Relazione progetto Architettura degli elaboratori

Giovanni Palmieri (7006086)

20 febbraio 2022

Indice

1 Introduzione

Il progetto prevede l'implementazione di una lista concatenata doppia. Ogni nodo della lista è strutturato secondo la Tabella 1. Nel caso in cui un nodo non abbia elemento successivo o precedente, PBACK o PAHEAD vengono impostati al valore -1 (0xFFFFFFFF in complemento a due).

Nome	Dimensione	Descrizione
PBACK	4 byte	Puntatore all'elemento precedente
DATA	1 byte	Informazione del nodo
PAHEAD	4 byte	Puntatore all'elemento successivo

Tabella 1: Struttura nodi

La lista prevede 5 operazioni:

- ADD (char): Aggiunta di un nuovo elemento in coda alla lista con DATA=char.
- DEL (char): Rimuove (se presente) il primo elemento della lista con DATA=char.
- PRINT: Stampa tutti i DATA degli elementi della lista in ordine di apparizione.
- SORT: Ordina in modo crescente gli elementi della lista in base a DATA.
- REV: Inverte gli elementi della lista.

I comandi da eseguire sono contenuti in una variabile *listInput* di tipo string. Sono separati da "~" e possono essere presenti spazi prima e dopo la tilde.

2 Descrizione del ciclo principale

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

2.1 Riconoscimento input

Dato che non esistono comandi diversi che abbiano la stessa lettera iniziale, distinguiamo i comandi in base alla prima lettera.

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

Tabella 2: Riconoscimento comandi

Dobbiamo però distinguere le lettere relative ai nomi dei comandi dai valori dei parametri dei comandi, esempio:

ⒶDD{Ⓐ}

Per evitare questo tipo di letture scorrette evitiamo direttamente di esaminare i parametri dei comandi. Per farlo ci avvaliamo del fatto che:

1. Il primo carattere che troviamo della stringa di input, non può essere il parametro di un comando.
2. Dato che non sono ammessi spazi all'interno dei comandi sappiamo la lunghezza di ogni comando.

Perciò ogni volta che riconosciamo la prima lettera di 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 3: Valori da sommare per evitare letture incorrette

Il "+1" in Tabella 3 ci permette di risparmiare la validazione di un char, dato che i comandi sono divisi da almeno un "~".

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 3

3 Add

3.1 Implementazione

Dato che la lista non ha un puntatore alla coda, ma solo un puntatore alla testa è necessario trovare l'ultimo elemento. Si può individuare perchè è l'unico elemento con PAHED uguale a -1. Una volta individuato l'ultimo elemento viene creato il nuovo nodo con PBACK uguale all' indirizzo dell'ultimo elemento e viene aggiornato il PAHEAD dell'ultimo elemento con l'indirizzo del nuovo elemento. L'allocazione dello spazio in memoria viene effettuata dalla funzione *getPseudoRandom* descritta nella sezione successiva. Nel caso in cui la lista non contenga elementi e quindi il punatore al primo elemento sia uguale a 0, basta creare il nuovo elemento con PBACK e PAHEAD uguali a -1 e impostare il puntatore al primo elemento all'indirizzo del nuovo elemento.

3.2 Funzione getPseudoRandom

Per prima cosa dobbiamo generare un numero casuale nel range della memoria del programma, per farlo usiamo l'LFSR (Linear-Feedback Shift Register). L'LFSR permette di generare un numero pseudo casuale eseguendo uno shift a destra e sostituendo il bit vuoto con il risultato di una XOR dei bit. Per funzionare la funzione ha bisogno di un **seed**, ossia di un valore iniziale, che viene impostato la prima volta che si chiama la funzione. Poi il **newBit** vine calcolato eseguendo lo XOR tra il primo, terzo, quarto e quinto bit. A questo punto si esegue lo shift a destra del valore corrente e si sostituisce il bit più valente con il **newBit**. Quest'operazione viene fatta su 16 bit, dato che risv-v usa 4 byte per indirizzare la memoria, ci restano i due byte più valenti da riempire. Per evitare overflow e/o sovrascritture di dati esistenti impostiamo i due byte a 0x00001.

3.3 Miglioramenti e variazioni

L'aggiunta del nuovo elento potrebbe essere resa più efficente salvando il puntatore all'ultimo elemento, in questo modo non ci sarebbe bisogno di scorrere tutta la lista per trovarlo. Lo svantaggio di questa implementazione è che si deve aggiornare il puntatore all'ultimo elemento ogni volta che l'ultimo elemento cambia.

4 Sort

4.1 Implementazione

L'algoritmo scelto per l'ordinamento è il selection sort. L'algoritmo prevede di scambiare ogni elemento della lista con l'elemento più piccolo tra gli elementi successivi per ogni posizione della lista. È adatto all'uso nelle liste concatenate dato che non si effettua mai l'accesso diretto ad una certa posizione ma si scorre sempre gli elementi. Sono quindi definite le seguenti funzioni:

- *getMinAddress* che restituisce l'indirizzo del nodo più piccolo tra i nodi successivi al nodo specificato.
- *sort* che scambia un nodo con il risultato di *getMinAddress*.

Entrambe le funzioni sono implementate ricorsivamente.

4.2 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

4.3 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

4.4 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 4: Tabella valore ordinamento

Come possiamo vedere da Tabella 4 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**).

4.5 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. Scambiando il dato dei nodi dobbiamo effettuare 2 letture e 2 scritture, mentre se volessimo scambiare i puntatori dovremmo effettuare ben 6 letture e 6 scritture, dato che dovremmo cambiare i due puntatori dei due nodi interessati e il puntatore al prossimo elemento dell'elemento precedente e il puntatore all'elemento precedente dell'elemento successivo. Come mostrato in Figura 1. Pertanto dato che il dato è di un solo byte, mentre i puntatori sono una word ciuascuno conviene scambiare i dati e lasciare i puntatori intatti.

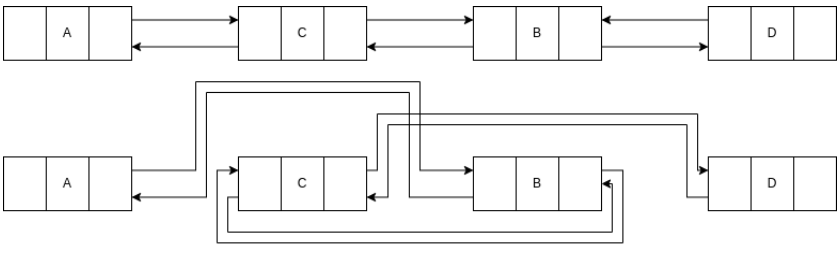

Figura 1: Prima e dopo ordinamento con scambio di puntatori

5 Delete

5.1 Implementazione

Il comando Del, rimuove il primo nodo il cui valore è uguale al parametro passato, se esiste, altrimenti non altera la lista. Per farlo la funzione scorre tutta la lista finchè non arriva alla fine, oppure finchè non trova un nodo con il valore da cancellare. Una volta trovato il nodo si distunguono 4 casi:

- Il nodo è l'ultimo nodo della lista, in questo caso si imposta il valore del penultimo nodo a -1, in questo modo diventa l'ultimo.
- Il nodo non è l'ultimo nodo della lista, in questo caso dobbiamo impostare i puntatori del nodo precedente e del nodo successivo come mostrato in Figura 2.
- Il nodo è il primo elemento della lista, in questo caso dobbiamo definire il nodo successivo come primo elemento della lista, aggiornado il puntatore al primo elemento della lista e mettendo il puntatore al nodo precedente del nuovo primo elemento.
- Il nodo è l'unico elemento della lista, bisogna ritornare allo stato iniziale del programma mettendo a 0 il puntatore al primo elemento.

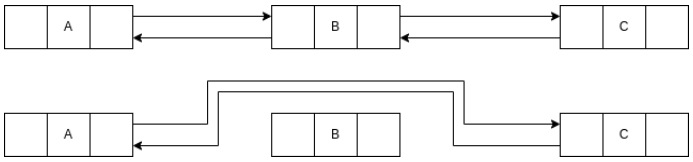

Figura 2: Prima e dopo rimozione di B

5.2 Miglioramenti e variazioni

Un possibile miglioramento sarebbe quello di azzerare l'area di memoria dei nodi una volta cancellati. Dato che in questo modo i nodi non vengono effettivamente cancellati dalla memoria ma semplicemente non vengono più indicizzati dalla lista. Così facendo l'area di memoria dei nodi cancellati non viene riconosciuta come libera ma non viene più utilizzata.

6 Reverse

6.1 Implementazione

Per invertire la lista basta invertire i puntatori PAHEAD e PBACK di ogni nodo e impostare il puntatore al primo elemento della lista con l'indirizzo dell'ultimo elemento. Come mostrato in Figura 3

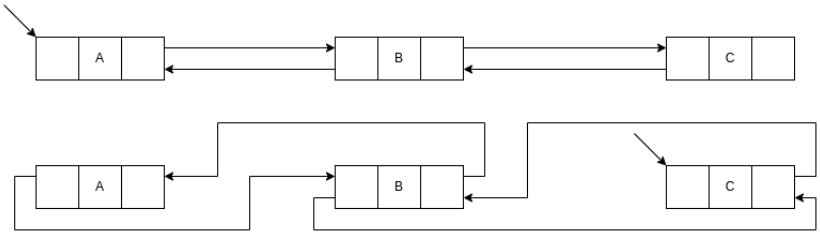

Figura 3: Rappresentazione lista prima e dopo reverse

7 Print

7.1 Implementazione

Per eseguire la stampa della lista si scorre tutti gli elementi e per ognuni si esegue la stampa in formato ascii del dato. Quando si arriva all'ultimo elemento viene stampata una newline.