



UNIVERSIDAD DIEGO PORTALES

TAREA N3: HADOOP

Profesor: Nicolas Hidalgo

Ayudantes:

Joaquín Fernandez

Nicolas Nuñez

Cristian Villavicencio

Integrantes: Abel Baulloza Almeida

Diego Carrillo Ormazabal

Fecha 27 de Noviembre 2022

Índice

1. Información fundamental	2
2. Diagrama de funcionamiento del código	4
3. Explicación de módulos de código	5
3.1. Wikipedia	5
4. Entorno docker y contenedor donde esta alojado hadoop	6
4.1. Mapper	6
4.2. Reducer	7
4.3. getJson	9
4.4. Buscador	11

1. Información fundamental

En el presente informe, se explicará la solución para el uso de una herramienta que funciona como sistema distribuido, esta es Hadoop.

Hadoop, es un gestor de archivos en grandes cantidades y tamaño, el cual, para este caso sera utilizado con el fin de simular un buscador de google. Esto mediante el uso de librerías de python asociadas a wikipedia, que nos permiten acceder a la información de esta enciclopedia para crear documentos con la información solicitada.

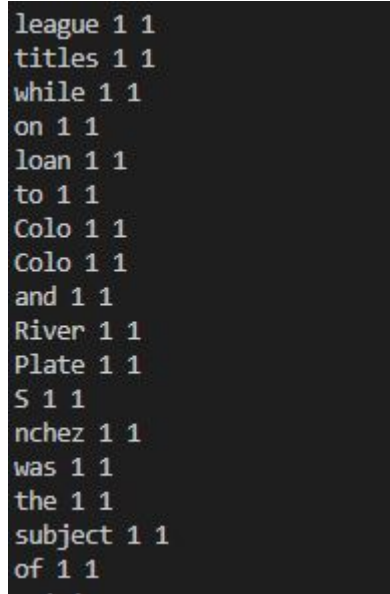
Una vez tomada toda la información necesaria, los datos son procesados mediante el uso de dos "softwares" y un método utilizado por Hadoop, estos son llamados como *Mapper* y *Reducer*, cada uno con una función principal diferente, pero que se complementan entre si.

A continuación, se definirán a grandes rasgos los funcionamientos de estos dos códigos, que fueron creados e implementados en python adaptados nuestras necesidades.

Mapper: Es encargado de leer todas las palabras dentro del texto e ir categorizándolas, detectando en que texto es encontrado y se presentan en el siguiente formato:

Word:(Texto , Contador)

En la implementación creada, se mostraron los siguientes resultados:



```
league 1 1
titles 1 1
while 1 1
on 1 1
loan 1 1
to 1 1
Colo 1 1
Colo 1 1
and 1 1
River 1 1
Plate 1 1
S 1 1
nchez 1 1
was 1 1
the 1 1
subject 1 1
of 1 1
```

Figura 1: Resultados Mapper

Reducer: Este, es encargado de tomar toda la información que es enviada mediante el mapper, para luego ser procesada. La forma en que actúa el reducer es tomando cada palabra y agrupándola según sean sus ubicaciones, es decir, tomo una palabra y si esta aparece repetida dentro de la información dada voy sumándole al contador las veces que esta esté repetida, y en caso que se repita, pero en otro texto, esta es agrupada de igual manera, pero para el nuevo texto asociado.

Los resultados obtenidos se mostraron de la siguiente manera, en donde se muestran agrupadas las palabras y contabilizadas.

```
Word      [(Document,Count)]
1      (1, 61) (2, 28) (3, 25) (4, 34) (5, 95) (7, 19) (8, 13) (9, 11)
START    (1, 1) (2, 1) (3, 1) (4, 1) (5, 1) (6, 1) (7, 1) (8, 1) (9, 1)
Alexis   (1, 6) (3, 1) (5, 1)
Alejandro (1, 1) (2, 2)
S        (1, 122) (2, 4) (3, 3) (4, 5) (5, 6) (7, 5) (8, 2) (9, 21)
nchez    (1, 122) (2, 1) (3, 1) (5, 1) (7, 3)
Spanish  (1, 1) (2, 33) (3, 2) (5, 7) (6, 3) (7, 140) (8, 9) (9, 7)
pronunciation (1, 1) (2, 1) (3, 1) (5, 1) (6, 1)
a        (1, 181) (2, 220) (3, 115) (4, 207) (5, 422) (6, 36) (7, 264) (8, 154) (9, 20)
leksis   (1, 1)
sant     (1, 1)
es       (1, 1) (7, 3)
born     (1, 1) (2, 3) (3, 2) (4, 2) (5, 5) (6, 1) (7, 6) (8, 2)
19       (1, 4) (2, 4) (3, 7) (4, 4) (5, 11) (7, 5) (8, 1) (9, 1)
December (1, 8) (2, 3) (3, 1) (4, 5) (5, 18) (6, 1) (7, 1) (8, 1) (9, 6)
1988     (1, 1)
also     (1, 12) (2, 29) (3, 13) (4, 41) (5, 37) (6, 3) (7, 47) (8, 30) (9, 5)
known    (1, 2) (2, 7) (3, 1) (4, 12) (5, 3) (6, 3) (7, 16) (8, 7) (9, 2)
mononymously (1, 1)
as       (1, 45) (2, 92) (3, 45) (4, 167) (5, 107) (6, 14) (7, 148) (8, 118) (9, 36)
is       (1, 8) (2, 135) (3, 13) (4, 164) (5, 33) (6, 5) (7, 130) (8, 102) (9, 11)
```

Figura 2: Resultados Reducer

2. Diagrama de funcionamiento del código

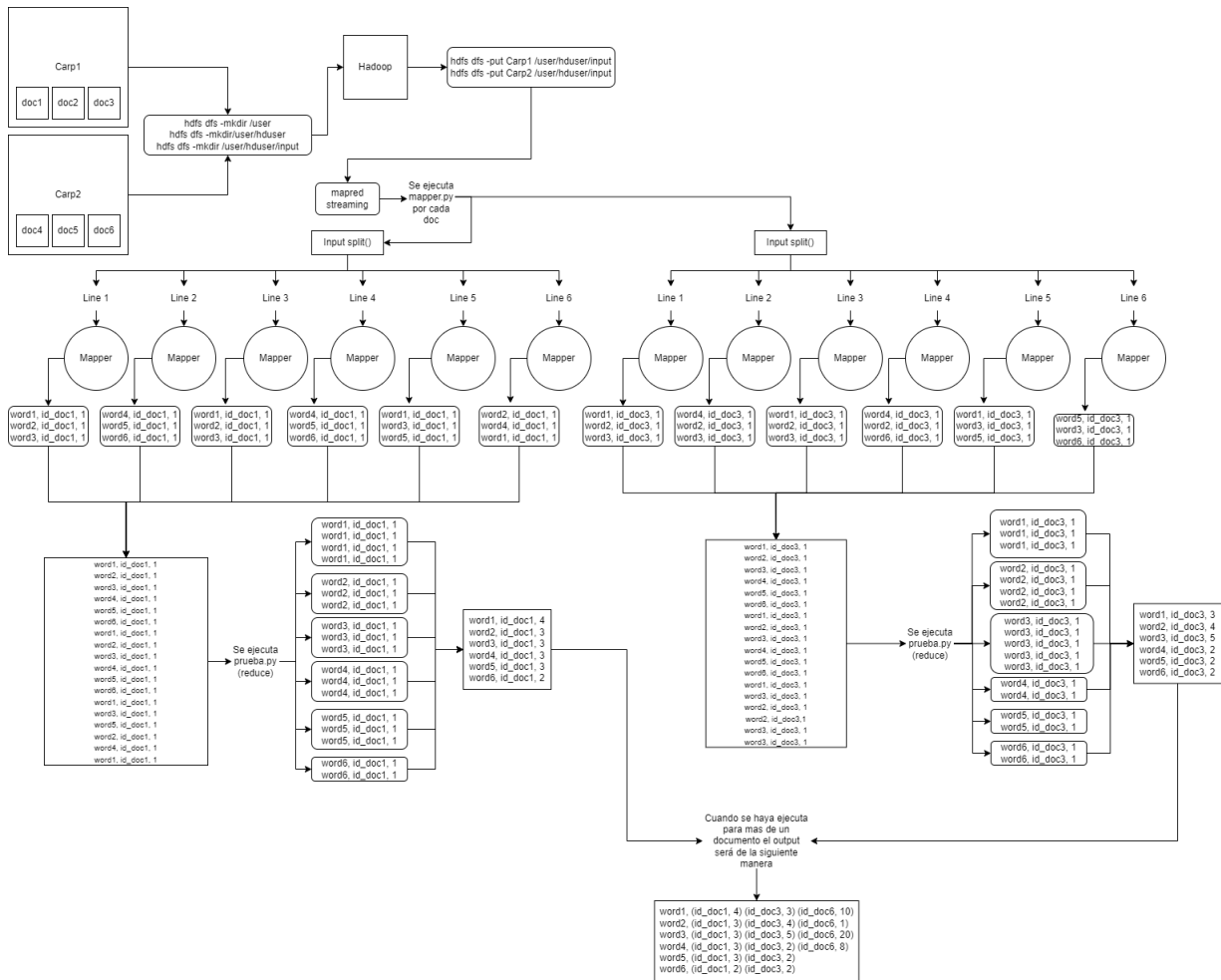


Figura 3: Diagrama de funcionamiento del código

3. Explicación de módulos de código

En la siguiente sección, se explicarán los códigos que permitieron que esta actividad se llevara a cabo.

3.1. Wikipedia

Como se mencionó en la sección anterior, es importante obtener o extraer la información de wikipedia, esto se realizó gracias a la librería wikipedia, la cual permite consumir su api, esta librería nos permite ingresar o entregarles palabras mediante un input, la api buscara en el sitio de wikipedia el documento que tenga el titulo de la palabra que se ingreso gracias la método `wiki.page`, este documento será extraído en la variable `document` y posteriormente almacenado en un archivo de texto titulado de la misma manera.

```
#Instalar previamente pip install wikipedia
import wikipedia as wiki

def wikipedia():
    #wiki.languages('es') cambiar lenguaje del documento.
    arr = ['Alexis Sanchez','Argentina','Arturo Vidal','Chile','Cristiano Ronaldo',
           'Diego Portales','España','Europa','Philippine Army','War']
    count = 1
    for nombres in arr:
        document = wiki.page(nombres)
        name = nombres + ".txt"
        name.replace("0", "")
        print(name)
        if count <= 5:
            ub = 'Documentos1/' + name
            arc = open(ub,"w",encoding='utf-8')
            info = str(count) + " START " + document.content
            arc.write(info)
            count = count + 1
        else:
            ub = 'Documentos2/' + name
            arc = open(ub,"w",encoding='utf-8')
            info = str(count) + " START " + document.content
            arc.write(info)
            count = count + 1
    wikipedia()
```

Figura 4: Código: Obtener información

Básicamente, este código nos permite obtener un archivo de texto con lo buscado.

4. Entorno docker y contenedor donde esta alojado hadoop

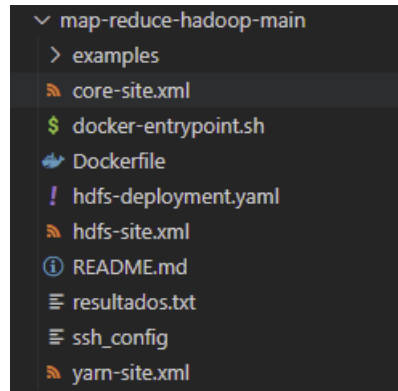


Figura 5: Ficheros ejecutados por el archivo Dockerfile

El archivo por decir así, es el archivo Dockerfile, este archivo contiene una serie de comandos que ejecutan los demás archivos mostrados en la Figura 4, estos archivos ejecutados por el Dockerfile son los encargados de crear el entorno de docker en el que se aloja hadoop.

4.1. Mapper

En la solución que fue implementada se leerá todo lo que es recibido por consola, asumiendo que los archivos de texto a leer fueron llamados mediante un *cat* con anterioridad. Para luego, detectar el número de documento, que fue ingresado al inicio de cada texto para así saber en cuál estoy posicionado, esto debido a que Hadoop no leerá todos los archivos por turnos, sino que leerá todos los archivos al mismo tiempo y por separado, es decir, de manera distribuida.

Una vez detectado el número de texto, se realiza un split que separará todas las palabras por cada línea leída, lo que permitirá recorrerlas una por una y así luego entregar o imprimir los resultados finales. Para cada palabra se retorna su llave(palabra), número de texto asociado y un **1** que representa a que es una palabra, ya que esto se repite por todas las palabras.

```
import sys
import re
import os

txt = 1
info = list()
for line in sys.stdin:
    line = re.sub(r'\W+', ' ', line.strip())
    words = line.split('START')
    if(len(words) > 1 ):
        txt = words[0]
        txt = int(txt)
    words = line.split()
    for word in words:
        print('{} {} {}'.format(word,txt,1))
```

Figura 6: Código: Mapper

4.2. Reducer

Primero, para esta implementación se pensó como es que los datos son recibidos desde el mapper, los cuales indican una llave, el texto y la cantidad. Para poder trabajar esto se creó una clase info que almacenará un texto y la cantidad de la palabra, esto pensando que será utilizado junto un diccionario que le otorgará la palabra o llave correspondiente. Esta clase posee todos sus métodos asociados.

```
class info:
    def __init__(self, texto, rep):
        self.texto = texto
        self.rep = rep

    def getTexto(self):
        return self.texto

    def getRep(self):
        return self.rep

    def setTexto(self, texto):
        self.texto = texto

    def setRep(self, rep):
        self.rep = rep

    def setTextoRepeticion(self, texto , rep):
        self.texto = texto
        self.rep = rep

    def getData(self):
        return (self.texto , self.rep)
```

Figura 7: Código:Clase info

Luego, se empezó a realizar el procesamiento de datos dentro de este reducer, en donde se leerá por consola el mapper como se mencionó anteriormente, donde primero se realiza la limpieza y segmentación de la entrada.

Primero, se crea un diccionario en donde se almacena toda la información, este será llamado como dPalabras, luego se entra las primeras condiciones en donde tenemos un if que pregunta si se encuentra esta palabra dentro del diccionario y en caso contrario esta es creada o ingresada dentro del diccionario.

Si es que esta palabra se encuentra dentro del diccionario, significará dos cosas, la primera es que la palabra se encontró repetida dentro de un mismo archivo de texto o si esta es necesario empezar a contarla dentro de otro archivo de texto.

Las condiciones funcionan de la siguiente manera, la primera condición analiza el texto en el que estoy parado, y si la cantidad de datos dentro de la llave asociada es mayor al largo de texto, ingreso a ese texto para aumentarles las repeticiones.

Luego, la segunda condición, revisa el ultimo texto que fue ingresado, si la nueva palabra que se encontró pertenece a este, le aumento el contador en uno, para simular que se sumó una nueva repetición.

Para terminar significa que donde estoy parado hace referencia a un texto que nunca fue ingresado o registrado, es por eso que lo agrego a la lista con la dirección del nuevo texto.

```
def main():
    dPalabras = dict(list())

    for lines in sys.stdin:

        tmp = lines.replace("\n", "").split('\t')
        #print(tmp)
        tmp = tmp[0].split()
        #print(tmp)
        tmp[1] = int(tmp[1])
        tmp[2] = int(tmp[2])
        if tmp[0] in dPalabras.keys():
            if (tmp[1]-1 <= len(dPalabras[tmp[0]])-1):
                dPalabras[tmp[0]][tmp[1]-1].setRep(dPalabras[tmp[0]][tmp[1]-1].getRep() +1 )
            elif (dPalabras[tmp[0]][-1].getTexto() == tmp[1]):
                dPalabras[tmp[0]][-1].setRep(dPalabras[tmp[0]][-1].getRep() +1 )
            else:
                dPalabras[tmp[0]].append(info(tmp[1], tmp[2]))

        else:
            dPalabras[tmp[0]] = []
            dPalabras[tmp[0]].append(info(tmp[1], tmp[2]))
```

Figura 8: Código: Funcionamiento Reducer

Una vez todos los datos fueron procesados, estos son impresos y escritos, tanto por consola como en un archivo de texto con el formato mostrado en la sección base de este informe.

```

arc = open('resultados.txt','w')
arc.write("Word\t[(Document,Count)\n")
for key in dPalabras.keys():
    count = 0
    for datos in dPalabras[key]:
        if count == 0 :
            arc.write('%s\t%s'%(key,datos.getData()))
            print('%s\t%s'%(key,datos.getData()),end=' ',sep='')
            count = 1
        else:
            arc.write(' (%s, %s) '%(datos.getData()))
            print(' (%s, %s) '%(datos.getData()),end=' ',sep='')

arc.write("\n")
print("\n",end=' ',sep='')

main()

```

Figura 9: Código: Impresión de Reducer

4.3. getJson

Una vez los datos fueron procesados mediante el reducer, significa que ya tenemos todo lo necesario para empezar a generar un segundo procesamiento, pero para esto es necesario generar una base de datos o un lugar en donde se puedan almacenar los datos. Para esto, se creó un archivo **json** que permite almacenar los datos respectivos.

Estos datos, fueron recogidos mediante la consola, tal como fue en los códigos anteriores, los cuales son tomados por la consola y luego limpiados. Posterior a esto, se crearon diccionarios con todas las palabras y dentro de estas palabras, se crearon arreglos que contuvieran los diccionarios, los cuales dentro tienen la información del texto asociado y los contadores asociados a cada texto.

Una vez procesados los datos, estos son ingresados dentro de nuestro nuevo archivo json creado con las llaves asociadas y con la comodidad que otorga la librería json a python.

```

5  dicc = {}
6  mat= list()
7  fl = True
8  arr = [0,0,0,0,0]
9  for line in sys.stdin:
10     if(fl):
11         fl = False
12         continue
13     else:
14         info = line.strip().split()
15         if(len(info) == 3):
16             info[1] = info[1].replace('(','')
17             info[1] = info[1].replace(',','')
18             info[2] = info[2].replace(')','')
19             dicc[info[0]] = {'text':int(info[1]), 'count':int(info[2])}

```

Figura 10: Código:

```

20     else:
21         tmp = list()
22         for values in info:
23             if (values == info[0]):
24                 continue
25             else:
26                 values = values.replace('(', '')
27                 values = values.replace(')', '')
28                 values = values.replace(' ', '')
29                 tmp.append(values)
30         imp = 0
31         count = 1
32         texto = 0
33         for datos in tmp:
34             if(count%2 != 0):
35                 texto = datos # valor en el que estoy parado
36                 count = 2
37             else:
38                 contador = datos
39                 dicaux = {'text':int(texto), 'count':int(contador)}
40                 dicc.setdefault(info[0], []).append(dicaux)
41                 count = 1

```

Figura 11: Código:

```

44     outf = open('bdd.json', 'w')
45     json.dump(dicc, outf, indent= 3, sort_keys= False)
46     outf.close()

```

Figura 12: Código:

Los resultados entregados por este código fueron los siguientes:

```

{
  "Mathematical": {
    "text": 4,
    "count": 1
  },
  "Olympiad": {
    "text": 4,
    "count": 4
  },
  "Informatics": {
    "text": 4,
    "count": 1
  },
  "..."
}

```

Figura 13: Resultados, archivo json

4.4. Buscador

El buscador implementado empieza pidiendo por consola la palabra o las palabras a buscar, al leer lo escrito por consola lee el archivo json que se creo en el apartado anterior conteniendo la información en la variable f. Se pregunta con una condición if si la palabra ingresada por consola esta compuesta por mas de una palabra. En el caso que este compuesta por mas de una palabra, se separa por palabra y se procesa cada palabra con el ciclo for. Pregunta con la condición if si la palabra contiene mas de un texto asociado mediante la función isSingular, esta función revisa si la clase de la palabra registrada en el archivo json es de tipo list y contiene mas de un documento asociado con su respectiva repetición, si cumple entonces retorna un false, también revisa si la clase de la palabra registrada en el archivo json es de tipo dict y contiene un único documento asociado a la palabra, si cumple entonces retorna un true.

```
4 def isBigger(str):
5     cont = False
6     for caracter in str:
7         if caracter == ' ':
8             cont = True
9     return cont
10 def sortSecond(arr):
11     return arr[1]
12
13 def isSingular(dicc):
14     bol = True
15     dic = {}
16     arr = []
17     if(len(dicc) > 1 and type(dicc) == type(arr)):
18         bol = False
19     elif(len(dicc) == 2 and type(dicc) == type(dic)):
20         bol = True
21     return bol
22 def is_valid(f):
23     try:
24         print(f)
25     except KeyError:
26         return False
27
28
29 def buscar():
30     texto = input('Ingrese lo que desea buscar:')
31     big = False
32
33     if(isBigger(texto)):
34         big = True
35     relevance= []
36     pathresult = []
```

Figura 14: Código buscador de documentos

Volviendo al proceso, en el caso que la función isSingular devuelva true, entonces ingresa y busca en el archivo json la palabra, al encontrarla la agrega con la función append a un array llamado tmp en el que simplemente almacenara la información correspondiente al id de referencia del documento en el que aparece la palabra y cuantas veces esta repetida en el. Esta información es insertada al array revelance el cual se le realiza un ordenamiento con el método sort en relación al numero de veces que se repite la palabra en el texto, posterior a esto mediante la función get_path se obtiene el path donde esta alojado el documento en el que aparece la palabra. Esta función simplemente busca el documento mediante el id de referencia de este en una lista con contiene todos los path de los documentos extraídos gracias a la función save_path, esta ultima función recorre cada documento alojado en la ruta Wikipedia/Wikipedia/Documentos1 gracias al método walk de la librería os y mediante un segundo for se recorre la variable files, la cual contiene el nombre de los documentos, por ultimo se ocupa el método path.abspath(os.path.join()) de la librería os y se guardan los path absolutos de los documentos. Lo mismo se realiza para los documentos alojados en la ruta Wikipedia/Wikipedia/Documentos2 y por ultimo se limpia el array relevance ya que no hay mas información asociada a esa palabra.

En caso contrario, si la función `isSingular` retorna `false`, entonces significa que la palabra contiene mas de un documento asociado a ella, entonces se ingresa a cada registro, el cual contiene el id de referencia del documento donde aparece la palabra y la cantidad de veces que se repite, se repite el proceso de agregar al array `tmp` cada atributo del registro, luego se agrega al array `relevance`, se aplica el ordenamiento con el método `sort` mediante la cantidad de repeticiones de la palabra por documento, se obtiene el path de cada documento con la función `get_path` y por ultimo se limpia el array `relevance`.

```
37     print('Buscado:', texto)
38     with open('bdd.json') as file:
39         f = json.load(file)
40
41     if(big):
42         texto = texto.split()
43         for palabras in texto:
44             try:
45                 if(isSingular(f[palabras])):
46                     tmp = []
47                     tmp.append(f[palabras]['text'])
48                     tmp.append(f[palabras]['count'])
49                     relevance.append(tmp)
50                     relevance.sort(key=sortSecond, reverse = True)
51                     print(palabras, relevance)
52                     get_path(texto, relevance)
53                     relevance.clear()
54                     continue
55
56                 for duplas in f[palabras]:
57                     tmp = []
58                     tmp.append(duplas['text'])
59                     tmp.append(duplas['count'])
60                     relevance.append(tmp)
61                     relevance.sort(key=sortSecond, reverse = True)
62                     print(palabras, relevance)
63                     get_path(texto, relevance)
64                     relevance.clear()
65             except:
66                 print("La palabra ", palabras, 'no está en nuestra base.')
67                 continue
```

Figura 15: Código buscador de documentos

Ahora, en el caso que la palabra ingresada a buscar por consola contenga solamente una palabra, nuevamente se pregunta mediante una condición `if` si la palabra contiene un único documento asociado a ella. Si la palabra contiene mas de un documento asociado, entonces se ingresa a cada registro y se repite el proceso de guardar en un array llamado `tmp` cada atributo, el id y cantidad de veces que se repite, se agrega al array `relevance` para luego ser ordenado con el método `sort` en base a la cantidad de repeticiones y con la función `get_path` obtener el path de los documentos en los que aparece la palabra. En caso contrario, que la palabra contenga un solo documentos asociado a ella, entonces se repite el proceso recién explicado pero solo para el registro único.

```

68         else:
69             try:
70                 if(isSingular(f[texto]) == False):
71                     for duplas in f[texto]:
72                         try:
73                             tmp = []
74                             tmp.append(duplas['text'])
75                             tmp.append(duplas['count'])
76                             relevance.append(tmp)
77                             relevance.sort(key=sortSecond, reverse = True)
78                         except:
79                             print("La palabra ", texto, 'no está en nuestra base.')
80                             continue
81             else:
82                 tmp = []
83                 tmp.append(f[texto]['text'])
84                 tmp.append(f[texto]['count'])
85                 relevance.append(tmp)
86                 relevance.sort(key=sortSecond, reverse = True)
87                 print(texto,relevance)
88                 get_path(texto,relevance)
89             except KeyError:
90                 print("La palabra ", texto, 'no está en nuestra base.')
91             ##### MENU #####
92
93         urls = save_path()
94         while(True):
95             print('¿Que texto desea abrir? Escriba exit para salir')
96             inp = input('Ingrese numero:')
97
98             if(inp == 'exit'):
99                 break
100             elif(inp.isnumeric()):

```

Figura 16: Código buscador de documentos

```

100         elif(inp.isnumeric()):
101             path = str(urls[int(inp)-1])
102             print(path)
103             arc = open(path,'r',encoding='utf8')
104             print(arc.read())
105             arc.close()
106         else:
107             print('Entrada como el pico, reinicia')
108
109
110     def get_path(texto, relevance):
111         path = save_path()
112         for values in relevance:
113             print(values)
114             tex = values[0]
115             print('url', tex , ':',path[tex-1])
116
117
118     def save_path():
119         dir = {}
120         i = 0
121         for root,dirs, files in os.walk(r'Wikipedia/Wikipedia/Documentos1'):
122             for i in range(i, len(files)):
123                 dir[i] = (os.path.abspath(os.path.join(root,files[i])))
124
125         for root, dirs , files in os.walk(r'Wikipedia/Wikipedia/Documentos2'):
126             for j in range(0, len(files)):
127                 dir[j+i+1] = (os.path.abspath(os.path.join(root,files[j])))
128         return dir
129
130     buscar()

```

Figura 17: Código buscador de documentos

Finalmente, se crea un pequeño menú en el que se espera el numero del documento que se quiere leer. Se crea una variable llamada url, la cual contiene todos los path de los documentos gracias a la función ya mencionada y explicada save_path, el menú esta dentro de un ciclo while, se solicita ingresar el numero de documento que desea leer o escribir exit si desea salir del menú. Al ingresar el numero de documento deseado, se extrae de url el path asociado al numero de documento solicitado, se abre tal documento y se muestra para ser leído.

```
PS C:\Users\abelb\OneDrive\Documentos\Tarea_3_SD\Tarea_3_SD\map-reduce-hadoop-main\examples> python buscador.2.py
Ingrese lo que desea buscar:Alexis Cristiano UCB palabranoeexistente
Buscado: Alexis Cristiano UCB palabranoeexistente
Alexis [[1, 6], [3, 1], [5, 1]]
url 1 : C:\Users\abelb\OneDrive\Documentos\Tarea_3_SD\Tarea_3_SD\map-reduce-hadoop-main\examples\Wikipedia\Wikipedia\Documentos1\Alexis_Sanchez.txt
url 3 : C:\Users\abelb\OneDrive\Documentos\Tarea_3_SD\Tarea_3_SD\map-reduce-hadoop-main\examples\Wikipedia\Wikipedia\Documentos1\Arturo_Vidal.txt
url 5 : C:\Users\abelb\OneDrive\Documentos\Tarea_3_SD\Tarea_3_SD\map-reduce-hadoop-main\examples\Wikipedia\Wikipedia\Documentos1\Cristiano_Ronaldo.txt
Cristiano [[5, 14]]
url 5 : C:\Users\abelb\OneDrive\Documentos\Tarea_3_SD\Tarea_3_SD\map-reduce-hadoop-main\examples\Wikipedia\Wikipedia\Documentos1\Cristiano_Ronaldo.txt
UCB [[4, 1], [7, 1]]
url 4 : C:\Users\abelb\OneDrive\Documentos\Tarea_3_SD\Tarea_3_SD\map-reduce-hadoop-main\examples\Wikipedia\Wikipedia\Documentos1\Chile.txt
url 7 : C:\Users\abelb\OneDrive\Documentos\Tarea_3_SD\Tarea_3_SD\map-reduce-hadoop-main\examples\Wikipedia\Wikipedia\Documentos2\Espana.txt
La palabra palabranoeexistente no está en nuestra base.
¿Que texto desea abrir? Escriba exit para salir
Ingrese numero:[]
```

Figura 18: Resultado buscador de documentos

```
Ingrese numero:3
C:\Users\abelb\OneDrive\Documentos\Tarea_3_SD\Tarea_3_SD\map-reduce-hadoop-main\examples\Wikipedia\Wikipedia\Documentos1\Arturo_Vidal.txt
3 START Arturo Erasmo Vidal Pardo (Spanish pronunciation: [aɾˈtuɾo eɾaˈzmo βiˈðal ˈpaɾdo]; born 22 May 1987) is a Chilean professional footballer who plays as a midfielder for Campeonato Brasileiro Série A club Flamengo and t
he Chile national team. His displays during his time at Juventus led him to be nicknamed El Guerrero ("The Warrior"), Ray Arturo ("King Arthur") and La Piranha by the Italian press due to his hard-tackling and aggressive, te
nacious style of play.Vidal started his career with Colo-Colo, where he won three Chilean Primera División titles. He relocated to Europe, where he joined Bundesliga club Bayer Leverkusen and played there for four seasons. He
then moved to Juventus in 2011, where he became widely recognized as one of the best midfielders in world football. At Juventus, he won the Scudetti in all four of his seasons and also was integral for them in reaching the 2
015 UEFA Champions League Final. Vidal was named to the ten-man shortlist for the 2015 UEFA Best Player in Europe Award following his performances. On 28 July 2015, Vidal returned to the Bundesliga, joining Bayern Munich and
won three consecutive Bundesliga titles. After three years at Munich, he signed for La Liga giants Barcelona, where he won his eighth straight league title. In 2020 he returned to the Serie A to sign for Inter Milan, where h
e won yet another Serie A title and Coppa Italia.Vidal has earned over 130 caps for the Chile national team since his debut in 2007, playing in the 2011, 2015, 2019 and 2021 Copa América tournaments, as well as the Copa Améri
ca Centenario, the 2010 and 2014 FIFA World Cups, and the 2017 FIFA Confederations Cup, helping his nation to Copa América victory in 2015 and 2016.

== Early life ==
Vidal was born in San Joaquín, a working class commune in the Chilean capital Santiago. His talent was noticed by his uncle, and he later joined the youth squads of local Primera División club Colo-Colo.

== Club career ==

=== Colo-Colo ===
Vidal's professional debut came in the first leg of the 2006 Torneo Apertura final against arch-rivals Universidad de Chile. Vidal came on as a late substitute for Gonzalo Fierro. Colo-Colo would go on to win the game 2-1 and
win the championship as well. In the following season (Torneo Clausura) he became a more important part of the squad and would lead Colo-Colo to their second championship win in a row. Vidal scored three goals in Colo-Colo's
Copa Sudamericana 2006 campaign. His good showing caught the eye of scouts from various European clubs.

=== Bayer Leverkusen ===
The 2007 Apertura Tournament was Vidal's last with Colo-Colo as he left for Bayer Leverkusen in the summer. Bayer had tracked his progress for some time and his good showing at the U-20 World Cup that year convinced Bayer Lev
erkusen director of football Rudi Völler to make the trip to Chile to convince the 20-year-old to sign. The two clubs then agreed on a fee of US$11 million with Bayer Leverkusen, paying $7.7 million for 70% ownership of his c
ontract. His transfer broke the previous national record of Matías Fernández's $9 million transfer to Villarreal.Vidal missed the first game of the season through injury but was soon thrust into the starting line-up and made
his debut on 19 August 2007 in the away loss against Hamburger SV. He started in half the season's games and scored his first goal for the club just three games into his Bayer career. He was ever present for the 2008-09 seaso
n and played a vital role in Bayer's run to the DFB-Pokal final. On 8 March, he suffered a concussion during the game against VfL Bochum and was out for a month. Upon his return, he scored a goal to break the deadlock in the
4-1 semifinal win over Mainz 05 in the DFB-Pokal, but Bayer eventually lost to Werder Bremen in the final.
The 2010-11 season would be Vidal's last with Bayer. He helped the club to a runner-up finish in the Bundesliga and topped the assist charts for his club with 11 assists, which was joint second in the League. He also contribu
ted two goals in the club's run to the round of 16 of the UEFA Europa League.

=== Juventus ===

==== 2011-12 season ====
After a good 2010-11 season, Vidal was linked with various clubs, including Bayer's Bundesliga rivals Bayern Munich. On 22 July 2011, however, Vidal joined Serie A club Juventus for €10.5 million on a five-year contract. He m
ade his competitive debut in the opening league game of the season against Parma on 11 September 2011, coming on as a second-half substitute for Alessandro Del Piero; he marked his first appearance with a goal six minutes aft
er his introduction in Juventus' 4-1 win at the club's new stadium. It was initially speculated that he would compete with Claudio Marchisio for a spot alongside Andrea Pirlo, but Juventus manager Antonio Conte instead played
all three effectively in a three-man midfield in his 3-5-2 formation. Vidal was an integral part of the 2011-12 Scudetto-winning side that went undefeated the entire season. He contributed seven league goals and three assist
```

Figura 19: Resultado buscador de documentos

Referencias

- [1] [Github de la tarea](#)
- [2] [Github de referencia](#)
- [3] [Video de demostracion](#)