Relazione progetto d'esame di algoritmi e strutture dati (revisione)

Introduzione

Questa relazione presenta le specifiche delle funzioni implementate allo scopo di risolvere il problema dato nella traccia d'esame "*Automi e segnali*".

Il problema richiede di studiare il movimento di automi puntiformi in un piano cartesiano contenente punti di richiamo (per gli automi) e ostacoli che gli automi devono evitare durante il loro spostamento. Lo spostamento può avvenire solamente nell'area del quadrato formato dall'automa e dal richiamo.

La relazione è divisa in due macro aree: la prima che definisce le scelte progettuali e le strutture dati utilizzate, la seconda si concentra sul fornire una breve spiegazione delle operazioni richieste e una più approfondita analisi di tutte le funzioni e i metodi che permettono la ricerca del percorso e lo spostamento degli automi verso il richiamo.

Nella relazione si farà riferimento alla distanza di Manhattan fra due punti del piano con D, al numero di automi nel piano con a e al numero di ostacoli con m.

Strutture dati e scelte progettuali

Per rappresentare il piano è stata usata una struttura con riferimento a due liste concatenate, una per contenere gli automi e un'altra per gli ostacoli. Questa scelta permette di gestire dinamicamente l'aggiunta e la modifica di automi e di ostacoli. Inoltre, la lista è una struttura relativamente leggera in termini di consumo di memoria e facile da manipolare.

Strutture dati principali

- **`punto`**: rappresenta un punto del piano. Struttura contenente: le coordinate `x` e `y`, un identificativo `id` e un puntatore a un tipo `punto`. Questo tipo di dato viene usato per rappresentare sia automi che ostacoli.
- **`Piano`**: struttura principale che mantiene riferimenti a una lista di ostacoli e a una di automi.
- **`piano`**: alias di un tipo puntatore a una variabile `Piano`.
- **`elementoPila`**: usata per gestire l'operazione di richiamo, memorizzando gli automi candidati allo spostamento. È una struttura composta da: un tipo `*punto`, che rappresenta l'automa candidato, un tipo `int` che rappresenta la distanza che il candidato ha dal richiamo e un tipo `*elementoPila` che collega la struttura a una pila. Questo tipo di dato è usata esclusivamente nel metodo `richiamo`.

Implementazione e tempi delle altre operazioni richieste

L'operazione `crea` viene implementata restituendo una nuova variabile di tipo `Piano`, se questa non esiste già. Sostituisce i puntatori alle liste degli ostacoli e degli automi del piano già esistente con puntatori vuoti. Questa operazione impiega tempo costante per essere eseguita. È implementata dalla funzione `newPiano`.

L'operazione `stato` viene implementata scorrendo la lista degli automi e degli ostacoli. L'operazione richiede tempo pari a $O(a + m)$ nel caso peggiore. L'operazione è implementata dal metodo `stato`.

L'operazione `stampa` scorre entrambe le liste, degli ostacoli e degli automi, del piano. Impiega tempo pari a $\Theta(a + m)$. L'operazione è implementata dal metodo `stampa`.

L'operazione `automa` viene implementata scorrendo la lista degli ostacoli e la lista degli automi facenti parte della struttura `Piano`. Questa operazione impiega $\Theta(a+m)$ ed è implementata dal metodo `automa`.

L'operazione `ostacolo` viene implementata scorrendo esclusivamente la lista degli automi del piano. Questa operazione impiega $\Theta(a)$. L'operazione è implementata dal metodo `ostacolo`.

L'operazione `posizioni` è implementata scorrendo la lista degli automi e verificando che l'automa considerato abbia il prefisso giusto. Questa operazione impiega $O(a)$ ed è implementata dal metodo `posizioni`.

Movimenti e percorsi degli automi

`forwardX` e `forwardY` sono due metodi utilizzati dalla funzione `avanza` (questa funzione è spiegata più avanti nel paragrafo) che permettono di determinare l'avanzamento dell'automa verso il richiamo. Sono i metodi che permettono di trovare un percorso di distanza minima D fra l'automa e il richiamo.

> Nota: Dato che la logica per `forwardX` e `forwardY` è la stessa, di seguito viene fornita esclusivamente la spiegazione di `forwardX`. Inoltre, si specifica che `forwardX` sposta l'automa solo sull'asse orizzontale e `forwardY` solo sull'asse verticale.

`forwardX` è un metodo che prende in ingresso due punti, `start` (l'automa) e `destination` (il richiamo). Restituisce un punto che si trova sullo stesso asse orizzontale (la stessa coordinata y) di `start` e più vicino all'asse verticale (la coordinata x) di `destination`. Il punto restituito è posizionato in modo che in seguito si possa poi fare uno spostamento verticale, cioè una `forwardY`, senza che l'automa resti bloccato da un ostacolo. Questo punto restituito verrà chiamato d'ora in poi λ.

In particolare, `forwardX` si comporta in questo modo:

controlla se c'è un ostacolo sull'asse verticale di `start`, nella direzione di `destination`. Da qui si hanno due possibilità:

1. Viene trovato un ostacolo. Allora, λ viene posizionato sulla prima coordinata x oltre l'ostacolo (per aggirarlo). A questo punto, `forwardX` restituisce λ e termina la sua esecuzione.

> Attenzione! Se λ viene posizionato all'interno di un ostacolo, è implicito che non ci siano percorsi liberi.

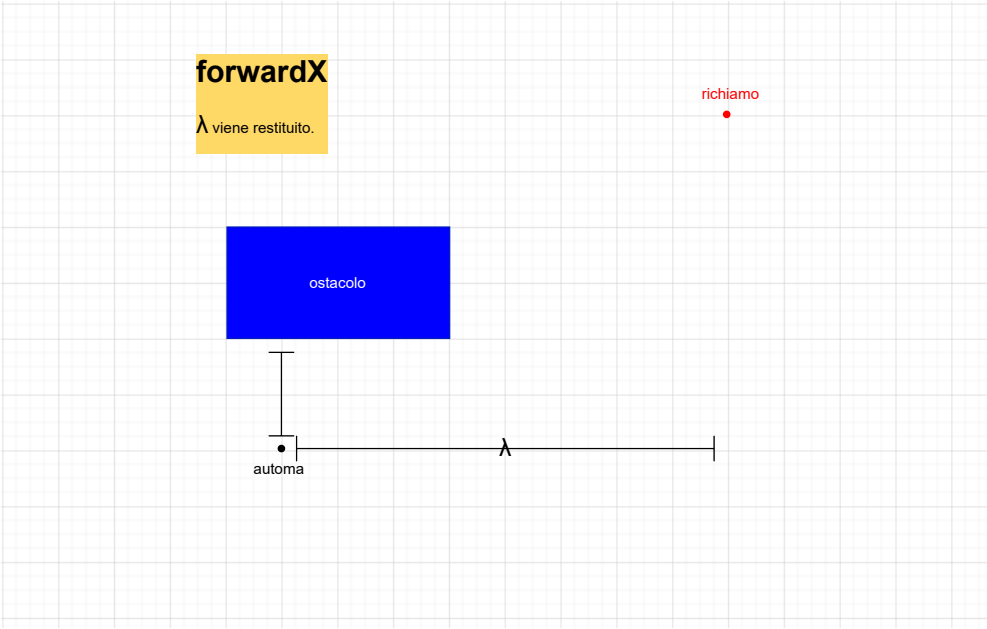
2. L'ostacolo non c'è. Allora si controlla che non ce siano anche sull'asse orizzontale (di `start`), fino alla coordinata x di `destination`. Da qui si presentano altri due casi possibili:

    - Non ci sono ostacoli sull'asse (almeno fino alla x di `destination`). Allora, λ viene posizionato sulla coordinata x di `destination`. Viene fatto un controllo degli ostacoli sull'asse verticale, questa volta di λ. Se non ci sono ostacoli, `forwardX` restituisce λ e termina la sua esecuzione. Altrimenti, λ viene retrocesso nella prima coordinata x utile per aggirare l'ostacolo. A questo punto `forwardX` restituisce λ e termina la sua esecuzione.

    - Viene trovato un ostacolo. λ viene posizionato sulla coordinata x subito precedente all'ostacolo trovato. Viene fatto un controllo per cercare ostacoli sull'asse verticale di λ. Se non ci sono ostacoli, `forwardX` restituisce λ e termina l'esecuzione. Altrimenti, λ viene retrocesso nella prima coordinata x utile per aggirare il nuovo ostacolo trovato.

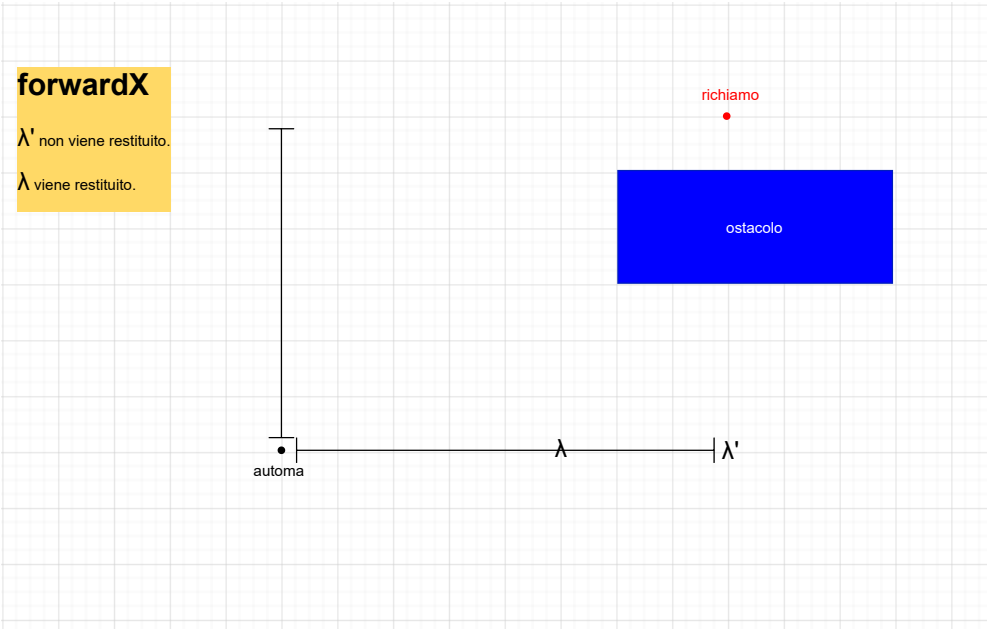
Esempi grafici.

1.



In questo caso l'automa trova un ostacolo sul suo asse verticale in direzione del richiamo. Allora λ viene messo sullo stesso asse orizzontale dell'automa in modo che aggiri l'ostacolo.

2.



Nel seguente caso non sono stati trovati ostacoli sull'asse orizzontale dell'automa, in direzione del richiamo. La `forwardX` quindi ha proceduto a controllare la presenza di ostacoli sull'asse orizzontale dell'automa (sempre in direzione del richiamo). Non trovandone, è stato posizionato λ' sulla stessa coordinata x del richiamo. Viene fatto un controllo sull'asse verticale di λ'. Trovato l'ostacolo si fa retrocedere λ in modo che possa aggirarlo.

Il metodo `forwardX`, quindi anche `forwardY`, impiega $O(m \cdot D)$. Questo tempo è dovuto al fatto che, per ogni punto dell'asse verticale e dell'asse orizzontale che il metodo deve controllare, scorre, nel caso peggiore, tutta la lista degli ostacoli del piano.

`avanza(Campo piano, p *punto, Sorgente *punto) *punto`: Questa funzione simula lo spostamento dell'automa `p` verso il richiamo, `Sorgente`. È una funzione ricorsiva che come caso base ha $D = 0$, dove D è la distanza di

Manhattan fra `p` e `Sorgente`. La funzione `avanza` restituisce il punto nel quale la simulazione si è fermata, ovvero se l'automa ha raggiunto la sorgente del richiamo oppure se è andato in stallo (non ha trovato percorsi liberi).

Questa funzione si serve dei metodi `forwardX` e `forwardY`.

`avanza` assume che `forwardX` e `forwardY` spostino sempre l'automa; se ciò non accade, l'automa è andato in stallo e la funzione termina. La funzione `avanza` calcola la distanza dagli ostacoli sull'asse orizzontale e verticale che l'automa ha in un determinato momento; in caso non vi siano ostacoli su almeno uno dei due assi dell'automa, si utilizza la distanza fra le rispettive coordinate dei due punti (la x e la y dell'automa con la x e la y della sorgente). L'automa quindi si sposterà solamente dove la distanza è maggiore. Per esempio, se la distanza è maggiore per l'asse verticale, allora verrà usata la `forwardY`, altrimenti la `forwardX`. Un caso particolare si ha quando la distanza dagli ostacoli è uguale su entrambi gli assi. In questo caso si esegue un passo unitario sull'asse orizzontale, a destra o a sinistra. Se questo non è possibile, allora l'automa è andato in stallo e la funzione termina.

La funzione `avanza` impiega $O(D^2 \cdot m)$ perché nel caso peggiore dovrà fare D passi ricorsivi e quindi eseguire altrettante volte la logica di forward.

L'operazione `richiamo` è implementata da un metodo omonimo. Questo metodo controlla gli automi più vicini al punto di richiamo e li fa spostare verso di esso. Per richiamare gli automi si controlla che il prefisso del loro id corrisponda al richiamo, se ciò avviene si esegue la funzione `avanza` per simulare lo spostamento dell'automa verso il richiamo. Quando l'avanza termina, il punto restituito, se ha le stesse coordinate del richiamo, viene inserito all'interno di una struttura `elementoPila` insieme ad un intero, che rappresenta la distanza di Manhattan tra l'automa e il richiamo. Dopo aver inserito tutti gli automi che si possono spostare all'interno della pila si controllano le distanze che essi hanno dal richiamo. Verranno effettivamente spostati solo quelli con distanza minima. Il tempo d'esecuzione del metodo `richiamo` è $O(a \cdot D^2 \cdot m)$ nel caso peggiore, ovvero se tutti gli automi del piano si possono muovere, sono tutti di eguale distanza dal richiamo e devono per forza eseguire D passi ricorsivi di `avanza`.

L'operazione `esistePercorso` è implementata come segue: si controlla prima che l'id cercato corrisponda a un automa esistente, poi si controlla se il punto di arrivo non corrisponda all'area di un ostacolo. Se una di queste condizioni non si verifica viene stampato su standard output `NO`. Altrimenti viene usata la funzione `avanza`. La funzione prende in ingresso l'automa e il punto d'arrivo e restituisce un punto. Se il punto restituito dalla funzione `avanza` ha le stesse coordinate del punto d'arrivo viene stampato su standard output `SI`, altrimenti `NO`. Il tempo impiegato per eseguire questa operazione è lo stesso della funzione `avanza` ($O(D^2 m)$).

Esempi di esecuzione e casi limite

Esempio 1: Inserimento di automi e ostacoli

```
c
a 2 3 101
a 5 6 110
o 4 4 6 6
S
```

Output atteso:

```
(
101: 2,3
110: 5,6
)
[
(4,4)(6,6)
]
```

Esempio 2: Verifica esistenza percorso libero

```
c
a -2000 -200 101
o 3 2 5 4
e 6 2 101
```

Output atteso:

```
NO
```

Caso limite 1: Automa circondato da ostacoli

```
c
a 7 4 11001
a 10 6 001
a 5 5 101
o 3 2 5 4
o 2 1 10 4
o 8 5 12 10
o 0 7 6 15
r 16 1 1
p 1
```

Output atteso:

```
(
101: 5,5
11001: 16,1
)
```