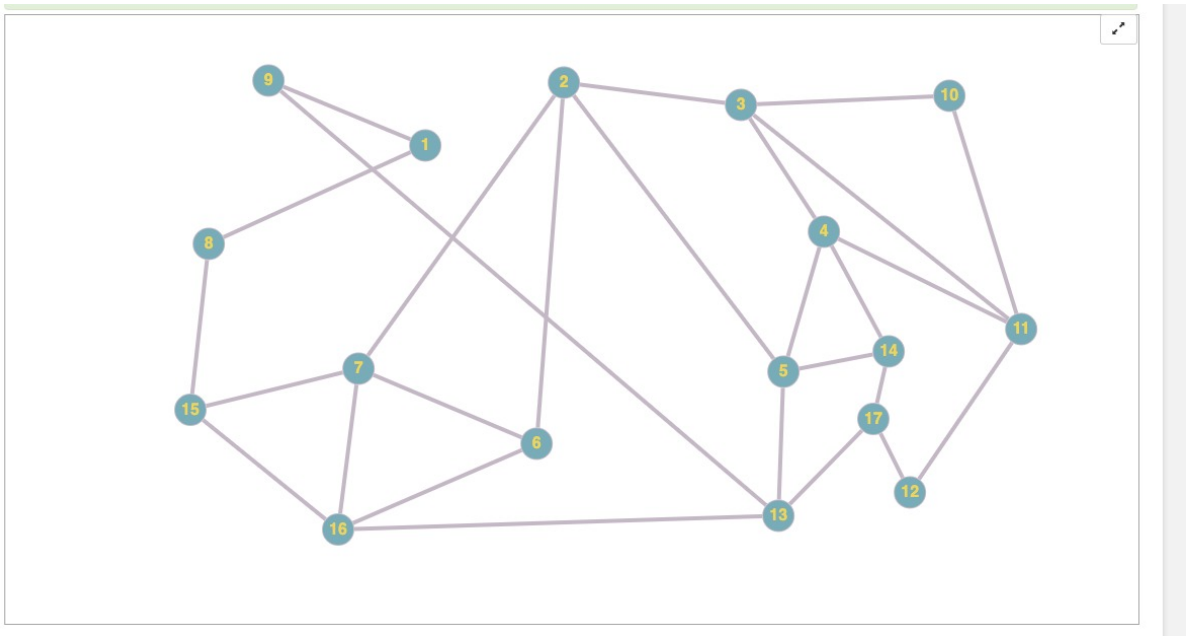
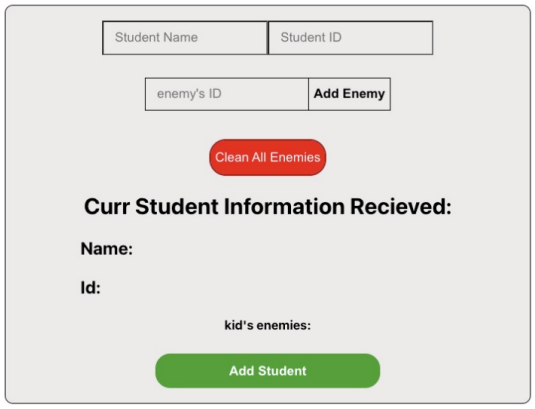
מיני פרויקט - נושאים ביישומיים במדעי המחשב

חלוקת שחקנים לקבוצות באמצעות פתרון בעיית צביעת גרף ב-k צבעים שגיא בלכר ושי שברצטוך



Distribution of Football players into groups with dependencies



Groups number: Not Defined yet

Name	ID	Enemies
s	0	1 2
s	1	8 9
s	2	3 5 6 7
s	3	4 10 11
s	4	5 11 14

מבוא ותיאור הבעיה:

כמעט בכל מסגרת שבה רוצים לחלק אנשים לקבוצות בצורה הוגנת, ישנם קשיים.
בסופו של דבר האדם האחראי נאלץ לעשות עבודה ידנית על מנת לחלק בצורה סבירה.
למשל, חלוקה לקבוצות במשחק כדורגל, או חלוקה לכיתות של שחקנים.
במסגרת פרויקט זה רצינו להשתמש בבעיית **צביעת גרף,** על מנת לבצע חלוקה בצורה הוגנת.
סיפור הרקע אותו בחרנו, הוא חלוקה של שחקנים לקבוצות בעת משחק כדורגל. כל שבוע, כשאנחנו משחקים עולה הסוגיה איך לחלק את הקבוצות.
אנחנו מציעים פתרון יעיל ונחמד לחלוקת השחקנים, בו באמצעות מעט מחשבה נוכל למצוא גם פתרון הוגן מאוד לחלוקה.
במהלך החודשים האחרונים, תחילה מימשנו פתרון לבעיית צביעת גרף באמצעות הספריה ec-kity.
לאחר מכן, הוספנו ממשק UI בו ניתן לחלק שחקנים לקבוצות, זאת על מנת להראות שימוש פרקטי לבעיה שפתרנו.
קישורים לגיט - מימוש הפתרון האלגוריתם וBE, מימוש הUI

תיאור פורמלי של הבעיה:

- בהינתן גרף המורכב מקודקודים וצלעות, נרצה לצבוע את בגרף בעד k צבעים (k הוא פרמטר וניתן לשינוי) - כך שלא ייתכן שקודקודים שכנים (חולקים צלע) יקבלו את אותו הצבע.
- במסגרת הפרויקט שלנו - כל שחקן מייצג קודקוד אחד ויחיד
- "אויבות" בין שני שחקנים, אשר לא רוצים לשחק יחד בקבוצה, תיוצג באמצעות חיבור ביניהם בגרף (כלומר צלע בין שני הקודקודים המייצגים אותם)
- מספר הצבעים איתם נרצה לצבוע את הגרף (k) - הם בעצם מספר הקבוצות אליהן נרצה לחלק את השחקנים
- תיתכן חלוקה "לא אפשרית" (תלוי בקשרי האויבות בין שחקנים). במקרה הזה ננסה לתת פתרון אופטימלי של כמה שפחות אויבים באותה קבוצה.
- בUI יהיה ניתן להוסיף שחקנים (קודקודים), ולהוסיף קשרי אויבות (צלעות). לאחר יצירת הגרף, יווצר גרף בעל קודקודים וצלעות אשר יהווה קלט לקוד אשר פותר את הבעיה באמצעות ec-kity.
- מידול הבעיה על מנת לספק חלוקה הוגנת:

- ראשית, נחלק את השחקנים ל"רמות איכות". כל קבוצה תכיל מספר שחקנים כמספר הקבוצות אליהן נרצה לחלק (למשל, אם נרצה לחלק ל4 קבוצות, ויהיו לנו 44 שחקנים, יהיו 11 קבוצות של 4 שחקנים).
- "קבוצת האיכות" הראשונה תכיל את 4 השחקנים הטובים ביותר, השנייה את הבאים אחריהם, וכן הלאה
- נחבר צלע בין כל הקודקודים הנמצאים באותה הקבוצה
- נחבר צלע בין שחקנים אויבים אשר לא אוהבים להיות יחד בקבוצה
- נחבר צלע בין שחקנים אשר יושבים על אותה העמדה (למשל, צלעות בין כל השוערים)
- נריץ את הבעיה ונקבל חלוקה הוגנת בה בכל קבוצת צבע שתכיל 11 שחקנים בהכרח, יהיו שחקן אחד מכל "קבוצת איכות" ומינימום של שחקנים אויבים יחד.

פרטי המימוש ותיאור הפתרון - צביעת הגרף

- קלט הבעיה - מספר הקודקודים, מספר הצבעים, ומערך של צלעות בין הקודקודים.
- פלט הבעיה - מערך בגודל מספר הקודקודים בו כל תא מייצג שחקן (קודקוד) והערך בו הינו הצבע (כיתה) הניתן לשחקן
- הרצת האלגוריתם
  - כפי שתואר למעלה, השתמשנו בספרייה eckity כפי שהוסבר לנו.
  - בעת הריצה, הקובץ EvolutionaryColores.py מריץ את אבולוציית האלגוריתם לפי ההגדרות שניתנו לו באמצעות טיפוס מסוג simpleEvolution, ע"י שימוש בפונקציה evolve.
  - תחילה יתבצע איתחול של אינדיבידואלים (גרפים צבועים) בצורה רנדומית.
  - לפי ההגדרות שהזנו, בעת כל דור (מקסימום 100), ייבחרו נציגים ע"פ שיטת tournament tree הממומשת בספרייה. הפרמטר `higher_is_better=False` מתאר את זה שבחירת הנציגים ה"טובים יותר" הם אלו בעלי הניקוד הנמוך יותר (גרפים עם פחות טעויות)
  - נציגים אלו יבנו את הדור הבא באמצעות פונקציות crossover ו mutate אשר אותן מימשנו אנו.
  - התהליך ייפסק בעת הגעה ל100 דורות, או כאשר יווצר אינדיבידואל (גרף) עם 0 טעויות.

- אינדיבידואל במימוש שלנו הינו coloredGraph אשר מכיל את הגרף עצמו, והצעה של צביעה.
- האוכלוסיה הינה רשימה של coloredGraphs אשר מיוצרים בצורה רנדומית בתחילת התהליך על ידי המחלקה `ColoredGraphCreator`
- האבולוציה תפעל באופן הבא:
  - בתחילת כל דור תתבצע הערכה של הפרטים באוכלוסיה (fitness) על ידי המחלקה `ColoredGraphEvaluator`.
  היא תפעל בצורה שבה נעבור על הגרף ועל הצבעים שנקבעו, ולכל "טעות" כלומר, צלע בה שני קודקודים קיבלו את אותו הצבע, נוסיף 1 למספר הטעויות (fitness) של האינדיבידואל.
  - בחירה (selection) של הדור הבא תתבצע על ידי tournament selection
  - המוטציה (mutation) מתבצעת על ידי המחקלה `ColoredMutate` אשר מימשנו ומרחיבה את geneticOperator של eckity.
  - בצורה דומה, כך גם crossover ממומש על ידינו על מנת יצירת אינדיבידואלים לדור הבא.

ארכיטקטורת התוכנה

פירוט הקבצים והמחלקות:

- The ColoredGraph class is a representation of an individual in the population. It has a reference to a graph object, a num_of_colors attribute, and a colors attribute. This class has several methods that allow it to interact with the graph, such as get_graph(), get_num_of_colors(), get_colors(), set_all_vertices_to_legal(), and paint_graph_vertices(). Additionally, it also has an evaluate_fitness() method which calculates the fitness of this individual, in this case, it calls the graph's evaluate_fitness() method.
- The ColoredCrossover class is a genetic operator that is applied to a pair of ColoredGraph individuals. It uses a crossover() method to combine the colors arrays of the two individuals and update their ColoredGraph accordingly.
- The ColoredGraphCreator class is a creator class that creates the initial population of ColoredGraph individuals. It creates n_individuals number of random ColoredGraph instances by creating a random colors array, loading it into a Graph object, and then creating a ColoredGraph with these attributes.
- The ColoredGraphEvaluator class is a SimpleIndividualEvaluator that evaluates the fitness of a ColoredGraph individual. It starts by marking all vertices as legal and then check if there is any collision between the color of the vertice and its neighbours.

- The **EvolutionaryColores** file creates the run of the algorithm, as described above.
- **Graph:**
  - Has 3 fields - num_of_vertices, fitness, and adjacency list
  - it has init function that builds the graph from arrays of vertices and edges
  - getters and setters to all fields
  - evaluate_fittness (and helpers for the function) - Goes through the adjacency list and finds all neighbors that are in the same color

הצגת הבעיה ו- UI

לאחר שהצלחנו להגיע לתוצאות מספקות באמצעות הקוד המתואר לעיל, התאמנו את הקוד שלנו לסיפור הרקע של חלוקת שחקנים לקבוצות.

השתמשנו בreact-js על מנת לממש את הUI. הוספנו אופציה להוסיף שחקנים (קודקודים), ולהוסיף"אויבות" (צלע). על המשתמש לצרף את הצלעות בהתאם לרצונות השחקנים.

רצינו להראות כמה שימושי יכול להיות פתרון הבעיה המתוארת. הUI מאפשר לנו לחלק שחקנים לקבוצות ומאחורי הקלעים בעצם מייצר גרף ומשתמש באלגוריתם שנוצר.

לאחר שמילאנו כל מה שרצינו, ולכמה קבוצות נרצה לחלק את השחקנים, על המשתמש ללחוץ על "get Results" וכך בעצם מתחיל החישוב:

- מאחורי הקלעים, מקבל הBE שלנו json בצורה הבאה:


```
[
{,
}
"kidName":"Ori",
"kidID":"5",
"kidEnemies":[

[
{,
}
"kidName":"Eitan",
"kidID":"6",
"kidEnemies":[
"7",
"8"
[
```

},.... (יש המשך לjson)

- ובהתאם לכך,מתבצעת פונקציית parse אשר מייצרת קודקוד לכל ילד וצלע לכל אויבות.
- התוצאה נראית ככה:

[(0, 1), (0, 2), (0, 3), (1, 0), (1, 2), (2, 0), (2, 1), (2, 8), (2, 9), (2, 15), (3, 4), (3, 5), (3, 11), (4, 3), (4, 5), (5, 3), (5, 4), (5, 10), (6, 7), (6, 8), (7, 6), (7, 8), (8, 6), (8, 7), (8, 16), (9, 10), (9, 11), (10, 9), (10, 11), (10, 5), (11, 9), (11, 10), (11, 5), (12, 13), (12, 14), (12, 4), (12, 16), (13, 12), (13, 14), (14, 12), (14, 13), (14, 17), (15, 16), (15, 17), (16, 15), (16, 17), (17, 15), (17, 16)]

- את הקלט הזה, האלגוריתם כפי שתואר למעלה יודע "לאכול" ומשם הלאה המימוש הוא בדיוק כמו שתיארנו.

תוצאות האלגוריתם

ניסינו למצוא את הפרמטרים הטובים ביותר לצורך פתרון אופטימאלי.
קשה מאוד היה לייצר את הגרפים עליהם נבחן ולוודא את הפתרון שלנו בצורה נכונה.
ייצרנו כ-50 גרפים להם אנו יודעים את הפתרון, ובדקנו עבור מספר פרמטרים מה הפתרון האידיאלי.

- הסתברות ה mutation - בדקנו את 0.1, 0.2, 0.3, 0.4
  עבור הגרפים הגדולים, ראינו שהפרמטר האידיאלי הוא 0.2

בדיקה לפי הmutate rate: כמה פעמים מגיעים לפיטנס טוב ביותר, אחרי כמה דורות, וממוצעים מתוך 20 הרצות לכל גודל. מסקנה: 0.2 הוא הmutate rate המשתלם לנו ביותר

0.1	
mutate rate	
מספר הדורות	רמת הפיטנס
9	0
100	2
100	2
100	2
100	4
100	2
100	4
100	2
100	4
100	4
100	4
100	4
9	0
11	0
100	4
פיטנס ממוצע:	2.53
מספר דורות ממוצע:	81.93
פעמים שהשיג פיטנס טוב ביותו	2

0.15	
mutate rate	
מספר הדורות	רמת הפיטנס
100	4
8	0
100	2
100	2
8	0
100	2
100	2
100	2
100	2
100	2
100	2
41	0
100	2
100	2
100	2
פיטנס ממוצע:	1.73
מספר דורות ממוצע:	83.8
פעמים שהשיג פיטנס טוב ביותר:	3

0.2	
mutate rate	
מספר הדורות	רמת הפיטנס
4	0
100	2
100	2
27	0
16	0
100	2
100	2
33	0
100	2
43	0
100	2
6	0
100	2
100	2
8	0
פיטנס ממוצע:	1.06
מספר דורות ממוצע:	62.46
פעמים שהשיג פיטנס טוב ביותר:	7

0.25	
mutate rate	
מספר הדורות	רמת הפיטנס
100	2
100	2
100	2
100	4
100	2
100	3
100	4
21	0
100	4
54	0
71	0
100	4
100	2
14	0
99	0
פיטנס ממוצע:	1.93
מספר דורות ממוצע:	83.93
פעמים שהשיג פיטנס טוב ביותר	5

- הסתברות ה crossover - בדקנו את 0.6, 0.7, 0.75, 0.8, 0.85, 0.9
  עבור הגרפים הגדולים, ראינו שהפרמטר האידיאלי הוא 0.85

בדיקה לפי crossover rate: כמה פעמים מגיעים לפיטנס טוב ביותר, אחרי כמה דורות, וממוצעים מתוך 20 הרצות לכל גודל. מסקנה: crossover rate ממלץ לצרכן: 0.85

0.7	
crossover rate	
מספר הדורות	רמת הפיטנס
100	4
100	2
100	2
100	2
100	2
100	4
100	3
7	0
100	3
100	2
100	2
100	4
100	2
100	4
100	2
פיטנס ממוצע:	2.53
מספר דורות ממוצע:	93.8
פעמים שהשיג פיטנס טוב ביותו	1

0.75	
crossover rate	
מספר הדורות	רמת הפיטנס
39	0
100	2
16	0
100	2
100	2
17	0
100	2
25	0
52	0
63	0
100	2
9	0
20	0
16	0
100	4
פיטנס ממוצע:	0.93
מספר דורות ממוצע:	57.13
פעמים שהשיג פיטנס טוב ביותר:	9

0.8	
crossover rate	
מספר הדורות	רמת הפיטנס
100	2
100	4
100	2
49	0
100	2
100	2
9	0
100	4
100	4
100	2
35	0
27	0
100	2
23	0
100	2
פיטנס ממוצע:	1.73
מספר דורות ממוצע:	76.2
פעמים שהשיג פיטנס טוב ביותר	5

0.85	
crossover rate	
מספר הדורות	רמת הפיטנס
28	0
100	2
100	2
86	0
100	2
5	0
9	0
18	0
24	0
100	2
100	2
100	2
100	2
14	0
73	0
פיטנס ממוצע:	0.93
מספר דורות ממוצע:	63.8
פעמים שהשיג פיטנס טוב ביותר	8

0.9	
crossover rate	
מספר הדורות	רמת הפיטנס
100	2
15	0
16	0
100	2
100	2
100	2
100	2
100	2
100	2
100	4
7	0
100	2
100	2
100	2
100	2
פיטנס ממוצע:	1.73
מספר דורות ממוצע:	82.53
פעמים שהשיג פיטנס טוב ביותר:	3

- גילינו, שלצורכי הגרפים שלנו (גרפים שרצינו להיות מסוגלים לבדוק ומספר הקודקודים בהם לא עובר את 20), מספר הדורות 150 הוא די ויותר. ייתכן שעבור גרפים גדולים יידרש מספר גדול יותר של דורות

בדיקה לפי כמות הפרטים באוכלוסייה: כמה פעמים מגיעים לפיטנס טוב ביותר, אחרי כמה דורות, וממוצעים מתוך 20 הרצות לכל גודל. מסקנה: עבור 150 נקבל את הריצה המשתלמת ביותר

גודל האוכלוסייה	50
מספר הדורות	רמת הפיטנס
100	2
100	4
100	4
100	2
17	0
100	2
100	4
100	4
100	2
100	2
100	2
100	2
100	2
8	0
100	3
פיטנס ממוצע:	2.33
מספר דורות ממוצע:	88.33
פעמים שהשיג פיטנס טוב ביותר:	2

גודל האוכלוסייה	100
מספר הדורות	רמת הפיטנס
100	2
100	4
82	4
49	2
100	2
100	4
100	2
100	4
100	2
100	2
100	2
100	2
100	5
8	2
100	2
פיטנס ממוצע:	2.73
מספר דורות ממוצע:	89.26
פעמים שהשיג פיטנס טוב ביותר:	0

גודל האוכלוסייה	150
מספר הדורות	רמת הפיטנס
100	2
100	4
100	2
16	0
9	0
100	2
64	0
100	2
15	0
100	0
16	0
23	0
100	2
11	0
100	2
פיטנס ממוצע:	1.06
מספר דורות ממוצע:	63.6
פעמים שהשיג פיטנס טוב ביותר:	8

גודל האוכלוסייה	200
מספר הדורות	רמת הפיטנס
100	2
96	0
14	0
100	2
100	2
9	0
100	2
80	0
12	0
100	2
82	0
100	2
100	2
100	4
35	0
פיטנס ממוצע:	1.2
מספר דורות ממוצע:	75.2
פעמים שהשיג פיטנס טוב ביותר:	6

ביצענו ניתוחים מדויקים לצורך הערכת הפרמטרים, והטבלאות וההסברים מפורטים בסוף.

דוגמאות הרצה בUI:

רצינו להראות חלוקה הוגנת לקבוצות כפי שתיארנו למעלה:

ראשית, יצרנו 18 שחקנים אותם נרצה לחלק ל3 קבוצות (6 שחקנים בקבוצה)

"דירגנו" את השחקנים לפי איכות (השחקן עם id 0 הוא הטוב ביותר והשחקן עם id 17 הוא החלש ביותר).

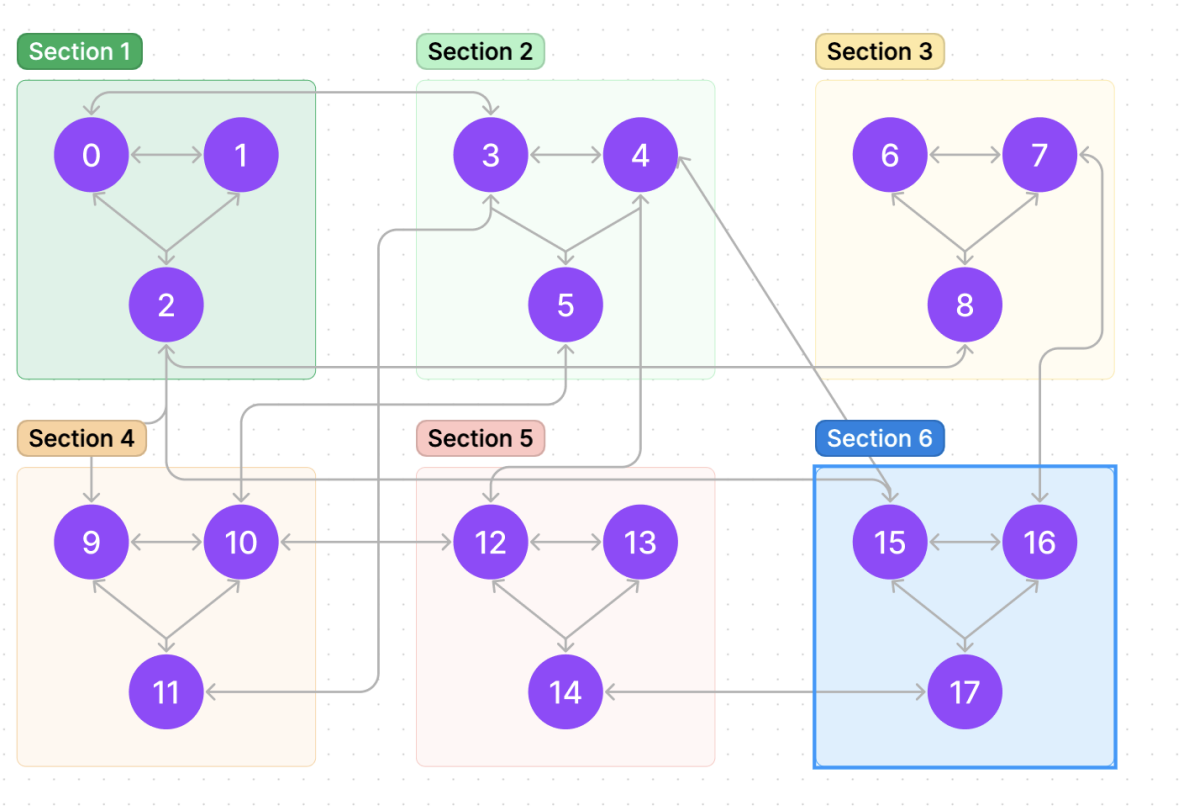
Groups number: 3

Name	ID	Enemies
David	0	1 2 3
Ariel	1	0 2
Lavi	2	0 1 8 9 15
Noam	3	4 5 11
Tamar	4	3 5
Shay	10	9 11 5
Sagy	11	9 10 5
Noa	12	13 14 4 16
Daniel	13	12 14
Itamar	14	12 13 17
Yair	15	16 17

בהתאם לאיכות, יצרנו צלעות בין כל "שלשת שחקנים" לפי הדירוג (משולשים, צלעות בין 0,1,2 צלעות בין 3,4,5 וכן הלאה)

הוספנו צלעות נוספות בין שחקנים שלא אוהבים לשחק יחד, וציינו שאנו רוצים לחלק ל3 קבוצות:

בעצם יצרנו גרף שנראה כך:



הפלט מהUI נראה כך:

["3",{"kidName":"David","kidID":"0","kidEnemies":["1","2","3"]},{"kidName":"Ariel","kidID":"1","kidEnemies":["0","2"]},{"kidName":"Lavi","kidID":"2","kidEnemies":["0","1","8","9","15"]},{"kidName":"Noam","kidID":"3","kidEnemies":["4","5","11"]},{"kidName":"Tamar","kidID":"4","kidEnemies":["3","5"]},{"kidName":"Ori","kidID":"5","kidEnemies":["3","4","10"]},{"kidName":"Eitan","kidID":"6","kidEnemies":["7","8"]},{"kidName":"Moshe","kidID":"7","kidEnemies":["6","8"]},{"kidName":"Maya","kidID":"8","kidEnemies":["6","7","16"]},{"kidName":"Libbi","kidID":"9","kidEnemies":["10","11"]},{"kidName":"Shay","kidID":"10","kidEnemies":["9","11","5"]},{"kidName":"Sagy","kidID":"11","kidEnemies":["9","10","5"]},{"kidName":"Noa","kidID":"12","kidEnemies":["13","14","4","16"]},{"kidName":"Daniel

","kidID":"13","kidEnemies":["12","14"]},{"kidName":"Itamar","kidID":"14","kidEnemies":["12","13","17"]},{"kidName":"Yair","kidID":"15","kidEnemies":["16","17"]},{"kidName":"Shimon","kidID":"16","kidEnemies":["15","17"]},{"kidName":"Imri","kidID":"17","kidEnemies":["15","16"]}]

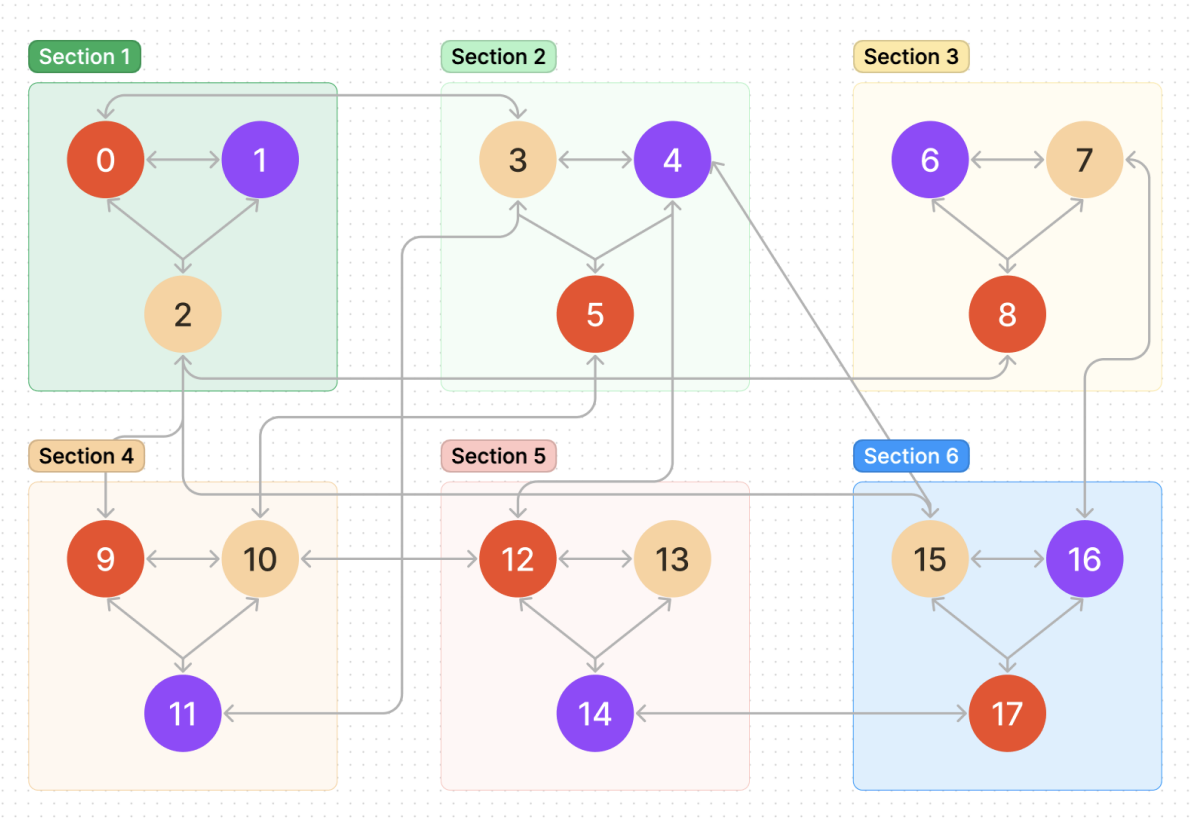
ולאחר קבלת הBE ותהליך הפארסינג, נקבל מערך צלעות שנראה כך:

[(0, 1), (0, 2), (0, 3), (1, 0), (1, 2), (2, 0), (2, 1), (2, 8), (2, 9), (2, 15), (3, 4), (3, 5), (3, 11), (4, 3), (4, 5), (5, 3), (5, 4), (5, 10), (6, 7), (6, 8), (7, 6), (7, 8), (8, 6), (8, 7), (8, 16), (9, 10), (9, 11), (10, 9), (10, 11), (11, 9), (11, 10), (9, 11), (5, 10), (10, 11), (5, 11), (12, 13), (12, 14), (12, 4), (13, 12), (13, 14), (14, 12), (14, 13), (14, 17), (15, 16), (15, 17), (16, 15), (16, 17), (17, 16), (17, 15), (17, 16)]

ה BE מיד מקבל את הקלט, מריץ את האלגוריתם ומספק מערך של צביעה לגרף:

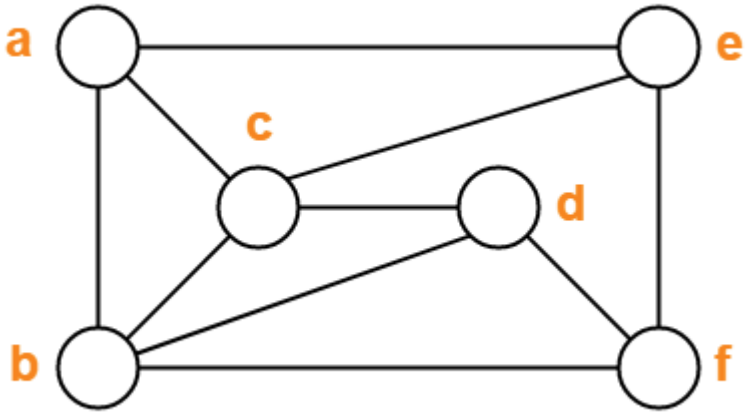
colors: [0, 2, 1, 2, 1, 0, 1, 2, 0, 0, 2, 1, 0, 2, 1, 2, 1, 0]

וכשנצבע את הגרף ניתן לראות שאכן התשובה מושלמת, חילקנו את השחקנים לקבוצות זהות בגודלן, בלי אויבויות, ובצורה הוגנת!



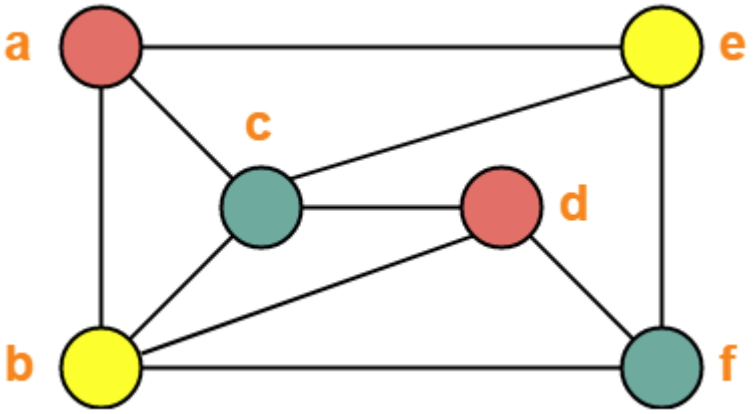
דוגמאות נוספות:

גרף הקלט: num of colors=3

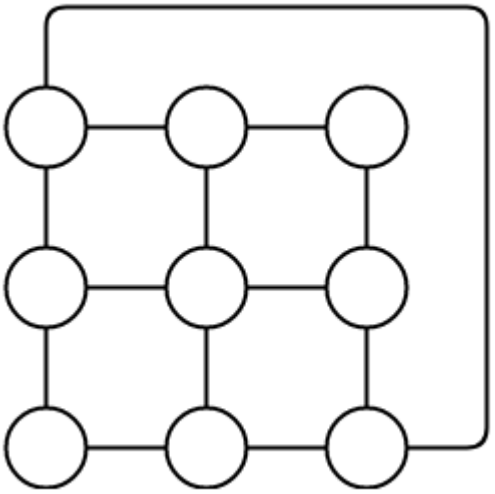


פלט:

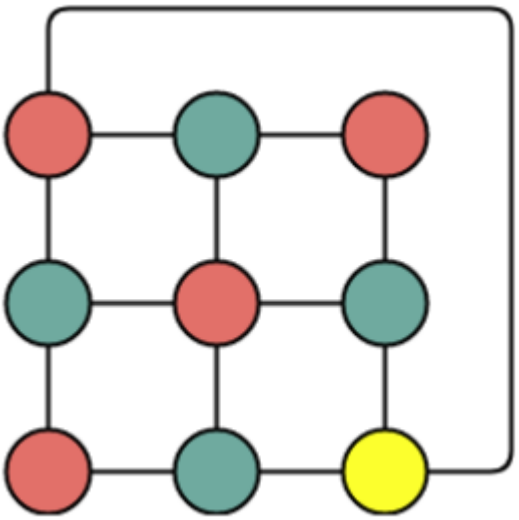
Color	C1	C2	C3	C1	C2	C3



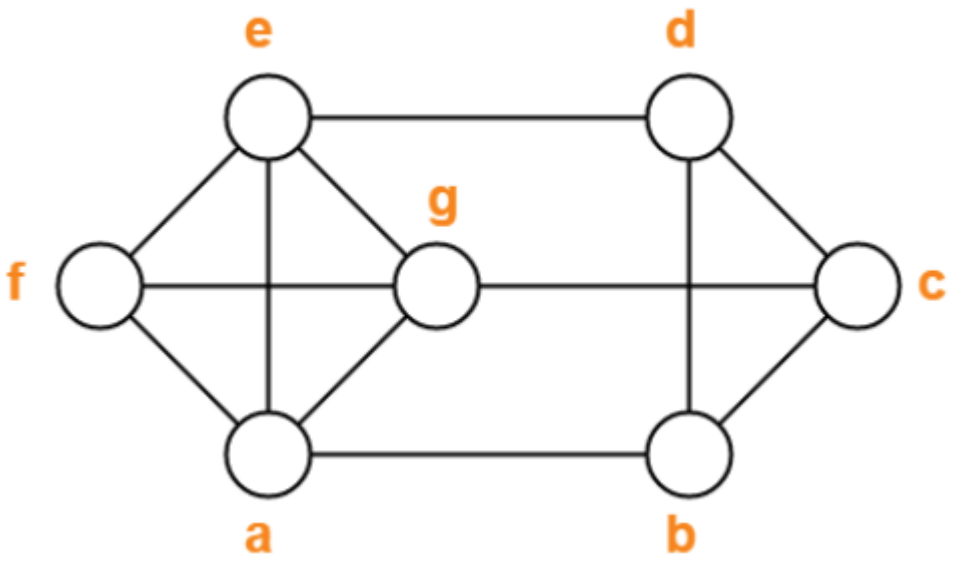
גרף הקלט: num of colors=3:



פלט:



גרף הקלט: num of colors=4:



פלט: