האוניברסיטה העברית בירושלים
בית הספר להנדסה ולמדעי המחשב ע"ש רחל וסלים בנין

סדנת תכנות C ו-++C - תרגיל 3

נושאי התרגיל: #מצביעים #מערכים_דינמיים #קריאה_מקבצים #מחולל_טקסט #ניהול_זיכרון

תאריך הגשה: 05.05.21 בשעה 23:59

1 רקע

עיבוד שפה טבעית - Natural Language Processing - NLP

אחד מהתחומים החשובים ביותר כיום במדעי המחשב הוא התחום של עיבוד שפה טבעית. NLP הוא תחום רחב המנסה לגשר בין שפה טבעית (שבה אנו משתמשים ביום יום) לבין המחשב. דוגמא לאחת מבין המשימות הרבות של NLP הוא לגרום למחשב "להבין" דיבור של אדם (לדוגמת Alexa, Siri, Google assistant וכדומה).

בתרגיל זה נתמקד בנושא נוסף של NLP, שימוש בתוכנת מחשב כדי לייצר משפטים/ציוצים (כינוי להודעות המתפרסמות בפלטפורמת טוויטר) חדשים בעזרת מאגר נתונים קיים. נבצע זאת בעזרת **שרשראות מרקוב** (Markov chains), ועל אופן פעולתן נרחיב בהמשך.

התוכנית שנכתוב תעבוד באופן הבא:

1. נקבל Text corpus, מאגר טקסט גדול, המכיל משפטים רבים.
2. נקרא את ה-corpus ונשמור את המילים הנתונות בו למילון. ניתן משקל לכל מילה.
3. נייצר משפטים באופן הסתברותי. נבחר מילה ראשונה למשפט ונעזר במשקלים משלב 2 כדי להגריל את המילים הבאות במשפט.

הפונקציות rand() ו-srand()

המחשב לא יכול ייצר ערך אקראי אמיתי, לכן מתכנתים נעזרים בפונקציות "**Pseudo-random generator**". פונקציות אלו מקבלות ערך **seed**, שהוא ערך התחלתי כלשהו (אנחנו נשתמש בפונקציה `srand(seed)` כדי לאתחל ערך זה), ובעזרת סדרת פעולות מתמטיות מחשבות ערך שלמראית עין נראה אקראי (בכל קריאה ל-rand() יוחזר ערך שונה, המושפע מערך ה-seed).

בחירת seed:

- במקרה בו לא נבחר seed, הקומפיילר מאתחל ערך זה ל-1, ולכן אם נריץ את התוכנית מספר פעמים ברצף, בכל ריצה נקבל תמיד את אותם ערכים "אקראיים".
- דרך נפוצה לעקוף בעיה זו היא לבחור seed בעזרת השעה בא החלה התוכנית לפעול.
- במקרה שלנו נקבל את ערך ה-seed בתור פרמטר שהתוכנית תקבל בתחילת הריצה.

כדי לקבל מספר אקראי, נעזר בפונקציה `rand()` שתחזיר לנו מספר שלם בין 0 (כולל) ל-RAND_MAX (לא כולל). RAND_MAX הוא פרמטר מוגדר מראש בספרייה stdlib.h.

הערה: שימו לב כי פונקציית rand() לא בטוחה לשימוש עבור מקרים שבהם אבטחה היא קריטית, שכן לאחר שצפינו במספיק ערכים אפשר לנחש באופן מוצלח מה יהיה המספר הבא שהפונקציה תחזיר.

2 פרמטרים וקריאת קובץ טקסט

פונקציית ה-main שלכם תקבל את הארגומנטים הבאים (ובסדר הבא):

1) **ערך seed** – מספר שיינתן לפונקציית ה-()srand בתחילת הריצה (ניתן להניח כי הוא מספר שלם).
2) **כמות הציוצים שנייצר** – ניתן להניח כי הפרמטר הוא מספר שלם וגדול ממש מ-0.
3) **נתיב** (path) **לקובץ ה-Text corpus** – אין להניח כי הנתיב שניתן תקין. במקרה בו הקובץ לא קיים או שלתוכנית אין הרשאות גישה אליו, יש להדפיס הודעת שגיאה מתאימה ל-stdout המתחילה ב-
"Error:"
ולצאת מהתוכנית עם **קוד שגיאה EXIT_FAILURE**.
4) **כמות המילים שיש לקרוא מהקובץ** – במקרה בו לא התקבל פרמטר רביעי יש לקרוא את הקובץ כולו (ניתן להניח כי הפרמטר הוא מספר שלם וגדול ממש מ-0, וכי המילה האחרונה שתקראו תהייה תמיד סוף משפט). אין להניח כי כמות המילים בקובץ הטקסט גדולה או שווה לפרמטר הנתון, גם במקרה זה יש לקרוא את הקובץ כולו.

דוגמא לקריאה תקנית לתוכנה בלינוקס:

```
tweetsGenerator 454545 30 "path-to-file\exmaple.txt" 100
```

במקרה בו כמות הפרמטרים שהתקבלו לא תואמת את הדרישות, יש להדפיס הודעה ל-stdout המתחילה ב-

"Usage:"

ומפרטת בקצרה את הפרמטרים הנדרשים, ולצאת מהתוכנית **עם קוד שגיאה EXIT_FAILURE**.
על הקובץ ממנו תקראו את הציוצים ניתן להניח את ההנחות הבאות:

1) הציוצים יכילו אותיות לועזיות ב-lower-case, מספרים, רווחים וסימני פיסוק סטנדרטים של ASCII בלבד.
2) כל שתי מילים יופרדו ע"י רווח אחד או יותר.
3) התו האחרון בכל שורה **תמיד יהיה התו נקודה '.'** (יתכן שתופיע מילה המסתיימת בנקודה גם באמצע השורה).
4) כל מילה לא תהיה ארוכה מ-MAX_WORD_LENGTH תווים.
5) אורך שורה לא יעבור את MAX_SENTNCE_LENGTH תווים.
<u>הערה</u>: שני ערכים אלו מוגדרים עבורכם בקובץ tweetsGenerator.c.
6) בקובץ יהיה לכל הפחות משפט אחד המכיל לכל הפחות 2 מילים.

לקריאה מהקובץ אנו ממליצים להשתמש ב-fgets() וב-strtok().

3 מבנה הנתונים

עליכם להשתמש במבני הנתונים הבאים שהוגדרו עבורכם **חלקית** בקובץ tweetsGenerator.c:

WordStruct

Struct המכיל:

- מצביע אל תוכן המילה.
- מערך **דינאמי** של WordProbability המכיל את כל המילים העוקבות האפשריות ע"פ הטקסט הנתון.

  הערה: עבור מילה שמסתיימת בנקודה (סוף משפט), המערך WordProbability יצביע ל-NULL.

  הערה 2: מערך זה בד"כ קטן ולכן נשתמש באסטרטגיה הבאה: בכל פעם שנרצה להכניס מילה חדשה למערך, נבצע realloc(), ונגדיל את גודלו ב-1.

- **כל שדה נוסף שתרצו לשימושכם.**

WordProbability

Struct המכיל:

- מצביע אל המילה הבאה האפשרית במשפט.
- פרמטר בעזרתו ניתן לדעת/לחשב את ההסתברות לבחור את מילה זו מבין כל המילים האפשריות.

  הנוסחה בעזרתה נחשב את ההסתברות לבחירת המילה היא:

$$Pr(word_1, word_2) = \frac{\#of\ occurences\ of\ word_2\ immediatly\ after\ word_1}{\#of\ occurences\ of\ word_1\ in\ text\ corpus}$$

  כאשר $word_2$ מופיע מיד אחרי $word_1$ בטקסט.

- **כל שדה נוסף שתרצו לשימושכם.**

WordsDictionary

רשימה מקושרת המכילה את כל המילים הייחודיות בטקסט. שימו לב כי אנו מתייחסים לסימני פיסוק כאל תווים במילה, ז"א המילים:

```
hello
#hello
hello,
hello.
```

הן 4 מילים שונות מבחינתנו, וכל אחת תקבל אובייקט WordStruct ייחודי עבורה במילון.

תוכלו להיעזר **ברשימה המקושרת הנתונה עבורכם** בתחילת הקובץ tweetsGenerator.c.

4 יצירת משפט

- נבחר את המילה הראשונה במשפט בהתפלגות אחידה (לכל מילה יש הסתברות שווה להיבחר) מבין כל המילים במילון **שאינן מילה אחרונה** (אינן מסתיימות בנקודה).
- בעזרת פונקציית ההסתברות נבחר את המילים הבאות במשפט.
- נסיים כשנגיע למילה שמסיימת משפט (מסתיימת בנקודה), או כשאורך המשפט הינו מקסימלי (מספר המילים המקסימלי מוגדר עבורכם כ-MAX_WORDS_IN_SENTENCE_GENERATION בקובץ tweetsGenerator.c).

<u>הערה</u>: במקרה זה אין להוסיף נקודה בסוף המשפט.

דוגמא ליצירת משפט

עיבדנו את הטקסט, מבין המילים במילון בחרנו את "my" המופיעה בטקסט 4 פעמים, ואחריה הופיעו:

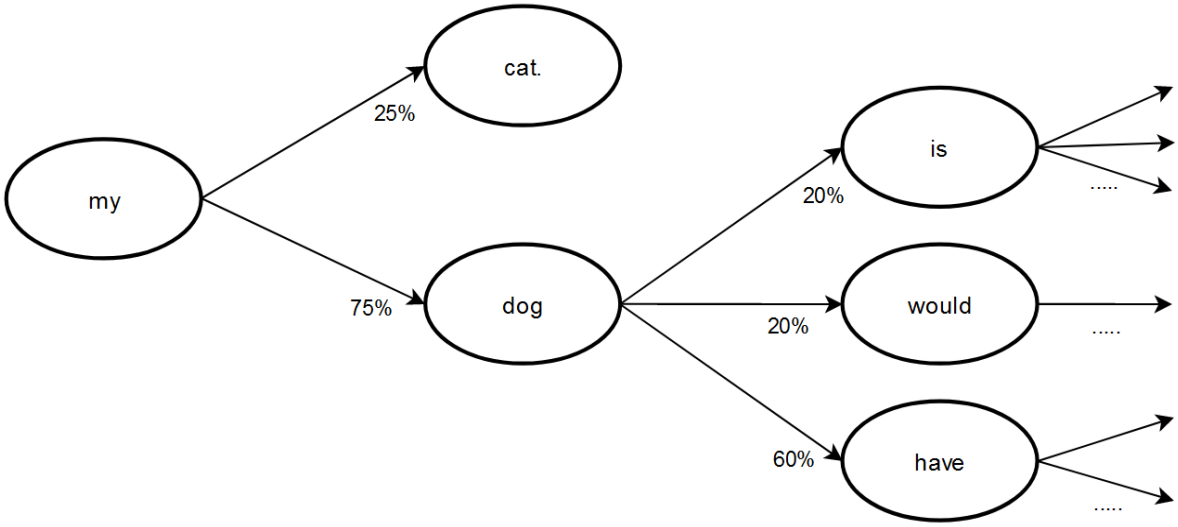
- המילה "dog" 3 פעמים.
- המילה "cat." פעם אחת.

ולכן מבין 2 מילים אלו נבצע הגרלה, שתבחר את המילה הבאה בציוץ. נגיד והגרלנו את המילה "dog". מילה זו הופיעה 10 פעמים בטקסט (לאו דווקא מיד אחרי המילה "my"), ואחריה הופיעו:

- המילה "is" פעמיים.
- המילה "would" פעמיים.
- המילה "have" 6 פעמים.

ולכן באופן דומה נבחר מבין מילים אלו מילה אחת שתהיה בציוץ. נמשיך באופן דומה ונסיים כשנגיע למילה שהיא סוף משפט (לדוגמת המילה "cat.", בה יש תו אחרון נקודה).

אם נצייר את המידע בגרף נקבל:



ניתן להסתכל על כל קודקוד כאל מופע של **WordStruct**, ולכל צלע כמופע של **WordProbability**.
לכל מילה במילון רשימת הסתברויות בלתי תלויה במילים שהגיעו לפניה במשפט. מודל שפועל לפי מערכת הסתברותית שכזו נקרא "שרשרת מרקוב".

במקרה פשוט זה קיבלנו גרף חסר מעגלים, אך עבור קובץ טקסט גדול, יתכן שנקבל גרף בו כן יש מעגלים (לדוגמא אם אחרי המילה "is" בטקסט יש את המילה "my").

5 פונקציות נוספות

בנוסף ל-main, עליכם לממש את הפונקציות הבאות:

- `int get_random_number (int max_number)`
- `WordStruct *get_first_random_word (LinkList *dictionary)`
- `WordStruct *get_next_random_word (WordStruct *word_struct_ptr)`
- `int generate_sentence (LinkList *dictionary)`
- `int add_word_to_probability_list (WordStruct *first_word, WordStruct *second_word)`
- `void fill_dictionary (FILE *fp, int words_to_read, LinkList *dictionary)`
- `void free_dictionary (LinkList *dictionary)`

את התיעוד המלא לכל פונקציה ניתן למצוא בקובץ **tweetsGenerator.c** במודל בו עליכם לכתוב את התוכנית.

אין לשנות את חתימות הפונקציות הנ"ל, אך מותר להוסיף פונקציות נוספות לשימושכם.

בנוסף מצורף עבורכם הקובץ justdoit-tweets.txt, המכיל בסביבות 4400 ציוצים בעזרתו תוכלו לבדוק את תוכנתם. השתדלנו ככל שביכולתנו לנקות את המאגר מכל תוכן פוגעני. אולם, אם קרה החמצנו משהו, אנו מתנצלים על כך מראש.

6 הרצת Presubmit

את בדיקת ה-presubmit תוכלו להריץ באמצעות הפקודה הבאה ב-CLI:

```
~labcc2/presubmit/ex3/run ex3.tar
```

כאשר ex3.tar מכיל את הקובץ tweetsGenerator.c בלבד.

7 דגשים והנחיות לתרגיל

- בסיום הריצה עליכם לשחרר את כלל המשאבים בהם השתמשתם, התוכנית שלכם תיבדק ע"י valgrind ויורדו נקודות במקרה של דליפות זיכרון.
- במקרה של שגיאת הקצאת זיכרון הנגרמה עקב **malloc()/realloc()/calloc()**, יש להדפיס הודעת שגיאה מתאימה ל-stdout המתחילה ב-

  "Allocation failure:"

  ולצאת מהתוכנית באמצעות **קוד שגיאה EXIT_FAILURE**. זהו **המקרה היחיד** בו אין צורך לשחרר זיכרון בסוף ריצת התוכנית.

- שימו לב שבחירת המילים שלכם למשפט באמת מבוצעת ע"פ ההתפלגות המוגדרת בתרגיל ושהיא אכן אקראית.
- אם אפשרי, תעדיפו תמיד לעבוד עם `int/long` מאשר `float/double`. ניתן לפתור את התרגיל כולו בעזרת שימוש במספרים שלמים בלבד.

- כל משפט יודפס בשורה בפני עצמה, **עליכם בתחילת השורה בלבד** להוסיף את הטקסט (שימו לב לרווח לאחר הסוגריים):

`Tweet {iteration number}:` `generated sentence`

לדוגמה, השורה השישית שהתרגיל שלכם ידפיס תראה כך:

`Tweet 6:` `hello, nice to meet you.`

(כמובן שהמשפט שלכם יראה אחרת).

- אין להשתמש ב-**vla**, שימוש שכזה יגרור הורדת ציון מתרגיל.

- כדי לקמפל את התוכנית תוכלו להיעזר בפקודה הבאה:

```
gcc -Wall -Wextra -Wvla -std=c99 tweetsGenerator.c -o tweetsGenerator
```

8 נהלי הגשה

- קראו בקפידה את הוראות תרגיל זה. זהו תרגיל מורכב ולכן אנו ממליצים להתחיל לעבוד עליו כמה שיותר מוקדם. כמו כן, זכרו כי התרגיל מוגש ביחידים, ואנו רואים העתקות בחומרה רבה!
- יש להגיש את התרגיל באמצעות ה-git האוניברסיטאי ע"פ הנהלים במודל.
- יש להגיש את הקובץ **`tweetsGenerator.c`** בלבד (ניתן להשאיר את קובץ README ב-repo).
- כחלק מהבדיקה האוטומטית תיבדקו על סגנון כתיבה.
- התרגיל נבדק על מחשבי האוניברסיטה, ולכן עליכם לבדוק כי הפתרון שלכם רץ ועובד גם במחשבים אלו.
- כשלון בקומפילציה או ב-presubmit יגרור ציון 0 בתרגיל.