

Prova finale di algoritmi e strutture dati

Obiettivi didattici e realizzazione

- Obiettivi
 - Applicazione pratica delle tecniche apprese nel modulo di algoritmi e strutture dati del corso di algoritmi e principi dell'informatica
 - Implementazione di una soluzione ad un problema prestando attenzione ad aspetti concreti di efficienza del codice
- Realizzazione
 - Linguaggio C (C11, VLA ammessi)
 - Nessuna libreria esterna al di là della libreria standard C
 - No multithreading
 - Dati in ingresso ricevuti via stdin, risultati da fornire via stdout

Modalità di realizzazione

- Il progetto è **strettamente individuale**
 - Non utilizzate **alcun frammento di codice altrui**
- Siete responsabili del vostro codice
 - Non caricatelo su repository pubblici
 - Non condividetelo con colleghi per "prendere ispirazione"
 - Non utilizzate alcun frammento di codice reperito
- In caso di plagio o uso di codice altrui, **tutti i progetti coinvolti** saranno annullati

Criteri di valutazione

- La correttezza e l'efficienza della soluzione proposta sono valutate con batterie di test automatizzate
- Verranno forniti input/output d'esempio per poter collaudare la soluzione in locale
 - Non sottoponete soluzioni senza aver verificato che funzionino localmente
 - Verrà fornito anche uno strumento di generazione automatica di casi di test (input/output), per facilitarvi il testing in locale
- Il sistema di verifica calcola il tempo macchina e la memoria utilizzati
- La valutazione è immediatamente calcolata (e subito visibile), mediante 3 batterie di test:
 - la prima vale 18 punti (*pass or fail*)
 - la seconda fino a 12 (6 test da 2 punti ognuno)
 - l'ultima per la lode

Criteri di valutazione

- Nessun limite al numero di sottoposizioni, né penalità per sottoposizioni multiple
- È possibile migliorare la valutazione quante volte si desidera
- **Avvertenza:** 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
- Verificatore disponibile al <https://dum-e.deib.polimi.it>
- Credenziali di accesso via mail polimi

Scadenze e pianificazione

- Per i laureandi di luglio
 - 4 luglio, ore 23.59 CEST. Segnalate (email al docente) la necessità di valutazione
- Per tutti gli altri
 - 9 settembre, ore 23.59 CEST, dopo di che la piattaforma verrà **chiusa**
- Per laureandi di gennaio/febbraio (SUPERATO 145 CFU (di qualsiasi genere) + essere iscritto all'esame di laurea)
 - la piattaforma sarà riaperta indicativamente tra il 31 gennaio 2023 e l'11 febbraio 2023, fino alle ore 23.59 CEST
- Iniziare a lavorare ad una settimana dalla scadenza è uno dei modi migliori per **non** riuscire a superare la prova

Tutoraggio

- Sezione A-D (Barenghi):
 - Susanna Sidoti (susanna.sidoti@mail.polimi.it)
 - Paolo Daolio (paolo.daolio@mail.polimi.it)
- Sezione E-O (Pradella):
 - Mauro Famà (mauro.fama@mail.polimi.it)
 - Luisa Cicolini (luisa.cicolini@mail.polimi.it)
- Sezione P-Z (Martinenghi):
 - Simone Cattaneo (simone6.cattaneo@mail.polimi.it)
 - Marco Tramontini (marco.tramontini@mail.polimi.it)

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 non c'è bisogno di controllare siano di lunghezza k o che siano duplicati
 - 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) non c'è bisogno di controllare che parola sia tra quelle inserite all'inizio
 - 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 non c'è bisogno di controllare
- 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

da qui devo iniziare a controllare che le parole date in input siano conformi all'alfabeto

- 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:

abcabcabcabc
bbaabccbccbcabc
/+ | +++ | ++ /+++++

- si noti che $\text{res}[1] = /$ 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, $\text{res}[10] = /$ 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

/|/////|/|/+//|/

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* la 1 è caso speciale della 3 dove il simbolo non può comparire in alcun posto
 - 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 di un confronto con esito "not_exists"

Un'esecuzione d'esempio

Input ricevuto

5
8adfs
5sjaH
KS06l
Hi23a
laj74
-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: 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