UNIVERSITÀ degli STUDI di CATANIA

DIPARTIMENTO DI MATEMATICA E INFORMATICA

Ricorrenze

Alessandro Ortis
Image Processing Lab - iplab.dmi.unict.it

alessandro.ortis@unict.it
www.dmi.unict.it/ortis/

IMAGE PROCESSING LABORATORY

Divide-and-Conquer

Secondo il paradigma ***divide et impera*** (o divide and conquer), risolviamo un problema in modo ricorsivo applicando tre passaggi ad ogni livello della ricorsione:

- ***Dividi:*** dividere il problema in un numero di sottoproblemi che sono istanze più piccole dello stesso problema.

- ***Conquista:*** risolvere i sottoproblemi ricorsivamente. Se le dimensioni del sottoproblema sono abbastanza piccole, si possono risolvere i sottoproblemi in modo semplice.

- ***Combina:*** combinare le soluzioni dei sottoproblemi ottenendo la soluzione del problema originale.

Relazioni di ricorrenza

La complessità di una funzione ricorsiva può essere espressa mediante una ***relazione di ricorrenza.***

L'analisi degli algoritmi ricorsivi si riduce spesso alla risoluzione di una o più equazioni di ricorrenza nelle quali si esprime il termine n-esimo di una sequenza in funzione dei termini precedenti.

Relazioni di ricorrenza

Le relazioni di ricorrenza descrivono in modo preciso le prestazioni degli algoritmi ricorsivi in esame.

Analizziamo adesso i metodi fondamentali per analizzare questi algoritmi e per derivare le soluzioni di alcune formule standard che ricorrono nell'analisi di molti algoritmi che studieremo.

La loro comprensione ci permetterà di capire a fondo le prestazioni degli algoritmi studiati in seguito.

Sommatorie

Quando un algoritmo contiene un ciclo, il suo tempo di esecuzione può essere espresso come sommatoria dei tempi impiegati in ogni esecuzione del corpo del ciclo.

Vediamo adesso le principali sommatorie e delle tecniche utili per trovare i loro limiti.

Sommatorie - Linearità

Per qualsiasi reale c e due sequenze finite $a_1 \dots a_n$ e $b_1 \dots b_n$ si ha:

$$\sum_{k=1}^{n}(ca_k + b_k) = c\sum_{k=1}^{n} a_k + \sum_{k=1}^{n} b_k$$

La linearità si applica anche alle sommatorie che includono termini asintotici:

$$\sum_{k=1}^{n} \Theta(f(k)) = \Theta\left(\sum_{k=1}^{n} f(k)\right)$$

Sommatorie – Serie aritmetiche

La sommatoria

$$\sum_{k=1}^{n} k = 1 + 2 + \cdots + n$$

è una serie aritmetica il cui valore è

$$\begin{aligned} \sum_{k=1}^{n} k &= \frac{1}{2}n(n+1) \\ &= \Theta(n^2) . \end{aligned}$$

Sommatorie – Quadrati e cubi

$$\sum_{k=0}^{n} k^2 = \frac{n(n+1)(2n+1)}{6},$$

$$\sum_{k=0}^{n} k^3 = \frac{n^2(n+1)^2}{4}.$$

Sommatorie – Serie geometriche

Per x reale diverso da 1, la sommatoria

$$\sum_{k=0}^{n} x^k = 1 + x + x^2 + \cdots + x^n$$

è una serie geometrica (o di potenze) il cui valore è

$$\sum_{k=0}^{n} x^k = \frac{x^{n+1} - 1}{x - 1} .$$

Sommatorie – Serie armoniche

Per n intero positivo, l'n-esimo numero armonico è

$$\begin{aligned} H_n &= 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{n} \\ &= \sum_{k=1}^{n} \frac{1}{k} \\ &= \ln n + O(1) . \end{aligned}$$

Limitare le sommatorie

Il metodo principale per calcolare il valore di una serie consiste nell’utilizzare l’induzione matematica.

Dimostriamo che: $$\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$$

Limitare le sommatorie

Il metodo principale per calcolare il valore di una serie consiste nell'utilizzare l'induzione matematica.

Dimostriamo che: $$\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$$

$$n = 1 \Rightarrow \sum_{k=1}^{1} k = 1 = \frac{1(1+1)}{2} = \frac{2}{2}$$

Per n=1 è vero, se vale anche per n+1 è vero per ogni n naturale.

Limitare le sommatorie

Per $n + 1$

Somma dei primi n+1 è pari alla somma dei primi n, più il (n+1)-esimo

$$
\begin{aligned}
\sum_{k=1}^{n+1} k &= \sum_{k=1}^{n} k + (n+1) \\
&= \frac{1}{2}n(n+1) + (n+1) \\
&= \frac{1}{2}(n+1)(n+2)\,.
\end{aligned}
$$

Limitare le sommatorie

Per $n + 1$

$$\begin{aligned}\sum_{k=1}^{n+1} k &= \sum_{k=1}^{n} k + (n+1) \\ &= \frac{1}{2}n(n+1) + (n+1) \\ &= \frac{1}{2}(n+1)(n+2)\,.\end{aligned}$$

Per IP induttiva sto supponendo che sia vero per la somma dei primi n numeri, quindi posso sostituire

Limitare le sommatorie

Per $n + 1$

$$\begin{aligned}
\sum_{k=1}^{n+1} k &= \sum_{k=1}^{n} k + (n+1) \\
&= \frac{1}{2}n(n+1) + (n+1) \\
&= \frac{1}{2}(n+1)(n+2) .
\end{aligned}$$

Fattorizziamo per (n+1)

Ricorrenze fondamentali

Questa formula si usa per programmi che ciclano sull'input ''eliminando'' un elemento per volta.

$$C_N = C_{N-1} + N \qquad per\ N \geq 2, C_1 = 1$$

C_N è circa $\dfrac{N^2}{2}$

Ricorrenze fondamentali

$$\boxed{N \geq 2, C_1 = 1}$$

$$C_N = C_{N-1} + N =$$

$$= C_{N-2} + (N-1) + N =$$

$$\cdot$$

$$\cdot$$

$$= C_1 + 2 + \ldots + (N-2) + (N-1) + N =$$

$$= 1 + 2 + \ldots + (N-2) + (N-1) + N =$$

$$= \frac{N(N+1)}{2}$$

Ricorrenze fondamentali

Questa formula si usa tipicamente con un programma ricorsivo che dimezza l'input ad ogni passo

$$C_N = C_{\frac{N}{2}} + 1 \qquad\qquad per\ N \geq 2, C_1 = 1$$

C_N è circa $\ln(N)$

NB. Per semplicità assumiamo che N sia pari.

Ricorrenze fondamentali

Questa formula si usa tipicamente con un programma ricorsivo che dimezza l'input ed esamina, eventualmente, ogni elemento di esso

$$C_N = C_{\frac{N}{2}} + N \qquad\qquad per\ N \geq 2, C_1 = 0$$

C_N è circa $2N$

NB. Per semplicità assumiamo che N sia pari.

Ricorrenze fondamentali

Questa formula si usa tipicamente con un programma ricorsivo che esegue una scansione lineare dell'input prima, durante, oppure dopo aver suddiviso l'input in due parti.

$$C_N = 2C_{\frac{N}{2}} + N \qquad per\ N \geq 2, C_1 = 0$$

C_N è circa $Nlog(N)$

Ricorrenze fondamentali

Questa formula si usa tipicamente con un programma ricorsivo che dimezza l'input ed esegue una quantità di lavoro addizionale costante.

$$C_N = 2C_{\frac{N}{2}} + 1 \qquad\qquad per\ N \geq 2, C_1 = 1$$

C_N è circa $2N$

Esempio: calcolo di potenze

Versione ricorsiva

$$\texttt{power(x,n)} = \begin{cases} \texttt{1} & \texttt{se n=0} \\ \texttt{x*power(x,n-1)} & \texttt{altrimenti} \end{cases}$$

```
power(x,n)
   if (n=0) return 1;
   else return x*power(x,n-1)
```

- n chiamate ricorsive
- tempo e spazio O(n)

Esempio: calcolo di potenze

Versione ricorsiva

$$\texttt{power(x,n)} = \begin{cases} \texttt{1} & \texttt{se n=0} \\ \texttt{x*power(x,(n-1)/2)}^2 & \texttt{se n è dispari} \\ \texttt{power(x,n/2)}^2 & \texttt{se n è pari} \end{cases}$$

```
power(x,n)
   if (n dispari)
   {  y=power(x,(n-1)/2);
      return x*y*y;
   }
   else {  y=power(x,n/2);
                return y*y; }
```

- log n chiamate ricorsive
- tempo e spazio O(log n)

Esempio: Segmenti di Somma Massima

- Segmento: sequenza di elementi consecutivi in una sequenza S
  - S array di n interi
  - S[i,j] segmento se $0 \leq i \leq j \leq n-1$
  - almeno un intero è positivo
- Determinare il segmento di somma massima
  - A parità di somma si predilige il segmento più corto

Prima Soluzione

Considero solo i segmenti di S: per ogni coppia di indici *(i,j)* considero la somma del segmento S[i…j].

```
int SommaMassima1(int S[], int n){
    int _max = 0;
    int somma;
    cout << "n= " << n << endl;
    for(int i = 0; i < n; i++) {
        for(int j = i; j < n; j++) {
            somma = 0;
            for(int k = i; k <= j; k++)
                somma = somma + S[k];
            if (somma > _max) _max = somma;
        }
    }
    return _max;
}
```

Prima Soluzione

Considero solo i segmenti di S: per ogni coppia di indici *(i,j)* considero la somma del segmento S[i…j].

```
int SommaMassima1(int S[], int n){
    int _max = 0;
    int somma;
    cout << "n= " << n << endl;
    for(int i = 0; i < n; i++) {
        for(int j = i; j < n; j++) {
            somma = 0;
            for(int k = i; k <= j; k++)
                somma = somma + S[k];
            if (somma > _max) _max = somma;
        }
    }
    return _max;
}
```

Costo: ~n^3

Prima Soluzione

```
for(int i = 0; i < n; i++)
    for(int j = i; j < n; j++) {
        somma = 0;
        for(int k = i; k <= j; k++)
            somma = somma + S[k];
        if (somma > _max) _max = somma;
    }
```

$$\sum_{i=0}^{n-1} \sum_{j=i}^{n-1} \sum_{k=i}^{j} t$$

- *t* ha un costo costante (es. 1).
- Il ciclo for più interno viene eseguito al massimo j – i + 1 volte (lunghezza del segmento esaminato).

Prima Soluzione

```
for(int i = 0; i < n; i++)
    for(int j = i; j < n; j++) {
        somma = 0;
        for(int k = i; k <= j; k++)
            somma = somma + S[k];
        if (somma > _max) _max = somma;
    }
```

$$\sum_{i=0}^{n-1}\sum_{j=i}^{n-1}\sum_{k=i}^{j} t$$

- t ha un costo costante (es. 1).
- Il ciclo for più interno viene eseguito al massimo j – i + 1 volte (lunghezza del segmento esaminato).

$$\sum_{i=0}^{n-1}\sum_{j=i}^{n-1}\sum_{k=i}^{j} t = \sum_{i=0}^{n-1}\sum_{j=i}^{n-1}(j-i+1)$$

Prima Soluzione

$$\sum_{i=0}^{n-1} \sum_{j=i}^{n-1} (j - i + 1)$$

Prima Soluzione

$$\sum_{i=0}^{n-1} \sum_{j=i}^{n-1} (j - i + 1)$$

$$\boxed{\sum_{j=i}^{n-1} (j - i + 1) = \sum_{j=1}^{n-i} j}$$

Prima Soluzione

$$\sum_{i=0}^{n-1} \sum_{j=i}^{n-1} (j - i + 1) = \sum_{i=0}^{n-1} \sum_{j=1}^{n-i} j$$

Prima Soluzione

$$\sum_{i=0}^{n-1}\sum_{j=i}^{n-1}(j-i+1)=\sum_{i=0}^{n-1}\sum_{j=1}^{n-i} j=$$

$$=\sum_{i=0}^{n-1}\frac{(n-i)(n-i+1)}{2}$$

Prima Soluzione

$$\sum_{i=0}^{n-1}\sum_{j=i}^{n-1}(j-i+1)=\sum_{i=0}^{n-1}\sum_{j=1}^{n-i}j=$$

$$=\sum_{i=0}^{n-1}\frac{(n-i)(n-i+1)}{2}\geq\sum_{i=0}^{n-1}\frac{(n-i)^2}{2}$$

Prima Soluzione

$$\sum_{i=0}^{n-1}\sum_{j=i}^{n-1}(j-i+1) = \sum_{i=0}^{n-1}\sum_{j=1}^{n-i} j =$$

$$= \sum_{i=0}^{n-1}\frac{(n-i)(n-i+1)}{2} \geq \sum_{i=0}^{n-1}\frac{(n-i)^2}{2} =$$

$$= \sum_{i=1}^{n}\frac{i^2}{2} = \frac{1}{2}\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{12} =$$

$$= \mathrm{O}(n^3)$$

$$\boxed{\sum_{k=0}^{n} k^2 = \frac{n(n+1)(2n+1)}{6},}$$

Seconda Soluzione: Idee

- Una volta calcolata somma(a[i,j-1]) evitiamo di ripartire da capo per somma(a[i,j])
- Utlizziamo il fatto che

  somma(a[i,j])=somma(a[i,j-1]) + a[j]
- Questo ci permette di risparmiare un ciclo for
  - Dunque otteniamo una soluzione quadratica

Seconda Soluzione

```
int SommaMassima2(int S[], int n){
    int _max = 0;
    int somma;
    for(int i = 0; i < n; i++) {
        somma = 0;
        for(int j = i; j < n; j++) {
            somma = somma + S[j];
            if (somma > _max) _max = somma;
        }
    }
    return _max;
}
```

Seconda Soluzione

Il ciclo esterno seleziona l'elemento iniziale, il ciclo interno trova la somma massima possibile con il primo elemento selezionato dal ciclo esterno e confronta questo massimo con il massimo complessivo.

Infine, restituisce il massimo complessivo. La complessità temporale è $O(n^2)$.