Laboratorio di Algoritmi e Strutture Dati

Docente: V. Lonati

Progetto "Automi e segnali"[1]

valido per gli appelli di gennaio e febbraio 2025

Indice

1 Organizzazione degli appelli e modalità di consegna

La realizzazione del progetto è una prova d'esame da svolgersi **individualmente**. I progetti giudicati frutto di **copiatura** saranno **estromessi** d'ufficio dalla valutazione.

Si richiede allo studente di effettuare un **adeguato collaudo** del proprio progetto su numerosi esempi diversi per verificarne la correttezza.

La versione aggiornata del progetto è pubblicata nella pagina del corso:
`https://myariel.unimi.it/course/view.php?id=3226`

Si consiglia di consultare periodicamente questo sito per eventuali correzioni e/o precisazioni relative al testo del progetto. Per ogni ulteriore chiarimento potete chiedere un appuntamento scrivendo una mail all'indirizzo `lonati@di.unimi.it` .

La presente traccia è valida per gli appelli di gennaio e febbraio 2025.

Il progetto svolto va consegnato entro una delle seguenti scadenze:

1. Prima scadenza: 15 gennaio 2025. Discussioni dei progetti: dopo il 22 gennaio 2025
2. Seconda scadenza: 29 gennaio 2025. Discussioni dei progetti: dopo il 5 febbraio 2025
3. Terza scadenza: 12 febbraio 2025. Discussioni dei progetti: dopo il 19 febbraio 2025

Le specifiche variano leggermente a seconda della scadenza, secondo quanto indicato nelle sezioni successive.

Occorre consegnare un programma Go e una relazione. Più precisamente:

[1] Ultima modifica 9 gennaio 2025 alle 12:41

1. Il programma Go (file sorgenti) deve essere preparato secondo le specifiche descritte nelle sezioni successive. Se il programma è formato da un solo file, il nome del file contenente il `main` deve essere nel formato `123456_cognome_nome.go`, dove `123456` è la matricola. Nel caso in cui il programma sia organizzato su più file, la relazione deve iniziare con un'indicazione di come è organizzato il programma e di come compilarlo.

2. La relazione deve essere in formato `.pdf`, e avere lunghezza compresa tra 3 e 10 pagine. La relazione deve descrivere in modo sintetico il programma, spiegando in che modo questo affronta il problema descritto nella traccia. Illustrate sia le scelte di progetto in relazione al problema (ad esempio; come è stato modellato il problema, quali strutture di dati sono state usate e perché) sia quelle implementative in relazione al programma (ad esempio: dettagli su come sono stati implementati gli algoritmi o su come sono state implementate le strutture di dati scelte in fase di progettazione).

   La relazione deve inoltre contenere una rassegna di esempi (diversi da quelli già proposti nella traccia).

3. Opzionalmente possono essere consegnati anche file di test in formato `.go`, oppure in formato `.txt` (coppie di input/output)

Tutti i file devono essere contenuti in un unico archivio `.zip` con nome `123456_cognome_nome.zip`. Tutti i file nell'archivio, compresa la relazione, devono riportare nome, cognome e matricola dell'autore.

Il file `.zip` va consegnato tramite il sistema di upload (`https://upload.di.unimi.it`) del Dipartimento entro le scadenze indicate; per ogni scadenza sarà aperta una specifica sessione di upload.

È possibile consegnare una sola volta; se un progetto presentasse delle mancanze o venisse valutato non pienamente sufficiente, potranno essere richieste integrazioni o revisioni del lavoro svolto.

Dopo ciascuna scadenza per la consegna, i progetti consegnati verranno corretti. Al termine della correzione verrà pubblicato un avviso con il calendario delle discussioni e l'elenco delle persone convocate per la discussione (la discussione non è obbligatoria per tutti).

È necessario presentarsi alla discussione con un computer portatile con i file consegnati (oppure richiedere con adeguato anticipo di svolgere la discussione in una aula con computer).

2 Criteri di valutazione dei progetti

Nella valutazione del progetto si terrà conto dei seguenti aspetti: capacità di modellare il problema mediante strutture di dati opportune; capacità di progettare soluzioni algoritmiche efficienti; chiarezza espositiva e proprietà di linguaggio nell'illustrare le scelte di modellazione e di implementazione fatte (sia in relazione al problema che in relazione al codice); capacità di implementare in Go le strutture di dati e gli algoritmi scelti, in maniera appropriata rispetto al tipo di elaborazione richiesta; correttezza formale (sintattica) del codice Go prodotto; funzionamento del programma (correttezza e completezza degli output prodotti dal programma); qualità del codice (codice non ripetuto/ridondante/intricato/oscuro, strutturazione del codice, ecc).

Il progetto verrà estromesso dalla valutazione in uno qualunque dei seguenti casi:

1. i file consegnati non rispettano il formato specificato sopra;

2. il programma non compila;

3. il programma non contiene la definizione dei tipi e delle funzioni indicate nelle *Specifiche di implementazione*;

4. il programma contiene le funzioni specificate, ma con segnatura diversa.

3 Il problema

Obiettivo del progetto è studiare gli spostamenti di automi puntiformi che si muovono su un terreno accidentato e sono sensibili a segnali che li richiamano verso un certo luogo.

Formalmente, chiamiamo *piano* l'insieme dei punti

$$\{\ (x, y) \in \mathbb{Z} \times \mathbb{Z}\ \}.$$

Un *passo unitario orizzontale* è un segmento orizzontale che collega due punti di coordinate (x, y) e $(x + 1, y)$. Un *passo unitario verticale* è un segmento verticale che collega due punti di coordinate (x, y) e $(x, y + 1)$.

Un *percorso* è una successione $S = p_1, p_2, \ldots, p_k$ di k passi unitari orizzontali e/o verticali tali che, per ogni $i \in \{1, \ldots, k - 1\}$, p_i e p_{i+1} hanno in comune solo un vertice e nessun punto è comune a più di due dei passi unitari. La *lunghezza* di S è k.

La *distanza* $D(A, B)$ tra due punti A e B è data dalla lunghezza minima dei percorsi che collegano. Dunque, se il punto A ha coordinate (x_A, y_A) e il punto B ha coordinate (x_B, y_B):

$$D(A, B) = |x_B - x_A| + |y_B - y_A|$$

Automi

Ogni *automa* è identificato univocamente da un *nome* η, che è una stringa finita sull'alfabeto $\{0, 1\}$ ($\eta = b_1 b_2 \cdots b_n$ per qualche intero positivo n e $b_i \in \{0, 1\}$ per ogni $i \in \{1, \ldots, n\}$).

La *posizione* $P(\eta)$ di un automa η in un dato momento è specificata dando le *coordinate* $(x_0, y_0) \in \mathbb{Z} \times \mathbb{Z}$ del punto del piano in cui l'automa si trova. Quindi, $P(\eta) = (x, y)$ significa che η si trova nel punto (x, y); in un punto può esservi più di un automa.

Ostacoli

Nel piano sono presenti degli *ostacoli* che limitano le possibilità di spostamento degli automi. Un ostacolo è definito come l'insieme di punti contenuti in un rettangolo con lati verticali e orizzontali. Un rettangolo si indica come $R(x_0, y_0, x_1, y_1)$, dove x_0, x_1, y_0, y_1 sono interi e (x_0, y_0) e (x_1, y_1) sono rispettivamente le coordinate dei vertici in basso a sinistra e in alto a destra del rettangolo.

È possibile che gli ostacoli si sovrappongano; nessun automa potrà posizionarsi in un punto appartenente a qualche ostacolo.

Un percorso è *libero* se non incontra ostacoli, ovvero se il rettangolo che lo definisce non ha punti in comune con alcun passo unitario del percorso.

Richiami

Una sorgente può emettere *segnali di richiamo* rappresentati da stringhe finite α sull'alfabeto $\{0, 1\}$. All'atto dell'emissione del segnale ogni automa determina se tale segnale lo riguarda o meno: l'automa di nome η deve rispondere al richiamo del segnale α se e solo se α è un prefisso di η, vale a dire $\eta = \alpha\, b_1 \cdots b_n$. Tra questi automi, si sposteranno verso la sorgente tutti e soli gli automi che possono muoversi lungo un percorso libero di distanza minima e che hanno distanza minima dalla sorgente.

Più precisamente, supponiamo che una sorgente posizionata in un punto del piano emetta il richiamo α e che l'insieme di automi a cui è diretto il richiamo sia $\{\eta_1, \ldots, \eta_k\}$ (dunque α è prefisso di ciascun η_i).

Per ogni i, indichiamo con d_i la distanza dell'automa η_i dalla sorgente α e sia

$$K = \{i \mid 1 \leq i \leq k \text{ e esiste un percorso libero di lunghezza } d_i \text{ per l'automa } \eta_i\}.$$

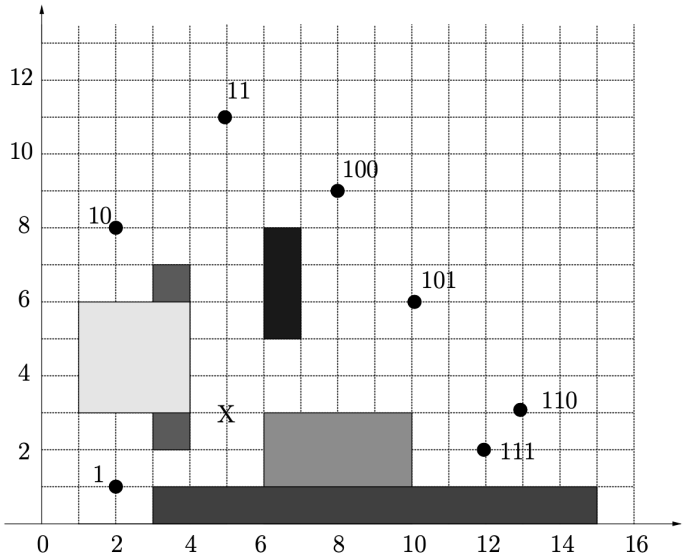
Definiamo quindi

$$d = \min\{d_i \mid i \in K\}$$

Allora, per ogni automa η_j con $j \in K$ e tale che $d_j = d$ si pone $P(\eta_j) = (x, y)$; per ogni altro automa, la posizione rimane immutata.

Esempio

Si supponga che nel piano ci siano gli automi 1, 10, 11, 100, 101, 110 e 111 e cinque ostacoli come nella figura



dove

$$P(1) = (2,1) \qquad P(10) = (2,8) \qquad P(11) = (5,11) \qquad P(100) = (8,9)$$
$$P(101) = (10,6) \qquad P(110) = (13,3) \qquad P(111) = (12,2)$$

e supponiamo che il richiamo 1 sia emesso dal punto $X = (5,3)$. Il segnale è recepito da tutti gli automi del piano.

L'automa 1 ha distanza 5 da X, tuttavia non esiste alcun percorso libero da $P(1)$ a X di lunghezza 5, quindi 1 non può raggiungere X.

L'automa 10 ha distanza $d_{10} = 8$ da X e può raggiungere X; lo stesso vale per gli automi 11 e 101.

L'automa 100 ha distanza $d_{100} = 9$ da X e può raggiungere X.

Gli automi 110 e 111 hanno distanza 8 da X, ma nessuno dei due può raggiungere X.

Quindi, $d = \min\{d_{10},\, d_{11},\, d_{101},\, d_{100}\}$ è 8 e si spostano in X gli automi 10, 11, 101 mentre gli altri stanno fermi.

Tortuosità - Solo per l'appello di febbraio 2025

Dato un percorso $S = p_1, \ldots, p_u$, siano $\delta_1, \ldots, \delta_u$ tali che, per ogni $i \in \{1, \ldots, u\}$, $\delta_i = O$ sse p_i è un passo unitario orizzontale, $\delta_i = V$ sse p_i è un passo unitario verticale. La *tortuosità* di S è data dal numero totale di indici $i \in \{1, \ldots, u-1\}$ tali che ($\delta_i = O$ e $\delta_{i+1} = V$) o ($\delta_i = V$ e $\delta_{i+1} = O$), ossia, dal numero di volte che seguendo il percorso si cambia direzione.

Ad esempio: 11 può andare direttamente in X con un percorso di tortuosità 0; 10 può andare in X con un percorso di tortuosità 1; la tortuosità minima di un percorso libero per 101 verso X è 2.

4 Specifiche di progettazione

Nota Bene: Le indicazioni seguenti potrebbero subire variazioni!

Si richiede di modellare la situazione con strutture di dati opportune e di progettare algoritmi che permettano di eseguire efficientemente le operazioni elencate sotto (si tenga presente che la minima porzione rettangolare di piano contenente tutti gli automi e tutti gli ostacoli può essere molto grande rispetto al numero di automi e alla dimensione degli ostacoli presenti, quindi *non è sicuramente efficiente rappresentare il piano mediante una matrice*).

Le operazioni indicate con `[FEBBRAIO]` devono essere considerate solo per l'appello di febbraio. Le altre operazioni devono essere considerate per ogni appello.

Le scelte di modellazione e di progettazione fatte devono essere discusse nella relazione, includendo l'analisi dei costi risultanti per le diverse operazioni.

Si richiede inoltre di predisporre una rassegna *esauriente* di esempi che potrebbero essere usati per testare il programma e che mettono in evidenza particolari caratteristiche del suo funzionamento (non solo casi tipici di input, ma anche casi limite e/o situazioni patologiche, oppure input che evidenzino la differenza di prestazioni tra le soluzioni progettuali scelte e altre meno interessanti).

- **crea** ()

  Crea un piano vuoto (eliminando l'eventuale piano già esistente).

- **stato** (x, y)

  Stampa un carattere che indica cosa si trova nella posizione (x, y): `A` se un automa, `O` se un ostacolo, `E` se la posizione è vuota.

- **stampa**

  Stampa l'elenco degli automi seguito dall'elenco degli ostacoli, secondo quanto indicato nelle *Specifiche di implementazione*.

- **automa** (x, y, η)

  Se il punto (x, y) è contenuto in qualche ostacolo, allora non esegue alcuna operazione. Altrimenti, se non esiste alcun automa di nome η lo crea e lo pone in (x, y). Se η esiste già, lo riposiziona nel punto (x, y).

- **ostacolo** (x_0, y_0, x_1, y_1)

  Se i punti nel rettangolo $R(x_0, y_0, x_1, y_1)$ non contengono alcun automa, inserisce nel piano l'ostacolo rappresentato da $R(x_0, y_0, x_1, y_1)$, altrimenti non compie alcuna operazione. Si può assumere che $x_0 < x_1$ e $y_0 < y_1$.

- **richiamo** (x, y, α)

  Viene emesso il segnale di richiamo α dal punto (x, y) (ovviamente, se il punto (x, y) appartiene a qualche ostacolo, esso non è raggiungibile da alcun automa).

- **posizioni** (α)

  Stampa le posizioni di tutti gli automi η tali che α è un prefisso di η, secondo il formato definito nelle note della sezione *Specifiche di implementazione*.

- **esistePercorso** (x, y, η)

  Stampa `SI` se esiste almeno un percorso libero da $P(\eta)$ a (x, y) di lunghezza $D(P(\eta), (x, y))$, `NO` in caso contrario (ovviamente, stampa `NO` se η non esiste o se (x, y) è un punto all'interno di un ostacolo).

- **tortuosità** (x, y, η) - `[FEBBRAIO]`

  Restituisce la tortuosità minima fra tutti i percorsi liberi da $P(\eta)$ a (x, y) di lunghezza $D(P(\eta), (x, y))$. Restituisce -1 se non esiste alcun siffatto percorso libero (ovviamente, se η non esiste, stampa -1).

5 Specifiche di implementazione

Nota Bene: Le indicazioni seguenti potrebbero subire variazioni!

1. Il programma deve leggere dallo standard input (`stdin`) una sequenza di linee (separate da a-capo), ciascuna delle quali corrisponde a una linea della prima colonna della Tabella 1, dove a, b, c, d sono numeri interi e ω è una stringa finita sull'alfabeto $\{0, 1\}$.

   I vari elementi sulla linea sono separati da uno o più spazi. Quando una linea è letta, viene eseguita l'operazione associata; le stampe sono effettuate sullo standard output (`stdout`), e ogni operazione deve iniziare su una nuova linea.

2. Il programma deve contenere:

   - la definizione di un tipo `piano` che rappresenta l'intero piano, incluso l'insieme degli automi e degli ostacoli;
   - una funzione con segnatura:

     `newPiano() piano`

     che crea un nuovo piano, lo inizializza e lo restituisce –ovvero che implementa l'operazione **crea**.
   - una funzione con segnatura:

     `esegui(p piano, s string)`

     che applica al piano rappresentato da `p` tutte le altre operazioni definite dalla stringa s, secondo quanto specificato nella Tabella 1.

Nota. Non devono essere presenti vincoli sulla dimensione della porzione di piano occupata da automi e ostacoli, sulla lunghezza delle stringhe binarie e sul numero di elementi nel piano.

	Linea di input	Operazione
	`c`	**crea** ()
	`S`	**stampa**
	`s` a b	**stato** (a, b)
	`a` a b ω	**automa** (a, b, ω)
	`o` a b c d	**ostacolo** (a, b, c, d)
	`r` a b ω	**richiamo** (a, b, ω)
	`p` ω	**posizioni** (ω)
	`e` a b ω	**esistePercorso** (a, b, ω)
	`f`	Termina l'esecuzione del programma
`[FEBBRAIO]`	`t` a b ω	**tortuosità** (a, b, ω)

Tabella 1: Specifiche del programma

Formato di output

1. L'operazione **stampa** visualizza prima l'elenco degli automi poi quello degli ostacoli.
2. Se si vuole stampare (ad esempio per l'operazione **stampa** o per l'operazione **posizioni**) l'elenco degli automi $\eta_1, \ldots, \eta_n$ con $P(\eta_i) = (x_i, y_i)$, allora l'elenco deve essere visualizzato nel seguente formato:

   (
   $\eta_1 : x_1, y_1$
   $\eta_2 : x_2, y_2$
   $\vdots$
   $\eta_h : x_h, y_n$
)

   L'ordine in cui appaiono gli automi $\eta_1, \ldots, \eta_h$ non è rilevante.
3. Siano $R_1, \ldots, R_m$ tutti i rettangoli che definiscono gli ostacoli e sia R_i definito dalla quadrupla $(x_{iA}, y_{iA}, x_{iB}, y_{iB})$. Allora l'elenco degli ostacoli deve essere visualizzato nel seguente formato:

   [
   $(x_{1A}, y_{1A})\,(x_{1B}, y_{1B})$
   $(x_{2A}, y_{2A})\,(x_{2B}, y_{2B})$
   $\vdots$
   $(x_{mA}, y_{mA})\,(x_{mB}, y_{mB})$

]

L'ordine in cui appaiono gli ostacoli non è rilevante.

6 Esempi di esecuzione

6.1 Automi e ostacoli in posizione x, y

Si supponga che le linee di input siano:

```
c
a 1 2 1
o 2 0 4 3
s 1 2
s 3 1
s 7 8
f
```

L'output prodotto dal programma deve essere:

```
A
0
E
```

6.2 Stampa di automi e ostacoli

Si supponga che le linee di input siano:

```
c
a 1 2 1
a 0 3 0
o 2 0 4 3
o -2 4 1 6
S
f
```

L'output prodotto dal programma deve essere (l'ordine degli automi e degli ostacoli è irrilevante):

```
(
1: 1,2
0: 0,3
)
[
(2,0)(4,3)
(-2,4)(1,6)
]
```

6.3 Percorsi liberi

Si supponga che le linee di input siano:

```
c
a 1 2 1
e 10 2 1
o 2 0 4 3
e 10 2 1
f
```

L'output prodotto dal programma deve essere:

```
SI
NO
```

6.4 Posizioni

Si supponga che le linee di input siano:

```
c
a 1 2 1
a 1 -2 010
a 0 0 11
p 1
f
```

L'output prodotto dal programma deve essere (l'ordine delle righe tra le parentesi è irrilevante):

```
(
1: 1,2
11: 0,0
)
```

6.5 Esecuzione di test automatici

Il file allegato `testXformato.zip` contiene dei file per eseguire test automatici Go: si invita all'utilizzo di tali test per verificare la correttezza del formato dell'output prodotto dal programma.

Per eseguire i test Go procedere come segue:

- aprire un terminale
- spostarsi della directory del programma
- salvare il programma con nome `solution.go`
- lanciare il comando `go mod init solution`
- creare l'eseguibile con il comando `go build solution.go`
- copiare nella directory i file contenuti nell'archivio con i test
- lanciare il comando `go test -v` (e osservare l'output)