Progetto di Laboratorio Algoritmi e Strutture Dati

Università di Bologna, corso di laurea in Ingegneria e Scienze Informatiche

Anno Accademico 2023/2024

Versione 1.0, 6 maggio 2024

Istruzioni

Il progetto consiste in un programma da realizzare in ANSI C (detto anche C89 o C90). Questo documento descrive le specifiche e le modalità di svolgimento e valutazione.

Modalità di svolgimento del progetto

I programmi devono essere conformi ANSI C, e verranno compilati usando il compilatore GCC con la seguente riga di comando:

```
gcc -std=c90 -Wall -Wpedantic file.c -o file
```

(dove il nome del file verrà sostituito dal nome del sorgente consegnato; maggiori dettagli nel seguito). Il compilatore non deve segnalare *warning*, soprattutto se relativi a problemi facilmente risolvibili come variabili/funzioni non utilizzate, variabili usate prima di essere inizializzate, funzioni non-void in cui non si restituisce un risultato, eccetera. Saranno ammessi alcuni *warning* specifici di Visual Studio (es., quelli relativi alla sostituzione di `fopen` con `fopen_s` e simili; tali warning devono essere ignorati, dato che `fopen_s` non fa parte di ANSI C).

La funzione `main()` deve restituire 0 (oppure `EXIT_SUCCESS`) al termine dell'esecuzione corretta; è lasciata libertà di restituire un codice di errore nel caso in cui il programma termini in modo anomalo. Tuttavia, non si dovrebbero mai verificare terminazioni anomale perché tutti gli input che verranno usati sono garantiti essere corretti.

I programmi devono produrre output a video **rispettando scrupolosamente il formato indicato in questo documento e negli esempi forniti**. I programmi verranno inizialmente controllati in modo semi-automatico, e respinti in caso di output non conforme alle specifiche.

I programmi **non devono interagire in alcun modo con l'utente**: non devono eseguire comandi di sistema (es., `system("pause")`), né richiedere all'utente di premere invio, inserire informazioni da tastiera, o altro.

I programmi non devono presentare accessi “out-of-bound” né altri tipi di accessi non validi alla memoria; devono inoltre liberare in modo esplicito con `free()` tutta la memoria allocata con `malloc()` prima di terminare. Questi aspetti sono importanti perché possono produrre problemi non immediatamente evidenti (es, gli accessi “out-of-bound” potrebbero causare un crash con certe combinazioni di compilatore e sistema operativo, e non con altre), per cui è necessaria la massima attenzione. Per verificare l'uso corretto della memoria verrà usato il programma *valgrind* in ambiente Linux. In particolare, i programmi verranno esaminati anche con il seguente comando:

```
valgrind ./nome_eseguibile nome_file_di_input
```

che non deve segnalare errori. Ricordarsi di chiudere il file di input con `fclose()`, dato che in caso contrario *valgrind* segnalerà memoria non liberata (si tratta di memoria che viene riservata internamente da `fopen()` e liberata da `fclose()`).

Vengono forniti alcuni file di input con i corrispondenti risultati attesi; in certi casi possono esistere più soluzioni ottime, e i programmi verranno considerati corretti se ne calcolano una qualsiasi (maggiori informazioni in seguito). Si tenga presente che **un programma che produce il risultato corretto con gli input forniti non è necessariamente corretto**. I programmi verranno testati anche con input diversi da quelli forniti.

I programmi devono rispettare per quanto possibile le indicazioni contenute nella dispensa *Programmazione: Breve guida di stile* disponibile sulla pagina del laboratorio, il cui contenuto fa parte integrante di questa specifica. Verranno quindi valutati anche aspetti non funzionali (efficienza, chiarezza del codice, ecc.) che, se non soddisfatti, potrebbero portare ad una valutazione negativa e la necessità di modificare il sorgente e riconsegnare.

Sulla piattaforma Virtuale è stato predisposto un forum di discussione, nel quale è possibile chiedere chiarimenti sulle specifiche dell'elaborato (ossia, su questo documento); tutte le domande devono essere poste esclusivamente sul forum. Poiché il progetto è parte dell'esame finale, va trattato con la dovuta serietà: di conseguenza, **non faremo debug del codice né risponderemo a quesiti di programmazione in C**. Si assume che chi ha seguito questo corso sappia già programmare in C (che come detto all'inizio del corso, è un prerequisito di questo laboratorio).

Il progetto deve essere svolto **individualmente**; non è consentito discutere il progetto o le soluzioni con altri studenti o con terzi. La similarità tra programmi verrà valutata con strumenti automatici e, se confermata, comporterà l'annullamento delle consegne per **tutti** gli studenti coinvolti, con la necessità di consegnare un nuovo progetto su nuove specifiche, a partire dalla sessione d'esame successiva.

È consentito l'uso del codice messo a disposizione dal docente durante il laboratorio, incluse le soluzioni degli esercizi proposti. **Non è consentito fare uso di altro codice, anche se liberamente disponibile in rete**. Si tenga presente che, sebbene sia stata messa la massima cura nello sviluppo dei programmi distribuiti a lezione, non se ne garantisce la correttezza. Ognuno sarà quindi **interamente responsabile del codice consegnato** e si assumerà la responsabilità di ogni errore, anche se presente nel codice fornito dal docente.

Modalità di consegna

L'elaborato va consegnato tramite la piattaforma "Virtuale" in un unico file sorgente il cui nome deve essere il proprio numero di matricola (es., `0000123456.c`). Tutte le funzioni devono essere definite all'interno di tale file, escluse quelle della libreria standard C. All'inizio del sorgente deve essere presente un commento in cui si indica nome, cognome, numero di matricola, classe (A oppure B) e indirizzo mail (`@studio.unibo.it`) dell'autore/autrice.

Le date di consegna sono indicate sulla piattaforme Virtuale, e sono indicativamente 7 giorni prima di ogni appello scritto. Dopo ciascuna scadenza non sono possibili nuove consegne fino alla data dell'esame, dopo la quale le consegne saranno riaperte fino alla scadenza successiva, e così via fino all'ultimo appello dell'anno accademico.

Valutazione

Dopo la chiusura delle consegne, i programmi riceveranno una valutazione binaria (0 = insufficiente, 1 = sufficiente). I progetti valutati positivamente consentono di sostenere la prova scritta in tutti gli appelli d'esame successivi, anche di anni accademici diversi: i progetti valutati positivamente non scadono mai, e pertanto non bisognerà riconsegnare alcun progetto anche se si sostiene l'esame in anni successivi.

In caso di valutazione negativa sarà necessario modificare il programma e ripresentarlo alla riapertura delle consegne.

Checklist

Viene riportata in seguito un elenco di punti da controllare prima della consegna:

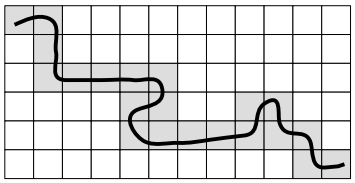
1. Il programma è stato consegnato in un unico file il cui nome coincide con il proprio numero di matricola?
2. È presente un commento iniziale che riporta cognome, nome, numero di matricola, gruppo (A/B) e indirizzo di posta (`@studio.unibo.it`) dell'autore?
3. Il programma è conforme allo standard ANSI C (detto anche C89 o C90)?
4. Nella stesura del codice sono stati seguiti (per quanto possibile) i suggerimenti nella dispensa?
5. Il programma compila senza *warning*?

6. Il programma è conforme alle specifiche? L'output sui casi di test è corretto e rispetta scrupolosamente il formato indicato in questo documento e nei file di esempio?
7. Il programma libera tutta la memoria allocata con malloc() prima della terminazione? È stata controllata (per quanto possibile) l'assenza di *memory leak* e accessi *out-of-bound*?

La strada

Una certa area geografica è descritta da una mappa suddivisa in una griglia regolare di n righe e m colonne. La matrice $H[0..n - 1, 0..m - 1]$ indica l'altezza media sul livello del mare di ciascuna cella: $H[i, j]$ indica l'altezza media sul livello del mare della cella (i, j) che si trova nella riga i e colonna j. Le altezze sono numeri interi anche negativi, perché una porzione di terreno può trovarsi sotto il livello del mare.

Si vuole costruire una strada che parta dalla cella (0, 0) (in alto a sinistra), e termini nella cella $(n - 1, m - 1)$ (in basso a destra). La strada deve passare attraverso celle adiacenti, cioè che hanno un lato in comune. Un esempio di strada valida è rappresentata nella figura seguente; la strada attraversa le celle evidenziate in grigio.



Supponiamo che la strada attraversi le k celle di coordinate (x_0, y_0), (x_1, y_1), … (x_{k-1}, y_{k-1}); è necessario rispettare i vincoli seguenti:

1. La prima cella (x_0, y_0) deve essere (0, 0).
2. L'ultima cella (x_{k-1}, y_{k-1}) deve essere $(n - 1, m - 1)$.
3. Ogni cella deve avere un lato in comune con la successiva. Formalmente, per ogni $i = 0, \ldots k - 2$, la cella (x_i, y_i) deve avere un lato in comune con (x_{i+1}, y_{i+1}).
4. La strada non può mai attraversare più di una volta la stessa cella.
5. La strada non può uscire dai bordi della griglia.

La costruzione della strada ha un costo che si compone di due parti:

1. Per ogni cella attraversata si sostiene un costo fisso C_{cell} intero non negativo.
2. Per ogni coppia di celle adiacenti attraversate, (x_i, y_i) e (x_{i+1}, y_{i+1}), si paga un ulteriore costo pari a C_{height} moltiplicato per il quadrato del dislivello, cioè $C_{height} \times (H[x_i, y_i] - H[x_{i+1}, y_{i+1}])^2$. Tale costo è zero nel caso in cui non ci sia dislivello tra (x_i, y_i) e (x_{i+1}, y_{i+1}). C_{height} è un intero non negativo.

Quindi, se la strada attraversa le k celle (x_0, y_0), … (x_{k-1}, y_{k-1}), il suo costo totale è dato da

$$C_{cell} \times k + C_{height} \times \sum_{i=0}^{k-2} \left(H[x_i, y_i] - H[x_{i+1}, y_{i+1}] \right)^2$$

Progettare e realizzare un algoritmo efficiente che determina le celle su cui costruire una strada di costo complessivo minimo, rispettando i vincoli di cui sopra. Nel caso in cui esistano più strade di costo minimo è sufficiente calcolarne una qualsiasi. Si noti che la soluzione ottima a questo problema non coincide necessariamente con quella che attraversa il minor numero di celle.

Per evitare possibili problemi di overflow (che comunque non dovrebbero verificarsi con gli input che verranno forniti), si usi il tipo `long int` oppure `double` per il costo totale della strada. Nel caso in cui si usi il tipo `double`, si tenga presente che il risultato va stampato come intero, cioè senza la parte decimale (che comunque deve essere sempre zero perché tutti i parametri del problema sono interi).

Input

Il programma deve accettare accetta un unico parametro sulla riga di comando, che rappresenta il nome di un file di testo il cui contenuto rispetta la struttura seguente:

- la prima riga contiene il costo C_{cell} (intero non negativo);
- la seconda riga contiene il costo C_{height} (intero non negativo)
- la terza riga contiene il numero di righe n della matrice (intero non negativo; $5 \leq n \leq 250$);
- la quarta riga contiene il numero m di colonne della matrice (intero non negativo, $5 \leq m \leq 250$);
- seguono n righe ciascuna contenente m interi separati da spazi o tabulazioni; ognuna di queste righe contiene i valori di una riga della matrice H descritta sopra.

Output

Il programma deve stampare a video:

- le coordinate delle k celle che fanno parte del cammino di costo minimo, una per riga; le coordinate (x_k, y_k) di ogni cella vanno stampate su una riga come coppia di valori interi x_k y_k separati da spazi o tabulazioni;
- segue una riga contenente la coppia di valori -1 -1, separati da spazi o tabulazioni, per indicare che il cammino è finito;
- segue una riga contenente il costo totale della strada, calcolata con la formula di cui sopra e stampato come intero (cioè senza parte decimale, che comunque deve sempre essere zero).

Nel caso in cui esistano più soluzioni ottime, il cammino e/o la sua lunghezza (intesa come numero di celle attraversate) potranno essere diversi da quelle riportate nei file di output; il costo totale però deve essere lo stesso. In tal caso una soluzione sarà considerata corretta se il cammino stampato è valido (anche se diverso dall'output fornito) e il suo costo è uguale a quello minimo indicato nella soluzione.

Esempi

Input:	*Output atteso:*
50 1000 6 5 100 123 129 98 100 99 124 108 102 101 98 99 127 105 104 102 112 100 102 103 105 107 182 198 104 105 106 170 207 100	0 0 1 0 2 0 3 0 4 0 5 0 5 1 4 1 3 1 3 2 3 3 3 4 4 4 5 4 -1 -1 220700
Input:	*Output atteso:*
100 80 5 5 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 -10 -10 -10 -10 -10 -10 -10 -10 -10 -10	0 0 1 0 2 0 3 0 4 0 4 1 4 2 4 3 4 4 -1 -1 968900

Si ricorda che negli esempi forniti, inclusi quelli sopra, possono esistere più soluzioni ottime; è sufficiente stamparne una qualsiasi. In ogni caso tutte le soluzioni ottime devono avere lo stesso costo totale (il valore stampato sull'ultima riga).