא אותו). האלגוריתם אינו קביל. הסיבה לכך היא המחיר הקשתות אינו אחיד ולכן פתרון לא בהכרח זול ביותר. לדוגמא במפה הבאה:

S	L	L	H
F	H	L	H
F	H	L	H
G	L	L	H

במפה זו הפתרון הזול ביותר הוא ללכת דרך משבצות ה - L ואילו הפתרון הקצר ביותר זה ללכת על משבצות ה - F ומחירם שונים ולכן במקרה זה BFS לא ימצא את הפתרון הזול ביותר.

2. כדי ששני האלגוריתמים ייצרו ויפתחו צמתים באותו הסדר, אז נדרש שעומק הפתרון הרדוד ביותר יהיה רדוד יותר מאשר כל מעגל בגרף (כלומר מהקשת האחרונה במעגל). ריצת שני האלגורתימים היא זהה עד לרגע שבוא אלגוריתם BFS-T יוצר צומת שהוא כבר יצר. לכן במקרה שעומק הפתרון הרדוד ביותר יהיה רדוד יותר מאשר כל מעגל בגרף (מהקשת האחרונה במעגל) אז אלגוריתם BFS-T לא יספיק לייצר אף צומת פעמיים (כי הוא ימצא קודם את הפתרון) ולכן שני האלגורייתמים ייצרו ויפתחו צמתים באותו הסדר.

3. נגדיר $f: G \to G'$ כך $f(G)$ יהיה אותו גרף כמו G רק שנכפה עליו שכל הקשתות יהיו במחיר 1. נעשה זאת כך שאם בגרף G היתה קשת ממצב v למצב u במחיר n כלשהו, אז בגרף החדש יהיו n קשתות במחיר 1 שיוצרות מסלול בין v ל u. כך, נשמור על המחירים המקוריים של המסלולים וגם הגרף החדש שייצרנו הוא בעל מחיר אחיד על הקשתות. בנוסף נסמן את הקשתות החדשות שהוספנו, כדי שנוכל לדעת איזה קשתות היו בגרף המקורי ואיזה לא (כדי שנוכל לשחזר מסלול בגרף המקורי). כעת אפשר להריץ BSF על הגרף $f(G)$ ולקבל פתרון אופטימלי לפי גרף זה (כי מחיר הקשתות אחיד) לאחר מכן נשחזר את המסלול בגרף המקורי. מסלול זה יהיה אופטימלי גם לפי הגרף המקורי מכיוון שלא שינינו את המחירים של המסלולים ובכך פתרנו את הבעיה.

4. יפותחו $N^2 - 2$ צמתים וייוצרו $4(N^2 - 2)$ צמתים. דבר זה נובע מההבחנה שהעומק של כל מצב בגרף המצבים זה מרחק מנהנטן מצומת ההתחלה. מרחק מנהטן של מצב מהמצב ההתחלתי חוסם מלטה את המרחק (כי אי אפשר לנוע באלכסון ואין פורטלים) והוא גם חוסם מלמעלה את המרחק כי יש מסלול באורך זה מהצומת ההתחלה לצומת. מכיוון ש BFS מוצא מסלולים קצרים ביותר אז העומק של כל צומת יהיה כמרחק מנהטן שלה. לכן, מכיוון שכל הצמתים בגרף מקיימים שמרחק מנהטן שלהן מצומת ההתחלה קטן מהמרחק מנהטן של צומת המטרה מצומת ההתחלה אז כל הצמתים בגרף ייוצרו (כי הן ימצאו לפני צומת הסיום). בנוסף, צומת הסיום לא תפותח וגם, מכיוון שיש 2 צמתים בגרף הניתן להגיע מהם לצומת הסיום, אז אחת מהן לא תפותח (כי ברגע שאנחנו נפתח את הראשונה אז, נסיים את הריצה) ומכיוון שמדובר ב BFS-G (כלומר אף צומת לא יפותח פעמים אז מספר הצמתים שיפותחו זה $N^2 - 2$ ומספר הצמתים שיווצרו זה $4(N^2 - 2)$ כי כל צומת מייצרת 4 צמתים.

שאלה 3.

1. קוד.

2. האלגוריתם שלם מכיוון שמדובר ב DFS-G, כלומר לא מפתחים את צומת כלשהי יותר מפעם אחת ולכן, מכיוון שהגרף סופי אז במקרה הגרוע נעבור על כל המצבים עד שבסוף נמצא פתרון. האלגוריתם אינו קביל בגלל שהמסלול שהוא ימצא לא בהרכח הזול ביותר. לדוגמא נסתכל על המפה:

S	L
F	G

אז המסלול שימצא האלגוריתם הוא הליכה למטה (ל F) ואז ימינה, אבל המסלול הזול ביותר הוא הליכה ימינה (ל A) ואז למטה ל G).

3. אלגוריתם dfs שרץ על עץ יתחיל לרדת למטה שוב ושוב (כי זו הפעולה שמוגדרת כראשונה). במידה והאלגוריתם לא יתקל בבור, אז הוא יגיע לשורה האחרונה ושם הוא יכנס ללואה אינסופית, מכיוון שהפעולה "צעד למטה" מוגדרת עבור מצבים בשורה האחרונה אך היא תחזיר אותו לאותו מצב בדיוק, ומכיוון שמדובר ב dfs שרץ על עץ אז נמשיך לפתח את אותו הצומת שוב ושוב (כל פעם נלך צעד אחד למטה וזה יחזיר אותנו לאותו מצב). במידה והאלגוריתם יתקל בבור, אז הוא ילך צעד אחד ימינה (כי זו הפעולה השניה) ולאחר מכן הוא שוב ינסה לרדת למטה כמה שהוא יכול והוא שוב עלול להיתקע בלולאה אין סופית מאותה הסיבה שהסברנו.

4. **סעיף 1.**

ייוצרו $4N + 4(N - 2) = 8N - 8$ ויפותחו $N + N - 2$ צמתים. זה בגלל שב DFS-G לא נפתח אף צומת פעמיים ולכן, מהמצב ההתחלתי נלך כל הזמן למטה ונייצר כל פעם את המצב את ארבעת המצבים שיווצרו מהפעולות האפשריות (למטה, ימינה, למעלנ ושמאלה). וכשהגענו לשורה האחרונה אז נתחיל ללכת ימינה ובכל פעם נייצר שוב את כל המצבים האפשריים. לכן בהתחלה נייצר $4N$ מצבים (כשהולכים למטה אז נייצר את 4 מצבים לכל מצב בטור הראשון) ואז כנתחיל ללכת ימינה אז נייצר $4(N - 2)$ מצבים. מספר הצמתים שנפתח נובע מכך במפתחים רק את הצמתים בטור הראשון (כי

בהתחלה כל הזמן הולכים למטה לפי סדר הפיתוח) ואז נלך ימינה עד שנגיע למצב הסופי (שאותו לא מפתחים) ולכן נפתח עוד N-2 מצבים.

סעיף 2.

אם נשתמש ב backtracking אז נייצר 2N-1 צמתים ונפתח 2N-2 צמתים. הסיבה לכך היא שהפעם לא נייצר את הצמתים בטור השני (כי מייצרים את הצמתים רק לפני שמפתחים אותם) וגם לא נייצר את הצמתים בשורה לפני האחרונה. לכן נייצר רק את הצמתים בטור הראשון ובשורה האחרונה. נפתח את אותה כמות צמתים כי כל צומת שמפתחים ב DFS-, אז מייצרים גם ב DFS-G backtracking, ולהפיך. היתרון בשיטה זו, כפי שרואים היא חסכון במקום.

5. סעיף 1.

כעת בשביל להתאים את הבעיה כך שאיימי תוכל למצוא פתרון מבלי להפר את מגבלת העומק נוכל לשנות את קבוצת האופרטורים O כך שנוסיף אופרטורים שמאפשרים לעשות שני צעדים. לדוגמא נוסיף אופרטור שמאפשר ללכת למטה ואז ימינה (והמחיר של אופרטור זה יהיה מחיר הסכום של שני הצעדים). הסיבה לכך שכעת נוכל למצוא פתרון היא שכעת, המסלול הקצר ביותר לצומת מטרה יהיה באורך d/2 וזה מכיוון שהוספנו אופרטורים חדשים שמצבעים צעד כפול. לכן כעת DFS-L עד לעומק d/2 ימצא את מסלול לצומת מטרה. בנוסף נגדיר את הסדר הפעולות כך שישמר הסדר פעולות לפי המרחב הקודם ואחרי הפעולות המקוריות אז הפעולות החדשות יגיעו בסדר שיהיה תואם לסדר הקודם (כלומר קודם למטה-למטה ואז למטה-ימינה ואז למטה-למעלה וכך הלאה).

סעיף 2.

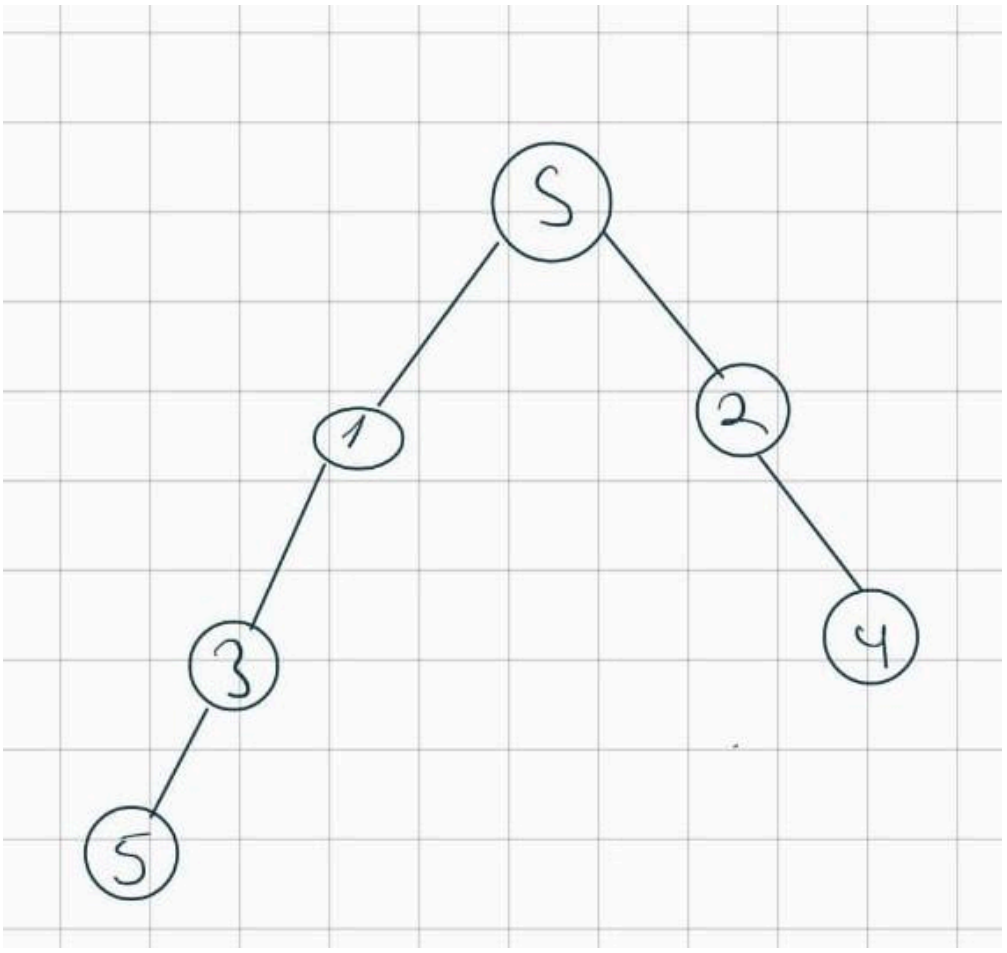
כעת, מקדם הסיעוף ישתנה וזה מכיוון שיש לאליס יותר פעולות אפשריות לבצע. אם בגרסא המקורית, אליס יכלה לבצע b פעולות (כלומר מקדם הסיעוף המקורי הוא b) אז כעת, אליס תוכל לבצע את b הפעולות המקוריות ועוד b פעולות על כל פעולה מהפעולות המקריות, כלומר מקדם הסיעוף החדש יהיה $b' = b + b^2$.

סעיף 3.

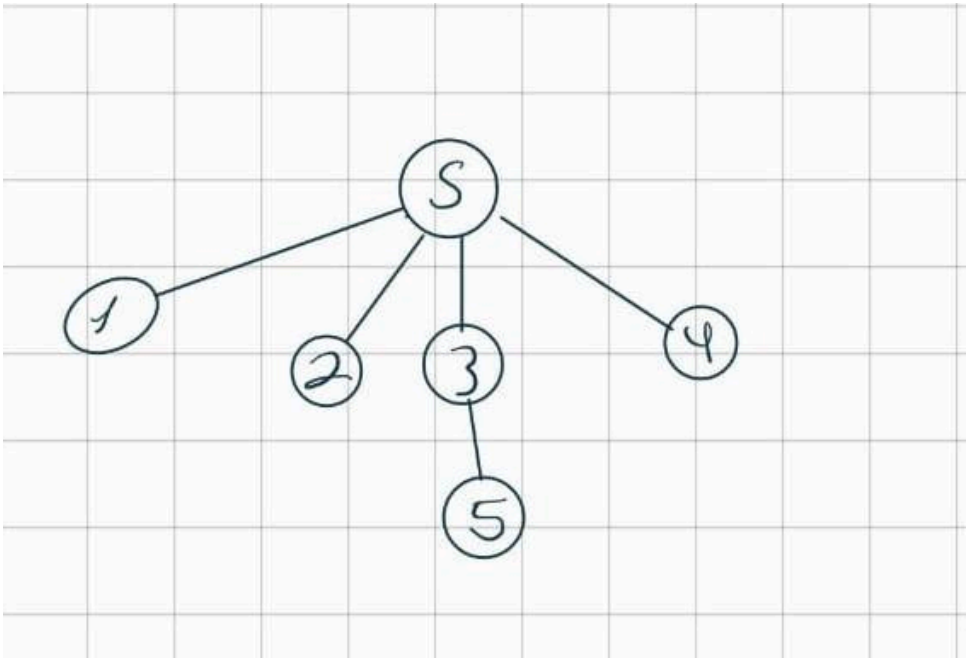
נשתמש בתוצאות הידועות על סיבוכיות המקום של והזמן של dfs-L רק שנציב את ערכי b ו d החדשים (כלומר $b + b^2$ ו d/2) ונקבל שסיבוכיות הזמן היא $O((b + b^2)^{d/2})$ וסיבוכיות המקום היא $O((b + b^2)\frac{d}{2})$.

סעיף 4.

דוגמא לכך ש dfs-L במרחב החדש טוב יותר, עץ המצבים הבא (נניח שזה העץ המקורי ושסדר הפעולות הוא מעבר על הבנים משאל לימין וגם הקשתות מכוונות מלמעלה למטה):



אז במקרה שמצב 4 יהיה מצב מטרה, אז עבור במרחב המקורי, נצטרך להריץ dfs-2 (כי המרחק לצומת מטרה זה 2) ובמקרה זה נפתח את צמתים s,1,2 ומצומת 2 נמצא את צומת 4. לעומת זאת, עץ החיפוש במרחב החדש יראה כך:



במרחב זה נצטרך להריץ dfs-1 (כי המרחק למטרה הוא 1) ולכן נפתח רק את צומת s שממנה נמצא את צומת 4. וזה דוגמא לבעיה שבה dfs-L במרחב החדש עדיף על המרחב המקורי. דוגמא למצב ההפוך היא אם ניקח את אותו מרחב חיפוש מקורי רק שהפעם, מצב המטרה יהיה מצב 5. במקרה זה, במרחב המקורי נריץ DFS-3 ונפתח את הצמתים S,1,3 (כלומר 3 צמתים). במרחב החדש, נצטרך להריץ DFS-2 ונצטרך לפתח את הצמתים s,1,2,3 ואז נמצא את מצב המטרה (כלומר פיתחנו 4 צמצים) וזו דוגמא שבא עדיף היה להריץ DFS-L על המרחב המקורי. כנדרש.

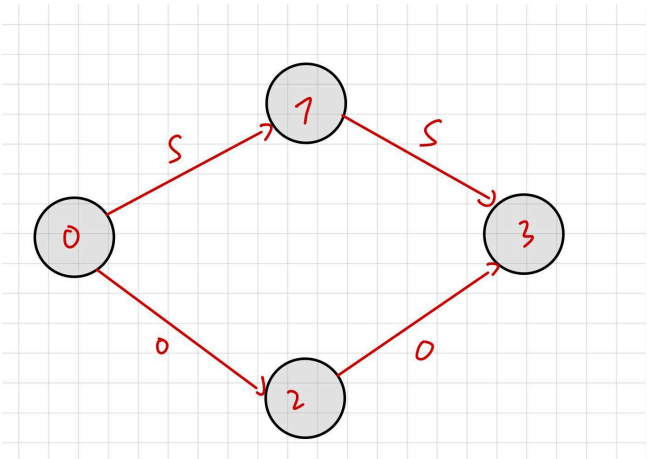
שאלה 4.

1. בקוד

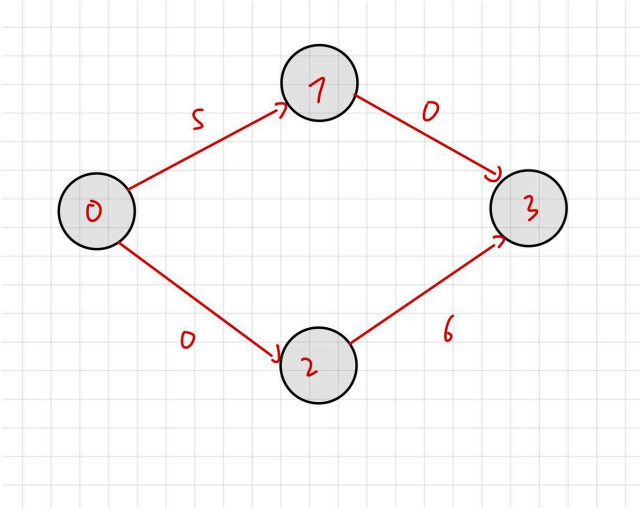
2. אלגוריתם UCS שלם מכיוון שלגרף יש מקדם סיעוף סופי ולכן כפי שלמדנו בקורס, האלגוריתם שלם (אם קיים פתרון הוא ימצא אותו). האלגוריתם הוא גם קביל כי הוא ימצא את הפתרון הזול ביותר כפי שראינו בהרצאות.

3. האלגוריתם UCS ו BFS-G יפעלו באותו אופן במקרה שבו המחיר של כל קשתות שווה, במקרה זה, המחיר של כל צומת לפי UCS יהיה לינארי במרחק המינימלי שלו מהשורש (בקשתות) ולכן הוא יפעל בדיוק כמו BFS-G (שגם הוא תלוי במרחק בקשתות מהשורש).

4. בדוגמה הבאה האלגוריתם יחזיר את המסלול הנכון. הוא יבחר בקשת עם עלות 0 מצומת 0 ל2 ואז שוב את הקשת עם עלות 0 ויגיע לצומת המטרה.



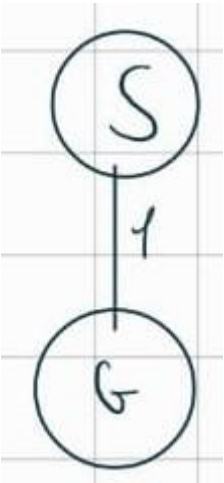
בדוגמה הבאה האלגוריתם יחזיר מסלול הלא נכון. הוא יבחר את הקשת עם עלות 0 מצומת 0 ל2 ואז את הקשת עם עלות 6
ויחזיר כי המסלול הקל ביותר הוא במשקל 6 אך יש מסלול קל יותר מצומת 0 ל1 ומשם ל3 במשקל 5.


0
1
2
3
5
0
0
6

שאלה 5.

1. **סעיף 1.**

זוהי טענה לא נכונה. נציג דוגמא נגדית.נסתכל על הגרף (הקשת מכוונת מלמעלה למטה):



ונסתכל על $h_1 = h_2 = h^*$. מתקיים ש h1 ו h2 קבילה (כי h* קבילה) אבל יתקיים ש $(h_1 + h_2)(s) = 2 > h^*(s)$ ולכן זוהי דוגמא נגדית.

סעיף 2.

זוהי טענה נכונה. יהיו h1 ו h2 עקביות. לכן מתקיים לכל מצב v:

$$0 \le \frac{(h_1+h_2)(v)}{2} = \frac{h_1(v)+h_2(v)}{2} \le \frac{2max\{h_1(v),h_2(v)\}}{2} \le max\{h_1(v), h_2(v)\} \le h^*(v)$$

כאשר המעבר האחרון נובע מכך ש h1 ו h2 עקביות ולכן הערך שלהן על כל צומת הוא לכל היותר שווה ל h*. כנדרש.

2. **סעיף 1.**

זוהי טענה לא נכונה. היוריסטיקה h* היא עקבית ולכן, הדוגמא הנגדית הקודמת זו דוגמא נגדית גם עבור סעיף זה. וזה נובע מכך ש $h^* + h^*$ לא תהיה קבילה ולכן גם לא עקבית (כי עקבית גוררת קבילה).

סעיף 2.

זוהי טענה נכונה. יהיו h1 ו h2 עקביות. לכן לכל שני מצבים v, v' עוקבים מתקיים:

$$\frac{(h_1+h_2)(v)}{2} - \frac{(h_1+h_2)(v')}{2} = \frac{h_1(v)-h_1(v')}{2} - \frac{h_2(v)+h_2(v')}{2}$$

אבל היוריסטיקות עקביות ולכן לפי הגדרה:

$$\frac{h_1(v)-h_1(v')}{2} - \frac{h_2(v)+h_2(v')}{2} \leq \frac{cost(v,v')}{2} + \frac{cost(v,v')}{2} = cost(v,v')$$

כלומר:

$$\frac{(h_1+h_2)(v)}{2} - \frac{(h_1+h_2)(v')}{2} \leq cost(v,v')$$

ולכן זוהי יוריסטיקה עקבית. כנדרש.

4. זוהי יוריסטיה קבילה כי אם המסלול הקרוב ביותר לצומת כלשהי עובר דרך פורטל אז המחיר האופלטימלי מצומת זו הוא לפחות כמו מחיר שימוש בפורטל, אחרת המחיר הוא לפחות כמו מחיר מנהטן כי המחיר של כל צעד במפה הוא לפחות 1 ואפשר ללכת רק בכיוונים מקבילים לצירים ולכן בכל מקרה המחיר האופטימלי מצומת כלשהי לצומת יעד הוא לפחות כמחיר היוריסטיקה.

5. זוהי יוריסטיקה עקבית. אפשר לקבל זאת מסתכלות על שני צמתים עוקבים v, v' וחלוקה למקרים לפי $h(v)$ ולפי הפעולה שנקטנו במעבר בין v ל v'

לדוגמא אם $h(v)$ זה מחיר מנהטן והצעד שביצענו הוא אינו צעד של פורטל אז מתקיים:

$$h(v) - h(v') \leq 1$$

ומחיר כל הפעולות הוא לפחות 1. אם הצעד שביצענו הוא צעד של פורטל, ומכיוון ש $h(v) \leq 100$ (המחיר של פעולת פורטל אז:

$$h(v) - h(v') \leq 100$$

(ו 100 זה מחיר הצעד). באותו אופן ניתן לראות בקלות לכל המקרים האחרים (לדוגמא אם $h(v)$ הוא מחיר פעולת פורטל).

שאלה 6.

1. האלגוריתם הוא שלם. מרחב המצבים בלוח שבמחברת הוא כאמור סופי ולכן האלגוריתם בהכרח ימצא פתרון ובמקרה הגרוע פשוט יאלץ לעבור על כל האפשרויות.
   האלגוריתם לא קביל. הרצת האלגוריתם הגרידי החזירה עלות גבוהה יותר מעלות UCS אשר מחזיר את העלות האופטימלית.

2. יתרון של Greedy Best first Search :
   האלגוריתם Greedy Best first Search לא מוגבל בגודל החזית לעומת לעומת Beam search שכן (הוא יזרוק משם צמתים במידה ויחרוג מהגודל המקסימלי) ולכן Greedy Best first יכול למצוא מסלול טוב יותר ש Beam search היה מפספס. בנוסף, לכן גם, Beam search אינו שלם, כי הוא עלול לזרוק צומת עם ערך h גבוהה אבל זה הצומת היחיד שממנו ניתן להגיע למצב מטרה.
   חיסרון של Greedy Best first Search :
   בגלל שהוא לא מוגבל בגודל החזית, אז הוא יותר בזבזני במקום מאשר Beam serach שחסכוני יותר במקום.

שאלה 7.

1. בקוד.

2. איימי צודקת מכיוון שבכל רגע נתון, עבור כל מצב v שכבר ביקרנו בו יתקיים ש: $f'(v) = \frac{f(v)}{2}$ ולכן, יחס הסדר בין ערכי f' על המצבים יהיה זהה ליחס הסדר בין ערכי f על המצבים. כלומר אם נסתכל על האיטרציה ה i של שני האלגוריתמים, אז לכל שני מצבים v1, v2 שכבר הסתכלנו עליהם יתקיים:
$f(v_1) < f(v_2) \Leftrightarrow f'(v_1) < f'(v_2)$ וגם
$f(v_1) = f(v_2) \Leftrightarrow f'(v_1) = f'(v_2)$ ולכן, סדר פיתוח הצמתים יהיה זהה בשני האלגוריתמים ולכן, גם המסלול וגם עלות המחיר המוחזר יהיו זהים.

3. בקוד.

4. יתרון של ID-A* ביחס ל A* הוא שבאלגוריתם ID-A* סיבכויות מקום היא לינארית ביחס לאורך המסלול לצומת מטרה. לעומת זאת, ב A* הסיבוכיות מקום כמספר הצמתים שנוצרו. חיסרון של ID-A* ביחס ל A* הוא שבאלגוריתם ID-A* נפתח מצבים מספר פעמים מבלי לדעת שכבר פיתחנו אותם ולעומת זאת ב A* אנחנו נמנעים מפיתוחים חוזרים (כאשר אין שיפור) כלומר ב ID-A*, נפתח יותר צמתים. נעדיף להשתמש ב ID-A* כאשר חשוב לנו לחסוך במקום (לדוגמא כאשר גרף המצבים הוא גדול מאוד) ונעדיף להשתמש ב A* כאשר חשוב לנו יותר מהירות מציאת הפתרון.

5. יתרון של A*-epsilon ביחס ל A* הוא שבאלגוריתם A*-epsilon אנחנו יכולים להוסיף פרמטר נוסף כדי לבחור את המצב הבא לפיתוח. באלגוריתם זה אנחנו מאפשרים התפשרות על איכות הפתרון כדי להרוויח ערך אחר (לדוגמא השגת פתרון מהיר). מצד שני, חיסרון של A*-epsilon ביחס ל A* הוא שלא מובטח לנו פתרון אופטימלי אלא קירוב כלשהו אליו. נעדיף להשתמש ב A*-epsilon כאשר אנחנו מוכנים להתפשר על האופטימליות של הפתרון כדי להשתפר במדד אחר לדוגמא במהירות מציאת פתרון, ונעדיף להשתמש ב A* כאשר לא נהיה מוכנים להתפשר על איכות הפתרון ונהיה מוכנים להשקיע את המשאבים שדורש A* כדי למצוא אותו.

שאלה 8.

.1

התוצאות שהתקבלו הן:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	map	DFS-G cos	DFS-G nun	UCS cost	UCS num (	A* cost	A* num of	W-A* (0.3)	W-A* (0.3)	W-A* (0.7)	W-A* (0.7)	W-A* (0.9)	W-A* (0.9)	num of expanded nodes		
2																
3	map12x12	121	33	87	97	87	92	87	94	87	89	89	26			
4																
5	map15x15	181	47	106	167	106	167	106	167	106	150	129	44			
6																
7	map20x20	282	57	175	309	175	308	175	308	175	298	202	56			
8																
9																

.2 התוצאות כן תואמות לציפיות שלנו.

נראה את זה במפורש.

DFS-G

- **עלות:**
  - גבוהה יותר מאשר בשאר האלגוריתמים, וזה נובע מכך שאלגוריתם זה לא מתחשב במחיר הקשתות או בהיוריסטיקה.
- **מספר הצמתים המפותחים**:
  - יחסית נמוך לאלגוריתמים האחרים, זה הגיוני כאשר יש הרבה מסלולים לפתרון כי אז האלגוריתם ימצא מהר מסלול כזה מבלי לחזור הרבה אחורה (כי הוא מחפש לעומק). מספר נמוך של צמתים מפותחים הגיוני כי אכן נראה שיש הרבה מסלולים לצומת מטרה.
- **יוריסטיקה מיודעת יותר**:
  - לא רלוונטי כי האלגוריתם לא משתמש ביוריסטיקה.
- **מסלול מוחזר:**
  - האלגוריתם יחזיר את המסלול הראשון שהוא מצא לצומת מטרה.

UCS

- **עלות:**
  - מוצא פתרונות אופטימליים (העלות הנמוכה ביותר האפשרית) כי זהו אלגוריתם שלם וזה אכן עומד בצפיות שלנו.
- **מספר הצמתים המפותחים**:
  - גבוה יחסית לשאר האלגוריתמים. דבר זה עומד בצפיות שלנו כי אלגוריתם זה לא משתמש ביוריסטיקה שתעזור לו לעבוד מהר יותר ולפתח פחות צמתים
- **יוריסטיקה מיודעת יותר**:
  - לא רלוונטי כי האלגוריתם לא משתמש ביוריסטיקה.
- **מסלול מוחזר:**
  - האלגוריתם מחזיר מסלול הקל ביותר (כלשהו) מצומת ההתחלה לצומת מטרה.

A*

- **עלות:**
  - מוצא פתרונות אופטימליים וזה אכן עומד בצפיות שלנו כי היוריסטיקה שבה אנו משתמשים קבילה ולכן לפי ההרצאות האלגוריתם קביל.
- **מספר הצמתים המפותחים**:
  - מפתח קצת פחות צמתים מ UCS (לא מובהק). אנחנו אכן ציפינו שאלגוריתם זה לא יפתח יותר צמתים מ UCS כי היוריסטיקה אמורה לגרום לאלגוריתם לבחור צמתים שיותר קרובים לצומת מטרה אך במקרה זה זה לא מבוהק אבל כן רואים שאנחנו לא מפתחים יותר צמתים מאשר UCS.
- **יוריסטיקה מיודעת יותר**:
  - אמורה לגרום לאלגוריתם לבחור לפתח צמתים כי היא תעריך את המרחק מצומת מטרה טוב יותר מה שיגרום לאלגוריתם להגיע לצומת מטרה מהר יותר.
- **מסלול מוחזר:**

- האלגוריתם מחזיר מסלול הקל ביותר (כלשהו) מצומת ההתחלה לצומת מטרה.

W-A*

- **עלות:**
  - אפשר לראות שככל שאנחנו נותנים יותר משקל ליוריסטיקה (ופחות למשקלים) אז העלות המוחזרת עולה (ב W-A*0.3 ו ב W-A*0.7 העלות שווה אבל היא לא יורדת). דבר זה הגיוני וצפוי כי כפי שלמדנו, ככל שנותנים יותר משקל ליוריסטיקה אז אנחנו מרוויחים אלגוריתים מהיר על חשבון אופטימליות הפתרון.
- **מספר הצמתים המפותחים**:
  - כפי שניתן לראות מהתוצאות, ככל שנותנים יותר משקל ליוריסטיקה, אז האלגוריתם מפתח פחות צמתים. זה הגיוני וצפוי כיוון שככל שנותנים יותר משקל ליוריסטיקה אז האלגוריתם יעדיף לפתח צמתים שאנו מעריכים שהם קרובים יותר למטרה ולכן הוא ימצא מסלול למטרה מהר יותר ויפתח פחות צמתים.
- **יוריסטיקה מיודעת יותר**:
  - אמורה לגרום לאלגוריתם לפתח פחות צמתים כי היא תעריך את המרחק מצומת מטרה טוב יותר מה שיגרום לאלגוריתם להגיע לצומת מטרה מהר יותר, אך אין הבטחה למשקל פתרון נמוך יותר.
- **מסלול מוחזר:**
  - האלגוריתם יעצור כאשר הוא ישלוף צומת מטרה מהחזית ויחזיר את המסלול על אליו. אם המשקל של היוריסטיקה גדול מ 0.5 אז אין הבטחה על איכות הפתרון (גם כאשר היוריסטיקה קבילה).

שאלה 9.

1. יש להגדיר את מצבי מרחב החיפוש כאוסף כל הסידורים האפשריים של המילים. כלומר כל מצב הוא סידור אפשרי של n המילים.

2. מספר המצבים הוא מספר הסידורים האפשריים של n המילים ולכן יש !n מצבים (כי אין אף מילה שחוזרת פעמים).

3. כן, מאחר שלכל מצב שאינו מצב שבו כל המילים במקומן (כלומר פונקציית הערך היא n) אז ניתן לשפר את פונקציית הערך בלפחות 1. נסביר זאת. אם מצב כלשהו אינו מצב של סידור מושלם של המילים אז קיימת לפחות מילה אחת (נסמנה a) שלא במקומה. נסתכל על המילה w שנמצאת במקום הנכון של a. מתקיים שגם w לא במקום הנכון שלה (כי w במקום של a ואין שתי מילים זהות) ולכן אם נחליף בין a ל w אז נגיע למצב שבו פונקציית הערך השתפרה בלפחות 1. ולכן קיבלנו מצב טוב יותר (יותר קרוב לפיתרון) מהאחד בו היינו. ומכיוון שפונקציית הערך חסומה על ידי n (כי עבור ערך זה אז כל המילים נמצאות במקומן) אז האלגוריתם ימצא פתרון אחרי לכל היותר n צעדים.
4. **סעיף 1.** כן, מאותו נימוק כמו בסעיף 3, בכל מצב נוכל לשפר את פונקציית הערך לפחות ב 1. ולכן לעולם לא נבחר בצעד שאינו משפר ונמצא פתרון אחרי לכל היותר n צעדים.

   סעיף 2. אלון טועה מכיוון שלפי הסעיף הקודם, אז לעולם לא נבצע צעד שלא משפר ולכן שני האלגוריתמים יפעלו אותו דבר ולכן אין עדיפות SAHC with sideways steps.

5. אלגוריתם זה ימצא פתרון מאותם נימוקים קודמים. מכל מצב נוכל לבצע פעולה שתשפר את פונקציית הערך בלכל הפחות 1. ולכן גם אם נבחר פעולה רנדומלית כלשהי שמשפרת את פונקציית הערך (שזה בדיוק מה שעושה

האלגוריתם) אז נשפר בלכל הפחות 1 ולכן גם אם נשתמש באלגוריתם זה, אז נגיע לפתרון תוך לכל היותר n צעדים.