23-10-2017

FECHAS IMPORTANTES

Entrega parcial 1: Viernes 3 Noviembre (vía webcursos)
Entrega parcial 2: Viernes 17 Noviembre (vía webcursos)
Entrega final: Domingo 26 Noviembre (vía webcursos)

El nombre del archivo que se sube a webcursos debe llamarse Apellido1_Apellido2_EntregaX.c. Donde Apellido1 y Apellido2 es el apellido de cada uno de los integrantes del grupo y X es el número de entrega. Si el archivo no tiene ese nombre, se descuenta 0.5 puntos en el control correspondiente.

EVALUACIÓN

Análogamente al caso de la Tarea 1, las entregas parciales no tienen evaluación directa. Se evalúan indirectamente a través de los controles. La nota de la Tarea 2 se basará exclusivamente en la entrega final. Aunque un grupo haya subido todas las entregas parciales, sino realiza la entrega final, tiene un 1.0 en la Tarea 2. El archivo de la tarea final debe llevar el nombre Apellido1_Apellido2_EntregaFinal.c. Los grupos que no cumplan con esta condición, tendrán un punto menos en la tarea.

GRUPOS

Siguen los mismos grupos de 2 personas formados para la Tarea 1.

GOOGLITO

En esta tarea cada grupo construirá el programa ***Googlito*** que simulará un sistema simple de búsqueda del tipo ***Google***. En vez de buscar en Internet, la búsqueda se realizará sobre un conjunto de documentos almacenados en el mismo computador en el que se ejecuta ***Googlito***. Para esto, se deberá realizar un programa que le solicite al usuario un string de palabras a buscar (la *consulta)*, y el programa deberá mostrar una lista de documentos (almacenados) más *relevantes* (ordenados de mayor a menor relevancia) ó "cercanos" a dicha consulta.

El algoritmo para construir ***Googlito*** se compone de los siguientes cuatro pasos:

1) Recolectar los documentos de texto puro sobre los que se realizarán las búsquedas.
2) Crear los vectores de frecuencia:
   a) Leer las palabras de cada documento almacenado para crear el vocabulario.

b) Convertir cada documento almacenado en un vector de *frecuencias de palabras* (qué palabra aparece cuántas veces en el documento).

3) Comparar la consulta con los vectores de frecuencia creados:
   a) Convertir la consulta de un usuario también en un vector de *frecuencias*.
   b) Comparar el vector de la consulta con cada uno de los vectores de los documentos con el fin de obtener un valor de similitud.
4) Mostrar los resultados de la búsqueda:
   a) Ordenar los documentos según su nivel de similitud con el vector de la consulta.

A continuación se muestra un ejemplo pequeño para ilustrar la ejecución de cada paso del algoritmo anterior:

Paso 1) Recopilación de documentos: Supón que se trabajará solo 3 documentos (en la tarea usaremos 50):

"doc0.txt": hola que tal juan que te pasa
"doc1.txt": pasa juan y entro lento que pasa
"doc2.txt": el tiempo pasa lento

Paso 2.a) Creación de vocabulario: Eliminando las palabras de 3 ó menos letras, estos 3 documentos forman un vocabulario total de 6 palabras: **hola, juan, pasa, entro, lento, tiempo**. Estas palabras se almacenan en el vector **Vocabulario**:

[0]	[1]	[2]	[3]	[4]	[5]
hola	Juan	pasa	entro	lento	tiempo

Paso 2.b) Creación de matriz de frecuencias: Usando el vector **Vocabulario**, se cuenta el número de veces que cada una de las palabras aparece en cada documento y se crea un vector de *frecuencias de palabras* para cada documento. Para no tener que manejar un vector por cada documento, esta información se condensará en una matriz (matriz con tantas filas como palabras tiene el vocabulario y tantas columnas como documentos se analizan). En este ejemplo, la matriz es de 6x3 (6 palabras , 3 documentos):

	Doc0 [0]	Doc1 [1]	Doc2 [2]
hola -> [0]	1	0	0
Juan -> [1]	1	1	0
pasa -> [2]	1	2	1
entro -> [3]	0	1	0
lento -> [4]	0	1	1
tiempo -> [5]	0	0	1

Paso 3.a) Creación del vector de la consulta: Supón que la consulta es "**juan lento**". Luego, el vector de frecuencia de palabras de la consulta es:

hola -> [0]	0
Juan -> [1]	1
pasa -> [2]	0
entro -> [3]	0
lento -> [4]	1
tiempo -> [5]	0

Paso 3.b) Cálculo de similitud: Se calcula la similitud del vector de la consulta con cada uno de los vectores de los documentos. Para esto, se aplicará la función coseno entre vectores (los detalles de cómo se realiza esto no son relevantes en este momento). Como resultado, se obtiene la similitud con cada documento:

Doc0 [0]	Doc1 [1]	Doc2 [2]
0.408	0.53	0.408

Paso 4) Despliegue de resultados: Los documentos se despliegan en orden decreciente de similitud. En el caso de este ejemplo, se debería mostrar que la búsqueda arrojó los siguientes documentos relevantes (suponiendo un máximo de 3 relevantes):

- Documento 1 (0.53)
- Documento 0 (0.408)
- Documento 2 (0.408)

ENTREGA PARCIAL 1

La ***Entrega Parcial 1*** consiste en completar los primeros 2 pasos del algoritmo de ***Googlito***. A continuación se entregan los detalles de implementación de cada paso.

Paso 1. Preparar los datos: se deberá recopilar *manualmente* (NO debes construir un programa para esto) 50 archivos de textos puro de no más de 1 página cada uno (ej. los ensayos que ha escrito para "Civilización Contemporánea", que también puedes compartir/copiar desde otros grupos y/o compañeros) y grabarlos en un directorio llamado **documentos**. Los archivos deben ser de texto puro. Es decir, si los escribieron en Word, deben "Guardar como" archivo con extensión .txt. Luego, a cada archivo se le deberá colocar el nombre "**DocN.txt**", donde **N** es el número del documento en el rango 0 a 49 (Ej. "**Doc0.txt**", "**Doc1.txt**", etc).

Paso 2. Generar la matriz de frecuencias de palabras. Se deberá implementar las siguientes funciones:

> **Paso 2.a). Creación del vocabulario.** Se debe construir la función:
> `void CrearVocabulario(char Vocabulario[MaxPal][LargoMaxPal])`
> que lee todos los archivos del directorio **documentos** y genera el *vocabulario total* de palabras. Este vocabulario se almacenará en el arreglo de caracteres llamado **Vocabulario** de tamaño máximo **MaxPal** con cada palabra de largo **LargoMaxPal** (ej. **MaxPal** es 1000 y **LargoMaxPal** es 20). Así **Vocabulario,** contendrá todas las palabras que ocurren en todos los archivos (con un máximo de **MaxPal**). Si una palabra ya existe en **Vocabulario** está no se debe agregar nuevamente. Para evitar agregar palabras sin importancia (ej. proposiciones, artículos, etc), sólo se deben agregar palabras cuyo largo es mayor que 3 caracteres.
>
> **Paso 2.b). Crear la matriz de frecuencias.** Se debe construir la función:
> `void CrearMatrizFrecuencias(char Vocabulario[MaxPal][LargoMaxPal], int Matriz[MaxPal][MaxDoc])`
> que genera una matriz de enteros de **NumPal x NumDocs** que contiene el *número de veces* (frecuencia) que cada palabra de **Vocabulario** aparece en cada archivo. La lista de frecuencias de palabras de cada archivo se debe obtener llamando a la función `GenerarListaFrecuencias(..)`, la cual se define como:

`void GenerarListaFrecuencias(char NombreArchivo[], char Vocabulario[MaxPal][LargoMaxPal], int ListaFrec[MaxPal])`

que lee el archivo **NombreArchivo**, y genera la lista **ListaFrec** que contiene el número de veces que aparece cada palabra de **Vocabulario**.

La estructura del programa de la ***Entrega parcial 1*** con las funciones principales, se debería ver como se muestra a continuación:

```
#include <stdio.h>
#include <cs50.h>
#include <string.h>


// El programa supone que los archivos de texto recopilados ya están
// en el directorio "documentos"

const int MaxDoc=50;        // Máximo Núm. de documentos
const int MaxPal=1000;   // Máximo núm. de palabras en Vocabulario
const int LargoMaxPal=20;  // Largo máximo de una palabra
const int LargoMinPal=4;   // Largo mínimo de una palabra
int NumPal;           // Núm. de palabras actuales del Vocabulario


// Función correspondiente a Paso 2

void GenerarMatrizDatos(char Vocabulario[MaxPal][LargoMaxPal],int
Matriz[MaxPal][MaxDoc]){

    CrearVocabulario(Vocabulario);
    CrearMatrizFrecuencias(Vocabulario,Matriz);

}

void MostrarMatriz(int Matriz[MaxPal][MaxDoc]){
int i,j,nf,nc;

    printf("Dado que la matriz no cabe en pantalla, debe
especificar el tamaño a mostrar..\n");
    printf("Ingrese núm de filas: ");
    nf = GetInt();
    printf("Ingrese núm de columnas: ");
```

```
    nc = GetInt();
    printf("Matriz resultante con dicho tamaño: \n");
    for(i=0; i < nf ; i++){
      for(j=0; j < nc ; j++)
           printf("(%i)",Matriz[i][j]);
      printf("\n");
    }
}


int main(void)
{
char Vocabulario[MaxPal][LargoMaxPal];
int Matriz[MaxPal][MaxDoc];

      NumPal=0;  // num de palabras actuales del Vocabulario
      GenerarMatrizDatos(Vocabulario, Matriz);
      MostrarMatriz(Matriz);
}
```