STARTUP NATION CENTRAL

Excellenteam

אקסלנטים באקדמיה – פרויקט גוגל!

"השלמת משפטים אוטומטית"

מבוא

לצורך הפיכת חווית המשתמשים במנוע החיפוש גוגל לטובה יותר, החליט צוות הפיתוח לאפשר השלמה אוטומטית של משפטים. זאת על בסיס מאגר של מאמרים, תיעוד וקבצי מידע בנושאים טכנולוגים שונים, אשר מהווים מושאי חיפוש נפוצים.

משימה: שלב א' – פונקציונאליות

עליכם לכתוב תוכנית אשר תומכת בשתי פונקציות עיקריות:

1) **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.

2) **פונקציית השלמה** – על הפונקציה לקבל מחרוזת אשר מהווה את הטקסט שהמשתמש הקליד, ולהחזיר את חמש ההשלמות הטובות ביותר (השלמה טובה תוגדר בהמשך). הפונקציה הזאת תיכתב בשפת פייתון.

מותר (ואף מומלץ!) להשתמש בכלי AI+LLM (למשל Gemini) בכדי להאיץ את ההגעה לקוד שפותר את האתגר הנ"ל. חשוב לבדוק ולוודא שהפתרון המתקבל אכן מתאים לבעיה ופותר אותה באופן מלא לפי ההנחיות, וחשוב <u>להבין היטב</u> את הפתרון במלואו כולל יכולת לתאר במדויק כיצד פועל.

הנחיות

<u>חתימת פונקציית ההשלמה – לכל השלמה שנמצאה (סה"כ 5 השלמות):</u>

- מה המשפט שהושלם
- לאיזה מקור שייך המשפט המלא
- באיזה מיקום/שורה בתוך המקור (OFFSET) נמצא המשפט שמתאים לטקסט שהמשתמש הזין
- מה הציון (SCORE) של ההשלמה

לנוחיותכם מצורפת 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
```

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 התחלה, אמצע או סוף של הפסוק)
- טקסט שבמידה ונבצע בו "תיקון" (אחד לכל היותר) אזי הוא יהווה תת-מחרוזת של הפסוק

"תיקון" מוגדר כאחד מבין שלוש הפעולות הבאות:

1. **החלפת תו**

- דוגמה: הטקסט **"**להיות או לו**"** נחשב כתת-מחרוזת של הפסוק **"**להיות או לא להיות זאת השאלה**"**, עם החלפת התו "ו" במילה "לו" לתו "א".

2. **הוספת תו**

- דוגמה: הטקסט **"**להות או לא**"** נחשב כתת-מחרוזת של הפסוק **"**להיות או לא להיות זאת השאלה**"**, עם הוספת התו "י" בשביל לייצר את המילה "להיות".

3. **מחיקת תו**

- דוגמה: הטקסט **"**להיות אוו לא**"** נחשב כתת-מחרוזת של הפסוק **"**להיות או לא להיות זאת השאלה**"**, עם מחיקת התו "ו" בשביל לייצר את המילה "או".

ציון התאמה:

במקרה של "התאמה" שנעשתה לטקסט שהוקלד, נגדיר לו "ציון" (score) באופן הבא:

- הניקוד הבסיסי הוא כפול ממספר התווים שהוקלדו ועבורם נמצאה "התאמה".

- החלפת תו מורידה ניקוד לפי הפירוט הבא:
תו ראשון – 5 נקודות, תו שני – 4, תו שלישי – 3, תו רביעי – 2, תו חמישי ואילך – נקודה אחת.

- הוספת תו או מחיקת תו תביא להורדה של 2 נקודות, אלא אם מדובר באחד מ-4 התווים הראשונים שמחירם גבוה יותר:
תו ראשון 10 נקודות, תו שני 8, תו שלישי 6, תו רביעי 4.

דוגמאות "ציון" עבור המשפט "להיות או לא להיות, זאת השאלה" (תיאור נוסף- בנספח):

- הטקסט "להיות או לא" מקבל 22 נק' כי זה מתאים (באופן מלא) למחרוזת של הטקסט ומכיל 11 תווים.

- הטקסט "להיות או לו" מקבל 19 נק' כי יש 10 תווים נכונים בהתאמה למחרוזת של הטקסט ותו אחד שהוחלף (1 – 10x2).

- הטקסט "להיות או לא" מקבל 18 נק' כי יש 10 תווים נכונים בהתאמה למחרוזת של הטקסט ותו אחד שהוחלף במקום הרביעי (2 – 10x2).

- הטקסט "להייות או לא" מקבל 18 נק' כי יש 11 תווים נכונים בהתאמה למחרוזת של הטקסט ותו אחד שנוסף במקום הרביעי (4 – 11x2).

- הטקסט "להות או לא" מקבל 14 נק' כי יש 10 תווים נכונים בהתאמה למחרוזת של הטקסט ותו אחד שחסר במקום השלישי (6 – 10x2).

הערות לגבי הקלט והפלט:

- אתם תקבלו קבצי טקסט שכתובים בשפה האנגלית בלבד וכוללים בנוסף גם סימני פיסוק (@!.,$ וכו'). קבצי הטקטס מכילים אוסף של משפטים (כאשר 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 ידייק בהקלדת אותיות גדולות\קטנות או סימני פיסוק בזמן הקלדת טקסט הקלט, בנוסף אין הגבלה על מספר הרווחים בין המילים. כלומר אם הפסוק המקורי הוא "להיות או לא להיות, זאת השאלה" – אזי גם אם המשתמש יקליד "להיות זאת," "להיות, זאת" או "להיות זאת" אזי על המערכת להתייחס לשלוש תתי-המחרוזות כשקולות. (הציון של הטקסטים הללו יהיה זהה)

- הפלט של המערכת אמור להיות שורה מתוך קבצי המקור בצורתו המקורית (כלומר כולל סימני פיסוק), ויכלול את הנתיב של הקובץ.

תוכנית שלמה

עליכם לספק תוכנית אשר רצה בשני שלבים:

1) שלב ראשון (offline) הוא שלב בו המערכת קוראת את קבצי הטקסט (ממקום ידוע מראש) ומכינה אותם לשלב השירות (serving).
2) 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 בעלות הציון הזהה לפי סדר האלפבית).

לאחר הצגת ההשלמות, המערכת מאפשרת למשתמש להמשיך להקליד מהמקום שבו הוא עצר.

אם המשתמש מקליד "#" המשמעות היא שהמשתמש סיים הקלדה עבור משפט זה וצריך לחזור למצב ההתחלתי.

<u>דוגמא לבדיקת תרגיל, ופלט אפשרי:</u>

```
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
```

מתודולוגיית עבודה:

כדי להשתמש בGemini באופן אופטימלי, מומלץ לחלק את העבודה לשני שלבים - בשלב הראשון נחקור את עולם הבעיה באמצעות prompt-ים, ובשלב השני ניעזר בידע ובמושגים הרלוונטיים לנו על מנת לכתוב שאילתות ממוקדות.

לדוגמה, אם ארצה לבקש מ-Gemini לכתוב קוד שמוצא מחרוזת בתוך משפט, אפשר לבקש:

python algorithm that finds a word in a sentence

אבל תוצאה טובה יותר תגיע אם נשתמש במושגים מעולם הבעיה:

python algorithm that finds a substring in a string using regular expressions

הערכת התרגיל

איכות הפתרון שלכם נמדדת לפי שני מדדים עיקריים (metrics):

1) נכונות הפתרון – כלומר האם המשתמש מקבל את ההשלמות הנכונות בעלות ה"ציון"-ים הגבוהים ביותר.
2) יעילות ההשלמה – כלומר כמה זמן לוקח עד שהמערכת מחזירה את ההשלמות.

מעבר לשלב הבא

אין לכם רעיונות נוספים לייעול קוד הפייתון? צרו pull request ל- main עם הגרסה הנוכחית של הקוד, וקראו לצוות המקצועי. ממשו את השלב הבא ב-branch חדש, על בסיס הקוד הקיים.

משימה: שלב ב' – אופטימיזציית זמן ריצה

מבוא

בשלב הראשון של הפרויקט, מימשתם poc להשלמת משפטים ב-python תוך דגש על נכונות ההשלמה. כדי להפוך את הקוד ליעיל עוד יותר, ראש הצוות שלכם מעוניין לבדוק האם שכתוב חלק מהחלקים בקוד (למשל ב- C++) יאפשר ביצועים טובים יותר. זאת כדי לשפר את חווית המשתמש במוצר.

שלבי העבודה:

1. profiling למערכת הקיימת:
   - קראו על profiling לזמן ריצה בpython, ועל כלי פרופיילינג כדוגמת cProfile
   - עברו על קוד ה-python שלכם, ונסו לענות על השאלות הבאות:
     i. איזה חלקים מהקוד משפיעים על חווית המשתמש? איזה לא?
     ii. איפה ישנם צווארי בקבוק בזמני הריצה? (זיהוי של חלקי קוד איטיים במיוחד)
     iii. כיצד תוכלו לראות את צווארי הבקבוק בנוחות?
   - שרטטו גבולות גזרה - איזה חלק מה-poc תרצו לשכתב ב- C++? בבחירתכם, שימו דגש על זמן פיתוח - כיצד תוכלו לפצל את הקוד באופן הכי פשוט?

2. שכתוב ב- C++:
   - לאחר שבחרתם את קטע הקוד, החלו במימוש, תוך תשומת לב לנקודות הבאות:
     i. איך ניתן לחבר את קטע הקוד בפייתון לקטע הקוד ב- C++? בתור התחלה, חשבו על הדרך הקלה ביותר
     ii. אילו אופטימיזציות ניתן לעשות ב- C++ לקטע הקוד שבחרתם?
     iii. כיצד ניתן לחלק את העבודה ביעילות?

3. שימוש ב-Protobuf:

<u>מטרה</u>: להשתמש ב-Protobuf לאחסון יעיל של הנתונים המאפשר גם שימוש בהם בתכניות שנכתבות בשפות אחרות.

<u>פעולות נדרשות:</u>

- תבינו כיצד Protobuf עובד וכיצד הוא יכול לסייע לייעול המערכת שלכם. התעמקו בעיקר במבנה הנתונים שהמערכת שלכם משתמשת בו, וראו כיצד אפשר לסדר אותו בצורה יעילה יותר באמצעות Protobuf.
- כתבו קובץ proto. המתאר את מבנה הנתונים הנדרש, בהתאם לנתונים הקיימים במערכת.
- המירו את מבנה הנתונים הנוכחי (בפייתון) לפורמט של Protobuf. ודאו שניתן לשמור ולטעון את הנתונים בצורה נכונה באמצעות Protobuf.

4. התמודדות עם נתונים גדולים:

מטרה: לבחון כיצד המערכת מתמודדת עם נתונים גדולים ולמצוא פתרונות במידה ויש בעיות.

פעולות נדרשות:

- לאחר שהמערכת עובדת בצורה טובה עם מעט נתונים, המירו כמות נתונים גדולה יותר לפורמט Protobuf ושמרו אותו.
- טענו את הנתונים ב- C++ ובדקו את הביצועים ואת יכולת המערכת להתמודד עם הנתונים הללו.
- במידה ומערכת לא מתמודדת עם הנתונים בצורה טובה, שקלו לחלק את הנתונים למספר קבצים קטנים יותר, להשתמש בגישות שונות לניהול זיכרון, או להיעזר בזרימה (streaming) כדי לטפל בנתונים גדולים.

5. בדיקות ואופטימיזציה נוספת:

- בדקו את זמן הריצה של המערכת המשודרגת בהשוואה למערכת המקורית.
- במידה ויש צורך, בצעו אופטימיזציה נוספת בהתבסס על תוצאות הבדיקות שלכם.

מעבר לשלב הבא

הגעתם לגרסה יעילה יותר מאשר קוד הפייתון מהשלב הקודם?יצרו pull request ל-main עם הגרסה הנוכחית של הקוד, וקראו לצוות המקצועי. ממשו את השלב הבא ב-branch חדש, על בסיס הקוד הקיים.

STARTUP NATION CENTRAL

Excellenteam

משימה: שלב ג' – פתרון מבוסס LLM

עכשיו כשיש לנו תוכנה המאפשרת השלמה אוטומטית ביעילות ובזמן קצר, צוות המוצר שלנו רוצה לשפר את החיפוש של המשתמש כך שיהיה מותאם לכל משתמש לפי ההיסטוריה האישית שלו. הוחלט שנשתמש במודלי ה- LLM ש- google מנגישים כדי לענות על הצורך בצורה מיטבית.

השתמשו במוצרים של Google **בלבד לכלל הסעיפים של שלב ג'.**

שלב ראשון

קראו על Gemini, הבינו איך נכון להתממשק איתו וממשו השלמה אוטומטית המותאמת להיסטוריית החיפוש - חשבו על Prompt יעיל כדי ש-Gemini יוכל לעזור לכם בכך.

<u>הנחיות</u>

- חשבו בנפרד איך שומרים על רלוונטיות ההשלמה ומתייחסים להעדפות המשתמש.
- נהלו דיון וגבשו שלוש גישות למימושים שונים של ה-feature, צוות המוצר הגיע עם דרישה ועליכם להראות היתכנות טכנולוגית למימוש - אתם המובילים הטכנולוגיים שמכירים את הקוד.
- קראו לסגל/צוות להציג את הגישות השונות - חשבו על היתרונות היחסיים של כל גישה לעומת האחרות. כל גישה צריכה להיות מגובה בהוכחת היתכנות (לדוגמה: prompt).
- בסיום ההצגה מול הצוות, אשרו עם צוות המוצר את הגישה הנבחרת והתחילו בעבודה.

.

שלב שני

נרצה להפיץ את התוכנה לכלל המשתמשים. יש לנו מיליון משתמשים. גם אם נניח שכל משתמש עושה חיפוש אחד בלבד, הגענו למיליון בקשות ל-Gemini API - הרבה מעל התקציב שלנו.

ראינו שיש ביקוש רב בקרב חלק גדול מן המשתמשים, ולכן נרצה עכשיו להגיע למצב שבו אנחנו רווחיים (cost-effective). איך הייתם מתמודדים עם הבעיה?

<u>הנחיות:</u>

- נסו להבין את סדר גודל הבקשות שאיתו המוצר שלכם יצטרך להתמודד. מה העלויות - מה עלות בקשה? מה עלות היומית למשתמש? נסו לכמת במספרים.
- זכרו - המטרה שלנו היא להיות רווחיים, לכן ככל שהעלות היומית של משתמשים נמוכה יותר - החברה שלנו רווחית יותר.

- ודאו שיש לכם אופציה לבדוק את היתכנות הרעיונות שלכם באופן יעיל ומהיר.
- שימו לב לאופן המימוש שלכם - מצופה שתוכלו להוסיף שימוש במודלים נוספים מבלי לפגוע במימוש הקיים.

בונוס נסתר

בזמן הרצת הפתרון הזול, האם שמתם לב שמשהו עובד לאט מדי? חשבו על דרכים לפתרון הבעיה וקראו לסגל/צוות להציג אותם.

Base score:

- 2 x the number of case-insensitive matching letters including space but not punctuation.

Incorrect letters in the query string:

- -5 if first letter is incorrect
- -4 if second letter is incorrect
- -3 if third letter is incorrect
- -2 if fourth letter is incorrect
- -1 if any other letter is incorrect

Missing or added letters:

- -10 if first letter is missing or added
- -8 if second letter is missing or added
- -6 if third letter is missing or added
- -4 if fourth letter is missing or added
- -2 if any other letter is missing or added

Examples-

For the sentence: *To be or not to be, that is the question.*

Query	**Score**	**Reason**
To be	10	All 5 letters (including space) match
or Not	12	All 6 letters (including space) match
be, that	14	All 7 letters (including space, ignoring comma) match
2o be	5	Base = 10 -5 for incorrect first letter
to pe	8	Base = 10 -2 for incorrect fourth letter
not be	6	“to “ is missing, should be “not to be” Needs more than one letter correction, so it cannot be a match.
or knot	8	Base = 12 -4 for added fourth letter k