Problem Set 2

Problem 1: Word Search II

הסבר על הטמפרטורה:

לוח הזמנים של הטמפרטורה קובע כיצד "הטמפרטורה" של המערכת יורדת עם הזמן, ומשפיעה על ההסתברות לקבל פתרונות גרועים יותר. לוח הזמנים מתחיל בטמפרטורה גבוהה יותר כדי לעודד חקר ומוריד אותה בהדרגה כדי לאפשר התכנסות לפתרון.

```
def temperature_schedule(t):
        return 100 / (1 + 0.1 * math.log(t + 1))
```

טמפרטורה התחלתית: האלגוריתם מתחיל בטמפרטורה גבוהה. זה מאפשר לחקור יותר את מרחב החיפוש באיטרציות הראשוניות.

קירור לוגריתמי: פונקציית הקירור, מורידה את הטמפרטורה בהדרגה. הפונקציה הלוגריתמית גדלה לאט, מבטיחה שירידות הטמפרטורה אינן חדות מדי, מה שמאפשר גמישות מסוימת במהלך החיפוש.

לסיכום: הטמפרטורה יורדת בהתמדה, מה שמאפשר חקר רב יותר בהתחלה ומחדד את החיפוש בהדרגה כשהמערכת מתקררת. זה עוזר להימנע מללכוד במינימום מקומי בשלבים המוקדמים תוך הבטחת התכנסות לקראת פתרון בהמשך.

סקירה כללית של אלגוריתם גנטי:

1. **ייצוג פתרון:**
   האלגוריתם שומר על אוכלוסייה של פתרונות פוטנציאליים (אינדיבידואלים), מעריך את האיכות שלהם באמצעות פונקציית התאמה (fitness), בוחר הורים לפי ההתאמה שלהם, ומבצע ריבוי (crossover) ומוטציה כדי לייצר צאצאים חדשים. באמצעות תהליך זה, האלגוריתם שואף למצוא מסלול של מילים על ידי חיפוש יעיל במרחב הפתרונות, תוך כדי שמירה על גיוון ומניעת קיבוע על פתרונות לא אופטימליים.
2. **הערכת איכות הפתרון:** האיכות של כל פתרון מוערכת באמצעות פונקציית fitness:

```
def fitness(individual, goal_word):
    last_word = individual[-1]
    matches = sum(1 for i in range(min(len(last_word), len(goal_word))) if last_word[i] == goal_word[i])
    length_difference = abs(len(last_word) - len(goal_word))
    return matches - (length_difference * 0.5)  # Adjust the weight based on experimentation
```

פונקציה זו מעריכה עד כמה קרובה המילה האחרונה ברצף למילת המטרה. זה סופר את התווים התואמים ומעניש על סמך הפרש האורך כדי לגזור ציון התאמה. ככל שהציון גבוה יותר, כך הפרט טוב יותר.

3. **שילוב פתרונות (רפרודוקציה):** האלגוריתם משלב שני פתרונות אב ליצירת צאצאים באמצעות פונקציית reproduce:

```
def reproduce(parent1, parent2, dictionary):
    # Now ensure the children are valid transformations by checking each crossover
    crossover_point = random.randint( a: 1, min(len(parent1), len(parent2)) - 1)
    new_individual = parent1[:crossover_point] + parent2[crossover_point:]

    # Check that the new individual is a valid sequence of words
    if all(word in dictionary for word in new_individual):
        return new_individual

    # If not valid, fallback to one of the parents
    return random.choice([parent1, parent2])
```

פונקציה זו מבצעת פעולת הצלבה בנקודה שנבחרה באקראי. לאחר ההצלבה, הוא בודק אם הרצף המתקבל תקף על ידי הבטחת כל המילים קיימות במילון. אם זה לא תקף, זה מחזיר את אחד ההורים.

4. **תהליך מוטציה:** מוטציה משמשת להחדרת שונות ולמניעת התכנסות מוקדמת של האוכלוסייה. הפונקציה mutate עושה זאת:

```
def mutate(individual, dictionary):
    if random.random() < MUTATION_RATE:
        current_word = individual[-1]
        # Randomly choose to either mutate a letter or replace the last word with a valid neighbor
        if random.choice([True, False]):
            mutation_point = random.randint( a: 0, len(current_word) - 1)
            for letter in 'abcdefghijklmnopqrstuvwxyz':
                if letter != current_word[mutation_point]:
                    new_word = current_word[:mutation_point] + letter + current_word[mutation_point + 1:]
                    if new_word in dictionary:
                        individual.append(new_word)
                        return individual  # Return after successful mutation
        else:
            # Replace the last word with a valid neighbor
            neighbors = get_neighbors(dictionary, current_word)
            if neighbors:
                new_word = random.choice(neighbors)
                individual.append(new_word)
    return individual  # If no mutation occurred, return original
```

פונקציה זו משתמשת בשיעור מוטציה של 0.1 וזה משנה אות בתוך המילה האחרונה או מחליף אותה בשכן שנבחר באקראי מהמילון. זה מאפשר לחקור נתיבים חדשים.

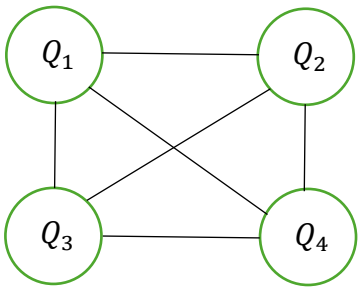
Problem 2: CSP I

פתרון הבעיה כ-CSP:

1. משתנים:
   - כל משתנה Q_i מייצג את העמודה i בה המלכה נמצאת, כך ש- i∈{1,2,3,4} .
2. דומיין:
   - התחום של כל משתנה Q_i הוא {1,2,3,4} ,המייצג את כל השורות האפשריות בהם ניתן למקם מלכה.
3. אילוצים:
   - **לא יהיו שתי מלכות באותה שורה:** $Q_i \neq Q_j$ לכל $i \neq j$
   - **לא יהיו שתי מלכות באותו אלכסון:** $|Q_i - Q_j| \neq |i - j|$ לכל $i \neq j$
   - דרך אופן הגדרת המשתנים אין צורך להגדיר את האילוץ שלא יהיו שתי מלכות באתה עמודה, אך האילוץ מתקיים.

גרף האילוצים:

כל קודקוד מייצג את מיקום אחת מהמלכה באחת העמודות, וקיימות קשתות בין כל הקודקודים בכדי לייצג את האילוצים בין כל זוג מלכות.



Problem 3: CSP II

פרמטריים מרכזיים:

n: מספר המשתנים

d: הגודל המקסימלי של הדומיין של המשתנים

c: מספר האילוצים בין המשתנים

ניתוח מורכבות זמן:

אתחול: בתחילה האלגוריתם בונה תור שמכיל את כל הקשתות עבור כל האילוצים. לכן יכולות להיות עד c קשתות. לכן זמן ריצת האתחול - **O(c)**

עיבוד: הלולאה המרכזית ממשיכה עד שהתור ריק, כל קשת יכולה להיות מעובדת מספר פעמים. במקרה הגרוע ביותר יהיה צורך לבקר מחדש בכל הקשתות עבור כל המשתנים, מה שיוביל למורכבות זמן של- **O(c)** עבור עיבוד התור.

תיקון קשת: עבור כל קשת קריאה לפונקציה REVISE תפעל על הדומיין של הקשת הספציפית, מול הקשת אליה הוא בודק. במקרה הגרוע ביותר אנו נבדוק על תרחיש בדומיינים של שתי הקשתות , מה שיגרור זמן ריצה של **O(d^2)**. בגלל שגודל הדומיין המקסימלי הינו d.

מסקנה: משמעות הדבר היא שלכל אחת מקשתות **O(c)**, אנו עלולים לעלות בעלות זמן של **O(d^2)**. ולכן זמן הריצה הכולל הינו **O(c ·d^2)**.