Corso di Programmazione e Algoritmi 2
Modulo di Laboratorio
A.A. 2021/2022

Regole per lo svolgimento del progetto:

- Il progetto può essere svolto **singolarmente** o **in gruppi di qualsiasi dimensione**.
  In ogni caso, ciascuno studente dovrà poi **singolarmente** discutere il progetto dimostrando di essere in grado di modificare e/o aggiungere alcune parti. In caso di esito sufficiente di tale discussione, **verrà attribuito il punteggio da 0 a 3 punti** che è additivo rispetto al voto della parte da 6 CFU di teoria.
- Il progetto deve essere consegnato **almeno 7 giorni prima** della prova orale di *Programmazione e Algoritmi 2* relativa all'appello in cui si desidera discutere il progetto. Non è necessario che la restante parte dell'esame (da 6 CFU) sia svolta nel medesimo appello.
- Il progetto deve contenere le classi pubbliche e i metodi richiesti **rispettandone esattamente il nome, il tipo e l'ordine dei parametri formali, ed il tipo di ritorno**. Si tenga presente che può essere necessario sviluppare anche altre classi (non pubbliche) oltre quelle richieste.
- Si tenga presente che **non sono sempre presenti** tutti i modificatori (***public, private, protected, static, final, …***) che devono essere opportunamente aggiunti da voi nella consegna.
- I **metodi di accesso e modifica** dei vari parametri non sono tutti presenti nella specifica. Deve essere vostra cura aggiungerli, dove occorrono, in modo opportuno, così come i metodi **equals** e **hashCode**.
- Si è naturalmente liberi di sviluppare (e anzi siete incoraggiati a farlo) metodi aggiuntivi di supporto ai metodi richiesti, laddove lo si ritenga utile.
- Deve essere consegnato via email all'indirizzo **luca.moscardelli@unich.it** un file compresso contenente tutti i file sorgenti e avente come nome
  [java] progetto 2021_2022.zip
  l'email deve avere come oggetto
  [java] consegna progetto programmazione 2021/2022
  e deve contenere nel testo la lista (Cognome, Nome, Matricola e Email) di tutti gli studenti che hanno svolto il progetto consegnato.
  In caso di consegna in gruppo, la consegna deve essere effettuata da uno dei componenti del gruppo, mettendo in CC nell'email anche l'indirizzo di tutti gli altri componenti.
- Tutte le classi pubbliche devono sovrascrivere il metodo **`toString()`** in modo opportuno
- Non è necessario consegnare il metodo **`main`** (che voi potete chiaramente utilizzare per i vostri test)
- Le uniche librerie del Java che potete usare sono **`java.util.ArrayList`** e **`java.util.HashMap`** (oltre naturalmente alle classi del package **`java.lang`**)

Testo del progetto:

Sviluppare in **Java** un package **cleii.turing** per la rappresentazione ed esecuzione di programmi tramite *Macchine di Turing*.
Il seguente testo è adattato da https://it.wikipedia.org/wiki/Macchina_di_Turing.
Una Macchina di Turing (**MdT**) agisce sopra un **nastro** che si presenta come una sequenza potenzialmente infinita di caselle nelle quali possono essere registrati simboli di un **alfabeto** finito; essa è dotata di una **testina** di lettura e scrittura (I/O) con cui è in grado di effettuare operazioni di lettura e scrittura su una casella del nastro. La macchina si evolve nel tempo e ad ogni istante si può trovare in uno **stato** interno facente parte di un insieme finito di stati. Inizialmente sul nastro viene posta una stringa di **input** che rappresenta i dati che caratterizzano il problema che viene sottoposto alla macchina. La macchina è dotata anche di un repertorio finito di **istruzioni** che determinano la sua **evoluzione** in conseguenza dei dati iniziali. L'evoluzione si sviluppa per passi successivi che corrispondono a una sequenza discreta di istanti successivi. Il nastro è potenzialmente infinito, cioè estendibile quanto si vuole, in entrambe le direzioni, qualora questo si renda necessario.
Ogni **passo dell'evoluzione** viene determinato dallo stato attuale s nel quale la macchina si trova e dal carattere a che la testina di I/O trova sulla casella del nastro su cui è posizionata e si concretizza nell'eventuale modifica del contenuto della casella, nell'eventuale spostamento della testina di una posizione verso destra o verso sinistra e nell'eventuale cambiamento dello stato. L'istruzione, che supponiamo unica, che ha come prime due componenti s e a, determina quali azioni vengono effettuate a ogni passo; le altre tre componenti dell'istruzione forniscono nell'ordine il nuovo stato, il nuovo carattere e, eventualmente, una richiesta di spostamento verso sinistra o verso destra.
Un'evoluzione della macchina consiste in una sequenza di sue possibili **configurazioni**. Ogni configurazione è costituita dallo *stato attuale*, dal *contenuto del nastro* (una stringa di lunghezza finita) e dalla *posizione sul nastro della testina* di I/O. L'evoluzione si arresta quando non esiste nessuna istruzione in grado di farla proseguire. Si noti che non è detto che l'arresto avvenga in uno stato finale. Inoltre, se l'arresto avviene in uno stato finale, quello che si trova scritto sul nastro all'atto dell'arresto rappresenta il risultato dell'elaborazione. Infine, può anche accadere che un'evoluzione non abbia mai fine; per questo motivo, in questo progetto, limitiamo il numero di passi di ogni possibile esecuzione a un parametro intero deciso nel momento della costruzione di una MdT.

Impostazione formale

Si definisce *Macchina di Turing deterministica a un nastro e istruzioni a cinque campi*, una sestupla $MdT = \langle S, s_0, F, A, \beta, \delta \rangle$, dove

- S è l'insieme degli stati della MdT;
- $s_0 \in S$ è lo stato iniziale della MdT;
- $F \subseteq S$ è l'insieme degli stati finali della MdT;
- A è l'alfabeto del nastro della MdT;

- $\beta \in A$ è il segno di casella vuota del nastro della MdT;
- $\delta: (S \setminus F) \times A \to S \times A \times \{-1,0,1\}$ è la funzione (parziale) di transizione della macchina. Se $\delta(s,a) = (t,b,m)$, la corrispondente quintupla $\langle s,a,t,b,m \rangle$ corrisponde all'istruzione che viene eseguita quando la macchina si trova nello stato s e la testina legge sul nastro, nella casella sulla quale è posizionata, il simbolo a; essa comporta la transizione allo stato t, la scrittura del carattere b al posto di a (se $b = \beta$ il carattere a viene cancellato) e, se $m \neq 0$, lo spostamento della testina (verso sinistra quanto $m = -1$ oppure verso destra quando $m = 1$).

Il package deve contenere le seguenti **classi pubbliche**:

- `Nastro`
  Classe pubblica che rappresenta il nastro di una MdT.
  Si tenga conto che il nastro è potenzialmente infinito sia nella direzione delle posizioni negative che in quella delle posizioni positive. Si *consiglia* di implementare tale classe tramite due ArrayList (uno per le posizioni non negative ed un altro per quelle negative), oppure con una HashMap.
  Contiene i seguenti metodi:
  - Costruttore
    `public Nastro(String s, char beta)`
    che costruisce un nastro in cui la stringa `s` sia scritta a partire dalla posizione 0 e in cui il carattere che denota la casella vuota è `beta`;
  - Costruttore
    `public Nastro(Nastro n)`
    che costruisce un nastro che sia il clone di n (in cui tutte le strutture contenute in n sono a loro volta clonate).
  - `public char get(int i)`
    che restituisce il carattere in posizione `i` (che può essere qualsiasi intero negativo, nullo o positivo)
  - `public void set(int i, char c)`
    che scrive in posizione `i` il carattere `c`. `i` può essere qualsiasi intero negativo, nullo o positivo.
  - Il metodo `toString` deve restituire la stringa ottenuta mettendo nell'ordine tutti i caratteri (compresi i caratteri di casella vuota) a partire dalla posizione del nastro, di indice più piccolo, che non contiene un carattere di casella vuota per finire alla casella alla posizione del nastro, di indice più grande, che non contiene un carattere di casella vuota.

- `Stato` e la sua sottoclasse `StatoFinale`
  Classe pubblica che rappresenta lo stato (**oggetti immutabili**).
  Contiene i seguenti metodi:
  - Costruttori
    `public Stato(String nome)`

`public StatoFinale(String nome)`
che costruisce uno stato (normale o finale) con il nome passato come parametro.
- `public String getNome()`
che restituisce il nome dello stato.

- `Istruzione`
Classe pubblica che rappresenta una istruzione della MdT (**oggetti immutabili**).
Contiene i seguenti metodi:
  - Costruttore
  `public Istruzione(Stato s, char a, Stato t, char b, byte m)`
  che costruisce una istruzione corrispondente alla quintupla $\langle s, a, t, b, m \rangle$.

- `Configurazione`
Classe pubblica che rappresenta una configurazione della MdT.
Contiene i seguenti metodi:
  - Costruttore
  `public Configurazione(Stato s, Nastro n, int posTestina)`
  che costruisce una configurazione data dallo stato `s`, da **un clone del** nastro `n` e dalla posizione corrente della testina `posTestina`.
  - Metodi pubblici di accesso `getStato(), getNastro(), getPosTestina()`.

- `MacchinaDiTuring`
Classe pubblica che rappresenta una MdT.
Contiene i seguenti metodi:
  - Costruttore
  `public MacchinaDiTuring (String input, char beta, ArrayList<Stato> stati, Stato statoIniziale, int maxLunghezzaEvoluzione, int posInizialeTestina)`
  che crea una MdT in cui *l'alfabeto è implicitamente dato da tutti i possibili caratteri del tipo primitivo char* (che non devono essere chiaramente tutti necessariamente utilizzati) e tale che
    - la stringa `input` sia scritta sul nastro a partire dalla posizione 0;
    - `beta` è il carattere che denota la casella vuota;
    - `stati` è un ArrayList contenente tutti gli stati (compresi quelli finali) della MdT;
    - `statoIniziale` è lo stato iniziale (qualora non sia già presente in `stati` deve comunuque essere considerato come uno stato della MdT);
    - `maxLunghezzaEvoluzione` è il massimo numero di passi di una esecuzione della MdT.
    - `posInizialeTestina` è la posizione della testina prima dell'inizio della computazione.

- Costruttore
  `public MacchinaDiTuring (String input, char beta, ArrayList<Stato> stati, Stato statoIniziale)`
  che richiama il costruttore precedente con `posInizialeTestina` uguale a 0 e `maxLunghezzaEvoluzione` uguale a 5000.
- `public void esegui()`
  che esegue il programma sulla MdT (qualora non sia stato già eseguito).
- `public boolean aggiungiIstruzione(Istruzione i)`
  che aggiunge alla MdT (qualora il programma non sia ancora stato mandato in esecuzione) l'istruzione `i` denotata dalla quintupla $\langle s, a, t, b, m \rangle$.
  Si *consiglia* di implementare la struttura dati che contiene l'insieme delle istruzioni con una HashMap in cui la chiave è costituita dalla coppia $\langle s, a \rangle$ e il valore dalla tripla $\langle t, b, m \rangle$.
  Si noti che s non può essere uno stato finale.
  Il metodo restituisce `true` se e solo se sia stato possibile aggiungere l'istruzione, e cioè:
  - s fa parte degli stati non finali della MdT;
  - t fa parte degli stati della MdT;
  - m appartiene all'insieme {-1,0,1}
  - non esiste nella MdT nessuna altra istruzione in cui le prime due componenti sono s e a;
  - il programma non è stato ancora mandato in esecuzione.
- `public ArrayList<Configurazione> getEvoluzione()`
  che restituisce la lista delle configurazioni ottenute durante l'esecuzione del programma sulla MdT (a partire dalla configurazione iniziale in cui il nastro contiene l'input), se il programma è stato già eseguito; restituisce `null` altrimenti.
- `public int getCondizione()`
  che restituisce un intero che codifica l'attuale **condizione** della MdT secondo la seguente convenzione:
  - `0` se il programma della MdT non è stato ancora eseguito;
  - `1` se il programma è stato eseguito terminando in uno stato finale;
  - `2` se il programma è stato eseguito terminando in uno stato non finale;
  - `3` se il programma è stato eseguito e l'esecuzione è stata interrotta (in uno stato non finale) per il raggiungimento del massimo numero di passi.
- `public Nastro getNastro()`
  che restituisce il nastro attuale della MdT.
- Il metodo `toString()` deve restituire una descrizione testuale della MdT che contenga:
  - tutti gli ingredienti della sua definizione, compresa la lista delle istruzioni;
  - la condizione in cui la MdT si trova;
  - nel caso in cui la condizione sia maggiore di zero, la lista delle configurazioni ottenute durante l'esecuzione del programma sulla MdT.