

Prova Finale

Algoritmi e Strutture Dati

2017 / 2018

Tutors

Davide Piantella

davide.piantella@*mail*.polimi.it

cognomi da E a LA



Giuseppe Mascellaro

giuseppe.mascellaro@*mail*.polimi.it

cognomi da LE a O

Come contattarci

- Email con oggetto [ProvaFinaleAPI]

★ Tutorati

- Rispondere alle domande di persona
guardando il vostro lavoro ci permette di essere più chiari

Agenda

1. Informazioni pratiche sui tutorati
2. Struttura della prova e valutazione
3. Consigli pratici
4. Analisi delle specifiche del progetto
5. Demo del tool di valutazione
6. Hands on
 - C - gcc
 - gdb - valgrind - callgrind - kcache-grind

Tutorati

- Indicativamente 2 a Luglio e 2 a Settembre
- Modalità:
 - Risposte a domande individuali
 - Possiamo rispondere solo a domande tecniche
- Prerequisiti:
 - Algoritmi e strutture dati visti nel corso
 - Programmazione in C
 - Compilazione con GCC da terminale (*lo ripasseremo oggi*)
 - Debugging (*lo ripasseremo oggi*)

Struttura della prova

Implementazione in C corretta ed efficiente di un
Simulatore di Macchine di Turing non deterministiche

- Specifiche e dettagli online sul [sito del corso](#)
 - [Attenzione agli aggiornamenti!](#)
 - [Twitter](#)
 - Annunci sul [portale](#)
- Valutazione automatica online

Deadlines

- Laureandi luglio: 11 luglio ore 24
- **Tutti gli altri: 12 settembre ore 24**

Non sono previsti recuperi

Tool di valutazione

Come procedere:

- ★ Upload del codice sorgente
- ★ Compilazione e valutazione automatica
- ★ Valutazione su casi di test pubblici
- ★ Valutazione su casi di test privati
- ★ Esito

Tool di valutazione

- Tentativi illimitati (*use with caution!*)
- Casi di test per la valutazione
- Utilizzabili solo librerie C standard
- *Efficienza*: tempo di esecuzione e memoria occupata
- Lettura e scrittura da stdin e stdout
 - `cat input.txt | ./executable_file > output.txt`
- Singolo e unico file sorgente

Tips

- Non iniziare subito a scrivere codice
- Iniziare ad impostare la soluzione (*prima di settembre*)
- Pensare alle strutture dati per soddisfare le specifiche sia *funzionali* che *di complessità*
- Sfruttare il paradigma procedurale (divide et impera)
- Testare sempre in locale prima di caricare il codice (no brute force)
- Il codice deve essere leggibile e ben commentato
- Esecuzione sequenziale (no multithreading)
- Attenzione alle malloc (no data leaks)

Repetita iuvant

Non copiare

- Verranno eseguiti controlli sui sorgenti
- Tutti i progetti coinvolti verranno annullati
- Non condividere il proprio sorgente
 - NB: caricare il proprio sorgente su GitHub = condividere

Specifiche

Specifiche

- I simboli di nastro sono *char*, gli stati sono *int*
- Il carattere `_` indica il simbolo *blank*
- Lo stato 0 è lo stato iniziale
- Inizialmente la testina è sul primo carattere della stringa in ingresso
- Nastro sbiancato bi-infinito
- I caratteri *L*, *R*, *S* indicano il movimento della testina

Esempio: $L = \{ww \mid w \in \{a, b\}^*\}$

input.txt

tr

0 a C R 1

0 b C R 2

1 a a R 1

1 b b R 1

[...]

acc

11

max

180

run

aabaab

bbabbb

aaaaaaaaaaaaaaaaaaaaaaaaa[...]

[...]

output.txt

1

0

U

[...]

Casi di test

- Effettuati su centinaia di stringhe
- Stringhe di lunghezza arbitraria
- Verifica della correttezza dell'output
- Vincoli di tempo e memoria incrementali
- Test pubblici da intendersi di base

Risposte alle vostre domande

- Potete assumere che i file di input siano sintatticamente corretti e coerenti con le specifiche
- La funzione di transizione può non essere ordinata per numero di stato
- Non ci saranno archi uscenti da uno stato di accettazione
- Se esiste lo stato N esistono anche gli stati $N-1, N-2, \dots, 0$
- Non ci sono vincoli riguardo alla lunghezza del file di input e delle stringhe di input
- Il parametro U in caso di macchina non-deterministica si riferisce al singolo percorso non-deterministico

Demo verificatore

<http://dum-e.deib.polimi.it>

Hands on

C - gcc - gdb - valgrind - callgrind - kcachegrind

Alcuni comandi utili

- GCC

- `gcc -ggdb sorgente.c -o eseguibile`
- elenco completo dei flag visibile sul verificatore

- 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

- `valgrind --tool=callgrind eseguibile`
- `kcachegrind callgrind.out.xxxx`