TUGAS BESAR 2 IF2123 ALJABAR LINIER DAN GEOMETRI
APLIKASI *DOT PRODUCT* PADA SISTEM TEMU-BALIK INFORMASI
SEMESTER I TAHUN 2020/2021

KELOMPOK 38

aman damai lancar mantap amin ^-^)b

Rhea Elka Pandumpi (13519047)

Michael Owen (13519055)

Jeanne D'Arc Amara Hanieka (13519082)

BAB I
DESKRIPSI MASALAH

Spesifikasi program adalah sebagai berikut:

1. Program mampu menerima search query. Search query dapat berupa kata dasar maupun berimbuhan.
2. Dokumen yang akan menjadi kandidat dibebaskan formatnya dan disiapkan secara manual. Minimal terdapat 15 dokumen berbeda sebagai kandidat dokumen. Bonus: Gunakan web scraping untuk mengekstraksi dokumen dari website.
3. Hasil pencarian yang terurut berdasarkan similaritas tertinggi dari hasil teratas hingga hasil terbawah berupa judul dokumen dan kalimat pertama dari dokumen tersebut. Sertakan juga nilai similaritas tiap dokumen.
4. Program disarankan untuk melakukan pembersihan dokumen terlebih dahulu sebelum diproses dalam perhitungan cosine similarity. Pembersihan dokumen bisa meliputi hal-hal berikut ini.
   a. Stemming dan Penghapusan stopwords dari isi dokumen.
   b. Penghapusan karakter-karakter yang tidak perlu

BAB 2
TEORI SINGKAT

2.1. Temu Balik Informasi

Temu-balik informasi (*information retrieval*) merupakan suatu proses menemukan kembali (*retrieval*) informasi yang relevan terhadap kebutuhan pengguna dari suatu kumpulan informasi secara otomatis. IR tidak sama dengan pencarian di dalam basis data (*database*). IR umumnya digunakan pada pencarian informasi yang isinya tidak terstruktur.

2.2. IR dengan Model Ruang Vektor

Salah satu model IR adalah model ruang vektor. Model ini menggunakan teori di dalam aljabar vektor. Misalkan terdapat n kata berbeda sebagai kamus kata (*vocabulary*) atau indeks kata (*term index*). Kata-kata tersebut membentuk ruang vektor berdimensi n. Setiap dokumen maupun *query* dinyatakan sebagai vektor $w = (w_1, w_2, ..., w_n)$ di dalam R^n. w_i merupakan bobot setiap kata i di dalam *query* atau dokumen. Nilai w_i dapat menyatakan jumlah kemunculan kata tersebut dalam dokumen (*term frequency*).

Penentuan dokumen mana yang relevan dengan *query* dipandang sebagai pengukuran kesamaan (*similarity measure*) antara *query* dengan dokumen. Semakin sama suatu vektor dokumen dengan vektor *query*, semakin relevan dokumen tersebut dengan query. Kesamaan (sim) antara dua vektor $Q = (q_1, q_2, \ldots, q_n)$ dan $D = (d_1, d_2, \ldots, d_n)$ diukur dengan rumus *cosine similarity* yang merupakan bagian dari rumus perkalian titik (*dot product*) dua buah vektor:

$$\mathbf{Q} \cdot \mathbf{D} = \|\mathbf{Q}\|\|\mathbf{D}\| \cos\theta \quad \Rightarrow \quad sim(\mathbf{Q}, \mathbf{D}) = \cos\theta = \frac{\mathbf{Q} \cdot \mathbf{D}}{\|\mathbf{Q}\|\|\mathbf{D}\|}$$

dengan Q D adalah perkalian titik yang didefinisikan sebagai

$$\mathbf{Q} \cdot \mathbf{D} = q_1 d_1 + q_2 d_2 + \ldots + q_n d_n$$

Jika cos = 1, berarti = 0, vektor Q dan D berimpit, yang berarti dokumen D sesuai dengan query Q. Jadi, nilai cosinus yang besar (mendekati 1) mengindikasikan bahwa dokumen

cenderung sesuai dengan query. Setiap dokumen di dalam koleksi dokumen dihitung kesamaannya dengan query dengan rumus cosinus di atas.

Selanjutnya hasil perhitungan di-ranking berdasarkan nilai cosinus dari besar ke kecil sebagai proses pemilihan dokumen yang yang “dekat” dengan query. Pe-ranking-an tersebut menyatakan dokumen yang paling relevan hingga yang kurang relevan dengan query. Nilai cosinus yang besar menyatakan dokumen yang relevan, nilai cosinus yang kecil menyatakan dokumen yang kurang relevan dengan query.

BAB 3
IMPLEMENTASI PROGRAM

Source code terdiri dari beberapa bagian, yaitu *back-end* menggunakan bahasa pemrograman Python (tersedia pada bagian functions.py) serta *front-end* yang memanfaatkan HTML serta Python menggunakan microframework Flask (upload.html, hello.html, found.html, notfound.html, app.py). Proses berjalannya program adalah sebagai berikut:

1. Web menerima file bertipe .txt yang di *upload* menuju *directory* tertentu menggunakan tombol "*Choose Files" dan "Submit"* pada halaman web. Proses ini memanfaatkan fungsi *request* dari Flask (digunakan pada fungsi `upload_file()` pada app.py) serta *button* dan *file input* dari HTML.

```
<-! bagian upload file dan tombol submit -->
<form method="post" action="/" enctype="multipart/form-data">
<input type="file" name="files[]" multiple="true" autocomplete="off" required>
<input type="submit" value="Submit">
```

found.html, upload.html, hello.html, notfound.html

2. Untuk menyimpan semua data, digunakan *array*. *Array* "file" berisi *directory* masing-masing dokumen, "judulfile" berisi judul masing-masing dokumen, "kalimatpertama" berisi kalimat pertama dari masing-masing dokumen, "teks" berisi vektor kata dari masing-masing dokumen, "sim" berisi similarity masing-masing dokumen, dan "K" berisi seluruh kalimat dalam masing-masing dokumen.

```
#list penyimpanan data
file=["filetxt/txt1.txt","filetxt/txt2.txt","filetxt/txt3.txt","filetxt/txt4.txt","file
txt/txt5.txt","filetxt/txt6.txt","filetxt/txt7.txt","filetxt/txt8.txt","filetxt/txt9.tx
t","filetxt/txt10.txt","filetxt/txt11.txt","filetxt/txt12.txt","filetxt/txt13.txt","fil
etxt/txt14.txt","filetxt/txt15.txt"]
judulfile=[0 for i in range(panjang(file))]
kalimatpertama=[0 for i in range (panjang(file))]
teks=[0 for i in range(panjang(file))]
sim=[0 for i in range(panjang(file))]
K=[0 for i in range(panjang(file))]
```

```
#pengisian list
for i in range(panjang(file)):
    openfile=open(file[i],"r")
    read=openfile.readline()
    judulfile[i]=read
    readlagi=openfile.readline()
    kalimatpertama[i]=readlagi
for i in range(panjang(file)):
    openfile=open(file[i],"r")
    readfile=openfile.read()
    K[i]=readfile
    d=Simplify(readfile)
    v=vektor(d)
    teks[i]=v
```

List penyimpan semua data.

3. Setelah file di-*upload*, maka proses pencarian dilakukan dengan memasukkan *query* dan memencet tombol "Cari". Pembacaan isi *query* pada app.py dilakukan dengan memanfaatkan `request.form`.

```
<-! HTML bagian form pencarian dan tombol cari  -->
<form action="" method="post" role="form">
   {{ form.csrf }}
   <div class="row align-items-center justify-content-center">
       <div class="form-inline">
              <input    type="text"    class="form-control"    id="name"    name="name"
              placeholder="Masukkan query anda" size="75">
             <button type="submit" class="btn btn-success">Cari</button>
       </div>
   </div>
</form>
```

found.html, upload.html, hello.html, notfound.html

4. Setelah *query* diterima, hasil pembacaan *query* oleh `request.form` dimasukkan ke dalam fungsi `Simplify(query)`. Fungsi ini akan me-*return query* yang sudah melalui proses *stemming* dan penghapusan *stopwords*. Setelah selesai di Simplify, *query* akan diubah menjadi sebuah vektor dan dimasukkan ke dalam variabel baru bernama Q.

```
    if request.method == 'POST':
        name = request.form['name']
        print(name)
    if form.validate():
        query = name
        q = Simplify(query)
        Q = vektor(q)
```

app.py

```
Stem(x): untuk membuat suatu kata menjadi bentuk paling sederhana
Stopword(x): menghilangkan kata-kata penghubung dan sebagainya
Simplify(x): menggabungkan Stem(x) dan Stopword (x)
vektor(x): mengubah teks biasa menjadi vektor
```

functions.py

5. Untuk membuat *term*, dibuat suatu *array* allDoc, dan allDoc akan menyimpan vektor kata dari seluruh dokumen. Term adalah vektor basis dari vektor kata *query* dan allDoc. Basis sendiri artinya bahwa tidak ada kata yang berulang (hanya diambil satu kata), sehingga fungsi basis ini agar *term* tidak terisi kata berulang. Kemudian, vektor kata akan mengalami vektorisasi yang berisi kemunculan kata dan disimpan pada *array* M. M adalah *array* yang menyimpan hasil vektorisasi dari setiap dokumen (untuk vektorisasi akan dilanjutkan pada bagian selanjutnya). Kemudian, akan dilakukan juga vektorisasi pada vektor kata Q.

```
# Term hasil concat dari kata-kata yang ada di dokumen sama query.
        allDoc = []
        for i in range(panjang(teks)):
            a = teks[i]
            allDoc += a
        term = basis(Q + allDoc)
```

```
            M = []
            for i in range(panjang(teks)):
                M.append(jadiinvektor(teks[i], term))

            Query = jadiinvektor(Q, term)
```

6. Untuk vektorisasi sendiri mengikuti langkah-langkah berikut:

   Pertama, setel frekuensi dari kemunculan sebesar 0. Dengan menelusuri kata-kata pada vektor kata suatu dokumen dan kata-kata pada term, apabila ditemukan bahwa kata tersebut merupakan salah satu elemen *array term*, maka dapat dikatakan bahwa frekuensi kemunculan salah satu kata pada *term* dalam dokumen tersebut bertambah.

```
def jadiinvektor(x, term): #Contoh x: D1, D2, query (yang sudah divektorbasiskan)
    #Inisialisasi vektor
    frekuensi = [0 for i in range (panjang(term))]
    #Inisialisasi vektor
    for token in x:
        for i in range (panjang(term)):
            if(token==term[i]):
                frekuensi[i]=frekuensi[i]+1
    return frekuensi
```

7. Apabila Q (*query*) kosong atau hanya terdiri dari *stopwords*, maka akan dimunculkan notfound.html yang ditampilkan menggunakan render_template dari Flask. Apabila ada, maka akan masuk ke proses selanjutnya.Dilakukan skema *sorting* menggunakan fungsi `sort(M, Query)`, dimana M merupakan kumpulan dokumen yang telah di upload dan telah dijadikan vektor. Setelah itu, judul-judul dari dokumen-dokumen tersebut diurutkan berdasarkan *similarity* di dalam *array* bernama data.

```
def sort(M,Query):
    i = 0
    while(i<panjang(M)):
        j=i+1
        while(j<panjang(M)):
```

```
            if(similarity(M[i],Query)<similarity(M[j],Query)):
                temp=M[i]
                M[i]=M[j]
                M[j]=temp
            j=j+1
        i=i+1
    return M
```

Skema *sorting* M berdasarkan *similarity.*

8. Untuk *cosine similarity* mengikuti algoritme sebagai berikut:

   Dengan diperolehnya vektorisasi dari fungsi jadiinvektor, maka panjang dari dokumen ataupun query dapat dihitung dengan fungsi panjangvektor(panjang vektor didapatkan dari akar dari penjumlahan kuadrat seluruh komponen vektor). Sedangkan fungsi dotProduct merupakan jumlah dari perkalian dari komponen vektor query pertama dengan komponen vektor kata pertama sampai komponen ke-*n* keduanya. *Similarity* sendiri adalah perkalian titik dibagi dengan panjang vektor dokumen dengan panjang vektor query.

```
def panjangvektor(x):
    sumofSquares=0
    for i in x:
        sumofSquares = sumofSquares + (i**2)
    return (sumofSquares)**(1/2)

def dotProduct(x, query):
    dotprod = 0
    for i in range (panjang(x)):
        dotprod = dotprod + query[i]*x[i]
    return dotprod

def similarity(x, query):
    sim = dotProduct(x,query)/(panjangvektor(x)*panjangvektor(query))
    return sim
```

```
dict = dictionary(K, M, Query)
judul = title(K, judulfile)
kalimat = firstline(K, kalimatpertama)
jumlahkata = hitungkata(K)
linkfile = fileteks(K, file)
```

Dictionary untuk menghubungkan seluruh teks dengan komponen-komponen lain. Untuk fungsi dictionary, title, firstline, hitungkata, dan fileteks, terdefinisi di file functions.py

```
sim = similar(data, Query, dict, K, judul)
kal = kalper(data, kalimat, K, judul)
li = link(data, linkfile, K, judul)
jum = jumlah(data, K, judul, jumlahkata)
```

Dictionary antara similarity, kalimat pertama, dan jumlah huruf, dengan judul yang telah terurut.

9. Lalu, setelah terurut, maka dibuat *dictionary* lagi antara *similarity*, kalimat pertama, dan jumlah huruf dengan judul yang telah terurut. Hal ini memanfaatkan *dictionary* yang telah dibuat sebelumnya, yang juga menghubungkan teks dengan komponen-komponen tersebut. Karena sebelumnya teks sudah dihubungkan dengan komponen-komponen tersebut, maka *dictionary* baru akan memiliki *data* sebagai *key* dan komponen-komponen tersebut sebagai *value* hanya bila *data* adalah judul dari dokumen-dokumen yang terkait dengan komponen-komponen tersebut. Hal ini akan menghubungkan ketiga hal ini agar nantinya bisa di buat *loop* yang dapat menampilkan berdasarkan urutan *similarity*. Pada akhirnya, hasil ini akan ditampilkan dalam bentuk tabel pada found.html.

```
        {% for key in data %}
          <tr>
              <tr>
            <th><a href=/{{li[key]}}/>{{ key }}</a></th>
              </tr>
              <tr>
                  <td> Jumlah kata: {{ jum[key] }} </td>
              </tr>
```

```
            <tr>
                <td> Tingkat kemiripan: {{ sim[key] }} % </td>
            </tr>
            <tr>
                <td> {{ kal[key] }} </td>
            </tr>
        </tr>
      {% endfor %}
```

Proses menampilkan judul, jumlah kata, tingkat kemiripan, dan kalimat pertama di Jinja.

10. Hasil akan ditunjukkan pada found.html, yang akan memunculkan judul, jumlah kata, tingkat kemiripan, serta kalimat pertama dari dokumen. Data ini diurutkan sesuai dengan similarity telah dicari sebelumnya dengan fungsi sort(M, Query), dengan *dictionary-dictionary* yang dibuat dengan menghubungkan antara judul dengan jumlah kata, judul dengan tingkat kemiripan, serta judul dengan kalimat pertama. Judul dari dokumen merupakan *link* yang akan mengarahkan pengguna ke dokumen artikel. *Link* tersebut akan mengarah ke rute yang sesuai dengan judul dari dokumen.
11. Selain itu, pada found.html juga ditampilkan tabel jumlah kemunculan kata-kata pada *query* pada masing-masing dokumen. Pada awalnya, dibuat terlebih dahulu *array* termTable dengan *array* baris pertama berisi judul dari tabel, yaitu “Term, Query, D1,... D15”, dengan kolom pertama baris selanjutnya berisi semua kata yang muncul di *query*, dan kolom selanjutnya menampilkan jumlah kemunculan pada masing-masing dokumen. Setelah dibuat *array* tersebut, maka pada found.html dilakukan *looping* untuk mencetak seluruh isi *array*.

```
base = basis(Q)
judultabel = ["Term", "Query", "D1", "D2", "D3", "D4", "D5", "D6", "D7", "D8",
"D9", "D10", "D11", "D12", "D13", "D14", "D15"]
termTable = [[0 for i in range(panjang(judultabel))] for j in range(panjang(base) +
1)]
termTable[0] = judultabel
for i in range(panjang(base)):
    termTable[i + 1][0] = base[i]
NewQuery = jadiinvektor(Q, base)
```

```
for i in range(panjang(base)):
     termTable[i + 1][1] = NewQuery[i]
NewM = []
for i in range(panjang(teks)):
     NewM.append(jadiinvektor(teks[i], base))
TransposeNewM = [[0 for i in range(panjang(teks))] for j in range(panjang(base))]
for i in range(panjang(base)):
     for j in range(panjang(teks)):
          TransposeNewM[i][j] = NewM[j][i]
for i in range(panjang(base)):
     for j in range(panjang(teks)):
          termTable[i + 1][j + 2] = TransposeNewM[i][j]
```

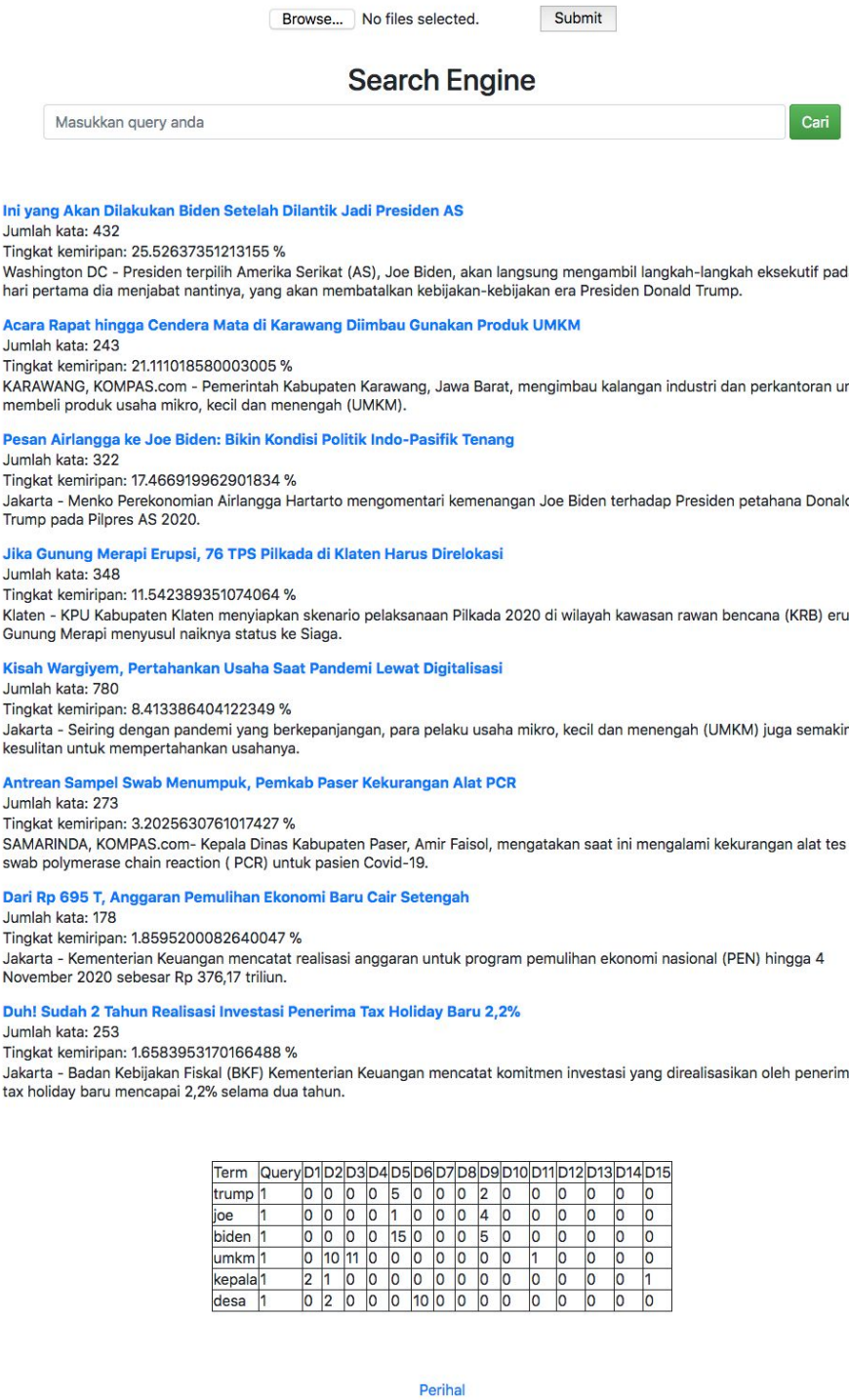
Pembuatan array sebelum di-print di Jinja.

```
       {% for i in termTable %}
          <tr>
              {% for j in i %}
              <td>{{j}}</td>
              {% endfor %}
          </tr>
        {% endfor %}
```

Pencetakan array di Jinja.

BAB 4
EKSPERIMEN

Pada saat query “trump joe biden umkm kepala desa”, muncul hasil pencarian sebagai berikut.



Judul dokumen merupakan sebuah link yang saat dipencet akan membawa pengguna menuju artikel yang ingin dibaca.

BAB 5
KESIMPULAN

1. Kesimpulan

   Dengan menerapkan pengetahuan *dot product* dalam *cosine similarity*, kita dapat membuat sebuah mesin pencari yang mampu melihat kesamaan antara isi *file* yang kita unggah dengan *search query* yang kita masukkan dan mengurutkannya sesuai *file* yang paling mirip dengan *search query* yang dimasukkan oleh pengguna. Mesin pencari ini dapat dibuat dengan menggunakan bahasa Python dan memanfaatkan HTML untuk membuat halaman web.

2. Saran

   Program ini dapat dikembangkan dalam beberapa aspek, yaitu:

   a. Menggunakan *user interface* yang lebih menarik agar program menjadi lebih *user friendly*.
   b. Membuat program dengan *response* yang lebih cepat agar hasil pencarian dapat langsung muncul ke *user* tanpa harus menunggu untuk jangka waktu tertentu.
   c. Dikembangkan sehingga mampu membaca artikel dari internet dengan menggunakan *web scraping*.

3. Refleksi

   Tugas ini telah mengajarkan penulis bahwa hal-hal yang dipelajari dalam perkuliahan sehari-hari seperti *dot product* pada aljabar mampu dimanfaatkan dan dikembangkan menjadi sesuatu yang bermanfaat seperti *search engine*. Selain itu, penulis belajar banyak tentang pengembangan web menggunakan Python dan Flask serta semakin mengerti materi *dot product*.

REFERENSI

Howard Anton, Elementary Linear Algebra, 10th edition, John Wiley and Sons, 2010

“Aplikasi Dot Product pada sistem temu balik aplikasi” oleh Rinaldi Munir
https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2020-2021/Algeo-12-Aplikasi-dot-product-pada-IR.pdf