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
  - Scadenze (strette e vincolanti)
    - 10 luglio (ore 24) per i laureandi di luglio
    - 12 settembre (ore 24) per tutti gli altri
    - **Tenete conto dei vincoli temporali per gestirvi!** (Es. “inizio a pensarci a Settembre” = fallimento quasi certo)
- Esecuzione del progetto
  - implementazione nel linguaggio C
  - esclusivamente con libreria standard (libc)
  - no thread o tecniche di parallelizzazione

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**
- 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 avrà le proprie credenziali per accedere al sito
- Il sistema di sottomissione verrà presentato esaustivamente nel primo incontro coi tutor il 4 Giugno alle 9:30 in aula III.B

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 (es. GitHub)
  2. Non passarlo per "ispirazione" a colleghi
- controllo plagi automatizzato
- in caso di copiatura tutti i progetti coinvolti vengono annullati

Un sistema di monitoraggio di relazioni tra elementi

Prova finale di Algoritmi e Strutture Dati AA 2018-19

Il tema

- Si vuole implementare un meccanismo di monitoraggio di relazioni tra entità (per esempio persone) che cambiano nel tempo
- Si immagini, per esempio, un social network, in cui nuovi utenti possono registrarsi, e utenti esistenti possono cancellare il proprio account, diventare “amici” di altri utenti, rompere la relazione di amicizia, ecc.
- Le relazioni tra entità non sono necessariamente simmetriche. Per esempio, Alice può essere “amica” di Bruno, ma l’amicizia non è reciprocata (Bruno non è amico di Alice)

- In maniera più astratta, il meccanismo monitora i seguenti fenomeni:
  - Una nuova entità comincia ad essere monitorata
  - Una entità monitorata smette di esserlo
  - Una nuova relazione viene stabilita tra 2 entità monitorate
  - Una relazione esistente tra 2 entità monitorate cessa di esistere
- Ogni entità ha un nome identificativo (per esempio "Alice", "Bruno", "Carlo")
- Ci possono essere diversi tipi di relazioni tra entità, ognuna identificata da un nome (per esempio, "amico_di", "segue", "coetaneo_di")
- Ogni relazione ha un verso (per esempio, se Alice è "amico_di" Bruno, il verso della relazione è da Alice a Bruno, quindi Bruno è il “ricevente” della relazione), e non è necessariamente simmetrica
- A seguito di un apposito comando, il sistema restituisce, per ogni relazione, l’entità che “riceve” più relazioni (se ci sono più entità il cui numero di relazioni ricevute è massimo, queste vengono stampate in ordine crescente di identificativo)
- L’applicativo dovrà essere ottimizzato per gestire un grande numero di entità e istanze di relazioni, ma generalmente pochi tipi (identificativi) di relazione

Il progetto

- Implementazione in linguaggio C standard (con la sola *libc*) di un programma che legge da *standard input* una sequenza di comandi, ognuno corrispondente ad un cambiamento nelle entità o nelle relazioni tra entità e, quando richiesto, produce su *standard output*, per ogni tipo di relazione monitorata, l'identificativo dell'entità che è il ricevente del maggior numero di istanze di quella relazione, e il numero di relazioni che l'entità riceve

Comandi

- I comandi che possono essere letti sono i seguenti:
  - addent ⟨id_ent⟩
    - aggiunge un'entità identificata da "id_ent" all'insieme delle entità monitorate; se l'entità è già monitorata, non fa nulla
  - delent ⟨id_ent⟩
    - elimina l'entità identificata da "id_ent" dall'insieme delle entità monitorate; elimina tutte le relazioni di cui "id_ent" fa parte (sia come origine, che come destinazione)
  - addrel ⟨id_orig⟩ ⟨id_dest⟩ ⟨id_rel⟩
    - aggiunge una relazione – identificata da "id_rel" – tra le entità "id_orig" e "id_dest", in cui "id_dest" è il ricevente della relazione. Se la relazione tra "id_orig" e "id_dest" già esiste, o se almeno una delle entità non è monitorata, non fa nulla. Il monitoraggio del tipo di relazione "id_rel" inizia implicitamente con il primo comando "addrel" che la riguarda.
  - delrel ⟨id_orig⟩ ⟨id_dest⟩ ⟨id_rel⟩
    - elimina la relazione identificata da "id_rel" tra le entità "id_orig" e "id_dest" (laddove "id_dest" è il ricevente della relazione); se non c'è relazione "id_rel" tra "id_orig" e "id_dest" (con "id_dest" come ricevente), non fa nulla
  - report
    - emette in output l’elenco delle relazioni, riportando per ciascuna le entità con il maggior numero di relazioni entranti, come spiegato in seguito
  - end
    - termine della sequenza di comandi

Osservazioni

- Gli identificativi (sia di entità che di relazione) sono sempre racchiusi tra ""
- Si assuma pure che ogni identificativo possa contenere solo lettere (maiuscole o minuscole), cifre, ed i simboli "_" e "-"
  - non serve controllare che gli identificativi ricevuti rispettino questa convenzione, la si può dare per scontata
- Tutti gli identificativi (sia delle entità che delle relazioni) sono "case sensitive", per cui "Alice" e "alice" sono identificativi diversi

Osservazioni (cont.)

- L’output del comando *report* è una sequenza fatta nel modo seguente: ⟨id_rel1⟩ ⟨id_ent1⟩ ⟨n_rel1⟩; ⟨id_rel2⟩ ⟨id_ent2⟩ ⟨n_rel2⟩; …
  - le relazioni in output sono ordinate in ordine crescente di identificativo
  - se per un tipo di relazione ci sono più entità che sono riceventi del numero massimo di relazioni, queste vengono prodotte in ordine crescente di identificativo, per esempio: ⟨id_rel1⟩ ⟨id_ent1_1⟩ ⟨id_ent1_2⟩ ⟨id_ent1_3⟩ … ⟨n_rel1⟩;
  - se vengono rimosse tutte le relazioni con un certo identificatore, esso non compare nei successivi output del comando *report*
  - se non ci sono relazioni tra le entità, l'output è *none* (senza virgolette)
- L'ordinamento degli identificativi segue la tabella dei caratteri ASCII, per cui vale il seguente ordine: - < 1 < A < _ < a
- Le varie parti di ogni comando e di ogni sequenza di output sono separate da spazi
- Il comando di *end* non ha output

Esempio di sequenza di comandi di input, con corrispondente output

input	output
addent "alice"	
addent "bruno"	
addent "carlo"	
addent "dario"	
report	none
addrel "carlo" "bruno" "amico_di"	
report	"amico_di" "bruno" 1;
addrel "carlo" "alice" "amico_di"	
report	"amico_di" "alice" "bruno" 1;
addrel "alice" "bruno" "amico_di"	
report	"amico_di" "bruno" 2;

Esempio di sequenza di comandi di input, con corrispondente output (cont.)

input	output
addrel "bruno" "dario" "compagno_di"	
report	"amico_di" "bruno" 2; "compagno_di" "dario" 1;
delrel "carlo" "alice" "amico_di"	
report	"amico_di" "bruno" 2; "compagno_di" "dario" 1;
addrel "carlo" "alice" "compagno_di"	
report	"amico_di" "bruno" 2; "compagno_di" "alice" "dario" 1;
addrel "carlo" "bruno" "compagno_di"	
report	"amico_di" "bruno" 2; "compagno_di" "alice" "bruno" "dario" 1;
delent "alice"	
report	"amico_di" "bruno" 1; "compagno_di" "bruno" "dario" 1;
end	