4.)בתרגיל זה מימשנו 3 אלגרתמים לפתרון בעיית 8puzzle , כאשר האלגורתמים שממשנו הינם DFS,BFS,A\* (בקוד הם נקראים DFSSearch,BFSSearch,AStarSearch ).

כמו כן ממימשנו אלגורתם A\* אם ממשק יורסטי (לממשק היה את היכולת להחזיר את ערך ה h ) כאשר ממשנו לממשק זה 3 אלגותמים יורסטים – 2 מוכרים ומפורסמים – ואחד שיפור של אחד מהם.

המימוש של הממשק היורסטי היה Manhattan ו hamming distanceבקוד וממימוש נוסף (בקוד הוא נקרא userHeuristic ) האלגורתם הזה נועד לענות על חסרון של שמצאנו בשני האלגורתמים הקודמים :

* ב Manhattan אתה מסוגל לדעת מה טיב המחרק של כל אלמנט בפאזל – האלגוריתם מחשב את המרחק מהמטרה –וסוכם את כל המספרים ) , ז"א הפקטור שישפיע הכי הרבה הוא המרחק מהמטרה האלגורתם לא מתחשב עד כמה צעדים יש לעשות הוא לוקח מרחק פיזי.
* אלגורתם השני hamming distance - פשוט מחשב כמה גורמים לא נמצאים במקום הבעיה עם אלגורתם זה שהוא לא מתחשב בכלל במרחקים אלה בכמה אלמנטים הם שגויים – ז"א יש אותו משקל לאלמנט שהוא רחוק מעט המטרה לעומת אלמנט שהוא קרוב מאוד.
* לכן האלגוירתם שאנו מציעים הוא עובד על עקרון של מרחק ומיקום הוא בדיקה של סכום של תת קוביות :
  + - סיכום קוביות : הפאזל שלנו מורכב קוביה 3\*3 אך בתוכה יש 4 תתי קוביות 2\*2 לכל קוביה יש סכום אשר מייצג את היחס בין כל ערך – ז"א בקוביית המטרה שלנו לדוגמא הסכום של הקוביה השמאלית עליונה הוא 12 (1+2+4+5) ועל הקוביה התחתונה ימנית הוא 18 (היצוג של החלק הריק אצלנו הוא -1 ביצוג קלאסי של 0 זה יהיה 19) , מהלך זה בצענו גם קובית מטרה וגם על מצב כל צומת בעץ (צומת נוכחית שחישבנו) .
    - בדיקת שוויון קוביות – בדיקה אם סכומי תתי הקוביות (מהפזל הנבדק) שונה מסכום תתי הקוביות בקובית המטרה – כך שהתוצאה החוזרת היא כמות סכומי הקוביות השונות.

כדי לתאר את התת אלגורתם המוצע נראה זאת באמצעות הנוסחה :

יחס הקוביות יאפשר לי לדעת עד כמה היחס אם רחוק מקוביית המטרה ,לא רק ברמת של קוביה המלאה אלה ברמה של סיכום תתי הקוביות שבתוכה.

יתרון של שיטה זו בדרך זו נקבל החלטה יותר טובה והערך שנחזיר יהיה בהרבה יותר מדוייק (הסיכוי לטעות).

אך עם כל זאת התהליך צפוי להיות איטי יותר מה שיכול להוריד זמן הריצה.

לכן ביצענו 18 ריצות כך לכל קובץ הרצנו BFS ואז A\*במימוש שלנו פונקצית הmain מריצה באופן אוטומטי את 3 יורסטיקות ואלו הם התוצאות :

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Example 1 | | | Example 2 | | | Example2 | | |
|  | Time(milisec | | steps | time | |  | time | |  |
| BFS | 33 | | 4 | 103 | | 7 | 400 | | 8 |
| A\* Manhattan | 14 | 5 | | 14 | 5 | | 14 | 9 | |
| A\* hamming | 1 | 5 | | 5 | 8 | | 6 | 9 | |
| A\* - cubes (my) | 1 | 5 | | 5 | 8 | | 18 | 9 | |

מסקנה

נובע כי A\* אכן הרבה יותר יעיל היות שאנו ממשים שתי מתודות אחת שעושה אומדן ושניה שמערכיה לפי אלגורתם יורסטי ולכן אפשר לנתח שA\* תמיד היה יעיל יותר ומהיר יותר (פרט למקרה הראשון שBFS מצא עומק רחד פחות) כמו כן היות שפונקצית המטרה עוזרת למצוא את הצומת היעילה ביותר (הקצרה ביותר למטרה – הינה f ) היא חוסכת ביקור בתת עץ .

המתודתה היורסטית הטובה ביותר שאנו רואים היא hamming היות שהיא פשוטה יותר היא תמיד מצאה את הערכה בצורה המהירה יותר.

פרטים על מימוש :

ממשנו את 8puzzle כמטריצה של 3\*3 (האוביקט puzzle ) שמסוגל ליצר צומת נוספת במידה ואפשר לבצע מהלך (ימינה ,שמאלה ,למטה למעלה) כמו כן הוספנו יצוג של string שמייצג את המטריצה.

לממשק Isearch ישנו 3 מימושים שונים DFSSearch,BFSSearch,AStarSearch שכל אחד יורש מ Asearch (מכיל את המתודות המשותפות ) , בנוסף יש לנו את ממשק Iheuristic שממשים אותו כל אחד מהיורסטיקות (ראה ManhattanPrio, HammingPrio, UserHeuristicPrio) .

אופן הפעלה :

פשוט להיות בתקיית bin ולהקליד:

Java CLI.Search <נתיב קובץ דוגמא> <אלגורתם >