Progetto di Laboratorio Algoritmi e Strutture Dati

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

Anno Accademico 2022/2023

Versione 1.0, 12 maggio 2023

Istruzioni

Il progetto consiste in un esercizio di programmazione da realizzare in ANSI C. Questo documento descrive le specifiche del progetto e le modalità di svolgimento e valutazione.

Modalità di svolgimento del progetto

Il progetto deve essere svolto **individualmente**; non è consentito discutere 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 specifiche diverse.

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.

I programmi 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). I programmi devono essere conformi allo standard ANSI C (detto anche C89 oppure C90). 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 che non ritornano 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 diverso nel caso in cui il programma termini in modo anomali. Tuttavia, non si dovrebbero mai verificare terminazioni anomale dato che gli input forniti saranno sempre corretti.

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

I programmi verranno testati prevalentemente in ambiente Ubuntu Linux, ma devono funzionare correttamente anche su Windows e MacOSX. 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 tutta la memoria allocata con malloc() prima di terminare. Questi aspetti sono molto insidiosi perché possono risultare 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 in fase di sviluppo e test. Per il controllo dell'uso corretto della memoria verrà usato il programma *valgrind* in ambiente Linux, come illustrato brevemente nella prima lezione di laboratorio. In particolare, i programmi verranno esaminati anche con il seguente comando:

```
valgrind ./nome_eseguibile nome_file_di_input
```

che non deve riportare alcun errore, né deve riportare memoria allocata ancora in uso al termine del programma.

Vengono forniti alcuni file di input con i corrispondenti risultati attesi. Si può assumere che i programmi vengano sempre eseguiti con input corretti, per cui non è richiesto di includere controlli (sebbene ciò sia una buona pratica). 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 scrupolosamente le indicazioni contenute nella dispensa *Programmazione: Breve guida di stile* disponibile sulla pagina del corso, 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 con la conseguente 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**. Infatti, come detto all'inizio del corso, si assume che chi ha seguito questo corso sappia già programmare in C (prerequisito del corso).

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 necessarie devono essere definite all'interno di tale file, escluse ovviamente quelle della libreria standard C. Tutti i programmi devono iniziare con un commento che indichi nome, cognome, numero di matricola, gruppo (A oppure B) e indirizzo mail (`@studio.unibo.it`) dell'autore/autrice.

Le date di consegna sono indicate sulla piattaforme Virtuale, e sono indicativamente 10 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 presentati 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).

In caso di valutazione negativa sarà necessario modificare il programma e riconsegnarlo alla riapertura delle consegne, che avverrà dopo l'appello scritto.

Checklist

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

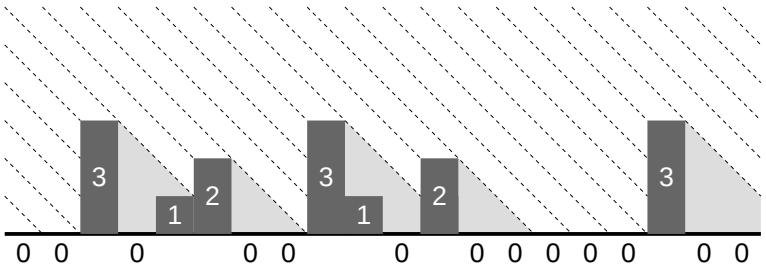
1. Il programma è stato consegnato in un unico file il cui nome coincide con il proprio numero di matricola?
2. È presente un commento iniziale che riporta cognome, nome, numero di matricola, gruppo (A/B) e indirizzo di posta (`@studio.unibo.it`) dell'autore?
3. Il programma è conforme allo standard 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*?

Camminando sotto la pioggia

La mappa di una città è rappresentata da una griglia rettangolare di caratteri di N righe e M colonne; la cella in alto a sinistra ha coordinate (0, 0), mentre quella in basso a destra ha coordinate (N - 1, M - 1). Nella griglia sono presenti unicamente i caratteri '0', … '9', dove '0' rappresenta un marciapiede e i caratteri $c \in$ {'1', ... '9'} indicano edifici alti c piani. Sta piovendo, e la pioggia cade da sinistra verso destra con un angolo di 45 gradi. Questo significa che le c caselle che eventualmente si trovano a destra di un palazzo alto c piani non vengono colpite dalla pioggia e quindi restano asciutte. La figura sottostante mostra con un esempio come va interpretata una ipotetica riga `0030120031020000300`:



Un pedone si trova nella casella (0, 0) e deve raggiungere la casella (N - 1, M - 1). Per farlo si può spostare dalla posizione corrente ad una delle quattro caselle adiacenti (in alto, in basso, a sinistra, a destra), purché la casella di destinazione non contenga edifici, cioè abbia etichetta '0' nel file di input. Si garantisce che la casella iniziale e quella finale siano sempre etichettate con '0'; tuttavia, potrebbe non esistere alcun cammino percorribile per raggiungere la destinazione (ad esempio, perché è circondata da palazzi).

Progettare e realizzare un algoritmo efficiente per determinare un cammino che consenta di raggiungere la destinazione attraversando il minor numero possibile di celle, se tale cammino esiste. Nel caso in cui esistano più cammini che attraversano lo stesso numero minimo di celle, il programma deve sceglierne uno che contenga il minor numero possibile di celle colpite dalla pioggia.

Solo per coloro che consegneranno il progetto entro la prima scadenza (per partecipare al primo appello scritto di entrambe le classi): in caso di più soluzioni ottime, cioè cammini che attraversano il minor numero possibile di caselle, non è richiesto di sceglierne una che contenga il minor numero di celle colpite da pioggia; in altre parole, la correttezza della soluzione verrà valutata solo in riferimento al primo valore stampato (si veda sotto). Il formato dell'output deve essere comunque lo stesso descritto nel seguito; in particolare, il secondo valore della prima riga deve comunque indicare il numero di celle colpite dalla pioggia che vengono attraversate. Tuttavia, nel caso in cui il programma non riceva valutazione positiva, a partire dalla consegna successiva non sarà più consentita questa semplificazione.

Il programma legge l'input da un file il cui nome (completo eventualmente di percorso) viene passato sulla riga di comando: ad esempio, in ambiente Linux/MacOSX:

```
./prog nome_file_input
```

oppure, in ambiente Windows:

```
.\prog nome_file_input
```

dove "prog" è il nome dell'eseguibile, e "nome_file_input" è il nome completo del file di input (che includerà l'eventuale percorso). Il programma deve stampare il risultato a video secondo il formato specificato nel seguito.

Input

Il file di input ha il seguente contenuto:

1. la prima riga contiene i valori di N (numero righe) e M (numero colonne) della matrice di input, separati da spazi o tab. Si garantisce che si avrà sempre $10 \leq N \leq 500$, $10 \leq M \leq 500$;

2. seguono *N* righe, ciascuna contenente *M* caratteri, che si garantisce saranno sempre cifre numeriche, ossia caratteri nell'insieme {'0', … '9'}.

Output

Al termine dell'esecuzione il programma deve produrre a video l'output seguente:

- Se la destinazione è raggiungibile, deve stampare due righe di testo:
  - la prima contiene il numero minimo *d* di caselle visitate per raggiungere la destinazione (incluse le caselle di origine e destinazione), e il numero *r* di caselle colpite da pioggia nel percorso minimo di cui sopra. Si dovrà sempre avere $r \leq d$;
  - la seconda riga contiene una sequenza di (*d* - 1) caratteri che indica gli spostamenti effettuati; gli unici caratteri ammessi, oltre all'eventuale "a capo" finale, sono le lettere maiuscole `E` (spostamento verso Est), `O` (spostamento verso Ovest), `S` (spostamento verso Sud), `N` (spostamento verso Nord).
- Se la destinazione non è raggiungibile, il programma deve stampare una unica riga di testo contenente i valori "`-1 -1`" (senza virgolette).

Possono esistere più cammini validi corrispondenti alla soluzione ottima. Di conseguenza, i valori *d* e *r* devono corrispondere alle soluzioni mostrate, ma i cammini stampati sulla seconda riga possono differire. In fase di correzione si verificherà che i cammini siano validi e attraversino il numero di celle indicato nella prima riga.

Esempi

Input:	*Output atteso:*

`10 10`
`0100202580`
`0100000006`
`0100111005`
`0100000607`
`0111101100`
`0000001100`
`2000220002`
`0111111902`
`0750001007`
`0334930000`
	`31 22`
	`SSSSSEEEEENNOONNEEEESESSSSSSSE`

Input:
 	
`10 20`
`00100202008000029795`
`06006008000005004020`
`06570009400010006087`
`00702000000100030400`
`08897067000001467033`
`49300000030350006029`
`76009103000408000030`
`00000000080473370600`
`70300090100900559000`
`70008700760019003710`
	Output atteso:
	
	`-1 -1`