POLITECNICO
MILANO 1863

Prova Finale di Algoritmi e Strutture Dati

Alcune indicazioni

Sezione prof. Pradella
Cognomi E - O
A.A. 2017/2018

Davide Piantella
Giuseppe Mascellaro

Rev. 08/06/2018

Riferimenti

Per quanto non specificato in queste indicazioni si rimanda agli altri documenti relativi a questo progetto:

- Note generali
- Presentazione del progetto e del verificatore
- Annunci all'interno del verificatore
- Annunci sul sito del corso

Chi contattare in caso di bisogno

Domande burocratiche

Prof. Pradella - tramite gli usuali canali

Domande di carattere generale sul progetto

Contattare il proprio tutor di riferimento tramite email:

- **Cognomi da E a LA:** Davide Piantella davide.piantella@mail.polimi.it
- **Cognomi da LE a O:** Giuseppe Mascellaro giuseppe.mascellaro@mail.polimi.it

Per abbreviare il più possibile i tempi di risposta, vi preghiamo di inserire nell'oggetto della email [ProvaFinaleAPI].

Domande specifiche sulla propria implementazione

Per questo genere di domande sono previsti 4 incontri di tutorato in luglio e settembre.
Per queste problematiche vi preghiamo di non contattarci per email in quanto una risposta esaustiva deve tener conto di fattori difficilmente valutabili non in presenza.
In generale vi preghiamo quindi di non inviarci per email il vostro codice sorgente.

Ambiente di sviluppo

Non ci sono vincoli che limitino la scelta dell'ambiente di sviluppo da utilizzare, tuttavia consigliamo una qualsiasi distribuzione Linux (se virtualizzata suggeriamo Xubuntu) per emulare al meglio il verificatore.
Sconsigliamo vivamente ambienti (IDE) strutturati e complessi.
Riteniamo che i tools mostrati durante il primo tutorato - e riportati nell'apposita sezione di questo documento - siano ideali per guidarvi nelle operazioni di debug e miglioramento del vostro progetto.
L'uso dei tools e in generale un attento debugging devono essere operazioni preliminari a qualsiasi richiesta di assistenza ai tutors.

Deadlines

Per i laureandi di luglio la scadenza è alle ore 24 del giorno 11 luglio.
Per tutti gli altri la scadenza è alle ore 24 del giorno 12 settembre.
Non sono previsti recuperi.

Specifiche e assunzioni

Input

Gli input vengono forniti da stdin.
Si assuma che la sequenza dei comandi di input sia sintatticamente corretta.
Il formato è il seguente

- tr
- *funzione di transizione*
- acc
- *stati finali (di accettazione)*
- max
- *numero massimo di passi*
- run
- *stringhe di input*

Funzione di transizione

Ogni riga della funzione è codificata seguendo l'espressione:
(il carattere [ ] indica un carattere spazio " ")
stato_partenza[ ]carattere_letto[ ]carattere_scritto[ ]movimento[ ]stato_arrivo

- I caratteri sono codificati come char
- Gli stati sono codificati come int
    - Lo stato iniziale è convenzionalmente lo stato 0
- Il carattere "_" indica il *blank*
- Il movimento della testina è codificato usando i caratteri
    - "L" Left
    - "R" Right
    - "S" Stop
- La funzione di trasferimento può non essere ordinata per stato di partenza
- Si assuma che se esiste lo stato N allora esistono anche gli stati N-1, N-2, …, 0

Stati finali

Gli stati finali (se presenti) sono elencati uno per riga.
Si assuma che non esistano archi uscenti dagli stati finali.

Numero massimo di passi

In caso di macchina non-deterministica questo parametro si riferisce al singolo percorso non-deterministico.

Stringhe di input

Non ci sono vincoli né sul numero delle stringhe di input né sulla loro lunghezza.
Ogni stringa è posta su una riga.

Configurazione iniziale

Macchina nello stato 0 con la testina posizionata sul primo carattere della stringa di ingresso, ovviamente inserita nel consueto nastro singolo sbiancato bi-infinito.

Output

L'output deve venire fornito su stdout.
Per ogni stringa di ingresso (un valore per riga):

- 1 se la stringa viene accettata
- 0 se la stringa non viene accettata
- U se la computazione non termina entro il numero massimo di passi specificato

Verificatore

Tasks, punteggio e **voto**

Il task di tutorial non contribuisce in alcun modo all'esito della prova.
Sono previsti 6 tasks. Ognuno di questi si compone di:

- 1 subtask pubblico - ovvero con input e output atteso pubblici (0 punti)
- 3 subtask privati di difficoltà crescente (rispettivamente 3 + 1 + 1 punti)

Il punteggio massimo ottenibile è 30 punti + 1 punto extra per la lode ottenibile superando un apposito task.
Per superare l'esame è necessario:

- Superare almeno tutti i subtasks pubblici e il subtask privato più semplice di ogni task
  - Non è sufficiente ottenere 18 punti se il vincolo di cui sopra non è soddisfatto
- Verrà valutata l'ultima implementazione sottomessa in ogni task
- L'implementazione valutata deve essere identica per tutti i tasks

Annullamento della prova

Verranno annullati tutti i progetti coinvolti in caso di:

- Copia (anche parziale)
- Distribuzione (con qualsiasi mezzo)
- Tentativi di manomissione della piattaforma di valutazione

Possibili messaggi di errore e relative soluzioni

Output is not correct

Possibili cause:

- L'implementazione è errata
- Parsing errato dell'input
- L'output è stampato nel formato sbagliato

Soluzioni:

- Testing in locale con subtask pubblico
- Prestare attenzione ad eventuali "\n" o stampate di debugging

Execution timed out - Execution killed, violating memory limits

Cause:

- L'implementazione non soddisfa i vincoli di tempo o di memoria
    - NB: Il valore di tempo e memoria eventualmente riportato non è indicativo

Soluzioni:

- Debugging locale
    - Valgrind, callgrind, kcachegrind per individuare
        - Funzioni dispendiose di tempo / memoria
        - Memory leaks

Execution killed with signal 11

Cause:

- L'implementazione non soddisfa i vincoli di memoria
- Potrebbe essersi verificato un crash dovuto a un errore di segmentazione

Soluzioni:

- Debugging locale
    - Valgrind, callgrind, kcachegrind per individuare
        - Funzioni dispendiose di tempo / memoria
        - Memory leaks
        - Segfault

Execution failed because the return code was nonzero

Cause:

- Il main non ritorna 0
- Comportamento inaspettato della propria implementazione

Soluzioni:

- Inserire return 0
- Debugging locale

Debugging

Pipelining

Di seguito due comandi utili per testare la propria implementazione, basandosi sui subtasks pubblici (input e relativo output atteso)

- cat ./input_pubblico | ./eseguibile > output
- diff ./output_pubblico ./output

Software e comandi utili

GCC

- gcc -ggdb *sorgente.c* -o *eseguibile*
- elenco completo dei flag visibile sul verificatore (“compilation commands”)

GDB

- gdb *eseguibile*
    - run [< file_di_input]
    - list [funzione | riga]
    - break [funzione | riga]
    - [x | print | explore] variabile
    - [continue | next | step]
    - backtrace
    - where

Valgrind

- valgrind *eseguibile*

Kcachegrind

A. valgrind --tool=callgrind *eseguibile*
B. kcachegrind *callgrind.out.xxxx*