השלמת משפטים אוטומטית

מבוא

לצורך הפיכת חווית המשתמשים במנוע החיפוש גוגל לטובה יותר, החליטו צוות הפיתוח לאפשר השלמה של משפטים מתוך מאמרים, תיעוד וקבצי מידע בנושאים טכנולוגים שונים.

משימה

עליכם לכתוב תוכנית אשר תומכת בשתי פונקציות עיקריות:

- פונקציית אתחול - מטרת הפונקציה היא לקבל רשימה של מקורות טקסט שעל בסיסם ירוץ מנוע החיפוש, כל מקור מכיל אוסף של משפטים. חלק זה ניתן לכתוב בכל שפה לפי שיקול דעתכם ולשמור את התוצאות בצורה המיטבית לטובת פונקציית ההשלמה.
- פונקציית השלמה - על הפונקציה לקבל מחרוזת - אשר מהווה את הטקסט שהמשתמש הקליד - על הפונקציה להחזיר את חמש ההשלמות הטובות ביותר (השלמה טובה תוגדר בהמשך). את הפונקציה הזאת צריך לכתוב בשפת פייתון.
  העשרה בלבד: כתיבת פונקציית ההשלמה בשפת *C++* על מנת לקבל ביצועים טובים
- חתימת פונקציית ההשלמה - לכל השלמה שנמצאה (סה"כ 5 השלמות):
  - מה המשפט שהושלם
  - לאיזה מקור המשפט המלא שייך
  - באיזה מיקום הטקסט שהמשתמש הזין נמצא ביחס למשפט שהושלם (OFFSET)
  - מה ה-SCORE של ההשלמה

לנוחיותכם מצורף החתימה של הפונקציה הנ"ל

```
get_best_k_completions(prefix: str) -> List[AutoCompleteData]
```

כאשר AutoCompleteData - היא המחלקה הבאה:

```
@dataclass
class AutoCompleteData:
    completed_sentence: str
    source_text: str
    offset: int
    score: int
    # methods that you need to define by yourself
```

ממשק בשפת C++:

```
vector<AutoCompleteData> GetBestKCompletions(const string& prefix);
```

כאשר AutoCompleteData - היא המחלקה הבאה:

```
class AutoCompleteData {
  private:
    string completed_sentence;
    string source_text;
```

```
    int offset;
    int score;
    // methods that you need to define by yourself
};
```

פעולת ההשלמה

המטרה של פעולת ההשלמה היא להקל על המשתמש למצוא את המשפט המתאים ביותר.
נגדיר התאמה בין פסוק (משפט) לבין טקסט שהמשתמש הקליד אם:

- הטקסט הוא תת מחרוזת של הפסוק (זה כולל תחילת, אמצע או סוף הפסוק).
- טקסט שבו אם נבצע תיקון אחד אז הטקסט יהווה תת מחרוזת של הפסוק.

תיקון מוגדר כ:

- החלפת תו
  - **דוגמה:** המחרוזת "**להיות או לו**" היא נחשבת כתת מחרוזת של הטקסט "**להיות או לא** להיות זאת היא השאלה", עם החלפת התוו "ו" במילה "לו" ל- "א".
- מחיקת תו.
  - **דוגמה:** המחרוזת "**להות או לא**" היא נחשבת כתת מחרוזת של הטקסט "**להיות או לא** להיות זאת היא השאלה", עם מחיקת התוו "י" במילה "להיות".
- הוספת תו.
  - **דוגמה:** המחרוזת "**להיות אוו לא**" היא נחשבת כתת מחרוזת של הטקסט **"להיות או לא** להיות זאת היא השאלה", עם הוספת התוו "ו" במילה "או".

ציון התאמה

במקרה של מספר התאמות לטקסט שהוקלד, נגדיר ציון (score) לכל התאמה:

- הניקוד הבסיסי הוא כפול ממספר התווים שהוקלדו ועבורם נמצאה התאמה.
- החלפת תו מורידה לפי הפירוט הבא: תו ראשון 5 נק', תו שני 4, תו שלישי 3, תו רביעי 2, תו חמישי והלאה 1.
- מחיקת תו או הוספת תו מקבלים הורדה של 2 נק' למעט 4 התווים הראשונים (תו ראשון 10 נק', תו שני 8, תו שלישי 6, תו רביעי 4).

דוגמאות עבור המשפט "**להיות או לא להיות, זאת היא השאלה**"

- הטקסט "להיות או לא" מקבל 22 נק' כי זה מתאים למחרוזת של הטקסט ומכיל 11 תוים.
- הטקסט "להיות או לו" מקבל 18 נק' כי יש 10 תווים נכונים בהתאמה למחרוזת של הטקסט ותו אחד שהוחלף.
- הטקסט "להיות או לא" מקבל 16 נק' כי יש 10 תווים נכונים בהתאמה למחרוזת של הטקסט ותו אחד שהוחלף במקום הרביעי.
- הטקסט "להייות או לא" מקבל 18 נק' כי יש 11 תווים נכונים בהתאמה למחרוזת של הטקסט ותו אחד שנוסף במקום הרביעי.
- הטקסט "להות או לא" מקבל 14 נק' כי יש 10 תווים נכונים בהתאמה למחרוזת של הטקסט ותו אחד שחסר במקום השלישי.

הערות לגבי הקלט והפלט:

שימו לב:

- אתם תקבלו קבצי טקסט שכתובים בשפה האנגלית בלבד בנוסף לפיסוקים ($,.!@ וכו...) שמכילים אוסף משפטים (משפט מוגדר כשורה שלמה בתוך הקובץ).
- קבצי הטקסט מאוחסנים במבנה של עץ תיקיות, והם נמצאים בתיקיה Archive.zip
  - הטקסטים ממוקמים בגבהים שונים בעץ, כלומר קובץ טקסט יכול להיות בתוך תיקיה, בתוך תיקיה שבתוך תיקיה, וכדומה.
- אין צורך שהמשתמש יקליד אותיות גדולות\קטנות, פיסוקים בזמן הקלדת טקסט הקלט, בנוסף אין הגבלה על מספר הרווחים בין המילים. כלומר אם המשפט המקורי הוא "להיות או לא להיות, זאת היא השאלה" - אז גם אם המשתמש הקליד "להיות זאת", "להיות, זאת" או "להיות זאת" אזי על המערכת להתייחס לשלוש תתי מחרוזות כשקולות.
- **הפלט** של המערכת אמור להיות שורה מתוך קבצי המקור בצורתו המקורית (כלומר כולל פיסוקים) ואת הנתיב של הקובץ.

תוכנית שלמה

- עליכם לספק תוכנית אשר רצה בשני שלבים:
  - שלב ראשון (offline) הוא שלב בו המערכת קוראת את קבצי הטקסט (ממקום ידוע מראש) ומכינה אותם לשלב השירות (serving)
  - שלב שני (online) הוא שלב בו המערכת מחכה לקלט.
    - ברגע שהמשתמש מקליד תווים ולוחץ על **Enter** המערכת מציגה את חמשת ההשלמות הטובות ביותר(במקרה של שוויון על המערכת למיין את המחרוזות עם ציון זהה לפי אלפבית).
    - לאחר הצגת ההשלמות, המערכת מאפשרת למשתמש להמשיך להקליד מהמקום בו הוא עצר.
    - אם המשתמש מקליד "#" המשמעות היא שהמשתמש סיים הקלדה עבור משפט זה וצריך לחזור למצב ההתחלתי.

דוגמה

```
Loading the files and prepariong the system...
The system is ready. Entar your text:
this is
Here are 5 suggestions
1. Store a Python object in a C object pointer.  This is similar to (arg 332)
2. like objects.  **This is the recommended way to accept binary (arg 121)
3. "PyObject_NewVar()".  This is normally called from the "tp_dealloc" (allocation 49)
4. "tp_itemsize" field of *type*.  This is useful for implementing (allocation 40)
5. Information about writing a good bug report.  Some of this is (bugs 87)
this is
```

בדיקת תרגיל

התרגיל נמדד לפי שני מדדים עיקריים (metrics)

- נכונות הפתרון - כלומר האם המשתמש קיבל את ההשלמות הנכונות.

- יעילות ההשלמה - כלומר כמה זמן לקח עד שהמערכת החזירה את ההשלמות.

שלבים מומלצים

להלן המלצה למימוש התרגיל אך אינה מחייבת

- שלב ראשון:
  - לחשוב על הבעיה ולהבין אותה טוב בלי לכתוב קוד, רק לנסות להבין איזה אובייקטים צריך ואיך הכל משתלב ביחד.
- שלב שני:
  - לבנות תוכנית מינימלית ושלמה, כלומר
    - לא להתעסק עם הנתונים - להניח שקיים נתונים בצורה שהייתם רוצים שיהיו
    - לא להתעסק עם פלט - להניח שיש לכם פונקצית טיפול בפלט
    - לא לממש את כל הפונקציונליות - כלומר לממש השלמה פשוטה בלבד
    - לממש את הקוד כך שיהיה אפשר להרחיבו בעתיד.
- שלב שלישי:
  - להשלים את הקוד שיעבוד בצורה מושלמת מבחינת נכונות הפונקציונליות
- שלב רביעי:
  - לעבד את הקלט
  - לעבד את הפלט