Relazione architettura client-server per computazione BWT

Mattia Carlino

5ft-i, anno scolastico 2024/2025, ITIS Archimede

Abstract

L'esercizio svolto, descritto in questa relazione, consisteva nel creare un sistema di comunicazione client-server, in cui il client invia una stringa e il server, calcola la Burrow Wheeler Transform della stringa e la rinvia al client. L'esercizio è stato implementato in python con successo, tramite l'uso della libreria "socket" per la gestione della comunicazione.

Introduzione

Si è partiti dalla creazione del sistema che permettesse lo scambio di dati tra i due processi, Server e Client. Come visto quest'anno in TPSI, è stata utilizzata l'architettura client-server, che prevede un processo (client) che acquisisce i dati dall'utente, e invia le richieste(requests), e un processo (server) che risponde, con dati o elaborazioni, a queste richieste. Questa architettura di rete, come qualsiasi altra per creare il canale logico,che identifichi la connessione tra i due host, usa l'interfaccia costituita da: Indirizzo IP e Process port, che prende il nome di Socket. Questi due argomenti affrontati in TPSI, quest'anno hanno reso possibile la comunicazione tra processo client e server. Dopo aver creato lo scheletro per permettere lo scambio dati, sul processo server è stato implementato l'algoritmo per la trasformata di Burrow Wheeler. Algoritmo di compressione delle stringhe, usato in applicazioni come la compressione dati o in bioinformatica per la compressione delle sequenze proteiche o genetiche.

Funzionamento della Trasformata da Burrow Wheeler

le operazioni per calcolare la trasformata di Burrow Wheeler sono scomponibili in 3 passaggi fondamentali:

- **1. Calcolo delle permutazioni della stringa**
- **2. Ordinamento permutazioni in ordine decrescente**
- **3. Unione delle ultime lettere dell'elenco delle permutazioni ordinate**

1.

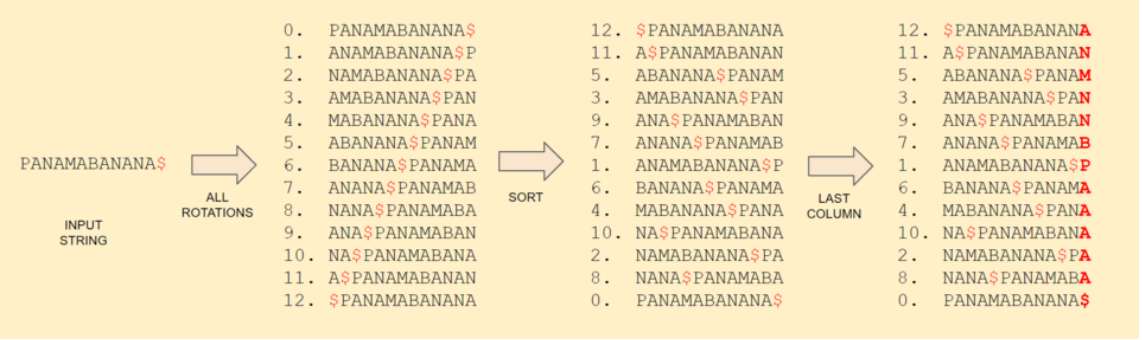
Data la stringa su cui si vuole calcolare la Burrow Wheeler, il primo step è la scelta del numero di caratteri speciali che verranno utilizzati per la codifica della stringa, per semplicità in questo esercizio è stato utilizzato un singolo carattere speciale: "$". Scelto il numero e il carattere speciale si inizia con il calcolo della permutazioni sulla stringa, posizionando il carattere speciale alla fine della stringa, ogni permutazione corrisponderà a un movimento del carattere speciale verso la direzione scelta, che sia destra o sinistra, in questo per il calcolo della trasformata di Burrow Wheeler è stato utilizzato un singolo carattere, appunto "$" e il movimento di esso avveniva verso sinistra.

2.

Una volta calcolate tutte le permutazioni, secondo i parametri scelti, devono essere ordinate secondo l'ordine alfabetico decrescente, come riferimento invece per ordinare le stringhe con carattere speciale all'inizio,è usata la tabella ASCII, in questo caso utilizzando il "$" come carattere speciale, le permutazioni che iniziavano con "$" erano le prime nell'ordinamento.

3.

Infine dall'elenco delle permutazioni ordinate viene presa l'ultima lettera per ogni permutazione, che va poi unita alle lettere di tutte le altre permutazioni.



Architettura dell'implementazione

è stata scelta per implementare il sistema appena descritto, il linguaggio di scripting python i due processi: Client e Server, sono stati divisi su due file distinti denominati appunto Client.py e Server.py, la scelta alternativa alla divisione del codice in questo modo poteva essere l'utilizzo di un terzo file, che raccoglieva il codice di tutte le funzioni chiamate e poi nei due file che ospitano i processi client e server, importare, come una libreria, il file con le funzioni e chiamare quelle necessarie per ogni processo. La divisione scelta per il codice su due file è stata fatta per semplicità di costruzione, avendo meno file si eliminano i possibili problemi legati al importare il file con le funzioni e per permettere una visualizzazione più immediata del codice, dato che le funzioni comunque mantengono tutte la regola fondamentale di essere più generali possibili.

Client.py

Il codice implementato in questo file è suddiviso su due funzioni in cui vengono svolte le due operazioni fondamentali che il client effettua:

- **gestione comunicazione con il server** *(funzione avvia client);*
- **convalida della stringa inserita dall'utente** *(funzione input validation messaggio).*

Funzione avvia client()

La funzione avvia client sfrutta per stabilire la connessione e scambiare informazioni con il server la libreria messa a disposizione da python: Socket, che è stata importata e utilizzata per creare la socket del client, socket che va a comunicare con la socket del server che ha come codici univoci di identificazione la coppia: indirizzo IP dell'host e numero di porta del processo. Queste due informazioni sono salvate rispettivamente nelle variabili: HOST e PORT.

```
1   import socket as s
2
3   HOST = "127.0.0.1"
4   PORT = 12345
5
```

Passando alla funzione vera e propria il suo funzionamento può essere suddiviso, a sua volta in tre blocchi:

- **1. Creazione socket e stabilimento della connessione con il server**
- **2. inserimento messaggio e verifica**
- **3. Invio e ricezione**

1.

In questa fase è utilizzato il costrutto **With**, un costrutto di python che permette di gestire in maniera migliore e più efficiente lo stream di dati con un'altra entità, che sia scrittura e lettura da file o scambio di dati tra socket. Dunque con il with è stata creata la socket client, l'oggetto rappresentante la socket del client, su cui sono state eseguite le operazioni di scambio dati tra socket e gestione della connessione, tramite il metodo delle socket .connect() è stata stabilita la connessione al server, alla funzione è passata la tupla con le informazioni della socket server. Per la gestione dei possibili errori derivanti dal non trovare il processo server, o dalla mancata connessione ad esso è stato utilizzato il costrutto **try/except** che cattura gli errori di tipo: **Connection Refused Error,** e stampa la possibile problematica. Nella creazione del socket client, non è necessario, almeno sul client fare la bind, ovvero l'operazione di creazione del socket sul host client, con l'assegnazione dell' indirizzo IP del client e del numero del processo client, questo perché in questo caso il client ha gestione dinamica delle porte, e dunque la sola identificazione dell'oggetto socket client sul nostro processo è sufficiente per permettere la comunicazione dato che l'assegnazione della porta del processo e dell'indirizzo IP sarà gestito automaticamente dal sistema operativo.

```
14
15  def avvia_client():
16
17      try:
18
19          with s.socket(s.AF_INET,s.SOCK_STREAM) as socket_client:
20              socket_client.connect((HOST,PORT))
```

La prima parte con il costrutto with e l'inizio del try/except

l'ultima parte del try/except con la cattura dell'errore e la stampa

```
38
39      except ConnectionRefusedError:
40          print("ci sono stati problemi di connesione al server")
41
```

2.
Una volta stabilita la connessione al server, è stato implementato un piccolo menù da terminale per permettere all'utente l'inserimento della stringa che desidera codificare con la BWT, realizzato con un ciclo **While True**, da cui l'utente può uscire inserendo la parola chiave **"exit"**. Se non viene inserita exit viene verificata che la stringa inserita sia composta solo da lettere,tramite la seconda funzione implementata dal processo client, ovvero **input validation messaggio(),** e se restituisce true, si passa alla terza fase con l'invio e la ricezione della risposta

```
while True:
    print("COMUNICAZIONE CON IL SERVER IN BWT")
    print("INSERISCI 'exit' per uscire dal programma")
    messaggio = input("Inserisci la stringa che vuoi convertire" + "\n")

    if(messaggio.lower() == 'exit'):
        print("sei uscito dalla comunicazione")
        break

    if(input_validation_messaggio(messaggio)):
```

3.
La terza fase comprende la gestione delle invio della stringa, inserita da parte dell'utente, e della ricezione del risultato, ovvero la stringa codificata in BWT da parte del server tramite la connessione stabilita dall'oggetto **socket client**. Il **formato dei messaggi** che è stato utilizzato per lo scambio delle stringhe tra client e server è : **"lunghezza stringa": "stringa"**, dunque un'unica stringa in cui la parte prima dei ":" è la lunghezza della stringa contenuta dopo i due punti. Dunque dopo la verifica viene calcolata la lunghezza della stringa e creato il messaggio da inviare poi al server, l'invio avviene tramite il metodo della socket client **.sendall()**, che prende in input il messaggio codificato tramite la funzione, nativa delle stringhe in python **.encode()**, a cui a sua volta è passato il tipo di codifica desiderato per il messaggio, in questo caso è stato utilizzato '**utf-8'**. Dopo l'invio è stato implementato il codice per gestire la ricezione della risposta da parte del server, grazie al metodo di socket client **.recv()**, un numero che rappresenta la lunghezza massima in byte del messaggio ricevuto, se la supera il messaggio viene scartato, successivamente si procede alla decodifica della stringa tramite il metodo delle stringhe **.decode()**, contrario di .encode(), a cui viene sempre passata la codifica della stringa, in questo caso sempre 'utf-8', e alla scomposizione della stringa tramite il metodo **.split()** delle stringhe, che restituisce un array con la stringa prima dei due punti e dopo i due punti. L'array ottenuto viene poi stampato sul terminale

```
lunghezza_stringa = len(messaggio)
print(f"il messaggio inviato dal client  e': {messaggio} ha lunghezza: {lunghezza_stringa}")
messaggio = str(lunghezza_stringa) + ":" + messaggio
socket_client.sendall(messaggio.encode('utf-8'))
messaggio_server = socket_client.recv(1024)
messaggio_server_diviso = messaggio_server.decode().split(":")
print(f"la stringa, dopo la conversione in BWT è {messaggio_server_diviso[1]} e ha lunghezza: {messaggio_server_diviso[0]}")
```

Funzione input validation messaggio()

Questa funzione è utilizzata per la verifica della stringa inserita dall'utente, controlla che siano tutte lettere minuscole e restituisce true, se tutte le lettere della stringa sono minuscole altrimenti false; è stata implementata semplicemente con l'uso di un ciclo for e di un if per la verifica dei singoli caratteri della stringa. Chiamata nella funzione avvia client() per verificare che non vengano inviate all'algoritmo di conversione in BWT presente sul processo client, numeri o caratteri speciali che potrebbero inficiare, non tanto l'esecuzione, quanto il risultato del conversione, non facendone comprendere il risultato.

```
def input_validation_messaggio(messaggio):

    messaggio = messaggio.lower()
    for i in range(0,len(messaggio)):
        if(ord(messaggio[i]) < 97 or ord(messaggio[i]) > 123):
            return False

    return True
```

Server.py

il codice implementato lato server è, come per il client, suddiviso in due funzioni, per le due operazioni fondamentali che in questa architettura il server svolge, come prima la gestione della comunicazione con il client *(funzione avvia server),* e il calcolo della Burrow Wheeler Transform partendo da una stringa *(funzione str_to_BWT).*

Funzione avvia server()

Così come per il processo client anche il processo server utilizza la libreria Socket fornita da python, sono utilizzate anche le stesse due variabili HOST e PORT per identificare la coppia indirizzo IP e porta del processo per la creazione della socket. E anche i valori di HOST e PORT sono identici a quelli del processo client, dato che il client non stabiliva in maniera rigida la propria socket tramite la bind ma usava HOST e PORT per connettersi direttamente alla socket del processo server.

è specificato questo appunto perché la funzione avvia server andrà ad utilizzare la libreria Socket e le due variabili HOST e PORT per la creazione e la gestione della socket server. Come la funzione avvia client anche la funzione avvia server può essere scomposta in 3 parti principali:

- **1. Creazione socket**
- **2. Stabilimento della connessione con il client**
- **3. Ricezione del messaggio e calcolo BWT**
- **4 .Invio del messaggio**

1.

La prima fase è identica nei comandi e nella logica alla prima fase della funzione avvia client, anche qua è stato sfruttato il costrutto **With,** ma a differenza di prima sul processo server è necessario fare la bind, con il metodo dell'oggetto socket server **.bind()**, dato che il server potendo gestire più processi connessi a diversi client, deve identificare in maniera rigida e stabile il processo per una specifica applicazione, per permettere ai client la connessione sulla socket corrispondente.
Successivamente sempre per gestire la connessione dei client al processo è stato utilizzato il metodo **.listen(),** a cui viene passato il numero massimo che il server accetta per la connessione al processo, in questo caso è stato passato 1 come numero di client connessi.

```
def avvia_server():
    with s.socket(s.AF_INET, s.SOCK_STREAM) as socket_server:
        socket_server.bind((HOST, PORT))
        socket_server.listen(1)
```

2.

La seconda fase gestisce la connessione sul server, da consegna la connessione con il client doveva essere persistente, finché il client, non avesse interrotto la connessione. Per intercettare eventuali errori derivanti dalla gestione o connessione del client, è stata racchiusa tutta la seconda parte del codice in un costrutto **try/except,** che intercetta un generico errore **Exception,** stampando il tipo di errore e terminando l'esecuzione del server. Sono stati implementati poi due cicli While True, il primo per l'attesa da parte del server della connessione di un client, attesa che consiste nel metodo della socket server **.accept(),** che quando un client si connette alla socket del server restituisce la tupla con il suo indirizzo IP e la porta del processo; il secondo una volta stabilita la connessione al client, si sfrutta il costrutto **with,** per gestire "**conn",** l'oggetto rappresentante la connessione stabilita, oggetto con verrà effettuato lo scambio dati con il client, e il secondo ciclo al compito di permettere che lo stesso client possa inviare più stringa, mantenendo sempre la stessa connessione.

```
    try:
        while True:
            conn, addr = socket_server.accept()
            print("il client si è connesso al server")
            with conn:
                while True:
```

prima parte con l'inizio del try/except e con i due cicli while True

la fine del try/except con la cattura dell'eccezione

```
        except Exception as e:
            print(e)
```

3.
La terza fase comprende la ricezione del messaggio, l'oggetto conn che condivide i metodi dei socket, con il metodo **.recv()**, gestisce la ricezione del messaggio, stabilendo la lunghezza massima di byte che il messaggio può avere, in questo caso è stato passato 1024, poi avviene la decodifica della stringa, con il metodo **.decode()**, sempre in codifica **'utf-8'** e poi l'estrazione della lunghezza del messaggio e del messaggio, con il metodo **.split(),** è stato inserito anche un controllo con un **if**, in caso che il messaggio ricevuto dal client sia nullo, esce dal ciclo con un **break.** Una volta verificato si passa la stringa inviata dal client e la sua lunghezza alla funzione per il calcolo della Burrow Wheeler, **str to BWT().**

```
messaggio_client = conn.recv(1024)
messaggio_client = messaggio_client.decode('utf-8')
stringa_divisa = messaggio_client.split(":")
print(f"il server ha ricevuto il messaggio: {stringa_divisa[1]} e ha lunghezza: {stringa_divisa[0]}")
if not messaggio_client:
    break
messaggio_BWT = str_to_BWT(stringa_divisa[1],int(stringa_divisa[0]))
```

4.
L'ultima fase consiste nella creazione, partendo dal risultato restituito dalla funzione per il calcolo della Burrow Wheeler, del messaggio da inviare al client, dunque si ripetono le stesse operazioni fatte dal client durante l'invio al server, con il calcolo della lunghezza del risultato e la creazione della stringa nel formato visto prima, e tramite il metodo sendall(), sull'oggetto conn, viene inviata la stringa codificata in BWT, che viene codificata tramite **.encode()**, con la codifica 'utf-8'.

```
messaggio_finale = str(len(messaggio_BWT)) + ":" + messaggio_BWT
conn.sendall(messaggio_finale.encode('utf-8'))
```

Funzione str to BWT()

La funzione per il calcolo della Burrow Wheeler Transformation implementata segue la divisione di passi che è stata elencata nell'introduzione a questa relazione, dunque prima il calcolo delle permutazioni, l'ordinamento di esse in ordine alfabetico e la selezione dell'ultima lettera di ogni permutazione, nell'ordine appena fatto, per unire ogni lettera in una stringa che dà come risultato la stringa iniziale codificata secondo la trasformata di Burrow Wheeler. Questi 3 steps li vediamo nelle 3 parti del codice della funzione qua sotto:

Step 1:

```
def str_to_BWT(messaggio_client,lunghezza_stringa):
    permutazioni_stringa = {
        0: messaggio_client +"$"
    }

    print("calcolo di tutte le permutazioni shiftando verso sinistra: ")
    print(f"permutazione {0}: {permutazioni_stringa[0]}")
    for i in range(0,len(messaggio_client)):
        parte_dopo_dollaro = messaggio_client[slice(i+1)]
        parte_prima_dollaro = messaggio_client[slice(i+1,lunghezza_stringa)]
        permutazioni_stringa[i+1] = parte_prima_dollaro + "$"+ parte_dopo_dollaro
        print(f"permutazione {i+1}: {permutazioni_stringa[i+1]}")
```

in questa parte viene creato il dizionario in cui verranno salvate tutte le permutazioni, chiamato **permutazioni stringa**, il calcolo delle permutazioni avviene tramite un ciclo for, che continua aumenta i per tutta la lunghezza della stringa, la logica dietro il calcolo delle permutazioni in questa funzione è che la stringa viene tagliata in punti diversi ad ogni ciclo, e il punto in cui viene tagliata corrisponde a dove andrà inserito poi il carattere speciale "$", dunque tramite metodo **slice()**, viene selezionato l'intervallo della stringa da tagliare, in funzione di i, e poi viene inserito tra le due parti della stringa chiamate **parte prima dollaro** e **parte dopo dollaro** il carattere speciale.

Step 2:

```
permutazioni_stringa_ordinate = dict(sorted(permutazioni_stringa.items(), key=lambda item: item[1]))
```

Una volta che sono state calcolate tutte le permutazioni le andiamo a ordinare tramite il metodo sorted, che restituisce un oggetto, che noi andiamo inserire dentro il nuovo dizionario **permutazioni stringa ordinate,** il sorted itera sugli elementi del dizionario, indicati con **permutazioni stringa.items()** e grazie a una funzione lambda riesce a riordinare in ordine decrescente

Step 3:

```
for permutazione in permutazioni_stringa_ordinate.values():
    contatore = 0
    print(f"permutazione {contatore}: {permutazione}")
    stringa_finale = stringa_finale + permutazione[lunghezza_stringa]
    contatore = contatore +1

print("prendiamo ogni lettera finale delle permutazioni, seguendo l'ordinamento appena fatto")
print(f"stringa convertita in BWT e': {stringa_finale}")
return stringa_finale
```

Infine tramite un **for each,** viene preso ogni elemento di **permutazioni stringa ordinate** e presa l'ultima lettera tramite, vengono poi concatenate tutte in **stringa finale** e la funzione poi restituisce la stringa finale.

Risultati e possibili miglioramenti

La consegna data è stata portata a termine con successo, tutte le consegne minime sono state rispettate e il programma funziona sia nella comunicazione tra client e server che nel calcolo della trasformata di Burrow Wheeler. I possibili miglioramenti includono i bonus che erano stati dati come opzioni, dunque il salvataggio e la visualizzazione di statistiche in merito alle operazioni di calcolo della BWT e della gestione della comunicazione con il client, e poi sicuramente l'implementazione dell'algoritmo di BWT inversa per ottenere partendo dalla stringa codificata in BWT la stringa originale da cui è partita la codifica.

References

Sono stati utilizzati per reperire informazioni e tutorial sul codice:

- Documentazione ufficiale di Python: https://docs.python.org/3/
- Pagina wikipedia sulla trasformata di Burrow Wheeler: https://en.wikipedia.org/wiki/Burrows%E2%80%93Wheeler_transform