**מעבדה מספר 3**

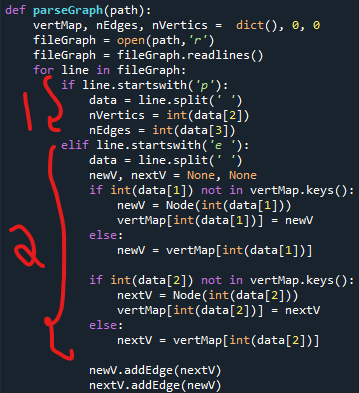
**אחמד מסאלחה 207923483**

**תחרות בצביעת גרפים CSP לאופטימיזציה דיסקרטית**

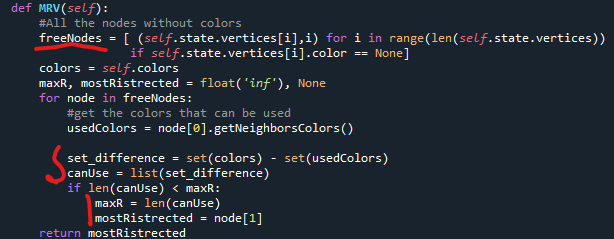
בכל אלגוריתם מתחילים עם ריצה חמדנית כדי להתחיל עם מספר צבעים כלשהו

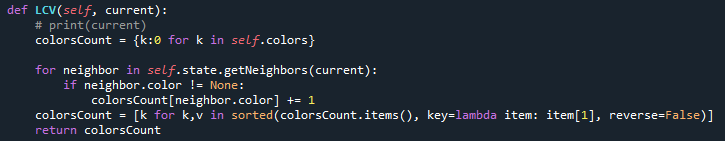
מצורף קובץ greedy.py

מצורף קובת EXE בשם main להרצה, אפשר גם עיי main.py (להריץ עיי python main.py)

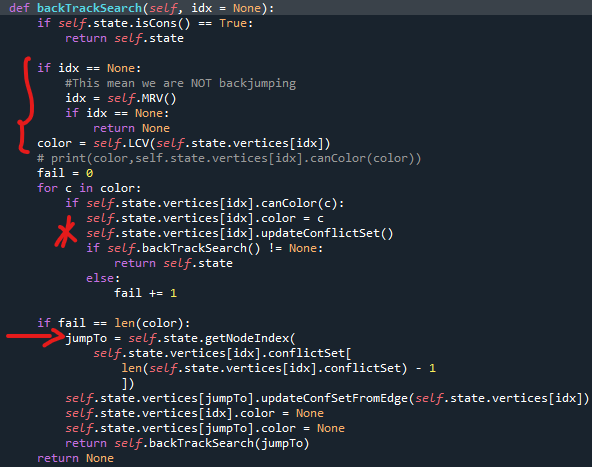
1. הקלט
   1. קריאת הקלט נמצאת בקובץ functions.py בה אנחנו עוברים על כל שורה, אם מתחילי ב p אז נקרא את מספר הצמתים ומספר הקשתות, אם מתחילים ב e נקרא איזה צתמתים יש וניצור קשת בינהם
      1. 
   2. חשבים צפיפות וגם מדפיסים פרטים
      1. 
2. **מימוש היוריסטיקות ו אלגוריתמים של backtracking + back jumping וגם forward checking + arc consistency**

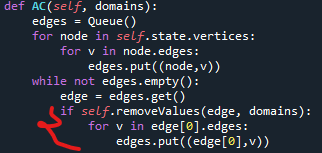
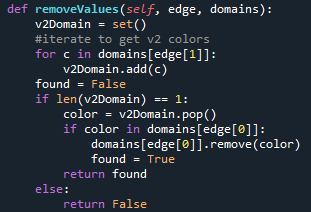
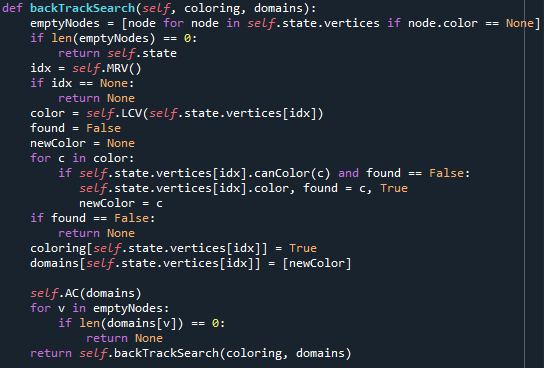
I used only LCV and MRV without HD

* 1. MRV
     1. בעצם הסתכלנו על הצמתים שעדיין לא עשינו שימוש בהם ועברנו אחד אחד תוך עדי שבדקנו איזה צומת יש עליו יותר אילוצים כלומר פחות צבעים ואז לקחנו אותו
        1. 
  2. LCV
     1. מיינו את הצבעים לפי הערכים שלהם, כלומר כמה שימוש עשינו בכל אחד, ולכן הם ממוינים מהפחות מאולץ עד למאולץ



* 1. BACKTRACKING עם BACKJUMPING
     1. כמו באלגוריתם חיפוש לאחור, בכל פעם אנחנו בוחרים צומת וצבע על ידי היוריסטיקות שבחרנו ומנסים לצבוע, ועושים backtracking, רק שההבדל כשמגיעים לצומת ולא מצליחים, אז נלך אחורה בקפיצה לצומת האחרון שנמצא conflict set שלו, ואפ הצלחנו, אז מעדכנים את כל ה conflict set של השכנים



* + 1. בחלק הראשון זה בעצם אם עשינו קפיצה, ואם לא נבחר צומת עיי MRV
    2. ה \* זה בעצם עדכון ה conflict set של השכנים אם מצליחים לבוע:
       1. 
    3. והחלק האחרון בו הולכים ל jumpTo שזה בעצם הצומת שנקפוץ אליו (הצומת האחרון שנמצא אצל הצומת הנוכחי ב conflictyset)
  1. ARC CONSISTENCY עם FORWARD CHECKING
     1. לגוריתם שמשתמש ב backtracking אך בעזרת forward checking ולכן כל פעם בעזר AC נוודא שהגרף קונסיסטנטי עיי שמוחקים מכל הדומיינים של השכנים את הצבע שבו צבענו את הצומת הנוכחית, ונבודק האם לכל צבע שנמצא בדומיין של צומת מסויימת קיים תמיד לפחות צבע אחד חוקי (שונה מזה שבחרנו) בדומיין של כל אחד מהשכנים.
     2. 
     3. 
     4. עצם אלה שתי הפונקציות של AC
     5. נעשה שימוש בדיוק ב backtracking עיי MRV ו LCV
     6. 
     7. בכל פעם ננסה לצבוע, אם יש domain ריק אז נכשלנו, אחרת ניצבע ונעדקן את הדומייים עיי AC ונעשה backtracking

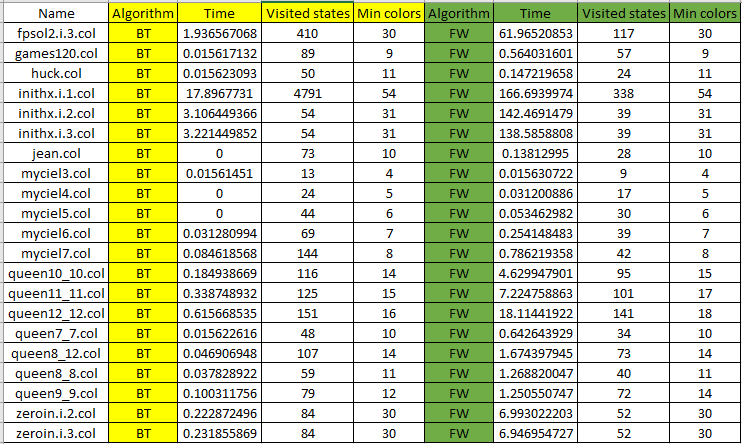
**השוואה בין שני האלגוריתמים:**

נשווה כעת בין שני האלגוריתמים מבחינת: זמן ריצה, מס צמתים שבוקרו, ועלות הפתרון, כלומר הצביעה המינימלית שנקבל backtracking

**לגבי זמן ריצה,** מצופה שה backtracking עם backjumping יהיה מהיר יותר, וזאת בגלל ה AC שעושים שלוקח זמן גבוה, שזה גם תלוי מימוש

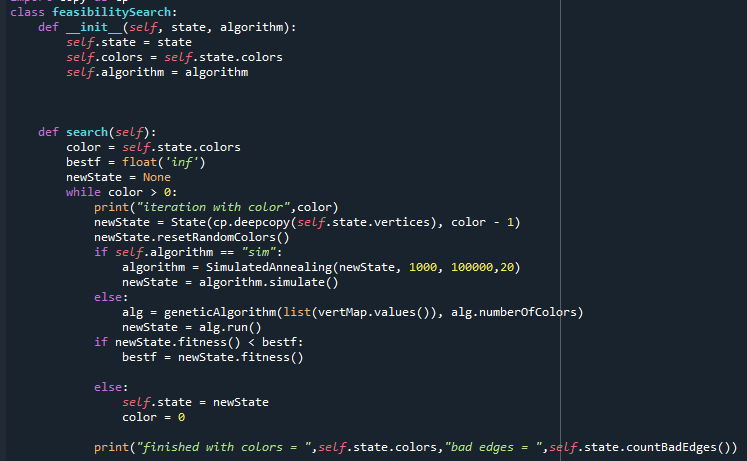
**לגבי צמתים שבוקרו,** מצופה מ FORWARD CHECKING לבקר בפחות צמתים, כי הוא מסתכל קדימה ואם איפשה בהמשך דומיין יגמר אז ניתקע החיפוש לפני ולא צריך לחפש עד זה, ולכן ב backtracking מספר הצמתים שבוקרו יהיה גדול יותר

**תוצאה:**

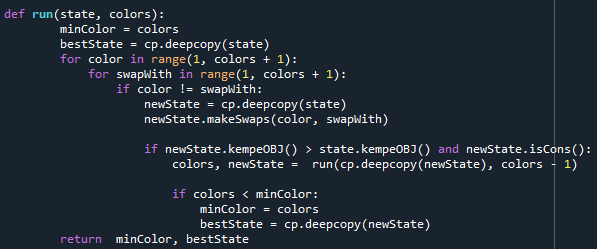


אכן מה שהסברתי לגבי מספר הצמתים וזמן ריצה זה נכון, **לגבי איכות הפתרון**, אז כמעט אותה צבעיה, ובמקרים קטנים ה backtracking הוא טוב יותר וזה טבעי כי הוא מחפש יותר

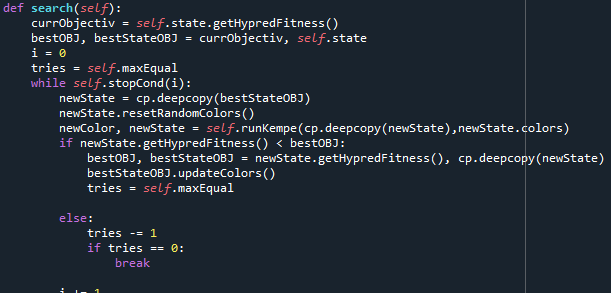
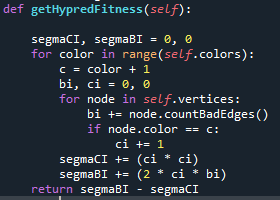
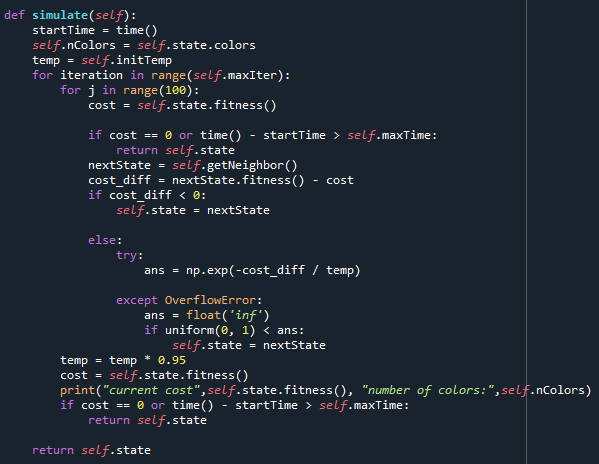
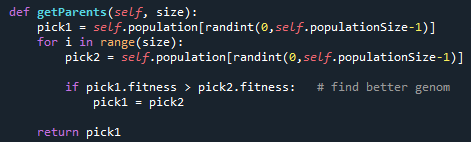
1. **החלק השני: שלושת השיטות**
   1. הגישה הפיזבילית:
      1. שיטה המנסה לצבוע את הגרף במספר הצבעים המינימלי כך שכל פעם נקטין את מספר הצבעים וננסה לצבוע עיי חיפוש לוקאלי

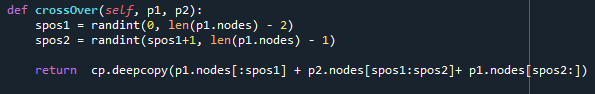


בפונקציית ה search כל פעם מנסים צבע חדש עד שלא מצליחים

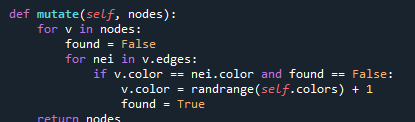
* 1. שרשראות KEMPE
     1. הגישה שמנסה למקסם את סכום ריבוע גדלי מחלקות הצבע, עיי שמעבירים צבעים ממחלקת צבע לאחרת
     2. 
     3. בעצם מנסים להעביר ממחלקה לאחרת, אם פונקציית סכום ריבוע גדלי מחלקות הצבע משתפרת אז מנסים אותה וכך הלאה
  2. הגישה ההיברידית
     1. בגישה הזאת אנחנו ניקח בחשבון את הפיזביליות וגם את סכום ריבוע גדלי מחלקות הצבע וננסה להביא את הביטוי: 

מינימום

* + 1. 
    2. בכל איטרציה ננסה להריץ את KEMPE ולנסות להביא למינימום את הביטוי עיי הפונקציה: getHypredFitness
    3. 
    4. כאשר segmaCI ו segmaBI הם שני הסיגמות בביטוי
  1. מימוש אלגוריתם SimulatedAnnealing
     1. אותו מימוש של מעבדה קודמת בכל פעם מביאים את השכן, ואם הוא משפר ניקח, אחרת ניקח בהסתברות בדיוק כמו במעבדה קודמת
     2. 
  2. מימוש אלגוריתם גיניטי (הביצועים יצאו מאוד לא טובים ולכן נעשה שימוש ב SA)
     1. אופירטור בחירת אבא ואמא עיי טורניר בין 25 רנדומלים
        1. 
     2. CrossOver
        1. Two point



* + 1. Mutation
       1. עושים בעצם מוטציה רק לצמתים שיש בהם conflict, כלומר צומת שצבע שלו זהה לאחד השכנים שלו



**השוואה בין שני האלגוריתמים:**

שלושת האלגוריתמים דיי קרובים מבחינת איכותת, בשלושתם מוץ בהתחלה אלגוריתם חמדני ללקיחת מספר התחלתי של צבעים וממנו נשפר, ולכן רואים שבערך שלושתם נותנים אותם תוצאות, כמעת!

מבחינת זמן ריצה ואיכות האלגוריתם הפיזבלי עדיף על השניים האחרים.

ההיפרידי מסתמך על תוצאות KEMPE בחלק מהמקרים (אופן הפונקציה של ה OBJECTIVE) ויתכן שהמימוש לא הכי יעיל וטוב.

**תוצאות: (דף הבא)**

