Prova Finale di Algoritmi e Strutture Dati

note generali

Introduzione

- Obiettivo: implementazione efficiente (e corretta!) di un algoritmo
- Logistica
  - codice sorgente sarà caricato su un server, compilato e fatto girare automaticamente
  - consegna entro metà settembre
  - metteremo delle date precedenti per i laureandi
- Esecuzione del progetto
  - implementazione nel linguaggio C
  - **esclusivamente** con libreria standard (libc)

Valutazione

- programma deve compilare e girare correttamente
  - verranno resi disponibili dei casi di test come prova per controllare il corretto funzionamento del programma
  - dopo il caricamento sul server, il programma viene fatto girare su test, divisi in 2 parti: pubblici e privati
- si misura la correttezza (risultati in uscita) e l'efficienza (tempi di risposta e memoria occupata) del programma su vari casi di test
- dipendentemente dai risultati sui casi di test, potrete **calcolare** il voto
  - il voto è assegnato in modo **automatico** in base a come vanno i test
- non c'è recupero

Sito per la sottomissione del progetto

- https://dum-e.deib.polimi.it/
- Ogni studente avrà le proprie credenziali per accedere al sito

Operazioni che si possono fare sul sito

- scaricare le specifiche del progetto
- caricare (*submit / sottoponi*) il file da compilare e lanciare
  - si può usare il nome del file che si vuole, viene rinominato in automatico
  - dopo il caricamento, in automatico il file viene compilato e testato su un insieme di test “pubblici”
- lanciare (*play / usa*) i test “privati”, quelli su cui viene valutato il progetto

Gestione del codice

- Vi è **richiesto** di condividere il codice con il docente tramite GitHub (www.github.com)
- A questo fine occorre:
  - Registrarsi su github, usando l'email del Politecnico
    - l'email del Politecnico dà la possibilità di creare repository gratis
    - guardate questo link: https://education.github.com/pack
  - Creare un repository **privato**, chiamandolo "PFAPI_<cognome>_<idnumber>"
  - "Invitare" il docente a condividere il repository
    - l'id del docente è matteo-g-rossi
- Se già non avete dimestichezza con git, online ci sono diversi tutorial
  - ma le operazioni che dovete fare voi per questo progetto sono molto semplici, di fatto solo delle "push" di codice sul repository

Plagi

- progetto da svolgere singolarmente ed in totale autonomia (no a gruppi)
- siete responsabili del vostro codice
  - vi consigliamo fortemente di non passarlo per "ispirazione" a colleghi
- controllo plagi automatizzato
- in caso di copiatura **tutti** i progetti coinvolti vengono **annullati**

Scadenza

- Mercoledì 12 settembre, ore 23.59
  - poi il sito per la sottomissione verrà chiuso

- Per i laureandi di luglio: mercoledì 11 luglio, ore 23.59
  - avvisate (mandando un email al docente) che volete avere la valutazione a luglio

Calendario incontri tutoraggio

- Luogo: ufficio docente (ufficio 6, primo piano)
- venerdì 8/6, 14.00-17.00
  - questo sarà in aula B.1.3
- giovedì 28/6, 12.00-13.00 (mandare un email di preavviso)
- venerdì 6/7, 10.00-13.00
- venerdì 20/7, 10.00-13.00
- giovedì 30/8, 14.00-17.00

Un simulatore di Macchine di Turing non-deterministiche

Prova finale di Algoritmi e Strutture Dati AA 2017-18

Il progetto

- Implementazione in linguaggio C standard (con la sola *libc*) di un interprete di Macchine di Turing non-deterministiche, nella variante a **nastro singolo** e solo **accettori**.
- Struttura del file di ingresso: prima viene fornita la funzione di transizione, quindi gli stati di accettazione e un limite massimo sul numero di passi da effettuare per una singola computazione (per evitare in maniera banale il problema delle macchine che non terminano), infine una serie di stringhe da far leggere alla macchina.
- In uscita ci si attende un file contenente 0 per le stringhe non accettate e 1 per quelle accettate; essendoci anche un limite sul numero di passi, il risultato può anche essere *U* se non si è arrivati ad accettazione.

Convenzioni utili

- Per semplicità i simboli di nastro sono dei *char*, mentre gli stati sono *int*. Il carattere "_" indica il simbolo "blank".
- La macchina parte sempre dallo stato 0 e dal primo carattere della stringa in ingresso.
- Si assume, come al solito, che il nastro sia illimitato sia a sinistra che a destra e che contenga in ogni posizione il carattere "_".
- I caratteri "L", "R", "S" vengono usati, come al solito, per il movimento della testina.
- Il file di ingresso viene fornito tramite lo standard input, mentre quello in uscita è sullo standard output.

Struttura del file di ingresso

Il file di ingresso è diviso in 4 parti:

- La prima parte, che inizia con "tr", contiene le *transizioni*, una per linea - ogni carattere può essere separato dagli altri da spazi.
  Per es. *0 a c R 1* significa che si va dallo stato 0 allo stato 1, leggendo *a* e scrivendo *c*; la testina viene spostata a destra (*R*).
- La parte successiva, che inizia con "acc", elenca gli stati di *accettazione*, uno per linea.
- Per evitare problemi di computazioni infinite, nella sezione successiva, che inizia con "max", viene indicato il numero di mosse massimo che si possono fare per accettare una stringa.
- La parte finale, che inizia con "run", è un elenco di stringhe da fornire alla macchina, una per linea.

Esempio: MT non-det. per ww^R

```
tr
0 a a R 0
0 b b R 0
0 a c R 1
0 b c R 2
1 a c L 3
2 b c L 3
3 c c L 3
3 a c R 4
3 b c R 5
4 c c R 4
4 a c L 3
5 c c R 5
5 b c L 3
3 _ _ R 6
6 c c R 6
6 _ _ S 7
```

```
acc
7
max
800
run
aababbabaa
aababbabaaaababbabaa
aababbabaaaababbabaab
aababbabaaaababbabaabbaababbabaaaababbabaa
aababbabbaaababbabaabbaababbabaaaababbabaa
```

Standard output:

```
1
1
0
U
0
```

Esempio: MT per ww

tr

```
0 a c R 1
0 b c R 2
1 a a R 1
1 a d L 3
1 b b R 1
2 a a R 2
2 b b R 2
2 b d L 3
3 a a L 3
3 b b L 3
3 c c R 4
4 d d R 10
4 a c R 5
4 b c R 6
5 a a R 5
5 b b R 5
5 d d R 7
6 a a R 6
6 b b R 6
6 d d R 8
7 d d R 7
7 a d L 9
8 d d R 8
8 b d L 9
9 d d L 9
9 a a L 3
9 b b L 3
9 c c R 10
10 d d R 10
10 _ _ S 11
```

```
acc
11
max
2000
run
aabaab
bbabbb
ababa
babaaababaaa
ababaababababbabbbbbabbbbabaaababaaaababaabababbabbbbbba-
bbbbabaaababaaa
```

Standard output:

```
1
0
0
1
U
```

Breve tutorial del sito

Overview

- “Tutorial” è un semplicissimo problema, creato al solo scopo di sperimentare il funzionamento del sito
- Tutti gli altri sono i problemi da risolvere (i test che **devono** essere eseguiti con successo)

Task overview

Task	Name	Time limit	Memory limit	Type	Files	Tokens
IncreasingStuff	IncreasingStuff	17 seconds	4 MiB	Batch	IncreasingStuff[.c]	Yes
FancyLoops	FancyLoops	20 seconds	1 MiB	Batch	FancyLoops[.c]	Yes
MindYourLeft	MindYourLeft	20 seconds	1 MiB	Batch	MindYourLeft[.c]	Yes
UnionStuck	UnionStuck	21 seconds	1 MiB	Batch	UnionStuck[.c]	Yes
DontGetLost	DontGetLost	21 seconds	7 MiB	Batch	DontGetLost[.c]	Yes
ToCOrNotToC	ToCOrNotToC	19 seconds	20 MiB	Batch	ToCOrNotToC[.c]	Yes
CumLaude	CumLaude	16 seconds	16 MiB	Batch	CumLaude[.c]	Yes

“Tutorial”

- Problema: leggere da standard input 2 numeri interi separati da uno spazio; produrre su standard output la loro somma

Some details

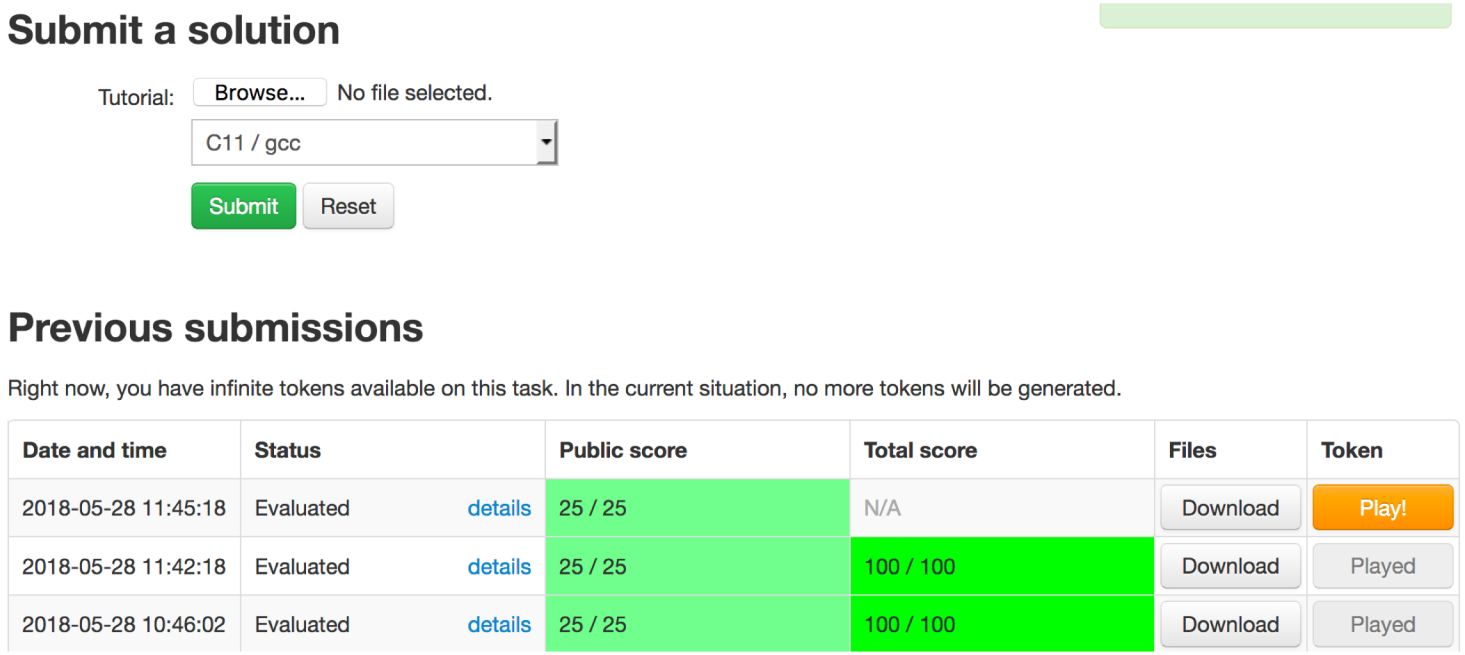
Type	Batch	
Time limit	1 second	
Memory limit	256 MiB	
Compilation commands	C11 / gcc	`/usr/bin/gcc -DEVAL -std=c11 -O2 -pipe -static -s -o Tutorial Tutorial.c -lm`
Tokens	You have an infinite number of tokens for this task.	

Passi da fare

- Step 1: creare il file Tutorial.c
- Step 2: compilare il file con la riga di comando (presa dal sito)
  `/usr/bin/gcc -DEVAL -static -std=c99 -O2 -o Tutorial Tutorial.c -lm`
- Step 3: preparate un file di test, per esempio Tutorial_test.in, con una riga di testo, in cui ci sono 2 interi separati da uno spazio; preparate anche un file Tutorial_test.out con il risultato atteso della computazione (una riga con il risultato della somma)
- Step 4: lanciate il comando
  `cat Tutorial_test.in | ./Tutorial > Tutorial_test.res`
  - crea un file, Tutorial_test.res, con il risultato della computazione

Passi da fare (cont.)

- Step 5: confrontate il risultato atteso con quello ottenuto mediante il comando `diff Tutorial_test.out Tutorial_test.res`
  - se non ci sono differenze il risultato è la stringa vuota
  - il server usa il comando *wdiff* invece del comando *diff*
- Step 6: caricate ("sottoponi") il file Tutorial.c sul sito, e lanciate ("usa") i test



Possibili strumenti utili

- valgrind (memory debugging, profiling)
  - http://valgrind.org
- gdb (debugging)
  - https://www.gnu.org/software/gdb/
- AddessSanitizer (ASan, memory error detector)
  - https://github.com/google/sanitizers/wiki/AddressSanitizer

Calcolo del voto

- Il task di tutorial non contribuisce in alcun modo all'esito della prova
- Sono previsti 6 task. 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 subtask 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 task