

המחלקה למתמטיקה שימושית

חקירת משחק האורות

מנחה:

אלכס גולוורד

מאת:

ולדיסלב ברקנס

8 באפריל 2022

תוכן העניינים

2	1 הקדמה
3	2 תאור של המשחק
3	2.1 תאור גרפי של המשחק
4	2.2 סוגיות בהן נעסוק בפרויקט
4	2.3 תיאור משחק על גרף
5	2.4 השוואה בין משחק על לוח למשחק על גרף
6	3 אלגוריתם למציאת פתרון
7	3.1 אלגוריתם שמבוסס על מטריצת שכנויות
11	3.2 אלגוריתם שמבוסס על מילוי עקבי של שורות
17	3.3 השוואה בין שתי השיטות למציאת פתרון
17	3.4 דיון לגבי משחק על גרף
18	4 קיום פתרון ומספר הפתרונות עבור משחק על גרף
18	4.1 הוכחת קיום פתרון על גרף
22	4.2 מספר הפתרונות עבור כל גרף
24	5 פתרון אופטימלי עבור לוחות מלבניים
25	5.1 הוכחת אי קיום לפתרון אופטימלי על לוחות ששני הממדים גדולים מ 4×4
27	5.2 אלגוריתם למציאת פתרון אופטימלי
29	6 נספחים
29	6.1 יצירת מטריצת שכנויות
31	6.2 מציאת פתרון
31	6.3 אלגוריתם מבוסס על מילוי עקבי
33	6.4 השוואה בין שתי שיטות
35	6.5 מציאת פתרון אופטימלי
36	6.6 מספר פתרונות על לוח
37	6.7 כל הפתרונות עבור לוח נתון

1 הקדמה

פרויקט זה הינו פרויקט סוף של סטודנט במחלקה למתמטיקה שימושית. הפרויקט חוקר את משחק האורות המטרה המקורית של הפרויקט היתה למצוא פתרונות למשחק, אך במהלך המחקר העלנו שאלות נוספות. נציג שתי שיטות למציאת פתרון של המשחק, בתחילה השיטות נראו שונות אבל בפרויקט הראינו דמיון ביניהם. דבר מרכזי נוסף שעסקנו בו הוא בחיפוש פתרונות אופטימלים, מהו פתרון אופטימלי נגדיר בהגדרה 5.1, פתרונות אילו הם מועטים וכן הוכחנו את כולם.

במהלך הפרויקט ראינו שבמשחק המתחיל כאשר כל הנורות דלוקות קיים לפחות פתרון אחד, תופעה זו העסיקה רבות את הפרויקט ובסיומו מצאנו הוכחה לקיום התופעה.

עבודה סוף זו הייתה מהנה עבורי אני מודה למחלקה למתמטיקה שימושית, במיוחד לאלכס גולוורד על הזדמנות לעשות עבודה מרתקת שכזה. עבודה זה לימדה אותי המון ונתנה לי את האומץ להשתמש בכלים שלמדתי במהלך התואר.

2 תאור של המשחק

משחק האורות, בלועזית Lights Out, מתקיים על לוח משבצות מלבני. כל משבצת יכולה להיות באחד משני מצבים, נקרא להם דלוק וכבוי. כאשר משתמשים בשמות האלו מתכוונים שבכל משבצת יש נורה והיא יכולה להיות דלוקה או כבויה. במצב התחלתי כל הנורות כבויות. יש לנו לוח בקרה שמאפשר בכל שלב של המשחק ללחוץ על משבצת ולשנות את מצב הנורה, אם היא דלוקה אז ניתן לכבות אותה ואם היא כבויה אז ניתן להדליק אותה. לוח הבקרה בנוי בצורה כזאת שכאשר מתבצעת לחיצה על משבצת אז מצבה של הנורה משתנה, בנוסף משתנים גם מצבם של הנורות הסמוכות לה. שתי נורות נקראות סמוכות אם הן נמצאות במשבצות בעלות צלע משותפת. המטרה של המשחק היא לעבור ממצב התחלתי שבו כל הנורות כבויות, למצב בו כל הנורות יהיו דולקות.

הערה 2.1: מצבם התחלתי של נורות המשחק אינה משנה את תוצאות המשחק.

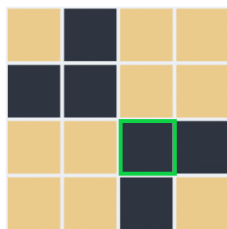
מטרתה של ההערה להדגיש כי מטרתו של המשחק היא להעביר את הלוח ממצב אחד בו נמצאים כל הנורות למצב האחר, וכי אין השפעה למראה של מצבים.

2.1 תאור גרפי של המשחק

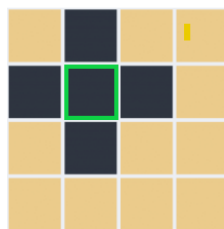
באיור הבאה נגדיר: מצב התחלתי הוא מצב בו כל נורות צהובות. מצב סופי הוא מצב בו כל נורות שחורות. לחיצה על משבצת תסומן על ידי צביעת גבולותיה בירוק.

איור 1: הסבר שינוי מצב הלוח לאחר לחיצה

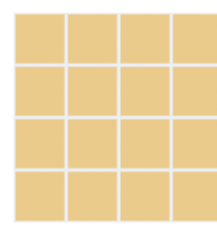
(ג) לוח לאחר שני לחיצות



(ב) לוח לאחר לחיצה בודדת



(א) לוח במצב התחלתי



פירוט: באיור 1A מתואר מצב התחלתי. באיור 1B ניתן לראות את השפעה של לחיצה על משבצת שמסומן בירוק. באיור 1G ניתן לראות השפעה של לחיצה על משבצת שמסומן בירוק.

לשם הבנה מומלץ לנסות את המשחק, כפי שנאמר "עדיף לראות פעם אחת, מאשר לשמוע מאה פעמים" או במקרה שלנו לשחק. את המשחק אפשר לשחק בקישור הבאה:

<https://www.geogebra.org/m/JexnDJpt#chapter/301822>

האתגר במשחק הוא שאין אסטרטגיה גלויה לכן, במשחקים רבים מנסים להגיע למצבים שפתרון כבר ידוע.

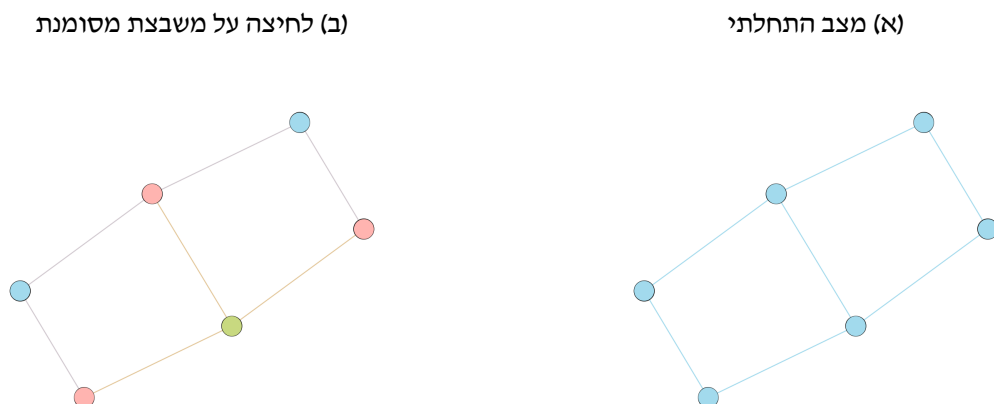
2.2 סוגיות בהן נעסוק בפרויקט

1. תיאור ודיון בשני אלגוריתמים למציאת פתרון המשחק.
 2. הוכחה לקיום פתרון המשחק לכל לוח $m \times n$.
 3. הרחבה של משחק על לוח למשחק על גרף.
 4. נעסוק במספר הפתרון האפשריים בלוח ונדבר על חסם מספר הפתרונות האפשריים.
 5. חיפוש לוחות בהם קיים פתרון בו הנורות שינו את מצבם פעם אחת.
 6. נציג שיטה למציאת פתרונות בהם כל נורה תשתנה את מצבה פעם אחת.
- בנוסף, קיימות שאלות רבות הקשורות למשחק ובפרויקט ננסה להציג פתרון לחלקן. יתרה מזאת, נרצה להציג תופעות מעניינות, ולהראות שהמשחק אינו רק מהנה אלא גם מהווה אתגר מתמטי לא קטן.

2.3 תיאור משחק על גרף

אחרי שתיארנו את המשחק על לוח, נתאר את המשחק על גרף. נזכיר שגרף זה מבנה המכיל קשתות וצמתים, קשתות מוגדרות כצירוף סדור של שני צמתים. כדי לתאר את משחק האורות על גרף נשתמש באותם כללים שהגדרנו. במשחק על גרף הצמתים הם המשבצות לכן, לחיצה על צומת הופכת את מצבה ומצב שכניה. נגדיר שזוג צמתים יקראו שכנים אם קיימת קשת המחברת ביניהם. מטרת המשחק לעבור מגרף שכל הצמתים במצב התחלתי למצב סופי.

איור 2: משחק על גרף לדוגמה



נמחיש זאת על דוגמה שבאיור 2. איור 2a מתאר את מצב התחלתי, נסמן את המצב התחלתי של צומת בצבע כחול. איור 2b מתאר לחיצה על צומת שצבוע בירוק. לחיצה זה שינתה את הצמתים השכנות למצבם הסופי

שמוסמן בצבע אדום.

הערה 2.2: בפועל צומת ירוקה גם נצבעת באדום הצביעה לירוק נועדה להצגה.

2.4 השוואה בין משחק על לוח למשחק על גרף

נרצה להראות כי משחק על לוח הוא סוג של משחק על גרף כלומר, כל משחק על לוח ניתן לתאר בעזרת משחק על גרף.

נתאר משחק על לוח כמשחק על גרף בעזרת הכללים הבאים:

1. כל משבצת במשחק על לוח נהפוך לצומת.

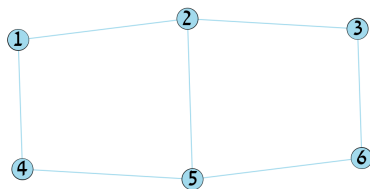
2. כל זוג משבצות סמוכות על לוח נחבר בקשת בגרף.

לדוגמה, ניקח לוח 2×3 נמספר את המשבצות כמו באיור 3א. הגרף המתקבל מתואר באיור 3ב.

איור 3: דוגמה למשחק על לוח שתורגם למשחק על גרף

(ב) משחק על גרף שתורגם מלוח 2×3

(א) משחק על לוח 2×3 שמשבצותיו ממוספר



1	2	3
4	5	6

הערה 2.3: קיימים משחקים רבים שניתן לתאר על גרף אך, לא ניתן לתאר אותם על לוח. לדוגמה, גרף בו יש צומת אם יותר מ-4 שכנים לא ניתן לתאר על לוח מכיוון שלכל משבצת על לוח יש לכל היותר 4 משבצות סמוכות.

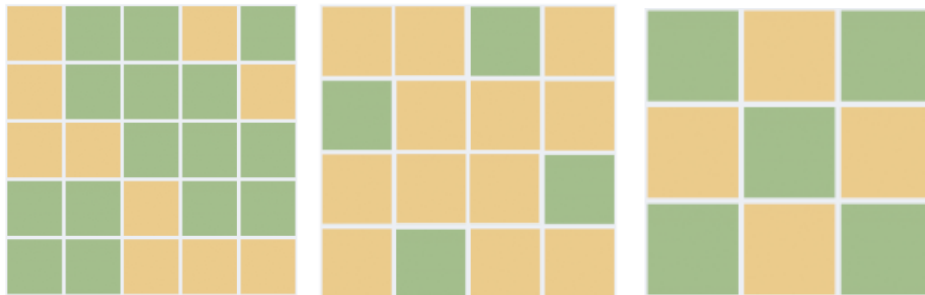
הערה 2.4: בעזרת השיטה שתיארנו אפשר לתאר כל משחק על לוח כמשחק על גרף, אבל ההפך הוא לא נכון. כלומר, לא כל משחק על גרף אפשר לתאר כמשחק על לוח.

מכיוון שמשחק על לוח ניתן לתאר כמשחק על גרף לכן, טענות שמתקיימות במשחק על גרף נכונות במשחק על לוח.

3 אלגוריתם למציאת פתרון

לפני שנציג את שיטות למציאת פתרון, נרצה להמחיש את האתגר במשחק על ידי הצגה מספר תופעות שמתקיימות במשחק. באיור 4 מוצגים מספר פתרונות אפשריים ללוחות שונים, לחיצה על הלחצנים הירוקים בסדר כלשהו תוביל לפתרון המשחק. ניתן לראות שמספר הלחיצות הנדרשות לפתרון לוח 4×4 קטן ממספר הלחיצות הנדרשות לפתרון לוח 3×3 . אפשר היה לחשוב שככל שהלוח גדול יותר נדרשות יותר לחיצות כדי להגיע לפתרון, אך, ניתן לראות באיור 4 זה לא נכון. תופעה נוספת המתקיימת במשחק היא שכמות הפתרונות עבור לוחות שונים משתנה. עבור לוח 3×3 קיים פתרון יחיד, אולם ללוח 4×4 קיימים 16 פתרונות. באופן מפתיע, ללוח 5×5 קיימים רק 4. תופעה זו מפתיע משום שאפשר היה לצפות שככל שהלוח גדול יותר כך, מספר הפתרונות יגדל. על מנת לחדד תופעה זו, נסתכל על לוחות ריבועיים $(n \times n)$. נשאל מהו המימד של הלוח אם הכי הרבה פתרונות ומהו מספר פתרונות ללוח זה כאשר $n \in [1, 20]$? התוצאה המתקבלת היא שמספר הפתרונות הגדול ביותר הוא כאשר $n = 19$ ומספר הפתרונות הוא 65,536. בנוסף $n = 19$ הוא הלוח היחיד ב $n \in [1, 20]$ המקבל את מספר פתרונות זה. לאומת זאת מספר הפתרונות השני בגודלו הוא 256 ומתקיים בעבור $n \in \{9, 16\}$.

איור 4: פתרונות של משחק על לוחות שונים



שתי גישות למציאת פתרון שנציג בעבודה מבוססות על מידול הבעיה לשדה לינארי ולמערכת משוואות שפתרונה יוביל לפתרון המשחק. נתאר את השיטות אומנם, בתחילה הן נראות שונות אך, נציג את הקשר ביניהן.

3.1 אלגוריתם שמבוסס על מטריצת שכנויות

כדי למדל את הבעיה על שדה לינארי נשתמש ביצוג גרפי, לחיצה על צומת משנה את הצומת ושכנותיה. נסמן את צמתים ב n_i . ניתן לתאר את המשחק בעזרת תיאור אלגברי:

1. כל צומת יכול להיות בשלי מצבים, את המצבים נסמן: $\{0, 1\}$.
2. מצב של צומת i נסמן ב n_i .
3. המצב התחלתי של משחק על גרף הוא שכל צמתים אם הערך התחלתי שהוא 0.
4. מצב סופי של משחק על גרף הוא שכל צמתים אם הערך הסופי שהוא 1.

הערה 3.1: עבור משחקים על לוח נתאר את המשבצות ומצבם הנוכחים ב $a_{i,j}$. זאת מכיוון שמשחק על לוח ניתן לתאר בעזרת מטריצה.

הערה 3.2: פעולת לחיצה על לחצן משנה את מצב המנורה, שינוי מצב מנורה ניתן לתאר בעזרת חיבור בשדה \mathbb{Z}_2 . נורה שמצבה התחלתי הוא n_i לאחר לחיצה תעבור למצב $n_i + 1$.

תיאור של משחק שכזה מאפשרת לנו לתאר המשחק בצורה וקטורית. אם ניקח לדוגמה משחק בגודל 2×2 נמספר את שורות ואז עמודות מלמעלה למטה כמו שמתואר באיור 5. נוכל לתאר את הלוח שכזה במצבו התחלתי כמטריצה

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

ואם נרצה לתאר את לוח ממצב התחלתי לאחר לחיצה על משבצת 1.

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \xrightarrow{n_1} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

כפי שתיארנו בהערה 3.2 אפשר לתאר שינוי מצב הנורה על ידי חיבור עם אחד.

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

אם נציג כל מטריצה ע"י וקטור קואורדינטות בסיס סטנדרטי של מרחב מטריצות אז נוכל לרשום את השוויון הנ"ל גם כך

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

וקטור שחיברנו עם מצב הלוח התחלתי הוא וקטור שמתאר את הלחיצה ונקראה לו בעבודה זה וקטור שינוי.

הגדרה 3.1: תהי משחק על גרף בעל n צמתים ממספרים מ 1 עד n , וקטור שינוי t_i של צומת i הוא וקטור שייד \mathbb{Z}_2^n , שאם תחבר אותו אם מצב הלוח הנוכחי התוצאה המתקבלת תהיה מצב הלוח לאחר לחיצה על צומת i .

כדי לבנות וקטור שינו של צומת i נשים ערך 1 בכל אינדקסים בהם האינדקס שווה למספור של צומת שכנה לצומת i ובאינדקס של צומת עצמה כלומר, באינדקס i . שאר הערכי הוקטור הם אפס.

איור 5: שיטת מספור משבצות על לוח

1	2	3
4	5	6

הערה 3.3: מספור לוח שעובר על שורות ואז עמודות מלמעלה למטה כמו שמתואר באיור 5, היה שיטת המספור הקבוע בפרויקט זה עבור משחקים על לוח.

נדגים על גרף תיאור וקטורי השינוי, נבצע זאת על גרף באיור 6. באיורים עלו נצבע אדום צמתים שמצבם 1 ובכחול צמתים שמצבם 0. בעזרת וקטור השינוי אפשר לתאר תוצאה של מספר לחיצות, נעשה זאת בעזרת חיבור וקטור שינויים וחיבור מצב הגרף. התוצאה שנקבל תהיה הגרף המתקבל לאחר לחיצה של צמתים הללו. נדגים רעיון זה כאשר מצב של גרף מתואר באיור 6א. נניח שצומת 1 היא יחידה שדלוקה. נרצה להראות איך הגרף יראה אם ילחצו על כפותרים 1, 3. וקטור שינוי של צומת 1 הוא

$$t_1 = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

$$t_1 + t_3 = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

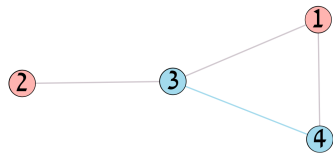
ומצב התחלתי שמתואר באיור נסמן ב S_0 לכן מתקבל

$$S_0 + t_1 + t_3 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

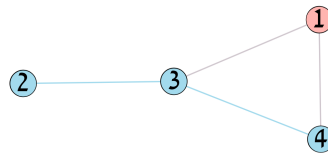
הגרף המתקבל לאחר חיבור אכן תואם לתוצאה המצופה מתואר באיור 6ב.

איור 6 : דוגמה לתיאור וקטור שינוי במהלך משחק על גרף

(ב) מצב של הגרף לפני לחיצה



(א) מצב של הגרף לפני לחיצה



הערה 3.4: היות וקטור שינוי שדה \mathbb{Z}_2^n חיבור בין וקטורים הינו חיבור בין האינדקסים מודולו 2 וכפל בסקלר הוא לכפול את כל ערכי וקטור בסקלר כאשר הסקלרים יכולים להיות 0 או 1

הערה 3.5: מספר זוגי של לחיצות אינו משנה את מצב הלוח

$$\text{היות ואנחנו עובדים על שדה מודולו } 2 \quad t_i + t_i = \vec{0}$$

הערה 3.6: לא משנה כמה תלחץ על לחצן בודד הלחצן יכול להעביר אותך לשני מצבים שונים בלוח.

מספר הלחיצות על אותו לחצן אינו משנה לחצן עכשיו לחוץ אם נלחץ מספר אי זוגי של פעמים כי מספר לחצות הזוגיות לא שינו את הלוח. בנוסף זה מסביר את הסיבה למה כפל בסקלר שאנחנו מוכנים לקבל הוא הערכים 0, 1. לכן בהמשך הפתרון יתואר אם יש צורך ללחוץ בלחצן או לא, לא תהיה התייחסות לכמות הלחיצות כי התוצאה מתקבלת רק תלויה בזוגיות של מספר הלחיצות.

הערה 3.7: היות ומצב התחלתי הוא שכל הצמתים במצב 0 לכן, ניתן לתאר את המצב עליו נעבור רק בעזרת צירוף לינארי של וקטורי שינוי בלבד.

היות ומצב התחלתי נסמן כרגע ב S_0 הוא כולו וקטור האפס מתקיים:

$$(1) \quad S_0 + \sum_{j=1}^n \vec{t}_j x_j = \sum_{j=1}^n \vec{t}_j x_j$$

בעקבות כך ניתן לתאר את בעיית המשחק לצורה הבאה:

$$(2) \quad \sum_{j=1}^n \vec{t}_j x_j = \vec{1}$$

כאשר $\vec{1}$ וקטור שכל ערכיו אחדים, שזהו מצב הסופי של הגרף ו n מספר הצמתים בגרף. נשים לב שאם ידוע צירוף $x = [x_1, x_2, \dots, x_n]$ שמקיים את המשוואה 2 קיבלנו פתרון של משחק על גרף. כדי להגיע לפתרון על גרף נלחץ על הצמתים שמספורם שווה לאינדקסים שמקיימים $x_j = 1$ בצירוף x .

בנוסחה 1 קיימים מספר תכונות שנרצה לציין.

למה 3.1: סדר הלחיצות לא משנה את התוצאה הסופית

בגלל אסוציאטיביות של חיבור בשדה סדר לחיצות לא משנה.

למה 3.2: כמות האפשרויות לחיצה על לוח $m \times n$ הוא $2^{m \cdot n}$

לפי הערה 3.6 כל לחצן יכול להיות בשתי מצבים והיות לפי למה 3.1 סדר הלחיצות לא משנה, לכן ללוח $m \times n$ מספר אפשרויות לחיצה $2^{m \cdot n}$. כדי להבין גודל מספר זה נסתכל במשחק על לוח 6×6 כמות אפשרויות לחיצה גדולה מכמות הסטנדרטית שמציגים מספר שלמים, 4 בתים כלומר המספר הגדול ביותר שאפשר להציג בעזרת 4 בתים הוא $2^{32} - 1$ המטרה של המחשה זה להדגיש כמה לא פרקטית לנסות לפתור בעזרת ניסיון כל האופציות האפשריות.

מערכת משוואות שמתוארת בנוסחה 2 אפשר לתאר במספר צורות. נפוצה מבניהם היא בעזרת מטריצה כמו שמתואר בנוסחה 3.

$$\begin{bmatrix} t_1 & t_2 & \dots & t_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} t_{1,1} & t_{1,2} & \dots & t_{1,n} \\ t_{2,1} & t_{2,2} & \dots & t_{2,n} \\ \dots & \dots & \dots & \dots \\ t_{i,j} & t_{i,2} & \dots & t_{i,n} \\ \dots & \dots & \dots & \dots \\ t_{n,1} & t_{n,2} & \dots & t_{n,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ \dots \\ 1 \end{bmatrix} \quad (3)$$

נשים לב שלמטריצה A במשוואה 3 מתקבל ש $A_{i,j} = 1$ כאשר i, j צמתים שהם שכנים או זהים $i = j$.

הגדרה 3.2: מטריצה שמתארת את משחק שקבלנו במשוואה 3 תקראה מטריצת שכנויות של משחק.

הערה 3.8: היות וכל צומת שכנה היא שכנה אחד לשני לכן במטריצה סימטרית

דוגמה למטריצה המתקבלת מגרף באיור 6ב:

$$\begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

הגדרה 3.3: תהי משחק ומערכת משוואות שמתארת אותו מצורה 3 נגדיר את וקטור פתרון כצירוף הערכים מביאים לפתרון של מערכת המשוואות ונסמן אותו ב \vec{x}

נזכיר שאם \vec{x} וקטור פתרון של המערכת ו $x_i = 1$ אז המשמעות שכדי לפתור את המשחק צריך ללחוץ על לחצן i . בנוסף נזכיר שאתכן שהיו כמה פתרונות אפשריים.

הגדרה 3.4: שיטת פתרון בעזרת יצירת מטריצה שכנויות על ידי וקטור שינויים תקראה אלגוריתם מבוסס מטריצת שכנויות.

תוצאה דומה לתיאור שיטת הפתרון מבוססת מטריצת השכנויות אפשר לראות מהספר [2]. מרגע שהצלחנו לתאר את הבעיה מערכת משוואות לינארית על שדה \mathbb{Z}_2^n מכך נוכל להיעזר בכלים של אלגברה לינארית כדי למצוא את הפתרון כמו מציאת פתרון בעזרת דירוג, מציאת מטריצה פסאודו הפוכה וכולי.

הערה 3.9: עבור משחק על לוח $m \times n$ גודל מערכת המשוואות המתקבל משיטה מבוססת מטריצת שכנויות הוא $m \cdot \dots \cdot n$ משתנים ומשוואות.

עבור לוח $[m \times n]$ כמות הלחצנים $m \cdot n$ ולכן קיבלנו גודל שכזה. השאלה הטבעית שעולה האם אפשר לצמצם את גודל זה?

3.2 אלגוריתם שמבוסס על מילוי עקבי של שורות

עד כה הצגנו בעבודה גישה פתרון הנעזרת במטריצת שכנויות, נרצה להראות שיטה נוספת למציאת פתרון. שיטת הפתרון שנציג נובעת מהערה 3.9 נרצה להציע שיפור ולצמצם את כמות המשתנים והמשוואות למציאת פתרון למשחק על לוח $m \times n$ ב $\min(m, n)$ משוואות ומשתנים. צמצום גודל המערכת המשוואות יכולה להוביל לחישוב מהיר יותר וניצול טוב יותר של מידע.

המאמר [1] מציג שיטה למציאת פתרון של משחק על לוח כלשהו, בגישה קצת שונה. בפרק זה נציג את הגישה שמתוארת [1] ונראה את הקשר של בין שתי השיטות. לגישה החדשה ניקרא לאורך כל הפרק שאלגוריתם שמבוסס על מילוי עקבי של שורות או בקצרה מילוי עקבי.

הגדרה 3.5: שאלגוריתם שמבוסס על מילוי עקבי של שורות, היא שיטה שמבוססת על עיקרון שאם, ידוע איזה

כפתורים צריכים להילחץ בשורה העליונה כדי להגיע לפתרון, אפשר לגלות את כל שאר הכפתורים שצריכים להילחץ כמעט מידית.

נתאר את שאלגוריתם שמבוסס על מילוי עקבי של שורות בשלבים, כל שלב נדגים על לוח 3×3 שמתואר באיור 7. הלוח 3×3 הנתון ממספור באינדקסים בשיטה דומה כפי שתיארנו בהערה 3.3. שאלגוריתם שמבוסס על מילוי עקבי של שורות מתבסס על רעיון, שפתרון של המשחק הוא סדרה של לחיצות על משבצות מסוימות. נשייך לכל משבצת משתנה שיכול לקבל שני ערכים: 0 אם משבצת הזאת מופיעה בסדרת לחיצות של פתרון ו-1 אם משבצת הזאת כן מופיעה בסדרת לחיצות של פתרון המשחק. מראש לא ידוע לנו האם משבצות של שורה ראשונה יופיעו בסדרה הזאת או לא. אז משתנים של משבצות בשורה ראשונה עבור איור 7 הם x_1, x_2, x_3 .

איור 7: לוח 3×3 עם אינדקסים

$$\begin{bmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{bmatrix}$$

נבחין בתופעה הבאה, על מנת שהנורה במשבצת ראשונה בשורה ראשונה תהיה דולקת סכום המשתנה שלה ומשתנים של משבצות השכנות במודולו 2 חייב להיות 1. המשוואה המתקבלת עבור איור 7:

$$(4) \quad x_1 + x_2 + x_4 = 1$$

לכן משתנה במשבצת ראשונה בשורה שניה, שסימנו במשתנה x_4 חייב להיות שווה ל $x_1 + x_2 + 1$ היות ומתקיים:

$$x_1 + x_2 + x_4 = 1 \Rightarrow x_4 = x_1 + x_2 + 1$$

על מנת שהנורה במשבצת שנייה בשורה ראשונה תהיה דולקת סכום משתנה שלה ומשתנים של משבצות שכנות במודולו 2 חייב להיות 1. לכן מתקבל שערכו של x_5 חייב להיות שווה ל $x_1 + x_2 + x_3 + 1$ היות ומתקיים:

$$x_1 + x_2 + x_3 + x_5 = 1 \Rightarrow x_5 = x_1 + x_2 + x_3 + 1$$

באופן דומה מחשבים ערכי משתנים של שאר המשבצות בשורה שנייה ואחר כך על פי אותם שיקולים ערכי משתנים של משבצות בשורות הבאות. נבחין שלאחר שמילאנו את כל הלוח כמו שמתואר באיור 8. אם היה ידוע איזה לחצנים משורה העליונה שייכים לסדרת הלחיצות של פתרון אז, פתרנו את המשחק.

הגדרה 3.6: המשוואה של סכום משתנים של משבצת i וכל שכניה תקראה משוואת אילוצים על לחצן i

איור 8 : תרגום כל הלחצנים לפי המשתנים של השורה העליונה

$$\begin{bmatrix} x_1 & x_2 & x_3 \\ 1 + x_1 + x_2 & 1 + x_1 + x_2 + x_3 & 1 + x_2 + x_3 \\ 1 + x_1 + x_3 & 0 & 1 + x_1 + x_3 \end{bmatrix}$$

משוואה 4 הינה משוואת האילוצים שללחצן 4. עבור משחק לוח ריבועי באורך שורה n שהלחצנים ממספרים לפי הערה 3.3 ניתן לנסח בנוסחה פשוטה :

$$(5) \quad x_{i-n}^* + x_{i-1}^* + x_i^* + x_{i+1}^* + x_{i+n}^* = 1 \quad x_i^* = \begin{cases} x_i & \text{if } i \in [1, n \cdot m] \\ 0 & \text{otherwise} \end{cases}$$

נדגים חישוב משתנה לא משורה השנייה, העליונה ביותר, נתאר את x_7 בעזרת משתנים משורה העליונה. היות ו- x_7 משורה שלישית לכן נצטרך שהמשתנים מהשורה שניה בוטאו בעזרת משתנים משורה העליונה. נחלץ את x_7 ממשוואת האילוצים של לחצן 4 שזה הלחצן שמעליו.

$$x_1 + x_4 + x_5 + x_7 = 1$$

לכן

$$x_7 = 1 + x_1 + x_4 + x_5$$

היות ושמשתנים משורה השנייה הוגדרו לפי משתנים משורה העליונה :

$$x_4 = 1 + x_1 + x_2$$

$$x_5 = 1 + x_1 + x_2 + x_3$$

נציב ערכים אילו

$$x_7 = 1 + x_1 + (1 + x_1 + x_2) + (1 + x_1 + x_2 + x_3) \Rightarrow x_7 = 1 + x_1 + x_3$$

על מנת ליצור מערכת משוואות שתפתור את המשחק מחבר המאמר [1] מוסיף לשורה האחרונה עוד שורה, שורה דמיונית ומחשב ערכי משתנים של המשבצות שלה לפי אותם שיקולים. משום שזאת שורה דמיונית, בעצם אנחנו לא מדליקים אף נורה בה ערכי המשתנים של משבצות שלה חייב להיות אפס. כך נוצרת מערכת משוואות עם n משוואות ו- n משתנים וזה ההסבר שנתן מחבר המאמר.

את השיטה הדגמנו על לוח 3×3 , אפשר היה להדגים על כל לוח ושיטה תעבוד. בנוסף, שיטה שתיארנו ביצע מעבר על שורות אפשר היה לעשות בניה דומה גם לעמודות. בגלל שאפשר להפעיל את השיטה על שורות או

איור 9: לוח 3×3 מלאה כולל שורה וירטואלית

$$\begin{bmatrix} x_1 & x_2 & x_3 \\ 1+x_1+x_2 & 1+x_1+x_2+x_3 & 1+x_2+x_3 \\ 1+x_1+x_3 & 0 & 1+x_1+x_3 \\ 1+x_2+x_3 & x_1+x_2+x_3 & 1+x_1+x_2 \end{bmatrix}$$

איור 10: מערכת המשוואות המתקבלת משיטה פתרון לפי שורה העליונה בלוח 3×3

$$\begin{cases} 1+x_1+x_2=0 \\ x_0+x_1+x_2=0 \\ 1+x_0+x_1=0 \end{cases}$$

עמודות כדי לקבל מערכת משוואות קטנה ביותר נבחר את כיוון עם פחות משבצות. המאמר [1] מתאר מספר רב של פתרונות בלוחות ריבועיים בגדלים שונה ואפילו על לוחות מלבניים. האתגר המרכזי בשיטה הספרדית היא להצדיק אותה למה יש שורה וירטואלית והאם יש קשר בין שני השיטות. בשלב זה נתרכז להראות את הקשר בין שיטה הספרדית ושיטה שהצגנו בפרק הקודם.

משפט 3.1: מטריצה המיצג של מערכת המשוואות האילוצים היא מטריצת שכנויות

משוואות האילוצים היא כל מהות של האלגוריתם למילוי עקבי מכיוון שהתקדמות בשורות מבוססת על המשוואות עלו. אם נפרס את משוואות האילוצים נקבל גם מערכת משוואות שפותרת את המשחק אבל כמות המשוואות הינה n^2 . נבחין שאם נציג אותם כמטריצה כאשר כל משוואת אילוצים מסודר לפי סדר הלחצנים נקבל את מטריצה שכנויות.

נדגים זאת על לוח 2×2 שמתואר באיור 11. וקטור השינויים:

איור 11: לוח 2×2

$$\begin{bmatrix} x_1 & x_2 \\ x_3 & x_4 \end{bmatrix}$$

$$t_0 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix}, t_1 = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix}, t_2 = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix}, t_3 = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix},$$

לכן מטריצת שכנויות נראת כך :

$$\begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix},$$

אם נסדר את המשוואות במערכת המשוואות לפי סדר האינדקסים של המשבצות נקבל את המערכת מהצורה

$$x_0 + x_1 + x_2 = 1$$

$$x_0 + x_1 + x_3 = 1$$

$$x_0 + x_2 + x_3 = 1$$

$$x_1 + x_2 + x_3 = 1$$

אפשר לראות שאם נבנה את המטריצה המייצגת של המערכת נקבל מטריצה דומה למטריצת השכנויות.

תעלומה נוספת שנשנה לפתור באלגוריתם למילוי עקבי היא למה צריך שורה וירטואלי, למה ערכה שווה ל 0, וכיצד באמת מתבצע צמצום כמות המשתנים ל n משתנים. הסבר לתופעה זה ניתן בעזרת תיאור אלגוריתם מילוי עקבי רק שהפעם את הפעולות במקום לעשות על טבלה שתיארה את הלוח, נבצע על המטריצה שמתארת את מערכת המשוואות, מטריצת השכנויות. נראה את הפעולות שעושים בשיטה על אותה דוגמה על לוח 3×3 . ונרצה להראות שאלגוריתם מילוי עקבי הוא דירג מתוחכם של מטריצה מורחבת $[M|\vec{1}]$.

כדי לחשב את x_4 באיור 7 השתמשנו במשוואת האילוצים של משבצת 1, אשר מתוארת במטריצה בשורה ראשונה.

$$x_1 + x_2 + x_4 = 1 \Rightarrow x_4 = x_1 + x_2 + x_3$$

נבחין שלושת השורות הראשונות של המטריצה 12 מאפשרות תיאור פשוט של המשתנים x_4, x_5, x_6 , בגלל ששורות עלו ניתן לבטא כמשוואות האילוצים של לחצנים 1, 2, 3 אם ננסה לתאר את משתנה x_7 באותה שיטה נסתכל על שורה ה 4 במטריצה ונראה שהיא תלויה ב x_4, x_5 היות ואמרנו שאפשר בקלות לתאר את משתנים עלו בעזרת שורות 1, 2 במטריצה לכן נעזר בשורות עלו כדי לתאר את x_7 , אופן שימוש בשורות עלו תהיה הפעלה פעולה שורות הבאה :

$$r_4 \leftarrow r_4 + r_1$$

$$r_4 \leftarrow r_4 + r_2$$

איור 12 : מערכת משוואות מורחבת של משחק על לוח 3×3

$$\left[\begin{array}{cccccccc|c} 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{array} \right]$$

שני פעולות שורות הללו הם שקולות לפעולה אלגברית הבאה :

$$(x_1 + x_4 + x_5 + x_7 + 1) + (x_1 + x_2 + x_4 + 1) = 0 \Rightarrow x_2 + x_5 + x_7 = 0$$

$$(x_2 + x_5 + x_7) + (x_1 + x_2 + x_3 + x_5 + 1) = 0 \Rightarrow x_1 + x_3 + x_7 + 1 = 0$$

ועכשיו קיבלנו תיאור של x_7 בעזרת המשתנים של שורה העליונה. בכך תרגמנו את אלגוריתם מילוי עקבי בעזרת פעולת שורות על מטריצה מורחבת. אחרי שנעבור על כל השורות בצורה שכזה נקבל את המטריצה **13**.

איור 13 : המטריצה לאחר פעולת שורות על כל שורות

$$\left[\begin{array}{cccccccc|c} 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array} \right]$$

נשים לב שבמטריצה **13** שלושת השורות התחתונות מתוארות אך ורק על ידי המשתנים x_1, x_2, x_3 , אם נתאר את

שורות עלו כמערכת משוואות נקבל את אותה מערכת משוואות של שאלגוריתם מילוי עקבי תיאר על לוח 3×3 את מערכת המשוואות הזה תיארנו במערכת 10.

אחת התוצאות שקיבלנו בזה שתיארנו את אלגוריתם שמבוסס על מילוי עקבי של שורות בעזרת פעולת שורות היא, שקבלנו הסבר לשורה דמיונית. משבצות בשורה הדמיונית הן השורות במטריצה שלאחר פעולת שורות שערך 1 מופיע אך ורק במשתנים של שורה העליונה בלבד. אם נסמן ב n את מספר המשבצות בשורה אז, היות ונקבל n שורות במטריצה של משבצות של משבצות בשורה הדמיונית, ולשורות במטריצה עלו יש לכל יותר n נעלמים של שורה העליונה לכן ניתן לפתור את הבעיה בעזרת שורות עלו בלבד.

3.3 השוואה בין שתי השיטות למציאת פתרון

לפי חישוב סיבוכיות לדרג מטריצה כללית בגודל $n^2 \times n^2$ זה $O(n^6)$ $O(n^2 \cdot n^4) =$.

דירוג בעזרת שיטה הספרדית אומרת שעל כל עמודה וקטור עמודה של מטריצת שכנויות יש לכל יותר 5 ערכים ששווים 1. כל החוכמה בדירוג לפי אלגוריתם מילוי עקבי הוא שפעולות השורות אינן פוגעות בשורות שנדרג בהמשך והמשתנים היחידים שמשתנים לאחר הפעולה הם המשתנים בשורה העליונה. לכן דירוג שורה היה חיבור של עד כ 5 שורות לכן הסיבוכיות $O(n^2 \cdot n^2) = O(n^4)$. לדרג את n משתנים הנותרים הוא בסיבוכיות $O(n \cdot n^2) = O(n^3)$.

ננסה להראות את סיבוכיות בפועל על ידי חישוב זמני חישוב. באיור 14 אפשר לראות ביצועים של שני האלגוריתמים ציר ה x גודל שורה של לוח המלבני. את שני האלגוריתמים הרצנו על לוחות בגדלים מ 10 עד 60 משבצות בשורה. ציר ה y זמן שלקח בשניות לפי התוצאות של איור 14 נראה שגישה הספרדית שבתאוריה יותר אופטימליות לוקחת יותר זמן. אחת הסיבות לקח שפונקציה שפותרת מערכת משוואות הינה פונקציה ספרייה וכנראה יש מימוש אופטימלי לפתרון הבעיה שאפילו שאלגוריתם מילוי עקבי מקטין את כמות המשתנים היא אינה יכולה להתחרות במימוש אופטימלי שמממשת ספרייה.

3.4 דיון לגבי משחק על גרף

כאשר תיארנו את המשחק על גרף אתכן והסיבות לקח היו:

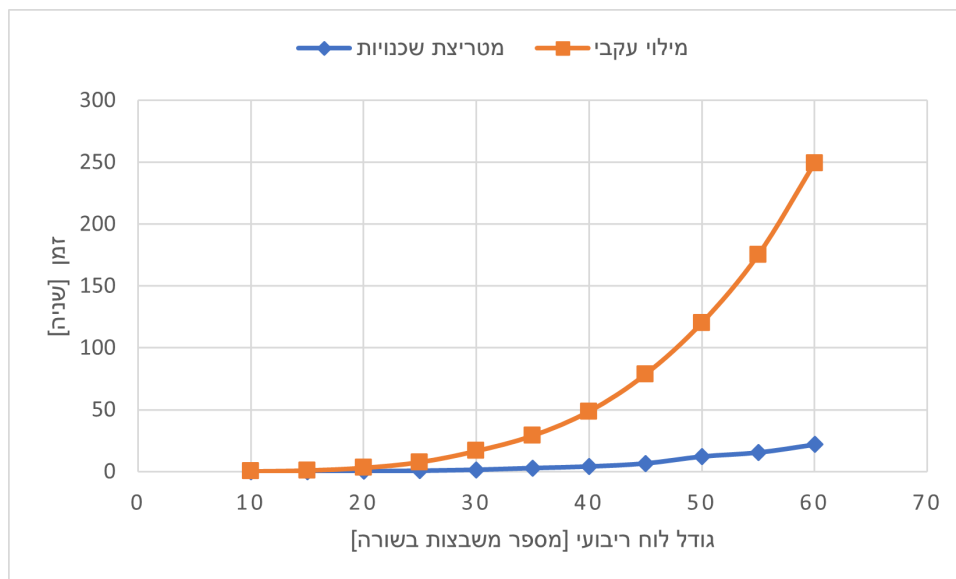
1. ככול שמבנה כללי יותר תאוריה שאתה מפתח מתאימה ליותר בעיות.

2. קיימת תאוריה רחבה שפותחה על גרפים ואתכן שנעזר בה.

3. מבליט את מהות הבעיה והגדרה הבסיסית ביותר של המשחק.

בפועל כשהצגנו את אלגוריתמים לפתרון המשחק התגלתה התמונה המלאה. שני האלגוריתמים שתיארנו מתארים את המשחק כגרף היות ושני האלגוריתמים בנויים על מטריצת השכנויות. קשר זה מדגיש ומראה

איור 14 : גרף מתאר ביצועים על לוח ריבועי גודל שורה מול זמן



שלפעמים רק תיאור מהות הבעיה מספיק כדי למצוא לבעיה פתרון.

4 קיום פתרון ומספר הפתרונות עבור משחק על גרף

עד כה הסתכלנו הצגנו שיטות למציאת פתרון, שיטות עלו האירו את העובדה ששאלת קיום הפתרון למשחק על גרף שקולה לשאלת קיום הפתרון למערכת משוואות לינאריות. בפרק זה נרצה להוכיח קיום פתרון למשחק לכל גרף. העובדה שקיים פתרון לכל כל גרף אינה מובנת מעליה. אחד המקומות ששאלה זה נשאלה היא בספר [4], בעבודתנו נראה הוכחה קצת שונה בעזרת הכלים שפיתחנו. אומנם הוכחה קיום פתרון הופיע לראשונה במאמר [3]. כמו כן המחבר כותב שיש רק הוכחה שמבוססת על אלגברה לינארית.

4.1 הוכחת קיום פתרון על גרף

הגדרה 4.1: תהי S קבוצה, פעולה בינארית על S היא פונקציה $S \times S \rightarrow S$ המתאימה לכל זוג סדור. פעולה בינארית עבור הזוג (s_1, s_2) תסומן $\langle s_1, s_2 \rangle$.

הגדרה 4.2: מכפלה פנימית היא פעולה בינארית שמוגדרת לכל זוג וקטורים במרחב וקטורי $S^n \subseteq \mathbb{R}^n$ ומעתיקה אותם ל S . תהי $S^n \subseteq \mathbb{R}^n$ מרחב וקטורי עם מכפלה פנימית, לכל $x, y, z \in S^n$ ו $\alpha \in S$, פעולה בינארית תקראה מכפלה פנימית אם היא מקיימת 4 אקסיומות הבאות:

$$1. \langle x, y \rangle = \langle y, x \rangle$$

$$\langle x + y, z \rangle = \langle x, z \rangle + \langle y, z \rangle \quad 2.$$

$$\langle \alpha x, y \rangle = \alpha \langle x, y \rangle \quad 3.$$

$$\langle x, x \rangle \geq 0 \quad (\text{א}) \quad 4.$$

$$\langle x, x \rangle = 0 \iff x = \vec{0} \quad (\text{ב})$$

הגדרה 4.3: לכל שני וקטורים $x, y \in \mathbb{Z}_2^n$ נגדיר פעולה הבאה:

$$(6) \quad x \cdot y = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} \cdot \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{bmatrix} = x_1 y_1 + x_2 y_2 + \dots + x_n y_n$$

לפעולה זה בין שני וקטורים ב \mathbb{Z}_2^n ניקרא מכפלה סקלרית.

הערה 4.1: פעולה שהגדרנו בהגדרה 4.3 נקראת מכפלה סקלרית למרות שהיא מקיימת רק 3 מתוך 4 תכונות של מכפלה סקלרית ב- \mathbb{R}^n . תכונה $\vec{u} = \vec{0} \iff \langle \vec{u}, \vec{u} \rangle = 0$ לא מתקיימת.

דוגמה שמסבירה את הערה 4.1:

$$\begin{bmatrix} 1 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 1 + 1 = 0$$

הערה 4.2: וקטורים $x, y \in \mathbb{Z}_2^n$ יקראו מאונכים אחד לשני נסמן זאת $x \perp y$ אם המכפלה הסקלרית שהלם שווה ל 0 $x \cdot y = 0$

משפט 4.1: תהי מטריצה $A \in \mathbb{Z}_2^{m \times n}$ אז $\text{Col}A \perp \text{Nul}A^T$ ו $\text{Col}A^T \perp \text{Nul}A$

הוכחה: תהי מטריצה $A \in \mathbb{Z}_2^{m \times n}$ ניקח $\vec{x} \in \text{Nul}A$ לכן $A\vec{x} = \vec{0}$. אז,

$$\vec{x} \perp \text{Row}A = \text{Col}A^T$$

רצוי לציין שעבור מטריצות $A \in \mathbb{Z}_2^{m \times n}$ מאותם שיקולים אותה הוכחה נכונה, רק עבור מכפלה הסקלרית שהגדרנו בהגדרה 4.3.

נרצה לחדד תופעה מעניינת שקוראת בשדה וקטורי \mathbb{Z}_2^n עם המכפלה הסקלרית שהגדרנו. עבור השדה הוקטורי \mathbb{R}^n לכל מטריצה $A \in \mathbb{R}^{m \times n}$ מתקיים $\text{Col}A \cap \text{Nul}A^T = \{\vec{0}\}$. טענה זה אינה נכונה בשדה \mathbb{Z}_2^n .

משפט 4.2: לכל משחק על גרף קיים פתרון.

הוכחה: כשפיתחנו את שיטת פתרון בעזרת מטריצת השכנויות שהגדרנו 3.4, הצלחנו לתאר את המשחק בעזרת מערכת משוואות. המטריצה שמתארת את המערכת קראנו לה מטריצה שכנויות נסמן ב $A \in \mathbb{Z}_2^{n \times n}$. נציין כמה עבודות על מטריצת השכנויות:

1. מטריצה סימטרית לפי 3.8

2. A המטריצה הינה ריבועית.

3. האיברים על האלכסון מטריצה A ערכם שווה ל 1.

כדי להראות שלמשחק יש פתרון צריך להראות שקיים פתרון למערכת

$$A\vec{x} = \vec{1}$$

במקרה ש A מטריצה הפיכה אז קיים פתרון יחיד. עבור המקרה שמטריצה אינה הפיכה כלומר $\text{Nul}A \neq \{\vec{0}\}$ ניקח $\vec{x} \in \text{Nul}A$ מהגדרה מתקיים $A\vec{x} = \vec{0}$ לכן מתקיים:

$$\vec{x}^T A \vec{x} = \vec{x}^T \vec{0} = 0$$

$$\vec{x} = [x_1, x_2, \dots, x_n]^T \text{ נסמן}$$

$$(7) \quad \begin{aligned} \vec{x}^T A \vec{x} = & a_{1,1}x_1^2 + 2(a_{1,2} + a_{2,1})x_1x_2 + \dots + 2(a_{1,n} + a_{n,1})x_1x_n + \\ & + a_{2,2}x_2^2 + 2(a_{2,3} + a_{3,2})x_2x_3 + \dots + 2(a_{2,n} + a_{n,2})x_2x_n + \dots \end{aligned}$$

היות ומטריצה סימטריות $a_{i,j} = a_{j,i}$ לכן מתקבל:

$$a_{i,j} - a_{j,i} = a_{i,j} + a_{j,i} = 1$$

נזכיר כי תוצאות של פעולת חיבור וחסור מודלו 2 זהות. לכן את המשוואה 7 אפשר לפשט:

$$\vec{x}^T A \vec{x} = a_{1,1}x_1^2 + a_{2,2}x_2^2 + a_{n,n}x_n^2$$

הבחנה נוספת לא משנה אם הערך 0 או 1 מתקיים השוויון, $x^2 = x$ לכן פשוט נוסף למשוואה 7:

$$\vec{x}^T A \vec{x} = a_{1,1}x_1 + a_{2,2}x_2 + a_{n,n}x_n$$

לכן קיבלנו $\vec{x}^T A \vec{x} = 0$ ומתקיים:

$$a_{1,1}x_1 + a_{2,2}x_2 + a_{n,n}x_n = 0$$

כלומר $\vec{1} \perp \vec{x}$ כאשר $x \in \text{Nul}A$ לפי משפט 4.1 מתקבל $\vec{1} \in \text{Col}A^T$ היות ומטריצה סימטרית $A^T = A$ לכן $\vec{1} \in \text{Col}A$ והוכחנו שלמערכת $A\vec{x} = \vec{1}$ יש פתרון.

הוכחת קיום הפתרון עבור המשחק כפי שהגדרנו על גרף הושגה, מסקנה נאיבית שניתן אולי לחשוב היא שלכל מצב התחלתי אפשרי היה ניתן לפתור את המשחק. בחלק זה של הפרק ננסה לחדד ולהעביר.

הגדרה 4.4: משחק אחר שאפשר להציע הוא משחק האורות כללי יותר מוגדר כך: לוח הבקרה נשאר זהה למשחק המקורי כלומר, שינוי נורות לאחר לחיצה מתנהג נשאר כפי שהוגדר במשחק המקורי. הבדל בין משחק החדש למקורי מצב התחלתי שחלק מנורות דולקות וחלק כבויות ורוצים להגיע למצב סופי שגם בו חלק מנורות דולקות וחלק כבויות.

עבור המשחק שהגדרנו 4.4 אותו קורא נאיבי יכול להניח שגם עבור משחק שכזה תמיד קיים פתרון. אם חושבים קצת לעומק קל מאוד לבנות דוגמה למשחק על לוח, שאין לו פתרון. דוגמה אפשרית למקרה שכזה היא לקחת משחק על לוח 2×1 בו מצב התחלתי הוא שהנורה השמאלית ביותר דלוקה ונרצה לעבור למצב הסופי בו כל הנורות דולקות. נרצה להראות שבאמת המשחק אינו פתיר וזה קל כי כמות הלוחות השונים שניתן להגיע בעזרת לחיצות היא מצומצמת. אפשר לתאר את כל המצבים האפשריים לפי כל צירופים לא סדורים האפשריים של לחיצות אפשריות. אם נמספר את הלחצנים לפי שיטת המספור שצינו בהערה 3.3. אז אוסף כל צירופים הלא סדורים של אוסף לחיצות אפשריים הם

$$(), (1), (2), (1, 2)$$

כדי לבדוק אם קיים פתרון למשחק הכללי שהגדרנו בהגדרה 4.4 נוכל להיעזר בהוכחה 4.2.

איור 15: מצבי הלוחות לאחר לחיצה של צירוף

(א) עבור צירוף $()$ (ב) עבור צירוף (1) (ג) עבור צירוף (2) (ד) עבור צירוף $(1, 2)$

$$\begin{bmatrix} 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 \end{bmatrix}$$

משפט 4.3: למשחק הזה יש פתרון אם ורק אם וקטור הפרש בין מצב סופי ומצב ההתחלתי אורתוגונלי למרחב העמודות של מטריצת השכנויות של המשחק.

הוכחה: נגדיר קודם את המשחק החדש אלגברית. לפי משוואה 1 אפשר לנסח אלגברית את המשחק כך. תהי A מטריצת השכנויות S_0 מצב התחלתי של המשחק ו S_e מצב הסופי של המשחק, נחפש צירוף לא סדור של לחיצות \vec{x} כך שמתקיים:

$$S_0 + Ax = S_e$$

נעביר אגפים ונקבל:

$$Ax = S_e - S_0$$

אם נחזור ונסתכל על הוכחה של משפט 4.2 כל הוכחה בנויה על להוכיח שמצב הסופי שבמקרה של המשחק המקורי הוא $\vec{1}$ שייך ל $ColA$. הראינו בהוכחה דרך קלה יותר לבדוק את זה והיא להראות שהוא שמצב

הסופי שייך $\text{Nul}A$. לכן כדי להוכיח שמשחק הכללי כפי שהגדרנו בהגדרה 4.4 פתיר, מספיק להראות $S_e - s_0 \in \text{Nul}A$

הערה 4.3: דרך קלה לבדוק שוקטור $\vec{v} \in \mathbb{R}^n$ שייך ל $\text{Nul}A$ של מטריצה $A \in \mathbb{R}^{m \times n}$ היא לוודא שמתקיים:

$$A\vec{v} = \vec{0}$$

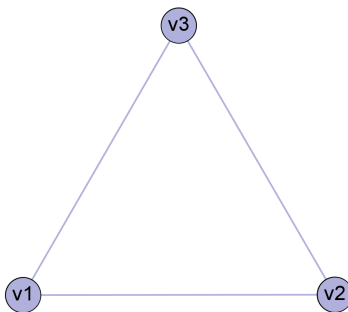
4.2 מספר הפתרונות עבור כל גרף

הוכחנו שלכל משחק על גרף שמתחל עם כל לחצנים במצב 0 יש פתרון ניזכר שסדר לחיצות אינו משנה את התוצאה על הלוח לכן אם נילחץ על הלחצנים בסדר כלשהו לפי פתרון נקבל גרף כולו דלוק.

השאלה שנשאל בפרק זה מה אפשר לומר על מספר פתרונות מפיתוח שעשינו. נציין קודם שניקרא לשני פתרונות שונים אם קיים לפחות לחצן אחד שמבדיל בין הפתרונות כלומר קיים לחצן ששייך לפתרון ראשון ולא שייך לפתרון שני כפי שציינו קודם סדר לחיצות לא משנה את הפתרון. לכן פתרון הינו קבוצה של לחצנים. בנוסף נזכר לפי הערה 3.6 מספר אי זוגי של לחיצות נחשב לחיצה לכן מספר הלחיצות על אותו לחצן לא משנה אלה רק זוגיות של מספר לחיצות לכן לכל לחצן יש רק שני מצבים שיכול להיות לחוץ או לא. כרגע נראה שקיים כמה פתרונות לדוגמא איור 16 המתאר משחק על גרף בו הצמתים כבויים. היות וגרף הינו קליקה לכן לחיצה בודדת על אחד הצמתים תדליק את כל הלחצנים.

קבלנו $G = \{\{v_1\}, \{v_2\}, \{v_3\}\}$ היא קבוצת של פתרונות. כלומר כבר הראינו שיש מקרים בהם יש יותר מפתרון אחד.

איור 16 : משחק על גרף



שאלה טבעית שנובעת שנשאלת היא כמה פתרונות יש למשחק מסוים. כדי לענות על שאלה נצטרך להציג כמה מושגים מאלגברה לינארית. בשיטת דירוג של גאוס אם ניזכר בקצרה בשיטה, אנחנו עוברים שורה ושורה ומנסים בעזרת פעולות של שורות ליצור עמודות בהם מופיע איבר בודד ששונה מאפס.

הגדרה 4.5: איבר מוביל בשורה הוא האיבר הראשון בשורה ששונה מאפס לאחר דירוג.

הגדרה 4.6: נעלמים שאברהם מובילים אחרי דירוג אקראי נעלמים מובילים.

הגדרה 4.7: במטריצה מדורגת בשיטת גאוס עבור עמודות שאין בהם איבר מוביל תיקרא עמודה חופשית.

הגדרה 4.8: נעלמים שעמודה שלהם חופשית אחרי דירוג יקראו נעלמים חופשים.

הגדרה 4.9: מספר נעלמים חופשיים במטריצה נקרא דרגת החופש.

ניתן דוגמה קטנה שתסכם את המושגים שהצגנו, ניקח מטריצה המדורגת הבאה:

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

בדוגמה צבענו בכחול את האיברים המובילים ובסגול את עמודות החופשיות. עבור הדוגמה דרגת החופש היא 3

הערה 4.4: דרך קלה לחשב את דרגת החופש $F(A)$ של מטריצה $A \in \mathbb{R}^{m \times n}$ כשידוע שדרגה של מטריצה A הוא $\text{rank}(A)$ בעזרת הנוסחה הבאה:

$$F(A) = n - \text{rank}(A)$$

לאחר שהגדרנו את מושג דרגת החופש נוכל לנסח את המפשט המרכזי של הפרק.

משפט 4.4: מספר הפתרונות של משחק שווה ל 2^k כאשר k שווה לדרגת החופש של מטריצה A של פתרון הסטנדרטי

היות לכל משחק ניתן להגיר מטריצת שכנויות של משחק שהגדרנו ב 3.2 ופתרונות של משחק וקטורים X של מערכת $A\vec{x} = \vec{1}$ כאשר A מטריצת שכנויות. ידוע שקיים פתרון למשחק ואם הוא משחק שמתחיל שמצב כל הנורות הוא 0 אז יש משפט 4.2. שמוכיח שקיים פתרון.

היות ומניחים שיש כמה פתרונות אפשר לתאר את כל פתרונות כ $x = x_n + x_0$ כאשר $x_0, x_n \in \text{Nul}(A)$ פתרון פרטי שידוע שקיים ו x כל פתרונות הכללים.

לכן מספר פתרונות כללים שווה למספר פתרונות במרחב האפס. ידוע שמספר פתרונות במרחב האפס תלוי לדרגת החופש ולכן מספר הווקטורים שפורשים את מרחב האפס שווה לדרגת החופש שנסמן ב k . כמות הווקטורים במרחב זה שווה לכל וקטורים שניתן ליצור בצירוף לינארי

$$x = a_1x_1 + a_2x_2 + \dots + a_kx_k$$

כאשר הערכים של $a_i \in \mathbb{Z}_2$ לכן לכל מקדם יכול להיות 2 ערכים, לכן כל הקונבנציות האפשריות 2^k ששווה לכמות הווקטורים במרחב האפס וכמות הפתרונות השונים של המשחק.

הבחנה נוספת ומעניינת שנרצה לציין היא בנושא חסם עליון לכמות הפתרונות. חסם עליון טריוויאלי לכמות המקסימלית של פתרונות היא 2^n פתרונות כאשר n שווה למספר הלחצנים כלומר לא יכול להיות יותר פתרונות מאשר כמות הלחיצות השונות האפשריות במשחק.

הערה 4.5: עבור משחק לוח מלבני בגודל $m \times n$ קיים לכל יותר 2^k כאשר $k = \min\{m, n\}$ פתרונות שונים. הערה זה נכונה לפי גישה פתרון הספרדית שהגדרנו 3.5 ניתן לתרגם את משחק ל k משוואות ש k יכול להיות מספר שורות או עמודות לכן ניקח את המספר הקטן יותר. לסיכום נרצה להציג טבלה של מספר הפתרונות כתלות מממדי הלוח. את הטבלה אפשר לראות באיור 17 כאשר, השורות ועמודות בטבלה מייצגות את ממדי הלוח בהתאמה.

איור 17: טבלה מתארת מספר פתרונות בלוחות $m \times n$

	1	2	3	4	5	6	7	8	9
1	1	2	1	1	2	1	1	2	1
2	2	1	4	1	2	1	4	1	2
3	1	4	1	1	8	1	1	4	1
4	1	1	1	16	1	1	1	1	16
5	2	2	8	1	4	1	16	2	2
6	1	1	1	1	1	1	1	64	1
7	1	4	1	1	16	1	1	4	1
8	2	1	4	1	2	64	4	1	2
9	1	2	1	16	2	1	1	2	256

5 פתרון אופטימלי עבור לוחות מלבניים

בפרק זה נציג סוג מסוים של פתרונות, סוג זה אקרה פתרון אופטימלי. סוג זה של פתרון מקל רבות על המשחק כיוון שמצמצם את כמות הלחיצות האפשריות בכל מצב במשחק.

הגדרה 5.1: פתרון אופטימלי של משחק הינו פתרון בו שינוי מצב כל הנורות היה יחיד. כלומר השחקן פתר את המשחק כאשר כל נורה עברו ממצב התחלתי למצב הסופי פעם אחת בלבד

באיור 18 ניתן דוגמא לפתרון מינמלי בלוח 2×3 . כשלוחצים על לחצנים 3, 4 על לוח כל נורות נדלקות ואף אחת מהם לא נכבה באף שלב של לחיצה.

איור 18 : פתרון מינמלי של משחק

1	2	3
4	5	6

נרצה לחדד ולהדגיש עד כמה קל למצוא פתרונות אופטימליים. אם ניקח לוח 2×3 כפי שמתואר באיור 18, ונתבונן במספר כל הפתרונות שיש ללוח זה כפי שמתואר בטבלה 17 נראה שיש 4 פתרונות. קיימים שני פתרונות אופטימליים ושני פתרונות לא אופטימליים, ופה נשאלת השאלה כמה פתרונות בהתבוננות חפזה הקרואה רואה. כנראה ששני הפתרונות האופטימליים מיידית נמצאו. כנראה גם שכדי למצוא פתרונות הנותרים נצטרך לקחת דף ועט ולחפש אותם גם עבור לוח בממד מצומצם שכזה. את כל ארבעת הפתרונות נציג באיור 19.

איור 19 : משחק על גרף לדוגמה

1	2	3
4	5	6

1	2	3
4	5	6

1	2	3
4	5	6

1	2	3
4	5	6

השאלה שנפתור בפרק זה לאיזה לוחות קיים פתרון מינמלי כאשר מצב התחלתי הוא שכל הנורות במצב 0.

5.1 הוכחת אי קיום לפתרון אופטימלי על לוחות ששני הממדים גדולים מ 4×4

לפני שבאמת נוכיח את הטענה נציין את משמעות ששני הממדים גדולים מ 4×4 .

הערה 5.1: קיים פתרון אופטימלי למשחק $2 \times m$ כאשר m הוא אי זוגי.

הפתרון הוא פשוט נלחץ פעם אחת בשורה ראשונה ופעם אחת בשורה השנייה כמו שמתואר באיור 20.

הסיבה שקיים פתרון אופטימלי לכל לוח $2 \times m$ שפשוט שיטת הפתרון שהצגנו ניתן להרחבה לכל m . ניקח לדוגמה את איור 20 אם נוסיף שתי עמודות ימינה אז לחיצה על שורה העליונה בעמודה הימנית ביותר תפתור את המשחק.

בגלל הערה 5.1 הגבלנו חיפוש לפתרון אופטימלי ללוחות ששני הממדים שלהם גדולים מ 4×4 . בכך פסלנו את אינסוף הלוחות שתיארנו בהערה 5.1.

איור 20 : פתרון ללוח 2×9

כדי להוכיח טענת הפרק נעזר בלוח שמתואר כך, ללוח יש משבצת שהיא ראשית הצירים נסמן אותה כ $a_{1,1}$. בלוח זה הצירים ממשיכים אינסוף ימינה ולמעלה. נתאר את המשחק הנתון בעזרת מטריצה אינסופית.

משפט 5.1: אין אף פתרון אופטימלי בו המשבצת בראשית הצירים נלחצה

הוכחה: לאחר לחיצה על $a_{1,1}$ מתקבל הלוח הבאה :

$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

כדי להדליק את הנורה $a_{2,2}$ נצטרך ללחוץ $a_{2,3}$ או $a_{3,2}$. אם נילחץ על $a_{2,3}$ נקבל את הלוח הבאה :

$$\begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

נשים לב כי הגענו למבוי סתום כי ניתן להדליק את $a_{3,2}$ רק על ידי לחיצה על $a_{4,2}$ ואז לא היה ניתן להדליק את $a_{3,1}$. ניתוח דומה אפשר לתאר עבור לחיצה על $a_{3,2}$.

לפי טענה 5.1 אם ברצוננו למצוא למשחק האינסופי שהגדרנו פתרון מינמלי לא נוכל להדליק את נורה $a_{1,1}$ על ידי לחיצה עליו. לכן על מנת להדליק את $a_{1,1}$ נלחץ על $a_{1,2}$.

משפט 5.2: אין אף פתרון אופטימלי בו נלחצה משבצת $a_{1,2}$

הוכחה: כדומה להוכחה טענה 5.1 נגיע למצב סתום. אם נסתכל על מצב הלוח לאחר לחיצה על משבצת $a_{1,2}$ ניראה כי יש סידרת לחיצות מאולצת כדי להדליק נורות מסוימות.

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

כדי להדליק את הנורות $a_{2,1}$ נהיה חייבים ללחוץ על $a_{3,1}$. כדי להדליק את הנורות $a_{2,3}$ נהיה חייבים ללחוץ על $a_{2,4}$. כדי להדליק את הנורות $a_{3,3}$ נהיה חייבים ללחוץ על $a_{4,3}$. כדי להדליק את הנורות $a_{5,2}$ נהיה חייבים ללחוץ על $a_{6,2}$. כדי להדגיש את הכפתורים שנלחצו נסמן אותם * ונקבל את הלוח הבאה:

$$\begin{bmatrix} 1 & * & 1 & 1 & 0 \\ 1 & 1 & 1 & * & 1 \\ * & 1 & 1 & 1 & 0 \\ 1 & 1 & * & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & * & 1 & 0 & 0 \end{bmatrix}$$

נשים לב ששוב הגענו למבוי סתום.

אפשר להחליף בין הצירים ולקבל בעזרת אותה הוכחה למה לא קיים פתרון בו נלחצת משבצת $a_{2,1}$. היות ועברנו על כל האפשרויות שאפשר לנסות להדליק את $a_{1,1}$ והראינו שבכל מצב מגיעים למבוי סתום.

מסקנה 5.1: למשחק אינסופי שכזה לא קיים פתרון אופטימלי.

מסקנה 5.2: למשחק $n \times 4$ לא קיים פתרון אופטימלי כאשר $n > 4$

עבור משחק שכזה אותם טענות 5.1 ו 5.2 מתקיימות. עבור תחילת משחק מלחיצה על $a_{2,1}$ נקבל את סידרת לחיצות המאולצת $\{a_{2,1}, a_{1,3}, a_{3,4}, a_{4,2}, a_{6,3}\}$ שמתוארת בלוח הבאה:

$$\begin{bmatrix} 1 & 1 & * & 1 \\ * & 1 & 1 & 1 \\ 1 & 1 & 1 & * \\ 1 & * & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & * & 1 \end{bmatrix}$$

והגענו שוב למבוי סתום.

מסקנה 5.3: במשחק על לוח $m \times n$ שמתקיים $\min(m, n) > 4$ למשחק אין פתרון אופטימלי

5.2 אלגוריתם למציאת פתרון אופטימלי

קיימים מספר דרכים למציאת פתרון אופטימלי. אפשר לנסות ולחפש פתרון ידנית. דרך נוספת היא לעבור על כל הפתרונות של משחק רגיל ולבדוק עם יש מבניהם פתרון אופטימלי. נציעה דרך אחרת לחפש פתרון מינמלי והיא

בעזרת להשתמש באותה מטריצה שכנויות כפי שהגדרנו רק להגדיר את זה שהיא על חוג \mathbb{Z} . בעזרת שימוש בחוג \mathbb{Z} מאלצים את שפתרונות המתקבלים שידליקו כל נורות אך ורק פעם אחת, זאת מתקיים בעקבות משוואות האילוצים שהגדרנו ב 3.6 שמאלצות את הסכום להיות שווה לאחד, אם נסתכל על נוסחה של משוואת האילוצים הכללים נוסחה 5 היות וחיבור על השלמים לכן מאולצים במשוואה זה שהיה לחצן בודד לחוץ לכן פתרון מערכת המשוואות מתאר פתרון מינמלי של משחק. התיאוריה שפיתחנו באלגברה לינארית הייתה תקפה לשדות אבל כלי תכנות שהשתמשנו בעבודה זה יודע לפתור גם על חוג של השלמים והסמכנו על הכלי כדי לבדוק את המקרים שממדים שייכם לקבוצה $\{(m, n) : 2 < m, n < 7\}$ וקיבלנו שהלוח היחיד בקבוצת הממדים העלו שיש לו פתרון מינמלי הוא לוח 4×4 ופתרון מתואר באיור 21.

איור 21 : פתרון ללוח 4×4

מימוש של הפרויקט בוצע על ידי שפת תוכנה Python בעזרת הכלי Sage וספריות מתמטיות נוספות.

6.1 יצירת מטריצת שכנויות

קוד זה יוצר מטריצת שכנויות של משחק על לוח $m \times n$.

```
[8]: import numpy as np
# to prove the minimal case on not square we need to build matrix for
↳ not rectangler board
def generate_neighbord_matrix_m_n(m,n) -> np.array:
    mat = np.zeros((m*n, m*n), dtype= np.int8)

    # the general case
    for j in range(0, m*n):
        if j-n > -1 :
            mat[j-n,j] = 1

        if j % n != 0 :
            mat[j-1,j] = 1

        mat[j,j] = 1

        if (j+1) % n != 0 :
            mat[j+1,j] = 1

        if j+n < m*n :
            mat[j+n,j] = 1

    return mat
def generate_neighbord_matrix(n) -> np.array:
```

```
    return generate_neighbord_matrix_m_n(n,n)

print('Adj matrix for 3,2 board:')
print(generate_neighbord_matrix_m_n(3,2))
```

Adj matrix for 3,2 board:

```
[[1 1 1 0 0 0]
 [1 1 0 1 0 0]
 [1 0 1 1 1 0]
 [0 1 1 1 0 1]
 [0 0 1 0 1 1]
 [0 0 0 1 1 1]]
```

6.2 מציאת פתרון

הצגה כיצד לקבל פתרון ממטריצת השכנויות.

```
[7]: from sage.all import *
n = 3
A = Matrix(Integers(2), generate_neighbord_matrix(n)) # A = adjacency
    ↪ matrix
Y = vector([1 for x in range(n**2)]) # Y = ( 1, 1, ..., 1)
X = A.solve_right(Y)
print('Solution for 3x3 board:')
print(X)
```

Solution for 3x3 board:

(1, 0, 1, 0, 1, 0, 1, 0, 1)

6.3 אלגוריתם מבוסס על מילוי עקבי

שיטת הפתרון השניה שהצגנו בעבודה. הקוד מחולק לשלושה פונקציות: פעולת דירוג של מטריצה לפי שיטה, פונקציה שפותרת את המערכת ופונקציה שעוטפת את שני הפונקציה וממש את השיטה כולה.

```
[3]: # row operation on mat to generate the solution for [n, n**2-1]
def gaussian_elimination_spanish_alg(mat : np.array, sol_vec : np.array):
    n = int(sqrt(mat.shape[0]))
    #all rows but the last one
    for i in range(0, n**2-n):
        # the lamp that is affected
        affected_lamp = i + n
        row_i = mat[i][:affected_lamp+1]
        # check rows below
        # for j in range(i+1, n**2):
        for j in [i-1 + n, i+n, i+n+1, i+ 2*n]:
            if j> -1 and j < n**2 and mat[j][affected_lamp] == 1:
```



```

        row_j = mat[j][:affected_lamp+1]
        row_j = row_j + row_i
        row_j = row_j % 2
        mat[j][:affected_lamp+1] = row_j
        sol_vec[j] = ( sol_vec[j] + sol_vec[i] ) % 2

# get result to [n, n**2-1] from solution [0, n-1]
def mul_mat_sol_based_on_res(mat : np.array, end_state : list, res : ↪list):
    n = int(sqrt(mat.shape[0]))
    for i in range(0,n**2-n):
        res_i_plus_n = int(end_state[i])
        for j in range(0,i+n):
            res_i_plus_n = (res_i_plus_n + mat[i][j] * res[j]) % 2
        res.append(res_i_plus_n)

# facade for the intire spanish method
def generate_mat_spanish_alg(mat : np.array):
    n = int(sqrt(mat.shape[0]))
    end_state = np.ones(n**2) # end_state = (1, 1, ... , 1)
    gaussian_elimination_spanish_alg(mat, end_state)
    # the matrix we need to solve for parmeter [0, n-1]
    new_mat = np.array(mat[n**2-n:n**2, 0:n], copy=True)
    # the solution vector after row operation
    new_sol = np.array(end_state[n**2-n:n**2], copy=True)

    # find solution for n variables
    A = Matrix(Integers(2),new_mat)
    Y = vector(Integers(2),new_sol)
    X = A.solve_right(Y)
    res = [x for x in X] # solution for parmeter [0, n-1]

```

```

    mul_mat_sol_based_on_res(mat, end_state, res)
    return res

mat = generate_neighbord_matrix(4)
A = Matrix(Integers(2),mat)
res = generate_mat_spanish_alg(mat)
print('solution for board n=4:')
print(res)

print('check solution by multiply matrix with souldion vector:')
X = vector(Integers(2),res)
Y = A*X
print(Y)

```

solution for board n=4:

[0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1]

check solution by multiply matrix with souldion vector:

(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)

6.4 השווה בין שתי שיטות

אחת התוצאות המעניינות היא לראות השווה זמני ריצה בין שתי אלגוריתמים.

```

[4]: import datetime
import numpy as np

def matrix_solve(mat):
    A = Matrix(Integers(2),mat)
    Y = vector([1 for x in range(n**2)])
    Z = vector([0 for x in range(n**2)])
    X = A.solve_right(Y)
    return X

```

```

val = []
# run on range(10 ,61,5)
for i,n in enumerate(range(10 ,15)):
    # print(i)
    mat = generate_neighbord_matrix(n)

    a0 = datetime.datetime.now()
    matrix_solve(mat)
    b0 = datetime.datetime.now()
    c0 = b0 - a0
    t0 = c0.total_seconds()
    # print(t0)

    a1 = datetime.datetime.now()
    generate_mat_spanish_alg(mat)
    b1 = datetime.datetime.now()
    c1 = b1 - a1
    t1 = c1.total_seconds()
    # print(t1)

    val.append((n, t0, t1))

res = np.array(val)
# np.savetxt("benchmark.csv", res, delimiter = ',')
print('board size, adj method, row by row method')
print(res)

```

board size, adj method, row by row method

```

[[10.      0.029358  0.319221]
 [11.      0.042352  0.406416]
 [12.      0.051597  0.548713]

```

```
[13.      0.064825  0.781002]
[14.      0.101306  1.072234]]
```

6.5 מציאת פתרון אופטימלי

הצגנו בפרויקט שיטה למציאת פתרונות מינמלי מבוססת פתרון מערכת משוואות על שלמים.

```
[5]: from sage.all import *
n = 3
m = 2
a = generate_neighbord_matrix_m_n(m,n)
A = Matrix(ZZ,a)
Y = vector([1 for x in range(m*n)])
Z = vector([0 for x in range(m*n)])
X = A.solve_right(Y)
print('Optimal solution:')
print(X)
```

Optimal solution:

(0, 0, 1, 1, 0, 0)

6.6 מספר פתרונות על לוח

קוד זה מציג מספר הפתרונות שיש על לוחות $m \times n$ כאשר $m, n \leq 9$.

```
[6]: def num_solution_board(m,n):
    a = generate_neighbord_matrix_m_n(m, n)
    A = Matrix(Integers(2),a)
    num_solutions = 2**A.kernel().dimension()
    return num_solutions

m = 9
n = 9
res = np.zeros((m, n), dtype= np.int32)
for i in range(1,m+1):
    for j in range(1,n+1):
        res[i-1][j-1] = num_solution_board(i,j)
print('Number solution based on m x n board size:')
print(res)
```

Number solution based on m x n board size:

	1	2	3	4	5	6	7	8	9
1	1	2	1	1	2	1	1	2	1
2	2	1	4	1	2	1	4	1	2
3	1	4	1	1	8	1	1	4	1
4	1	1	1	16	1	1	1	1	16
5	2	2	8	1	4	1	16	2	2
6	1	1	1	1	1	1	1	64	1
7	1	4	1	1	16	1	1	4	1
8	2	1	4	1	2	64	4	1	2
9	1	2	1	16	2	1	1	2	256

6.7 כל הפתרונות עבור לוח נתון

קוד זה מציג כל הפתרונות עבור לוח בגודל $m \times n$.

```
[81]: """
Phased for getting all soultion for game sized m,n
"""
def get_all_sol(m,n):
    """
    helper function to recursively sum all combinations for sol_vector +
    ↪null_vector
    """
    def get_all_sol_rec(cur_sol,index_in_null_base):
        if len(null_base) == index_in_null_base:
            all_sol.append(cur_sol)
            return
        get_all_sol_rec(cur_sol + null_base[index_in_null_base],
    ↪index_in_null_base+1)
        get_all_sol_rec(cur_sol, index_in_null_base+1)

    # generates all structer that the helper function needs
    a = genenerate_neighbord_matrix_m_n(m,n)
    A = Matrix(Integers(2),a)
    Y = vector([1 for x in range(m*n)]) # Y = ( 1, 1, ..., 1)
    X = A.solve_right(Y)

    null_base = A.right_kernel_matrix().rows()
    all_sol = []
    get_all_sol_rec(X,0)
    return all_sol

m = 2
```

```

n = 3
res = get_all_sol(m,n)
print('All solution(each solution is row vector) based on m x n board_
↳size:')
print(*res, sep = '\n')
print(f'number of souldion generated: {len(res)}')

```

All solution(each solution is row vector) based on m x n board size:

(1, 1, 0, 1, 1, 0)

(1, 0, 0, 0, 0, 1)

(0, 1, 1, 0, 1, 1)

(0, 0, 1, 1, 0, 0)

number of souldion generated: 4

מקורות

- [1] Rafael Losada Translated from Spanish by Ángeles Vallejo, *ALL LIGHTS AND LIGHTS OUT*, SUMA magazine's
- [2] Jamie Mulholland *Permutation Puzzles* Lecture 24: Light out Puzzle , SFU faculty of science department of mathematic
- [3] K. Sutner, *Linear Cellular Automata and the Garden-of-Eden*, The Mathematical Intelligencer, Vol. 11, No. 29, 1989, Springer-Verlag, New York.
- [4] אברהם ברמן, בן-ציון קון, אלגברה ליניארית, תיאוריה ותרגילים , הוצאת בק, חיפה, 1999.