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. **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`.

1.4 3. 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

Public Attributes

- char name [MAX_NAME_DOC]
- int doc_id

4.1.1 Detailed Description

Definition at line 32 of file graph.h.

4.1.2 Member Data Documentation

4.1.2.1 doc_id

```
int DocumentMapping::doc_id
```

Definition at line 35 of file graph.h.

4.1.2.2 name

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

Definition at line 34 of file graph.h.

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

- incs/graph.h

4.2 Graph Struct Reference

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

Definition at line 38 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 41 of file graph.h.

4.2.2.2 mapping_docs

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

Definition at line 42 of file graph.h.

4.2.2.3 output_adjacent_list

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

Definition at line 40 of file graph.h.

4.2.2.4 total_docs

```
int Graph::total_docs
```

Definition at line 43 of file graph.h.

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

- incs/graph.h

4.3 InvertedIndex Struct Reference

Public Attributes

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

4.3.1 Detailed Description

Definition at line 7 of file inverted_index.h.

4.3.2 Member Data Documentation

4.3.2.1 docs_list

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

Definition at line 10 of file inverted_index.h.

4.3.2.2 next

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

Definition at line 11 of file inverted_index.h.

4.3.2.3 word

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

Definition at line 9 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

Public Attributes

- int doc_id
- struct Node * next

4.4.1 Detailed Description

Definition at line 26 of file graph.h.

4.4.2 Member Data Documentation

4.4.2.1 doc_id

```
int Node::doc_id
```

Definition at line 28 of file graph.h.

4.4.2.2 next

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

Definition at line 29 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 Carcamo

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 86 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);
}
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
00010 #ifndef GENERATOR_H
00011 #define GENERATOR_H
00012
00013 #include <stdio.h>
00014
00020 void generate_text_files(int, int);
00021
00031 void generate_random_text(FILE *, const char *, int, int, int, int *);
00032
00033 #endif
```

5.3 graph.h

```
00001 #ifndef GRAPH_H
00002 #define GRAPH_H
00003
00004 /* Librerías estándar */
00005 #include <stdbool.h>
00006 #include <string.h>
00007 #include <stdlib.h>
00008 #include <stdio.h>
00009 #include <math.h>
00010 #include <ctype.h>
00011 #include <dirent.h>
00012 #include <unistd.h>
00013 #include <time.h>
00014
00015 /* Macros generales */
00016 #define CONVERGENCE_THRESHOLD 0.0001 // Umbral de convergencia del PageRank.
00017 #define MAX_CHARACTERS_DOC 50        // Máximo de caracteres por página.
00018 #define DAMPING_FACTOR 0.85          // Factor de amortiguación del PageRank.
00019 #define MAX_ITERATIONS 100           // Máximo de iteraciones del algoritmo.
00020 #define MAX_WORD_SIZE 50             // Máximo de longitud de palabras.
```

```
00021 #define MAX_NAME_DOC 20              // Máximo de longitud de nombres de archivos.
00022 #define MAX_DOCS 100                 // Máximo de documentos soportados.
00023 #define HASH_TABLE_SIZE 30           // Tamaño de la tabla hash.
00024
00025 /* Estructuras comunes */
00026 typedef struct Node
00027 {
00028     int doc_id;
00029     struct Node *next;
00030 } Node;
00031
00032 typedef struct DocumentMapping
00033 {
00034     char name[MAX_NAME_DOC];
00035     int doc_id;
00036 } DocumentMapping;
00037
00038 typedef struct Graph
00039 {
00040     Node *output_adjacent_list[MAX_DOCS];
00041     Node *input_adjacent_list[MAX_DOCS];
00042     DocumentMapping mapping_docs[MAX_DOCS];
00043     int total_docs;
00044 } Graph;
00045
00046 /* Funciones del grafo */
00047 void initialize_graph(Graph *);
00048 void add_edge(Graph *, int, int);
00049 void build_graph(Graph *);
00050 void release_graph(Graph *);
00051 int count_output_links(Graph *, int);
00052 int count_input_links(Graph *, int);
00053 int get_doc_id(Graph *, char *);
00054 bool is_doc_name(char *);
00055 void show_graph(Graph *);
00056 void generate_eps(const Graph *, const double *, const char *);
00057
00058 #endif // GRAPH_H
```

5.4 inverted_index.h

```
00001 #ifndef INVERTED_INDEX_H
00002 #define INVERTED_INDEX_H
00003
00004 #include "graph.h"
00005
00006 /* Estructura InvertedIndex */
00007 typedef struct InvertedIndex
00008 {
00009     char word[MAX_WORD_SIZE];
00010     Node *docs_list;
00011     struct InvertedIndex *next;
00012 } InvertedIndex;
00013
00014 /* Funciones del índice invertido */
00015 InvertedIndex *create_new_node(char *);
00016 void add_document(InvertedIndex **, int, char *);
00017 void tokenize_text(char *, int, InvertedIndex **);
00018 void print_inverted_index(InvertedIndex **);
00019 unsigned int hash_function(char *);
00020 Node *search_word(InvertedIndex **, char *);
00021 void print_search_word(InvertedIndex **, char *);
00022 void build_index(Graph *, InvertedIndex **);
00023 void release_inverted_index(InvertedIndex **);
00024
00025 #endif // INVERTED_INDEX_H
```

5.5 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.

5.5.1 Detailed Description

Prototipos de funciones para la creación del PageRank.

Date

18-11-2024

Authors

Miguel Loaiza, Diego Sanhueza, Miguel Maripillan y Felipe Carcamo

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

Definition in file pagerank.h.

5.5.2 Function Documentation

5.5.2.1 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.

5.5.2.2 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:\n\n");
for (int i = 0; i < graph->total_docs; i++)
    fprintf(stdout, "Documento (%s): PageRank = %.6f\n", graph->mapping_docs[i].name, pagerank[i]);
```

Definition at line 126 of file pagerank.c.

5.5.2.3 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.

5.6 pagerank.h

Go to the documentation of this file.

```
00001
00010 #ifndef PAGERANK_H
00011 #define PAGERANK_H
00012
00013 #include "graph.h"
00014
00020 void initialize_pagerank(double *, int);
00021
00027 void calculate_pagerank(Graph *, double *);
00028
00034 void display_pagerank(Graph *, double *);
00035
00036 #endif
```

5.7 graph.c

```
00001 #include "graph.h"
00002
00003 // Inicializa el grafo estableciendo listas de adyacencia y total de documentos en cero.
00004 void initialize_graph(Graph *graph)
00005 {
00006     // Inicializa listas de adyacencia a NULL.
00007     for (int i = 0; i < MAX_DOCS; i++)
00008     {
00009         graph->output_adjacent_list[i] = NULL;
00010         graph->input_adjacent_list[i] = NULL;
00011     }
00012     graph->total_docs = 0; // Inicializa el contador de documentos.
00013 }
00014
00015 // Agrega un enlace dirigido entre dos documentos en el grafo.
00016 void add_edge(Graph *graph, int source, int destination)
00017 {
00018     // Crea nodo en la lista de enlaces salientes desde el documento de origen al destino.
00019     Node *newOutputNode = (Node *)malloc(sizeof(Node));
00020     if (newOutputNode == NULL) // Verifica la asignación de memoria.
00021     {
00022         fprintf(stderr, "Error al asignar memoria para enlace saliente\n");
00023         exit(EXIT_FAILURE);
00024     }
00025     newOutputNode->doc_id = destination;                        // Asigna el ID de destino al nodo.
00026     newOutputNode->next = graph->output_adjacent_list[source]; // Inserta al inicio de la lista
00027     graph->output_adjacent_list[source] = newOutputNode;
00028
00029     // Crea nodo en la lista de enlaces entrantes al documento destino desde el origen.
00030     Node *newInputNode = (Node *)malloc(sizeof(Node));
00031     if (newInputNode == NULL) // Verifica la asignación de memoria.
00032     {
00033         fprintf(stderr, "Error al asignar memoria para enlace entrante\n");
00034         exit(EXIT_FAILURE);
00035     }
00036     newInputNode->doc_id = source;                                  // Asigna el ID del origen al nodo.
00037     newInputNode->next = graph->input_adjacent_list[destination]; // Inserta al inicio de la lista.
00038     graph->input_adjacent_list[destination] = newInputNode;
00039 }
00040
00041 // Construye el grafo a partir de documentos que contienen enlaces a otros documentos.
00042 void build_graph(Graph *graph)
00043 {
00044     DIR *dir; // Puntero al directorio actual.
00045     struct dirent *ent;
00046
00047     if ((dir = opendir(".")) == NULL) // Abre el directorio actual
00048     {
00049         fprintf(stderr, "No se pudo abrir el directorio");
00050         exit(EXIT_FAILURE);
00051     }
00052
00053     // Itera sobre los archivos en el directorio.
00054     while ((ent = readdir(dir)) != NULL)
00055     {
00056         char *file_name = ent->d_name;
00057
00058         if (is_doc_name(file_name)) // Verifica si el archivo es un documento válido.
00059         {
00060             int doc_id = get_doc_id(graph, file_name); // Obtiene el ID del documento.
00061
00062             FILE *file = fopen(file_name, "r"); // Abre el archivo para leer enlaces.
```

```
00063             if (file == NULL)
00064             {
00065                 fprintf(stderr, "No se pudo abrir el archivo %s\n", file_name);
00066                 continue;
00067             }
00068 
00069             char line[256];
00070 
00071             while (fgets(line, sizeof(line), file))
00072             {
00073                 char *ptr = line; // Apuntador para buscar enlaces en la línea.
00074 
00075                 while ((ptr = strstr(ptr, "link: doc")) != NULL) // Busca "link: doc".
00076                 {
00077                     ptr += 9; // Avanza el puntero después de "link: doc".
00078                     int doc_number;
00079 
00080                     if (sscanf(ptr, "%d", &doc_number) != 1) // Extrae el número del documento
      enlazado.
00081                     {
00082                         fprintf(stderr, "Formato de enlace inválido en %s\n", file_name);
00083                         continue;
00084                     }
00085 
00086                     if (doc_number <= 0 || doc_number > MAX_DOCS) // Verifica si el número de
      documento es válido
00087                     {
00088                         fprintf(stderr, "Número de documento inválido en enlace: %d\n", doc_number);
00089                         continue;
00090                     }
00091 
00092                     // Construye el nombre del documento destino.
00093                     char destination_name[MAX_NAME_DOC];
00094                     snprintf(destination_name, sizeof(destination_name), "doc%d.txt", doc_number);
00095 
00096                     FILE *destination_file = fopen(destination_name, "r"); // Verifica si el destino
      existe.
00097                     if (destination_file == NULL)
00098                     {
00099                         fprintf(stderr, "El documento %s enlazado desde %s no existe\n",
      destination_name, file_name);
00100                         continue;
00101                     }
00102                     fclose(destination_file);
00103 
00104                     // Obtiene el ID del documento destino.
00105                     int destination_id = get_doc_id(graph, destination_name);
00106 
00107                     // Agrega el enlace entre documentos.
00108                     add_edge(graph, doc_id, destination_id);
00109                 }
00110                 ptr++;
00111             }
00112             fclose(file);
00113         }
00114     }
00115     closedir(dir);
00116 }
00117 
00118 // Libera la memoria asignada al grafo para evitar fugas de memoria.
00119 void release_graph(Graph *graph)
00120 {
00121     for (int i = 0; i < MAX_DOCS; i++)
00122     {
00123         // Libera nodos en la lista de enlaces salientes.
00124         Node *current_output = graph->output_adjacent_list[i];
00125         while (current_output != NULL)
00126         {
00127             Node *temp = current_output;
00128             current_output = current_output->next;
00129             free(temp);
00130         }
00131         graph->output_adjacent_list[i] = NULL;
00132 
00133         // Libera nodos en la lista de enlaces entrantes.
00134         Node *current_input = graph->input_adjacent_list[i];
00135         while (current_input != NULL)
00136         {
00137             Node *temp = current_input;
00138             current_input = current_input->next;
00139             free(temp);
00140         }
00141         graph->input_adjacent_list[i] = NULL;
00142     }
00143 }
00144 
00145 // Cuenta el número de enlaces salientes de un documento.
```

```
00146 int count_output_links(Graph *graph, int doc_id)
00147 {
00148     int count = 0;
00149     Node *current = graph->output_adjacent_list[doc_id];
00150 
00151     // Recorre la lista y cuenta los enlaces salientes.
00152     while (current != NULL)
00153     {
00154         count++;
00155         current = current->next;
00156     }
00157 
00158     return count;
00159 }
00160 
00161 // Cuenta el número de enlaces entrantes a un documento.
00162 int count_input_links(Graph *graph, int doc_id)
00163 {
00164     int count = 0;
00165     Node *current = graph->input_adjacent_list[doc_id];
00166 
00167     // Recorre la lista y cuenta los enlaces entrantes.
00168     while (current != NULL)
00169     {
00170         count++;
00171         current = current->next;
00172     }
00173 
00174     return count;
00175 }
00176 
00177 // Obtiene el ID de un documento a partir de su nombre de archivo.
00178 int get_doc_id(Graph *graph, char *file_name)
00179 {
00180     // Busca el ID en el mapeo de documentos existentes.
00181     for (int i = 0; i < graph->total_docs; i++)
00182         if (strcmp(graph->mapping_docs[i].name, file_name) == 0)
00183             return graph->mapping_docs[i].doc_id;
00184 
00185     // Verifica si se ha alcanzado el límite de documentos permitidos.
00186     if (graph->total_docs >= MAX_DOCS)
00187     {
00188         fprintf(stderr, "Se ha alcanzado el número máximo de documentos\n");
00189         exit(EXIT_FAILURE);
00190     }
00191 
00192     int num_doc;
00193 
00194     // Extrae el número del documento del nombre del archivo.
00195     if (sscanf(file_name, "doc%d.txt", &num_doc) != 1)
00196     {
00197         fprintf(stderr, "Nombre de archivo inválido: %s\n", file_name);
00198         exit(EXIT_FAILURE);
00199     }
00200 
00201     // Guarda el mapeo del nuevo documento.
00202     strcpy(graph->mapping_docs[graph->total_docs].name, file_name);
00203     graph->mapping_docs[graph->total_docs].doc_id = num_doc;
00204     graph->total_docs++;
00205 
00206     return num_doc;
00207 }
00208 
00209 // Verifica si el nombre del archivo sigue el patrón "docN.txt" con N >= 1.
00210 bool is_doc_name(char *file_name)
00211 {
00212     int len = strlen(file_name);
00213     if (len < 8) // Longitud mínima para cumplir el formato.
00214         return false;
00215 
00216     if (strncmp(file_name, "doc", 3) != 0) // Verifica prefijo "doc".
00217         return false;
00218 
00219     if (strcmp(file_name + len - 4, ".txt") != 0) // Verifica sufijo ".txt".
00220         return false;
00221 
00222     // Verifica que el número entre "doc" y ".txt" sea válido.
00223     for (int i = 3; i < len - 4; i++)
00224     {
00225         if (!isdigit(file_name[i]))
00226             return false;
00227         if (file_name[i] == '0' && i == 3) // No permite doc0.txt.
00228             return false;
00229     }
00230 
00231     return true;
00232 }
```

```
00233
00234 // Muestra el grafo de enlaces, imprimiendo los documentos y sus enlaces salientes.
00235 void show_graph(Graph *graph)
00236 {
00237     fprintf(stdout, "\nGrafo de enlaces:\n\n");
00238     for (int i = 0; i < MAX_DOCS; i++)
00239     {
00240         if (graph->output_adjacent_list[i] != NULL) // Solo muestra documentos con enlaces.
00241         {
00242             fprintf(stdout, "Documento %d enlaza a: ", i);
00243             Node *current = graph->output_adjacent_list[i];
00244             while (current != NULL)
00245             {
00246                 fprintf(stdout, "%d ", current->doc_id); // Imprime el ID de documentos enlazados.
00247                 current = current->next;
00248             }
00249             fprintf(stdout, "\n");
00250         }
00251     }
00252 }
```

5.8 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.*

5.8.1 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 Carcamo

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

Definition in file graphic.c.

5.8.2 Function Documentation

5.8.2.1 generate_eps()

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

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

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.

5.9 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     // Coordenadas de los nodos.
00072     double positions[MAX_DOCS][2];
00073
00074     // Dibujar nodos con PageRank e Identificador.
00075     for (int i = 0; i < graph->total_docs; i++)
00076     {
00077         double angle = i * scale;
00078         positions[i][0] = centerX + (centerX - margin) * cos(angle);
00079         positions[i][1] = centerY + (centerY - margin) * sin(angle);
00080
00081         // Asignar colores al nodo.
```

```
00082         double red = (rand() % 256) / 255.0;
00083         double green = (rand() % 256) / 255.0;
00084         double blue = (rand() % 256) / 255.0;
00085         fprintf(file, "%f %f %f setrgbcolor\n", red, green, blue);
00086 
00087         // Dibujar nodo.
00088         fprintf(file, "newpath\n");
00089         fprintf(file, "%.2f %.2f %d 0 360 arc fill\n", positions[i][0], positions[i][1], radius);
00090 
00091         // Texto Identificador del Nodo y PageRank.
00092         fprintf(file, "0 setgray\n");
00093         fprintf(file, "%.2f %.2f moveto (%d: %.3f) show\n", positions[i][0] - radius, positions[i][1]
      - 2 * radius, i, pagerank[i]);
00094     }
00095 
00096     // Dibujar enlaces entre nodos. ESTO NO SE PUEDE VISUALIZAR, *ARREGLAR*
00097     fprintf(file, "0 0 0 setrgbcolor\n"); // Color negro para las líneas.
00098     for (int i = 0; i < graph->total_docs; i++)
00099     {
00100         Node *current = graph->output_adjacent_list[i];
00101         while (current)
00102         {
00103             int destination = current->doc_id;
00104             fprintf(file, "newpath\n");
00105             fprintf(file, "%.2f %.2f moveto %.2f %.2f lineto stroke\n", positions[i][0],
      positions[i][1], positions[destination][0], positions[destination][1]);
00106             current = current->next;
00107         }
00108     }
00109 
00110     fprintf(file, "showpage\n");
00111     fclose(file);
00112 
00113     fprintf(stdout, "\nArchivo EPS generado: %s\n\n", filename);
00114 }
```

5.10 inverted_index.c

```
00001 #include "graph.h"
00002 #include "inverted_index.h"
00003 
00004 // Crea un nuevo nodo en el índice invertido para una palabra específica.
00005 InvertedIndex *create_new_node(char *word)
00006 {
00007     // Asigna memoria para el nuevo nodo del índice invertido.
00008     InvertedIndex *new_node = (InvertedIndex *)malloc(sizeof(InvertedIndex));
00009     if (new_node == NULL)
00010     {
00011         fprintf(stderr, "Error al asignar memoria\n");
00012         exit(EXIT_FAILURE);
00013     }
00014 
00015     // Copia la palabra en el nodo y establece la lista de documentos y el puntero siguiente a NULL.
00016     strcpy(new_node->word, word);
00017     new_node->docs_list = NULL; // Inicializa en NULL porque aún no hay documentos con esta palabra.
00018     new_node->next = NULL;
00019     return new_node;
00020 }
00021 
00022 // Agrega un documento al índice invertido, asociándolo con una palabra específica.
00023 void add_document(InvertedIndex **hash_table, int doc_id, char *word)
00024 {
00025     // Calcula el índice en la tabla hash para la palabra.
00026     unsigned int index = hash_function(word);
00027     InvertedIndex *current = hash_table[index];
00028 
00029     // Busca la palabra en el índice.
00030     while (current != NULL)
00031     {
00032         if (strcmp(current->word, word) == 0)
00033         {
00034             // si la comparacion se cumple, la palabra ya esta en el indice
00035             Node *new_doc = (Node *)malloc(sizeof(Node));
00036             new_doc->doc_id = doc_id;
00037             new_doc->next = current->docs_list;
00038             current->docs_list = new_doc;
00039             return;
00040         }
00041         current = current->next;
00042     }
00043 
00044     // Si la palabra no se encuentra en el índice, crea un nuevo nodo en el índice.
00045     InvertedIndex *new_node = create_new_node(word);
```

```
00046     new_node->next = hash_table[index]; // Posiciona el nuevo nodo en el índice calculado.
00047     hash_table[index] = new_node;
00048
00049     // Crea y asigna el nodo de documento.
00050     Node *new_doc = (Node *)malloc(sizeof(Node));
00051     if (new_doc == NULL)
00052     {
00053         fprintf(stderr, "Error al asignar memoria para nuevo documento\n");
00054         exit(EXIT_FAILURE);
00055     }
00056     new_doc->doc_id = doc_id;
00057     new_doc->next = NULL;
00058     new_node->docs_list = new_doc;
00059 }
00060
00061 // Verifica si un token es una palabra irrelevante (stopword) que debe ignorarse en el índice.
00062 bool is_stopword(char *token)
00063 {
00064     static const char *stopwords[] = {"a", "e", "i", "o", "u", "link"};
00065     static const int num_stopwords = sizeof(stopwords) / sizeof(stopwords[0]);
00066
00067     // Convertir el token a minúsculas para que no haya problemas de mayúsculas/minúsculas.
00068     for (int i = 0; token[i]; i++)
00069     {
00070         token[i] = tolower((unsigned char)token[i]);
00071     }
00072
00073     // Compara el token con las stopwords.
00074     for (int i = 0; i < num_stopwords; i++)
00075     {
00076         if (strcmp(token, stopwords[i]) == 0)
00077         {
00078             return true; // El token es una stopword.
00079         }
00080     }
00081
00082     // Verificación del patrón "docN" donde N es cualquier número.
00083     if (strncmp(token, "doc", 3) == 0 && isdigit(token[3]))
00084     {
00085         return true; // El token comienza con "doc" seguido de un número.
00086     }
00087
00088     return false; // El token no es una stopword.
00089 }
00090
00091 // Tokeniza el texto de entrada, eliminando puntuación y stopwords, e indexa cada palabra.
00092 void tokenize_text(char *text, int doc_id, InvertedIndex **index)
00093 {
00094     char *token;
00095
00096     // Convierte todo el texto a minúsculas.
00097     for (int i = 0; text[i]; i++)
00098         text[i] = tolower(text[i]);
00099
00100     // Reemplaza caracteres de puntuación por espacios.
00101     for (int i = 0; text[i]; i++)
00102         if (ispunct(text[i]))
00103             text[i] = ' ';
00104
00105     // Separa el texto en palabras utilizando el espacio como delimitador.
00106     token = strtok(text, " ");
00107
00108     // Procesa cada token.
00109     while (token != NULL)
00110     {
00111         // Verifica si el token es muy largo.
00112         if (strlen(token) >= MAX_WORD_SIZE)
00113         {
00114             fprintf(stderr, "La palabra es muy larga\n");
00115             exit(EXIT_FAILURE);
00116         }
00117
00118         // Agrega las palabras válidas al índice invertido.
00119         if (!is_stopword(token))
00120             add_document(index, doc_id, token);
00121         token = strtok(NULL, " "); // Avanza al siguiente token.
00122     }
00123 }
00124
00125 // Imprime el índice invertido mostrando cada palabra y los documentos asociados este es solo para ver
      el funcionamiento.
00126 void print_inverted_index(InvertedIndex **index)
00127 {
00128     fprintf(stdout, "\nIndice invertido:\n");
00129     for (int i = 0; i < HASH_TABLE_SIZE; i++)
00130     {
00131         if (index[i] != NULL) // Solo muestra las palabras que existen en este índice.
```

```
00132         {
00133             InvertedIndex *current = index[i];
00134             while (current != NULL)
00135             {
00136                 fprintf(stdout, "Palabra: %s - Documentos: ", current->word);
00137                 Node *doc_node = current->docs_list;
00138                 while (doc_node != NULL)
00139                 {
00140                     fprintf(stdout, "%d ", doc_node->doc_id); // Imprime el ID de documentos
      asociados.
00141                     doc_node = doc_node->next;
00142                 }
00143                 fprintf(stdout, "\n");
00144                 current = current->next;
00145             }
00146         }
00147     }
00148 }
00149
00150 // Función hash para obtener un índice basado en el valor ASCII de los caracteres de una palabra.
00151 unsigned int hash_function(char *word)
00152 {
00153     unsigned int hash = 0;
00154     for (int i = 0; word[i] != '\0'; i++) // Recorre cada carácter en la palabra.
00155         hash = (hash * 31) + word[i];     // Calcula el hash utilizando la codificación ASCII de cada
      letra.
00156
00157     return hash % HASH_TABLE_SIZE; // Retorna el hash limitado por el tamaño máximo de palabras.
00158 } /*Edit por segfault, podria causar error por usar MAX_WORD_SIZE en vez de HASH_TABLE_SIZE*/
00159
00160 // Busca una palabra en el índice invertido y retorna la lista de documentos asociados.
00161 Node *search_word(InvertedIndex **hash_table, char *word)
00162 {
00163     unsigned int index = hash_function(word); // Obtiene el índice de la palabra en la tabla hash.
00164     InvertedIndex *current = hash_table[index];
00165
00166     // Busca la palabra en el índice y retorna la lista de documentos si se encuentra.
00167     while (current != NULL)
00168     {
00169         if (strcmp(current->word, word) == 0)
00170             return current->docs_list;
00171         current = current->next;
00172     }
00173
00174     return NULL; // Retorna NULL si la palabra no se encuentra.
00175 }
00176
00177 void print_search_word(InvertedIndex **index, char *word_to_search)
00178 {
00179     Node *results = search_word(index, word_to_search);
00180     if (!results)
00181     {
00182         fprintf(stderr, "Palabra '%s' no encontrada.\n", word_to_search);
00183         return;
00184     }
00185
00186     // Contador para la cantidad de veces que aparece la palabra en cada documento
00187     fprintf(stdout, "\nLa palabra '%s' se encuentra en los siguientes documentos:\n\n",
      word_to_search);
00188     // contador por documento
00189     int doc_count[MAX_DOCS] = {0};
00190     Node *current = results;
00191     while (current != NULL)
00192     {
00193         doc_count[current->doc_id]++;
00194         current = current->next;
00195     }
00196
00197     for (int i = 0; i < MAX_DOCS; i++)
00198     {
00199         if (doc_count[i] > 0)
00200         {
00201             fprintf(stdout, "Doc%d: %d veces\n", i, doc_count[i]);
00202         }
00203     }
00204 }
00205 // Lee los archivos que se encuentran guardados en el grafo y crea el indice
00206 void build_index(Graph *graph, InvertedIndex **index)
00207 {
00208     // Iterar sobre los documentos ya identificados en el grafo.
00209     for (int i = 0; i < graph->total_docs; i++)
00210     {
00211         FILE *file = fopen(graph->mapping_docs[i].name, "r");
00212         if (file == NULL)
00213         {
00214             fprintf(stderr, "No se pudo abrir el archivo %s\n", graph->mapping_docs[i].name);
00215             continue;
```

```
00216         }
00217
00218         char buffer[1024];
00219         while (fgets(buffer, sizeof(buffer), file))
00220         {
00221             // Procesa cada línea de texto y la pasa al índice.
00222             tokenize_text(buffer, graph->mapping_docs[i].doc_id, index);
00223         }
00224         fclose(file);
00225     }
00226 }
00227
00228 // Libera la memoria del índice invertido y los nodos de documentos asociados.
00229 void release_inverted_index(InvertedIndex **hash_table)
00230 {
00231     if (hash_table == NULL)
00232         return; // Verifica si la tabla hash es nula.
00233
00234     // Recorre cada índice en la tabla hash.
00235     for (int i = 0; i < HASH_TABLE_SIZE; i++)
00236     {
00237         InvertedIndex *current = hash_table[i];
00238
00239         // Libera la memoria de cada palabra y su lista de documentos.
00240         while (current != NULL)
00241         {
00242             Node *doc_node = current->docs_list;
00243             while (doc_node != NULL)
00244             {
00245                 Node *temp_doc_node = doc_node;
00246                 doc_node = doc_node->next;
00247                 free(temp_doc_node); // Libera cada nodo de documento.
00248             }
00249             InvertedIndex *temp = current;
00250             current = current->next;
00251             free(temp); // Libera el nodo de la palabra.
00252         }
00253         hash_table[i] = NULL;
00254     }
00255 }
```

5.11 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.*

5.11.1 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 Carcamo

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

Definition in file main.c.

5.11.2 Function Documentation

5.11.2.1 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.

5.12 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 = 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     for (int i = 0; i < HASH_TABLE_SIZE; i++) // esto lo hice para probar si arreglaba el segmentation
      fault, nose si es necesario
00087         index[i] = NULL;
00088
00089     initialize_graph(&graph);
00090     build_graph(&graph);
00091     calculate_pagerank(&graph, pagerank);
00092     display_pagerank(&graph, pagerank);
00093     show_graph(&graph);
00094     build_index(&graph, index);
00095     print_search_word(index, word_to_search);
00096     release_inverted_index(index);
00097     release_graph(&graph);
00098     generate_eps(&graph, pagerank, "graph.eps");
00099
00100     return EXIT_SUCCESS;
00101 }
```

5.13 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.*

5.13.1 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.

5.13.2 Function Documentation

5.13.2.1 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.

5.14 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 }
```

5.15 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.

5.15.1 Detailed Description

Archivo que contiene las funciones de PageRank.

Date

18-11-2024

Authors

Miguel Loaiza, Diego Sanhueza, Miguel Maripillan y Felipe Carcamo

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

Definition in file pagerank.c.

5.15.2 Function Documentation

5.15.2.1 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.

5.15.2.2 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:\n\n");
for (int i = 0; i < graph->total_docs; i++)
     fprintf(stdout, "Documento (%s): PageRank = %.6f\n", graph->mapping_docs[i].name, pagerank[i]);
```

Definition at line 126 of file pagerank.c.

5.15.2.3 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.

5.16 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 {
00136     fprintf(stdout, "\nValores de PageRank:\n\n");
00137     for (int i = 0; i < graph->total_docs; i++)
00138         fprintf(stdout, "Documento (%s): PageRank = %.6f\n", graph->mapping_docs[i].name,
      pagerank[i]);
00139 }
```

5.17 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.

5.17.1 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.

5.17.2 Function Documentation

5.17.2.1 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 86 of file doc.c.

5.17.2.2 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);
}
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.

5.18 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 {
00049     if (num_docs <= 0 || num_docs >= 100)
00050     {
00051         fprintf(stderr, "El número de archivos web a generar debe ser MAYOR a 0 y MENOR a 100.\n\n");
00052         exit(EXIT_FAILURE);
00053     }
00054
00055     int *links = malloc(num_docs * sizeof(int));
00056
00057     for (int i = 0; i < num_docs; i++)
00058         links[i] = (i + 1) % num_docs + 1;
00059
```

```
00060     for (int i = 1; i <= num_docs; i++)
00061     {
00062         char doc_name[MAX_NAME_DOC];
00063         snprintf(doc_name, sizeof(doc_name), "doc%d.txt", i);
00064 
00065         FILE *doc = fopen(doc_name, "w");
00066         if (doc == NULL)
00067         {
00068             fprintf(stderr, "Error al abrir el archivo web.\n");
00069             free(links);
00070             exit(EXIT_FAILURE);
00071         }
00072         generate_random_text(doc, doc_name, num_docs, num_characters, i, links);
00073         fclose(doc);
00074     }
00075 }
00076 
00086 void generate_random_text(FILE *doc, const char *doc_name, int num_docs, int num_characters, int
      current_doc, int *links)
00087 {
00100     for (int i = 0; i < num_characters; i++)
00101     {
00102         char letter = 'A' + rand() % 26;
00103         fprintf(doc, "%c", letter);
00104         if (i < num_characters - 1)
00105             fprintf(doc, " ");
00106     }
00107 
00121     fprintf(doc, "\nlink: doc%d", links[current_doc - 1]);
00122 
00123     int extra_links = rand() % num_docs;
00124     for (int i = 0; i < extra_links; i++)
00125     {
00126         int link_doc = rand() % num_docs + 1;
00127         if (link_doc != current_doc && link_doc != links[current_doc - 1])
00128             fprintf(doc, "\nlink: doc%d", link_doc);
00129     }
00130 
00131     fprintf(stdout, "ARCHIVO '%s' generado con ÉXITO con %d letras y enlaces.\n", doc_name,
      num_characters);
00132 }
```

Index