Relazione progetto Architettura degli elaboratori

Giovanni Palmieri (7006086)

24 giugno 2021

Indice

1 Introduzione

Introduzione

2 Descrizione del ciclo principale

Il compito del ciclo principale *mainLoop* è quello di leggere l'array di input *listInput* e in base ad esso chiamare le funzioni relative ai comandi

2.1 Riconoscimento input

Un comando viene riconosciuto semplicemente dalla prima lettera, dato che non esitono comandi che abbiano la stessa lettera iniziale.

Carattere	Significato
A	ADD{...}
D	DEL{...}
P	PRINT
S	SORT
R	REV

Tabella 1: Riconoscimento comandi

Bisogna però considerare il caso in cui, una lettera relativa ad un comando sia in realtà il parametro di un comando, esempio:

ⒶDD{Ⓐ}

Per distunguere questi due casi ci avvaliamo del fatto che:

1. Il primo carattere che troviamo,relativo ad un comando, non può essere il parametro di un comando.
2. Per ogni comando sappiamo quanto è lungo, dato che non sono ammessi spazi all'interno dei comandi.

Perciò ogni volta che leggiamo un comando, spostiamo il puntatore alla lista di input alla fine del comando.

Comando	Valore da sommare
ADD	6 + 1
DEL	6 + 1
PRINT	5 + 1
SORT	4 + 1
REV	3 + 1

Tabella 2: Tabella valori da sommare per evitare letture incorrette

Il "+1" in Tabella 2 ci permette di risparmiare la validazione di un char, dato che i comandi sono divisi da almeno un "˜" e eventuali spazi.

2.2 Chiamata delle operazioni

Ogni volta che viene riconosciuto un comando, viene eseguita una jump al blocco di codice che gestisce la chiamata al relativo comando.
(*callAdd*, *callDel*, *callSort*, *callRev*, *callPrint*)

Prima di ogni chiamata vengono salvati i registri nello stack, e in caso di comandi con un parametro (*ADD*, *DEL*), il parametro viene messo in **a0**. Poi viene incrementato il puntatore alla stringa di input secondo la Tabella 2

3 Descrizione Add

Descrizione Il comando Add, aggiunge un nodo in coda alla lista, (ossia come ultimo elemento). Dato che la lista è una lista concatenata deve essere allocato dinamicamente lo spazione per l'ultimo elemento.

Implementazione In base all'implementazione della lista concatenata l'operazione per eseguire la Add è diversa. Nel caso del programma non abbiamo un puntatore alla coda, perciò la prima fase della Add è trovare l'ultimo elemento. La ricerca dell'ultimo elemento viene effettuata scorrendo la lista, finchè non si trova un nodo che non ha il punatore al prossimo elemento. A questo punto deve essere allocata un area libera di memoria, e ci viene messo il dato da aggiungere. Dopo di che viene aggiornato il puntatore all'elemento successivo del penutlimo elemento, e i puntatori all'elemento precedente e successivo dell'ultimo elemento. È inoltre necessario distunguere il caso in cui la lista sia vuota e inizializzare il puntatore al primo elmento.

3.1 Funzione getPseudoRandom

Descrizione Come abbiamo detto nella descrizione dell'operazione è necessario trovare un area libera di memoria. Per fare ciò usiamo un generatore di numeri pseudorandomici.

4 Implementazione Sort

Descrizione L'algoritmo utilizzato è un selection sort. L'algoritmo scorre tutta la lista, e ad ogni elemento lo sostituisce con l'elemento più piccolo, tra quelli successivi.

Implementazione L'algoritmo è composto principalmente da due funzione.

1. La funzione che trova il minimo tra i nodi successivi: *getMinAddress*
2. La funzione che per ogni nodo lo scambia con il risultato di getMinAdress: *sort*

Entrambe le funzioni sono implementate ricorsivamente.

4.1 Funzione getMinAddress

La funzione getMinAddress ha due parametri, **a0** è l'indirizzo del nodo corrente, **a1** è l'indirizzo del nodo con il valore minimo. La funzione confronta i valori dei due nodi, se il nodo corrente è minore del minimo allora il nodo minimo diventa il nodo corrente. Dopo di che, controlla se il nodo corrente ha un nodo successivo, in caso non ci sia la funzione ritorna. Mentre se il nodo successivo è presente la funzione si richiama sostituendo al nodo corrente il nodo successivo.

```
private Node getMinAddress(Node current, Node min){
    if(current.getValue() < min.getValue()){
        min = current;
    }
    if(current.getNext() != null){
        min = getMinAddress(current.getNext(), min);
    }
}
```

Listing 1: Codice java algoritmo getMinAddress

4.2 Funzione sort

La funzione sort ha un solo parametro, **a0** l'indirizzo nodo da sostituire con il minimo. La funzione chiama *getMinAddress* sul nodo corrente, e lo scambia con il risultato della funzione. Dopo di che, controlla che ci sia il nodo

successivo e in caso sia presente richiama la funzione *sort* sul prossimo nodo.

```
private void sort(Node current){
    Node min = getMinAddress(current, current);
    swap(current, min);
    if(current.getNext() != null){
        sort(current.getNext());
    }
}
```

Listing 2: Codice java algoritmo sort

4.3 Criteri di ordinamento

Descrizione Il programma non ordina i nodi in base al valore ASCII, ma ha un ordinamento specifico:

- Lettere Maiuscole ASCII(da 65 a 90 compresi)
- Lettere Minuscole ASCII(da 97 a 122 compresi)
- Numeri ASCII(da 48 a 57 compresi)
- Tutti gli altri simboli

All'interno dei gruppi gli elementi sono ordinati in base al loro valore ASCII.

Implementazione Per fare questo ordinamento associamo ad ogni valore ASCII un valore di ordinamento, di modo che gli elementi possano essere ordinati in base al loro valore di ordinamento. Abbiamo perciò bisogno di una funzione che associa un numero ad ogni carattere di modo che:
Lettere Maiuscole > Letter Minuscole > Numeri > Resto.
Per fare questa funzione quello che facciamo è sommare un certo valore in base al gruppo di apparentenza di un carattere.

Caratteri	Valore ASCII	Valore da sommare	Valore restituito
A-Z	65-90	97	162-187
a-z	97-122	39	136-161
1-9	48-57	78	126-135
Resto	32-125	0	32-125

Tabella 3: Tabella valore ordinamento

Come possiamo vedere da Tabella 3 il valore restituito dopo la somma permette di determinare facilmente l'ordine degli elementi.

Per semplicità di utilizzo la funzione *getOrderValue* prende due elementi per volta (**a0**,**a1**), dato che ogni volta che la chiamiamo dobbiamo eseguire un confronto tra due elementi. Essa ritorna il valore di ordinamento dei due elementi (**a0**,**a1**).

4.4 Funzione swap

Descrizione La funzione *swap* scambia di posizione due elementi della lista.

Implementazione La funzione ha come parametri (**a0**,**a1**) gli indirizzi dei due nodi. Essa legge il dato di ogni nodo e scrive il valore del primo nel secondo e viceversa. Lo scambio del dato permette di eseguire lo scambio eseguendo due scritture e due letture. Un Implementazione che scambia gli indirizzi sarebbe più costosa dato che bisognerebbe cambiare anche gli indirizzi dei nodi adiacenti.