פרויקט גמר אופטימיזציה

מגישים:

יאיר חנינה 211513890

מרים קורן 318514536

רקע:

בהינתן גרף ממושקל (משקלים אי שליליים) ניתן למצוא את המסלול הקצר ביותר מנקודת התחלה לנקודת יעד בגרף.

דייקסטרה פותר את בעית המינימיזציה של מציאת כל המסלולים עם ערך מינימלי מצומת ההתחלה לכל צומת אחר שמחובר אליו בגרף.

האלגוריתם Dijkstra:

- נסמן ב S את נקודת ההתחלה.
- עבור כל צומת, מסומן האם ביקרו בו או לא וגם את מרחקו מנקודת S (בהתחלה כל הצמתים מסומנים שלא ביקרו בהם עם מרחק אינסוף).
- כל עוד נותרו צמתים שלא ביקרנו בהם:
  - מסמנים את הקודקוד הנוחכי (X) כקודקוד שביקרנו בו.
  - עבור כל קודקוד (Y) שכן של X ועדיין לא ביקרנו בו:
    - מעדכנים את המשקל של Y להיות הערך המינימלי בין משקלו הנוכחי, לבין משקל הקשת המחברת בין X לבין Y ועוד המשקל של המסלול מצומת S עד ל-X.
  - נבחר קודקוד X חדש תור הקודקוד שמרחקו בשלב הזה מצומת המקור S הוא הקצר ביותר מבין כל הקודקודים בגרף שטרם ביקרנו בהם.

בעיית האופטימיזציה היישומית:

בהינתן מבוך עם דלתות, נרצה למצוא את הדרך הטובה ביותר לצאת ממנה בזמן הקצר ביותר כאשר הדלתות יכולות להיפתח ולהיסגר באופן אקראי. לצורך הפשטות, בחרנו לקחת שלוש דלתות שנסגרות בכל איטרציה, כלומר צמתים.

הגדרות:

- נסתכל על הבעיה כגרף כאשר הקודקודים מסמלים דלתות
- קשתות הן הדרך שמחבר בין שתי דלתות
- משקל הקשתות נגדיר כזמן שלוקח להגיע מדלת אחת לדלת אחרת.

הבנה של הבעיה:

ניתן לראות את הבעיה כמציאת מסלול מינימלי בגרף אך עם הוספת אילוצים(דלתות סגורות או פתוחות) מה שאין באלגוריתם הרגיל של דייקסטרה.

יש עוד אלגוריתמים למציאת מסלול קצר ביותר בגרף כמו בלמן פורד. אך בלמן פורד הוא אלגוריתם שיכול לכלול משקלים שליליים מה שלא נצרך בבעיה שלנו (אין זמן שלילי) ודורש זמן ריצה גדול יותר מאשר דייקסטרה.

יישום האלגוריתם בהתאמה לבעיה:

הבעיה הינה משחק הגעה לצומת מטרה מהצומת הנוכחי, בכל מהלך של השחקן משתנות הדלתות הסגורות באופן אקראי.

יוצרים רשימה של הצמתים במסלול שהשחקן הלך ומשתנה הסופר את כמות הזמן(המשקל) עד להגעה לצומת היעד.

עד הגעה לצומת היעד בכל מהלך הפונקציה play:

- יוצרים רשימה של צמתים שהיו במסלול
- יוצרים משתנה שיספור את הזמן
- בלולאה עד שנגיע לצומת המטרה או שלא תהיה אפשרות לנוע ואז המשחק יסתיים:
- הפונקציה מקבלת מפונקציה get_random_blocked_list, רשימה חדשה של דלתות סגורות.
- מריצה את Dijkstra לפיהן עם הצומת הנוכחי.
  - בלולאה כל עוד יש צמתים שלא עברנו עליהם:
  - יש באלגוריתם רשימה פתוחה של צמתים להגיע אליהם ולוקחים בכל פעם צומת חדש שהוא עם הערך הגעה מינימלי אליו. רשימה סגורה של צמתים שביקרנו בהם. ו – blocked_list רשימת הדלתות הסגורות.
  - מוסיפים את הצומת הנוכחי לרשימה closed_list ומורידים אותו מהרשימה open_list.
  - בנוסף לכך אנחנו יצרנו מילון אשר מכיל בתור מפתח את הצומת שמגיעים אליו והערך שלו הוא רשימה המכילה את העלות להגיע אליו והאבא שלו. העלות להגיע אליו בהתחלה הינה: sys.maxsize והאבא שלו הינו None.
  - מחשבים את הזמנים המינימליים שהוא יכול להגיע לשכנים שלו ומעדכנים בטבלת הזמנים(המילון) כשצריך.
  - כאשר מחפשים את הצומת הבא המינימלי מחפשים את הערך המינימלי בטבלה שגם לא נמצא בclosed_list ולא ב – blocked_list.
  - לאחר לולאה מחזירים את הטבלה עם הערכים המינימליים, יכול להיות שיהיה צומת שלא ניתן להגיע אליו ואז האבא שלו הוא None וכמות הזמן להגיע אליו היא מקסימלית, תיאורטית אינסוף.
- מהטבלה עם כמות הזמנים המינימליים הפונקציה יוצרת את המסלול האופטימלי לצומת המטרה.
- מוסיפה את הזמן של הקשת לזמן הכללי של המסלול שהשחקן הלך.
- מקדמת את הצומת הנוכחי לצומת הבא במסלול.
- מוסיפה את הצומת הבא שהשחקן הלך אליו לרשימת הצמתים שהיו במסלול
- בודק האם הצומת הינו צומת המטרה שלנו, אם כן:
- מדפיסה את העלות והמסלול עצמו, אחרת ממשיכים בלולאה

דוגמא:

מתחילים מצומת 0

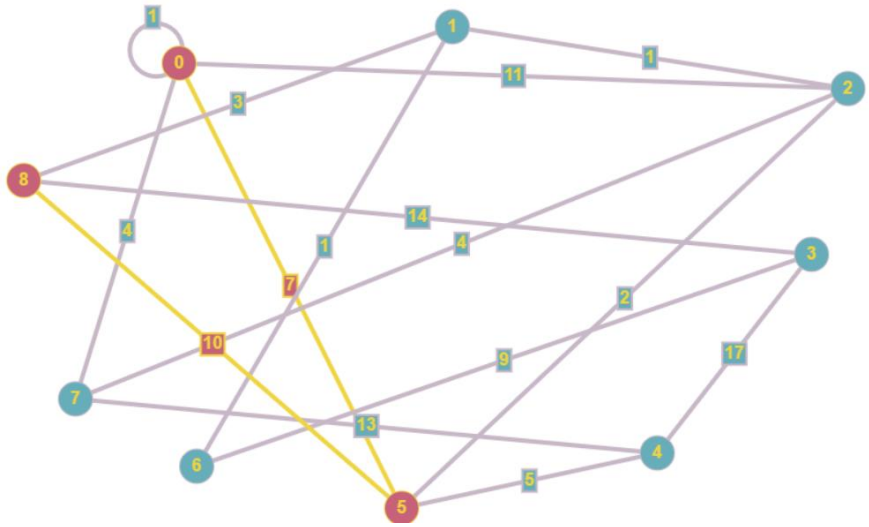
מטרה: 8

רשימת הדלתות הסגורות הן: [1,3,4]

קיבלנו את המסלול האופטימלי הבא: [0,5,8]

מתקדמים ל – 5

המשקל הכולל הינו 7



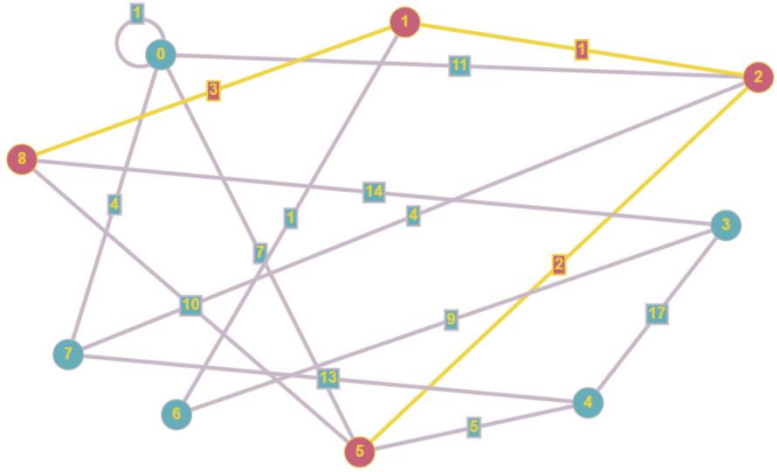
צומת 5

רשימת הדלתות הסגורות הן: [4,6,7]

קיבלנו את המסלול האופטימלי הבא: [5,2,1,8]

מתקדמים ל – 2

המשקל הכולל הינו 9



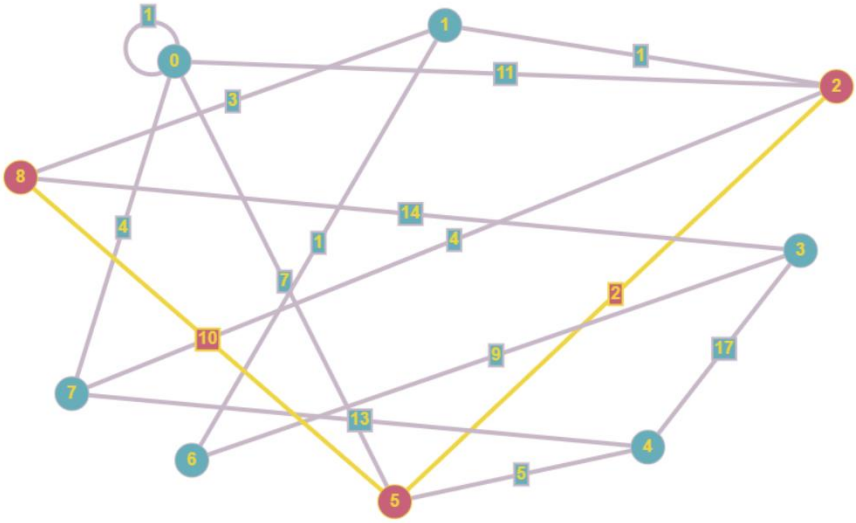
צומת 2

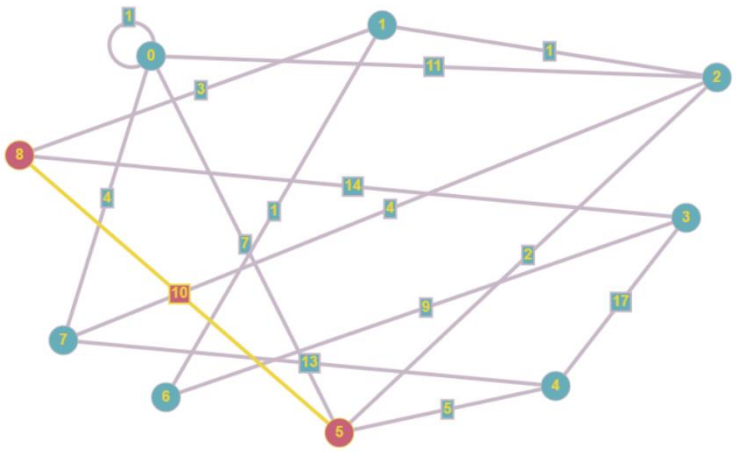
רשימת הדלתות הסגורות הן: [1,7]

קיבלנו את המסלול האופטימלי הבא: [2,5,8]

מתקדמים ל – 5

המשקל הכולל הינו 11





צומת 5

רשימת הדלתות הסגורות הן: [0,1]

קיבלנו את המסלול האופטימלי הבא: [5,8]

מתקדמים ל – 8

המשקל הכולל הינו 21

לבסוף עברנו בצמתים: [0,5,2,5,8]

```
This is S:  5
This is the blocked nodes:  [0, 1]
{0: [2147483647, None], 1: [2147483647, None], 2: [2, 5], 3: [22, 4], 4: [5, 5], 5: [0, None], 6: [31, 3], 7: [6, 2], 8: [10, 5]}
there is a way to the goal, choosing the best node to get there
------------
the path:  [8]
the cost to move from 5 to 8 is 10
got to the goal with cost:  21
The path was:  [0, 5, 2, 5, 8]
```

הסבר הקוד:

הקוד מחולק לשלושה קבצי פייתון Graph.py main.py Game.py אשר מצורפים בעבודה

מחלקת graph:

יש בה את כמות הקודקודים ואת הגרף עצמו שקיבלה כקלט שהוא מטריצה. ניתן להפעיל בה את פונקציית Dijkstra על הגרף שיש לנו. בנוסף ישנן פונקציות עזר בעבור מציאת הצומת הבא ב – Dijkstra והדפסה.

מחלקת Game:

מכילה את משתנה מסוג מחלקת graph, הצומת ההתחלתי, צומת המטרה שלנו ורשימת הדלתות הסגורות.

בנוסף יש בה את הפונקציה: play(self) שבה מתחילים את משחק ההגעה לצומת המטרה מהצומת הנוכחי. יוצרת את המסלול למטרה בכל איטרציה.

- מקבלת רשימה חדשה של דלתות סגורות.
- מריצה את Dijkstra לפיהן עם הצומת הנוכחי.
- יוצרת את המסלול האופטימלי לצומת המטרה.
- מוסיפה את המשקל למשקל הכללי של המסלול שהשחקן הלך.
- מקדמת את הצומת הנוכחי לצומת הבא במסלול.
- מוסיפה את הצומת הבא שהשחקן הלך אליו לרשימת הצמתים.

פונקציית main:

יוצרים את הקלט ידני שם ומריצים את התוכנית.