Immagine che contiene testo

Descrizione generata automaticamenteImmagine che contiene testo

Descrizione generata automaticamenteNel caso degli algoritmi bottom up e relative ricorrenze, nelle immagini sono illustrati i passaggi generici di programmazione dinamica nel caso Iterativo (prima immagine) e caso Ricorsivo (seconda immagine).

Si consideri in particolare che essendo un algoritmo bottom up, i costi sono *sempre* dati da una percorrenza inversa della matrice rispetto al senso di andata.

Cosa vuol dire? Se per esempio devo calcolare una LCS so per certo che, muovendomi nella matrice dall’alto in basso nella fase di andata per controllare dove si ha avuto un match, la fase di ritorno ragiona considerando gli indici precedenti (*i-1, j-1*), questo perché *si è già calcolata la soluzione ottima*, che detto in termini normali significa semplicemente che ho già calcolato con il costo ottimo quello che devo fare.

Sotto riassunto il caso LCS:

Immagine che contiene testo

Descrizione generata automaticamente

Si noti che la tabella è una struttura che viene sempre utilizzata; possono essere diversi sensi di scansione, per righe (*row major*), per colonne (*column major*) o al contrario (*reverse column major*/*reverse row major*). Dall’immagine si vede la scansione (in reverse i numeri vanno semplicemente invertiti).

Immagine che contiene testo, cruciverba

Descrizione generata automaticamente

Si ricordi che le ricorrenze considerano i casi

* i=0, j=0, nel qual caso il costo è sempre 0
* nel caso il nostro indice oltrepassi la grandezza dell’input, allora si pone ad infinito
* nel caso il nostro indice arrivi ad essere uguale alla taglia dell’input (n-1) o 0, sarà una costante
* quando si hanno i, j > 0 ma Xi = Xj  si ha il min o max *(dipende dal problema se min o max ovviamente)* degli indici calcolati
* quando si hanno i, j > 0 ma Xi != Xj  si ha min o max + la soluzione ottima calcolata

In alcuni casi come costo si somma anche +1 o +2 alla soluzione ottima calcolata; ciò dipende dal tipo del problema e generalmente si fa proprip perché uno dei casi basi è la costante 1 oppure 2 oppure per non considerare nullo l’input (come il caso degli alberi dove si somma +1 perché si può avere output nullo dopo la ricorsione a sinistra oppure a destra).

Per riassumere queste considerazioni, prendiamo il caso del calcolo di costi in una scacchiera, come:

Immagine che contiene testo

Descrizione generata automaticamente

Si vedano i valori commentati che rispecchiano quanto è stato descritto. Si veda anche che il movimento che noi compiamo all’interno della scacchiera è sempre dall’alto al basso, quindi posso avere il caso dove si muove la riga/la colonna insieme e ne calcolo il costo massimo.

Al di là del senso di scansione, generalmente negli algoritmi di programmazione dinamica si ha sempre:

* *un ciclo di inizializzazione*, in uno o due indici che sistema i casi base (pongo una matrice o array a 3 indici come 0, +infty, o similari. Il ciclo di inizializzazione serve a capire *se ho già calcolato una particolare soluzione ottima*.

In particolare posso definire una situazione similare alla precedente, introducendo il controllo struttura attuale e parametro che devo calcolare:

Immagine che contiene testo

Descrizione generata automaticamente

* *un ciclo di calcolo della soluzione ottima*, sempre basato su confronti (minore o maggiore)
* *ritorno della soluzione ottima*

Si intende quindi come ottimizzazione significhi aver già calcolato in modo “efficiente” una certa soluzione, ciclando il numero strettamente necessario di volte in base ai limiti di indici individuati.

Nel caso *greedy* invece, l’algoritmo si basa normalmente sulla scelta di un indice, o primo o ultimo. L’algoritmo greedy ricordiamo che fa una scelta degli input ottimali, in maniera tale da minimizzare il numero di confronti e risolvere facilmente un determinato sottoproblema.

Ad esempio nella selezione delle attività:

Immagine che contiene testo

Descrizione generata automaticamente

I passi si ripetono sempre fondamentalmente identici, in particolare si sceglie un punto particolare del nostro input e ciclo facendo un confronto.

Nel caso invece dovessi avere un intervallo di valori devo sempre considerare se è contiguo (caso intervalli la cui differenza è 1, sotto riportato) o è un intervallo minimo, nel qual caso devo tenere un indice in più *first* controllando in maniera identica agli algoritmi presenti.

