S283331
Manuel Firrera

Relazione esame 30-06-2023 **Traccia da 18 punti**

Strutture dati ed acquisizione

La struttura dati scelta è una matrice `grid` di struct `tessera_t` contenenti ciascuna il vettore di 4 interi corrispondenti alla tessera stessa e un identificatore di tipo, dato dal conteggio dei valori 1 nel vettore stesso.

La scelta è stata compiuta nell'ottica di delegare alla funzione di acquisizione dati il compito di riconoscere il tipo di tessera.

Non sono state effettuate modifiche degne di nota.

Problema di verifica

Il formato del file contenente la soluzione è:

<N> <m1> <m2> … <mn>

Con N=numero di mosse ed mn=mossa n.

Le mosse sono codificate nella maniera seguente:

1=SU, 2=GIU, 3=SINISTRA, 4=DESTRA;

La funzione di verifica parte leggendo la soluzione proposta per poi compiere i passi scelti su una matrice-copia temporanea in forma row-major realizzata da `grid_copy()`.

0	1	2
3	4	5
6	7	8

Rappresentazione degli indici della griglia row-major

Nella versione completa è stata implementata la funzione `grid_copy()`.

Nella versione corretta viene utilizzata questa copia, `grid_1`, per memorizzare gli spostamenti.

Nella versione del programma in aula i valori N=dimensione e N=numero di mosse sono dichiarati utilizzando la stessa variabile, nel programma corretto n=numero di mosse.

Il primo `if` effettua la prima verifica di validità, il numero N di mosse DEVE essere minore di M numero massimo di mosse.

Seguono poi le varie casistiche gestite tramite lo switch-case, tutte basate sulla considerazione che lo spostamento, assimilabile ad uno scambio tra tessera vuota e tessera selezionata, è possibile solamente se il tipo di tessera-target è 0, ovvero tessera vuota.

I casi SU e GIU garantiscono spostamenti leciti interrompendo il ciclo alla penultima riga della matrice,

i casi DX ed SX controllano i resti di divisioni dipendenti da N=dimensione matrice per la stessa ragione.

E' stato necessario inserire le divisioni in dei costrutti `if-continue` che fanno saltare il ciclo for di un passo.

Infine la copia viene passata alla funzione di verifica wrapper `verifica_conn()`.

`Verifica_conn()` compie delle operazioni preliminari di identificazione della posizione di una delle due sorgenti e della prima uscita della sorgente per poi richiamare l'effettiva funzione di verifica ricorsiva `verifica_connR()`.

Quest'ultima segue passo-passo il percorso realizzato e verifica man-mano che l'uscita del tubo selezionato combaci con l'ingresso del tubo successivo, viene inoltre verificata che la direzione dell'uscita non porti ad uscire dai bordi della matrice.

Nel programma completo è stata passata la variabile `src_index` ed inserita nell'`if` di chiusura della ricorsione per evitare di considerare la sorgente di inizio come sorgente di fine.

Problema di ricerca ed ottimizzazione

Il problema di ricerca ed ottimizzazione genera tutte le possibili disposizioni ripetute di mosse codificate nella stessa modalità del file di input.

Le disposizioni sono di dimensioni i crescenti fino ad M, la verifica è demandata alle funzioni descritte al punto 2.

La funzione chiamante `cerca_sol()` effettua un controllo sulla variabile globale flag ad ogni iterazione della dimensione i, in modo da interrompere il ciclo nel caso in cui la soluzione fosse trovata prima del raggiungimento del valore massimo M;

Note:

La versione completa contiene dei pezzi commentati utili alla visualizzazione delle strutture dati, dei passi intermedi di verifica e della ricerca della soluzione ottima.

S283331
Manuel Firrera