ELABORATO PER L'ESAME DI TECNICHE DI PROGETTAZIONE DI ALGORITMI

Bruno Degli Esposti – A.A. 2018/2019

Sfogliando il capitolo sulla programmazione dinamica del testo *Introduction to Algorithms*, mi ha incuriosito il seguente problema (pag. 409):

> ***15-8 Image compression by seam carving***
> We are given a color picture consisting of an $m \times n$ array $A[1..m, 1..n]$ of pixels, where each pixel specifies a triple of red, green, and blue (RGB) intensities. Suppose that we wish to compress this picture slightly. Specifically, we wish to remove one pixel from each of the m rows, so that the whole picture becomes one pixel narrower. To avoid disturbing visual effects, however, we require that the pixels removed in two adjacent rows be in the same or adjacent columns; the pixels removed form a "seam" from the top row to the bottom row where successive pixels in the seam are adjacent vertically or diagonally.
>
> ***a.*** Show that the number of such possible seams grows at least exponentially in m, assuming that $n > 1$.
>
> ***b.*** Suppose now that along with each pixel $A[i, j]$, we have calculated a real-valued disruption measure $d[i, j]$, indicating how disruptive it would be to remove pixel $A[i, j]$. Intuitively, the lower a pixel's disruption measure, the more similar the pixel is to its neighbors. Suppose further that we define the disruption measure of a seam to be the sum of the disruption measures of its pixels.
>
> Give an algorithm to find a seam with the lowest disruption measure. How efficient is your algorithm?

La ricerca del "seam with the lowest disruption measure" corrisponde alla ricerca di un cammino di valore minimo su una matrice, problema che in effetti si presta bene a essere risolto con un algoritmo di programmazione dinamica. Un tale algoritmo costa $\Theta(mn)$ ed è ottimo. Ho preparato un programma che risolve questo problema come progetto da portare all'esame orale. Per la realizzazione dell'elaborato ho utilizzato MATLAB e il relativo image processing toolbox. Un primo ostacolo che ho incontrato è che il testo dell'esercizio non fornisce un metodo per il calcolo di una

disruption measure. Per fortuna nelle note di fine capitolo gli autori rimandano, come approfondimento, a un articolo di libero accesso di Avidan e Shamir che contiene tutti i dettagli necessari, dal titolo *Seam Carving for Content-Aware Image Resizing*. Ho scaricato l'articolo, ho compreso il risultato principale e l'ho riprodotto in MATLAB con successo. Ho incluso l'articolo nel materiale che le ho inviato. Se ha tempo, la invito a leggere le prime pagine (fino al paragrafo 4.2).

La tecnica introdotta nell'articolo è stata in seguito migliorata e oggi è impiegata nei software di computer grafica come photoshop per ridimensionare un'immagine lasciando invariate le dimensioni dei soggetti in primo piano, limitandosi a ingrandire o ridurre le dimensioni dello sfondo (a patto che sia sufficientemente omogeneo). Ecco un esempio di output prodotto dal programma che ho scritto:

Sopra: immagine originale
A fianco: output del programma

Segue una breve descrizione di ciascun file sorgente:

- Energy_matrix.m - Funzione che calcola la disruption measure, o energia, di ogni pixel in un'immagine. Seguendo il lavoro di Avidan e Shamir, l'energia di un pixel è definita come la norma 1 del gradiente dell'immagine in quel punto. Il gradiente viene approssimato mediate l'operatore differenziale discreto di Sobel. Se l'immagine ha più canali di colore, per esempio 3 per un'immagine RGB, l'energia totale è data dalla somma dell'energia presente in ogni canale.
- Energy_mean.m – Funzione che calcola l'energia media dei pixel in un'immagine. Ogni volta che da un'immagine viene rimosso un cammino di minima energia, ci si aspetta che l'energia media cresca. Date due possibili rimozioni, per esempio una orizzontale e una verticale, risulta preferibile quella che produce il maggior aumento dell'energia media.
- Seam_h.m – Funzione che contiene l'algoritmo di programmazione dinamica e risolve il problema 15-8 del libro di testo.

- Seam_v.m – Funzione analoga alla precedente, ma che calcola il cammino di minima energia verticale anziché orizzontale. La funzione è meno ottimizzata, ma forse più leggibile. In ogni caso non sarà mai performante quanto seam_h, perché nella programmazione dinamica i pattern di accesso alla matrice seguono le righe e MATLAB memorizza le matrici colonna per colonna, quindi gli accessi alla memoria non sono localizzati e la cache del processore non viene utilizzata al meglio.
- Carve_h.m – Funzione che rimuove un cammino orizzontale da un'immagine, lasciandola così con una riga in meno.
- Carve_v.m – Funzione che rimuove un cammino verticale da un'immagine, lasciandola così con una colonna in meno.
- Main.m – Script principale, pronto per essere eseguito. Alle variabili remove_h e remove_v vengono assegnati i numeri di righe e colonne da rimuovere nell'immagine. Sorge un problema: per alcune immagini è preferibile rimuovere prima le righe e poi le colonne, per altre immagini il contrario, per altre immagini ancora procedere alternando le due modalità di rimozione. In breve, la rimozione di righe e colonne non "commuta". Il paragrafo 4.2 dell'articolo suggerisce di applicare nuovamente una tecnica di programmazione dinamica per determinare l'ordine ottimo in cui rimuovere righe e colonne, arrivando a produrre un'immagine finale con la massima energia media tra tutte quelle possibili.
  Ho incontrato due difficoltà nel replicare questo approccio. In primo luogo, non mi è chiaro perché i sottoproblemi presentino una struttura ottima, ipotesi fondamentale per poter impiegare la programmazione dinamica. L'articolo non fornisce alcuna giustificazione al riguardo. In secondo luogo, le funzioni seam_h e seam_v non sono sufficientemente ottimizzate per essere eseguite $\Theta(\text{remove\_h} * \text{remove\_v})$ volte in un tempo ragionevole. Così ho preferito un'euristica greedy dal costo $\Theta(\text{remove\_h} + \text{remove\_v})$, che produce comunque risultati molto buoni.

Oltre ai file sorgente, ho incluso diverse immagini di prova su cui poter eseguire il programma.