**ממן 15 מבוא לבינה מלאכותית – בעיית המיסיונרים והקניבלים**

**דור עיידן**

1. בתוכנית זו השתמשתי באלגוריתמים BFS, IDDFS, GBFS ו- A\* על מנת לפתור את בעיית המיסיונרים והקניבלים - הבעיה עוסקת בשלושה מיסיונרים ושלושה קניבלים הנמצאים בגדה השמאלית של נהר. בגדה הזו מצויה סירה, שיכולה לשאת אדם אחד או שני אנשים. אנו מעוניינים להעביר את ששת האנשים לגדה הימנית של הנהר בעזרת הסירה. מסיבות מובנות, אין לאפשר, אפילו לרגע אחד, מצב שבו מספר הקניבלים גדול ממספר המיסיונרים באחת הגדות של הנהר. העברת הסירה מגדה לגדה איננה יכולה להתבצע בלי שיהיה בה לפחות אדם אחד.

באלגוריתם BFS השתמשתי במבנה הנתונים תור (queue) לצורך ניהול צמתי החזית (frontier), באלגוריתם IDDFS השתמשתי במבנה הנתונים מחסנית (stack) לצורך ניהול צמתי החזית ואילו באלגוריתמים GBFS ו- A\* השתמשתי בתור קדימויות (PriorityQueue) לצורך ניהול צמתי החזית. על מנת לבדוק שאין אנו מפתחים\חוקרים מצב שכבר פותח\נחקר (מהסיבה הפשוטה שאיננו רוצים לולאות אינסופיות), בתוכנית המצורפת לקובץ readme הנ"ל באלגוריתמים BFS, GBFS וב-A\* נעשה גם השימוש ברשימה בשם explored שתפקידה לעקוב אחרי המצבים שכבר נחקרו היות ובחרתי לממש את אותם אלגוריתמים שציינתי כחיפוש בגרף ולא בעץ. לעומתם, בחרתי לממש את אלגוריתם IDDFS כחיפוש בעץ, כלומר באלגוריתם זה אין מניעה לפתח צומת שכבר פותח בעבר (גם אם פותח עבור depth=x עומק כלשהו). חשוב לציין שהשימוש ב"explored" ב-IDDFS הוא רק לשם החזרת כמות הצמתים שפותחו, (שכן אלה הם דרישות התרגיל) ולא לשם מניעה מפיתוח חוזר של צומת שפותח בעבר.

בנוסף, ניתן לראות בתוכנית המוגשת את הפונקציה getValidNextStates(currentState) אשר מקבלת את המצב הנוכחי (או במילים אחרות את המצב שכרגע חוקרים) ומחזירה רשימה של צמתים חוקיים (חוקי הכוונה עומדים בתנאי הבעיה שאותה התבקשנו לפתור) שאליהם ניתן להגיע מהמצב הנוכחי. כמו כן פונקציה זו משתמשת בפונקציה IsValid(newState) שמקבלת את המצב שממנו ניתן להגיע מ-currentState ומחזירה האם מצב זה חוקי, כלומר עומד בתנאי השאלה.

כמו כן, ניתן לראות כי הפונקציה ()IDFS המממשת את אלגוריתם IDDFS משתמשת בפונקציה depthLimitedSearch(MaxDepth) על מנת לבצע את הבדיקה לעומק כל פעם עד ל-maxDepth שהוגדר בתוכנית הנוכחית כ-20.

1. בחרתי לייצג מצב כווקטור של שלושה ערכים, כאשר המקום הראשון מייצג את מספר המיסיונרים הנמצאים בגדה השמאלית, המקום השני מייצג את מספר הקניבלים שנמצאים בגדה השמאלית ובמקום השלישי נמצא ערך בוליאני אשר נותן לנו אינדיקציה אם הסירה כעת נמצאת בגדה השמאלית או בגדה הימנית (ערך 1 = גדה שמאלית, ערך 0 = גדה ימנית של הנהר). כל מצב באשר הוא (בלי קשר לאם הסירה נמצאת כרגע בגדה השמאלית או הימנית) מייצג את מספר הקניבלים והמיסיונרים הנמצאים בגדה השמאלית תמיד.

המצב ההתחלתי יהיה (3,3,1) ואילו המצב הסופי (מטרה) יהיה (0,0,0). (0,0,0) משמעותו הוא "נשאר לנו 0 קניבלים ו-0 מיסיונרים להעביר אל הגדה השמאלית (0) ואנחנו גם נמצאים בגדה השמאלית" – כלומר העברנו את כל הקניבלים ואת כל המיסיונרים מהגדה הימנית אל הגדה שמאלית.

המעברים הוגדרו כרשימה של פעולות אפשריות של העברת קניבלים או\ו מיסיונרים מצד אחד לצד השני. המעבר בין מצב למצב נעשה באמצעות חיסור מערכים בעזרת numpy אם אנחנו נמצאים בגדה השמאלית וחיבור מערכים אם אנחנו נמצאים בגדה הימנית. לדוגמה אם אנחנו במצב ההתחלתי (3,3,1) ואנו כעת מחשבים את המצב הבא במידה ואנחנו מעבירים קניבל אחד לגדה הימנית, אזי נתאר את המעבר כ- (0,1,1( ונבצע חיסור בין המצב (3,3,1) למעבר (0,1,1) 🡨 נקבל את המצב (3,2,0) שבעצם מתאר כי אנחנו בגדה הימנית ובגדה זו יש 3-3 = 0 מיסיונרים ו- 3-2=1 קניבלים ובעקבות סימטריה נוכל להסיק גם כי 3-0 =3 מיסיונרים בגדה השמאלית ו- 3-1 = 2 קניבלים גם כן בגדה השמאלית.

1. עבור האלגוריתמים GBFS ו- A\* בחרתי ביוריסטיקה h(n) שהיא מחשבת את ממוצע האנשים (קניבלים ומיסיונרים) הנמצאים בגדה השמאלית חלקי 2. ידוע כי ב-GBFS f(n) = h(n). עבור A\* g(n) מייצגת את מספר הקשתות שלקח לנו להגיע עד לצומת n מצומת ההתחלה, וסה"כ ב-A\* היוריסטיקה שנבחרה היא f(n) = h(n) + g(n).

קבילות ועקביות היוריסטיקה:

היוריסטיקה h(n) שנבחרה, קבילה ועקבית : ראה הוכחה שמצורפת בדף לקובץ ה-zip.

1. אופטימליות ושלמות:

BFS - שלם ואופטימלי:

אלגוריתם ה-BFS שהוצע כחלק ממטלה זו שלם כי b (דרגת ההסתעפות) היא סופית – מספר המעברים\הפעולות הישימות במצב חסום ע"י מספר קבוע, שכן ניתן לראות בתוכנית את מבנה הנתונים possibleMoves הנמצא בפונקציה getValidNextStates(currentState) שמספר האיברים בו הוא קבוע (מספרים האיברים בו הוא 5 🡨 b=5). אלגוריתם ה-BFS המוצע הוא גם אופטימלי שכן ניתן לראות כי מחיר מסלול הוא פונקציה לא יורדת של עומק הצומת – עבור כל הפעולות\המעברים (הקשתות) אותו מחיר וניתן להתייחס על מחיר של כל קשת\מעבר כ-1.

IDDFS – שלם ואופטימלי:

אלגוריתם ה- IDDFS שמימשתי הינו שלם כמו אלגוריתם BFS, כאשר דרגת ההסתעפות b סופית (כבר ראינו כי b סופית לפי ההסבר על השלמות והאופטימליות של BFS) – אם יש צומת מטרה בעומק X אזי האלגוריתם ימצא אותו באיטרציה X. אלגוריתם ה-IDDFS המוצע הוא גם אופטימלי, גם כן בדומה לאלגוריתם ה-BFS, ניתן לראות כי מחיר מסלול הוא פונקציה לא יורדת של עומק הצומת (כמו כן, היות והשתמשנו בחיפוש עץ, האלגוריתם מבטיח שאם הפתרון בעומק המינימלי נמצא בעומק k הוא יימצא באיטרציה k מאחר שעד לאיטרציה זו האלגוריתם לא סורק בעומקים k ומעלה ובעומק k הצומת עם הפתרון תיסרק ותוחזר).

GBFS – שלם ואופטימלי (אופטימלי ב"מזל"):

אלגוריתם ה-GBFS שמימשתי הינו שלם מכיוון שמרחב המצבים של הבעיה שהתבקשנו לפתור במטלה זו הוא סופי וכן במימוש האלגוריתם מתבצעת בדיקה למניעת מצבים חוזרים, לכן מובטח שיימצא פתרון כאשר קיים פתרון כפי שניתן לראות בפלא התוכנית בסעיף 5 (ואנו יודעים שקיים פתרון לבעיית המיסיונרים והקניבלים). אלגוריתם ה-GBFS המוצע הוא גם אופטימלי ספציפית לבעיה הנוכחית (לעומת בעיות אחרות שבהן בכלל לא מובטח ש-GBFS ימצא פתרון אופטימלי) היות והוא מוצא פתרון של מסלול באורך מינימלי.

A\* - שלם ואופטימלי:

אלגוריתם A\* שמימשתי הינו שלם – בפלט התוכנית בסעיף 5' ניתן לראות כי האלגוריתם מצא מסלול למצב מטרה (0,0,0) (שלמות דורשת שיהיה מספר סופי של צמתים עם f(n) קטן או שווה ל-C\*. זה נכון אם לכל צעד יש מחיר חיובי גדול או שווה ל- ε, עבור ε כלשהו, וכן דרגת הסיעוף b סופי כפי שדנו קודם.) אלגוריתם A\* המוצע הוא גם אופטימלי היות ובסעיף הקודם הוכחתי כי הפונקציה היוריסטית h שנבחרה היא עקבית (מונוטונית) ולפי מדריך הלמידה בחיפוש גרף (כמו שהשתמשתי במימוש אלגוריתם זה) זה מבטיח לנו את אופטימליות האלגוריתם.

1. אופן הרצת התוכנית:

יש לטעון את התוכנית המצורפת (בעלת הסיומת .py) בקובץ ה-zip לאחת מסביבות העבודה הנמצאות במחשב שלך התומכות בשפת python (גרסה 3.7 ומעלה) ולהריץ אותה, או לחלופין ניתן להריץ משורת הפקודה מהמיקום בו שמרת את קובץ ה- .py במחשב שלך את הפקודה "python {"file\_name"}". כמו כן, אפשר לנסות גם לפתוח את קובץ ישירות באמצעות double click עליו.

**פלט התוכנית:**

