UNICAM
LERERI CAROLETTA
ALESSANDRA
MATRICOLA : 105532
LABORATORIO DI SISTEMI OPERATIVI
25 settembre 2020

RELAZIONE PROGETTO FILEDISTANCE

INDICE:

DESCRIZIONE DEL PROGETTO

La distanza di edit è una funzione che consente di verificare quanto due stringhe (o sequenze di byte) siano lontane una dall'altra. Questa distanza viene calcolata sulla base del numero di operazioni necessarie a trasformare una data sequenza di byte nell'altra . Le operazioni sono:

- Aggiungere un byte;
- Eliminare un byte;
- Modificare un byte.

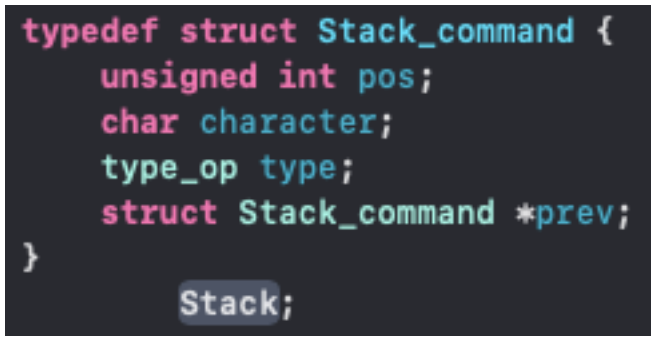
Questo software permette di:

1. Calcolare la distanza di edit tra due file.
2. Scrivere su un file le operazioni da eseguire per trasformare un determinato file in un altro.
3. Prendere in input un file contenete le modifiche, un file contenente una stringa iniziale e un file dove andare a scrivere la stringa dopo averci applicato le modifiche.
4. Cercare in una directory (e nelle relative sottodirectory) tutti i file che hanno distanza minima da un determinato file
5. Cercare in una directory (e nelle relative sottodirectory) tutti i file che hanno distanza inferiore ad un limite da un determinato file

STRUTTURE DATI UTILIZZATE

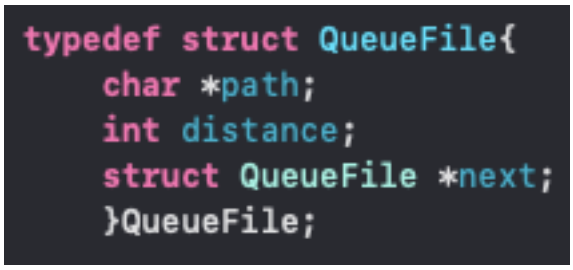
Le strutture dati utilizzate nella realizzazione del software sono principalmente 2:

StackCommand



È una struttura dati utilizzata per memorizzare le operazioni di edit.
All'interno contiene un intero senza segno che rappresenta la posizione, un carattere, un enumerabile di tipo type_op che rappresenta il tipo di operazione (può assumere solo tre valori : ADD, DEL, SET) e un puntatore , prev, che fa riferimento al nodo precedente dello stack.
Essendo appunto uno stack, segue la politica LIFO(Last In First Out).
L'utilizzo di questa struttura facilita la scrittura ordinata delle operazioni dallo stack al file che dovrà contenere le modifiche.

PriorityQueueFile



È una struttura dati utilizzata per memorizzare file.
All'interno contiene un puntatori a caratteri che identifica il path assoluto del file, un interno 'distance' che rappresenta la distanza di edit e un puntatore al nodo successivo.
Essendo una coda a priorità, dove la priorità più alta è determinata da un più basso numero nella distanza, i nodi all'interno non vengono inseriti in ordine temporale ma rispettano appunto questo ordinamento: nella parte alta della coda si trovano i nodi con distanza di più bassa mentre nella parte bassa della coda si trovano i nodi con distanza di edit più grande.
L'utilizzo di questa struttura facilita poi la scelta di quali path di file vanno stampati (ad esempio con il serchAll, quando si incontra il primo nodo con distanza più alta del limite, si è certi che anche tutti quelli successivi abbiano la distanza uguale o superiore.)

LIBRERIE STANDARD UTILIZZATE

Le librerie standard utilizzate all'interno del codice sono le seguenti:

- Stdio.h : per la gestione dell'Input/Output.
- Stdlib.h : per la gestione della memoria.
- String.h : per la manipolazione delle stringhe.
- Dirent.h : per la gestione delle directory.
- Time.h : per la gestione del tempo.

LIBRERIE PERSONALIZZATE

Gli headers file creati hanno i seguenti compiti:

- **DISTANCE.H** per la gestione delle funzionalità del programma.

  - **void** getDistance(char *firstFile, char *secondFile);
    Questa funzione fa partire il timer e richiama le funzioni necessarie per calcolare la distanza di Levenshtein tra i due file i cui path vengono presi in input. Il timer viene fermato non appena la funzione pe ril calcolo della distanza di edit restituisce un valore. Viene stampato a video la distanza e il tempo necessario per calcolarla.

  - **void** getCommand(char *firstFile, char *secondFile, char *fOutput);
    Questa funzione prende in input i path di tre file: sui primi due file viene calcolata la distanza di edit e nel terzo vengono scritte le modifche necessarie a trasformare il secondo file nel primo.

  - **void** applyMod(char *fileInput, char *fileModify, char *fileOutput); Questa funzione prende in input tre file :
    - il primo è il file di partenza
    - il secondo è il file che continene le modifiche da applicare alla stringa contenuta nel file di partenza
    - il terzo è il file dove viene salvata la stringa modificata.

  - **void** search(char *fileInput, char *directory);
    Questa funzione prende in input un file e una directory e utilizza lo standardOutput per produrre i file contenuti in dir (e nelle sue sottodirectory) che hanno minima distanza da fileInput.

  - **void** searchAll(char *fileInput, char *directory, int limit);
    Questa funzione prende in input un file, una directory e un intero e visualizza tutti i file presenti nella directory e nelle sue sottodirectory che hanno distanza di edit uguale o inferiore al limite.

- **LEVENSHTEINDISTANCE.H** implementazione dell'algoritmo di Levenshtein e calcolo delle operazioni di edit.

  - **int** calculateLevenshtein(char *firstString, char *secondString); Questa funzione prende in input due stringhe e calcola la distanza di edit tra le due. Viene utilizzato l'algoritmo di Levenshtein ottimizzato: Invece di utilizzare una matrice m*n (dove m-1 è la lunghezza della prima stringa e n-1 è la lunghezza della seconda stringa, viene utilizzata una matrice 2*n.

  - **stackCommand** *getOperations (char *firstString, char *secondString); La funzione restituisce uno Stack contenente le operazioni da eseguire per trasformare la seconda Stringa nella prima.

  - **void** recursiveLevensthein(char *fileInput, char *path, unsigned int distanceLimit);
    Funzione per fare una scansione ricorsiva della directory "path" e visualizzare a console i file che hanno distanza di edit rispetto a "fInput" inferiore a "limit".

- **STACKCOMMAND.H** per la memorizzazione delle operazioni di edit in una struttura dati dinamica.

  - **void** pushCommand (stackCommand **root, unsigned int pos, char character, type_op type);
    Funzione per inserire un nodo all'interno dello stack. Prende in input un puntatore a puntatore dello Stack(la root), un intero senza segno(indica la posizione in cui "modificare" il carattere), un carattere(il carattere da aggiungere o da modificare, se la modifica è DEL, il carattere passato è '\0') e una variabile ti tipo type_op(che definisce il tipo di operazione da fare: DELL, ADD, SET).

  - **void** popCommand(StackCommand **root);
    Funzione per rimuovere un nodo dallo Stak. Prende in input un puntatore a puntatore allo stack e, se esiste almeno un nodo, dealloca la memoria del nodo in testa, quando lo stack è vuoto dealloca la memoria del puntatore stesso allo stack.

- **char** *getType(StackCommand *node);
  Preso in input un nodo, restituisce il tipo dell'operazione che contiene.
- **int** checkEmptyCommand(StackCommand *root);
  Verifica se lo stack è vuoto o ci sono ancora elementi. Il valore di ritorno della funzione è 0 finche ci sono ancora elementi.

- **IOFILE.H** per la gestione dell'input e dell'output su file.

  - **void** stringToFile(char *fileOutput, char *string);
    Funzione utilizzata per convertire una sequenza di caratteri in un file.

  - **char** *fileToString(char *fileInput);
    Funzione per convertire qualsiasi tipo di file in una sequenza di caratter

  - **void** writeModFile(stackCommand *stack, FILE *fileOutput);
    Funzione che, preso uno Stack e un file (binario), scrive le operazioni dello stack con corrispettiva posizione e carattere (a meno che non si tratti della DEL, in cui non è presente in carattere) sul file in forma binaria.StackCommand *readBinFile(char *fModify);

  - **void** modifyFile(char *fInput, char *fModify, char *fOutput);
    Funzione che prende n input un file iniziale, un file in cui sono scritte le modifiche da apportare e un file in cui scrivere il contenuto del file iniziale modificato con le azioni scritte nel secondo file. Il file in cui sono contenute le azioni è un file binario. Questa funzione si serve di un altro metodo, chiamato ReadModFile che legge il contenuto del file in cui sono scritte le modifiche e ritorna uno Stack. Dallo Stack, la funzione applyModToFile applica le modifiche al file in input e genera il file di output.

- **PRIORITYQUEUEFILE.H** per memorizzazione delle distanze di edit di un gruppo di file in una struttura dati dinamica.

  - **int** checkEmptyQueue(queueFile *root);
    Funzione per controllare se la coda sia vuota. La funzione restituisce 0 finche ci sono elementi all'interno

- **void** enqueueFile(queueFile **root, char *path, unsigned int distance);
  Funzione per aggiungere un elemento alla coda con priorità. Essendo una coda con priorità, l'elemento aggiunto non viene aggiunto in testa ma in una determinata posizione. La sua priorità nella coda è determinata dalla distanza. Più la distanza è piccola, più sarà 'in alto' nella coda
  Funzione per aggiungere un elemento alla pila ordinati in modo crescente secondo la distanza memorizzata nel nodo.

- **void** dequeueFile(queueFile **root);
  Questa funzione rimuove il nodo della coda con maggiore priorità: cioè il nodo che sta più 'in alto '.

• **TIME.H** calcolo del tempo impiegato ad eseguire una determinata funzione.

- **void** start_time(**void**);
  Funzione per avviare il timer;

- **double** end_time(**void**);
  Funzione per stoppare il timer e ritornare il valore del timer

FUNZIONI DISPONIBILI ALL'UTENTE

Per poter eseguire le funzionalità descritte sopra il programma potrà essere eseguito con le seguenti opzioni:
Calcolo distanza tra file:

```
filedistance distance <file1> <file2>
filedistance distance <file1> <file2> <output>
```

Nel primo caso, dove vengono passati come argomenti i due file da confrontare, viene calcolata la distanza dal file e generato un output della seguente forma:

EDIT DISTANCE: distanza
TIME: tempo di calcolo

Nel secondo caso, invece, verranno salvate nel file *output* la sequenza delle istruzioni necessarie per trasformare il primo file nel secondo.

Applicazione delle modifiche:

```
filedistance apply <inputfile> <fileMod> <outputfile>
```

Applica a inputfile le modifiche contenute nel file fileMod e salva il risultato nel file outputfile .

Ricerca di file:

```
filedistance search <inputfile> <dir>
```

Restituisce in output i file contenuti in *dir* (e nelle sue sottodirectory) che hanno minima distanza da *inputfile*. Il path assoluto di ogni file viene presentato in una riga dello *standard output.*

```
filedistance searchall <inputfile> <dir> <limit>
```

Vengono restituiti in output tutti i file contenuti in dir (e nelle sue sottodirectory) che hanno una distanza da *inputfile* minore o uguale di *limit* (che è un intero). I file vengono presentati nello standard output in ordine crescente secondo il seguente formato:

distanza pathassolutofile