

Insegnamento di Laboratorio di algoritmi e strutture dati

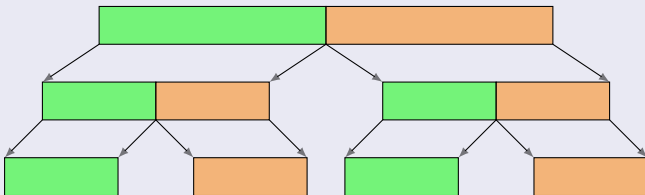
Programmazione dinamica

Roberto Posenato

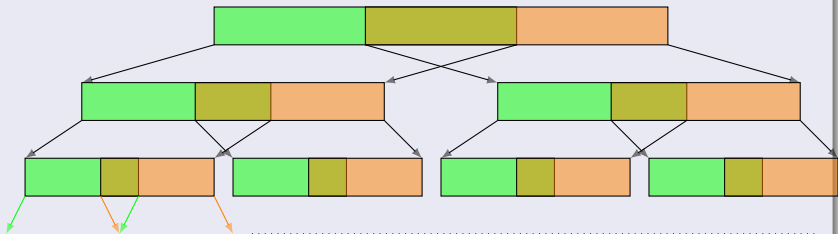
ver. 0.7, 29/01/2008

Introduzione

Quando si deve applicare la ricorsione



Quando **non** si deve applicare la ricorsione, ma la progr. dinamica



Introduzione

Calcolo numero di Fibonacci

Esempio 1 (Soluzione ricorsiva)

```
public long fib(int n) {  
    if (n < 0) throw new  
        IllegalArgumentException(  
            "Il parametro non può essere negativo." );  
  
    if (n <= 1) return n;  
    else return ( fib(n-1) + fib(n-2) );  
}
```

- primi termini calcolati un numero esponenziale di volte;
- complessità in tempo = $\Omega\left(\left(\frac{\sqrt{5}+1}{2}\right)^n\right)$

Introduzione

Calcolo numero di Fibonacci

Esempio 2 (Approccio programmazione dinamica)

```
public long fibD(int n) {  
    if (n < 0) throw new  
        IllegalArgumentException(...);  
    if (n <= 1) return n; //caso base  
    long risultato[] = new long[n+1];  
    risultato[0] = 0; risultato[1] = 1;  
    for (int k = 2; k <= n; k++)  
        risultato[k] = risultato[k-1]  
            + risultato[k-2];  
    return risultato[n];  
}
```

- memorizzazione risultati parziali
- complessità tempo = complessità in spazio = $\Theta(n)$;

Introduzione

Calcolo numero di Fibonacci

Esempio 3 (Approccio programmazione dinamica ottimizzato)

```
public long fibDO(int n) {  
    if (n < 0) throw new Illegal...  
    if (n <= 1) return n;  
    long terminePrecedente=0, risultato=1,  
    temp=0;  
    for (int k = 2; (k <= n); k++) {  
        temp = risultato;  
        risultato += terminePrecedente;  
        terminePrecedente = temp;  
    }  
    return risultato;  
}
```

- memorizzazione solo dei risultati parziali necessari
- complessità tempo = $\Theta(n)$
- complessità in spazio = $\Theta(1)$

Programmazione dinamica

Generalità

- Si consideri un programma ricorsivo, rappresentato dalla chiamate $\{P_1, P_2, \dots, P_M\}$
- Sia P_1 la chiamata principale con **un solo parametro** (per semplicità)
- Sia $[P_k, x]$ con $1 \leq k \leq M$, l'invocazione con il parametro x
- $[P_k, x]$ **dipende** da $[P_s, y]$ se esecuzione $P_k(x)$ richiede calcolo $P_s(y)$
- Dato $[P_1, z]$, si definisce l'**ordine lineare** $(L[P_1, z], <)$:
 - $L[P_1, z] = \{[P_s, y] \mid [P_1, z] \text{ dipende da } [P_s, y]\}$
 - Se $[P_k, x], [P_s, y] \in L[P_1, z]$ e $[P_k, x]$ dipende $[P_s, y]$, allora $[P_s, y] < [P_k, x]$

Programmazione dinamica

Generalità

- Dato $(L[P_1, z], <)$, si definiscano
 - **FIRST** = elemento più piccolo
 - **LAST** = $[P_1, z]$
 - $Succ([P_s, x])$ = elemento successivo nell'ordine lineare
 - $Succ([P_1, z])$ = null, in quanto non esiste!
- Algoritmo di programmazione dinamica che emula P_1
 - Dato z , costruisce vettore V con indici in $L[P_1, z]$ a partire da **FIRST**
 - restituisce $V[[P_1, z]]$

Programmazione dinamica

Generalità

Procedura $DP_1(z)$

```
1: Definisci ordine lineare  $L[P_1, z]$ ;  
2:  $i = FIRST$ ;  
3: while ( $i \neq null$ ) do  
4:   if ( $i == [P_k, x]$ ) then  
5:     esegui  $P_k(x)$  con le seguenti interpretazioni::  
6:     - eventuali chiamate ricorsive ' $b = P_s(l)$ ' come ' $b = V[[P_s, l]]$ ';  
7:     - l'istruzione ' $return E$ ' come ' $V[i] = E$ ';  
8:   endif  
9:    $u = i$ ;  
10:   $i = Succ(i)$ ;  
11: endw  
12: return  $V[u]$ ;
```

Programmazione dinamica

Generalità

Praticamente

- programmazione dinamica impiegata per risolvere problemi di ottimizzazione
- Passi per sviluppo algoritmo:
 - 1 caratterizzare struttura soluzioni ottimali
 - 2 definire in modo ricorsivo la soluzione ottimale
 - 3 fissare un naturale ordine lineare sulle chiamate ricorsive (calcolare valore dal basso-verso- alto)
 - 4 sostituire le chiamate con gli assegnamenti
 - 5 ottimizzare spazio usato, se possibile

Programmazione dinamica

Problema della massima sottosequenza comune

Problema della massima sottosequenza comune (MaxSSC)

Definizioni preliminari:

- Dato $X = \langle x_1, \dots, x_m \rangle$, $Z = \langle z_1, \dots, z_k \rangle$ è sotto sequenza di X se esiste sequenza crescente $\langle i_1, \dots, i_k \rangle$ di indici t.c.
 $\forall j \in \{1, \dots, k\} \ x_{i_j} = z_j$
- Esempio:
 - $X = \langle a, b, c, b, d, a, b \rangle$
 - $Z = \langle b, c, d, b \rangle$ in quanto la sequenza associata, $\langle 2, 3, 5, 7 \rangle$, soddisfa le condizioni

Programmazione dinamica

Problema della massima sottosequenza comune

Problema della massima sottosequenza comune (MaxSSC)

- Date X e Y , Z è **sotto sequenza comune** di X e Y se Z è sotto sequenza di X e Y
- Esempio:
 - $X = \langle a, b, c, b, d, a, b \rangle$
 - $Y = \langle b, d, c, a, b, a \rangle$
 - $Z = \langle b, c, a, b \rangle$
- **Problema massima sotto sequenza comune**
Date X e Y , determinare la sotto sequenza comune Z di lunghezza massima.

Programmazione dinamica

Problema della massima sottosequenza comune

Algoritmo forza *bruta*

- determinare tutte le sotto sequenze di X
- per ciascuna sotto sequenza, verificare se è sotto sequenza di Y , mantenendo traccia della più lunga
- **complessità?**

Programmazione dinamica

Problema della massima sottosequenza comune

Proprietà sotto sequenze massime

- Notazione: data $X = \langle x_1, \dots, x_i, \dots, x_m \rangle$, $X_i \stackrel{\text{def}}{=} \langle x_1, \dots, x_i \rangle$
- Date $X = \langle x_1, \dots, x_m \rangle$ e $Y = \langle y_1, \dots, y_n \rangle$ sia $Z = \langle z_1, \dots, z_k \rangle$ la maxSSC di X e Y
- Si dimostrano le seguenti proprietà:
 - 1 $x_m = y_n \implies z_k = x_m = y_n$ e Z_{k-1} è maxSSC di X_{m-1}, Y_{n-1}
 - 2 $x_m \neq y_n \implies \begin{cases} z_k \neq x_m \implies Z \text{ è maxSSC di } X_{m-1}, Y \\ z_k \neq y_n \implies Z \text{ è maxSSC di } X, Y_{n-1} \end{cases}$

Programmazione dinamica

Problema della massima sottosequenza comune

Osservazioni circa le proprietà sotto sequenze massime

- $x_m = y_n \implies$ si deve determinare $\max\text{SSC}(X_{m-1}, Y_{n-1})$ e poi appendere x_m
- $x_m \neq y_n \implies$ si devono risolvere 2 problemi
 - 1 $\max\text{SSC}(X_{m-1}, Y)$
 - 2 $\max\text{SSC}(X, Y_{n-1})$e scegliere la sotto sequenza più lunga.

Nota!

Entrambi i sottoproblemi dei punti 1 e 2 richiedono di determinare $\max\text{SSC}(X_{m-1}, Y_{n-1})$ durante la loro risoluzione.

Programmazione dinamica

Problema della massima sottosequenza comune

Schema programma

- $C[i, j]$ = lunghezza maxSSC per X_i e Y_j
- se $i = 0$ o $j = 0$ allora $C[i, j] = 0$
- $$C[i, j] = \begin{cases} 0 & \text{se } i = 0 \vee j = 0 \\ C[i-1, j-1] + 1 & \text{se } x, y > 0 \wedge x_i = y_j \\ \max(C[i-1, j], C[i, j-1]) & \text{se } x, y > 0 \wedge x_i \neq y_j \end{cases}$$

Programmazione dinamica

Problema della massima sottosequenza comune

Metodo per il calcolo di C

```
/** Costruisce la matrice delle lunghezze delle sotto  
 * stringhe comuni tra x e y.  
 * @param x prima stringa  
 * @param y seconda stringa  
 * @return la matrice delle lunghezze delle sottostringhe  
 * comuni. Le righe rappresentano tutte le sottostringhe  
 * di x, le colonne tutte le sottostringhe di y.  
 * @exception NullPointerException se x o y sono nulle.  
 */  
public static int[][] buildAllLengths(String x,  
                                     String y){  
    if (x == null || y == null) throw ...  
    int m = x.length() + 1; // la riga=0 per supporto  
    int n = y.length() + 1; // la colonna=0 per supporto  
    ...  
}
```

Programmazione dinamica

Problema della massima sottosequenza comune

continua metodo per il calcolo di C

```
int[][] c = new int[m][n];

for (int i = 1; (i < m); i++) {
    for (int j = 1; (j < n); j++) {
        if ( x.charAt(i-1) == y.charAt(j-1) ) {
            // indice stringa inizia a 0
            c[i][j] = c[i - 1][j - 1] + 1;
        } else {
            c[i][j] = (c[i-1][j] >= c[i][j-1])
                ? c[i - 1][j]
                : c[i][j - 1];
        }
    }
}

return c; // c[m-1][n-1] = lunghezza di MaxSSC
}
```

Programmazione dinamica

Esercizio

Esercizio 1

- completare la classe implementando il metodo `maxSSC()` visto a lezione;
- Eseguire dei test di correttezza. Provare con $X = \langle \rangle$ e Y qualsiasi. Provare con X completamente diverso da Y .
- Stampare il risultato della seguente istanza:

$X = dsuouitdltsmoahsfeskqsluefnezmaawmnasjsimflkiaismoaqq$
 $Y = ksokttjosemqmuaednjzkadmdalssdidmwjawnhdxy$