ระบบสืบค้นหนังสือ

จัดทำโดย

นนท์ปวิธ อุดมวงศ์ 07570503

ปฏิวัติ ชำกรม 07590511

เสนอ

อาจารย์ ดร.วีณาวดี ม่วงอ้น

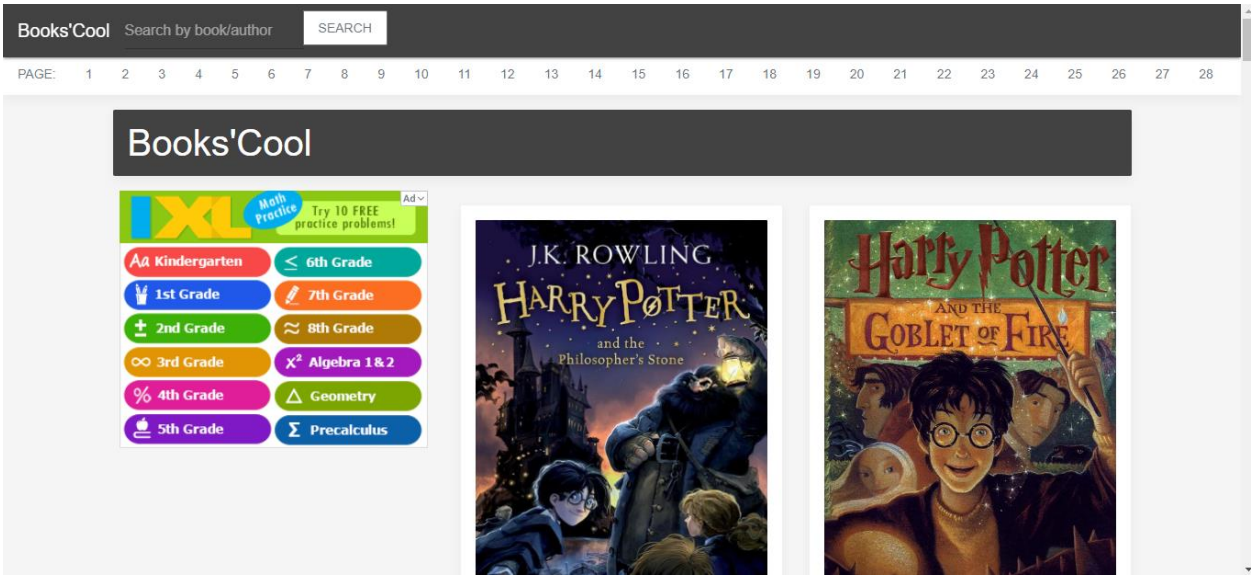
รายงานนี้เป็นส่วนหนึ่งของรายวิชา 520421 INFORMATION STORAGE AND RETRIEVAL

ภาคการศึกษาปลาย ปีการศึกษา 2562 ภาควิชาคอมพิวเตอร์ คณะวิทยาศาสตร์

มหาวิทยาลัยศิลปากร วิทยาเขตพระราชวังสนามจันทร์

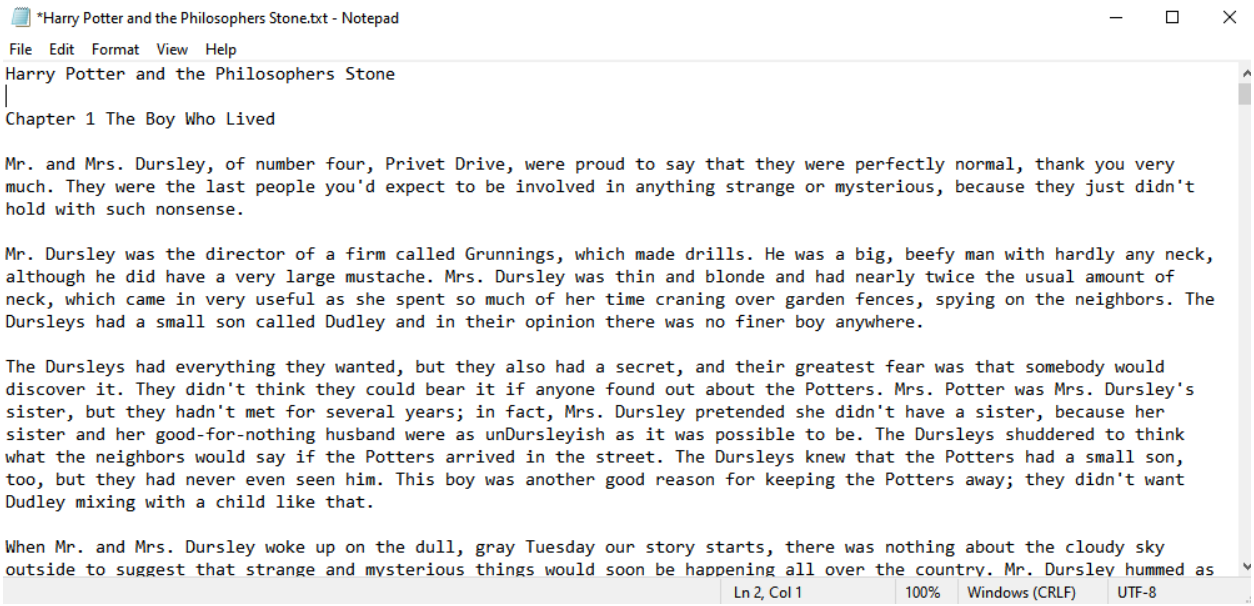
ค้นหาข้อมูล

ทำการค้นหาข้อมูลหนังสือจากเว็บไซต์ https://www.bookscool.com/en/index.php เพื่อเก็บรวบรวมเป็นคลังข้อมูลที่ใช้ในระบบสืบค้น



การจัดเตรียมข้อมูล

ทำการจัดเตรียมข้อมูลเองแบบ manual โดยบันทึกเนื้อหาในหนังสือมาเก็บไว้ในไฟล์ .txt ผ่านโปรแกรม notepad



ขั้นตอนการทำงาน

1.import module ต่างๆที่จำเป็นต้องใช้

```
import nltk,math                                        # ทำการ import library ต่างๆ ที่จำเป็นต้องใช้
from nltk.corpus import *                               # nltk เป็นโมดูลในภาษาไพทอนที่ช่วยในการประมวลภาษาธรรมชาติ
from nltk.stem import WordNetLemmatizer, PorterStemmer  # nltk,math เป็นฟังก์ชันที่ช่วยในการคำนวณทางคณิตศาสตร์
from nltk.tokenize import *                             # nltk.corpus เป็นฟังก์ชันที่ช่วยในการทำ corpus
from collections import Counter                         # nltk.tokenize เป็นฟังก์ชันที่ช่วยในการทำ tokenizing
import glob                                             # nltk.stem เป็นฟังก์ชันที่ช่วยในการทำ stemming
import re                                               # re เป็นโมดูลที่ช่วย ค้นหา/คัดกรอง ข้อความที่มีลักษณะและรูปแบบตรงกับที่เราต้องการ
import os                                               # Counter เป็นฟังก์ชันที่ช่วยนับจำนวนใน list
import numpy as np                                      # os เป็นฟังก์ชันให้เรียกใช้งานที่เกี่ยวกับการเชื่อมต่อกับระบบปฏิบัติการ
import sys                                              # numpy เป็นโมดูลใช้คำนวณทางคณิตศาสตร์และวิทยาศาสตร์
Stopwords = set(stopwords.words('english'))
```

2.นำไฟล์ที่จัดเก็บไว้ มาทำเป็น corpus

```
corpusroot = 'C:\\Users\\User\\Desktop\\irfile' #นำไฟล์ที่เก็บไว้ในโฟลเดอร์ irfile มาใส่ในรูปแบบ corpus
vectors={}                          #ทำ tf-idf vectors สำหรับทุกๆเอกสาร
df=Counter()                        #จัดเก็บค่าความถี่ของเอกสาร
tfs={}                              #จัดเก็บ tfs สำหรับทุกๆ tokens ในเอกสารทั้งหมด
lengths=Counter()                   #ใช้เพื่อคำนวณความยาวของเอกสาร
postings_list={}                    #ทำการ posting แต่ละรายการที่จัดเก็บไว้ สำหรับแต่ละ token ใน corpus
wordlists = PlaintextCorpusReader(corpusroot, '.*txt')
wordlists.fileids()
```

3.

-ทำการอ่านไฟล์

-ทำ tokenizing แต่ละเอกสาร

-กำจัด stopword

-ทำการ stemming

```
tokenizer = RegexpTokenizer(r'[a-zA-Z]+')
for filename in os.listdir(corpusroot):
    Stemmer = PorterStemmer()
    file = open(os.path.join(corpusroot, filename), "r", encoding='UTF-8')
    doc = file.read()                                           # ทำการอ่านไฟล์
    file.close()
    doc = doc.lower()                                           # แปลงค่าตัวอักษรให้เป็นตัวพิมพ์เล็ก
    tokens = tokenizer.tokenize(doc)                            # ทำ tokenizing แต่ละเอกสาร
    sw=stopwords.words('english')
    tokens = [Stemmer.stem(token) for token in tokens if token not in sw]  #กำจัด stopword และทำการ stemming
    #print(tokens)
    tf=Counter(tokens)
    df+=Counter(list(set(tokens)))
    tfs[filename]=tf.copy()                                     # copy ค่า tf ไปใส่ใน tfs ใน filename
    tf.clear()                                                  # ทำการเคลียร์ค่า tf ใหม่ ทำให้เอกสารถัดไปมีค่า tf ว่าง
```

4.ทำการหาคำที่ไม่ซ้ำกัน และตัดอักขระพิเศษออก

```
def finding_all_unique_words_and_freq(words):                    #ทำการหาคำที่ไม่ซ้ำกัน และตัดอักขระพิเศษออก
    words_unique = []
    word_freq = {}
    for word in words:
        if word not in words_unique:
            words_unique.append(word)
    for word in words_unique:
        word_freq[word] = words.count(word)
    return word_freq

def finding_freq_of_word_in_doc(word,words):
    freq = words.count(word)

def remove_special_characters(text):
    regex = re.compile('[^a-zA-Z0-9\s]')
    text_returned = re.sub(regex,'',text)
    return text_returned
```

5.คำนวณค่า weight ของ token ในเอกสาร

```
def calWeight(filename, token):                                  # คำนวณค่า weight ของ token ในเอกสาร โดยไม่ได้ทำการ normalizing
    idf=getidf(token)
    return (1+(tfs[filename][token]))*idf
def getidf(token):
    if df[token]==0:
        return -1
    return len(tfs)/df[token]
```

6.คำนวณ tf-idf vectors และความยาวของเอกสาร

```
# loop สำหรับการคำนวณ tf-idf vectors และความยาวของเอกสาร
for filename in tfs:
    vectors[filename]=Counter()                    #initializing tf-idf vector แต่ละเอกสาร
    length=0
    for token in tfs[filename]:
        weight = calWeight(filename, token)        #calWeight คำนวณค่า weight ของ token ในเอกสาร โดยไม่ได้ทำการ normalizing
        vectors[filename][token]=weight            #ค่า weight ของ token
        length += weight**2                        #คำนวณความยาวสำหรับการทำ normalizing ในภายหลัง
    lengths[filename]=math.sqrt(length)
```

7.ทำ normalizing ค่า weights

```
# loop สำหรับการทำ normalizing ค่า weights
for filename in vectors:
    for token in vectors[filename]:
        vectors[filename][token]= vectors[filename][token] / lengths[filename]      #หารค่า weights ด้วยความยาวของเอกสาร
        if token not in postings_list:
            postings_list[token]=Counter()
        postings_list[token][filename]=vectors[filename][token]     #ทำการ copy ค่าที่ normalize แล้ว ลงใน posting list
def getweight(filename,token):
    return vectors[filename][token]
```

8.ทำการคำนวณหาค่า cosine similarity

```
def query(qstring):                             #ฟังก์ชันที่จะ return ค่าที่ match กับ query มากที่สุด
    qstring=qstring.lower()                     #ทำการแปลงคำศัพท์ให้เป็น lower case
    qtf={}
    qlength=0
    flag=0
    loc_docs={}
    tenth={}
    cos_sims=Counter()                          #คำนวณ หาค่า cosine similarity
    for token in qstring.split():
        token=Stemmer.stem(token)               #ทำ stemming token โดยใช้ PoterStemmer
        if token not in postings_list:          #ถ้า token ไม่ตรงกับในคำศัพท์ ก็ไม่ต้องทำอะไร
            continue
        if getidf(token)==0:        #ถ้า token มีค่า idf = 0 ค่าทั้งหมดใน posting list จะเป็น 0 แล้วค่าที่มากสุด 10 ค่า จะถูกเลือกออกมาอย่างสุ่ม
            loc_docs[token], weights = zip(*postings_list[token].most_common())   #เพื่อที่จะไม่ให้เป็นเช่นนั้น เราจะจัดเก็บเอกสารทั้งหมด
        else:
            loc_docs[token],weights = zip(*postings_list[token].most_common(10))        #แล้วทำ top 10 in postings list
        tenth[token] = weights[-1]
        if flag==1:
            commondocs=set(loc_docs[token]) & commondocs
        else:
            commondocs=set(loc_docs[token])
            flag=1
        qtf[token]=1+(qstring.count(token))         #อัพเดทค่าความถี่ของ token ใน query
        qlength+=qtf[token]**2                      #คำนวณค่าความยาว สำหรับการทำ normalizing query tf ในภายหลัง
    qlength=np.sqrt(qlength)
    for doc in vectors:
        cos_sim=0
        for token in qtf:
            if doc in loc_docs[token]:
                cos_sim = cos_sim + (qtf[token] / qlength) * postings_list[token][doc]   #คำนวณ score ถ้าเอกสารอยู่ใน 10 อันดับแรก
```

9.ทำการ print ค่า similarity ระหว่าง query ในแต่ละ document และ print ค่าที่ match ที่สุดออกมาด้วย

```
            else:
                cos_sim = cos_sim + (qtf[token] / qlength) * tenth[token]
        cos_sims[doc]=cos_sim
    max=cos_sims.most_common(10)
    ans,wght=zip(*max)

    print("----------------------------------------")
    print("Simirality between query and each document")         # print ค่า similarity ระหว่าง query ในแต่ละ document
    for item in max:
        print(item)
        print()
    print("----------------------------------------")

    print("Best Matching")                                      # print ค่าที่ match ที่สุด
    try:
        if ans[0] in commondocs:
            return ans[0],wght[0]
        else:
            return "fetch more",0
    except UnboundLocalError:
        return "None",0
```

```
x = input('Enter your query:')      # ป้อนคำที่ต้องการค้นหา
query(x)
```

```
Enter your query:
```

9.ผลลัพธ์ คือ หนังสือที่ match กับคำค้นหา(query) และค่า similarity ของ 10 อันดับแรก โดยเรียงจากค่าที่มากที่สุด ไปหาค่าที่น้อยที่สุด และผลลัพธ์ในบรรทัดสุดท้าย แสดงหนังสือกับค่า similarity ที่ match ที่สุด

ตัวอย่างผลลัพธ์ที่ 1

```
x = input('Enter your query:')      # ป้อนคำที่ต้องการค้นหา
query(x)
```

```
Enter your query:hollow
----------------------------------------
Simirality between query and each document
('Harry Potter and the Deathly Hallows.txt', 0.009866148813257631)

('1-The-Hunger-Games.txt', 0.002578350953540027)

('2-Catching-Fire.txt', 0.002343138103220843)

('3.2-Mockingjay.txt', 0.0013592755266202117)

('Harry Potter and the Half-Blood Prince.txt', 0.001166324337701512)

('Harry Potter and the Order of the Phoenix.txt', 0.0010710815818025599)

('Harry Potter and the Prisoner of Azkaban.txt', 0.0008611776689488861)

('The maze runner.txt', 0.0008262901348869669)

('Harry Potter and the Chamber of Secrets.txt', 0.0007963755328331616)

('3.1-Mockingjay.txt', 0.0007122534761523213)

----------------------------------------
Best Matching

('Harry Potter and the Deathly Hallows.txt', 0.009866148813257631)
```

ตัวอย่างผลลัพธ์ที่ 2

```
x = input('Enter your query:')      # ป้อนคำที่ต้องการค้นหา
query(x)
```

```
Enter your query:the maze runner
----------------------------------------
Simirality between query and each document
('The maze runner.txt', 0.0424642983117306)

('Harry Potter and the Goblet of Fire.txt', 0.005039668112967894)

('3.1-Mockingjay.txt', 0.0037514145889238557)

('3.2-Mockingjay.txt', 0.001843625577025754)

('2-Catching-Fire.txt', 0.0016350521863296823)

('Harry Potter and the Prisoner of Azkaban.txt', 0.0016072095202483015)

('1-The-Hunger-Games.txt', 0.001382586280911919)

('Harry Potter and the Philosophers Stone.txt', 0.0011246755092980728)

('Harry Potter and the Chamber of Secrets.txt', 0.0010473332001518752)

('Harry Potter and the Deathly Hallows.txt', 0.0010473332001518752)

----------------------------------------
Best Matching

('The maze runner.txt', 0.0424642983117306)
```

ตัวอย่างผลลัพธ์ที่ 3

```
x = input('Enter your query:')      # ป้อนคำที่ต้องการค้นหา
query(x)
```

```
Enter your query:azkaban
----------------------------------------
Simirality between query and each document
('Harry Potter and the Prisoner of Azkaban.txt', 0.01789335823260463)

('Harry Potter and the Goblet of Fire.txt', 0.010393785201360197)

('Harry Potter and the Half-Blood Prince.txt', 0.009622175786037475)

('Harry Potter and the Order of the Phoenix.txt', 0.008247328179879712)

('Harry Potter and the Deathly Hallows.txt', 0.005481193785143128)

('1-The-Hunger-Games.txt', 0.0029200436203882596)

('2-Catching-Fire.txt', 0.0029200436203882596)

('3.1-Mockingjay.txt', 0.0029200436203882596)

('3.2-Mockingjay.txt', 0.0029200436203882596)

('Harry Potter and the Chamber of Secrets.txt', 0.0029200436203882596)

----------------------------------------
Best Matching

('Harry Potter and the Prisoner of Azkaban.txt', 0.01789335823260463)
```

ตัวอย่างผลลัพธ์ที่ 4

```
x = input('Enter your query:')      # ป้อนคำที่ต้องการค้นหา
query(x)
```

```
Enter your query:phoenix
----------------------------------------
Simirality between query and each document
('Harry Potter and the Order of the Phoenix.txt', 0.006230848360886245)

('Harry Potter and the Deathly Hallows.txt', 0.00500941668483798)

('Harry Potter and the Half-Blood Prince.txt', 0.004376328265907242)

('Harry Potter and the Philosophers Stone.txt', 0.003989717659155444)

('Harry Potter and the Goblet of Fire.txt', 0.0036241602495333675)

('1-The-Hunger-Games.txt', 0.0029704639058366926)

('2-Catching-Fire.txt', 0.0029704639058366926)

('3.1-Mockingjay.txt', 0.0029704639058366926)

('3.2-Mockingjay.txt', 0.0029704639058366926)

('Angels and demons.txt', 0.0029704639058366926)

----------------------------------------
Best Matching

('Harry Potter and the Order of the Phoenix.txt', 0.006230848360886245)
```

ตัวอย่างผลลัพธ์ที่ 5

```
x = input('Enter your query:')      # ป้อนคำที่ต้องการค้นหา
query(x)
```

```
Enter your query:angel demon
----------------------------------------
Simirality between query and each document
('Angels and demons.txt', 0.37081100251169624)

('Harry Potter and the Chamber of Secrets.txt', 0.0023990411853497514)

('Harry Potter and the Half-Blood Prince.txt', 0.0022898171135454395)

('Harry Potter and the Philosophers Stone.txt', 0.0019597214105939323)

('Harry Potter and the Prisoner of Azkaban.txt', 0.001958481856118378)

('1-The-Hunger-Games.txt', 0.0013728580117241648)

('2-Catching-Fire.txt', 0.0013728580117241648)

('3.1-Mockingjay.txt', 0.0013728580117241648)

('3.2-Mockingjay.txt', 0.0013728580117241648)

('Harry Potter and the Deathly Hallows.txt', 0.0013728580117241648)

----------------------------------------
Best Matching

('Angels and demons.txt', 0.37081100251169624)
```