search-simulator

Generated by Doxygen 1.13.0

Chapter 1

Simulador de Sistema de Recuperación de Información

Este proyecto implementa un **simulador de un sistema de recuperación de información**, utilizando estructuras de datos como grafos, listas enlazadas y tablas hash. El sistema combina dos técnicas principales: **índice invertido** y **PageRank**, para gestionar datos de manera eficiente.

1.1 Resumen

El simulador permite:

- Construir un grafo dirigido que representa la conexión entre documentos.
- Crear un índice invertido que facilita búsquedas rápidas por términos en los documentos.
- Calcular la relevancia de los documentos mediante el algoritmo de PageRank.
- Realizar búsquedas de términos con resultados ordenados por relevancia.

Este trabajo fue programado en **C**.

1.2 Características

- **Generación automática de documentos simulados.**
  Se crean archivos de texto con contenido generado dinámicamente.
- **Construcción de un grafo dirigido.**
  Cada documento es un nodo, y los enlaces entre ellos representan las conexiones.
- **Índice invertido para búsquedas eficientes.**
  Permite localizar documentos que contienen un término específico.
- **Algoritmo PageRank.**
  Calcula la importancia relativa de los documentos basándose en las conexiones entre ellos.
- **Visualización de resultados.**
  Los resultados de PageRank y las búsquedas se imprimen en consola.

1.3 Uso del Programa

Antes de ejecutar el programa, es necesario compilarlo y prepararlo mediante las siguientes instrucciones en la línea de comandos:

1.4 Compilación del proyecto

1. **Compilación del programa**:
   Ejecute el comando `make` en el directorio raíz del proyecto. Esto utilizará el `Makefile`.

   ```
   make
   ```

2. **Ejecución del programa** Ahora, una vez hecho el `make`, debe ejecutar `make run`, que tiene definido parametros adicionales (esto se encuentra en el Makefile), estos parametros adicionales nos permiten colocar la cantidad `-d` y una ayuda del programa `-h`.

1.5 Generación de documentación con Doxygen

1. **Generar la documentación**

   Ejecute el comando `make dxygn` en el directorio raíz del proyecto para generar la documentación automática en formato HTML y LaTeX.

2. **Compilar en formato PDF**

   Una vez generados los archivos con Doxygen, puede convertir la documentación en un archivo PDF utilizando LaTeX. Para hacerlo, ejecute `make ltx`.

3. **Ubicación del archivo PDF**

   El archivo PDF generado estará disponible en el directorio docs/latex/ o en la ubicación configurada en el `Makefile`.

4. **Visualización de la documentación HTML**

   Los archivos HTML generados se encontrarán en el directorio docs/html/. Para visualizar la documentación, abra el archivo index.html con su navegador.

1.6 Opción de vista adicional

1. Dentro del programa, hay una vista adicional que permite visualizar las palabras almacenadas en el índice invertido, junto con la cantidad de veces que se repiten. Esto se puede lograr mediante la función `print_inverted_index(index);`. Idealmente, esta función puede añadirse antes o después de la función print_search_word.

1.7 Salida del programa

Al ejecutar `make run`, el programa generará una salida en consola que incluye:

- El **índice invertido** construido, mostrando en qué documentos se encuentra cada palabra.
- El **grafo de enlaces**, indicando cómo están conectados los documentos entre sí.
- Los valores de **PageRank** calculados para cada documento, reflejando su relevancia.

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 DocumentMapping Struct Reference

Estrucutra de un mapeo de documentos.

`#include <graph.h>`

Public Attributes

- char name [MAX_NAME_DOC]
- int doc_id

4.1.1 Detailed Description

Estrucutra de un mapeo de documentos.

```
typedef struct DocumentMapping
{
     char name[MAX_NAME_DOC];
     int doc_id;
} DocumentMapping;
```

Definition at line 86 of file graph.h.

4.1.2 Member Data Documentation

4.1.2.1 doc_id

`int DocumentMapping::doc_id`

Definition at line 89 of file graph.h.

4.1.2.2 name

```
char DocumentMapping::name[MAX_NAME_DOC]
```

Definition at line 88 of file graph.h.

The documentation for this struct was generated from the following file:

- incs/graph.h

4.2 Graph Struct Reference

Estrucutra de un grafo.

```
#include <graph.h>
```

Public Attributes

- Node * output_adjacent_list [MAX_DOCS]
- Node * input_adjacent_list [MAX_DOCS]
- DocumentMapping mapping_docs [MAX_DOCS]
- int total_docs

4.2.1 Detailed Description

Estrucutra de un grafo.

```
typedef struct Graph
{
     Node *output_adjacent_list[MAX_DOCS];
     Node *input_adjacent_list[MAX_DOCS];
     DocumentMapping mapping_docs[MAX_DOCS];
     int total_docs;
} Graph;
```

Definition at line 105 of file graph.h.

4.2.2 Member Data Documentation

4.2.2.1 input_adjacent_list

```
Node* Graph::input_adjacent_list[MAX_DOCS]
```

Definition at line 108 of file graph.h.

4.2.2.2 mapping_docs

```
DocumentMapping Graph::mapping_docs[MAX_DOCS]
```

Definition at line 109 of file graph.h.

4.2.2.3 output_adjacent_list

```
Node* Graph::output_adjacent_list[MAX_DOCS]
```

Definition at line 107 of file graph.h.

4.2.2.4 total_docs

```
int Graph::total_docs
```

Definition at line 110 of file graph.h.

The documentation for this struct was generated from the following file:

- incs/graph.h

4.3 InvertedIndex Struct Reference

< Incluye la definición de las estructuras y funciones del grafo.

```
#include <inverted_index.h>
```

Public Attributes

- char word [MAX_WORD_SIZE]
- Node * docs_list
- struct InvertedIndex * next

4.3.1 Detailed Description

< Incluye la definición de las estructuras y funciones del grafo.

Estructura de un nodo en el índice invertido.

```
typedef struct InvertedIndex
{
     char word[MAX_WORD_SIZE];
     Node *docs_list;
     struct InvertedIndex *next;
} InvertedIndex;
```

Definition at line 26 of file inverted_index.h.

4.3.2 Member Data Documentation

4.3.2.1 docs_list

```
Node* InvertedIndex::docs_list
```

Definition at line 29 of file inverted_index.h.

4.3.2.2 next

```
struct InvertedIndex* InvertedIndex::next
```

Definition at line 30 of file inverted_index.h.

4.3.2.3 word

```
char InvertedIndex::word[MAX_WORD_SIZE]
```

Definition at line 28 of file inverted_index.h.

The documentation for this struct was generated from the following file:

- incs/inverted_index.h

4.4 Node Struct Reference

Estrucutra de un nodo.

```
#include <graph.h>
```

Public Attributes

- int doc_id
- struct Node * next

4.4.1 Detailed Description

Estrucutra de un nodo.

```
typedef struct Node
{
     int doc_id;
     struct Node *next;
} Node;
```

Definition at line 69 of file graph.h.

4.4.2 Member Data Documentation

4.4.2.1 doc_id

```
int Node::doc_id
```

Definition at line 71 of file graph.h.

4.4.2.2 next

```
struct Node* Node::next
```

Definition at line 72 of file graph.h.

The documentation for this struct was generated from the following file:

- incs/graph.h

Chapter 5

File Documentation

5.1 incs/doc.h File Reference

Prototipos de funciones para la creación de texto y archivos.

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

Functions

- void generate_text_files (int, int)
  Genera archivos txt simulando páginas web.
- void generate_random_text (FILE ∗, const char ∗, int, int, int, int ∗)
  Genera texto aleatorio dentro de cada archivo web.

5.1.1 Detailed Description

Prototipos de funciones para la creación de texto y archivos.

Date

18-11-2024

Authors

Miguel Loaiza, Diego Sanhueza, Miguel Maripillan y Felipe Cárcamo

Contiene los prototipos de las funciones dedicadas a la creación de texto (aleatorio) y archivos (webs).

Definition in file doc.h.

5.1.2 Function Documentation

5.1.2.1 generate_random_text()

```
void generate_random_text (
            FILE * doc,
            const char * doc_name,
            int num_docs,
            int num_characters,
            int current_doc,
            int * links)
```

Genera texto aleatorio dentro de cada archivo web.

Parameters

doc	Archivo a escribir.
doc_name	Nombre del archivo.
num_docs	Número de documentos.
num_characters	Número de caracteres.
current_doc	Documento actual.
links	Conexiones entre documentos.

Genera texto aleatorio dentro de cada archivo web.

Parameters

doc	Archivo a escribir
doc_name	Nombre del archivo
num_docs	Número de documentos
num_characters	Número de caracteres
current_doc	Documento actual
links	Conexiones entre documentos

Genera texto aleatorio dentro de cada archivo web (letras entre A y Z).

```
for (int i = 0; i < num_characters; i++)
{
     char letter = 'A' + rand() % 26;
     fprintf(doc, "%c", letter);
     if (i < num_characters - 1)
         fprintf(doc, " ");
}
```

Asegura al menos un enlace único en cada archivo web.

```
fprintf(doc, "\nlink: doc%d", links[current_doc - 1]);
int extra_links = rand() % num_docs;
for (int i = 0; i < extra_links; i++)
{
     int link_doc = rand() % num_docs + 1;
     if (link_doc != current_doc && link_doc != links[current_doc - 1])
         fprintf(doc, "\nlink: doc%d", link_doc);
}
```

Definition at line 97 of file doc.c.

5.1.2.2 generate_text_files()

```
void generate_text_files (
             int num_docs,
             int num_characters)
```

Genera archivos txt simulando páginas web.

Parameters

num_docs	Número de documentos.
num_characters	Número de caracteres dentro del archivo.

Genera archivos txt simulando páginas web.

< Incluye la definición de las estructuras y funciones del grafo.

Generar archivos txt

Parameters

num_docs	Cantidad de archivos a generar
num_characters	Cantidad de caracteres por archivo

Verifica que el número de archivos a generar sea válido. Crea un array de enlaces para asegurar que cada documento tenga al menos un enlace. Llama a la función generate_random_text para generar el contenido de cada archivo.

```
if (num_docs <= 0 || num_docs >= 100)
{
     fprintf(stderr, "El número de archivos web a generar debe ser MAYOR a 0 y MENOR a 100.\n\n");
     exit(EXIT_FAILURE);
}
if (num_characters <= 0 || num_characters >= 50)
{
     fprintf(stderr, "El número de caracteres por archivo debe ser MAYOR a 0 y MENOR a 50.\n\n");
     exit(EXIT_FAILURE);
}
int *links = malloc(num_docs * sizeof(int));
for (int i = 0; i < num_docs; i++)
     links[i] = (i + 1) % num_docs + 1;
for (int i = 1; i <= num_docs; i++)
{
     char doc_name[MAX_NAME_DOC];
     snprintf(doc_name, sizeof(doc_name), "doc%d.txt", i);
     FILE *doc = fopen(doc_name, "w");
     if (doc == NULL)
     {
         fprintf(stderr, "Error al abrir el archivo web.\n");
         free(links);
         exit(EXIT_FAILURE);
     }
     generate_random_text(doc, doc_name, num_docs, num_characters, i, links);
   fclose(doc);
}
```

Definition at line 18 of file doc.c.

5.2 doc.h

Go to the documentation of this file.

```
00001
00009 #ifndef GENERATOR_H
00010 #define GENERATOR_H
00011
00012 #include <stdio.h>
00013
00019 void generate_text_files(int, int);
00020
00030 void generate_random_text(FILE *, const char *, int, int, int, int *);
00031
00032 #endif
```

5.3 incs/graph.h File Reference

Prototipos de funciones para la creación de los grafos.

```
#include <stdbool.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <ctype.h>
#include <dirent.h>
#include <unistd.h>
#include <time.h>
```

Classes

- struct Node

  Estrucutra de un nodo.
- struct DocumentMapping

  Estrucutra de un mapeo de documentos.
- struct Graph

  Estrucutra de un grafo.

Macros

- #define CONVERGENCE_THRESHOLD 0.0001
- #define MAX_CHARACTERS_DOC 50
- #define DAMPING_FACTOR 0.85
- #define MAX_ITERATIONS 100
- #define HASH_TABLE_SIZE 30
- #define MAX_WORD_SIZE 50
- #define MAX_NAME_DOC 20
- #define MAX_DOCS 100

Typedefs

- typedef struct Node **Node**
- typedef struct DocumentMapping **DocumentMapping**
- typedef struct Graph **Graph**

Functions

- void initialize_graph (Graph ∗)

  Inicializa el grafo.
- void add_edge (Graph ∗, int, int)

  Agrega un enlace dirigido entre dos documentos en el grafo.
- void build_graph (Graph ∗)

  Construye el grafo.
- void release_graph (Graph ∗)

  Libera la memoria del grafo (nodos).
- int count_output_links (Graph ∗, int)

  Cuenta el número de enlaces salientes de un documento.
- int count_input_links (Graph ∗, int)

  Cuenta el número de enlaces entrantes a un documento.
- int get_doc_id (Graph ∗, char ∗)

  Obtiene el ID de un documento a partir de su nombre de archivo.
- bool is_doc_name (char ∗)

  Verifica si el nombre del archivo sigue el patrón "docN.txt" con N >= 1.
- void show_graph (Graph ∗)

  Muestra el grafo.
- void generate_eps (const Graph ∗, const double ∗, const char ∗)

  Genera un archivo .eps con la representación gráfica del grafo.

5.3.1 Detailed Description

Prototipos de funciones para la creación de los grafos.

Date

18-11-2024

Authors

Miguel Loaiza, Diego Sanhueza, Miguel Maripillan y Felipe Cárcamo

Contiene los prototipos de las funciones dedicadas a la creación de grafos, las estructuras generales y macros.

Definition in file graph.h.

5.3.2 Macro Definition Documentation

5.3.2.1 CONVERGENCE_THRESHOLD

```
#define CONVERGENCE_THRESHOLD 0.0001
```

Librerías utilizadas en el proyecto.

```
#include <stdbool.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <ctype.h>
#include <dirent.h>
#include <unistd.h>
#include <time.h>
```

Macros utilizadas en el proyecto.

```
#define CONVERGENCE_THRESHOLD 0.0001
#define MAX_CHARACTERS_DOC 50
#define DAMPING_FACTOR 0.85
#define MAX_ITERATIONS 100
#define HASH_TABLE_SIZE 30
#define MAX_WORD_SIZE 50
#define MAX_NAME_DOC 20
#define MAX_DOCS 100
```

Definition at line 49 of file graph.h.

5.3.2.2 DAMPING_FACTOR

```
#define DAMPING_FACTOR 0.85
```

Definition at line 51 of file graph.h.

5.3.2.3 HASH_TABLE_SIZE

```
#define HASH_TABLE_SIZE 30
```

Definition at line 53 of file graph.h.

5.3.2.4 MAX_CHARACTERS_DOC

```
#define MAX_CHARACTERS_DOC 50
```

Definition at line 50 of file graph.h.

5.3.2.5 MAX_DOCS

```
#define MAX_DOCS 100
```

Definition at line 56 of file graph.h.

5.3.2.6 MAX_ITERATIONS

```
#define MAX_ITERATIONS 100
```

Definition at line 52 of file graph.h.

5.3.2.7 MAX_NAME_DOC

```
#define MAX_NAME_DOC 20
```

Definition at line 55 of file graph.h.

5.3.2.8 MAX_WORD_SIZE

```
#define MAX_WORD_SIZE 50
```

Definition at line 54 of file graph.h.

5.3.3 Function Documentation

5.3.3.1 add_edge()

```
void add_edge (
            Graph * graph,
            int source,
            int destination)
```

Agrega un enlace dirigido entre dos documentos en el grafo.

Parameters

graph	Grafo.
source	Documento de origen.
destination	Documento de destino.

Agrega un enlace dirigido entre dos documentos en el grafo.

Parameters

graph	Grafo.
source	Documento de origen.
destination	Documento de destino.

Crea nodo en la lista de enlaces salientes del documento origen al destino. Si no se puede asignar memoria, se muestra un mensaje de error.

```
Node *newOutputNode = (Node *)malloc(sizeof(Node));
if (newOutputNode == NULL) // Verifica la asignación de memoria.
{
    fprintf(stderr, "Error al asignar memoria para enlace saliente\n");
   exit(EXIT_FAILURE);
}
newOutputNode->doc_id = destination;
newOutputNode->next = graph->output_adjacent_list[source];
graph->output_adjacent_list[source] = newOutputNode;
```

Crea nodo en la lista de enlaces entrantes del documento destino al origen. Si no se puede asignar memoria, se muestra un mensaje de error.

```
Node *newInputNode = (Node *)malloc(sizeof(Node));
if (newInputNode == NULL)
{
   fprintf(stderr, "Error al asignar memoria para enlace entrante\n");
  exit(EXIT_FAILURE);
}
newInputNode->doc_id = source;
newInputNode->next = graph->input_adjacent_list[destination];
graph->input_adjacent_list[destination] = newInputNode;
```

Definition at line 42 of file graph.c.

5.3.3.2 build_graph()

```
void build_graph (
              Graph * graph)
```

Construye el grafo.

Parameters

graph	Grafo.

Construye el grafo.

Parameters

graph	Grafo.

Abre el directorio actual y verifica si se pudo abrir. Si no se pudo abrir, muestra un mensaje de error y termina el programa.

```
DIR *dir;
struct dirent *ent;
if ((dir = opendir(".")) == NULL)
{
     fprintf(stderr, "No se pudo abrir el directorio");
     exit(EXIT_FAILURE);
}
```

Itera sobre los archivos en el directorio.

```
while ((ent = readdir(dir)) != NULL)
{
```

```
   *Contenido*
}
closedir(dir);
```

Obtiene el nombre del archivo. Verifica si el archivo es un documento válido.

```
char *file_name = ent->d_name;
if (is_doc_name(file_name))
{
     *Contenido*
}
```

Obtiene el ID del documento. Abre el archivo para leer enlaces. Si no se pudo abrir, muestra un mensaje de error.

```
int doc_id = get_doc_id(graph, file_name);
FILE *file = fopen(file_name, "r");
if (file == NULL)
{
     fprintf(stderr, "No se pudo abrir el archivo %s\n", file_name);
     continue;
}
char line[256];
while (fgets(line, sizeof(line), file))
{
     *Contenido*
}
fclose(file);
```

Busca enlaces en la línea. Extrae el número del documento enlazado. Verifica si el número de documento es válido.

```
char *ptr = line;
while ((ptr = strstr(ptr, "link: doc")) != NULL)
{
     ptr += 9;
     int doc_number;
     if (sscanf(ptr, "%d", &doc_number) != 1)
     {
         fprintf(stderr, "Formato de enlace inválido en %s\n", file_name);
         continue;
     }
     if (doc_number <= 0 || doc_number > MAX_DOCS)
     {
         fprintf(stderr, "Número de documento inválido en enlace: %d\n", doc_number);
         continue;
     }
}
ptr++;
```

Construye el nombre del documento destino. Verifica si el destino existe. Obtiene el ID del documento destino. Agrega el enlace entre documentos.

```
char destination_name[MAX_NAME_DOC];
snprintf(destination_name, sizeof(destination_name), "doc%d.txt", doc_number);
FILE *destination_file = fopen(destination_name, "r");
if (destination_file == NULL)
{
    fprintf(stderr, "El documento %s enlazado desde %s no existe\n", destination_name, file_name);
   continue;
}
fclose(destination_file);
int destination_id = get_doc_id(graph, destination_name);
add_edge(graph, doc_id, destination_id);
```

Definition at line 97 of file graph.c.

5.3.3.3 count_input_links()

```
int count_input_links (
            Graph * graph,
            int doc_id)
```

Cuenta el número de enlaces entrantes a un documento.

Parameters

graph	Grafo.
doc↩ _id	ID del documento.

Returns

Número de enlaces entrantes.

Cuenta el número de enlaces entrantes a un documento.

Parameters

graph	Grafo.
doc↩ _id	ID del documento.

Returns

int

Recorre la lista y cuenta los enlaces entrantes.

```
int count = 0;
Node *current = graph->input_adjacent_list[doc_id];
while (current != NULL)
{
     count++;
     current = current->next;
}
return count;
```

Definition at line 351 of file graph.c.

5.3.3.4 count_output_links()

```
int count_output_links (
             Graph * graph,
             int doc_id)
```

Cuenta el número de enlaces salientes de un documento.

Parameters

graph	Grafo.
doc↩ _id	ID del documento.

Returns

Número de enlaces salientes.

Cuenta el número de enlaces salientes de un documento.

Parameters

graph	Grafo.
doc↩ _id	ID del documento.

Returns

int

Recorre la lista y cuenta los enlaces salientes.

```
int count = 0;
Node *current = graph->output_adjacent_list[doc_id];
while (current != NULL)
{
     count++;
     current = current->next;
}
return count;
```

Definition at line 318 of file graph.c.

5.3.3.5 generate_eps()

```
void generate_eps (
             const Graph * graph,
             const double * pagerank,
             const char * filename)
```

Genera un archivo .eps con la representación gráfica del grafo.

Parameters

graph	Grafo.
pagerank	Arreglo de PageRank.
file_name	Nombre del archivo.

Genera un archivo .eps con la representación gráfica del grafo.

Generar archivo EPS con el grafo.

Parameters

graph	Grafo
pagerank	Arreglo de PageRank
filename	Nombre del archivo EPS

Crear archivo EPS, en caso de no poder abrir el archivo, tira error. Inicia parámetros para ajustar el tamaño y la posición de los nodos. Dibuja los nodos con el PageRank e Identificador.

```
FILE *file = fopen(filename, "w");
if (!file)
{
   fprintf(stderr, "No se pudo crear el archivo EPS para el grafo.\n");
  exit(EXIT_FAILURE);
}
const int width = 800;
const int height = 800;
const int radius = 30;
const int margin = 100;
const int centerX = width / 2;
const int centerY = height / 2;
const double scale = 2 * M_PI / graph->total_docs;
fprintf(file, "%%!PS-Adobe-3.0 EPSF-3.0\n");
fprintf(file, "%%%%BoundingBox: 0 0 %d %d\n", width, height);
fprintf(file, "/Courier findfont 10 scalefont setfont\n");
fprintf(file, "1 setlinecap\n");
fprintf(file, "0.5 setlinewidth\n");
fprintf(file, "newpath\n");
double positions[MAX_DOCS][2];
for (int i = 0; i < graph->total_docs; i++)
     *Dibuja el Nodo*
```

Definition at line 17 of file graphic.c.

5.3.3.6 get_doc_id()

```
int get_doc_id (
            Graph * graph,
            char * file_name)
```

Obtiene el ID de un documento a partir de su nombre de archivo.

Parameters

graph	Grafo.
file_name	Nombre del archivo.

Returns

ID del documento.

Parameters

graph	Grafo.
file_name	Nombre del archivo.

Returns

int

Busca el ID en el mapeo de documentos existentes.

```
for (int i = 0; i < graph->total_docs; i++)
     if (strcmp(graph->mapping_docs[i].name, file_name) == 0)
         return graph->mapping_docs[i].doc_id;
```

Verifica si se ha alcanzado el límite de documentos permitidos.

```
if (graph->total_docs >= MAX_DOCS)
{
     fprintf(stderr, "Se ha alcanzado el número máximo de documentos\n");
     exit(EXIT_FAILURE);
}
```

Extrae el número del documento del nombre del archivo.

```
int num_doc;
if (sscanf(file_name, "doc%d.txt", &num_doc) != 1)
{
     fprintf(stderr, "Nombre de archivo inválido: %s\n", file_name);
     exit(EXIT_FAILURE);
}
```

Guarda el mapeo del nuevo documento.

```
strcpy(graph->mapping_docs[graph->total_docs].name, file_name);
graph->mapping_docs[graph->total_docs].doc_id = num_doc;
graph->total_docs++;
return num_doc;
```

Definition at line 384 of file graph.c.

5.3.3.7 initialize_graph()

```
void initialize_graph (
            Graph * graph)
```

Inicializa el grafo.

Parameters

graph	Grafo.

Inicializa el grafo.

Inicializar el Grafo.

Parameters

graph	Grafo.

Inicializa las listas de enlaces de cada documento.

```
for (int i = 0; i < MAX_DOCS; i++)
{
     graph->output_adjacent_list[i] = NULL;
     graph->input_adjacent_list[i] = NULL;
}
graph->total_docs = 0; // Inicializa el contador de documentos.
```

Definition at line 15 of file graph.c.

5.3.3.8 is_doc_name()

```
bool is_doc_name (
             char * file_name)
```

Verifica si el nombre del archivo sigue el patrón "docN.txt" con N >= 1.

Parameters

file_name	Nombre del archivo.

Returns

true si cumple el patrón, false en caso contrario.

Parameters

file_name	Nombre del archivo.

Returns

bool

Verifica si el nombre del archivo sigue el patrón "docN.txt" con N >= 1.

```
int len = strlen(file_name);
if (len < 8)
     return false;
if (strncmp(file_name, "doc", 3) != 0)
     return false;
if (strcmp(file_name + len - 4, ".txt") != 0)
     return false;
for (int i = 3; i < len - 4; i++)
{
     if (!isdigit(file_name[i]))
         return false;
     if (file_name[i] == '0' && i == 3)
         return false;
}
return true;
```

Definition at line 450 of file graph.c.

5.3.3.9 release_graph()

```
void release_graph (
            Graph * graph)
```

Libera la memoria del grafo (nodos).

Parameters

graph	Grafo.

Libera nodos en la lista de enlaces salientes.

```
Node *current_output = graph->output_adjacent_list[i];
while (current_output != NULL)
{
     Node *temp = current_output;
     current_output = current_output->next;
     free(temp);
}
graph->output_adjacent_list[i] = NULL;
```

Libera nodos en la lista de enlaces entrantes.

```
Node *current_input = graph->input_adjacent_list[i];
while (current_input != NULL)
{
     Node *temp = current_input;
     current_input = current_input->next;
     free(temp);
}
graph->input_adjacent_list[i] = NULL;
```

Definition at line 262 of file graph.c.

5.3.3.10 show_graph()

```
void show_graph (
            Graph * graph)
```

Muestra el grafo.

Parameters

graph	Grafo.

Muestra el grafo.

Parameters

graph	Grafo.

Muestra los enlaces de cada documento.

```
fprintf(stdout, "\nGrafo de enlaces:\n\n");
for (int i = 0; i < MAX_DOCS; i++)
{
     if (graph->output_adjacent_list[i] != NULL)
     {
          fprintf(stdout, "Documento %d enlaza a: ", i);
          Node *current = graph->output_adjacent_list[i];
          while (current != NULL)
          {
               fprintf(stdout, "%d ", current->doc_id);
               current = current->next;
          }
          fprintf(stdout, "\n");
     }
}
```

Definition at line 497 of file graph.c.

5.4 graph.h

Go to the documentation of this file.

```
00001
00009 #ifndef GRAPH_H
00010 #define GRAPH_H
00011
00026 #include <stdbool.h>
00027 #include <string.h>
00028 #include <stdlib.h>
00029 #include <stdio.h>
00030 #include <math.h>
00031 #include <ctype.h>
00032 #include <dirent.h>
00033 #include <unistd.h>
00034 #include <time.h>
00035
00049 #define CONVERGENCE_THRESHOLD 0.0001
00050 #define MAX_CHARACTERS_DOC 50
00051 #define DAMPING_FACTOR 0.85
00052 #define MAX_ITERATIONS 100
00053 #define HASH_TABLE_SIZE 30
00054 #define MAX_WORD_SIZE 50
00055 #define MAX_NAME_DOC 20
00056 #define MAX_DOCS 100
00057
00069 typedef struct Node
00070 {
00071     int doc_id;
00072     struct Node *next;
00073 } Node;
00074
00086 typedef struct DocumentMapping
00087 {
00088     char name[MAX_NAME_DOC];
00089     int doc_id;
00090 } DocumentMapping;
00091
00105 typedef struct Graph
00106 {
00107     Node *output_adjacent_list[MAX_DOCS];
00108     Node *input_adjacent_list[MAX_DOCS];
00109     DocumentMapping mapping_docs[MAX_DOCS];
00110     int total_docs;
00111 } Graph;
00112
00117 void initialize_graph(Graph *);
00118
00125 void add_edge(Graph *, int, int);
00126
00131 void build_graph(Graph *);
00132
00137 void release_graph(Graph *);
00138
00145 int count_output_links(Graph *, int);
00146
00153 int count_input_links(Graph *, int);
00154
00161 int get_doc_id(Graph *, char *);
00162
00168 bool is_doc_name(char *);
00169
00174 void show_graph(Graph *);
00175
00182 void generate_eps(const Graph *, const double *, const char *);
00183
00184 #endif
```

5.5 incs/inverted_index.h File Reference

Prototipos de funciones para la creación y manejo del índice invertido.

```
#include "graph.h"
```

Classes

- struct InvertedIndex
  < *Incluye la definición de las estructuras y funciones del grafo.*

Typedefs

- typedef struct InvertedIndex **InvertedIndex**

Functions

- InvertedIndex * create_new_node (char *)
  Crea un nuevo nodo en el índice invertido para una palabra específica.
- void add_document (InvertedIndex **, int, char *)
  Agrega un documento al índice invertido, asociándolo con una palabra específica.
- bool is_stopword (char *)
  Convierte todo el texto a minúsculas y reemplaza caracteres de puntuación por espacios.
- void tokenize_text (char *, int, InvertedIndex **)
  Tokeniza el texto y agrega las palabras al índice invertido.
- void print_inverted_index (InvertedIndex **)
  Imprime el índice invertido mostrando cada palabra y los documentos asociados.
- void print_search_word (InvertedIndex **, char *)
  Imprime el índice invertido mostrando cada palabra asociada a los documentos.
- unsigned int hash_function (char *)
  Función hash para obtener un índice basado en el valor ASCII de los caracteres de una palabra.
- Node * search_word (InvertedIndex **, char *)
  Busca una palabra en el índice invertido.
- void build_index (Graph *, InvertedIndex **)
  Lee los archivos que se encuentran guardados en el grafo y crea el índice.
- void release_inverted_index (InvertedIndex **)
  Libera la memoria del índice invertido y los nodos de documentos asociados.

5.5.1 Detailed Description

Prototipos de funciones para la creación y manejo del índice invertido.

Date

18-11-2024

Authors

Miguel Loaiza, Diego Sanhueza, Miguel Maripillan y Felipe Cárcamo

Contiene los prototipos de las funciones y estructura dedicadas a la creación y manejo del índice invertido.

Definition in file inverted_index.h.

5.5.2 Function Documentation

5.5.2.1 add_document()

```
void add_document (
            InvertedIndex ** hash_table,
            int doc_id,
            char * word)
```

Agrega un documento al índice invertido, asociándolo con una palabra específica.

Parameters

hash_table	Tabla hash.
doc_id	ID del documento.
word	Palabra a agregar.
hash_table	Tabla hash que contiene el índice invertido.
doc_id	ID del documento donde aparece la palabra.
word	Palabra que se indexará.

Calcula el índice en la tabla hash para la palabra. Busca la palabra en la lista enlazada correspondiente al índice.

```
unsigned int index = hash_function(word);
InvertedIndex *current = hash_table[index];
while (current != NULL)
{
    if (strcmp(current->word, word) == 0)
    {
        // La palabra ya está en el índice; agrega el documento.
        Node *new_doc = (Node *)malloc(sizeof(Node));
        new_doc->doc_id = doc_id;
        new_doc->next = current->docs_list;
        current->docs_list = new_doc;
        return;
    }
    current = current->next;
}
```

Si la palabra no está en el índice, crea un nuevo nodo y la agrega al índice. También asigna memoria para un nodo que representa el documento.

```
InvertedIndex *new_node = create_new_node(word);
new_node->next = hash_table[index];
hash_table[index] = new_node;

Node *new_doc = (Node *)malloc(sizeof(Node));
if (new_doc == NULL)
{
    fprintf(stderr, "Error al asignar memoria para nuevo documento\n");
    exit(EXIT_FAILURE);
}
new_doc->doc_id = doc_id;
new_doc->next = NULL;
new_node->docs_list = new_doc;
```

Definition at line 59 of file inverted_index.c.

5.5.2.2 build_index()

```
void build_index (
            Graph * graph,
            InvertedIndex ** index)
```

Lee los archivos que se encuentran guardados en el grafo y crea el índice.

Parameters

graph	Grafo.
index	Índice invertido.

Lee los archivos que se encuentran guardados en el grafo y crea el índice.

Parameters

graph	Grafo que contiene los datos de los documentos.

index	Tabla hash que contiene el índice invertido.

Itera sobre los documentos identificados en el grafo.

```
for (int i = 0; i < graph->total_docs; i++)
{
    FILE *file = fopen(graph->mapping_docs[i].name, "r");
    if (file == NULL)
    {
        fprintf(stderr, "No se pudo abrir el archivo %s\n", graph->mapping_docs[i].name);
        continue;
    }
}
```

Intenta abrir el archivo correspondiente al documento. Si el archivo no puede ser abierto, muestra un error y pasa al siguiente documento.

```
FILE *file = fopen(graph->mapping_docs[i].name, "r");
if (file == NULL)
{
    fprintf(stderr, "No se pudo abrir el archivo %s\n", graph->mapping_docs[i].name);
    continue;
}
```

Define un buffer para leer cada línea del archivo.

```
char buffer[1024];
while (fgets(buffer, sizeof(buffer), file))
{
    tokenize_text(buffer, graph->mapping_docs[i].doc_id, index);
}
```

Procesa cada línea del archivo y la pasa a la función tokenize_text, que procesa el texto y lo agrega al índice invertido.

```
tokenize_text(buffer, graph->mapping_docs[i].doc_id, index);
```

Cierra el archivo una vez que ha terminado de procesarlo.

```
fclose(file);
```

Definition at line 509 of file inverted_index.c.

5.5.2.3 create_new_node()

```
InvertedIndex * create_new_node (
            char * word)
```

Crea un nuevo nodo en el índice invertido para una palabra específica.

Parameters

word	La palabra a agregar.

Returns

new_node

Crea un nuevo nodo en el índice invertido para una palabra específica.

< Incluye la definición de las estructuras y funciones del grafo.

Crea un nuevo nodo en el índice invertido para una palabra específica.

Parameters

word	La palabra a agregar.

Returns

Un puntero al nuevo nodo creado.

Asigna memoria para el nuevo nodo del índice invertido. Si la asignación falla, se imprime un mensaje de error y el programa termina.

```
InvertedIndex *new_node = (InvertedIndex *)malloc(sizeof(InvertedIndex));
if (new_node == NULL)
{
    fprintf(stderr, "Error al asignar memoria\n");
    exit(EXIT_FAILURE);
}
```

Copia la palabra en el nodo y establece la lista de documentos y el puntero siguiente a NULL. Inicialmente, no hay documentos asociados con esta palabra, y el puntero al siguiente nodo también es NULL.

```
strcpy(new_node->word, word);
new_node->docs_list = NULL;
new_node->next = NULL;
```

Definition at line 17 of file inverted_index.c.

5.5.2.4 hash_function()

```
unsigned int hash_function (
            char * word)
```

Función hash para obtener un índice basado en el valor ASCII de los caracteres de una palabra.

Parameters

word	Palabra a procesar.

Returns

hash

Parameters

word	Palabra a procesar.

Returns

Valor hash de la palabra.

Inicializa el valor hash en cero.

```
unsigned int hash = 0;
```

Recorre cada carácter en la palabra y actualiza el valor hash. El hash se calcula multiplicando el valor anterior por 31 y sumando el código ASCII de cada carácter.

```
for (int i = 0; word[i] != '\0'; i++)
    hash = (hash * 31) + word[i];
```

Limita el valor hash por el tamaño máximo de la tabla hash.

```
return hash % HASH_TABLE_SIZE;
```

Definition at line 330 of file inverted_index.c.

5.5.2.5 is_stopword()

```
bool is_stopword (
            char * token)
```

Convierte todo el texto a minúsculas y reemplaza caracteres de puntuación por espacios.

Parameters

token	Token a procesar.

Convierte todo el texto a minúsculas y reemplaza caracteres de puntuación por espacios.

Parameters

token	El token a analizar.

Returns

`true` si el token es una stopword, `false` en caso contrario.

Lista de palabras consideradas como stopwords.

```
static const char *stopwords[] = {"a", "e", "i", "o", "u", "link"};
static const int num_stopwords = sizeof(stopwords) / sizeof(stopwords[0]);
```

Convierte el token a minúsculas para evitar problemas con mayúsculas/minúsculas.

```
for (int i = 0; token[i]; i++)
{
    token[i] = tolower((unsigned char)token[i]);
}
```

Compara el token con las stopwords. Si el token es una stopword, devuelve `true`.

```
for (int i = 0; i < num_stopwords; i++)
{
    if (strcmp(token, stopwords[i]) == 0)
    {
        return true;
    }
}
```

Verificación del patrón "docN", donde N es un número. Si el token comienza con "doc" seguido de un número, devuelve `true`.

```
if (strncmp(token, "doc", 3) == 0 && isdigit(token[3]))
{
    return true;
}
```

Definition at line 137 of file inverted_index.c.

5.5.2.6 print_inverted_index()

```
void print_inverted_index (
            InvertedIndex ** index)
```

Imprime el índice invertido mostrando cada palabra y los documentos asociados.

Parameters

index	Índice invertido.
index	Tabla hash que contiene el índice invertido.

Recorre la tabla hash y muestra las palabras y sus documentos asociados.

```
for (int i = 0; i < HASH_TABLE_SIZE; i++)
{
    if (index[i] != NULL)
    {
        InvertedIndex *current = index[i];
        while (current != NULL)
        {
            fprintf(stdout, "Palabra: %s - Documentos: ", current->word);
            Node *doc_node = current->docs_list;
            while (doc_node != NULL)
            {
                fprintf(stdout, "%d ", doc_node->doc_id);
                doc_node = doc_node->next;
            }
            fprintf(stdout, "\n");
            current = current->next;
        }
    }
}
```

Definition at line 275 of file inverted_index.c.

5.5.2.7 print_search_word()

```
void print_search_word (
            InvertedIndex ** index,
            char * word_to_search)
```

Imprime el índice invertido mostrando cada palabra asociada a los documentos.

Parameters

index	Índice invertido.

Imprime el índice invertido mostrando cada palabra asociada a los documentos.

Parameters

index	Tabla hash que contiene el índice invertido.
word_to_search	Palabra a buscar en el índice.

Llama a la función search_word para obtener los documentos asociados a la palabra.

```
Node *results = search_word(index, word_to_search);
```

Si no se encuentran resultados, muestra un mensaje de error.

```
if (!results)
{
    fprintf(stderr, "Palabra '%s' no encontrada.\n", word_to_search);
    return;
}
```

Muestra los documentos en los que se encuentra la palabra.

```
fprintf(stdout, "\nLa palabra '%s' se encuentra en los siguientes documentos:\n\n", word_to_search);
```

Inicializa un contador para registrar cuántas veces aparece la palabra en cada documento.

```
int doc_count[MAX_DOCS] = {0};
```

Recorre la lista de documentos y cuenta cuántas veces aparece la palabra en cada documento.

```
while (current != NULL)
```

```
{
    doc_count[current->doc_id]++;
    current = current->next;
}
```

Muestra la cantidad de veces que aparece la palabra en cada documento.

```
for (int i = 0; i < MAX_DOCS; i++)
{
    if (doc_count[i] > 0)
        fprintf(stdout, "Doc%d: %d veces\n", i, doc_count[i]);
}
```

Definition at line 427 of file inverted_index.c.

5.5.2.8 release_inverted_index()

```
void release_inverted_index (
            InvertedIndex ** hash_table)
```

Libera la memoria del índice invertido y los nodos de documentos asociados.

Parameters

hash_table	Tabla hash.
hash_table	Tabla hash que contiene el índice invertido.

Verifica si la tabla hash es nula y retorna si es el caso.

```
if (hash_table == NULL)
    return;
```

Recorre cada índice en la tabla hash.

```
for (int i = 0; i < HASH_TABLE_SIZE; i++)
{
    InvertedIndex *current = hash_table[i];
    ...
}
```

Procesa la lista de palabras asociada a cada índice en la tabla hash.

```
InvertedIndex *current = hash_table[i];
while (current != NULL)
{
    Node *doc_node = current->docs_list;
    ...
}
```

Libera la memoria de cada palabra y su lista de documentos.

```
while (current != NULL)
{
    Node *doc_node = current->docs_list;
    while (doc_node != NULL)
    {
        Node *temp_doc_node = doc_node;
        doc_node = doc_node->next;
        free(temp_doc_node);
    }
    InvertedIndex *temp = current;
    current = current->next;
    free(temp);
}
```

Libera cada nodo de documento.

```
Node *temp_doc_node = doc_node;
doc_node = doc_node->next;
free(temp_doc_node);
```

Libera el nodo de la palabra.

```
InvertedIndex *temp = current;
current = current->next;
free(temp);
```

Establece el índice en la tabla hash como NULL después de liberar su memoria.

```
hash_table[i] = NULL;
```

Definition at line 583 of file inverted_index.c.

5.5.2.9 search_word()

```
Node * search_word (
            InvertedIndex ** hash_table,
            char * word)
```

Busca una palabra en el índice invertido.

Parameters

index	Índice invertido.
word_to_search	Palabra a buscar.

Returns

results

Busca una palabra en el índice invertido.

Parameters

hash_table	Tabla hash que contiene el índice invertido.
word	Palabra que se busca en el índice.

Returns

Lista de nodos de documentos donde aparece la palabra, o `NULL` si no se encuentra.

Convierte la palabra a minúsculas para una comparación uniforme.

```
for (int i = 0; word[i]; i++)
    word[i] = tolower(word[i]);
```

Obtiene el índice en la tabla hash utilizando la función hash.

```
unsigned int index = hash_function(word);
```

Inicia la búsqueda en la lista de nodos de la tabla hash.

```
InvertedIndex *current = hash_table[index];
```

Recorre los nodos en busca de la palabra, comparando cada nodo con la palabra buscada. Si se encuentra, se retorna la lista de documentos asociados.

```
while (current != NULL)
{
    if (strcmp(current->word, word) == 0)
        return current->docs_list;
    current = current->next;
}
```

Retorna NULL si la palabra no se encuentra en la tabla hash.

```
return NULL;
```

Definition at line 366 of file inverted_index.c.

5.5.2.10 tokenize_text()

```
void tokenize_text (
            char * text,
            int doc_id,
            InvertedIndex ** index)
```

Tokeniza el texto y agrega las palabras al índice invertido.

Parameters

text	Texto a tokenizar.
doc↩ _id	ID del documento.
index	Índice invertido.

Tokeniza el texto y agrega las palabras al índice invertido.

Parameters

text	Texto de entrada que será tokenizado.
doc↩ _id	ID del documento al que pertenece el texto.
index	Tabla hash que contiene el índice invertido.

Convierte todo el texto a minúsculas.
```
for (int i = 0; text[i]; i++)
    text[i] = tolower(text[i]);
```

Reemplaza caracteres de puntuación por espacios.
```
for (int i = 0; text[i]; i++)
    if (ispunct(text[i]))
        text[i] = ' ';
```

Separa el texto en palabras utilizando el espacio como delimitador.
```
token = strtok(text, " ");
```

Procesa cada token.
```
while (token != NULL)
{
    if (strlen(token) >= MAX_WORD_SIZE)
    {
        fprintf(stderr, "La palabra es muy larga\n");
        exit(EXIT_FAILURE);
    }
    if (!is_stopword(token))
        add_document(index, doc_id, token);
    token = strtok(NULL, " ");
}
```

Definition at line 207 of file inverted_index.c.

5.6 inverted_index.h

Go to the documentation of this file.
```
00001
00009 #ifndef INVERTED_INDEX_H
00010 #define INVERTED_INDEX_H
00011
00012 #include "graph.h"
00013
00026 typedef struct InvertedIndex
00027 {
00028     char word[MAX_WORD_SIZE];
00029     Node *docs_list;
00030     struct InvertedIndex *next;
00031 } InvertedIndex;
00032
00038 InvertedIndex *create_new_node(char *);
00039
00046 void add_document(InvertedIndex **, int, char *);
00047
00052 bool is_stopword(char *);
00053
```

```
00060 void tokenize_text(char *, int, InvertedIndex **);
00061
00066 void print_inverted_index(InvertedIndex **);
00067
00072 void print_search_word(InvertedIndex **, char *);
00073
00079 unsigned int hash_function(char *);
00080
00087 Node *search_word(InvertedIndex **, char *);
00088
00094 void build_index(Graph *, InvertedIndex **);
00095
00100 void release_inverted_index(InvertedIndex **);
00101
00102 #endif
```

5.7 incs/pagerank.h File Reference

Prototipos de funciones para la creación del PageRank.

```
#include "graph.h"
```

Functions

- void initialize_pagerank (double *, int)
  < *Incluye la definición de las estructuras y funciones del grafo.*
- void calculate_pagerank (Graph *, double *)
  Calcular PageRank.
- void display_pagerank (Graph *, double *)
  Mostrar PageRank.

5.7.1 Detailed Description

Prototipos de funciones para la creación del PageRank.

Date

18-11-2024

Authors

Miguel Loaiza, Diego Sanhueza, Miguel Maripillan y Felipe Cárcamo

Contiene los prototipos de las funciones dedicadas a la creación del PageRank.

Definition in file pagerank.h.

5.7.2 Function Documentation

5.7.2.1 calculate_pagerank()

```
void calculate_pagerank (
            Graph * graph,
            double * pagerank)
```

Calcular PageRank.

Parameters

graph	Grafo
pagerank	Arreglo de PageRank

Inicializa variables y PageRank.

```
int num_docs = graph->total_docs;
initialize_pagerank(pagerank, num_docs);
double temp_rank[MAX_DOCS];
```

Itera hasta que se cumple el criterio de convergencia. Calcula la contribución de cada nodo a los que apunta. Distribuye la contribución de PageRank a cada nodo de la lista de adyacencia.

```
for (int iteration = 0; iteration < MAX_ITERATIONS; iteration++)
{
     for (int i = 0; i < num_docs; i++)
         temp_rank[i] = (1 - DAMPING_FACTOR) / num_docs;
     for (int i = 0; i < num_docs; i++)
     {
         int num_links = count_output_links(graph, i);
         if (num_links == 0)
             continue;
         double rank_contribution = pagerank[i] * DAMPING_FACTOR / num_links;
         Node *current = graph->output_adjacent_list[i];
         while (current != NULL)
         {
             temp_rank[current->doc_id] += rank_contribution;
             current = current->next;
         }
     }
```

Calcula el error y actualiza los valores de PageRank. Si el error es menor al umbral de convergencia, se detiene el algoritmo.

```
double error = 0;
for (int i = 0; i < num_docs; i++)
{
     error += fabs(pagerank[i] - temp_rank[i]);
     pagerank[i] = temp_rank[i];
}
if (error < CONVERGENCE_THRESHOLD)
   break;
```

Definition at line 34 of file pagerank.c.

5.7.2.2 display_pagerank()

```
void display_pagerank (
             Graph * graph,
             double * pagerank)
```

Mostrar PageRank.

Parameters

graph	Grafo
pagerank	Arreglo de PageRank

Muestra los valores de PageRank de cada documento.

```
fprintf(stdout, "\nValores de PageRank ordenados por importancia:\n\n");
int num_docs = graph->total_docs;
int indices[num_docs];
for (int i = 0; i < num_docs; i++)
    indices[i] = i;
for (int i = 0; i < num_docs - 1; i++)
{
    for (int j = i + 1; j < num_docs; j++)
    {
```

```
        if (pagerank[indices[i]] < pagerank[indices[j]])
        {
            int temp = indices[i];
            indices[i] = indices[j];
            indices[j] = temp;
        }
    }
}
for (int i = 0; i < num_docs; i++)
{
    int doc_id = indices[i];
     fprintf(stdout, "Documento (%s): PageRank = %.6f\n", graph->mapping_docs[doc_id].name,
      pagerank[doc_id]);
}
```

Definition at line 126 of file pagerank.c.

5.7.2.3 initialize_pagerank()

```
void initialize_pagerank (
              double * pagerank,
              int num_docs)
```

< Incluye la definición de las estructuras y funciones del grafo.

Inicializar PageRank

Parameters

pagerank	Arreglo de PageRank
num_docs	Número de documentos

Inicializa cada documento con el mismo valor de PageRank.

```
for (int i = 0; i < num_docs; i++)
    pagerank[i] = 1.0 / num_docs;
```

Definition at line 16 of file pagerank.c.

5.8 pagerank.h

Go to the documentation of this file.

```
00001
00009 #ifndef PAGERANK_H
00010 #define PAGERANK_H
00011
00012 #include "graph.h"
00013
00019 void initialize_pagerank(double *, int);
00020
00026 void calculate_pagerank(Graph *, double *);
00027
00033 void display_pagerank(Graph *, double *);
00034
00035 #endif
```

5.9 src/graph.c File Reference

Archivo que contiene las funciones de los Grafos.

```
#include "graph.h"
```

Functions

- void initialize_graph (Graph *graph)

  < *Incluye la definición de las estructuras y funciones del grafo.*
- void add_edge (Graph *graph, int source, int destination)

  Agregar un enlace dirigido entre dos documentos en el grafo.
- void build_graph (Graph *graph)

  Construir el Grafo a partir de documentos que contienen enlaces a otros documentos.
- void release_graph (Graph *graph)

  Libera la memoria del grafo (nodos).
- int count_output_links (Graph *graph, int doc_id)

  Contar el número de enlaces salientes de un documento.
- int count_input_links (Graph *graph, int doc_id)

  Contar el número de enlaces entrantes a un documento.
- int get_doc_id (Graph *graph, char *file_name)

  Obtiene el ID de un documento a partir de su nombre de archivo.
- bool is_doc_name (char *file_name)

  Verifica si el nombre del archivo sigue el patrón "docN.txt" con N >= 1.
- void show_graph (Graph *graph)

  Muestra el grafo de enlaces, imprimiendo los documentos y sus enlaces salientes.

5.9.1 Detailed Description

Archivo que contiene las funciones de los Grafos.

Date

18-11-2024

Authors

Miguel Loaiza, Diego Sanhueza, Miguel Maripillan y Felipe Cárcamo

Contiene la implementación de las funciones que tiene que ver con los grafos.

Definition in file graph.c.

5.9.2 Function Documentation

5.9.2.1 add_edge()

```
void add_edge (
            Graph * graph,
            int source,
            int destination)
```

Agregar un enlace dirigido entre dos documentos en el grafo.

Agrega un enlace dirigido entre dos documentos en el grafo.

Parameters

graph	Grafo.
source	Documento de origen.
destination	Documento de destino.

Crea nodo en la lista de enlaces salientes del documento origen al destino. Si no se puede asignar memoria, se muestra un mensaje de error.

```
Node *newOutputNode = (Node *)malloc(sizeof(Node));
if (newOutputNode == NULL) // Verifica la asignación de memoria.
{
    fprintf(stderr, "Error al asignar memoria para enlace saliente\n");
   exit(EXIT_FAILURE);
}
newOutputNode->doc_id = destination;
newOutputNode->next = graph->output_adjacent_list[source];
graph->output_adjacent_list[source] = newOutputNode;
```

Crea nodo en la lista de enlaces entrantes del documento destino al origen. Si no se puede asignar memoria, se muestra un mensaje de error.

```
Node *newInputNode = (Node *)malloc(sizeof(Node));
if (newInputNode == NULL)
{
   fprintf(stderr, "Error al asignar memoria para enlace entrante\n");
  exit(EXIT_FAILURE);
}
newInputNode->doc_id = source;
newInputNode->next = graph->input_adjacent_list[destination];
graph->input_adjacent_list[destination] = newInputNode;
```

Definition at line 42 of file graph.c.

5.9.2.2 build_graph()

```
void build_graph (
              Graph * graph)
```

Construir el Grafo a partir de documentos que contienen enlaces a otros documentos.

Construye el grafo.

Parameters

graph	Grafo.

Abre el directorio actual y verifica si se pudo abrir. Si no se pudo abrir, muestra un mensaje de error y termina el programa.

```
DIR *dir;
struct dirent *ent;
if ((dir = opendir(".")) == NULL)
{
     fprintf(stderr, "No se pudo abrir el directorio");
     exit(EXIT_FAILURE);
}
```

Itera sobre los archivos en el directorio.

```
while ((ent = readdir(dir)) != NULL)
{
   *Contenido*
}
closedir(dir);
```

Obtiene el nombre del archivo. Verifica si el archivo es un documento válido.

```
char *file_name = ent->d_name;
if (is_doc_name(file_name))
```

```
{
     *Contenido*
}
```

Obtiene el ID del documento. Abre el archivo para leer enlaces. Si no se pudo abrir, muestra un mensaje de error.

```
int doc_id = get_doc_id(graph, file_name);
FILE *file = fopen(file_name, "r");
if (file == NULL)
{
     fprintf(stderr, "No se pudo abrir el archivo %s\n", file_name);
     continue;
}
char line[256];
while (fgets(line, sizeof(line), file))
{
     *Contenido*
}
fclose(file);
```

Busca enlaces en la línea. Extrae el número del documento enlazado. Verifica si el número de documento es válido.

```
char *ptr = line;
while ((ptr = strstr(ptr, "link: doc")) != NULL)
{
     ptr += 9;
     int doc_number;
     if (sscanf(ptr, "%d", &doc_number) != 1)
     {
         fprintf(stderr, "Formato de enlace inválido en %s\n", file_name);
         continue;
     }
     if (doc_number <= 0 || doc_number > MAX_DOCS)
     {
         fprintf(stderr, "Número de documento inválido en enlace: %d\n", doc_number);
         continue;
     }
}
ptr++;
```

Construye el nombre del documento destino. Verifica si el destino existe. Obtiene el ID del documento destino. Agrega el enlace entre documentos.

```
char destination_name[MAX_NAME_DOC];
snprintf(destination_name, sizeof(destination_name), "doc%d.txt", doc_number);
FILE *destination_file = fopen(destination_name, "r");
if (destination_file == NULL)
{
    fprintf(stderr, "El documento %s enlazado desde %s no existe\n", destination_name, file_name);
   continue;
}
fclose(destination_file);
int destination_id = get_doc_id(graph, destination_name);
add_edge(graph, doc_id, destination_id);
```

Definition at line 97 of file graph.c.

5.9.2.3 count_input_links()

```
int count_input_links (
            Graph * graph,
            int doc_id)
```

Contar el número de enlaces entrantes a un documento.

Cuenta el número de enlaces entrantes a un documento.

Parameters

graph	Grafo.
doc↩ _id	ID del documento.

Returns

int

Recorre la lista y cuenta los enlaces entrantes.

```
int count = 0;
Node *current = graph->input_adjacent_list[doc_id];
while (current != NULL)
{
     count++;
     current = current->next;
}
return count;
```

Definition at line 351 of file graph.c.

5.9.2.4 count_output_links()

```
int count_output_links (
              Graph * graph,
              int doc_id)
```

Contar el número de enlaces salientes de un documento.

Cuenta el número de enlaces salientes de un documento.

Parameters

graph	Grafo.
doc↩ _id	ID del documento.

Returns

int

Recorre la lista y cuenta los enlaces salientes.

```
int count = 0;
Node *current = graph->output_adjacent_list[doc_id];
while (current != NULL)
{
     count++;
     current = current->next;
}
return count;
```

Definition at line 318 of file graph.c.

5.9.2.5 get_doc_id()

```
int get_doc_id (
              Graph * graph,
              char * file_name)
```

Obtiene el ID de un documento a partir de su nombre de archivo.

Parameters

graph	Grafo.
file_name	Nombre del archivo.

Returns

int

Busca el ID en el mapeo de documentos existentes.

```
for (int i = 0; i < graph->total_docs; i++)
     if (strcmp(graph->mapping_docs[i].name, file_name) == 0)
         return graph->mapping_docs[i].doc_id;
```

Verifica si se ha alcanzado el límite de documentos permitidos.

```
if (graph->total_docs >= MAX_DOCS)
{
     fprintf(stderr, "Se ha alcanzado el número máximo de documentos\n");
     exit(EXIT_FAILURE);
}
```

Extrae el número del documento del nombre del archivo.

```
int num_doc;
if (sscanf(file_name, "doc%d.txt", &num_doc) != 1)
{
     fprintf(stderr, "Nombre de archivo inválido: %s\n", file_name);
     exit(EXIT_FAILURE);
}
```

Guarda el mapeo del nuevo documento.

```
strcpy(graph->mapping_docs[graph->total_docs].name, file_name);
graph->mapping_docs[graph->total_docs].doc_id = num_doc;
graph->total_docs++;
return num_doc;
```

Definition at line 384 of file graph.c.

5.9.2.6 initialize_graph()

```
void initialize_graph (
             Graph * graph)
```

< Incluye la definición de las estructuras y funciones del grafo.

Inicializa el grafo.

Inicializar el Grafo.

Parameters

graph	Grafo.

Inicializa las listas de enlaces de cada documento.

```
for (int i = 0; i < MAX_DOCS; i++)
{
     graph->output_adjacent_list[i] = NULL;
     graph->input_adjacent_list[i] = NULL;
}
graph->total_docs = 0; // Inicializa el contador de documentos.
```

Definition at line 15 of file graph.c.

5.9.2.7 is_doc_name()

```
bool is_doc_name (
             char * file_name)
```

Verifica si el nombre del archivo sigue el patrón "docN.txt" con N >= 1.

Parameters

file_name	Nombre del archivo.

Returns

bool

Verifica si el nombre del archivo sigue el patrón "docN.txt" con N >= 1.

```
int len = strlen(file_name);
if (len < 8)
     return false;
if (strncmp(file_name, "doc", 3) != 0)
     return false;
if (strcmp(file_name + len - 4, ".txt") != 0)
     return false;
for (int i = 3; i < len - 4; i++)
{
     if (!isdigit(file_name[i]))
         return false;
     if (file_name[i] == '0' && i == 3)
         return false;
}
return true;
```

Definition at line 450 of file graph.c.

5.9.2.8 release_graph()

```
void release_graph (
              Graph * graph)
```

Libera la memoria del grafo (nodos).

Parameters

graph	Grafo.

Libera nodos en la lista de enlaces salientes.

```
Node *current_output = graph->output_adjacent_list[i];
while (current_output != NULL)
{
     Node *temp = current_output;
     current_output = current_output->next;
     free(temp);
}
graph->output_adjacent_list[i] = NULL;
```

Libera nodos en la lista de enlaces entrantes.

```
Node *current_input = graph->input_adjacent_list[i];
while (current_input != NULL)
{
     Node *temp = current_input;
     current_input = current_input->next;
     free(temp);
}
graph->input_adjacent_list[i] = NULL;
```

Definition at line 262 of file graph.c.

5.9.2.9 show_graph()

```
void show_graph (
              Graph * graph)
```

Muestra el grafo de enlaces, imprimiendo los documentos y sus enlaces salientes.

Muestra el grafo.

Parameters

graph	Grafo.

Muestra los enlaces de cada documento.

```
fprintf(stdout, "\nGrafo de enlaces:\n\n");
for (int i = 0; i < MAX_DOCS; i++)
{
     if (graph->output_adjacent_list[i] != NULL)
     {
         fprintf(stdout, "Documento %d enlaza a: ", i);
         Node *current = graph->output_adjacent_list[i];
         while (current != NULL)
         {
             fprintf(stdout, "%d ", current->doc_id);
             current = current->next;
         }
         fprintf(stdout, "\n");
     }
}
```

Definition at line 497 of file graph.c.

5.10 graph.c

Go to the documentation of this file.

```
00001
00009 #include "graph.h"
00010
00015 void initialize_graph(Graph *graph)
00016 {
00028     for (int i = 0; i < MAX_DOCS; i++)
00029     {
00030         graph->output_adjacent_list[i] = NULL;
00031         graph->input_adjacent_list[i] = NULL;
00032     }
00033     graph->total_docs = 0;
00034 }
00035
00042 void add_edge(Graph *graph, int source, int destination)
00043 {
00058     Node *newOutputNode = (Node *)malloc(sizeof(Node));
00059     if (newOutputNode == NULL)
00060     {
00061         fprintf(stderr, "Error al asignar memoria para enlace saliente\n");
00062         exit(EXIT_FAILURE);
00063     }
00064     newOutputNode->doc_id = destination;
00065     newOutputNode->next = graph->output_adjacent_list[source];
00066     graph->output_adjacent_list[source] = newOutputNode;
00067
00082     Node *newInputNode = (Node *)malloc(sizeof(Node));
00083     if (newInputNode == NULL)
00084     {
00085         fprintf(stderr, "Error al asignar memoria para enlace entrante\n");
00086         exit(EXIT_FAILURE);
00087     }
00088     newInputNode->doc_id = source;
00089     newInputNode->next = graph->input_adjacent_list[destination];
00090     graph->input_adjacent_list[destination] = newInputNode;
00091 }
00092
00097 void build_graph(Graph *graph)
00098 {
00111     DIR *dir;
00112     struct dirent *ent;
00113
00114     if ((dir = opendir(".")) == NULL)
00115     {
00116         fprintf(stderr, "No se pudo abrir el directorio");
00117         exit(EXIT_FAILURE);
00118     }
00119
00130     while ((ent = readdir(dir)) != NULL)
00131     {
00142         char *file_name = ent->d_name;
```

```
00143
00144         if (is_doc_name(file_name))
00145         {
00164             int doc_id = get_doc_id(graph, file_name);
00165
00166             FILE *file = fopen(file_name, "r");
00167             if (file == NULL)
00168             {
00169                 fprintf(stderr, "No se pudo abrir el archivo %s\n", file_name);
00170                 continue;
00171             }
00172
00173             char line[256];
00174
00175             while (fgets(line, sizeof(line), file))
00176             {
00199                 char *ptr = line;
00200
00201                 while ((ptr = strstr(ptr, "link: doc")) != NULL)
00202                 {
00203                     ptr += 9;
00204                     int doc_number;
00205
00206                     if (sscanf(ptr, "%d", &doc_number) != 1)
00207                     {
00208                         fprintf(stderr, "Formato de enlace inválido en %s\n", file_name);
00209                         continue;
00210                     }
00211
00212                     if (doc_number <= 0 || doc_number > MAX_DOCS)
00213                     {
00214                         fprintf(stderr, "Número de documento inválido en enlace: %d\n", doc_number);
00215                         continue;
00216                     }
00217
00235                     char destination_name[MAX_NAME_DOC];
00236                     snprintf(destination_name, sizeof(destination_name), "doc%d.txt", doc_number);
00237
00238                     FILE *destination_file = fopen(destination_name, "r");
00239                     if (destination_file == NULL)
00240                     {
00241                         fprintf(stderr, "El documento %s enlazado desde %s no existe\n",
      destination_name, file_name);
00242                         continue;
00243                     }
00244                     fclose(destination_file);
00245
00246                     int destination_id = get_doc_id(graph, destination_name);
00247
00248                     add_edge(graph, doc_id, destination_id);
00249                 }
00250                 ptr++;
00251             }
00252             fclose(file);
00253         }
00254     }
00255     closedir(dir);
00256 }
00257
00262 void release_graph(Graph *graph)
00263 {
00264     for (int i = 0; i < MAX_DOCS; i++)
00265     {
00279         Node *current_output = graph->output_adjacent_list[i];
00280         while (current_output != NULL)
00281         {
00282             Node *temp = current_output;
00283             current_output = current_output->next;
00284             free(temp);
00285         }
00286         graph->output_adjacent_list[i] = NULL;
00287
00301         Node *current_input = graph->input_adjacent_list[i];
00302         while (current_input != NULL)
00303         {
00304             Node *temp = current_input;
00305             current_input = current_input->next;
00306             free(temp);
00307         }
00308         graph->input_adjacent_list[i] = NULL;
00309     }
00310 }
00311
00318 int count_output_links(Graph *graph, int doc_id)
00319 {
00333     int count = 0;
00334     Node *current = graph->output_adjacent_list[doc_id];
```

```
00335
00336     while (current != NULL)
00337     {
00338         count++;
00339         current = current->next;
00340     }
00341
00342     return count;
00343 }
00344
00351 int count_input_links(Graph *graph, int doc_id)
00352 {
00366     int count = 0;
00367     Node *current = graph->input_adjacent_list[doc_id];
00368
00369     while (current != NULL)
00370     {
00371         count++;
00372         current = current->next;
00373     }
00374
00375     return count;
00376 }
00377
00384 int get_doc_id(Graph *graph, char *file_name)
00385 {
00402     for (int i = 0; i < graph->total_docs; i++)
00403         if (strcmp(graph->mapping_docs[i].name, file_name) == 0)
00404             return graph->mapping_docs[i].doc_id;
00405
00406     if (graph->total_docs >= MAX_DOCS)
00407     {
00408         fprintf(stderr, "Se ha alcanzado el número máximo de documentos\n");
00409         exit(EXIT_FAILURE);
00410     }
00411
00412     int num_doc;
00413
00432     if (sscanf(file_name, "doc%d.txt", &num_doc) != 1)
00433     {
00434         fprintf(stderr, "Nombre de archivo inválido: %s\n", file_name);
00435         exit(EXIT_FAILURE);
00436     }
00437
00438     strcpy(graph->mapping_docs[graph->total_docs].name, file_name);
00439     graph->mapping_docs[graph->total_docs].doc_id = num_doc;
00440     graph->total_docs++;
00441
00442     return num_doc;
00443 }
00444
00450 bool is_doc_name(char *file_name)
00451 {
00472     int len = strlen(file_name);
00473     if (len < 8)
00474         return false;
00475
00476     if (strncmp(file_name, "doc", 3) != 0)
00477         return false;
00478
00479     if (strcmp(file_name + len - 4, ".txt") != 0)
00480         return false;
00481
00482     for (int i = 3; i < len - 4; i++)
00483     {
00484         if (!isdigit(file_name[i]))
00485             return false;
00486         if (file_name[i] == '0' && i == 3)
00487             return false;
00488     }
00489
00490     return true;
00491 }
00492
00497 void show_graph(Graph *graph)
00498 {
00519     fprintf(stdout, "\nGrafo de enlaces:\n\n");
00520     for (int i = 0; i < MAX_DOCS; i++)
00521     {
00522         if (graph->output_adjacent_list[i] != NULL)
00523         {
00524             fprintf(stdout, "Documento %d enlaza a: ", i);
00525             Node *current = graph->output_adjacent_list[i];
00526             while (current != NULL)
00527             {
00528                 fprintf(stdout, "%d ", current->doc_id);
00529                 current = current->next;
```

```
00530             }
00531             fprintf(stdout, "\n");
00532         }
00533     }
00534 }
```

5.11 src/graphic.c File Reference

Archivo que contiene las funciones de generación de dibujos eps.

```
#include "graph.h"
```

Functions

- void generate_eps (const Graph *graph, const double *pagerank, const char *filename)
  < *Incluye la definición de las estructuras y funciones del grafo.*

5.11.1 Detailed Description

Archivo que contiene las funciones de generación de dibujos eps.

Date

18-11-2024

Authors

Miguel Loaiza, Diego Sanhueza, Miguel Maripillan y Felipe Cárcamo

Contiene la implementación de las funciones que generan un archivo EPS con los grafos.

Definition in file graphic.c.

5.11.2 Function Documentation

5.11.2.1 generate_eps()

```
void generate_eps (
            const Graph * graph,
            const double * pagerank,
            const char * filename)
```

< Incluye la definición de las estructuras y funciones del grafo.

Genera un archivo .eps con la representación gráfica del grafo.

Generar archivo EPS con el grafo.

Parameters

graph	Grafo
pagerank	Arreglo de PageRank
filename	Nombre del archivo EPS

Crear archivo EPS, en caso de no poder abrir el archivo, tira error. Inicia parámetros para ajustar el tamaño y la posición de los nodos. Dibuja los nodos con el PageRank e Identificador.

```
FILE *file = fopen(filename, "w");
if (!file)
{
   fprintf(stderr, "No se pudo crear el archivo EPS para el grafo.\n");
  exit(EXIT_FAILURE);
}
const int width = 800;
const int height = 800;
const int radius = 30;
const int margin = 100;
const int centerX = width / 2;
const int centerY = height / 2;
const double scale = 2 * M_PI / graph->total_docs;
fprintf(file, "%%!PS-Adobe-3.0 EPSF-3.0\n");
fprintf(file, "%%%%BoundingBox: 0 0 %d %d\n", width, height);
fprintf(file, "/Courier findfont 10 scalefont setfont\n");
fprintf(file, "1 setlinecap\n");
fprintf(file, "0.5 setlinewidth\n");
fprintf(file, "newpath\n");
double positions[MAX_DOCS][2];
for (int i = 0; i < graph->total_docs; i++)
     *Dibuja el Nodo*
```

Definition at line 17 of file graphic.c.

5.12 graphic.c

Go to the documentation of this file.

```
00001
00009 #include "graph.h"
00010
00017 void generate_eps(const Graph *graph, const double *pagerank, const char *filename)
00018 {
00049     FILE *file = fopen(filename, "w");
00050     if (!file)
00051     {
00052         fprintf(stderr, "No se pudo crear el archivo EPS para el grafo.\n");
00053         exit(EXIT_FAILURE);
00054     }
00055
00056     const int width = 800;
00057     const int height = 800;
00058     const int radius = 30;
00059     const int margin = 100;
00060     const int centerX = width / 2;
00061     const int centerY = height / 2;
00062     const double scale = 2 * M_PI / graph->total_docs;
00063
00064     fprintf(file, "%%!PS-Adobe-3.0 EPSF-3.0\n");
00065     fprintf(file, "%%%%BoundingBox: 0 0 %d %d\n", width, height);
00066     fprintf(file, "/Courier findfont 10 scalefont setfont\n");
00067     fprintf(file, "1 setlinecap\n");
00068     fprintf(file, "0.5 setlinewidth\n");
00069     fprintf(file, "newpath\n");
00070
00071     double positions[MAX_DOCS][2];
00072
00073     for (int i = 0; i < graph->total_docs; i++)
00074     {
00075         double angle = i * scale;
00076         positions[i][0] = centerX + (centerX - margin) * cos(angle);
00077         positions[i][1] = centerY + (centerY - margin) * sin(angle);
00078
00079         double red = (rand() % 256) / 255.0;
00080         double green = (rand() % 256) / 255.0;
00081         double blue = (rand() % 256) / 255.0;
```

```
00082         fprintf(file, "%f %f %f setrgbcolor\n", red, green, blue);
00083
00084         fprintf(file, "newpath\n");
00085         fprintf(file, "%.2f %.2f %d 0 360 arc fill\n", positions[i][0], positions[i][1], radius);
00086
00087         fprintf(file, "0 setgray\n");
00088         fprintf(file, "%.2f %.2f moveto (%d: %.3f) show\n", positions[i][0] - radius, positions[i][1]
      - 2 * radius, i, pagerank[i]);
00089     }
00090
00091     // Dibujar enlaces entre nodos. ESTO NO SE PUEDE VISUALIZAR, *ARREGLAR*
00092     fprintf(file, "0 0 0 setrgbcolor\n"); // Color negro para las líneas.
00093     for (int i = 0; i < graph->total_docs; i++)
00094     {
00095         Node *current = graph->output_adjacent_list[i];
00096         while (current)
00097         {
00098             int destination = current->doc_id;
00099             fprintf(file, "newpath\n");
00100             fprintf(file, "%.2f %.2f moveto %.2f %.2f lineto stroke\n", positions[i][0],
      positions[i][1], positions[destination][0], positions[destination][1]);
00101             current = current->next;
00102         }
00103     }
00104
00105     fprintf(file, "showpage\n");
00106     fclose(file);
00107
00108     fprintf(stdout, "\nArchivo EPS generado: %s\n\n", filename);
00109 }
```

5.13 src/inverted_index.c File Reference

Archivo que contiene las funciones del índice invertido.

```
#include "graph.h"
#include "inverted_index.h"
```

Functions

- InvertedIndex * create_new_node (char *word)

  < *Incluye la definición de las estructuras y funciones del índice invertido.*
- void add_document (InvertedIndex **hash_table, int doc_id, char *word)

  Agrega un documento al índice invertido, asociándolo con una palabra específica.
- bool is_stopword (char *token)

  Verifica si un token es una palabra irrelevante (stopword) que debe ignorarse en el índice.
- void tokenize_text (char *text, int doc_id, InvertedIndex **index)

  Tokeniza el texto de entrada, eliminando puntuación y stopwords, e indexa cada palabra.
- void print_inverted_index (InvertedIndex **index)

  Imprime el índice invertido mostrando cada palabra y los documentos asociados.
- unsigned int hash_function (char *word)

  Función hash para obtener un índice basado en el valor ASCII de los caracteres de una palabra.
- Node * search_word (InvertedIndex **hash_table, char *word)

  Busca una palabra en el índice invertido y retorna la lista de documentos asociados.
- void print_search_word (InvertedIndex **index, char *word_to_search)

  Imprime los documentos y la frecuencia de aparición de una palabra en el índice.
- void build_index (Graph *graph, InvertedIndex **index)

  Lee los archivos asociados al grafo y crea el índice invertido.
- void release_inverted_index (InvertedIndex **hash_table)

  Libera la memoria del índice invertido y los nodos de documentos asociados.

5.13.1 Detailed Description

Archivo que contiene las funciones del índice invertido.

Date

18-11-2024

Authors

Miguel Loaiza, Diego Sanhueza, Miguel Maripillan y Felipe Cárcamo

Contiene la implementación de las funciones que crean y manipulan el índice invertido.

Definition in file inverted_index.c.

5.13.2 Function Documentation

5.13.2.1 add_document()

```
void add_document (
            InvertedIndex ** hash_table,
            int doc_id,
            char * word)
```

Agrega un documento al índice invertido, asociándolo con una palabra específica.

Parameters

hash_table	Tabla hash que contiene el índice invertido.
doc_id	ID del documento donde aparece la palabra.
word	Palabra que se indexará.

Calcula el índice en la tabla hash para la palabra. Busca la palabra en la lista enlazada correspondiente al índice.

```
unsigned int index = hash_function(word);
InvertedIndex *current = hash_table[index];
while (current != NULL)
{
    if (strcmp(current->word, word) == 0)
    {
        // La palabra ya está en el índice; agrega el documento.
        Node *new_doc = (Node *)malloc(sizeof(Node));
        new_doc->doc_id = doc_id;
        new_doc->next = current->docs_list;
        current->docs_list = new_doc;
        return;
    }
    current = current->next;
}
```

Si la palabra no está en el índice, crea un nuevo nodo y la agrega al índice. También asigna memoria para un nodo que representa el documento.

```
InvertedIndex *new_node = create_new_node(word);
new_node->next = hash_table[index];
hash_table[index] = new_node;

Node *new_doc = (Node *)malloc(sizeof(Node));
if (new_doc == NULL)
{
    fprintf(stderr, "Error al asignar memoria para nuevo documento\n");
    exit(EXIT_FAILURE);
}
new_doc->doc_id = doc_id;
new_doc->next = NULL;
new_node->docs_list = new_doc;
```

Definition at line 59 of file inverted_index.c.

5.13.2.2 build_index()

```
void build_index (
            Graph * graph,
            InvertedIndex ** index)
```

Lee los archivos asociados al grafo y crea el índice invertido.

Lee los archivos que se encuentran guardados en el grafo y crea el índice.

Parameters

graph	Grafo que contiene los datos de los documentos.
index	Tabla hash que contiene el índice invertido.

Itera sobre los documentos identificados en el grafo.

```
for (int i = 0; i < graph->total_docs; i++)
{
    FILE *file = fopen(graph->mapping_docs[i].name, "r");
    if (file == NULL)
    {
        fprintf(stderr, "No se pudo abrir el archivo %s\n", graph->mapping_docs[i].name);
        continue;
    }
}
```

Intenta abrir el archivo correspondiente al documento. Si el archivo no puede ser abierto, muestra un error y pasa al siguiente documento.

```
FILE *file = fopen(graph->mapping_docs[i].name, "r");
if (file == NULL)
{
    fprintf(stderr, "No se pudo abrir el archivo %s\n", graph->mapping_docs[i].name);
    continue;
}
```

Define un buffer para leer cada línea del archivo.

```
char buffer[1024];
while (fgets(buffer, sizeof(buffer), file))
{
    tokenize_text(buffer, graph->mapping_docs[i].doc_id, index);
}
```

Procesa cada línea del archivo y la pasa a la función tokenize_text, que procesa el texto y lo agrega al índice invertido.

```
tokenize_text(buffer, graph->mapping_docs[i].doc_id, index);
```

Cierra el archivo una vez que ha terminado de procesarlo.

```
fclose(file);
```

Definition at line 509 of file inverted_index.c.

5.13.2.3 create_new_node()

```
InvertedIndex * create_new_node (
            char * word)
```

< Incluye la definición de las estructuras y funciones del índice invertido.

Crea un nuevo nodo en el índice invertido para una palabra específica.

< Incluye la definición de las estructuras y funciones del grafo.

Crea un nuevo nodo en el índice invertido para una palabra específica.

Parameters

word	La palabra a agregar.

Returns

Un puntero al nuevo nodo creado.

Asigna memoria para el nuevo nodo del índice invertido. Si la asignación falla, se imprime un mensaje de error y el programa termina.

```
InvertedIndex *new_node = (InvertedIndex *)malloc(sizeof(InvertedIndex));
if (new_node == NULL)
{
    fprintf(stderr, "Error al asignar memoria\n");
    exit(EXIT_FAILURE);
}
```

Copia la palabra en el nodo y establece la lista de documentos y el puntero siguiente a NULL. Inicialmente, no hay documentos asociados con esta palabra, y el puntero al siguiente nodo también es NULL.

```
strcpy(new_node->word, word);
new_node->docs_list = NULL;
new_node->next = NULL;
```

Definition at line 17 of file inverted_index.c.

5.13.2.4 hash_function()

```
unsigned int hash_function (
            char * word)
```

Función hash para obtener un índice basado en el valor ASCII de los caracteres de una palabra.

Parameters

word	Palabra a procesar.

Returns

Valor hash de la palabra.

Inicializa el valor hash en cero.

```
unsigned int hash = 0;
```

Recorre cada carácter en la palabra y actualiza el valor hash. El hash se calcula multiplicando el valor anterior por 31 y sumando el código ASCII de cada carácter.

```
for (int i = 0; word[i] != '\0'; i++)
    hash = (hash * 31) + word[i];
```

Limita el valor hash por el tamaño máximo de la tabla hash.

```
return hash % HASH_TABLE_SIZE;
```

Definition at line 330 of file inverted_index.c.

5.13.2.5 is_stopword()

```
bool is_stopword (
            char * token)
```

Verifica si un token es una palabra irrelevante (stopword) que debe ignorarse en el índice.

Convierte todo el texto a minúsculas y reemplaza caracteres de puntuación por espacios.

Parameters

token	El token a analizar.

Returns

`true` si el token es una stopword, `false` en caso contrario.

Lista de palabras consideradas como stopwords.

```
static const char *stopwords[] = {"a", "e", "i", "o", "u", "link"};
static const int num_stopwords = sizeof(stopwords) / sizeof(stopwords[0]);
```

Convierte el token a minúsculas para evitar problemas con mayúsculas/minúsculas.

```
for (int i = 0; token[i]; i++)
{
    token[i] = tolower((unsigned char)token[i]);
}
```

Compara el token con las stopwords. Si el token es una stopword, devuelve `true`.

```
for (int i = 0; i < num_stopwords; i++)
{
    if (strcmp(token, stopwords[i]) == 0)
    {
        return true;
    }
}
```

Verificación del patrón "docN", donde N es un número. Si el token comienza con "doc" seguido de un número, devuelve `true`.

```
if (strncmp(token, "doc", 3) == 0 && isdigit(token[3]))
{
    return true;
}
```

Definition at line 137 of file inverted_index.c.

5.13.2.6 print_inverted_index()

```
void print_inverted_index (
            InvertedIndex ** index)
```

Imprime el índice invertido mostrando cada palabra y los documentos asociados.

Parameters

index	Tabla hash que contiene el índice invertido.

Recorre la tabla hash y muestra las palabras y sus documentos asociados.

```
for (int i = 0; i < HASH_TABLE_SIZE; i++)
{
    if (index[i] != NULL)
    {
        InvertedIndex *current = index[i];
        while (current != NULL)
        {
            fprintf(stdout, "Palabra: %s - Documentos: ", current->word);
            Node *doc_node = current->docs_list;
            while (doc_node != NULL)
            {
                fprintf(stdout, "%d ", doc_node->doc_id);
                doc_node = doc_node->next;
            }
            fprintf(stdout, "\n");
            current = current->next;
        }
    }
}
```

Definition at line 275 of file inverted_index.c.

5.13.2.7 print_search_word()

```
void print_search_word (
            InvertedIndex ** index,
            char * word_to_search)
```

Imprime los documentos y la frecuencia de aparición de una palabra en el índice.

Imprime el índice invertido mostrando cada palabra asociada a los documentos.

Parameters

index	Tabla hash que contiene el índice invertido.
word_to_search	Palabra a buscar en el índice.

Llama a la función search_word para obtener los documentos asociados a la palabra.

```
Node *results = search_word(index, word_to_search);
```

Si no se encuentran resultados, muestra un mensaje de error.

```
if (!results)
{
    fprintf(stderr, "Palabra '%s' no encontrada.\n", word_to_search);
    return;
}
```

Muestra los documentos en los que se encuentra la palabra.

```
fprintf(stdout, "\nLa palabra '%s' se encuentra en los siguientes documentos:\n\n", word_to_search);
```

Inicializa un contador para registrar cuántas veces aparece la palabra en cada documento.

```
int doc_count[MAX_DOCS] = {0};
```

Recorre la lista de documentos y cuenta cuántas veces aparece la palabra en cada documento.

```
while (current != NULL)
{
    doc_count[current->doc_id]++;
    current = current->next;
}
```

Muestra la cantidad de veces que aparece la palabra en cada documento.

```
for (int i = 0; i < MAX_DOCS; i++)
{
    if (doc_count[i] > 0)
        fprintf(stdout, "Doc%d: %d veces\n", i, doc_count[i]);
}
```

Definition at line 427 of file inverted_index.c.

5.13.2.8 release_inverted_index()

```
void release_inverted_index (
            InvertedIndex ** hash_table)
```

Libera la memoria del índice invertido y los nodos de documentos asociados.

Parameters

hash_table	Tabla hash que contiene el índice invertido.

Verifica si la tabla hash es nula y retorna si es el caso.
```
if (hash_table == NULL)
    return;
```

Recorre cada índice en la tabla hash.
```
for (int i = 0; i < HASH_TABLE_SIZE; i++)
{
    InvertedIndex *current = hash_table[i];
    ...
}
```

Procesa la lista de palabras asociada a cada índice en la tabla hash.
```
InvertedIndex *current = hash_table[i];
while (current != NULL)
{
    Node *doc_node = current->docs_list;
    ...
}
```

Libera la memoria de cada palabra y su lista de documentos.
```
while (current != NULL)
{
    Node *doc_node = current->docs_list;
    while (doc_node != NULL)
    {
        Node *temp_doc_node = doc_node;
        doc_node = doc_node->next;
        free(temp_doc_node);
    }
    InvertedIndex *temp = current;
    current = current->next;
    free(temp);
}
```

Libera cada nodo de documento.
```
Node *temp_doc_node = doc_node;
doc_node = doc_node->next;
free(temp_doc_node);
```

Libera el nodo de la palabra.
```
InvertedIndex *temp = current;
current = current->next;
free(temp);
```

Establece el índice en la tabla hash como NULL después de liberar su memoria.
```
hash_table[i] = NULL;
```

Definition at line 583 of file inverted_index.c.

5.13.2.9 search_word()

```
Node * search_word (
            InvertedIndex ** hash_table,
            char * word)
```

Busca una palabra en el índice invertido y retorna la lista de documentos asociados.

Busca una palabra en el índice invertido.

Parameters

hash_table	Tabla hash que contiene el índice invertido.
word	Palabra que se busca en el índice.

Returns

Lista de nodos de documentos donde aparece la palabra, o `NULL` si no se encuentra.

Convierte la palabra a minúsculas para una comparación uniforme.

```
for (int i = 0; word[i]; i++)
    word[i] = tolower(word[i]);
```

Obtiene el índice en la tabla hash utilizando la función hash.

```
unsigned int index = hash_function(word);
```

Inicia la búsqueda en la lista de nodos de la tabla hash.

```
InvertedIndex *current = hash_table[index];
```

Recorre los nodos en busca de la palabra, comparando cada nodo con la palabra buscada. Si se encuentra, se retorna la lista de documentos asociados.

```
while (current != NULL)
{
    if (strcmp(current->word, word) == 0)
        return current->docs_list;
    current = current->next;
}
```

Retorna NULL si la palabra no se encuentra en la tabla hash.

```
return NULL;
```

Definition at line 366 of file inverted_index.c.

5.13.2.10 tokenize_text()

```
void tokenize_text (
            char * text,
            int doc_id,
            InvertedIndex ** index)
```

Tokeniza el texto de entrada, eliminando puntuación y stopwords, e indexa cada palabra.

Tokeniza el texto y agrega las palabras al índice invertido.

Parameters

text	Texto de entrada que será tokenizado.
doc↩ _id	ID del documento al que pertenece el texto.
index	Tabla hash que contiene el índice invertido.

Convierte todo el texto a minúsculas.

```
for (int i = 0; text[i]; i++)
    text[i] = tolower(text[i]);
```

Reemplaza caracteres de puntuación por espacios.

```
for (int i = 0; text[i]; i++)
    if (ispunct(text[i]))
        text[i] = ' ';
```

Separa el texto en palabras utilizando el espacio como delimitador.

```
token = strtok(text, " ");
```

Procesa cada token.

```
while (token != NULL)
{
    if (strlen(token) >= MAX_WORD_SIZE)
    {
        fprintf(stderr, "La palabra es muy larga\n");
        exit(EXIT_FAILURE);
    }
    if (!is_stopword(token))
        add_document(index, doc_id, token);
    token = strtok(NULL, " ");
}
```

Definition at line 207 of file inverted_index.c.

5.14 inverted_index.c

Go to the documentation of this file.

```
00001
00009 #include "graph.h"
00010 #include "inverted_index.h"
00011
00017 InvertedIndex *create_new_node(char *word)
00018 {
00031     InvertedIndex *new_node = (InvertedIndex *)malloc(sizeof(InvertedIndex));
00032     if (new_node == NULL)
00033     {
00034         fprintf(stderr, "Error al asignar memoria\n");
00035         exit(EXIT_FAILURE);
00036     }
00037
00047     strcpy(new_node->word, word);
00048     new_node->docs_list = NULL;
00049     new_node->next = NULL;
00050     return new_node;
00051 }
00052
00059 void add_document(InvertedIndex **hash_table, int doc_id, char *word)
00060 {
00082     unsigned int index = hash_function(word);
00083     InvertedIndex *current = hash_table[index];
00084
00085     while (current != NULL)
00086     {
00087         if (strcmp(current->word, word) == 0)
00088         {
00089             Node *new_doc = (Node *)malloc(sizeof(Node));
00090             new_doc->doc_id = doc_id;
00091             new_doc->next = current->docs_list;
00092             current->docs_list = new_doc;
00093             return;
00094         }
00095         current = current->next;
00096     }
00097
00117     InvertedIndex *new_node = create_new_node(word);
00118     new_node->next = hash_table[index];
00119     hash_table[index] = new_node;
00120
00121     Node *new_doc = (Node *)malloc(sizeof(Node));
00122     if (new_doc == NULL)
00123     {
00124         fprintf(stderr, "Error al asignar memoria para nuevo documento\n");
00125         exit(EXIT_FAILURE);
00126     }
00127     new_doc->doc_id = doc_id;
00128     new_doc->next = NULL;
00129     new_node->docs_list = new_doc;
00130 }
00131
00137 bool is_stopword(char *token)
00138 {
00146     static const char *stopwords[] = {"a", "e", "i", "o", "u", "link"};
00147     static const int num_stopwords = sizeof(stopwords) / sizeof(stopwords[0]);
00148
00158     for (int i = 0; token[i]; i++)
00159     {
00160         token[i] = tolower((unsigned char)token[i]);
00161     }
00162
00175     for (int i = 0; i < num_stopwords; i++)
00176     {
00177         if (strcmp(token, stopwords[i]) == 0)
00178         {
00179             return true;
00180         }
00181     }
00182
00193     if (strncmp(token, "doc", 3) == 0 && isdigit(token[3]))
00194     {
00195         return true;
00196     }
00197
00198     return false;
00199 }
00200
00207 void tokenize_text(char *text, int doc_id, InvertedIndex **index)
00208 {
00209     char *token;
00210
```

```
00218     for (int i = 0; text[i]; i++)
00219         text[i] = tolower(text[i]);
00220
00229     for (int i = 0; text[i]; i++)
00230         if (ispunct(text[i]))
00231             text[i] = ' ';
00232
00239     token = strtok(text, " ");
00240
00257     while (token != NULL)
00258     {
00259         if (strlen(token) >= MAX_WORD_SIZE)
00260         {
00261             fprintf(stderr, "La palabra es muy larga\n");
00262             exit(EXIT_FAILURE);
00263         }
00264
00265         if (!is_stopword(token))
00266             add_document(index, doc_id, token);
00267         token = strtok(NULL, " ");
00268     }
00269 }
00270
00275 void print_inverted_index(InvertedIndex **index)
00276 {
00277     fprintf(stdout, "\nIndice invertido:\n");
00278
00303     for (int i = 0; i < HASH_TABLE_SIZE; i++)
00304     {
00305         if (index[i] != NULL)
00306         {
00307             InvertedIndex *current = index[i];
00308             while (current != NULL)
00309             {
00310                 fprintf(stdout, "Palabra: %s - Documentos: ", current->word);
00311                 Node *doc_node = current->docs_list;
00312                 while (doc_node != NULL)
00313                 {
00314                     fprintf(stdout, "%d ", doc_node->doc_id);
00315                     doc_node = doc_node->next;
00316                 }
00317                 fprintf(stdout, "\n");
00318                 current = current->next;
00319             }
00320         }
00321     }
00322 }
00323
00330 unsigned int hash_function(char *word)
00331 {
00338     unsigned int hash = 0;
00339
00348     for (int i = 0; word[i] != '\0'; i++)
00349         hash = (hash * 31) + word[i];
00350
00357     return hash % HASH_TABLE_SIZE;
00358 }
00359
00366 Node *search_word(InvertedIndex **hash_table, char *word)
00367 {
00375     for (int i = 0; word[i]; i++)
00376         word[i] = tolower(word[i]);
00377
00384     unsigned int index = hash_function(word);
00385
00392     InvertedIndex *current = hash_table[index];
00393
00406     while (current != NULL)
00407     {
00408         if (strcmp(current->word, word) == 0)
00409             return current->docs_list; // Retorna la lista de documentos si se encuentra la palabra.
00410         current = current->next;
00411     }
00412
00419     return NULL;
00420 }
00421
00427 void print_search_word(InvertedIndex **index, char *word_to_search)
00428 {
00435     Node *results = search_word(index, word_to_search);
00436
00447     if (!results)
00448     {
00449         fprintf(stderr, "Palabra '%s' no encontrada.\n", word_to_search);
00450         return;
00451     }
00452
```

```
00459     fprintf(stdout, "\nLa palabra '%s' se encuentra en los siguientes documentos:\n\n",
      word_to_search);
00460
00467     int doc_count[MAX_DOCS] = {0};
00468     Node *current = results;
00469
00480     while (current != NULL)
00481     {
00482         doc_count[current->doc_id]++;
00483         current = current->next;
00484     }
00485
00496     for (int i = 0; i < MAX_DOCS; i++)
00497     {
00498         if (doc_count[i] > 0)
00499         {
00500             fprintf(stdout, "Doc%d: %d veces\n", i, doc_count[i]);
00501         }
00502     }
00503 }
00509 void build_index(Graph *graph, InvertedIndex **index)
00510 {
00525     for (int i = 0; i < graph->total_docs; i++)
00526     {
00539         FILE *file = fopen(graph->mapping_docs[i].name, "r");
00540         if (file == NULL)
00541         {
00542             fprintf(stderr, "No se pudo abrir el archivo %s\n", graph->mapping_docs[i].name);
00543             continue;
00544         }
00545
00556         char buffer[1024];
00557         while (fgets(buffer, sizeof(buffer), file))
00558         {
00566             tokenize_text(buffer, graph->mapping_docs[i].doc_id, index);
00567         }
00568
00575         fclose(file);
00576     }
00577 }
00578
00583 void release_inverted_index(InvertedIndex **hash_table)
00584 {
00592     if (hash_table == NULL)
00593         return; // Verifica si la tabla hash es nula.
00594
00605     for (int i = 0; i < HASH_TABLE_SIZE; i++)
00606     {
00618         InvertedIndex *current = hash_table[i];
00619
00638         while (current != NULL)
00639         {
00640             Node *doc_node = current->docs_list;
00641             while (doc_node != NULL)
00642             {
00651                 Node *temp_doc_node = doc_node;
00652                 doc_node = doc_node->next;
00653                 free(temp_doc_node); // Libera cada nodo de documento.
00654             }
00663             InvertedIndex *temp = current;
00664             current = current->next;
00665             free(temp); // Libera el nodo de la palabra.
00666         }
00673         hash_table[i] = NULL;
00674     }
00675 }
```

5.15 src/main.c File Reference

Función principal de menejo de funciones (grafos, pagerank e indice invertido).

```
#include "graph.h"
#include "inverted_index.h"
#include "pagerank.h"
#include "doc.h"
```

Functions

- int main (int argc, char ∗argv[ ])
  < *Librería que contiene las funciones para generar archivos.*

5.15.1 Detailed Description

Función principal de menejo de funciones (grafos, pagerank e indice invertido).

Date

18-11-2024

Authors

Miguel Loaiza, Diego Sanhueza, Miguel Maripillan y Felipe Cárcamo

Contiene la función principal del programa que manipula los grafos y genera el PageRank y el Índice Invertido.

Definition in file main.c.

5.15.2 Function Documentation

5.15.2.1 main()

```
int main (
            int argc,
            char * argv[ ])
```

< Librería que contiene las funciones para generar archivos.

< Librería que contiene las funciones del grafo. < Librería que contiene las funciones del índice invertido. < Librería que contiene las funciones del PageRank.

Función principal del programa.

Parameters

argc	Cantidad de argumentos.
argv	Argumentos.

Returns

EXIT_SUCCESS si el programa termina correctamente.

Se LEEN los ARGUMENTOS de la TERMINAL y se guardan en la variable word_to_search. Mientras se LEEN los ARGUMENTOS, se VALIDAN y se IMPRIMEN MENSAJES de AYUDA o ERROR.

```
int opt;
char *word_to_search = NULL;
while ((opt = getopt(argc, argv, "hs:")) != -1)
{
  switch (opt)
     *Casos como 'h' que es la ayuda*
     *Casos como 's' que es la palabra buscada*
     *Casos como '?' que es un error*
}
```

Se inicializan las variables necesarias para el cálculo del PageRank, la construcción del índice invertido y la creación del grafo. Se llaman a las funciones primordiales.

```
srand(time(NULL));
double pagerank[MAX_DOCS];
Graph graph;
InvertedIndex *index[HASH_TABLE_SIZE];
for (int i = 0; i < HASH_TABLE_SIZE; i++)
    index[i] = NULL;
initialize_graph(&graph);
build_graph(&graph);
calculate_pagerank(&graph, pagerank);
display_pagerank(&graph, pagerank);
show_graph(&graph);
build_index(&graph, index);
print_search_word(index, word_to_search);
release_inverted_index(index);
release_graph(&graph);
generate_eps(&graph, pagerank, "graph.eps");
return EXIT_SUCCESS;
```

Definition at line 20 of file main.c.

5.16 main.c

Go to the documentation of this file.

```
00001
00009 #include "graph.h"
00010 #include "inverted_index.h"
00011 #include "pagerank.h"
00012 #include "doc.h"
00013
00020 int main(int argc, char *argv[])
00021 {
00036     int opt;
00037     char *word_to_search = NULL;
00038
00039     while ((opt = getopt(argc, argv, "hs:")) != -1)
00040     {
00041         switch (opt)
00042         {
00043         case 'h':
00044             fprintf(stdout, "\nPara ingresar la palabra a buscar, por favor coloque el parámetro <-s>
      <numero_de_archivos>\n\n");
00045             break;
00046         case 's':
00047                 word_to_search = strdup(optarg);
00048             break;
00049         case '?':
00050             fprintf(stderr, "Opción no reconocida: -%c\n", optopt);
00051             exit(EXIT_FAILURE);
00052         default:
00053             fprintf(stderr, "Uso: %s [-h] [-s palabra_a_buscar]\n", argv[0]);
00054             exit(EXIT_FAILURE);
00055         }
00056     }
00057
00080     srand(time(NULL));
00081
00082     double pagerank[MAX_DOCS]; // Array para almacenar los valores de PageRank.
00083     Graph graph;
```

```
00084     InvertedIndex *index[HASH_TABLE_SIZE];
00085
00086     initialize_graph(&graph);
00087     build_graph(&graph);
00088     calculate_pagerank(&graph, pagerank);
00089     display_pagerank(&graph, pagerank);
00090     show_graph(&graph);
00091     build_index(&graph, index);
00092     print_search_word(index, word_to_search);
00093
00094     free(word_to_search);
00095     release_inverted_index(index);
00096     release_graph(&graph);
00097     generate_eps(&graph, pagerank, "graph.eps");
00098
00099     return EXIT_SUCCESS;
00100 }
```

5.17 src2/main.c File Reference

Función principal de creación de archivos.

```
#include "graph.h"
#include "doc.h"
```

Functions

- int main (int argc, char ∗argv[ ])

  < *Librería que contiene las funciones para generar archivos.*

5.17.1 Detailed Description

Función principal de creación de archivos.

Date

18-11-2024

Authors

Miguel Loaiza, Diego Sanhueza, Miguel Maripillan y Felipe Carcamo

Contiene la función principal del programa que crea los archivos de texto.

Definition in file main.c.

5.17.2 Function Documentation

5.17.2.1 main()

```
int main (
            int argc,
            char * argv[ ])
```

< Librería que contiene las funciones para generar archivos.

< Librería que contiene las funciones del grafo.

Función principal del programa.

Parameters

argc	Cantidad de argumentos.
argv	Argumentos.

Returns

EXIT_SUCCESS si el programa termina correctamente.

Se LEEN los ARGUMENTOS de la TERMINAL y se guardan en las variables num_docs y num_characters. Mientras se LEEN los ARGUMENTOS, se VALIDAN y se IMPRIMEN MENSAJES de AYUDA o ERROR.

```
int opt;
int num_docs = 0;
int num_characters = 0;
while ((opt = getopt(argc, argv, "hd:c:")) != -1)
{
  switch (opt)
     *Casos como 'h' que es la ayuda*
     *Casos como 'd' que es el numero de archivos*
     *Casos como 'c' que es el numero de caracteres*
     *Casos como '?' que es un error*
}
```

Se VALIDAN los ARGUMENTOS de la TERMINAL. Se verifica que se hayan proporcionado valores válidos para -d y -c. Se llama a la función para crear archivo de texto.

```
if (num_docs <= 0 || num_characters <= 0)
{
     fprintf(stderr, "Error: Debes especificar valores positivos para -d y -c.\n");
     fprintf(stderr, "Uso: %s [-h] [-d numero_de_archivos] [-c numero_de_caracteres]\n", argv[0]);
     exit(EXIT_FAILURE);
}
srand(time(NULL));
generate_text_files(num_docs, num_characters);
return EXIT_SUCCESS;
```

Definition at line 18 of file main.c.

5.18 main.c

Go to the documentation of this file.

```
00001
00009 #include "graph.h"
00010 #include "doc.h"
00011
00018 int main(int argc, char *argv[])
00019 {
00036     int opt;
00037     int num_docs = 0;
00038     int num_characters = 0;
00039
00040     while ((opt = getopt(argc, argv, "hd:c:")) != -1)
00041     {
00042         switch (opt)
00043         {
00044         case 'h':
00045             fprintf(stdout, "\nUso del programa:\n");
00046             fprintf(stdout, "  -d <numero_de_archivos> : Número de archivos a crear.\n");
00047             fprintf(stdout, "  -c <numero_de_caracteres> : Número de caracteres por archivo.\n");
00048             fprintf(stdout, "  -h : Muestra esta ayuda.\n\n");
00049             exit(EXIT_SUCCESS);
00050             break;
00051         case 'd':
00052             num_docs = atoi(optarg);
00053             break;
00054         case 'c':
00055             num_characters = atoi(optarg);
00056             break;
00057         case '?':
00058             fprintf(stderr, "Opción no reconocida: -%c\n", optopt);
00059             exit(EXIT_FAILURE);
00060         default:
```

```
00061             fprintf(stderr, "Uso: %s [-h] [-d numero_de_archivos] [-c numero_de_caracteres]\n",
      argv[0]);
00062             exit(EXIT_FAILURE);
00063         }
00064     }
00065
00080     if (num_docs <= 0 || num_characters <= 0)
00081     {
00082         fprintf(stderr, "Error: Debes especificar valores positivos para -d y -c.\n");
00083         fprintf(stderr, "Uso: %s [-h] [-d numero_de_archivos] [-c numero_de_caracteres]\n", argv[0]);
00084         exit(EXIT_FAILURE);
00085     }
00086
00087     srand(time(NULL));
00088
00089     fprintf(stdout, "\nCREANDO %d archivos de texto con %d caracteres cada uno...\n\n", num_docs,
      num_characters);
00090     generate_text_files(num_docs, num_characters);
00091     fprintf(stdout, "\nArchivos de texto creados con ÉXITO.\n\n");
00092
00093     return EXIT_SUCCESS;
00094 }
```

5.19 src/pagerank.c File Reference

Archivo que contiene las funciones de PageRank.

```
#include "graph.h"
```

Functions

- void initialize_pagerank (double *pagerank, int num_docs)

  < *Incluye la definición de las estructuras y funciones del grafo.*
- void calculate_pagerank (Graph *graph, double *pagerank)

  Calcular PageRank.
- void display_pagerank (Graph *graph, double *pagerank)

  Mostrar PageRank.

5.19.1 Detailed Description

Archivo que contiene las funciones de PageRank.

Date

18-11-2024

Authors

Miguel Loaiza, Diego Sanhueza, Miguel Maripillan y Felipe Cárcamo

Contiene la implementación de las funciones que calculan el PageRank de cada documento en el grafo.

Definition in file pagerank.c.

5.19.2 Function Documentation

5.19.2.1 calculate_pagerank()

```
void calculate_pagerank (
            Graph * graph,
            double * pagerank)
```

Calcular PageRank.

Parameters

graph	Grafo
pagerank	Arreglo de PageRank

Inicializa variables y PageRank.

```
int num_docs = graph->total_docs;
initialize_pagerank(pagerank, num_docs);
double temp_rank[MAX_DOCS];
```

Itera hasta que se cumple el criterio de convergencia. Calcula la contribución de cada nodo a los que apunta. Distribuye la contribución de PageRank a cada nodo de la lista de adyacencia.

```
for (int iteration = 0; iteration < MAX_ITERATIONS; iteration++)
{
     for (int i = 0; i < num_docs; i++)
         temp_rank[i] = (1 - DAMPING_FACTOR) / num_docs;
     for (int i = 0; i < num_docs; i++)
     {
         int num_links = count_output_links(graph, i);
         if (num_links == 0)
             continue;
         double rank_contribution = pagerank[i] * DAMPING_FACTOR / num_links;
         Node *current = graph->output_adjacent_list[i];
         while (current != NULL)
         {
             temp_rank[current->doc_id] += rank_contribution;
             current = current->next;
         }
     }
```

Calcula el error y actualiza los valores de PageRank. Si el error es menor al umbral de convergencia, se detiene el algoritmo.

```
double error = 0;
for (int i = 0; i < num_docs; i++)
{
     error += fabs(pagerank[i] - temp_rank[i]);
     pagerank[i] = temp_rank[i];
}
if (error < CONVERGENCE_THRESHOLD)
   break;
```

Definition at line 34 of file pagerank.c.

5.19.2.2 display_pagerank()

```
void display_pagerank (
             Graph * graph,
             double * pagerank)
```

Mostrar PageRank.

Parameters

graph	Grafo
pagerank	Arreglo de PageRank

Muestra los valores de PageRank de cada documento.

```
fprintf(stdout, "\nValores de PageRank ordenados por importancia:\n\n");
int num_docs = graph->total_docs;
int indices[num_docs];
for (int i = 0; i < num_docs; i++)
    indices[i] = i;
for (int i = 0; i < num_docs - 1; i++)
{
    for (int j = i + 1; j < num_docs; j++)
    {
```

```
        if (pagerank[indices[i]] < pagerank[indices[j]])
        {
            int temp = indices[i];
            indices[i] = indices[j];
            indices[j] = temp;
        }
    }
}
for (int i = 0; i < num_docs; i++)
{
    int doc_id = indices[i];
     fprintf(stdout, "Documento (%s): PageRank = %.6f\n", graph->mapping_docs[doc_id].name,
      pagerank[doc_id]);
}
```

Definition at line 126 of file pagerank.c.

5.19.2.3 initialize_pagerank()

```
void initialize_pagerank (
            double * pagerank,
            int num_docs)
```

< Incluye la definición de las estructuras y funciones del grafo.

Inicializar PageRank

Parameters

pagerank	Arreglo de PageRank
num_docs	Número de documentos

Inicializa cada documento con el mismo valor de PageRank.

```
for (int i = 0; i < num_docs; i++)
    pagerank[i] = 1.0 / num_docs;
```

Definition at line 16 of file pagerank.c.

5.20 pagerank.c

Go to the documentation of this file.

```
00001
00009 #include "graph.h"
00010
00016 void initialize_pagerank(double *pagerank, int num_docs)
00017 {
00025     for (int i = 0; i < num_docs; i++)
00026         pagerank[i] = 1.0 / num_docs;
00027 }
00028
00034 void calculate_pagerank(Graph *graph, double *pagerank)
00035 {
00044     int num_docs = graph->total_docs;
00045     initialize_pagerank(pagerank, num_docs);
00046     double temp_rank[MAX_DOCS];
00047
00072     for (int iteration = 0; iteration < MAX_ITERATIONS; iteration++)
00073     {
00074         for (int i = 0; i < num_docs; i++)
00075             temp_rank[i] = (1 - DAMPING_FACTOR) / num_docs;
00076
00077         for (int i = 0; i < num_docs; i++)
00078         {
00079             int num_links = count_output_links(graph, i);
00080
00081             if (num_links == 0)
```

```
00082                 continue;
00083
00084             double rank_contribution = pagerank[i] * DAMPING_FACTOR / num_links;
00085             Node *current = graph->output_adjacent_list[i];
00086
00087             while (current != NULL)
00088             {
00089                 temp_rank[current->doc_id] += rank_contribution;
00090                 current = current->next;
00091             }
00092         }
00093
00108         double error = 0;
00109
00110         for (int i = 0; i < num_docs; i++)
00111         {
00112             error += fabs(pagerank[i] - temp_rank[i]);
00113             pagerank[i] = temp_rank[i];
00114         }
00115
00116         if (error < CONVERGENCE_THRESHOLD)
00117             break;
00118     }
00119 }
00120
00126 void display_pagerank(Graph *graph, double *pagerank)
00127 {
00155     fprintf(stdout, "\nValores de PageRank ordenados por importancia:\n\n");
00156
00157     int num_docs = graph->total_docs;
00158     int indices[num_docs];
00159
00160     for (int i = 0; i < num_docs; i++)
00161         indices[i] = i;
00162
00163     for (int i = 0; i < num_docs - 1; i++)
00164     {
00165         for (int j = i + 1; j < num_docs; j++)
00166         {
00167             if (pagerank[indices[i]] < pagerank[indices[j]])
00168             {
00169                 int temp = indices[i];
00170                 indices[i] = indices[j];
00171                 indices[j] = temp;
00172             }
00173         }
00174     }
00175
00176     for (int i = 0; i < num_docs; i++)
00177     {
00178         int doc_id = indices[i];
00179         fprintf(stdout, "Documento (%s): PageRank = %.6f\n", graph->mapping_docs[doc_id].name,
    pagerank[doc_id]);
00180     }
00181 }
```

5.21 src2/doc.c File Reference

Archivo que contiene el manejo de archivos y rellenado de estos.

```
#include "graph.h"
#include "doc.h"
```

Functions

- void generate_text_files (int num_docs, int num_characters)

  < *Incluye la definición de las funciones de generación de archivos.*
- void generate_random_text (FILE *doc, const char *doc_name, int num_docs, int num_characters, int current_doc, int *links)

  Generar texto aleatorio.

5.21.1 Detailed Description

Archivo que contiene el manejo de archivos y rellenado de estos.

Date

18-11-2024

Authors

Miguel Loaiza, Diego Sanhueza, Miguel Maripillan y Felipe Carcamo

Contiene la implementación de las funciones que generan archivos de texto simulando documentos web con contenido aleatorio.

Definition in file doc.c.

5.21.2 Function Documentation

5.21.2.1 generate_random_text()

```
void generate_random_text (
            FILE * doc,
            const char * doc_name,
            int num_docs,
            int num_characters,
            int current_doc,
            int * links)
```

Generar texto aleatorio.

Genera texto aleatorio dentro de cada archivo web.

Parameters

doc	Archivo a escribir
doc_name	Nombre del archivo
num_docs	Número de documentos
num_characters	Número de caracteres
current_doc	Documento actual
links	Conexiones entre documentos

Genera texto aleatorio dentro de cada archivo web (letras entre A y Z).

```
for (int i = 0; i < num_characters; i++)
{
     char letter = 'A' + rand() % 26;
     fprintf(doc, "%c", letter);
     if (i < num_characters - 1)
         fprintf(doc, " ");
}
```

Asegura al menos un enlace único en cada archivo web.

```
fprintf(doc, "\nlink: doc%d", links[current_doc - 1]);
int extra_links = rand() % num_docs;
for (int i = 0; i < extra_links; i++)
{
     int link_doc = rand() % num_docs + 1;
     if (link_doc != current_doc && link_doc != links[current_doc - 1])
         fprintf(doc, "\nlink: doc%d", link_doc);
}
```

Definition at line 97 of file doc.c.

5.21.2.2 generate_text_files()

```
void generate_text_files (
             int num_docs,
             int num_characters)
```

< Incluye la definición de las funciones de generación de archivos.

Genera archivos txt simulando páginas web.

< Incluye la definición de las estructuras y funciones del grafo.

Generar archivos txt

Parameters

num_docs	Cantidad de archivos a generar
num_characters	Cantidad de caracteres por archivo

Verifica que el número de archivos a generar sea válido. Crea un array de enlaces para asegurar que cada documento tenga al menos un enlace. Llama a la función generate_random_text para generar el contenido de cada archivo.

```
if (num_docs <= 0 || num_docs >= 100)
{
     fprintf(stderr, "El número de archivos web a generar debe ser MAYOR a 0 y MENOR a 100.\n\n");
     exit(EXIT_FAILURE);
}
if (num_characters <= 0 || num_characters >= 50)
{
     fprintf(stderr, "El número de caracteres por archivo debe ser MAYOR a 0 y MENOR a 50.\n\n");
     exit(EXIT_FAILURE);
}
int *links = malloc(num_docs * sizeof(int));
for (int i = 0; i < num_docs; i++)
     links[i] = (i + 1) % num_docs + 1;
for (int i = 1; i <= num_docs; i++)
{
     char doc_name[MAX_NAME_DOC];
     snprintf(doc_name, sizeof(doc_name), "doc%d.txt", i);
     FILE *doc = fopen(doc_name, "w");
     if (doc == NULL)
     {
         fprintf(stderr, "Error al abrir el archivo web.\n");
         free(links);
         exit(EXIT_FAILURE);
     }
     generate_random_text(doc, doc_name, num_docs, num_characters, i, links);
   fclose(doc);
}
```

Definition at line 18 of file doc.c.

5.22 doc.c

Go to the documentation of this file.

```
00001
00010 #include "graph.h"
00011 #include "doc.h"
00012
00018 void generate_text_files(int num_docs, int num_characters)
00019 {
00054     if (num_docs <= 0 || num_docs >= 100)
00055     {
00056         fprintf(stderr, "El número de archivos web a generar debe ser MAYOR a 0 y MENOR a 100.\n\n");
00057         exit(EXIT_FAILURE);
00058     }
00059
```

```
00060     if (num_characters <= 0 || num_characters >= 51)
00061     {
00062         fprintf(stderr, "El número de caracteres por archivo debe ser MAYOR a 0 y MENOR a 50.\n\n");
00063         exit(EXIT_FAILURE);
00064     }
00065
00066     int *links = malloc(num_docs * sizeof(int));
00067
00068     for (int i = 0; i < num_docs; i++)
00069         links[i] = (i + 1) % num_docs + 1;
00070
00071     for (int i = 1; i <= num_docs; i++)
00072     {
00073         char doc_name[MAX_NAME_DOC];
00074         snprintf(doc_name, sizeof(doc_name), "doc%d.txt", i);
00075
00076         FILE *doc = fopen(doc_name, "w");
00077         if (doc == NULL)
00078         {
00079             fprintf(stderr, "Error al abrir el archivo web.\n");
00080             free(links);
00081             exit(EXIT_FAILURE);
00082         }
00083         generate_random_text(doc, doc_name, num_docs, num_characters, i, links);
00084         fclose(doc);
00085     }
00086 }
00087
00097 void generate_random_text(FILE *doc, const char *doc_name, int num_docs, int num_characters, int
      current_doc, int *links)
00098 {
00111     for (int i = 0; i < num_characters; i++)
00112     {
00113         char letter = 'A' + rand() % 26;
00114         fprintf(doc, "%c", letter);
00115         if (i < num_characters - 1)
00116             fprintf(doc, " ");
00117     }
00118
00132     fprintf(doc, "\nlink: doc%d", links[current_doc - 1]);
00133
00134     int extra_links = rand() % num_docs;
00135     for (int i = 0; i < extra_links; i++)
00136     {
00137         int link_doc = rand() % num_docs + 1;
00138         if (link_doc != current_doc && link_doc != links[current_doc - 1])
00139             fprintf(doc, "\nlink: doc%d", link_doc);
00140     }
00141
00142     fprintf(stdout, "ARCHIVO '%s' generado con ÉXITO con %d letras y enlaces.\n", doc_name,
      num_characters);
00143 }
```

Index