⚒️

L2 - Modello RAM

Created	@September 22, 2022
Class	Alg. Strutt. Dati
Materials	

Esempio di programma

Creiamo una macchina di Turing per l'inversione di una stringa. Di questa macchina definiamo:

- Σ → Alfabeto classico
- Funzione di transizione:

$$\delta = \{(\alpha, \beta, \rightarrow), (\alpha, \beta, \leftarrow) \mid \alpha, \beta \in \Sigma\} \cup \{(\alpha, \rightarrow), (\alpha, \leftarrow) \mid \alpha \in \Sigma \setminus \sqcup\}$$

Prendiamo in input la stringa "TURING":

$$\begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|}\hline \sqcup & \sqcup & \sqcup & T & U & R & I & N & G & \sqcup & \sqcup & \sqcup \\ \hline\end{array}$$

Assumiamo che la testina inizi alla lettera più a sinistra che non sia un blank. Come viene memorizzata una lettera?

Sappiamo che:

- $q_i = (\sqcup, \sqcup, \rightarrow)$
- $q_f = (\sqcup, \sqcup, \leftarrow)$

L'idea di base è:

- Memorizziamo la lettera più a sinistra e spostiamoci con la testina fino alla fine della stringa, identificata dal primo spazio blank trovato. Successivamente torniamo con la testina indietro di una cella.
- Sostituiamo l'ultima lettera con la prima memorizzata inizialmente e memorizziamo l'ultima lettera. La testina a questo punto si sposterà verso l'inizio della stringa per posizionare l'ultima lettera al posto della prima.

- Il procedimento viene ripetuto per tutte le successive lettere fino al termine dell'inversione

Le funzioni di transizione possono essere rappresentate come:

$$\begin{aligned}\delta((\sqcup,\sqcup,\rightarrow),\alpha) = ((\alpha,\sqcup,\rightarrow),\sqcup,+1)\\ \delta((\alpha,\sqcup,\rightarrow),\beta) = ((\alpha,\beta,\rightarrow),\sqcup,+1)\end{aligned}$$

Il tutto può essere anche rappresentato con una tabella del tipo:

Input :

$Q \setminus q_f$	Σ
$(\sqcup,\sqcup,\rightarrow)$	A
$(\sqcup,\sqcup,\rightarrow)$	B
$(A,\sqcup,\rightarrow)$	A

Output :

Q	Σ	$\{-1,0,+1\}$
$(A,\sqcup,\rightarrow)$	$\sqcup$	$+1$
$(B,\sqcup,\rightarrow)$	$\sqcup$	$+1$
$(A,\sqcup,\rightarrow)$	A	$+1$

Modello RAM

- RAM → Random Access Machine

In un modello RAM la memoria è un nastro infinito a destra e finito a sinistra. A ogni cella del nastro è assegnato un indice che permette l'accesso diretto ai dati. Il numero di celle è al più uguale al numero di $\mathbb{N}$

La differenza sostanziale con la macchina di Turing è che è possibile l'accesso diretto a ogni cella di memoria conoscendo solo l'indice.

I programmi per un modello RAM possono essere scritti attraverso uno pseudo-linguaggio di programmazione, dotato di 4 costrutti principali, con relativo calcolo del tempo:

1. Assignment / Reading
2. Sequence
3. Selection
4. Iteration

1. $T(ass) = O(1)$
2. $T(p_1, p_2) = T(p_1) + T(p_2)$
3. $T(if) = T(cond) + max(T(p_1), T(p_2))$
4. Per l'iterazione non è possibile calcolare un tempo, siccome potrebbero non terminare mai

Esempio di programma

Scriviamo un programma per la ricerca delle coppie di valori all'interno dell'insieme:

$$\{(i,j) \in \mathbb{N} \times \mathbb{N} \mid i \leq j \leq n\}$$

Utilizziamo quattro differenti algoritmi, e per ognuno di essi ne calcoliamo la **complessità:**

> ## Calcolo del tempo necessario per l'esecuzione di un algoritmo

ALGORITMO 1:

```
1) counter <- 0
2) for i = 0 to n do:
3)  for j = 0 to n do:
4)    if i <= j then:
5)      counter++
6) return counter
```

Definiamo per ogni riga una costante di tempo necessaria per l'esecuzione dell'istruzione in modo atomico, e successivamente prenderemo in considerazione la ripetizione dei cicli e dei condizionali:

- 1) → t_1
- 2) → t_2
- 3) → t_3
- 4) → t_4
- 5) → t_5
- 6) → t_6

Calcoliamo ora il tempo complessivo dei cicli e dei condizionali:

- Il blocco if alla riga 4) avrà tempo al più $t_4 + t_5$
- Il blocco for alla riga 3) è composto sia da un calcolo del singolo incremento (t_3) sia il tempo per eseguire tutto il blocco (al più $n + 1$ volte). Il tempo complessivo del ciclo sarà quindi:

$$(n+1) * (t_3 + t_4 + t_5)$$

- Vale lo stesso ragionamento anche per il for alla riga 2):

$$(n+1) * [t_2 + (n+1) * (t_3 + t_4 + t_5)]$$

La formula complessiva sarà quindi:

$$t_1 + (n+1)t_2 + (n+1)^2(t_3 + t_4 + t_5)$$

Svolgiamo l'equazione mettendo in evidenza la variabile n, dal quale dipende il calcolo del tempo:

$$(t_3 + t_4 + t_5)n^2 + [2(t_3 + t_4 + t_5) + t_2]n + (t_1 + t_2 + t_3 + t_4 + t_5 + t_6)$$

Siccome le cariabili all'interno delle parentesi non sono da considerare nel calcolo della complessità, possiamo considerarle come variabili costanti:

$$c_2 n^2 + c_1 n + c_0$$

Da questa espressione capiamo che l'algoritmo utilizzato ha andamento (e quindi complessità) quadratico.

ALGORITMO 2:

Il secondo algoritmo che prendiamo in esame rimuove il condizionale if facendo partire il secondo ciclo dal valore di i:

```
1) counter <- 0
2) for i = 0 to n do:
3)  for j = i to n do:
4)    //if i <= j then:
5)      counter++
6) return counter
```

Calcoliamo quindi il tempo necessario per le singole istruzioni, prendendo sempre come variabili t quelle definite nell'algoritmo precedente:

- L'istruzione 5) verrà eseguita atomicamente con un tempo t_5 ma, insieme al ciclo 3), verrà eseguita interamente con un tempo: $(t_3 + t_5) * (n - i + 1)$
- Il ciclo 2) verrà considerato come: $\sum_{i=0}^{n}[t_2 + (n - i + 1) * (t_3 + t_5)]$, ovvero un calcolo complessivo di entrambi i cicli annidati

Scriviamo quindi l'equazione completa e svolgiamo i calcoli:

$$t_1 + \sum_{i=0}^{n}[t_2 + (n - i + 1)(t_3 + t_5)] + t_6 =$$

$$= t_1 + t_6 + \sum_{i=0}^{n}[t_2 + (n + 1)(t_3 + t_5) - i(t_3 + t_5)] =$$

$$= t_1 + t_6 + \sum_{i=0}^{n}[t_2 + (n + 1)(t_3 + t_5)] - \sum_{i=0}^{n}[i(t_3 + t_5)] =$$

$$= t_1 + t_6 + (n + 1)[t_2(n + 1)(t_3 + t_5)] - (t_3 + t_5)\sum_{i=0}^{n} i =$$

Sappiamo che $\sum_{i=0}^{n} i$ è la somma dei primi n numeri naturali: $\frac{n(n+1)}{2}$:

$$
\begin{aligned}
t_1 + t_6 + (n+1)[t_2(n+1)(t_3+t_5)] - [(t_3+t_5)(\frac{n(n+1)}{2})] = \\
= (n+1)^2(t_3+t_5) - \frac{n(n+1)}{2}(t_3+t_5) + (n+1)t_2 + t_1 + t_6 = \\
= (t_3+t_5)(n+1)[(n+1) - \frac{n}{2}] + (n+1)t_2 + t_1 + t_6 = \\
= (t_3+t_5)(n+1)[n+1-\frac{n}{2}] + nt_2 + (t_1+t_2+t_6) = \\
= [\frac{n^2}{2} + n + \frac{n}{2} + 1](t_3+t_5) + nt_2 + (t_1+t_2+t_6) = \\
= \frac{t_3+t_5}{2}n^2 + \frac{3(t_3+t_5)}{2}n + nt_2 + (t_1+t_2+t_6) = \\
= \frac{t_3+t_5}{2}n^2 + [\frac{3}{2}(t_3+t_5) + t_2]n + (t_1+t_2+t_3+t_5+t_6)
\end{aligned}
$$

Consideriamo tutte le variabili dipendenti da n come costanti:

$$c_2 n^2 + c_1 n + c_0$$

Abbiamo trovato nuovamente che l’algoritmo ha andamento quadratico, ma analizzando la grandezza delle costani possiamo assumere che questo sia più efficiente, essendo le variabili stesse più “piccole” delle precedenti

<u>ALGORITMO 3</u>:

Costruiamo il terzo algoritmo eliminando il secondo ciclo for:

```
1) counter <- 0
2) for i = 0 to n do:
3)  counter += (n-i+1)
4) return counter
```

Consideriamo il tempo necessario per il ciclo 2) come: $(n+1)(t_2+t_3)$

L’equazione sarà dunque:

$$
\begin{aligned}
t_1 + (n+1)(t_2+t_3) + t_4 = \\
= t_1 + t_4 + nt_3 + nt_2 + t_3 + t_2 = \\
= [(t_2+t_3)n + (t_1+t_2+t_3+t_4) = \\
c_1 n + c_0
\end{aligned}
$$

Questo algoritmo risulta estremamente più efficiente dei precedenti avendo andamento lineare. Si può ancora più semplificare semplicemente scrivendo:

ALGORITMO 4:

```
1) counter <- 0
2) return (n+1)((n/2)+1)
```

Questo algoritmo può essere scritto considerando il tempo necessario per l'esecuzione del ciclo for, che può essere scritto come $\sum_{i=0}^{n}(n-i+1)$. Eseguendo semplici operazioni l'equazione per il calcolo diventa: $(n+1)(\frac{n}{2}+1)$

Questo nuovo algoritmo è il più efficiente siccome, pur avendo un andamento quadratico, le uniche operazioni da calcolare sono puramente eseguite in modo atomico, di conseguenza possono essere considerate costanti