Tecniche Algoritmiche/1
Divide et Impera

Moreno Marzolla
http://www.moreno.marzolla.name/

Introduzione

- Dato un problema
  - Non ci sono "ricette generali" per risolverlo in modo efficiente
- Tuttavia, è possibile evidenziare quattro fasi
  1. Classificazione del problema
  2. Caratterizzazione della soluzione
  3. Tecnica di progetto
  4. Utilizzo di strutture dati

Non sono fasi strettamente sequenziali!

Classificazione di un problema

Fa parte di una classe più ampia di problemi?

- Problemi decisionali
  - Il dato di ingresso soddisfa una certa proprietà?
  - Soluzione: risposta sì/no
  - Es: Stabilire se un grafo è connesso
- Problemi di ricerca
  - Spazio di ricerca: insieme di "soluzioni" possibili
  - Soluzione ammissibile: soluzione che rispetta certi vincoli
  - Es: posizione di una sottostringa in una stringa

Introduzione

- Problemi di ottimizzazione
  - Ogni soluzione è associata ad una funzione di costo
  - Vogliamo trovare la soluzione di costo minimo
  - Es: cammino più breve fra due nodi
- Problemi di approssimazione
  - A volte, trovare la soluzione ottima è computazionalmente impossibile
  - Ci si accontenta di una soluzione approssimata:
    - costo basso, ma non sappiamo se ottimo
  - Es: problema del commesso viaggiatore

Caratterizzazione della soluzione

- Definire la soluzione dal punto di vista matematico
  - Spesso la formulazione è banale...
  - ... ma può suggerire una prima idea di soluzione
    - Es: Selection Sort: *"Data una sequenza di n elementi, una permutazione ordinata è data dall'elemento minimo seguito da una permutazione ordinata dei restanti n-1 elementi"*
- Le caratteristiche formali della soluzione possono suggerire una possibile tecnica algoritmica

Tecniche di progetto / 1

- Divide-et-impera
  - Un problema viene suddiviso in sotto-problemi *indipendenti*, che vengono risolti ricorsivamente (top-down)
  - Ambito: problemi di decisione, ricerca
- Programmazione dinamica
  - La soluzione viene costruita (bottom-up) a partire da un insieme di sotto-problemi *potenzialmente ripetuti*
  - Ambito: problemi di ottimizzazione

Tecniche di progetto / 2

- Tecniche greedy (algoritmi "golosi")
  - Ad ogni passo si fa sempre la scelta localmente ottima
- Backtrack
  - Procediamo per "tentativi", tornando ogni tanto sui nostri passi
- Ricerca locale
  - La soluzione ottima viene trovata "migliorando" soluzioni esistenti, ma non ottime

Divide-et-impera

Divide-et-impera

- Tre fasi:
  - *Divide*: Dividi il problema in sotto-problemi più piccoli e *indipendenti*
  - *Impera*: Risolvi i sotto-problemi ricorsivamente
  - *Combina*: "unisci" le soluzioni dei sottoproblemi
- Non esiste una ricetta "unica" per divide-et-impera:
  - Quick Sort: "divide" complesso, niente fase di "combina"
  - Merge Sort: "divide" banale, "combina" complesso
- È necessario uno sforzo creativo

Le torri di Hanoi

http://en.wikipedia.org/wiki/Tower_of_Hanoi

- Gioco matematico
  - tre pioli
  - *n* dischi di dimensioni diverse
  - Inizialmente tutti i dischi sono impilati in ordine decrescente (più piccolo in alto) nel piolo di sinistra
- Scopo del gioco
  - Impilare in ordine decrescente i dischi sul piolo di destra
  - Senza mai impilare un disco più grande su uno più piccolo
  - Muovendo al massimo un disco alla volta
  - Utilizzando se serve anche il piolo centrale

Le torri di Hanoi
Soluzione divide-et-impera

Sposta n dischi da p1 a p3 usando p2 come appoggio

http://en.wikipedia.org/wiki/Tower_of_Hanoi

```
hanoi(Stack p1, Stack p2, Stack p3, int n)
  if (n == 1) then
    p3.push(p1.pop())
  else
    hanoi(p1, p3, p2, n-1)
    p3.push(p1.pop())
    hanoi(p2, p1, p3, n-1)
  endif
```

- Divide:
  - n-1 dischi da p1 a p2
  - 1 disco da p1 a p3
  - n-1 dischi da p2 a p3
- Impera
  - Esegui ricorsivamente gli spostamenti

Le torri di Hanoi
Soluzione divide-et-impera

```
hanoi(Stack p1, Stack p2, Stack p3, int n)
  if (n = 1) then
    p3.push(p1.pop())
  else
    hanoi(p1, p3, p2, n-1)
    p3.push(p1.pop())
    hanoi(p2, p1, p3, n-1)
  endif
```

- Costo computazionale:
  - T(1) = 1
  - T(n) = 2T(n-1)+1 per n>1
- **Domanda:** Quale è la soluzione della ricorrenza?

Moltiplicazione di interi di grandezza arbitraria

- Consideriamo due interi di n cifre decimali, X e Y

$$X = x_{n-1}x_{n-2}\ldots x_1 x_0 = \sum_{i=0}^{n-1} x_i \times 10^i$$

$$Y = y_{n-1}x_{n-2}\ldots y_1 y_0 = \sum_{i=0}^{n-1} y_i \times 10^i$$

- Vogliamo calcolare il prodotto XY
  - L'algoritmo che abbiamo imparato a scuola ha costo $\Theta(n^2)$
  - Proviamo a fare di meglio con un algoritmo di tipo divide et impera

Idea

- Supponiamo che X e Y abbiano un numero pari di cifre
  - Altrimenti aggiungiamo zeri all'inizio
- Dividiamo le sequenze di cifre di X e Y in due parti uguali

$$X = X_1 \times 10^{n/2} + X_0$$
$$Y = Y_1 \times 10^{n/2} + Y_0$$

- Possiamo quindi calcolare il prodotto come:

$$X \times Y = (X_1 10^{n/2} + X_0) \times (Y_1 10^{n/2} + Y_0)$$
$$= (X_1 Y_1) \times 10^n + (X_1 Y_0 + X_0 Y_1) \times 10^{n/2} + X_0 Y_0$$

Osservazione

$$X \times Y = (\underline{X_1 Y_1}) \times 10^n + (\underline{X_1 Y_0} + \underline{X_0 Y_1}) \times 10^{n/2} + \underline{X_0 Y_0}$$

- La moltiplicazione per 10^n richiede tempo O(n)
  - Equivale ad uno shift di sinistra di n posizioni
- Ci sono 4 prodotti di numeri con n/2 cifre
- Possiamo scrivere la relazione di ricorrenza

$$T(n) = \begin{cases} 4T(n/2) + O(n) & n > 1 \\ 1 & n = 1 \end{cases}$$

- Soluzione (Master Theorem, caso 1): $\Theta(n^2)$
  - Non abbiamo migliorato nulla rispetto all'algoritmo banale :-(

Miglioramento

- Se poniamo

$$P_1=(X_1+X_0)\times(Y_1+Y_0)$$
$$P_2=(X_1 Y_1)$$
$$P_3=(X_0 Y_0)$$

possiamo scrivere

$$X\times Y=P_2\,10^n+(P_1-P_2-P_3)\times 10^{n/2}+P_3$$

- Il calcolo di P1, P2 e P3 richiede in tutto solo 3 prodotti tra numeri di n/2 cifre

Analisi del miglioramento

- Otteniamo la nuova relazione di ricorrenza

$$T(n)=\begin{cases}3\mathrm{T}(n/2)+O(n) & n>1\\ 1 & n=1\end{cases}$$

- In base al Master Theorem (caso 1) la soluzione è

$$T(n)=\Theta(n^{\log_2 3})=\Theta(n^{1.59})$$

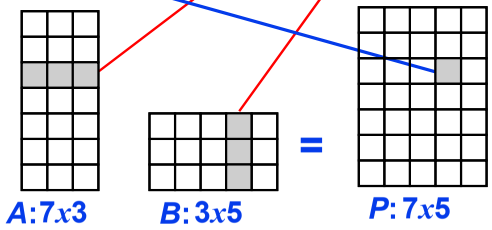
Moltiplicazione di matrici

- P = AB
  - *P[p, q] = A[p, c] x B[c, q]*

$$P[i,j]=\sum_{k=1}^{c} A[i,k]\times B[k,j]$$

```
mat-prod(A[p,c],B[c,q])→matrice p,q
  P := matrice p*q;
  for i := 1 to p do
    for j := 1 to q do
      sum := 0;
      for k := 1 to c do
        sum:=sum + A[i,k]*B[k,j];
      endfor
      P[i,j] = sum;
    endfor
  endfor
  return P;
```

Complessità
- *T(p,c,q) = p·c·q*
- *T(n) = Θ(n³)*



Come migliorare il prodotto fra matrici

- Suddividiamo le matrici n·n in quattro matrici n/2·n/2

$$A=\begin{pmatrix}A_{1,1} & A_{1,2}\\ A_{2,1} & A_{2,2}\end{pmatrix}\qquad B=\begin{pmatrix}B_{1,1} & B_{1,2}\\ B_{2,1} & B_{2,2}\end{pmatrix}$$

- Calcolo matrice:

$$P=\begin{pmatrix}A_{1,1}\cdot B_{1,1}+A_{1,2}\cdot B_{2,1} & A_{1,1}\cdot B_{1,2}+A_{1,2}\cdot B_{2,2}\\ A_{2,1}\cdot B_{1,1}+A_{2,2}\cdot B_{2,1} & A_{2,1}\cdot B_{1,2}+A_{2,2}\cdot B_{2,2}\end{pmatrix}$$

- Equazione di ricorrenza:

$$T(n)=\begin{cases}8\mathrm{T}(n/2)+O(n^2) & se\ \ n>1\\ 1 & se\ \ n=1\end{cases}$$

La soluzione della ricorrenza è **T(n) = Θ(n³)** cioè non meglio dell'algoritmo "banale"

Come migliorare il prodotto fra matrici

- Calcoliamo alcuni termini intermedi

$M_1=(A_{2,1}+A_{2,2}-A_{1,1})\cdot(B_{2,2}-B_{1,2}+B_{1,1})$
$M_2=A_{1,1}\cdot B_{1,1}$
$M_3=A_{1,2}\cdot B_{2,1}$
$M_4=(A_{1,1}-A_{2,1})\cdot(B_{2,2}-B_{1,1})$
$M_5=(A_{2,1}+A_{2,2})\cdot(B_{1,2}-B_{1,1})$
$M_6=(A_{1,2}-A_{2,1}+A_{1,1}-A_{2,2})\cdot B_{2,2}$
$M_7=A_{2,2}\cdot(B_{1,1}+B_{2,2}-B_{1,2}-B_{2,1})$

7 moltiplicazioni di matrici $n/2 \cdot n/2$

- Matrice finale: $P=\begin{pmatrix} M_2+M_3 & M_1+M_2+M_5+M_6 \\ M_1+M_2+M_4-M_7 & M_1+M_2+M_4+M_5 \end{pmatrix}$

Analisi del miglioramento

- Si ottiene la seguente relazione di ricorrenza:

$$T(n)=\begin{cases} 7T(n/2)+O(n^2) & se \quad n>1 \\ 1 & se \quad n=1 \end{cases}$$

- Usando il Master Theorem (caso 1) si ottiene la soluzione

$$T(n)=\Theta(n^{\log_2 7})=\Theta(n^{2.81})$$

Alcune informazioni storiche

- Algoritmo di Strassen (1969):
  - $\Theta(n^{2.81})$
  - Il primo ad "scoprire" che era possibile moltiplicare due matrici in meno di n^3 moltiplicazioni scalari
- Coppersmith and Winograd (1990):
  - $\Theta(n^{2.376})$
  - Attuale algoritmo migliore
- Limite inferiore
  - $\Omega(n^2)$
  - **Domanda**: perché?

$10000^3 = 1.00 \cdot 10^{12}$
$10000^{2.81} = 1.74 \cdot 10^{11}$
$10000^{2.38} = 3.31 \cdot 10^{9}$

Metodo divide-et-impera

- Quando applicare divide-et-impera
  - I passi "divide" e "combina" devono essere semplici
  - Ovviamente, i costi devono essere migliori del corrispondente algoritmo iterativo
  - Esempio ok: sorting
  - Esempio non ok: ricerca del minimo