המחלקה למתמטיקה שימושית

# חקירת משחק האורות


מאת :

ולדיסלב ברקנס

מנחה :

אלכס גולוורד


26 במאי 2022

# תוכן העניינים

**1 הקדמה** **2**

**2 תיאור של המשחק** **3**

2.1 תיאור גרפי של המשחק . . . . . . . 3

2.2 סוגיות בהן נעסוק בפרויקט . . . . . . . 4

2.3 תיאור משחק על גרף . . . . . . . 4

2.4 השוואה בין משחק על לוח למשחק על גרף . . . . . . . 5

**3 אלגוריתם למציאת פתרון** **7**

3.1 אלגוריתם שמבוסס על מטריצת שכנויות . . . . . . . 8

3.2 אלגוריתם שמבוסס על מילוי עקבי של שורות . . . . . . . 13

3.3 השוואה בין שתי השיטות למציאת פתרון . . . . . . . 16

3.4 דיון לגבי משחק על גרף . . . . . . . 17

**4 קיום פתרון ומספר הפתרונות עבור משחק על גרף** **17**

4.1 הוכחת קיום פתרון על גרף . . . . . . . 18

4.2 מספר הפתרונות עבור כל גרף . . . . . . . 22

**5 פתרון אופטימלי עבור לוחות מלבניים** **24**

5.1 הוכחת אי קיום לפתרון אופטימלי על לוחות ששני הממדים גדולים מ $4 \times 4$ . . . . . . . 26

5.2 אלגוריתם למציאת פתרון אופטימלי . . . . . . . 27

**6 נספחים** **28**

6.1 יצירת מטריצת שכנויות . . . . . . . 28

6.2 אלגוריתם שמבוסס על מטריצת השכנויות . . . . . . . 30

6.3 אלגוריתם מבוסס על מילוי עקבי של שורות . . . . . . . 31

6.4 השווה בין שתי האלגוריתמים . . . . . . . 33

6.5 מציאת פתרון אופטימלי . . . . . . . 36

6.6 מספר הפתרונות על לוח . . . . . . . 37

6.7 כל הפתרונות עבור לוח נתון . . . . . . . 38

# 1 הקדמה

פרויקט זה הינו פרויקט סוף של סטודנט במחלקה למתמטיקה שימושית. הפרויקט חוקר את משחק האורות המטרה המקורית של הפרויקט הייתה 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 אופטימליים, מהו פתרון אופטימלי נגדיר בהגדרה 5.1 , פתרונות אילו הם מועטים והוכחנו הגבלה לגודל הלוחות הקיימים להם פתרונות.

במהלך הפרויקט ראינו שבמשחק המתחיל כאשר כל הנורות דלוקות קיים לפחות פתרון אחד, תופעה זו העסיקה רבות את הפרויקט ונציג 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 תיאור של המשחק

משחק האורות, בלועזית Lights Out , הוא חידה המנוסחת עבור לוח משבצות מלבני שהתפרסמה כמשחק אלקטרוני. המשחק פורסם בשם זה בשנת 1995 על לוח $5 \times 5$. קיימים משחק דומים שעוד פורסמו לפני כן, כמו מרלין , Merlin , שפורסם בשנת 1970 עבור לוח $3 \times 3$. במשחק האורות 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 הנרות כבויות, למצב בו כל הנורות יהיו דולקות.

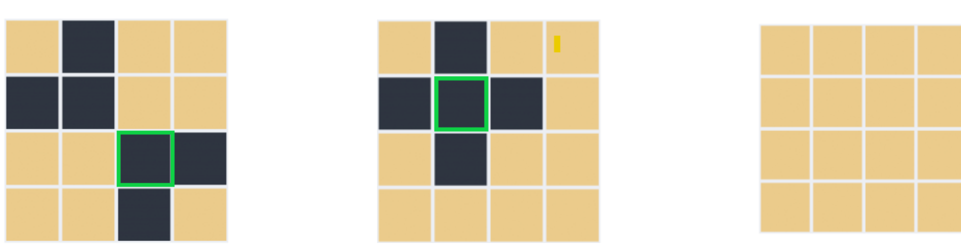
**הערה 2.1 :** מצבם התחלתי של נורות המשחק, דלוקות או כבויות, אינה משנה את תוצאות המשחק.

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 תיאור 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 2.1 : הסבר שינוי מצב הלוח לאחר לחיצה


(א) לוח במצב התחלתי
(ב) לוח לאחר לחיצה בודדת
(ג) לוח לאחר שני לחיצות

פירוט : באיור 2.1א מתואר מצב התחלתי. באיור 2.1ב ניתן לראות את השפעה של לחיצה על משבצת שמסומן בירוק. באיור 2.1ג ניתן לראות השפעה לחיצה נוספת.

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 הבא .

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 תשנה את מצבה פעם אחת.

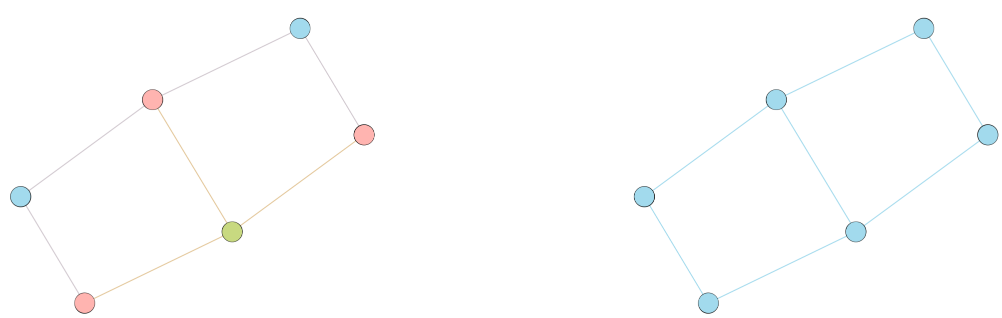
בנוסף, קיימות שאלות רבות הקשורות למשחק ובפרויקט ננסה להציג פתרון לחלקן. יתרה מזאת, נרצה להציג תופעות מעניינות, ולהראות שהמשחק אינו רק מהנה אלא גם מהווה אתגר מתמטי לא קטן.

## 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 2.2: משחק על גרף לדוגמה

(א) מצב התחלתי　　　(ב) לחיצה על משבצת מסומנת



נמחיש זאת על דוגמה שבאיור 2.2. איור 2.2א מתאר את מצב התחלתי, נסמן את המצב התחלתי של צומת בצבע כחול. איור 2.2ב מתאר לחיצה על צומת שצבוע בירוק. לחיצה זה שינתה את הצמתים השכנות למצבם הסופי שמסומן בצבע אדום.

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

בפרקים מתקדמים יותר נראה קשר בין המשחק ולאחת משיטות ייצוג גרפים , ייצוג בעזרת מטריצת שכנויות.

**הגדרה 2.1:** תהי גרף $G = (V, E)$, כאשר $V$ קבוצת הקודקודיים ו $E$ קבוצת הקשתות של הגרף. נסמן $|V| = n$. נגדיר את מטריצה $A \in \mathbb{R}^{n,n}$ כך :

$$A[i, j] = \begin{cases} 1 & \text{if } (i, j) \in E \\ 0 & \text{if } (i, j) \notin E \end{cases}$$

מטריצה $A$ נקראת מטריצת שכנויות של הגרף.

## 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.

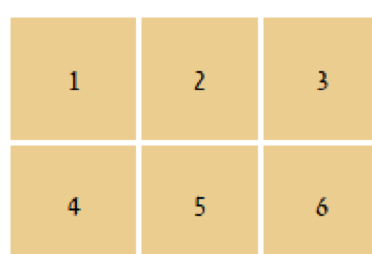
נתאר משחק על לוח כמשחק על גרף בעזרת הכללים הבאים :

1. כל משבצת במשחק על לוח נהפוך לצומת.
2. כל זוג משבצות סמוכות על לוח נחבר בקשת בגרף.

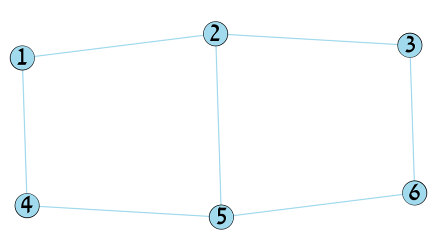
לדוגמה, ניקח לוח $3 \times 2$ נמספר את המשבצות כמו באיור 2.3א. הגרף המתקבל מתואר באיור 2.3ב.

איור 2.3 : דוגמה למשחק על לוח שתורגם למשחק על גרף

(א) משחק על לוח $3 \times 2$ שמשבצותיו ממוספר




(ב) משחק על גרף שתורגם מלוח $3 \times 2$




**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.

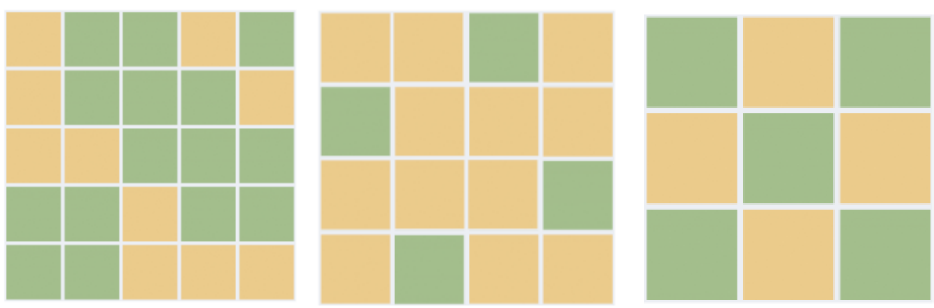
**הערה 2.4:** בעזרת השיטה שתיארנו אפשר לתאר כל משחק על לוח כמשחק על גרף, אבל ההפך הוא לא נכון. כלומר, לא כל משחק על גרף אפשר לתאר כמשחק על לוח.

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 3.1 מוצגים מספר פתרונות אפשריים ללוחות שונים, לחיצה על הלחצנים הירוקים בסדר כלשהו תוביל לפתרון המשחק. ניתן לראות שמספר הלחיצות הנדרשות לפתרון לוח $4 \times 4$ קטן ממספר הלחיצות הנדרשות לפתרון לוח $3 \times 3$. אפשר היה לחשוב שככל שהלוח גדול יותר נדרשות יותר לחיצות כדי להגיע לפתרון, אך, ניתן לראות באיור 3.1 זה לא נכון. תופעה נוספת המתקיימת במשחק היא שכמות הפתרונות עבור לוחות שונים משתנה. עבור לוח $3 \times 3$ קיים פתרון יחיד, אולם ללוח $4 \times 4$ קיימים 16 פתרונות. באופן מפתיע, ללוח $5 \times 5$ קיימים רק 4. תופעה זה מפתיע משום שאפשר היה לצפות שככל שהלוח גדול יותר כך, מספר הפתרונות יגדל. על מנת לחדד תופעה זו, נסתכל על לוחות ריבועים ($n \times n$). 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 3.1 : 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 ידי מערכת משוואות לינאריות נשתמש במשחק כפי שהוא מתואר על גרף. לחיצה על צומת משנה את מצב הצומת ומצב שכנותיה. נסמן את הצמתים ב $i$. נתאר את המשחק בצורה אלגברית:

1. כל צומת יכול להיות בשנים מצבים, את המצבים נסמן: $\{0,1\}$.
2. מצב של צומת $i$ נסמן ב $n_i$.
3. בתחילת המשחק מצבו של כל צומת הוא $0$.
4. משחק מסתיים כאשר מצבם של צמתים הוא $1$.

**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 הוא $n_i$ לאחר לחיצה תעבור למצב $n_i+1$.

**למה 3.1:** נניח שלוח נמצא במצב $X$ . לחיצה על משבצת $i$ ואחר כך על משבצת $j$ מעבירות את הלוח מצב $X$ למצב $Y$. לחיצה על משבצת $j$ ואחר כך על משבצת $i$ מעבירות את הלוח ממצב $X$ למצב $Z$ . נוכיח ש $Z=Y$.

הוכחה. אם למשבצות $i,j$ אין שכנים משותפים אז מובן ש $Z=Y$ . נניח ש $k$ משבצת שכנה ל $i$ ול $j$ . לכן מצב של $k$ משתנה פעמיים, גם כאשר לוחצים קודם על $i$ ואחר כך על $j$ וגם כאשר לוחצים קודם על $j$ . □

**מסקנה 3.1:** התכונה הזאת מאפשרת למדל את סדרת הלחיצות על ידי חיבור כי הוכחנו שהרכבה של שתי לחיצות היא פעולה קומוטטיבית וחיבור היא פעולה קומוטטיבית.

**הערה 3.3:** למה 3.1 נכונה גם עבור מספר משבצות שמעוניינים ללחוץ גדול מ 2.

**הערה 3.4:** מספר זוגי של לחיצות על צומת יחידה אינו משנה את מצב הלוח.

הוכחה. כאשר מספר הלחיצות הוא זוגי מספר השינויים של משבצת ושל שכנותיה הוא זוגי כלומר, מצבן לא ישתנה. □

**מסקנה 3.2:** בעבור משבצת מסוימת שנלחצה $m$ פעמים. מצב הלוח היה זהה אם למצב בו הינו לוחצים על אותה משבצת $m$ מודולו 2 פעמים.

מסקנה 3.2 ממחישה שכדי לתאר פתרון של משחק מספיק לתאר רק את המשבצות שנלחצו.

**למה 3.2:** מספר הווריאציות השונות של לחיצות על לוח $n \times m$ הוא $2^{m\cdot n}$

הוכחה. לפי הערה 3.2 כל לחצן יכול להיות בשתי מצבים, לכן כל לחצן יש לו 2 וריאציות, נלחץ או לא נלחץ. לפי למה 3.1 סדר הלחיצות לא משנה, לכן ללוח $m \times n$ מספר האפשרויות ללחיצה $2^{m \cdot n}$. $\square$

כדי להבין כמה גדול $2^{m \cdot n}$ נסתכל על לוח $6 \times 6$. כמות האפשרויות ללחיצה גדולה מכמות המספרים שמציגים מספרים שלמים במחשב ( 4 בתים ). המספר הגדול ביותר שאפשר להציג בעזרת 4 בתים הוא $2^{32} - 1$. המטרה של המחשה זו היא להדגיש כמה לא פרקטי לנסות לפתור בעזרת מעבר על כל האופציות.

נתאר את המשחק בצורה וקטורית ( בעזרת הערה 3.2 ). ניקח לדוגמה משחק בגודל $2 \times 2$, נתאר את הלוח במצבו התחלתי כמטריצה

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

נתאר לחיצה על משבצת $(1,1)$:

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

כפי שתיארנו בהערה 3.2 אפשר לתאר שינוי מצב הנורה על ידי חיבור מצבה 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 בבסיס 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 המתאר הלחיצה על משבצת מסוימת יקרא וקטור שינוי שלה.

**הגדרה 3.1:** תהי משחק על גרף בעל $n$ צמתים הממוספרים מ 1 עד $n$, וקטור שינוי $\vec{t_i}$ של צומת $i$ הוא:

- וקטור השייך $\mathbb{Z}_2{}^n$.
- וקטור שתוצאת החיבור עם וקטור לוח הוא וקטור לוח לאחר לחיצה על $i$.

כדי לבנות וקטור שינו $\vec{t_i}$:

- ערכי הוקטור שיקבלו ערך 1 היו באינדקסים של הצומת ושכנותיה
- שאר הערכי וקטור היו 0.

**הגדרה 3.2:** וקטור המתאר את מצב הלוח יקרא וקטור הלוח.

**הערה 3.5:** שיטת המספור בפרויקט היא מעבר על שורות ואז על עמודות כמו שמתואר באיור 3.2.

איור 3.2: שיטת מספור משבצות על לוח


**דוגמה 3.1:** תהי גרף בעל $4$ צמתים, כפי שמתואר באיור 3.3. צמתים שמצבם $1$ יצבעו באדום, בכחול יצבעו צמתים שמצבם $0$. נדגים צירוף לחיצות במשחק בעזרת וקטורי השינויים ווקטור הלוח.

באיור 3.3א מתואר גרף במצבו הנוכחי. נראה את שינוי הגרף לאחר לחיצה על הצמתים $1,3$.

וקטור שינוי של צומת $1$:

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

נשרשר את וקטורי השינוי של שתי הלחיצות:

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

אם נחבר וקטור זה עם וקטור הלוח שנסמן ב $\vec{S_0}$ נקבל:

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

וקטור הלוח שמתקבל לאחר חיבור אכן תואם לתוצאה המצופה, כפי שמתואר באיור 3.3ב.

איור 3.3 : דוגמה לתיאור וקטור שינוי במהלך משחק על גרף


**הערה 3.6:** היות ווקטור שינוי שייך לשדה $\mathbb{Z}_2{}^n$, ניתן לתאר צירוף לחיצות במשחק בעזרת צירוף לינארי בשדה $\mathbb{Z}_2{}^n$. כפי שתיארנו במסקנה 3.2 סקלרים בצירוף לינארי זה הם $0$ או $1$.

**הערה 3.7:** מכיוון שכל משחק מתחיל כאשר מצב כל הצמתים הינו $0$, ניתן להשמיט את וקטור הלוח ההתחלתי.

כאשר נסמן מצב הלוח ההתחלתי ב $S_0$ , מתקיים :

$$\vec{S_0} + \sum_{j=1}^{n} \vec{t_j} x_j = \sum_{j=1}^{n} \vec{t_j} x_j \tag{1}$$

בעקבות כך ניתן לתאר את בעיית המשחק בצורה הבאה :

$$\sum_{j=1}^{n} \vec{t_j} x_j = \vec{1} \tag{2}$$

כאשר $\vec{1}$ מתאר את וקטור הלוח כאשר המשחק פתור. $n$ מתאר את מספר הצמתים בגרף. נשים לב שאם ידוע צירוף $x = \begin{bmatrix} x_1, & x_2, & \cdots, x_n \end{bmatrix}$ שמקיים את המשוואה 2 אז הוא פתרון של משחק על גרף. כדי להגיע לפתרון על גרף נלחץ על הצמתים שמספרם שווה לאינדקסים $j$ שמקיימים $x_j = 1$ בצירוף $x$.

מערכת משוואות שמתוארת בנוסחה 2 אפשר לתאר בעזרת מטריצה כמו שמתואר בנוסחה 3.

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

$$\begin{bmatrix} t_{1,1} & t_{1,2} & \cdots & t_{1,n} \\ t_{2,1} & t_{2,2} & \cdots & t_{2,n} \\ \cdots & \cdots & \cdots & \cdots \\ t_{i,j} & t_{i,2} & \cdots & t_{i,n} \\ \cdots & \cdots & \cdots & \cdots \\ t_{n,1} & t_{n,2} & \cdots & t_{n,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_i \\ \cdots \\ x_n \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ \cdots \\ 1 \\ \cdots \\ 1 \end{bmatrix} \tag{3}$$

נשים לב שלמטריצה במשוואה 3, נסמנה ב $A$, ערכי המטריצה שמקיימים $A_{i,j} = 1$ הם באינדקסים $i, j$, בהם צמתים $n_i, n_j$ שהם שכנים או זהים ( $i = j$ ).

**הערה 3.8:** נשים לב שהמטריצה במשוואה 3 כמעט זהה למטריצת שכנויות של הגרף שהגדרנו בהגדרה 2.1. ההבדל היחיד הוא שהאלכסון הראשי במטריצה שבמשוואה 3 מורכב כולו מערך $1$. במידה והינו מגדירים את הגרף כך שכל צומת הינה שכנה לעצמה, נקבל זהות בין מטריצת שכנויות של הגרף למטריצה במשוואה 3.

**הגדרה 3.3:** מטריצה שקבלנו במשוואה 3 תקראה מטריצת שכנויות של משחק.

**הערה 3.9:** היות ובגרף יחס שכנות הוא סימטרי מטריצה שכנות היא גם סימטרית.

**דוגמה 3.2:** מטריצה שכנות עבור גרף באיור 3.3 :

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

**הגדרה 3.4:** וקטור פתרון של משחק הוא וקטור $\vec{x}$ שניראה במשוואה 3.

נזכיר שאם $\vec{x}$ וקטור פתרון של המערכת ו $x_i = 1$ אז כדי לפתור משחק צריך ללחוץ על לחצן $i$. בנוסף נציין שאתכן קיימים כמה פתרונות אפשריים.

**הגדרה 3.5:** שיטת פתרון הנעזרת ביצירת מטריצה שכנויות ומציאת וקטור פתרון תקראה אלגוריתם מבוסס מטריצת שכנויות.

לפי מה שידוע לנו השיטה שקראנו לה בפרויקט שיטת מטריצת השכנויות לראשונה הופיעה במאמר של K. Sutner [2]. נציין שבמאמר של K. Sutner לא רק הוצגה שיטת זו אלה גם ניתנה הוכחה לקיום פתרון. מרגע שהצלחנו לתאר את הבעיה בצורה משוואות לינארית על שדה $\mathbb{Z}_2{}^n$, נוכל להיעזר בכלים של אלגברה לינארית כדי למצוא פתרון, כמו מציאת פתרון בעזרת דירוג או מציאת מטריצה פסאודו הפוכה וכולי.

**הערה 3.10:** עבור משחק על לוח $m \times n$ גודל מערכת המשוואות המתקבל משיטה מבוססת מטריצת שכנויות הוא $m \cdot n$ משתנים ומשוואות.

עבור לוח $[m \times n]$ מטריצת השכנות בגודל $[m \cdot n \times m \cdot n]$. השאלה שנרצה לענות בפרק הבאה היא: האם קיימות שיטות לפתור את המשחק, בעזרת מערכת המשוואות יותר מצומצמת?

### 3.2 אלגוריתם שמבוסס על מילוי עקבי של שורות

בפרק 3.1 תיארנו איך מתקבלת מערכת משוואות לינאריות מעל שדה $\mathbb{Z}_2$. מטריצה של מערכת הזאת היא מטריצת שכנויות. עכשיו נראה איך אפשר להגיע לאותה מערכת משוואות על בסיס של שיקולים שונים .נראה גם שבמקום לפתור את המערכת הזאת ניתן לפתור מערכת הרבה יותר קטנה. השיטה שנציג מבוססת על המאמר [1]. לגישה החדשה ניקרא "אלגוריתם שמבוסס על מילוי עקבי של שרות" או בקצרה מילוי עקבי.

ללא מגבלת הכלליות נסביר את הגישה החדשה עבור לוח משבצות $3 \times 3$. נשייך לכל משבצת שלו משתנה באופן הבא.

| | | |
|---|---|---|
| $x_1$ | $x_2$ | $x_3$ |
| $x_4$ | $x_5$ | $x_6$ |
| $x_7$ | $x_8$ | $x_9$ |

כל משתנה מסמן האם משבצת שלו נלחצה מספר זוגי או מספר אי-זוגי של פעמים כאשר עברנו ממצב הלוח בו כל המשבצות כבויות למצב בו כל המשבצות דלוקות. אז $x_i = 1$ כאשר משבצת הזאת נלחצה מספר אי-זוגי של פעמים ו- $x_i = 0$ כאשר משבצת הזאת נלחצה מספר זוגי של פעמים. נציין שמשבצת מסוימת תהיה דלוקה אחרי סדרת לחיצות אם סכום לחיצות עליה ושכנות שלה הוא מספר אי-זוגי. כאשר ננסח את התנאי הזה עבור כל משבצת נקבל מערכת משוואות מעל שדה $\mathbb{Z}_2$. ממה שכתבנו קודם נובע שמטריצת מקדמים שלה היא מטריצת שכנויות.

איור 3.4: מטריצת השכנויות עבור לוח $3 \times 3$

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

נדגים עבור משבצת עם משתנה $x_1$. על מנת שהיא תהיה דלוקה אחרי סדרת לחיצות סכום לחיצות עליה ועל שכנותיה חייב להיות אי-זוגי, כלומר, בשדה $\mathbb{Z}_2$ מתקיים השוויון

$$x_1 + x_2 + x_4 = 1$$

עבור משבצת $x_2$ מתקבל השוויון

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

עכשיו נראה איך ניתן במקום מערכת עם מטריצת שכנויות שעבור לוח $3 \times 3$ לבנות מערכת הרבה יותר קטנה. מערכת המשוואות שנקבל תהיה מבוססת על שלושת המשתנים של משבצות בשורה הראשונה $x_1, x_2, x_3$.

לשם כך נחלץ את $x_4$ ממשוואה ראשונה:

$$x_4 = 1 + x_1 + x_2$$

נחלץ $x_5$ ממשוואה שנייה:

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

נחלץ את $x_6$ ממשוואה שלישית ונקבל:

$$x_6 = 1 + x_2 + x_3$$

נחלץ את $x_7$ ממשוואה רביעית, כאשר נעזר בערכי המשתנים שכבר חולצו:

$$x_1 + x_4 + x_5 + x_7 = 1 \Rightarrow x_1 + (1 + x_1 + x_3) + (1 + x_1 + x_2 + x_3) + x_7 = 1$$

לכן מתקבל:

$$x_7 = 1 + x_3$$

נחלץ את $x_8$ ממשוואה חמישית:

$$x_8 = 0$$

נחלץ את $x_9$ ממשוואה שישית: נקבל:

$$x_9 = 1 + x_1 + x_3$$

היות וחילצנו את כל המשתנים פרט למשתנים $x_1, x_2, x_3$ נפשט את המשוואות הנותרות ונקבל מערכת של 3 משוואות עם שלושה משתנים.

פישוט משוואה שביעית:

$$x_1 + x_2 + x_3 = 1$$

פישוט משוואה שמינית:

$$x_2 + x_3 = 0$$

פישוט משוואה תשיעית:

$$x_1 + x_2 = 1$$

למערכת המשוואות יש פתרון יחיד והוא $x_1 = 1, x_2 = 0, x_3 = 1$ .

מפה נובע ש $x_4 = 0, x_5 = 1, x_6 = 0, x_7 = 1, x_8 = 0, x_9 = 1$.

נבחין שכל הפעולות שנעשו בגישה מילוי עקבי הם פעולות אלמנטריות. לכן מימוש השיטה בקוד נעשה ע"י שימוש בפעולות אלמנטריות של גאוס אבל סדר של פעולות האלה שונה מסדר של פעולות אלמנטריות בשיטת הדירוג גאוס. סדר פעולות היה לפי סדר חילוץ המשתנים שהצגנו בגישה מילוי עקבי. נציג את סדר פעולות הדירוג על אותה דוגמה, על לוח $3 \times 3$. נציין שפעולות הדירוג כמובן מתבצעות על מטריצת המורחבת כלומר מטריצת השכנויות עם וקטור הלוח הפתור $\vec{1}$.

כאשר תיארנו את האלגוריתם מילוי עיקבי עבור שלושת המשוואות הראשונות רק חילצנו את המשתנים ולכן עבור שלושת השורות הראשונות במטריצה לא נבצע אף פעולה אלמנטרית.

עבור משוואה רביעית הצבנו את המשתנים המחולצים של $x_4, x_5$, לכן עבור משוואה רביעית ננסה להיפטר ממשתנים עלו בעזרת פעולות שורות:

$$r_4 \leftarrow r_4 + r_1$$

$$r_4 \leftarrow r_4 + r_2$$

עבור משוואה חמישית הצבנו את המשתנים המחולצים של $x_4, x_5, x_6$, לכן עבור משוואה חמישית ננסה להיפטר ממשתנים עלו בעזרת פעולות שורות:

$$r_5 \leftarrow r_5 + r_1$$

$$r_5 \leftarrow r_5 + r_2$$

$$r_5 \leftarrow r_5 + r_3$$

עבור משוואה שישית הצבנו את המשתנים המחולצים של $x_5, x_6$, לכן עבור משוואה שישית ננסה להיפטר ממשתנים עלו בעזרת פעולות שורות:

$$r_6 \leftarrow r_6 + r_2$$

$$r_6 \leftarrow r_6 + r_3$$

עבור משוואה שביעית הצבנו את המשתנים המחולצים של $x_7, x_8$, לכן עבור משוואה שביעית ננסה להיפטר

ממשתנים עלו בעזרת פעולות שורות:

$$r_7 \leftarrow r_7 + r_4$$
$$r_7 \leftarrow r_7 + r_5$$

עבור משוואה שמינית הצבנו את המשתנים המחולצים של $x_7, x_8, x_9$, לכן עבור משוואה שמינית ננסה להיפטר ממשתנים עלו בעזרת פעולות שורות:

$$r_8 \leftarrow r_8 + r_4$$
$$r_8 \leftarrow r_8 + r_5$$
$$r_8 \leftarrow r_8 + r_6$$

עבור משוואה תשיעית הצבנו את המשתנים המחולצים של $x_7, x_8$, לכן עבור משוואה תשיעית ננסה להיפטר ממשתנים עלו בעזרת פעולות שורות:

$$r_9 \leftarrow r_9 + r_5$$
$$r_9 \leftarrow r_9 + r_6$$

לאחר פעולות דירוג מערכת המשוואות המתקבלת בשורות שבע שמונה ותשעה היא:

$$x_1 + x_2 + x_3 = 1$$
$$x_2 + x_3 = 0$$
$$x_1 + x_2 = 1$$

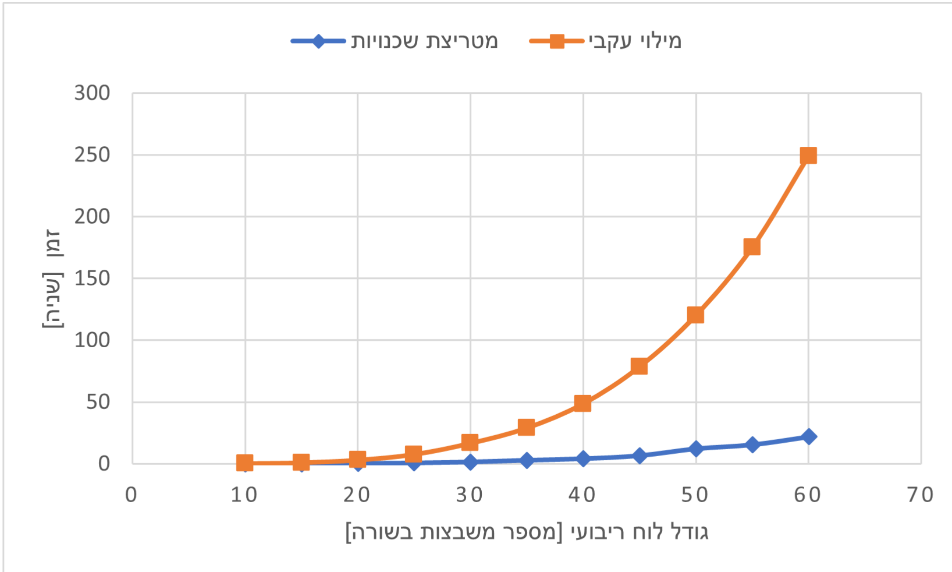
### 3.3 השוואה בין שתי השיטות למציאת פתרון

חישוב סיבוכיות של דירוג מטריצה כללית בגודל $n^2 \times n^2$ הוא $O(n^2 \cdot n^4) = O(n^6)$.

דירוג שורה בשיטת מילוי עקבי מבצעים פעולות שורות כמספר השכנים עבור אותה משוואה. לכן יש לכל יותר 4 פעולות שורות שצריך לבצע כדי לקבל מערכת משוואות מצומצמת. מחישוב הסיבוכיות פעולה זה תיקח $O(n^2 \cdot n^2) = O(n^4)$. דירוג $n$ המשתנים הנותרים הוא בסיבוכיות $O(n \cdot n^2) = O(n^3)$.

נראה את הסיבוכיות בפועל על ידי הצגת זמני חישוב. באיור 3.5 אפשר לראות את הביצועים של שני האלגוריתמים, ציר ה $x$ מתאר גודל שורה של לוח ריבועי. ציר ה $y$ מתאר זמן בשניות שלוקח לאלגוריתם לרוץ. לפי התוצאות של איור 3.5 ניראה ששיטה למילוי עקבי שבתאוריה יותר אופטימליות לוקחת יותר זמן. אחת הסיבות לקח שפונקציה שפותרת מערכת משוואות הינה פונקציית ספרייה, היודע כאופטימלי.

איור 3.5 : גרף מתאר ביצועים על לוח ריבועי גודל שורה מול זמן




### 3.4 דיון לגבי משחק על גרף

תיארנו את המשחק על גרף מסיבות הבאות :

1. ככול שהמבנה כללי יותר התאוריה שמפתחים מתאימה ליותר בעיות.
2. תאוריית הגרפים רחבה מאד וקל לתאר בעזרתה בעיות.
3. מבליט את מהות הבעיה והגדרה הבסיסית ביותר של המשחק.

בפועל כשהצגנו את האלגוריתמים לפתרון משחק התגלתה התמונה המלאה. כלומר שני האלגוריתמים שתיארנו מתארים את המשחק כגרף, היות ושני האלגוריתמים בנויים על מטריצת השכנויות, מטריצת שכנויות היא שיטת ייצוג קלאסי לגרפים. קשר זה מדגיש ומראה שלפעמים רק תיאור מדויק של הבעיה מספיק כדי למצוא לבעיה פתרון.

## 4 קיום פתרון ומספר הפתרונות עבור משחק על גרף

עד כה הצגנו שיטות למציאת פתרון, שיטות אלו 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]. עובדה מעניינת נוספת שהוצגה במאמר היא, שיש רק הוכחה שמבוססת על אלגברה לינארית לשאלת קיום פתרון.

## 4.1 הוכחת קיום פתרון על גרף

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

**הגדרה 4.2:** לכל שני וקטורים $\vec{x}, \vec{y} \in \mathbb{Z}_2{}^n$ נגדיר פעולה הבאה:

$$\vec{x} \cdot \vec{y} = \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{bmatrix} \cdot \begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_n \end{bmatrix} = x_1y_1 + x_2y_2 + \cdots x_ny_n \tag{4}$$

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

**הערה 4.1:** פעולה שהגדרנו בהגדרה 4.2 נקראת מכפלה סקלרית למרות שהיא מקיימת רק 3 מתוך 4 תכונות של מכפלה סקלרית ב - $R^n$. תכונה $\vec{u} = \vec{0} \Leftrightarrow < \vec{u}, \vec{u} >= 0$ לא מתקיימת.

**דוגמה 4.1:** המחשה להערה 4.1:

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

**הערה 4.2:** וקטורים $\vec{x}, \vec{y} \in \mathbb{Z}_2{}^n$ יקראו מאונכים אחד לשני אם, תוצאת מכפלה הסקלרית ביניהם שווה ל 0 כלומר, $\vec{x} \cdot \vec{y} = 0$. וקטורים מאונכים זה לזה יסומנו ב $\vec{x} \perp \vec{y}$.

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

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

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

□

**הערה 4.3:** עבור המכפלה הסקלרית שהגדרנו על השדה הוקטורי $\mathbb{Z}_2{}^n$ לא לכל מטריצה $A$ מתקיים:

$$\mathrm{Col}A \cap \mathrm{Nul}A^T = \{\vec{0}\}$$

**דוגמה 4.2:** המחשב להערה 4.3:

עבור מטריצה:

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

מתקיים:

$$\begin{bmatrix} 1 \\ 1 \end{bmatrix} \in \mathrm{Col}A \cap \mathrm{Nul}A^T$$

**משפט 4.2:** לכל משחק על גרף קיים פתרון.

הוכחה. תהי $A \in Z_2{}^{n\times n}$ מטריצת השכנויות של משחק שהגדרנו ב 3.5. למטריצה זו מתקיימים תכונות הבאות:

1. מטריצה סימטרית לפי הערה 3.9
2. האיברים על האלכסון מטריצה ערכם שווה ל $1$.

כדי להראות שלמשחק יש פתרון צריך להראות שקיים פתרון למערכת:

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

במקרה ש $A$ מטריצה הפיכה קיים פתרון, ופתרון יחיד. עבור המקרה שמטריצה אינה הפיכה כלומר, $\mathrm{Nul}A \neq \{\vec{0}\}$. תהי $\vec{x} \in \mathrm{Nul}A$ מהגדרה זו מתקיים $A\vec{x} = \vec{0}$ לכן:

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

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

$$\vec{x}^T A\vec{x} = a_{1,1}x_1^2 + 2(a_{1,2} + a_{2,1})x_1x_2 + \cdots + 2(a_{1,n} + a_{n,1})x_1x_n + \\ + a_{2,2}x_2^2 + 2(a_{2,3} + a_{3,2})x_2x_3 + \cdots + 2(a_{2,n} + a_{n,2})x_2x_n + \cdots \quad (5)$$

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

$$a_{i,j} - a_{j,i} = a_{i,j} + a_{j,i} = 0$$

לכן את המשוואה 5 אפשר לפשט כך:

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

היות ומתקיים $x_i^2 = x_i$ כי $x_i \in \mathbb{Z}_2$ $i \in 1, \cdots, n$ לכן, ניתן לפשט את משוואה 5:

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

היות ו $\vec{x}^T A\vec{x} = 0$ לכן מתקיים:

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

כלומר $\vec{x} \perp \vec{1}$ כאשר $x \in \mathrm{Nul}A$ לפי משפט 4.1 מתקבל $\vec{1} \in \mathrm{Col}A^T$. מטריצה $A$ הינה סימטרית לכן $A^T = A$ ומתקיים $\vec{1} \in \mathrm{Col}A$. למעשה הוכנו שקיים $\vec{x}$ שמקיים פתרון למערכת $A\vec{x} = \vec{1}$ כלומר, קיים פתרון למשחק. $\square$

הוכחת קיום הפתרון עבור המשחק כפי שהגדרנו על גרף הושגה, מסקנה נאיבית שניתן אולי לחשוב היא, שלכל מצב התחלתי אפשרי היה לפתור את המשחק. בחלק זה של הפרק ננסה לחדד מתי קיים פתרון כאשר מצבים התחלתיים וסופיים של אותו משחק שונים מהמשחק המקורי.

**הגדרה 4.3:** 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 הוא שחלק מנורות דולקות וחלק כבויות ורוצים להגיע למצב סופי שגם בו חלק מנורות דולקות וחלק כבויות.

**הערה 4.4:** רק בפרק זה נשתמש במשחק כפי שהגדרה בהגדרה 4.3. בכל שאר פרקים נשתמש בהגדרה המקורית של המשחק.

עבור המשחק שהגדרנו 4.3 אותו קורא נאיבי יכול להניח שגם עבור משחק שכזה תמיד קיים פתרון.

**דוגמה 4.3:** דוגמה למשחק על לוח לפי הגדרה 4.3, שאין לו פתרון. תהי לוח $1 \times 2$ בו מצב התחלתי הוא שהנורה הימנית ביותר דלוקה ונרצה לעבור למצב הסופי בו כל הנורות דולקות. נראה שהמשחק אינו פתיר לפי הצגת כל הלוחות האפשריים להתקבל במשחק על ידי כל הצירופים השונים של לחיצות על הלוח. אם נמספר את המשבצות לפי שיטת המספור שציינו בהערה 3.5, אז אוסף כל צירופי הלחיצות הם:

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

כפי שניתן לראות באיור 4.1 באף צירוף לחיצות לא ניתן להגיע למצב הסופי שהוגדר במשחק.

איור 4.1: מצבי הלוחות לאחר לחיצה של צירוף

| (א) עבור צירוף $()$ | (ב) עבור צירוף $(1)$ | (ג) עבור צירוף $(2)$ | (ד) עבור צירוף $(1,2)$ |
|---|---|---|---|
| $\begin{bmatrix} 0 & 1 \end{bmatrix}$ | $\begin{bmatrix} 1 & 0 \end{bmatrix}$ | $\begin{bmatrix} 1 & 0 \end{bmatrix}$ | $\begin{bmatrix} 0 & 1 \end{bmatrix}$ |

כדי לבדוק אם קיים פתרון למשחק הכללי שהגדרנו בהגדרה 4.3 נוכל להיעזר בהוכחה 4.2.

**משפט 4.3:** למשחק קיים פתרון אם ורק אם וקטור הפרש בין מצב סופי ומצב ההתחלתי שייך למרחב העמודות של מטריצת שכנויות.

הוכחה. נגדיר קודם את המשחק שהגדרנו בהגדרה 4.3 אלגברית. לפי משוואה 1 אפשר לנסח אלגברית את המשחק כך. תהי $A$ מטריצת השכנויות, $\vec{S_0}$ מצב התחלתי של המשחק, ו $\vec{S_e}$ מצב הסופי של המשחק. נחפש צירוף לא סדור של לחיצות $\vec{x}$ כך שמתקיים:

$$\vec{S_0} + A\vec{x} = \vec{S_e}$$

נעביר אגפים ונקבל:

$$A\vec{x} = \vec{S_e} - \vec{S_0}$$

הוכחה של משפט 4.2 בנויה על עובדה שצריך להוכיח שווקטור $\vec{1}$ שייך ל $\mathrm{Col}A$. במקרה של המשחק החדש כדי להראות שקיים פתרון למשחק מספיק להוכיח ש $\vec{S_e} - \vec{S_0}$ שייך ל $\mathrm{Col}A$. □

**הערה 4.5:** כדי לבדוק שוקטור $\vec{v} \in \mathbb{R}^n$ שייך ל $\mathrm{Nul}A$ של מטריצה $A \in \mathbb{R}^{m\times n}$ מספיק להראות שמתקיים:

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

**הערה 4.6:** בגלל הערה 4.3 לא ניתן לומר שוקטורים $z \in \mathbb{Z_2}^n$ המקיים $z \in \mathrm{Col}A$ אז הוא לא שייך ל $\mathrm{Nul}A$. לו היה ניתן לומר זאת, היה קל לבדוק שלמשחק קיים פתרון על ידי בדיקה:

$$\vec{S_e} - \vec{S_0} \notin \mathrm{Nul}A$$

**משפט 4.4:** תהי משחק לפי הגדרה 4.3, $\vec{S_e}$ מצב הסופי, $\vec{S_0}$ התחלתי של משחק, ו $A$ מטריצת שכנויות. אם $\vec{S_e} - \vec{S_0} \notin \mathrm{Nul}A$ אז למשחק קיים פתרון.

הוכחה. אם $\vec{S_e} - \vec{S_0} \notin \mathrm{Nul}A$ אז בהכרח

$$\vec{S_e} - \vec{S_0} \in \mathrm{Col}A^T = \mathrm{Col}A$$

לפי משפט 4.3 למשחק קיים פתרון. □

משפט 4.4 יכול להקל על הבדיקה אם המשחק פתיר. אך, קיימים מקרים שמתקיים:

$$\vec{S_e} - \vec{S_0} \in \mathrm{Nul}A$$

ועדיין קיים למשחק פתרון.

## 4.2 מספר הפתרונות עבור כל גרף

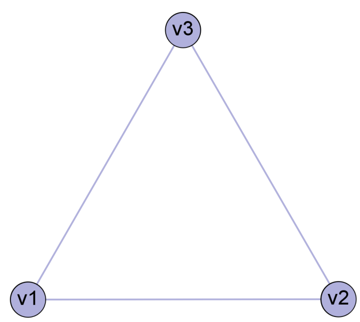
הוכחנו שלכל משחק על גרף יש פתרון שסדר לחיצות אינו משנה את התוצאה על הלוח. השאלה שנשאל בפרק זה מה אפשר לומר על מספר פתרונות בעזרת הכלים שהצגנו. נציין ששני פתרונות יקראו שונים אם, קיים לפחות לחצן אחד שמבדיל בין הפתרונות. זאת אומר שקיים לחצן ששייך לפתרון ראשון ולא שייך לפתרון שני. כפי שציינו קודם סדר הלחיצות לא משנה את הפתרון לכן, פתרון הינו קבוצה של לחצנים שצריכים להילחץ כדי להגיע למצב סופי של המשחק.

**דוגמה 4.4:** נרצה להראות שקיימים מספר פתרונות למשחק. ניקח לדוגמה משחק על גרף שמתואר באיור 4.2. היות וגרף הינו קליקה לכן לחיצה בודדת על אחד הצמתים תדליק את כל הלחצנים. מתקבל שקבוצה

$$G = \{\{v_1\}, \{v_2\}, \{v_3\}\}$$

היא חלק מקבוצת הפתרונות של המשחק על גרף. כפי שניתן לראות קיימים לפחות שלושה פתרונות למשחק הנתון.

איור 4.2: משחק על גרף




נשאל פרק זה איך אפשר למצוא את כמות הפתרונות במשחק.

**הגדרה 4.4:** דרגת חופש $F(A)$ של מטריצה $A \in \mathbb{R}^{m \times n}$ היא מספר העמודות של המטריצה פחות דרגה של המטריצה. נתאר את דרגת החופש בעזרת הנוסחה הבאה:

$$F(A) = n - rank(A)$$

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

**משפט 4.5:** מספר הפתרונות של משחק שווה ל $2^k$. כאשר $k$ שווה לדרגת החופש של מטריצה השכנויות של המשחק.

הוכחה. נזכיר שאת מטריצה השכנויות הגדרנו בהגדרה 3.3. נסמן ב $A$ את מטריצת השכנויות, ב $X$ את כל הוקטורים שהם פתרון של המערכת:

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

לפי משפט 4.2 ידוע שקיים לפחות פתרון אחד למשחק. היות וידוע שיש לפחות פתרון אחד אפשר לתאר את כל פתרונות כך:

$$X = X_n + x_0$$

כאשר $X_n$ קבוצת כל הוקטורים השייכים לפתרון מרחב האפס של מטריצה $A$, $x_0$ פתרון פרטי, ו $X$ קבוצת כל פתרונות של המשחק. לכן גודל קבוצת הפתרונות שווה לגודל מרחב האפס. ידוע שגודל מרחב האפס תלוי בדרגת החופש לכן, מספר הווקטורים בבסיס מרחב האפס שווה לדרגת החופש שנסמן ב $k$. כמות הווקטורים במרחב האפס שווה לכל וקטורים שמוגדרים בצירוף לינארי:

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

כאשר הערכים של $a_i \in Z_2$ לכן, לכל מקדם יכול להיות 2 ערכים. היות ווקטורים בלתי תלויים לינארית, קיבלנו שגודל קבוצת הפתרונות הוא בדיוק $2^k$. $\square$

הבחנה נוספת שנציין, אפשר לחסום את כמות הפתרונות. חסם עליון טריוויאלי לכמות המקסימלית של פתרונות הוא $2^n$ פתרונות כאשר $n$ שווה למספר הלחצנים. זאת מכיוון שלא יכול להיות יותר פתרונות מאשר כמות הלחיצות השונות האפשריות במשחק. השאלה שנשאלת האם אפשר למצוא חסם הדוק יותר?

**הערה 4.7:** עבור משחק לוח מלבני בגודל $m \times n$ קיים לכל יותר $2^k$ כאשר $k = \min\{m, n\}$ פתרונות שונים

הוכחה. הערה זה נכונה לפי אלגוריתם שמבוסס על מילוי עקבי של שורות שהגדרנו ?? ניתן לתרגם את משחק ל $k$ משוואות ש $k$ יכול להיות מספר שורות או עמודות. $\square$

לסיכום נציג באיור 4.3 את כמות הפתרונות שיש בלוח. השורות והעמודות בטבלה מייצגות את ממדי הלוח. טבלה זאת חושבה על ידי משפט 4.5.

איור 4.3: 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 |

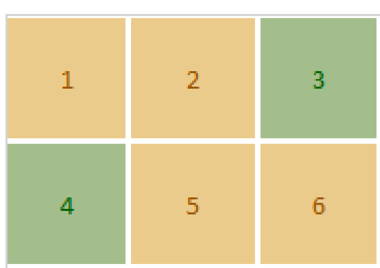
## 5 פתרון אופטימלי עבור לוחות מלבניים

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

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

באיור 5.1 ניתן דוגמא לפתרון מינמלי בלוח $3 \times 2$. כשלוחצים על לחצנים $3,4$ כל הנורות נדלקות, ואף אחת מהם לא נכבית באף לחיצה.

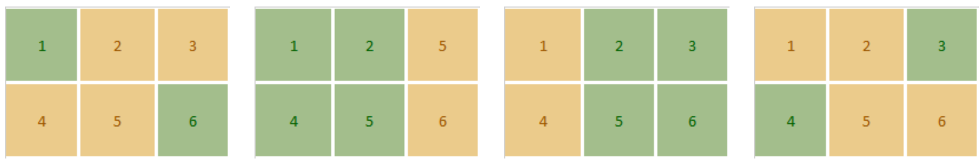
איור 5.1: פתרון מינמלי של משחק




נרצה לחדד ולהדגיש עד כמה קל למצוא פתרונות אופטימליים. אם ניקח לוח $3 \times 2$ כפי שמתואר באיור 5.1, ונתבונן במספר כל הפתרונות שיש ללוח זה, כפי שמתואר בטבלה 4.3, ניראה שיש 4 פתרונות. שני פתרונות אופטימליים ושני פתרונות לא אופטימליים, נזמין את הקורא לחפש את כל הפתרונות. כנראה ששני הפתרונות

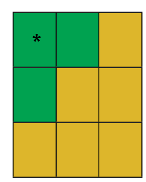
האופטימליים נמצאו מידית. כנראה שכדי למצוא את הפתרונות הנותרים נצטרך לקחת דף ועט, ולחפש אותם גם עבור לוח בממד מצומצם שכזה. את כל ארבעת הפתרונות נציג באיור 5.2.

איור 5.2 : משחק על גרף לדוגמה

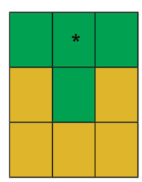



נשים לב ניסוח של משחק האורות בו הפתרונות המתקבלים הם רק פתרון אופטימלי מזכיר, מאוד משחקים כמו פאזלים או טטריס. הדימיון בין המשחקים נובע מכיוון שהמטרה במשחקים אלו היא למלאה אזור ריק במספר צורות שונות. משחק האורות בו הפתרונות המתקבלים הם רק פתרונות אופטימליים, נהיה למשחק שמנסים למלא את הלוח באמצעות לבנים בצורה שונות. משחק שכזה זה הוא מקרה פרטי של אוסף משחקי ריצוף של אבני פוליאומינו. פוליאומינו הוא אובייקט קומבינטורי המורכב מריבועים המחוברים זה לזה, על ידי הצמדת צלעות הריבועים. במשחק האורות בו הפתרונות המתקבלים הם רק פתרונות אופטימליים, אז הפוליאומינו של המשחק הם בצורות :

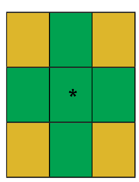
צורת פינה



צורת שפה לא בפינה



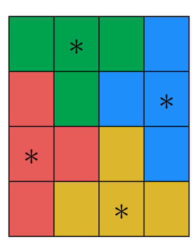
צורת פנימית



כאשר המשבצות שנלחצו מסומנות בסימן $*$.

חיפוש פתרון אופטימלי על לוח נהיה שקול לשאלה : האם אפשר לרצף את הלוח עם פוליאומינו של המשחק?

דוגמה לריצוף משחק $4 \times 4$



בפרק זה נפתור את השאלה : 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 $4 \times 4$

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

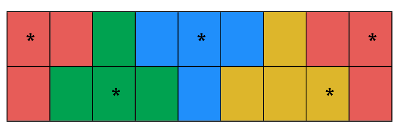
הוכחה. נוכיח בעזרת אינדוקציה. נתחיל על לוח $1 \times 2$, כל לחיצה בודדת על משבצת מובילה לפתרון אופטימלי של המשחק. נניח וקיים לוח $m \times 2$ בו מתקיים פתרון אופטימלי כאשר $m$ אי זוגי. ללוח קיים פתרון אופטימלי לכן הנורות השתנו רק פעם אחת, מה שאומר שמשבצות בעמודה האחרונה בהכרח שונו על ידי אותה לחיצה. לחיצה זו בהכרח תהיה על משבצת בעמודה האחרונה. נניח, בלי הגבלת הכלליות, שהמשבצת שנלחצה היא בשורה 1. עבור לוח $(m+2) \times 2$ נבצע את אותם מהלכים כמו שביצענו עבור לוח $m \times 2$. נשים לב שהנורות שנותרו במצבם ההתחלתי הן:

$$\{n_{1,m+2}, n_{2,m+1}, n_{2,m+2}\}$$

נשים לב שלחיצה על משבצת $n_{2,m+1}$ פותרת את המשחק. $\square$

**הערה 5.1:** טענה 5.1 אינה נכונה עבור לוח $m$ זוגי. לדוגמה ללוח $2 \times 2$ כל לחיצה בודדת תוביל למשבצת בודדת שתישאר במצבה ההתחלתי. לכן, כל לחיצה נוספת תשנה את שאר הנורות והפתרון שיתקבל אינו אופטימלי.

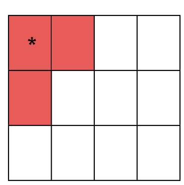
איור 5.4: פתרון ללוח $9 \times 2$



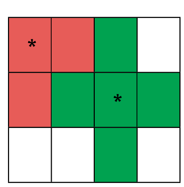
כדי להוכיח טענת הכותרת של תת הפרק הזה, נעזר בלוח שרירותי כלשהו.

**משפט 5.2:** עבור לוח בעל מימד כלשהו גדול מ 3, אין פתרון אופטימלי בו המשבצת בראשית הצירים נלחצה.

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



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

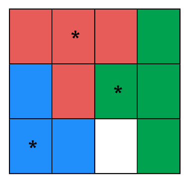


נשים לב שהגענו למבוי סתום משום שניתן להדליק את $a_{3,2}$ רק על ידי לחיצה על $a_{4,2}$, לאחר מכן לא היה ניתן להדליק את $a_{3,1}$. ניתוח דומה אפשר לתאר עבור לחיצה על $a_{3,2}$. $\square$

לפי טענה 5.2 אם ברצוננו למצוא פתרון אופטימלי ללוח האינסופי לא נוכל להדליק את נורה $a_{1,1}$ על ידי לחיצה עליו. לכן על מנת להדליק את $a_{1,1}$ נהיה חייבים ללחוץ על $a_{1,2}$ או $a_{2,1}$. בשלב זה נראה שגם לחיצה על $a_{1,2}$ לא תוביל למציאת פתרון אופטימלי. היות והלוח סימטרי גם לחיצה על $a_{2,1}$ לא תוביל למציאת פתרון אופטימלי.

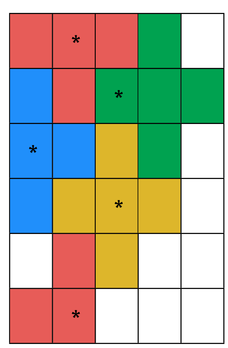
אם לחצנו על משבצת $a_{1,2}$ אז על מנת להדליק את $a_{2,1}$ בלי ללחוץ עליו חייבים ללחוץ על $a_{3,1}$ . אז על מנת להדליק את $a_{2,3}$ בלי ללחוץ עליו נלחץ על $a_{2,4}$.

מתקבל הלוח הבאה :



זה כבר מוכיח שלכל לוח $3 \times n$ כאשר $n > 2$ אין פתרון.

אחרי שלחצנו על $a_{2,4}$ על מנת להדליק $a_{3,3}$ בלי ללחוץ עליה חייבים ללחוץ על $a_{4,3}$ . אם מדובר על לוח $4 \times 4$ זה נותן פתרון אופטימלי עבורו. אם לא, אז על מנת להדליק $a_5, 2$ בלי ללחוץ עליה חייבים ללחוץ על $a_{6,2}$ וזה מביא למובי סתום, כפי שניתן לראות בלוח הבאה :



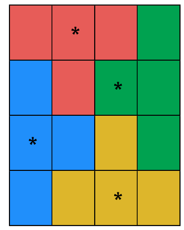
קח הוכחנו שלכל לוח $m \times n$ כאשר $\min\{m, n\} > 4$ אין פתרון אופטימלי.

## 5.2 אלגוריתם למציאת פתרון אופטימלי

נציעה דרך לחפש פתרון אופטימלי בעזרת שימוש במטריצה שכנויות כפי שהגדרנו בהגדרה 3.3, ההבדל הוא שהפעם נגדיר את המטריצה על החוג $\mathbb{Z}$. שימוש בחוג $\mathbb{Z}$ מאלץ את הפתרונות המתקבלים להדליק את כל נורות רק פעם אחת. התיאוריה שפיתחנו באלגברה לינארית הייתה תקפה לשדות אבל כלי התכנות שהשתמשנו בעבודה זו יודע לפתור גם על חוג השלמים.

אלגוריתם זה ממוש ומופיע בפרק נספחים בדקנו שאכן אלגוריתם מוצא פתרונות אופטימלי.

איור 5.5 : פתרון ללוח $4 \times 4$




# 6 נספחים

מימוש של הפרויקט בוצע על ידי שפת Python בעזרת הספריות Sage ו numpy.

## 6.1 יצירת מטריצת שכנויות

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

[8]:
```
import numpy as np
def genenerate_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
```

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

    return mat
def genenerate_neighbord_matrix(n) -> np.array:
    return genenerate_neighbord_matrix_m_n(n,n)

print('Adj matrix for 3,2 board:')
print(genenerate_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 אלגוריתם שמבוסס על מטריצת השכנויות

קוד זה מוצא פתרון לפי אלגוריתם שמבוסס על ממטריצת שכנויות. הפתרון המתקבל הוא וקטור שורה שאינדקסים של וקטור הם אינדקסים של משבצות לפי שיטת המספור שהצגנו בפרויקט. ניקח לדוגמה את הפתרון עבור לוח $3 \times 3$ שהתקבל בפלט:

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

את אותו פתרון נתאר בעזרת מטריצה כך:

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

[7]:
```
from sage.all import *
n = 3
A = Matrix(Integers(2),genenerate_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 של שורות

קוד זה מוצא פתרון לפי אלגוריתם שמבוסס על מילוי עקבי של שורות. הקוד מחולק לשלושה פונקציות:

gaussian_elimination_spanish_alg:

הפונקציה מקבלת כקלט: מטריצת שכנויות של המשחק ווקטור לוח במצבו הסופי כלומר, לוח בו כל הנורות דולקות. באלגוריתם הנ״ל נעשה שימוש בפעולות אלמנטריות על שורות אבל סדר פעולות שונה מסדר שלהן מסדר שלהן בדירוג של גאוס. המטרה שלנו היא לבטא את כל נעלמי המשבצות על ידי נעלמים של משבצות של שורה הראשונה. התוצאה המתקבלת מהפונקציה היא מטריצת שכנויות עם משתנים מחולצים, מצב הלוח הסופי לאחר פעולות על שורות.

mul_mat_sol_based_on_res:

הפונקציה מקבלת כקלט: מטריצת שכנויות עם משתנים מחולצים, מצב הלוח הסופי לאחר פעולות על שורות ופתרון חלקי עבור משבצות בשורה הראשונה בלוח. פונקציה זו לוקחת את הפתרונות של המשבצות בשורה הראשונה של הלוח ומוצאת לשאר המשבצת בלוח את מצבן בפתרון, לחוץ או לא. התוצאה המתקבלת היא שהפתרון החלקי שקיבלנו בקלט התמלאה והפתרון מתייחס לכל המשבצות על הלוח.

generate_mat_spanish_alg:

הפונקציה מקבלת כקלט מטריצת שכנויות. פונקציה זו משתמש בשי הפונקציות הקודמות כדי להחזיר את הפתרון של המשחק בעזרת שיטת מילוי עקבי. הפתרון מוצג כוקטור שורה כפי שתיארנו ב 6.2.

[3]:
```
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 = genenerate_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 soultion 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 soultion vector:
(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)
```

### 6.4 השווה בין שתי האלגוריתמים

קוד זה דוגם זמני ריצה של שני האלגוריתמים ומחזיר כפלט טבלה זמני ריצה.

[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 = genenerate_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

קוד זה מוצא פתרון אופטימלי. מציאת הפתרון מבוססת על גישה של מציאת פתרון במערכת משוואות על שלמים.

[5]:
```
from sage.all import *
n = 3
m = 2
a = genenerate_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 הפתרונות על לוח

קוד זה מחשב מספר הפתרונות שיש על לוחות $n \times m$ כאשר $m,n \leq 9$. הפלט שמתקבל הוא טבלה, שהשורות והעמודות מתארות את מימדי הלוח. לדוגמה אפשר לראות מטבלת התוצאות שללוח $3 \times 5$ כמות הפתרונות הוא 8 כפי שמתואר בשורה 3 ובעמודה 5.

[6]:
```
def num_solution_board(m,n):
    a = genenerate_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 מחשב את כל הפתרונות עבור לוח בגודל $m \times n$. הפתרון שמתקבל הוא רשימה של פתרונות כאשר כל פתרון הוא וקטור שורה כפי שתיארנו ב 6.2. מציאת כל הפתרונות מסתמכת על הגישה של חיבור פתרון פרטי עם כל הוקטורים במרחב האפס.

[81]:
```
def get_all_sol(m,n):
    """
    helper function to recursivly 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 soultion 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 soultion 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.