מבוא למדעי המחשב 67101 - סמסטר א' 2024

תרגיל 6: Moogle - חיפוש ודירוג דפי אינטרנט

להגשה בתאריך: **21/02/2024** בשעה 22:00

הקדמה

בתרגיל זה נכתוב מעין מנוע חיפוש שיאפשר חיפוש באוסף דפי אינטרנט. נעשה זאת על ידי הורדת דפים מהרשת ומימוש אלגוריתם ה-Page Rank המפורסם של גוגל. לאורך כל התרגיל נעבוד עם ספריות לחיפוש וניתוח תוכן בדפי אינטרנט, וכן נממש שיטות לדירוג אתרי אינטרנט לפי חשיבות, וכן חיפוש בהם.

דגשים לתרגיל

- אין להשתמש בספריות **numpy, scipy, pandas** וכל ספריה אחרת, ללא אישור מפורש בפורום.
- התוכנית תופעל דרך הקובץ **moogle.py**, כפי שיפורט בהמשך. מומלץ לפרק את התוכנית לקבצים נוספים ולהגישם גם כן. את החלוקה לקבצים ולפונקציות עליכם לבצע על פי שיקול דעתכם, לפי העקרונות שנלמדו בקורס. הדגש צריך להיות על קוד מודולרי (ללא כפל קוד), ברור וקריא.
- כל הקבצים יוגשו בתוך קובץ **ex6.zip**. ניתן להניח שכל הקבצים המוגשים בקובץ זה יימצאו בתיקיה בעת הקריאה לסקריפט. שימו לב לא להגיש קבצים מיותרים (בפרט את קבצי הסביבה הווירטואלית או קבצי קלט/פלט)
- סגנון: הקפידו על תיעוד נאות ובחרו שמות משתנים משמעותיים. הקפידו להשתמש בקבועים (שמות משתנים באותיות גדולות) על פי הצורך.
- **קראו את הרקע לHTML וצפו בסרטון הנלווה לתרגיל לפני שתתחילו במימוש התרגיל**
- לאורך כל התרגיל ניתן להניח **שהקלט תקין אלא אם כן נאמר אחרת במפורש**, כלומר:
  - לא יינתנו קישורים שאינם קיימים או תקינים.
  - לא יינתנו קישורים יחסיים שהחיבור שלהם עם קישור ה- Base URL ייתן קישור שלא קיים.
  - כל הארגומנטים שיינתנו לתתי התוכניות יהיו חוקיים (ארגומנט שצריך להיות int יהיה int, מספר הארגומנטים יהיה כנדרש וכו').
  - **שימו לב: ייתכנו מילונים ריקים (כלומר קבצי pickle שלא מכילים למעשה כלום) - עליכם להתמודד עם זה!**
- שימו לב שכל תתי התוכניות צריכות להסתיים בהצלחה, וללא שגיאות.
- שימו לב שכל תתי התוכניות לא צריכות להחזיר שום ערך!
- **מומלץ להריץ את הקוד בסביבה וירטואלית, המכילה את הספריות הנדרשות בתרגיל (requests, bs4).**

ניתן להתקין את הספריות בעזרת `pip install requests bs4`

- **מותר (אך לא חובה) להשתמש בספריות Collections ו-argparse**

רקע: אלגוריתם Page Rank, שפת HTML וחבילת BeautifulSoup4

אלגוריתם Page Rank

בתרגיל זה נממש את הגרסא הפשוטה של אלגוריתם Page Rank המפורסם ושל מנוע החיפוש, שהוצג לעולם בשנת 1998 על-ידי המוגלגים לארי פייג' וסרגיי ברין, בראשיתה של חברת גוגל.
האלגוריתם מדרג דפי אינטרנט הרלוונטים ביותר לשאילתת חיפוש מסוימת. האלגוריתם מחשב עבור כל דף אינטרנט את מידת החשיבות שלו. דירוג של דף נקבע על פי כמות הדפים המקשרים אליו וחשיבותם של הדפים המקשרים. כלומר, אם דפים רבים מקשרים אל דף מסוים, האלגוריתם קובע את מידת החשיבות של דפים אלה ומדרג את הדף על פי מידת חשיבותם. כמו כן, האלגוריתם לוקח בחשבון את שאילתת החיפוש ומספר המופעים שלה בדף. ככל שמילות החיפוש מופיעות יותר בדף האינטרנט, כך החשיבות שלו עולה והוא עולה בדירוג הדפים.

שפת HTML

HTML היא השפה העיקרית בה משתמשים כיום לתצוגה ולעיצוב של דפי אינטרנט. השפה היא מבוססת תגיות, כאשר לכל תגית תפקיד עיצובי אחר בעמוד. בקוד HTML, החלקים העיקריים של כל תגית לרוב הם שם, תכונות, ותוכן, והמבנה התחבירי הכללי נראה כך:

```
<tag prop1=val1, prop2=val2, … prop=valN> content </tag>
```

כאשר:

- tag – שם התגית
- prop1,…,propN – התכונות של התגית
- val1,…,valN – ערכי התכונות בהתאמה
- content – התוכן של התגית

שמות התגיות, וכן התכונות של כל תגית, הם לרוב מוגדרים מראש. תגיות יכולות אף להיות מקוננות, כלומר חלק התוכן של כל תגית יכול להכיל תגיות פנימיות נוספות. לצורך התרגיל, אנו נתמקד ב-2 תגיות שמעוניינות אותנו בלבד:

- התגית text המייצגת פסקה של טקסט
- התגית link המשמשת לצורך הגדרת קישורים (לינקים), כאשר href הוא התכונה שהערך שלה "url" מכיל את כתובת האתר אליו הלינק מקשר, ו-link הוא הטקסט המוצג למשתמש שישמש כלינק עצמו.

קוד HTML שמור לרוב בקבצי HTML, בעלי הסיומת html.

ספרית BeatuifulSoup4

בפייתון קיימת ספריה בשם BeautifulSoup4 שמקלה מאוד על העבודה עם קבצי HTML. היא מאפשרת לגשת לתגיות השונות של קוד HTML. ראשית יש לייבא אותה בראש הקובץ כך:

```
import bs4
```

לאחר מכן, כדי להשתמש בספריה כדי לגשת לתגיות השונות של קוד HTML כלשהו, יש ליצור אובייקט BeautifulSoup באופן הבא:

```
soup = bs4.BeautifulSoup(my_html)
```

כאשר my_html הוא משתנה מטיפוס מחרוזת, המכיל את כל קוד ה-HTML של קובץ כלשהו. לאחר מכן תוכלו למצוא את כל התגיות מסוג מסוים בעזרת הפונקציה **find_all**. למשל, כדי למצוא את כל תגיות הפסקאות , יש לכתוב, לאחר שורת הקוד הקודמת, את השורה הבאה:

```
paragraphs = soup.find_all("p")
```

על המשתנה paragraphs, שמכיל את כל הפסקאות שבקוד השמור במשתנה html, אפשר לעבור בעזרת לולאת for, על כל פסקה, ולמצוא בה, למשל, את כל תגיות <a> המוכלות בה, כך:

```
for p in paragraphs:
    links = p.find_all("a")
```

בדוגמה זו, עבור כל פסקה (השמורה במשתנה הלולאה p), נקבל את רשימת כל התגיות a המוכלות בפסקה לתוך המשתנה links, שגם עליו אפשר לעבור בלולאה פנימית עבור כל תגית <a> בנפרד.

פונקציה נוספת, שיכולה להיות שימושית עבורכם, היא הפונקציה **get** המאפשרת לקבל את הערך של תכונה מסוימת של תגית. אם למשל l הוא משתנה המייצגת תגית <a> כלשהי (למשל אחד האיברים של המשתנה links מהדוגמה הקודמת), אפשר לגשת לערך התכונה href שלו כך:

```
target = l.get("href")
```

תוכלו לקרוא עוד על הספריה BeautifulSoup4 בלינק הבא:

https://www.crummy.com/software/BeautifulSoup/bs4/doc/

תיאור התרגיל

בתרגיל זה אתם מתבקשים לבנות מנוע חיפוש בסיסי, הנקרא **Moogle**, בו ניתן לחפש ביטויים שונים בתוך אוסף דפים מתוך הויקי של הארי פוטר (ניתן לעיין בדפי המקור פה). היות והויקי המלא הוא מאוד גדול יצרנו עבורכם קבוצה קטנה יותר של דפים אליהם ניתן לגשת בכתובת:
https://www.cs.huji.ac.il/w~intro2cs1/ex6/wiki/
כדי לגשת לדף מסויים בתוך הכתובת עליכם לגשת ל:
https://www.cs.huji.ac.il/w~intro2cs1/ex6/wiki/<some_page_name>.html
כאשר עליכם להחליף את <some_page_name> בשם של הדף המסויים.
אם, לדוגמא, אתם רוצים לגשת לדף של Ronald Weasley עליכם להשתמש ב:
https://www.cs.huji.ac.il/w~intro2cs1/ex6/wiki/Ronald_Weasley.html

שימו לב: לדפי ה-wiki ניתן לגשת דרך מחשבי האקוריום או דרך התחברות מרחוק. אם ברצונכם לגשת מהמחשב האישי תוכלו להשתמש בuser_agent:

```
header = {'User-Agent': str(ua.chrome)}
my_url = "Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101
Firefox/115.0"
response = requests.get(my_url, headers=header)
```

תיאור התרגיל כולל מספר חלקים:

- חלק (א) - הורדת הדפים מהאינטרנט וגילוי הקישורים בינם
- חלק (ב) - דירוג אוסף הדפים לפי חשיבות
- חלק (ג) - יצירת מיפוי ממילים לדפים בהם הן מופיעות
- חלק (ד) - ביצוע חיפושים של מילים בדפים, על בסיס החלקים הקודמים

- כל חלק בתרגיל ירוץ עם ארגומנטים משורת הפקודה (command line arguments) כאשר פקודת ההרצה תיראה כך:

```
python3 moogle.py <task_name> <arg1> <arg2> ...
```

כלומר, הקובץ moogle.py ירוץ כל פעם עם **task_name** אחר בהתאם שבחרנו להריץ מבין 4 החלקים של התרגיל, ויקבל ארגומנטים נוספים כפי שמפורט בתיאור של כל אחד מהחלקים.

חלק (א) - גישה אל הדפים דרך הרשת וגילוי הקישורים בינם

בחלק הראשון של התרגיל, ניצור מילון המפרט את הקישורים בין הדפים אותם אנו סורקים. כלומר, מילון שיפרט עבור כל דף אינטרנט את הדפים שהוא מצביע עליהם ואת כמות ההצבעות על כל אחד מהדפים.

כדי לבנות את מילון הקישורים, נקרא לתוכנית **moogle.py** משורת הפקודה באופן הבא:

```
python3 moogle.py crawl <BASE_URL> <INDEX_FILE> <OUT_FILE>
```

כאשר:

- **<BASE_URL>** היא כתובת אתר האינטרנט בו נמצאים הדפים
- **<INDEX_FILE>** הינו שם של קובץ טקסט אשר מכיל רשימת דפים בתוך האתר הנתון **Harry_Potter.html**, **Albus_Dumbledore.html** ועוד.
- **<OUT_FILE>** הינו שמו של קובץ הפלט בו תשמרו את מילון הקישורים.

לדוגמא הפקודה:

```
python3 moogle.py crawl https://www.cs.huji.ac.il/w~intro2cs1/ex6/wiki/
small_index.txt out.pickle
```

תאסוף קישורים מהאתר של מדעי המחשב (הדפים שמופיעים בקובץ `small_index.txt`) ותשמור את התוצאה בקובץ `out.pickle` **(בהמשך מוסבר איך שומרים לקובץ pickle)**

לנוחיותכם, סיפקנו לכם קובץ בשם `small_index.txt` המכיל רשימה של 12 דפי אינטרנט שונים בוויקי שלנו. הקישורים נתונים ככתובות יחסיות. למשל, הכתובת היחסית באתר מדעי המחשב בה נמצא דף הויקי של הארי פוטר היא `Harry_Potter.html` בעוד הכתובת המלאה של דף זה היא:

```
https://www.cs.huji.ac.il/w~intro2cs1/ex6/wiki/Harry_Potter.html
```

<u>מילון הקישורים</u>

מילון הקישורים שלנו יהיה בעל המבנה הבא: ניצור מבנה נתונים traffic_dict שבעזרתו נוכל לגלות כמה קישורים קיימים מדף אינטרנט ששמו page_name לכל אחד מדפי האינטרנט האחרים בויקי.

נוכל לעשות זאת באמצעות שימוש במילונים של פייתון. הטיפוס של traffic_dict יהיה:

```
traffic_dict: Dict[str, Dict[str, int]]
```

כלומר מילון הממפה מחרוזת למילון.

נרצה ש:

```
traffic_dict[page_name][linked_page_name]
```

יתן לנו את מספר הפעמים שהדף האינטרנט ששמו page_name מכיל קישורים לדף ששמו linked_page_name.

למשל:

```
traffic_dict["Harry_Potter.html"]["Tom_Riddle.html”]
```

יהיה מספר הקישורים שקיימים בדף אינטרנט ששמו "Harry_Potter.html" לדף אינטרנט ששמו "Tom_Riddle.html".

הערות:

- קישור מדף A לדף B, הוא לינק המצוין על-ידי תגית הקישור <a>, בקוד ה-HTML של דף A. בתוך התגית מופיע הערך href שאליו משויכת הכתובת של דף B. לדוגמא:

  ```
  <a href="Tom_Riddle.html">Lord Voldemort</a>
  ```

  הוא קישור לעמוד הויקי של לורד וולדמורט.

- דף יכול להצביע לעצמו.
- יש לקחת בחשבון כפילויות. דף כלשהו יכול לכלול יותר מקישור בודד לכל יעד.

- אנחנו נמנה במילון אך ורק דפים ששמותיהם מופיעים בקובץ <INDEX_FILE>. יש להתייחס **רק לקישורים יחסיים** (ולא לקישורים אבסולוטים) באותו אתר.
- בקובץ <INDEX_FILE> כל כתובת מופיעה בשורה נפרדת וניתן להניח שהקובץ מתקבל בפורמט תקין.
- ניתן להניח שלכל דף יש לפחות קישור יוצא אחד (רלוונטי לדירוג האתרים בהמשך התרגיל).
- ישנם מקרים בהם הערך של href יהיה מחרוזת ריקה ("")". הדבר אומר שתגית הקישור במקרה זה אינה מקשרת לאף דף וניתן להתעלם ממנה
- כדי לחבר יחד כתובת של אתר (BASE_URL) וכתובת יחסית יש לכתוב:

```
import urllib.parse
full_url = urllib.parse.urljoin(base_url, relative_url)
```

הנחיות נוספות:

- **שימו לב:** הגישה לקריאה של דפי הHTML תתאפשר דרך מחשבי האוניברסיטה/התחברות מרחוק. אם ברצונכם להריץ מהמחשב

- כדי לקבל את קוד ה-HTML של דף אינטרנט מסוים, עליכם להגיש בקשת HTTP לשרת האינטרנט המכיל את הדף. לשם כך תוכלו להשתמש בספריה **requests** באופן הבא:

```
response = requests.get(url_path)
my_html = response.text
```

כאשר url_path הוא משתנה המכיל את כתובת האינטרנט המלאה לדף האינטרנט שאנו מעוניינים לקבל את קוד ה-HTML שלו. לתוך המשתנה my_html יכנס קוד ה-HTML עצמו, והוא יהיה משתנה מטיפוס מחרוזת (str).

- אם למשל המשתנה my_html מכיל את קוד HTML של דף אינטרנט מסויים, תוכלו לעבור על כל הפסקאות והלינקים שבתוך פסקאות בעזרת הפונקציה find_all() ולקבל את ערך התכונה href של כל לינק, כך:

```
soup = bs4.BeautifulSoup(my_html, 'html.parser')
for p in soup.find_all("p"):
      for link in p.find_all("a"):
              target = link.get("href")
```

בדוגמה זו, בכל איטרציה של הלולאה הפנימית, המשתנה target יקבל את ערך התכונה href של הלינק, כלומר את כתובת הקישור.

שמירת המילון

על התוכנית לשמור את מילון הקישורים שיצרתם. אנחנו נשמור את הקובץ בפורמט של קובץ **pickle**, תחת השם המתקבל בפרמטר **<OUT_FILE>** משורת הפקודה. שמירה בפורמט זה (ולא כקובץ טקסט) מאפשרת לשמור את המילון בצורה קומפקטית וקלה. תוכלו לקרוא עוד על פורמט **pickle** פה.

ראשית נייבא את המודול **pickle** כך:

```
import pickle
```

כדי לשמור את המילון בקובץ **pickle** יש צורך לפתוח אותו במצב כתיבה בינארית, ולאחר מכן לקרוא לפונקציה pickle.dump() כך:

```
with open(filename, 'wb') as f:
      pickle.dump(d, f)
```

בדוגמה זו המשתנה filename מכיל את שם קובץ הפלט שאליו יישמר המילון, והמשתנה d מכיל את המילון שברצונכם לשמור.

לסיכום: נעבור על קוד ה-HTML של כל אחד מהדפים בקובץ small_index, נמנה את כמות הקישורים (היחסיים) שיש לו לכל אחד מהדפים האחרים בקובץ, ונכניס את המידע הזה למילון traffic_dict אותו נשמור בסוף הריצה בקובץ הpickle (עם השם <out_file> שקיבלנו בשורת הפקודה.

חלק (ב) - דירוג דפי אינטרנט לפי חשיבות

בחלק השני, נעשה שימוש במילון הקישורים כדי לדרג את דפי האינטרנט שלנו. הדירוג של האתרים ייעשה על ידי גרסה פשוטה של אלגוריתם Page Rank של גוגל, אשר משמש לדירוג אתרים לפי מספר "הצבעות" יחסי, כלומר: ככל שדף אינטרנט X הינו דף שמצביעים עליו יותר אתרים אחרים, כך הדירוג של X יהיה גבוה יותר ביחס לשאר הדפים.

הדירוג ישמר במילון של פייתון, כאשר כל מפתח הוא שם של דף אינטרנט, כפי שהופיע בקובץ <INDEX_FILE> שהתקבל כפרמטר בשורת הפקודה בחלק א'. הערך של כל מפתח יכיל את דירוג ה-Page Rank של אותו דף. כדי ליצור את מילון הדירוגים, נקרא לתוכנית **moogle.py** משורת הפקודה באופן הבא:

```
python3 moogle.py page_rank <ITERATIONS> <DICT_FILE> <OUT_FILE>
```

כאשר:

- <ITERATIONS> הוא מספר שלם אי-שלילי המתאר את מספר האיטרציות להרצת אלגוריתם ה-Page Rank, כפי שיפורט בהמשך.
- <DICT_FILE> הוא שמו של קובץ pickle המכיל את מילון הקישורים באותו פורמט שיצרתם בחלק א'
- <OUT_FILE> הוא שמו של קובץ הפלט שבו ישמר מילון הדירוגים, גם כן בפורמט pickle.

שלב 1 - יצירת דירוג לדפי האינטרנט

כעת, אנחנו נתחיל ביצירה של מדרג חשיבות בין אתרי האינטרנט השונים, באוסף הדפים שהורדנו. עבור N דפי אינטרנט, אנחנו ניצור מילון **r** בעל N רשומות, ונעדכן בכל איטרציה את ערכיו של המילון **r**, שהינם מטיפוס float. הטיפוס של המילון r יהיה:

```
r: Dict[str, float]
```

מספר האיטרציות שבהם נעדכן את ערכי **r** הינו הערך <ITERATIONS>, המתקבל כפרמטר בשורת הפקודה בעת הפעלת התוכנית.

אופן החישוב של מילון הדירוגים **r** ייעשה בשלבים, לפי התיאור (אלגוריתם) הבא:

1. תחילה, כל דף אינטרנט יקבל דירוג שווה בעל הערך 1.
2. **בכל** איטרציה (סיבוב) נעדכן את המילון **r** באופן הבא:
   - ניצור מילון בשם **new_r** שבו הדירוג של כל דף יתחיל כ-0.
   - כל דף יחלק את הדירוג שקיבל בסיבוב הקודם (בסיבוב הראשון זה 1, ובהמשך ערך אי-שלילי אחר) בין כלל הדפים שעליהם הוא מצביע.
   - כלומר, עבור דף **i** שמצביע לדף **j** כלשהו, הדף **j** יקבל מהדף **i** את הערך הבא:

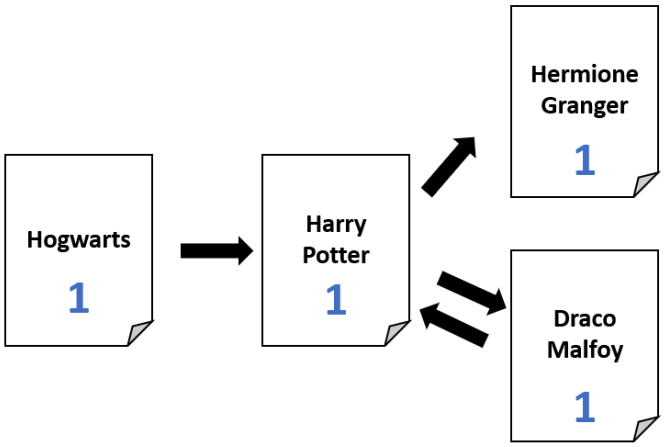
$$new\_r[j] += r[i] \times \frac{<num\ links\ i \to j>}{<total\ num\ links\ from\ i>}$$

שימו לב שהערכים בשבר הנ"ל ניתנים לחישוב בקלות ממילון הקישורים בין הדפים.

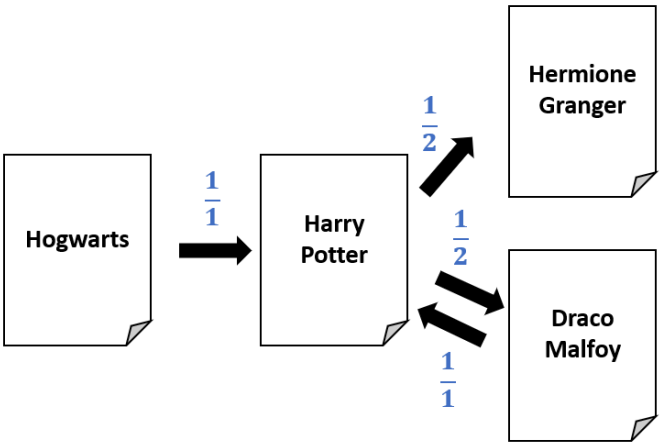
   - בסוף כל איטרציה נבצע השמה של המילון **new_r** אל **r** (כך שלמעשה המילון "החדש" יחליף את המילון "הישן").

כל דף מקבל למעשה את סכום הערכים שתרמו לו הדפים שמצביעים עליו (מהדירוג שלהם בסיבוב הקודם). שימו לב שהערך של כל דף יכול לקטון \ לגדול \ לא להשתנות בכל סיבוב, כתלות בערכים של הדפים האחרים, אך הינו תמיד אי-שלילי.

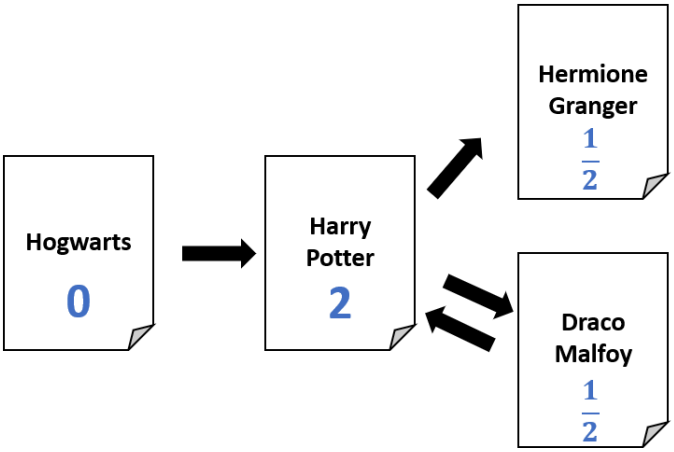
למשל, הסקיצה הבאה מראה דף במרכז, ששמו **Harry Potter** אשר כולל שתי הצבעות יוצאות: הצבעה אחת לדף **Hermione Granger**, והצבעה אחת לדף **Draco Malfoy**, וכן שתי הצבעות נכנסות מהדפים **Hogwarts** ו-**Draco Malfoy** .



בתחילת האיטרציה הראשונה לדף **Harry Potter** יהיה ערך **1**, והוא יעניק $(1 * \frac{1}{2})$ *לדף* **Hermione Granger**, *ו*יעניק $(1 * \frac{1}{2})$ *לדף* **Draco Malfoy**. כמו כן, הדף **Harry Potter** הינו בעצמו דף שיקבל דירוגים מהדפים אשר מצביעים אליו. *בדוגמה הזו, הדף* **Harry Potter** *יקבל* $(1 * \frac{1}{1})$ מהדף **Hogwarts**, ו- $(1 * \frac{1}{1})$ מהדף **Draco Malfoy** . הציור הבא ממחיש את הענקת הדירוגים של הדף **Harry Potter** לדפים אחרים, ושל הדפים האחרים אליו:



בסיום האיטרציה הראשונה, מצב הערכים של כל דף בדוגמה הנ"ל יהיה כדלקמן:



לאחר ביצוע העדכון כמספר האיטרציות שמתקבל בפרמטר <ITERATIONS>, אנחנו נקבל לבסוף מילון r, אשר אמור לקיים:

- מכיל ערכים אי שליליים.
- מכיל ערכים נסכמים לערך שהינו נומרית קרוב מאוד ל-N, מספר הרשומות במילון הקישורים (תחשבו למה!)

מילון זה ייצג לנו את הדירוגים הפנימיים של N האתרים השונים - ככל שאתר יותר חשוב, ומצביעים אליו יותר אתרים אחרים, כך הוא צפוי לקבל ערך גבוה יחסית, ואילו אתרים אשר הינם פחות חשובים, ומצביעים אליהם פחות אתרים אחרים - ידורגו עם ערך נמוך יותר (קרוב ל-0).

שימו לב, שעבור מספר גבוה של איטרציות, אנחנו צפויים לראות התכנסות של ערכי המילון **r** למילון עם ערכים (כמעט) קבועים, כלומר המילון **r** יכלול ערכים שלא ישתנו אחרי איטרציית עדכון.

הנחיות נוספות:

- שימו לב לנוסחא לחישוב ערך ה- page rank של כל דף ולפסאודו-קוד של חישוב ה- page rank של כל הדפים בכל איטרציה!
- ניתן להניח שמספר האיטרציות שיתקבל יהיה תמיד מספר שלם אי שלילי.
- אם מספר האיטרציות שווה ל- 0 עליכם להחזיר מילון שבו ערך ה- page rank של כל דף הוא הערך ההתחלתי (שהוא 1).
- על התוכנית שלכם ליצור מילון דירוגים בהינתן מילון הקישורים, השמור בפורמט **pickle** שנוצר בחלק א'. שם קובץ ה-pickle המכיל את מילון הקישורים מתקבל בפרמטר <DICT_FILE> הניתן בשורת הפקודה בהרצת התוכנית.
- כדי לטעון את מילון הקישורים מתוך קובץ **pickle** למילון של פייתון, עליכם לפתוח את הקובץ במצב קריאה בינארית, ולטעון את תוכן הקובץ בעזרת הפונקציה pickle.load, כפי שניתן לראות בקוד הבא:

```
with open(filename, "rb") as f:
    d = pickle.load(f)
```

בדוגמה זו המשתנה filename מכיל את שם קובץ הקלט שממנו נקרא את המילון, ולמשתנה d יכנס המילון שברצונכם לקרוא.

- שימו לב כי ייתכן ומילון הקישורים שנטען יהיה ריק. יש לטפל גם במקרה זה.

<u>שמירת מילון הדירוגים</u>

לבסוף, על התוכנית לשמור את תוצאות מילון הדירוגים **r**, בסוף ריצת האלגוריתם. יש לשמור את המילון לקובץ בפורמט **pickle** גם כן, בדומה לחלק א'. שם קובץ הפלט ניתן בפרמטר <OUT_FILE> משורת הפקודה.

חלק (ג) - יצירת מיפוי ממילים לדפים

כדי לחפש במנוע החיפוש שלנו, נרצה לדעת עבור מילים שיופיעו בשאילתת החיפוש אילו דפי אינטרנט רלוונטיים לחיפוש. ניצור מבנה נתונים word_dict (שמבנהו דומה למבנה מילון הקישורים שתיארנו בחלק א') שבעזרתו נוכל לגלות כמה פעמים מילה word הופיעה בדף ששמו page_name.
נוכל לעשות זאת באמצעות שימוש במילונים. הטיפוס של word_dict יהיה:

```
word_dict: Dict[str, Dict[str, int]]
```

כלומר מילון הממפה מחרוזת, למילון.
נרצה ש:

```
word_dict[word][page_name]
```

יתן לנו את מספר הפעמים שהמילה word מופיעה בדף ששמו page_name.
למשל:
`word_dict["Harry"]["Ronald_Weasley.html"]` יהיה מספר המופעים של המילה "Harry" בדף ששמו "Ronald_Weasley.html" ברשימת הדפים שלנו.
שימו לב: היות ורוב המילים לא מופיעות בהרבה דפים, אין צורך להכניס רשומות 0 למילון. כלומר: אם אין רשומה במילון עבור מילה מסוימת (או אין רשומה לדף מסוים במילון הפנימי) פירוש הדבר שהמילה אינה מופיעה כלל בדפים או אינה מופיעה כלל בדף הספציפי.

כדי לבנות את מילון המילים, נקרא לתוכנית **moogle.py** משורת הפקודה באופן הבא:

```
python3 moogle.py words_dict <BASE_URL> <INDEX_FILE> <OUT_FILE>
```

כאשר:

- `<BASE_URL>` היא כתובת האתר בו נמצאים הדפים שיש לגשת אליהם
- `<INDEX_FILE>` הינו קובץ טקסט אשר מכיל רשימת שמות של דפי אינטרנט שונים (בדומה לחלק א')
- `<OUT_FILE>` הינו נתיב (path) לקובץ פלט בו תשמרו את מילון המילים, בפורמט **pickle**.

נייצר את מילון המילים על-ידי מעבר על כל פסקאות הטקסט ברשימת הדפים שבקובץ `<INDEX_FILE>`.
בשלב השני נשמור את המילון לקובץ **pickle**, בדומה לחלקים הקודמים.

יצירת המילון

עליכם לגשת, בעזרת בקשת HTTP, לכל דף ברשימת דפי האינטרנט המתקבלים בפרמטר `<INDEX_FILE>` משורת הפקודה, ולעבור על כל הפסקאות שמכיל כל דף אינטרנט. מכל פסקה נוציא את כל רשימת המילים המופיעות בה, ונעדכן בהתאם את מילון המילים לפי ההנחיות הבאות:

- עליכם לגשת לדפים ששמותיהם מופיעים בקובץ `<INDEX_FILE>` דרך בקשות HTTP בעזרת הספרייה **requests** (בדומה לחלק א').
- מכל אחד מהדפים, עליכם לאסוף את כל המילים המופיעים בתוכן של תגיות הפסקה בלבד. השתמשו בספריה **bs4** בפונקציה **bs4.find_all()** כדי למצוא את כל תגיות בכל מסמך. על כל אובייקט p של תגית הקיימת במסמך, ניתן לגשת לערך p.text שמכיל את התוכן של התגית. למשל, בקטע הקוד הבא:

```
for p in soup.find_all("p"):
    content = p.text
```

הלולאה עוברת על כל תגית פסקה הקיימת במסמך, והמשתנה content יכיל את התוכן של אותה פסקה.

- המילים בטקסט מופרדות ע"י whitespaces.
שימו לב ש- whitespaces כוללים גם רווחים, גם ירידות שורה וגם טאבים - כלומר עליכם לפצל את המילים בטקסט על פי כל אחד מהנ"ל (**ולא על פי רווחים בלבד**).
(העזרו ב-**str.split()** לצורך ההפרדה למילים).

- **שימו לב** שמחרוזת ריקה אינה (!) מהווה מילה חוקית ולכן **אינה** צריכה להיספר. וודאו כי אתם לא סופרים מחרוזות ריקות.
- אין צורך לסנן את המילים בתוך הפסקאות. ייתכן שיופיעו בהן תגיות HTML וכל מיני דברים מוזרים אחרים. זה לא יפריע לחיפוש מילים תקינות.
- מילה שנכתבת ב-2 צורות שונות עם אותיות גדולות וקטנות תחשב ל-2 מילים נפרדות. למשל המילים "HARRY", "Harry" ו-"harry" יחשבו ל-3 מילים שונות במילון.
- כמו כן, מילים שאליהן צמודים סימני פיסוק יחשבו כמילים שונות מאלו שללא סימני פיסוק. למשל המילים "harry" ו-"harry," יחשבו כמילים שונות.
- בקובץ <INDEX_FILE> כל כתובת מופיעה בשורה נפרדת וניתן להניח שהקובץ מתקבל בפורמט תקין.

שמירת מילון המילים

לבסוף, על התוכנית לשמור את מילון המילים שיצרתם לקובץ פלט. אנחנו נשמור את המילון לקובץ בפורמט **pickle** גם כן, בדומה לחלקים הקודמים. שם קובץ הפלט ניתן בפרמטר <OUT_FILE> משורת הפקודה.

חלק (ד) - ביצוע חיפוש ודירוג התוצאות

בחלק זה, אנחנו נתבסס על קובץ מילון הדירוגים שיצרתם בסוף חלק ב', וכן על מילון המילים שיצרתם בחלק ג', ונשתמש בהם כדי ליצור את מנוע החיפוש שלנו - Moogle.

שאילתת חיפוש תופעל משורת הפקודה. יתבצע סינון ודירוג של התוצאות, ולבסוף תתבצע הדפסה של תוצאות החיפוש.

הרצת הקובץ משורת הפקודות

את מנוע החיפוש נריץ משורת הפקודה עם הפרמטרים הבאים:

```
python moogle.py search <QUERY> <RANKING_DICT_FILE>
<<WORDS_DICT_FILE> <MAX_RESULTS
```

כאשר:

- <QUERY> - שאילתת החיפוש, המתקבלת בפורמט שיפורט בהמשך.
- <RANKING_DICT_FILE> - נתיב (path) במערכת הקבצים לקובץ מילון הדירוגים, כפי שיצרתם ושמרתם בחלק (ב).
- <WORDS_DICT_FILE> - נתיב (path) במערכת הקבצים לקובץ מילון המילים, כפי שיצרתם ושמרתם בחלק (ג).
- <MAX_RESULTS> - מספר טבעי המייצג לנו את מספר התוצאות המקסימאלי עבור החיפוש, כפי שיפורט בהמשך.

שאילתת החיפוש, המתקבלת בפרמטר <QUERY> הינה מחרוזת של מספר מילים בודדות, מופרדות בתו רווח בודד. על מנוע החיפוש שלכם להציג תוצאות חיפוש אשר יכילו את **כל** המילים של שאילתת החיפוש.

סינון התוצאות לפי ציון משוקלל, והדפסת התוצאות עבור החיפוש

כעת, נשתמש בשלושת הארגומנטים הנוספים שאיתם קראנו לתוכנית משורת הפקודות, <WORDS_DICT_FILE>, <RANKING_DICT_FILE> ו-<MAX_RESULTS> על מנת לדרג את התוצאות למשתמש.

ראשית, נבחר את <MAX_RESULTS> הדפים הראשונים מתוך ויקי הארי פוטר שמכילים את **כל** מילות החיפוש, ממוינים לפי הציון הניתן להם במילון הדירוגים שהתקבל בפרמטר <RANKING_DICT_FILE>. במידה ויש פחות מ- <MAX_RESULTS> תוצאות, אז יש לבחור רק את תוצאות אלה.

לאחר שביצענו את בחירת התוצאות, יש להדפיס את שמות הדפים שבהם נמצאו התוצאות למשתמש, לפי דירוג משוקלל, אשר מבוסס גם על ערך החשיבות של כל דף (במילון הדירוגים ב-<RANKING_DICT_FILE>) **וגם** על מספר ההופעות של הביטוי עצמו באותו דף (שאפשר לקבל ממילון המילים ב-<WORDS_DICT_FILE>).

בדירוג שלנו, הציון המשוקלל של כל דף בנוי מהמכפלה של ערך הדף במילון הדירוגים, כפול מספר ההופעות של מילות החיפוש בדף. למשל - אם נחפש את המילה **Scar** בדף **Harry_Potter**.html וכן:

- ✔ ערך הדף **Harry_Potter**.html במילון הדירוגים הינו Y
- ✔ מספר ההופעות של **Scar** בדף זה הינו Z

אזי הציון המשוקלל של הדף **Harry_Potter**.html, ספציפית עבור החיפוש של **Scar**, הינו: Y*Z. אם השאילתא מכילה יותר ממילה אחת: עבור **כל אחד** מהדפים הנבחרים בדירוג, המספר Z יהיה מספר ההופעות של המילה בעלת מספר ההופעות המינימלי באותו דף מבין המילים בשאילתא. כלומר, יתכן שהמספר Z שונה בין דף לדף, ומתאר שכיחות של מילים שונות בשאילתא. *אינטואיטיבית, נרצה לבחור את הדפים הרלוונטים ביותר גם עבור המילים שפחות שכיחות בהם.*

פורמט הפלט הינו:

<page1> <score1>

<page2> <score2>

...

כאשר בכל שורה יש את שם דף הויקי (Harry_Potter.html, Albus_Dumbledore.html וכו') ע"פ הציון המשוקלל, וכן הציון המשוקלל עצמו, מופרדים בתו רווח בודד, וממוינים מהציון הגבוה לנמוך.

הנחיות נוספות:

- ניתן להניח שמילון המילים ומילון הדירוגים נוצרו ע"י אותו קובץ אינדקס. כלומר, ניתן להניח שכל הדפים שמופיעים במילון המילים מופיעים גם במילון הדירוגים ולהיפך.
- שימו לב שגם כן ייתכן שאחד מהמילונים (או שניהם) ריק. יש לטפל גם במקרה קצה זה.
- אם אחת ממילות החיפוש אינה מופיעה במילון המילים, יש לסיים את החיפוש ולא להדפיס דבר.
- תיתכן קבלה של QUERY ריק (כלומר מחרוזת ריקה שלא מכילה אף מילה) - במקרה כזה עדיין יש להחזיר תוצאות חיפוש העונות לשאר דרישות הדירוג, חישבו באיזה אופן.
- שימו לב שאת קובץ ה- results.txt (בהנחיה למטה) עליכם ליצור באופן ידני, כלומר, התוכנית שלכם לא צריכה ליצור את הקובץ הזה באופן אוטומטי.

שאילתות חיפוש לדוגמא

עליכם לבצע חיפוש עבור השאילתות הבאות, ולהוציא את התוצאות לקובץ ***results.txt:***

1. Dementor
2. Quidditch
3. Muggles
4. broom
5. Sirius Black
6. secret

הקובץ ***results.txt*** צריך להיות בעל הפורמט הבא:

```
<OUTPUT OF SEARCH QUERY 1>
**********
<OUTPUT OF SEARCH QUERY 2>
**********
<OUTPUT OF SEARCH QUERY 3>
**********
<OUTPUT OF SEARCH QUERY 4>
**********
<OUTPUT OF SEARCH QUERY 5>
```

****
<OUTPUT OF SEARCH QUERY 6>
****
יש להריץ את אלגוריתם ה-Page Rank ל-100 איטרציות, ולהציג (עד) ארבע התוצאות עם הציון המשוקלל הגבוה ביותר.
כלומר הפלט של התוכנית עבור כל שאילתת חיפוש, ולאחריה שורה המכילה עשר כוכביות, ללא שורות רווח לפני ואחרי פלט של כל שאילתא ושורת כוכביות.

הוראות הגשה

עליכם להגיש את הקובץ **ex6.zip** (בלבד) בקישור ההגשה של תרגיל 6 דרך אתר הקורס על ידי לחיצה על "Upload file". אנו ממליצים להתחיל לעבוד על התרגיל בשלב מוקדם.
ex6.zip צריך לכלול לפחות את הקבצים **moogle.py** ו-**results.txt**, אך יכול גם להכיל קבצי פייתון נוספים במידה והתוכנית שלכם פוצלה לכמה קבצים. **שימו לב לא לכלול קבצי קלט/פלט וקבצים של הסביבה הווירטואלית. כמו כן, אין להגיש את הקובץ small_index.txt.**

הנחיות כלליות בנוגע להגשה

- הנכם רשאים להגיש תרגילים דרך מערכת ההגשות באתר הקורס מספר רב של פעמים. ההגשה האחרונה בלבד היא זו שקובעת ושתיבדק.
- לאחר הגשת התרגיל, ניתן ומומלץ להוריד את התרגיל המוגש ולוודא כי הקבצים המוגשים הם אלו שהתכוונתם להגיש וכי הקוד עובד על פי ציפיותיכם.
- באחריותכם לוודא כי PDF הבדיקות נראה כמו שצריך.
- קראו היטב את קובץ נהלי הקורס לגבי הנחיות נוספות להגשת התרגילים.
- שימו לב - יש להגיש את התרגילים בזמן!

בהצלחה!