

Prova Finale di Algoritmi e Strutture Dati

A.A. 2021/2022

note generali

Introduzione

- Obiettivo: implementazione efficiente (e corretta!) di un algoritmo
- Logistica
 - codice sorgente sarà caricato su un server, compilato e fatto girare automaticamente
 - Scadenze (strette e vincolanti):
 - 4 luglio (ore 23.59) per i laureandi di luglio
 - 9 settembre (ore 23.59) per tutti gli altri
 - piattaforma riaperta tra il 31 gennaio e 11 febbraio 2023 (fino alle 23.59) per i SOLI laureandi di febbraio
 - Occorre essere iscritti all'appello di laurea perché il voto sia registrato
 - Tenete conto dei vincoli temporali per gestirvi!
 - Es. "inizio a pensarci a settembre" = fallimento quasi certo

Esecuzione del progetto

- Esecuzione del progetto
 - implementazione nel linguaggio C (C11, VLA ammessi)
 - esclusivamente con libreria standard (libc), niente librerie esterne
 - no multithreading o tecniche di parallelizzazione
 - dati in ingresso ricevuti via stdin, risultati da fornire via stdout

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
 - verrà fornito anche uno strumento di generazione automatica di casi di test, per facilitarvi il testing in locale
- 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, ma il numero di “appelli” (= sottomissioni) è praticamente illimitato

Sito per la sottomissione del progetto

- <https://dum-e.deib.polimi.it/>
- Ogni studente ha 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 privati gratis
 - guardate questo link: <https://education.github.com/pack>
 - Creare un repository **privato**, chiamandolo "PFAP122_<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, quindi vi consigliamo fortemente di:
 1. Non caricarlo in repository **pubblici**
 2. Non passarlo per "ispirazione" a colleghi
 3. Non utilizzate frammenti di codice reperito online
- controllo plagi automatizzato
- in caso di copiatura **tutti** i progetti coinvolti vengono **annullati**

Scadenze

- Venerdì 9 settembre, ore 23.59
 - poi il sito per la sottomissione verrà chiuso
- Per i laureandi di luglio: lunedì 4 luglio, ore 23.59
 - avvisate (mandando un email al docente) che volete avere la valutazione a luglio
- Per i laureandi di febbraio 2023: riapertura del sito da martedì 31 gennaio, fino a sabato 11 febbraio, ore 23.59
 - avvisate (mandando un email al docente) che volete avere la valutazione a febbraio
 - Perché il voto (se positivo) sia registrato, occorre essere iscritti all'appello di laurea di febbraio

Tutoraggio

- Possibili interazioni:
 - Webeep (**specifico per la Prova Finale!**)
 - Forum su Webeep: <https://webeep.polimi.it/mod/forum/view.php?id=14867>
 - Chat di Teams
 - Specifica per la Prova Finale: <https://bit.ly/3yX4jEx>
 - Email (per esempio per prendere appuntamento)
 - Tutoraggio in presenza (vedi slide successiva)

Tutoraggio in presenza

- Luogo: ufficio a Cremona TBD
- Quando: tipicamente in corrispondenza ad esami:
 - martedì 31/5, a fine lezione
 - giovedì 23/6, dopo esame (fin che serve)
 - martedì 12/7, dopo esame (fin che serve)
 - martedì 30/8, dopo esame (fin che serve)
- Siete invitati a mandare un messaggio per avvisare che avete bisogno di un incontro

WordChecker

- L'obiettivo del progetto di quest'anno è di realizzare un sistema che, al suo cuore, controlla la corrispondenza tra le lettere di 2 parole di ugual lunghezza.
- Le parole sono intese come sequenze di simboli che possono essere caratteri alfabetici minuscoli (a-z) o maiuscoli (A-Z), cifre numeriche (0-9), oppure i simboli - (trattino) e _ ("underscore")
 - esempio di parola di 20 simboli: djHD1af9fj7g__l-ssOP

WordChecker

- Il sistema legge da standard input una sequenza di informazioni e istruzioni, e produce delle stringhe in output a seconda dei casi.
- Più precisamente, il sistema legge:
 - un valore k , che indica la lunghezza delle parole
 - una sequenza (di lunghezza arbitraria) di parole, ognuna di lunghezza k , che costituisce l'insieme delle parole ammissibili
 - si dia pure per scontato che la sequenza di parole non contenga duplicati
- A quel punto, viene letta da standard input una sequenza di "partite", in cui l'inizio di ogni nuova partita è marcato dal comando (letto sempre da input) *+nuova_partita*

WordChecker

- Le sequenze di stringhe in input per ogni partita (successive al comando *+nuova_partita*) sono fatte nel seguente modo:
 - parola di riferimento (di lunghezza k caratteri)
 - si assuma che la parola di riferimento appartenga all'insieme di parole ammissibili
 - numero n massimo di parole da confrontare con la parola di riferimento
 - sequenza di parole (ognuna di k caratteri) da confrontare con la parola di riferimento
- Ogni tanto, nella sequenza di stringhe in input, può comparire il comando *+stampa_filtrate*, il cui effetto è spiegato in seguito
- Inoltre, sia durante una partita, che tra una partita e l'altra, possono comparire i comandi *+inserisci_inizio* e *+inserisci_fine* che racchiudono tra di loro una sequenza di nuove parole da aggiungere all'insieme delle parole ammissibili
 - le parole aggiunte sono anch'esse di lunghezza k , e si dà sempre per scontato che non ci siano parole duplicate (neanche rispetto alle parole già presenti nell'insieme di quelle ammissibili)

WordChecker

- Per ogni parola letta (che nel seguito indichiamo con p), da confrontare con la parola di riferimento (che nel seguito indichiamo con r), il programma scrive su standard output una sequenza di k caratteri fatta nella seguente maniera.
 - nel seguito, indichiamo con $p[1], p[2], \dots p[k]$ i caratteri della parola p , con $r[1], r[2], \dots r[k]$ quelli della parola r , e con $res[1], res[2], \dots res[k]$ quelli della sequenza stampata
- Per ogni $1 \leq i \leq k$, si ha che
 - $res[i]$ è il carattere + se l' i -esimo carattere di p è uguale all' i -esimo carattere di r
 - cioè se vale che $p[i] = r[i]$, quindi $p[i]$ è "in posizione corretta"
 - $res[i]$ è il carattere / se l' i -esimo carattere di p non compare da nessuna parte in r
 - $res[i]$ è il carattere | se l' i -esimo carattere di p (indicato nel seguito come $p[i]$) compare in r , ma non in posizione i -esima; tuttavia, se in r compaiono n_i istanze di $p[i]$, se c_i è il numero di istanze del carattere $p[i]$ che sono in posizione corretta (chiaramente deve valere che $c_i \leq n_i$) e se ci sono prima del carattere i -esimo in p almeno $n_i - c_i$ caratteri uguali a $p[i]$ che sono in posizione scorretta, allora $res[i]$ deve essere / invece che |

WordChecker

- Per esempio, se
 $r = \text{abcabcabcabc}$
(quindi $r[1] = a$, $r[2] = b$, ecc.) e
 $p = \text{bbaabccbccbcabc}$
(con $p[1] = b$, $p[2] = b$, ...), abbiamo che res risulta la seguente sequenza:

```
abcabcabcabcabc
bbaabccbccbcabc
/+|+++|++/+++++
```

- si noti che $res[1] = /$ perché in r ci sono solo 5 b, p ne ha 6, e tutte le b successive a $p[1]$ sono nel posto corretto
- similmente, $res[10] = /$ perché r ha 5 c, p ne ha 6, di cui 4 al posto giusto, e c'è già una c prima di $p[10]$ (in $p[7]$) che è al posto sbagliato

WordChecker

- Altri esempi di confronti (dove la prima riga è la parola di riferimento r , la seconda è p , e la terza è l'output res)
 - djPDi939-s__e-s
gioSON-we2_w234
/|/////|/|/+//|/
 - djPDi939-s__e-s
kiidsa92KFaa94-
/|/||/|/////|/|
 - djPDi939-s__e-s
ewi-n4wp-sesr-v
|/|/////++/|/+/
 - DIk834k249kaoe_
48kDkkkf-saancd
||+||/+////+////

WordChecker

- Se da standard input viene letta una parola che non appartiene all'insieme di quelle ammissibili, il programma scrive su standard output la stringa *not_exists*
- Se invece viene letta la parola r (cioè se $p = r$), allora il programma scrive *ok* (senza stampare il risultato dettagliato del confronto) e la partita termina
- Se, dopo avere letto n parole ammissibili (con n , si ricordi, numero massimo di parole da confrontare con r), nessuna di queste era uguale a r , il programma scrive *ko* (dopo avere stampato il risultato del confronto dell'ultima parola), e la partita termina
- Dopo che la partita è finita:
 - Non ci possono essere altre parole da confrontare (ma ci potrebbe essere l'inserimento di nuove parole ammissibili)
 - Se in input c'è il comando *+nuova_partita*, ha inizio una nuova partita

WordChecker

- Ogni confronto tra p e r produce dei vincoli appresi dal confronto
- Per esempio, dal seguente confronto
abcabcabcabcabc
bbaabccbccbcabc
/+|+++|++/+++++
si apprende che b è in posizioni 2, 5, 8, 11, 14, che ci sono solo 5 b in r (la sesta b dà luogo a /), che c è in posizioni 6, 9, 12, 15, che non è in posizioni 7 e 10, che ci sono solo 5 c (come prima, la sesta dà luogo a /), che a è in posizioni 4 e 13, ma non è in posizione 3
- In maniera analoga, dal seguente confronto
djPDi939-s__e-s
gioSON-we2_w234
7|/////|/|/±//|/
si apprende che in r non ci sono g , né o , né S , che in r c'è almeno una i e che questa non è in posizione i , che c'è almeno un $-$ e che non è in posizione 7, ecc.

WordChecker

- Quando, durante una partita, da input si legge il comando *+stampa_filtrate*, il programma deve produrre in output, in **ordine lessicografico**, l'insieme delle parole ammissibili che sono compatibili con i vincoli appresi fino a quel momento nella partita, scritte una per riga
 - si noti che i vincoli appresi riguardano, per ogni simbolo:
 1. se il simbolo non appartiene a *r*
 2. posti in cui quel simbolo deve comparire in *r*
 3. posti in cui quel simbolo non può comparire in *r*
 4. numero *minimo* di volte che il simbolo compare in *r*
 5. numero *esatto* di volte che il simbolo compare in *r*
 - si noti che il vincolo 5 è più forte del vincolo 4
 - l'ordine dei simboli (usato per stabilire l'ordine lessicografico delle parole) è quello specificato dallo standard ASCII
- Inoltre, dopo ogni confronto, il programma deve stampare in output il numero di parole ammissibili ancora compatibili con i vincoli appresi
 - Tranne nel caso in cui la parola da confrontare non sia ammissibile

Un'esecuzione d'esempio

Input ricevuto

```
5
8adfs
5sjaH
KS06l
Hi23a
1aj74
-s9k0
sm_ks
okauE
+nuova_partita
5sjaH
4
KS06l
had7s
okauE
```

Commenti e Output Atteso

Le parole sono tutte di lunghezza 5 simboli

Elenco di parole ammissibili

Inizio nuova partita

Parola di riferimento

numero massimo di parole da confrontare in questa partita

Output (su 2 righe, in colonna): /////
5

Output (non si stampa il numero di parole compatibili con i vincoli): not_exists

Output: //|///
3

Un'esecuzione d'esempio

Input ricevuto

```
+stampa_filtrate
+inserisci_inizio
PsjW5
asHdd
paF7s
+inserisci_fine
-s9k0
sghks
+stampa_filtrate
sm_ks
+inserisci_inizio
_fah-
0D7dj
+inserisci_fine
```

Commenti e Output Atteso

Output (in colonna): 5sjaH, 8adfs, Hi23a

Nuove parole da aggiungere a quelle ammissibili (e quelle che sono compatibili con i vincoli appresi sono da aggiungere all'insieme delle parole compatibili con i vincoli appresi)

Output: /+///, 2

Output: not_exists

Output (in colonna): 5sjaH, asHdd

Output: |////, 2

Output subito dopo mappa confronto (raggiunto numero massimo parole): ko

Nuove parole da aggiungere a quelle ammissibili

Un'esecuzione d'esempio

Input ricevuto

```
+nuova_partita  
okauE  
3  
laj74  
+stampa_filtrate  
sm_ks  
okauE
```

Commenti e Output Atteso

```
Inizio nuova partita  
Parola di riferimento  
numero massimo di parole da confrontare in questa partita  
Output: /|///, 4  
Output (in colonna): Hi23a, _fah-, asHdd, okauE  
Output: ///|/, 1  
Output: ok
```

Breve tutorial del sito

Overview

- “Tutorial” è un semplicissimo problema, creato al solo scopo di sperimentare il funzionamento del sito
- "Open" è il task con i test pubblici, e con il codice per la generazione automatica dei test
- 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 |
|----------|----------|---------------|--------------|-------|--------------|--------|
| Tutorial | Tutorial | N/A | N/A | Batch | Tutorial[.c] | Yes |
| Open | Open | N/A | N/A | Batch | Open[.c] | Yes |
| UpTo18 | UpTo18 | 2.400 seconds | 90.0 MiB | Batch | UpTo18[.c] | Yes |
| UpTo30 | UpTo30 | 2.080 seconds | 82.0 MiB | Batch | UpTo30[.c] | Yes |
| CumLaude | CumLaude | 1.700 seconds | 1.00 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 | |
| Compilation commands | C11 / gcc | <pre>/usr/bin/gcc -DEVAL -Wall -Werror -std=gnu11 -O2 -pipe -static -s -o Open Open.c -lm</pre> |

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 -Wall -Werror -std=gnu11 -O2 -pipe -static -s -o Tutorial Tutorial.c -lm`
- Step 3: preparare un file di test, per esempio `Tutorial_test.in`, con una riga di testo, in cui ci sono 2 interi separati da uno spazio; preparare 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

Submit a solution

Tutorial: no file selected

Previous submissions

Right now, you have infinite tokens available on this task. In the current situation, no more tokens will be generated.

| Time | Status | Score | Files | Token |
|-------------|-----------------------------------|---|---------------------------------------|-------|
| 12:40:06 PM | Evaluated details | <input type="button" value="Download"/> | <input type="button" value="Played"/> | |

Possibili strumenti utili

- valgrind (memory debugging, profiling)
 - <http://valgrind.org>
- gdb (debugging)
 - <https://www.gnu.org/software/gdb/>
- AddressSanitizer (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
- La valutazione è immediatamente calcolata (e subito visibile), mediante 3 batterie di test (task):
 - il primo task, *UpTo18*, vale 18 punti (**pass** or **fail**)
 - il secondo task, *UpTo30*, vale fino a 12 (6 test da 2 punti ognuno)
 - l'ultimo task, *CumLaude*, vale 1 punto per la lode
- Il punteggio massimo ottenibile è 30 punti + 1 punto extra per la lode ottenibile superando il task *CumLaude*
 - il punteggio è dato dalla somma dei punti dei test superati dei primi 2 task
 - se tutti i test sono superati, inclusi i test per la lode, 30 e lode
 - Il punto della lode viene assegnato solo se tutti gli altri test sono superati

Avvertenze

- Nessun limite al numero di sottoposizioni, né penalità per sottoposizioni multiple
- È possibile migliorare la valutazione quante volte si desidera
- Viene valutata **l'ultima** sottoposizione fatta ad ogni batteria di test
- Tutte le sottoposizioni valutate devono utilizzare **lo stesso sorgente**
 - se siete in dubbio, ri-sottoponete lo stesso sorgente a tutte le batterie di test per buona misura