מבוא למדעי המחשב 67101

תרגיל 11 - Backtracking או גישוש נסוג
להגשה בתאריך 10/01/18 בשעה 22:00

Part A: One algorithm to rule them all

בחלק זה תדרשו לבנות פונקציה שתממש Backtracking כללי, על פי מה שנלמד בכיתה.
האלגוריתם שתכתבו יצטרך לפתור באופן רקורסיבי הן את משחק הסודוקו והן את בעיית צביעת מפות (ראו תיאור לגבי שתיהן למטה), ובנוסף בעיה נוספת שאינכן יודעות מראש מה היא.

כתבו פונקציה בשם
general_backtracking(list_of_items, dict_items_to_vals, index, set_of_assignments, legal_assignment_func, *args):

הפרמטרים של הפונקציה:
list_of_items: הוא רשימה של האיברים להם נרצה לתת השמה.
dict_items_to_vals - הוא מילון המכיל מפתחות מהרשימה **list_of_items** וערכים של השמה מתוך **set_of_assignments.** ,
Index - מספר שלם (int) המציין את המיקום בlist_of_item של האיבר אותו אנו מנסים לעדכן בשלב זה של הרקורסיה.
set_of_assignments- קבוצת ההשמות המותרת (sequence כלשהו המכיל את כל ההצבות חוקיות). כלומר רשימת הערכים הרלוונטיים שיהיו הvalues במילון **dict_items_to_vals** (למשל ערכים 1-9 למשחק הסודוקו),
legal_assignment_func- פוינטר לפונקציה יעודית הבודקת את תקינותה של הצבה אחת.
args- רשימה של משתנים נוספים שיכולים להיות מועברים לפונקציה.

המטרה של הפונקציה היא לפתור כל בעיית Backtracking באלגוריתם הנאיבי הבא: עבור כל איבר ברשימה **list_of_items,** הפונקציה תנסה את כל ההצבות, כלומר את כל האיברים ב**set_of_assignments**. עבור כל אחד מהם היא תבדוק האם ההצבה חוקית באמצעות **legal_assignment_func.** במידה וכן, הרקורסיה תתקדם לאיבר הבא ב**list_of_items** וגם עבורו תנסה את כל ההצבות החוקיות, רק שהפעם ההצבה תהיה תלויה בכך שהאיבר בקודם ברשימה כבר קיבל את הצבתו. שימו לב שזהו Backtracking ולכן במידה ולאיבר ברשימה אין שום השמה חוקית, יש לעקוב לאחור ולנסות הצבות אחרות לאיברים הקודמים ברשימה.
הפונקציה תחזיר True במידה ומצאה הצבה חוקית כלשהי לכל איברי **list_of_items** (כלומר פתרון חוקי בו כל איבר ברשימה הוא מפתח ב **dict_items_to_vals** המצביע על ערך מתוך **set_of_assignments**).
במידה ולא מצאנו שום השמה חוקית אזי שלא קיים פתרון לבעיה ו**נחזיר False.**

רמז: legal_assignment_func תהיה פונקציה המקבלת את dict_items_to_vals, איבר מסויים X, ואת *args ובודקת האם ההשמה של X במילון חוקית.
יתכן ויקל עליכן לקרוא קודם את הסעיף הבא של הסודוקו לפני שתגשו לפתרון בעייה זו.
כתבו את הפתרון לפונקציה זו בקובץ **ex11_backtrack.py**

כמו שהוסבר, פונקציה זו היא כללית וצריכה לפתור מספר בעיות באמצעות אותו האלגוריתם:

1. **בעיית הסודוקו** https://en.wikipedia.org/wiki/Sudoku

את הפתרון לבעיה זו יישמו בקובץ ex11_sudoku.py, על ידי יישום של הפונקציות הבאות:

I. def load_game(sudoku_file):

הפונקציה תקרא קובץ המכיל טבלת סודוקו חצי מלאה כמו אלו המצורפות בקבצים sudoku_tables/sudoku_table2.txt, sudoku_tables/sudoku_table2.txt ו sudoku_tables/sudoku_table3.txt.
אתן יכולות להניח שהטבלאות חוקיות, כלומר הן מייצגות לוחות סודוקו התחלתיים בגודל 9X9 וערכיהם תקינים. כמו כן מקומות המסומנים '0' מגדירים מקום ריק שצריך למלא.
על הפונקציה להחזיר מילון שמפתחותיו הן צמדים של שורה וטור וערכיו הם המספר במיקום זה בלוח.
למשל עבור הלוח (המוקטן) הבא:

0,0
2,7

יהיה עליכן להחזיר את המילון:

{(0,0):0, (0,1):0, (1,0):2, (1,1):7}

פונקציה זו לא תיבדק על ידינו.

II. def check_board(board, X,*args):

פונקציה זו מקבלת כקלט את:

board - מילון בדומה למבנה המתואר בפונקציה הקודמת
X - צמד (tuple) המייצג מיקום בלוח row, column
*args - שיכיל את מה שתבחרו להעביר אליו מתוך general_backtrack.

הפונקציה בודקת האם מה שמוצב ב board[X] הוא הצבה חוקית. הצבה חוקית צריכה לקיים את התנאים הבאים:

- כל שורה צריכה להכיל עד הופעה אחת של כל ספרה
- כל טור צריך להכיל עד הופעה אחת של כל ספרה
- כל אחד מתשעת הריבועים של הלוח צריכים להכיל עד הופעה אחת של כל ספרה

שימו לב כי הפונקציה מניחה שהלוח היה חוקי לפני ההצבה האחרונה (כלומר מה שהוצב בX).
כמו כן שימו לב כי בתחילת התכנית, בלוח יהיו הרבה אפסים. הופעה כפולה (או יותר) של אפס בכל אחת משלוש הקטגוריות לעיל לא צריכה לפגוע בחוקיות של ההצבה בX (שאמורה להיות מספר בין 1 ל9).

הפונקציה תחזיר True במקרה וההשמה חוקית, False אחרת.

רמז: פונקציה זו צריכה להיות הפונקציה המועברת לgeneral_backtracking בתור legal_assignment_func: אתם אמורים לעשות שימוש בפונקציה זו בכדי לבדוק את תקינותה של

הצבה. שימו לב כי במידה ויש מספר אפסים באותה שורה/טור זה לא אומר שההצבה לא חוקית, רק שהתאים הללו עוד לא מולאו. כמו כן, ניתן לפתור את פונקציה זו בלי להעביר שום דבר ב*args.
פונקציה זו לא תיבדק על ידינו.

III. def run_game(sudoku_file, print_mode = False):

הפונקציה מקבלת קבצים מהפורמט שצוין ב**load_game**, ומנסה למצוא פתרון חוקי ללוח החלקי באמצעות general_backtracking. על הפונקציה להחזיר ערך בוליאני - True במידה וקיים פתרון ו False במידה ולא קיים.
שימו לב כי קבוצת ההשמות החוקיות ללוח סודוקו היא

set_of_assignments = range(1,10)

במידה וקיים פתרון וprint_mode=True, הדפיסו אותה למסך על פי ההוראות למטה. סעיף זה צריך לעשות שימוש בסעיפים הקודמים **וייבדק על ידינו** שאכן מצליח לפתור לוחות סודוקו שונים.

הערה: שימו לב שבמצב ההתחלתי הלוח יכול להיות עם ערכים שאסור לכם לשנות בזמן חיפוש הפתרון, כלומר מיקומים מסוימים בלוח כבר קבלו הצבה. חישבו כיצד להתמודד עם זה ב backtracking הכללית.

לנוחיותכן, צרפנו לקובץ פונקציה בשם print_sudoku. פונקציה זו מקבלת מילון שאיבריו הם צמדים של שורה וטור, המצביעים על השמה מסוימת, והיא מדפיסה את הלוח הנוצר על ידי צמדים אלו למסך.
קלט חוקי לפונקציה למשל:

borad ={(0,0):1, (0,1):7}

זמן הריצה לא אמור להיות ארוך במיוחד, אך במידה ואתן מעוניינות לדבג, אתן מוזמנות להוסיף בעצמכן טבלאות קטנות יותר, למשל של 3x3 לנוחיות הבדיקה.
רמז: מבין הקבצים שהעלנו עם התרגיל: לטבלאות 1 ו-2 קיים פתרון בעוד לטבלה 3 לא קיים.

2. **בעיית צביעת הגרפים** https://en.wikipedia.org/wiki/Graph_coloring
משימת צביעת גרפים היא בעיה קלאסית בה אנחנו רוצות לצבוע קדקודים בגרף במספר צבעים, כך שקדקודים שכנים (שיש להם צלע משותפת) יצבעו בצבעים שונים. אנחנו ניישם את בעיית צביעת הגרפים על מפות שונות של העולם, כך שקדקודי הגרף שלנו יהיו מדינות, ואלו תהיינה שכנות אם קיים להן גבול משותף. למשל עבור מפת העולם קדקודים אפשריים יכולים להיות ישראל, ירדן ועיראק. בין ישראל וירדן תהיינה צלע שכן הן מדינות שכנות, אך בין ישראל ועיראק לא תהיה צלע.

לצורך כך עליכן ליישם בקובץ **ex11_map_coloring.py** את שתי הפונקציות הבאות:

I. **def read_adj_file(filename):**

פונקציה היודעת לקרוא קובץ שכנויות, כמו אלו המצורפים ב,adj_example1.txt adj_usa_ex11.txt וadj_world_ex11.txt אשר נמצאים כולם בתיקייה adjacency_files.
הפונקציה הזו צריכה להחזיר מילון שמפתחותיו הן המדינות בקובץ, וכל מפתח מצביע על רשימת שכניו.

למשל עבור הקובץ:

A:B,C
B:A
C:A
D:

עליכן להחזיר את המילון:

{'A':['B','C'],'B':['C'],'C':['A'],'D':[]}

פונקציה זו לא תיבדק.

II. **def run_map_coloring(adjacency_file,num_colors = 4,map_type = None):**

פונקציה המקבלת קובץ שכנויות כפי שפורט לעיל, ומחפשת פתרון בו לכל קודקוד בקובץ (כלומר לכל מדינה) מושם צבע, כך שלכל שתי מדינות שכנות צבע שונה.
ערך ההחזרה של הפונקציה צריך להיות **מילון של הצבות חוקיות** שמפתחותיו הן המדינות בקובץ (מחרוזות), וערכיו הן הצבות חוקיות (מחרוזת המתארת צבע). במידה ולא קיימת השמה חוקית עליכן להחזיר **None**.

למשל עבור קובץ המכיל את המדינות הבאות ושכנותיהן:

A:B,C
B:A
C:A
D:

מקרה 1: קריאה עבורה **num_colors = 3**
פתרונות אפשריים יהיו למשל :

{'A': 'red', 'B': 'blue', 'C': 'green', 'D': 'red'}

או למשל:

{'A': 'red', 'B': 'blue', 'C': 'bule', 'D': 'red'}

שימו לב שיתכן פתרון שלא משתמש בכל הצבעים המותרים...
לעומת זאת הפתרון:

{'A': 'red', 'B': 'red', 'C': 'green', 'D': 'red'}

אינו חוקי כי A וB מדינות שכנות ובעלות אותו הצבע.

אתן יכולות להניח כי המספר המקסימלי של צבעים שתקבלו הוא שישה ולהשתמש בחלק או בכל הצבעים הבאים (כתלות ב-num_colors**):**

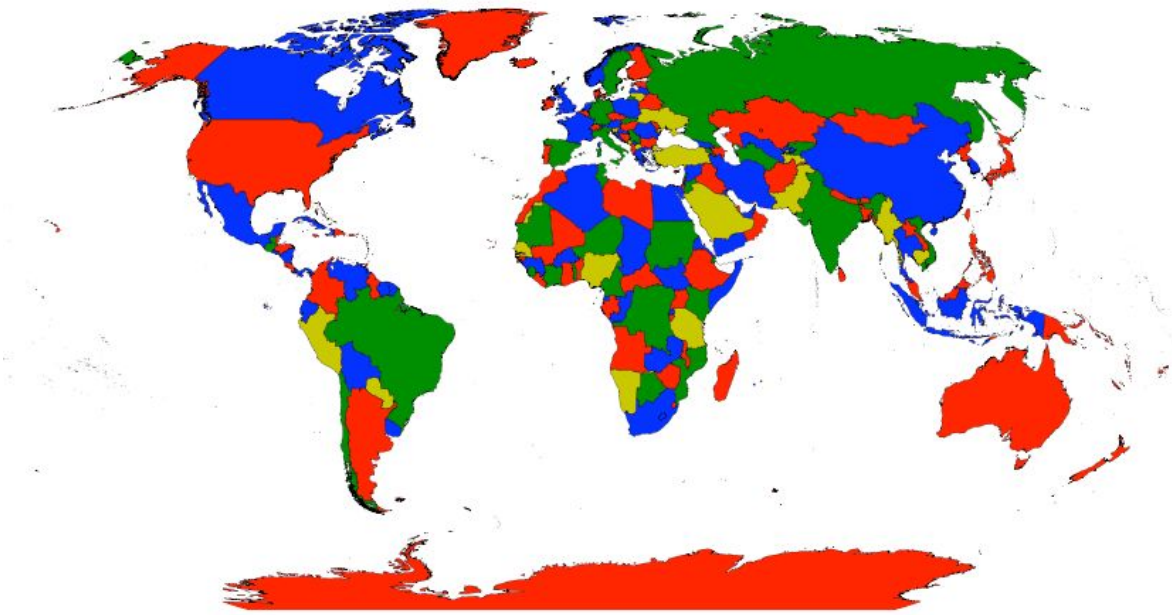
COLORS = ['red','blue','green','magenta','yellow','cyan']

מקרה 2: קריאה עבורה **num_colors = 1** אין אף פתרון חוקי ועליכן להחזיר None.

כמו בבעיית הסודוקו, גם כאן עליכן להשתמש בפונקציה general_backtracking. חישבו שוב מה הם המשתנים שברצונכן להעביר לפונקציה. זכרו כי עליכן להעביר מצביע לפונקציה הבודקת האם הצבה מסוימת היא חוקית, הלא היא legal_assignment_func. כלומר הגדירו פונקציה , בשם לבחירתכן, שתבדוק את תקינות השמת הצבע במשחק המפות.

רמז: מומלץ להשתמש ב*args של general_backtracking ע"י העברת מבנה הנתונים אותו יצרתן ב **read_adj_file,** כלי זה יעזור לכן לקבוע האם הצבה ספציפית היא חוקית.
פונקציה זו **כן** תיבדק על ידינו.

אם הפונקציה הצליחה למצוא השמה, תוכלו לבדוק את עצמכם ע"י ציור מפת התוצאות- לנוחיותכן יישמנו בקובץ map_coloring_gui.py פונקציה בשם color_map המקבלת כהשמה מילון של מדינות והשמה של הצבע שלהן, וכן את המשתנה map_type שיכול להיות "world" או "USA", ומייצרת מפה גרפית על המסך כמו זו המופיעה כאן.



שימו לב, לצורך תצוגה זו עליכן להתקין מספר ספריות פייתון למחשב שלכן. על כן, חלק זה אינו חובה ואתן יכולות להניח שהקלט לפונקציה **run_map_coloring** הוא תמיד map_type=None. יחד עם זאת, אנחנו ממליצות להתנסות בהתקנת ספריות, זה כלי חשוב וגם המפות שיוצאות הן נחמדות. במחשבים של מדעי המחשב הספריות מותקנות. פרטים על התקנת הספריות ניתן למצוא בתחתית הקובץ.

כדאי לקרוא גם על משפט ארבעת הצבעים הגורס כי כל מפה דו ממדית ניתנת לצביעה על ידי ארבעה צבעים בלבד:
https://en.wikipedia.org/wiki/Four_color_theorem

קיים פתרון כזה עבור מפת העולם (למשל זה המופיע לעיל), אך מציאתו עלולה להיות ממושכת באלגוריתם סטנדרטי של גישוש נסוג ועל כן יגיע חלק ב של התרגיל.

3. **פתרון בעיות אחרות:** בנוסף נבדוק את **general_backtracking** על בעיה קלאסית נוספת הניתנת לפתרון ברקורסיה, לא נאמר לכן מראש מה היא, אך במידה ויישמתן את הפונקציה בצורה כללית מספיק היא תהיה מסוגלת להתמודד עם כל פונקציה רקורסיבית.

Part B: improve backtrack

את הפתרון לחלק זה עליכן לכתוב בקובץ **ex11_improve_bactrack.py**.

פתרון בעיות ברקורסיה עלול להיות מורכב מאוד. חישבו למשל על בעיית צביעת המפות. בקובץ המדינות המלא שקבלתן, ישנן 239 מדינות. בפתרון הפשוט ביותר, ננסה את כל האפשרויות בBruthforce, כלומר השמה של צבע כלשהו לכל מדינה. כמות האפשרויות, שישנן היא 4 בחזקת 239, מספר אסטרונומי.
Backtracking מונע את הצורך בסריקת מרחב האפשרויות כולו, שכן השמות שאינן חוקיות מונעות בדיקה של המפה כולה (למשל, במידה ונבחר את שתי המדינות השכנות הראשונות שסרקנו לאותו הצבע, אזי שכל הפתרונות הלא חוקיים המכילים את האפשרות הזו לא יבדקו), אך עדיין דורש סריקה של מספר אקספוננציאלי של השמות. על כן ננסה בחלק זה לשפר את האלגוריתם שכתבתן בחלק א.
שימו לב 1! אתן לא צריכות בשלב זה לעבוד עם האלגוריתם מחלק א. בחלק זה הפונקציות שתיישמו מקבלות כקלט מילונים בדומה לאלו שייצרתן ב**read_adj_file.** הפונקציות תקראנה לפונקציות עזר רקורסיביות שפועלות חכם יותר מאשר general_becktrack.
שימו לב 2!! פונקציות העזר הרקורסיבית יכולות לקבל איזה מבני נתונים שתגדירו, אין חובה להיצמד לפרוטוטייפ כמו של general_becktrack (אך מומלץ להעזר בו), חישבו איזה מבני נתונים חשובים לכם עבור פילטור ההשמות החוקיות שהעדכון שלהם יהיה הכי נוח לשימוש באמצעות הרקורסיה.
בפרט מותר לכן לייבא פונקציות שיישמתן בex11_map_coloring.py.

לצורך שיפור האלגוריתם תיישמו את ארבעת השיפורים המוצעים, בשלב הראשון כל אחד מהם בנפרד.
שימו לב שלכל אחת מהן אתן יכולות להגדיר פונקציות עזר.

1. יישמו את הפונקציה

def back_track_degree_heuristic(adj_dict, colors):

המקבלת את:
adj_dict - מילון בדומה למילון הפלט של **read_adj_file**,
ו**colors** - רשימה של צבעים שהשמתם חוקית (כלומר אורכה של הרשימה הוא מספר ההצבות החוקיות, למשל הרשימה ['red','blue','green','magenta'], תייצג דרישה לפתור את הבעייה עם ארבעה צבעים בלבד)
ופותרת גם היא את בעיית צביעת המפות אך פועלת לפי הכלל הבא:

מבין כל המדינות שעדיין לא קבלו השמה, **נבחר להשים צבע קודם כל למדינה בעלת מספר השכנים המקסימלי**. במידה וישנן מספר מדינות בעלות מספר דומה של שכנים נבחר אחת מהן ללא חשיבות לסדר. למשל אם המדינות שנותרו לנו הן ישראל בעלת חמש שכנות, לבנון עם שלוש שכנות וקפריסין ללא מדינות שכנות, אזי שקודם כל נחפש השמה חוקית לישראל, אז ללבנון ורק אז לקפריסין.
הפונקציה מחזירה מילון תקין בדומה למילון ב**run_map_coloring** עם השמה חוקית אם קיים כזה, ו None אחרת.

2. יישמו את הפונקציה

def back_track_MRV(adj_dict, colors):

עבורה אלגוריתם הרקורסיה מפעיל את הכלל minimum_remaining_values:

סדר ההתקדמות על פני הקדקודים (המדינות) השונים יהיה על פי לאיזה קדקוד יש בכל רגע נתון הכי מעט השמות חוקיות. למשל אם נותרו שלוש מדינות שעדיין לא קבלו השמה, לאחת מהן עדיין כל ההשמות חוקיות, אך לאחרת שתי השמות כבר אינן חוקיות (שכן שכנותיה קבלו כבר צבע מסוים ועל כן היא לא יכולה לקבל אותו), ולמדינה השלישית נותרה רק השמה חוקית אחת, אזי שנבחר בצעד הבא להתחיל דווקא במדינה השלישית, שכן לה באותו רגע נתון הכי מעט השמות חוקיות. במידה ולמספר מדינות אותו מספר מינימלי של השמות חוקיות, נבחר אחת מהן ללא חשיבות לסדר. שימו לב שבאלגוריתם זה סדר ההתקדמות על המדינות יכול להשתנות מאיטרציה לאיטרציה.

הפונקציה מחזירה מילון תקין עם השמה חוקית אם קיים כזה, וNone אחרת.

3. יישמו את הפונקציה

def back_track_FC(adj_dict, colors):

הבודקת האם השמה מסוימת היא חוקית לא רק עבור המדינה שאותה השמתן הרגע, אלא גם האם ההשמה הנ"ל יצרה מצב בו למדינה מסוימת אחרת לא נותרו השמות חוקיות בכלל (Forward Checking). כלומר האם בעקבות ההשמה החדשה, לאחת השכנות שלה לא נותרו השמות חוקיות, במידה ולא, השמה זו אינה חוקית.
הפונקציה מחזירה מילון תקין עם השמה חוקית אם קיים כזה, וNone אחרת.

4. יישמו את הפונקציה

def back_track_LCV(adj_dict, colors):

הרומזת לנו באיזה צבע כדאי לבחור על פי הכלל Least Constraining Value.
כלומר הרקורסיה מתקדמת בסדר כלשהו (יכול להיות אקראי), אך עבור כל מדינה בודקת איזה צבע כדאי לנסות להציב לה על באופן הבא:
נניח המדינות שנותרו הן ישראל סוריה וירדן כולן שכנות.
לישראל מותר עדיין לבחור את הצבע האדום והירוק
לירדן את הצבע האדום והכחול
ולסוריה רק את הצבע האדום.

כעת תור ישראל, האלגוריתם שלכן יבדוק את המשמעות של הצבת אדום או ירוק לישראל. בחירת הצבע האדום תותיר לירדן אפשרות אחת (הצבע הכחול) ולסוריה 0 אפשרויות (סה"כ 1+0 = 1).

בחירת הצבע הירוק לישראל תותיר לירדן שתי אפשרויות וסוריה אפשרות אחת (סה"כ 3 אפשרויות) ולכן הרקורסיה תתקדם כך שהיא תתחיל בניסיון להציב לישראל את הצבע הירוק, כי הצבע הזה יוצר פחות מגבלות על שכנותיה, ורק אם ניסיון זה לא צלח תנסה להציב לישראל את הצבע האדום.
הפונקציה מחזירה מילון תקין עם השמה חוקית אם קיים כזה, וNone אחרת.

אתן רשאיות כמובן להוסיף פונקציית main, שתקרא את קבצי השכנויות, תריץ את הפתרונות השונים ואף תציגם למסך. פונקציה זו לא תיבדק (בניגוד כמובן לארבע פונקציות לעיל).

Part C: Bonus Competition

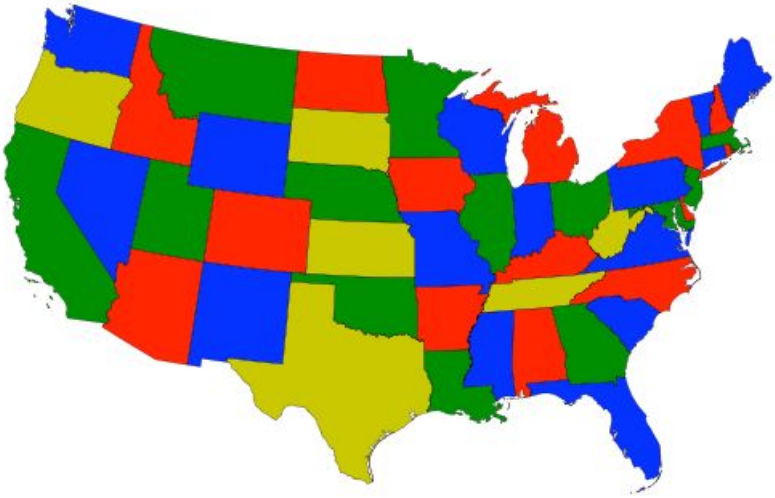
בונוס:
כתבו באותו קובץ פונקציה הנקראת

def fast_back_track(adj_dict, colors):

הפותרת את בעיית צביעת המפה בצורה המהירה ביותר. אתן רשאיות להשתמש בכל אחת מהשיטות שהוצעו לעיל וברעיונות נוספים בכדי שאלגוריתם שלכן יחזיר פתרון מהיר ככל הניתן. שימו לב שנבדוק את הבעיה על מפה חדשה ולא על אחת המפות שקבלתן (בכדי שהפונקציה שלכן לא פשוט תחזיר פתרון חוקי).
הפונקציה מחזירה מילון תקין עם השמה חוקית אם קיים כזה, וNone אחרת.
הפתרונות במיקום 1-30 יזכו ל10 נקודות בונוס בתרגיל זה. מהירות הפתרון תיבדק על המחשבים של האוניברסיטה והזוכות יוכרזו יחד עם קבלת הציונים.

שימו לב שהקוד שלכם חייב להיכתב בפייתון, ואין לנסות לקמפל את הקוד מראש. עליכן לכתוב קוד הפותר את העקיבה לאחור בצורה המהירה ביותר.

הצגה למסך של המפות.
כאמור יישמנו עבורכם בקובץ map_coloring_gui.py פונקציות המסוגלות להציג את מפת העולם ואת מפת ארצות הברית. למשל:



בכדי להריץ את הקובץ תצטרכו מספר ספריות:

1. שתי הספריות הראשונות הן numpy ומatplotlib , המגיעות עם רוב הספריות הסטנדרטיות של פייתון. במידה והשורות

```
import matplotlib.pyplot as plt
import numpy as np
```

לא עובדות במחשבים שלכן, אנא התקינו את חבילת scipy הכוללת את שתיהן:
https://scipy.org/install.html

2. החלק השני מעט מורכב יותר ודורש התקנת של ספריה בשם Basemap:
הוראות התקנה נמצאות כאן:
https://matplotlib.org/basemap/users/installing.html
בסיום ההתקנה השורה:

```
from mpl_toolkits.basemap import Basemap
```

צריכה לרוץ ללא תקלות.

3. בנוסף באתר הקורס תמצאו בתוך קובץ הzip המצורף לתרגיל את התיקיות world_map וusa_map. שימרו אותן בתיקייה בהם אתם מריצים את התרגיל.

4. לבדיקה האם ההתקנה הצליחה אתן יכולות לקרוא לפונקציה color_map עם מילון ריק או מילון המכיל המדינה אחת. למשל

```
from map_coloring_gui import color_map
color_map({'Argentina':'red'},map_type='world')
```

או

```
color_map({'New York':[1.0,0.0,0.0]},map_type='USA')
```

כאשר הצבעים יכולים להיות כל צבע מהספרייה הסטנדרטית של matplotlib (ערכי RGB מנורמלים ל1) או שמות של צבעים מתוך: https://matplotlib.org/examples/color/named_colors.html

הנחיות הגשה

הגישו קובץ zip הנקרא ex11.zip ומכיל את ארבעת הקבצים של התרגיל:
ex11_backtrack.py, ex11_sudoku.py, ex11_map_coloring.py, ex11_improve_backtrack.py
וכן קובץ **README**
קובץ הzip צריך להכיל את חמשת הקבצים הללו בלבד.
בפרט אין להגיש את הקבצים הנוספים המופיעים בספריית התרגיל.

* התרגיל מנוסח בלשון נקבה אך פונה לכל התלמידים והתלמידות בקורס.