Filedistance

Damiano Cacchiarelli 105101

Sistemi Operativi – Laboratorio

Corso di Laurea in Informatica

Anno 2019/2020

Introduzione

Filedistance - Damiano Cacchiarelli 105101

Il programma *Filedistance* permette di calcolare la distanza di edit tra uno o più file. Le funzionalità implementate sono le seguenti:

- **Calcolare la distanza di edit tra due file**: tramiate il comando *distance [file1] [file2]* verrà prodotto in output la distanza tra i due file e il relativo tempo di computazione;
- **Generare un file di output contenente le operazioni di modifica**: tramite il comando *distance [file1] [file2] [output_file]* verrà salvata nel file di output la sequenza di operazioni minime per trasformare il file *file1* nel file *file2*;
- **Applicare le modifiche ad un file**: tramite il comando *apply [input_file] [filem] [output_file]* verrà applicato al file di input le modifiche descritte in *filem* e il risultato verrà salvato nel file di output;
- **Ricercare file con distanza minima**: tramite il comando *search [input_file] [directory]* è possibile ricercare nella directory e nelle sotto-directory tutti i file con distanza minima dal file di input. Verranno quindi restituite in output la distanza e la directory di ogni file che rispetta tale condizione;
- **Ricercare file con distanza minore di *limit***: tramite il comando *searchall [input_file] [directory] [limit]* è possibile ricercare nella directory e nelle sotto-directory tutti i file con distanza di edit minore o uguale a *limit*. Verranno quindi restituite in output la distanza e la directory di ogni file che rispetta tale condizione;

Nel caso in cui *Filedistance* verrà eseguito senza argomenti in input, restituirà in output i comandi che è possibile eseguire:

```
List of commands:
1) distance <file1> <file2>
2) distance <file1> <file2> <outputFile>
3) apply <inputFile> <filem> <outputFile>
4) search <inputFile> <directory>
5) searchall <inputFile> <directory> <limit>
```

Struttura e Implementazione

Filedistance - Damiano Cacchiarelli 105101

Il codice è stato strutturato in otto file *headers* e rispettivi file *sources*:

- **distance.h / distance.c :** *distance.h* definisce due funzioni: *calculate_distance* e *generate_operation*. La prima permette di calcolare la distanza di edit tra i due file passati come argomenti; l'implementazione presente in *distance.c* utilizza l'algoritmo ottimizzato in memoria per calcolare e restituire in output tale valore. La seconda funzione invece permette di generare uno *Stack* contenente le operazioni di modifica da apportare al primo file per poterlo trasformare nel secondo; l'implementazione presente in *distance.c* utilizza l'algoritmo per il calcolo della distanza di Levenshtein che prevede l'utilizzo di una matrice e l'algoritmo in backtrack per risalire tale matrice e generare lo *Stack* di operazioni. Inoltre in *distance.c* sono presenti procedure che permetto di allocare, elaborare e deallocare la matrice relativa al calcolo della distanza di edit.

- **filebuffer.h / filebuffer.c :** filebuffer.h definisce la struttura *File_Buffer* che permette di memorizzare alcune caratteristiche di un file. Inoltre sono presenti anche due funzioni, *get_File_Buffer* e *free_File_Buffer*, che permettono rispettivamente di allocare e inizializzare una struttura *File_Buffer* e di deallocare l'area di memoria occupata di tale struttura. In *filebuffer.c* sono implementate, oltre a queste due procedure, una funzione che permette di calcolare la dimensione, espressa in numero di byte, di un file (*get_dim_file*) e una funzione che permette di leggere i byte di un file (*read_from_file*).

- **filechange.h / filechange.c :** *filechange.h* definisce due funzioni: *edit_file* permette di modificare un file seguendo una determinata sequenza di operazioni e salvare il file modificato nel file di output definito; *save_operation*, invece, permette di salvare le operazioni di modifica contenute in uno *Stack* nel file di output definito. In *filechange.c* sono implementate inoltre diverse procedure di supporto per le vari operazioni da eseguire sul file tra cui *create_stack_operation* e *apply_to_file* che permettono, rispettivamente, di creare uno *Stack* di operazioni partendo da un file e di applicare tali operazioni a un vettore di byte.

- **filedistance.h / filedistance.c :** *filedistance.h* definisce le cinque procedure che descrivono le cinque operazioni principali del programma *Filedistance*. La funzione *distace* permette di calcolare la distanza di edit tra due file. La funzione *out_distance*, invece salva nel file di output le operazioni da eseguire per trasformare un file in un altro. *apply_change* permette di modificare un file. Infine *search* e *search_all* permettono di visualizzare i file con distanza di edit minore o uguale ad un determinato limite.

- **search.h / search.c :** search.h definisce la funzione *search_files_and_print* che permette di visualizzare una lista dei file contenuti in un directory che hanno distanza di edit minore di un determinato limite. In *search.c* tale procedura è stata implementata in modo tale che se il limite viene impostato minore di zero verranno visualizzati i file con distanza minima. Inoltre viene fatto uso della struttura *Sorted_list* per poter memorizzare tutti i file analizzati e per poterli ordinare in modo crescente.

- **sortedlist.h /sortedlist.c :** *sortedlist.h* definisce due strutture: *Sorted_List* e *Element_List.* Una *Sorted_List* è una struttura che rappresenta una lista ordinata in modo crescente secondo la distanza di edit degli elementi; è costituita da un puntatore al primo elemento della lista. Un *Element_List*, invece, rappresenta un elemento della lista ed è costituito dai dati di un file e da un puntatore all'elemento successivo. Inoltre vengono definite procedute che permettono di allocare e deallocare tali strutture dati. In *sortedlist.c* viene implementata anche la funzione *extract_first* che permette di rimuovere il primo elemento della lista.

- **stack.h / stack.c :** *stack.h* definisce tre strutture dati : *Stack*, *Stack_Node*, *Element_Stack*. Lo *Stack* è una struttura che rappresenta una coda con algoritmo LIFO; è costutuito da una sequenza di *Stack_Node* i quali memorizzano i dati tramite un *Element_Stack*. Viene inoltre definita l'enum *Opr* costituita da tre elementi SET, ADD, DEL che rappresentano rispettivamente l'operazione di sostituzione di un byte con un altro, l'operazione di inserite un byte e l'operazione di eliminare un byte. Vengono inoltre definite altre funzioni che rappresentano le operazioni di *pop*, *push* e *peek* per gli elementi della coda. Vengono inoltre definite le procedure per allocare e deallocare le varie strutture.

- **tools.h / tools.c :** tools.h definisce alcune procedure di supporto per le altre. Tra queste abbiamo: *isNull*, che permette di determinare se un puntatore è NULL; *start_timer*, *reset_timer* e *print_timer*, permettono di avviare, resettare e visualizzare il tempo del timer; *open_file* permette di aprire un file; *free_buffer* permette di deallocare due aree di memoria; Infine *get_name_operation* permette di ottenere la stringa corrispondente all'enum *Opr*.

- **main.c :** qui vengono implementate la procedura per riconoscere il comando inserito come argomento del programma e quella per visualizzare in output tutti i comandi disponibili.

Struct

Filedistance - Damiano Cacchiarelli 105101

Ho implementato tre strutture dati: *Stack*, *Sorted List*, *File Buffer*.

- ***Stack***

Uno Stack è la struttura dati che rappresenta una pila con algoritmo LIFO. Questa struttura viene utilizzata per memorizzare le operazioni di modifica. È costituito da tre elementi: *Stack*, *Stack_Node* e *Element_Stack*.

```
typedef struct Stack{
    Node *firstNode;
}Stack;
```

Struct Stack o ***Stack*** rappesenta la pila ed è costituito da *firstNode*, ovvero il puntatore al primo nodo dello stack.

Struct Stack_Node o ***Node*** rappresenta un nodo della pila; è costituito da un puntatore al nodo successivo (*next*) e da un puntatore a un *Element_Stack* che conterrà i dati dell'operazione.

```
typedef struct Stack_Node{
    Element_Stack *data;
    struct Stack_Node *next;
}Node;
```

```
typedef struct Element_Stack{
    char byte;
    unsigned int position;
    Opr operation;
}Element_Stack;
```

Struct Element_Stack o ***Element_Stack*** conterrà i dati relativi all'operazione; è infatti costituito da *position*, che rappresenta la posizione in cui inserire il byte, *byte*, il byte da inserire o da sostituire a uno già esistente e infine *operation* che rappresenta l'operazione da eseguire.

- ***Sorted List***

Una Sorted List è la struttura dati che rappresenta una lista ordinata in ordine crescente, secondo la distanza di edit dei vari elementi. Questa struttura viene utilizzata per memorizzare le distanze di Levenshtein dei file contenuti in una specifica directory. È costituita da due elementi: *Sorted_List* e *Element_List*.

```c
typedef struct Sorted_List Sorted_List;
struct Sorted_List {
    Element_List *firstElement;
};
```

Struct Sorted_List o ***Sorted_List*** rappresenta una lista ordinata in modo crescente; è costituita da un puntatore al primo elemento della lista (*firstElement*).

Struct Element_List o ***Element_List*** rappresenta un elemento della lista; è costituito da un puntatore (*next*) all'elemento successivo e da tre campi che rappresentano: *filename*, il nome del file; *path*, il path assoluto del file; *editDistance*, la distanza di edit.

```c
typedef struct Element_List Element_List;
struct Element_List {
    char *fileName;
    char *path;
    int editDistance;
    Element_List *next;
};
```

- ***File Buffer***

La struttura dati ***struct File_Buffer*** o ***File*** permette di raccogliere e memorizzare alcune informazioni e caratteristiche di un file: *nameFile* rappresenta il nome del file; *dimFile* rappresenta la dimensione del file espressa in numero di byte; *bufferFile* invece è un puntatore a un vettore contenente i byte letti dal file.

```c
typedef struct File_Buffer {
    char *nameFile;
    unsigned int dimFile;
    char *bufferFile;
} File;
```