Relazione progetto d'esame di algoritmi e strutture dati

Introduzione

Il progetto implementa un sistema per la gestione del movimento di automi puntiformi su un piano, rispettando vincoli di ostacoli e segnali di richiamo. L'obiettivo è studiare il comportamento degli automi in un contesto dove vi sono ostacoli e richiami che definiscono un percorso minimo che l'automa dovrà intraprendere, se questo esiste. I movimenti possibili di un'automa in posizione $A(x_A, y_A)$ ad un richiamo di posizione $R(x_R, y_R)$ deono essere compresi nella distanza $D(A, R) = |x_R - x_A| + |y_R - y_A|$.

Strutture dati e scelte progettuali

Per la rappresentazione del piano si è utilizzata una lista doppiamente concatenata. Questa scelta permette di gestire dinamicamente l'aggiunta e la modifica di automi e l'aggiunta di ostacoli. Inoltre è una struttura relativamente leggera in termini di consumo di memoria. Gli automi sono slvati nella parte superiore della lista mentre gli ostacoli nella parte inferiore. Questo permette di cercare gli automi e gli ostacoli in maniera più efficiente.

Strutture dati principali

- `punto`: rappresenta un nodo del piano, contenente coordinate `(x, y)`, un identificativo (`id`) e riferimenti al nodo precedente e successivo.
- `piano`: alias di un tipo puntatore ad una variabile `Piano`.
- `Piano`: struttura principale che mantiene riferimenti ai nodi iniziale e finale della lista doppiamente concatenata.
- `nodoPila`: usato per gestire le operazioni di richiamo degli automi, memorizzando i candidati allo spostamento.

Questa modellazione consente un accesso rapido agli automi e agli ostacoli, facilitando operazioni come la ricerca, l'inserimento e la gestione dei percorsi.

Implementazione delle operazioni

Le operazioni implementate nel programma seguono le specifiche fornite. Di seguito una descrizione delle principali:

Creazione e gestione del piano

- `newPiano()`: crea un nuovo piano vuoto, eliminando i dati esistenti. In particolare assegna due puntatori vuoti da assegnare al campo `inizio` e al campo `fine`.
- `stampa()`: stampa la lista degli automi e degli ostacoli secondo il formato richiesto.
- `stato(x, y)`: restituisce il contenuto della posizione `(x, y)` (`A` per automi, `O` per ostacoli, `E` per posizioni vuote).

Inserimento e gestione di automi e ostacoli

- `automa(x, y, eta)`: posiziona un nuovo automa o riposiziona uno esistente, in caso questo abbia lo stesso nome ma coordinate differenti, verificando che non sia in un ostacolo.
- `ostacolo(x0, y0, x1, y1)`: aggiunge un nuovo ostacolo se non vi sono automi nella sua area. L'ostacolo viene riconosciuto dalla segnatura `"ostacolo"` presente all'interno dell'id.

Movimenti e richiami

- `richiamo(x, y, alpha)` : emette un segnale che richiama gli automi con prefisso `alpha` verso `(x, y)` , se possibile. Questi automi vengono effetivamente spostati se: hanno distanza minima, rispetto agli altri automi, minore e se il percorso di distanza minima è libero dagli ostacoli.
- `esistePercorso(x, y, eta)` : verifica se esiste un percorso libero minimo da un automa `(eta)` alla destinazione `(x, y)` . Questa metodo stampa `SI` in caso effettivamente esista un percorso di distanza D dall'automa `eta` al punto `(x,y)` altrimenti stampa `NO` .

Spiegazione dettagliata delle funzioni e metodi

Funzioni principali

- `esegui(p piano, s string)` : interpreta e esegue i comandi ricevuti nel main da standard input.
- `newPiano() piano` : inizializza una nuova struttura `Piano` .
- `(Piano)stampa()` : stampa automi e ostacoli.
- `(Piano)stato(x, y int)` : restituisce informazioni sulla posizione `(x, y)` .
- `(*Piano)posizioni(alpha string)` : stampa le posizioni degli automi che iniziano con `alpha` .
- `(Campo *Piano)automa(x, y int, eta string)` : aggiunge un automa, o ne modifica la posizione, se la posizione è libera.
- `(Campo *Piano)ostacolo(x0, y0, x1, y1 int)` : aggiunge un ostacolo controllando che non vi siano automi nella sua area.
- `(Campo *Piano)richiamo(x, y int, alpha string)` : gestisce il richiamo degli automi compatibili.
- `(Campo *Piano)esistePercorso(x, y int, eta string)` : verifica se un automa ha un percorso libero verso `(x, y)` .

Metodi e funzioni

- `(Campo *Piano)cercaOstacolo(x, y int) *punto` : cerca l'stacolo che contine il punto di coordinate `(x, y)` e lo restituisce.
- `estraiCoordinate(id string) (x0, y0, x1, y1 int)` : estrae le coordinate dall'id di un'ostacolo.
- `(Campo *Piano)cerca(x, y int, id string) *punto` : ricerca un automa o un ostacolo nel piano.
- `calcolaDistanza(x0, y0, x1, y1 int) int` : calcola la distanza di Manhattan tra due punti (x_0, y_0) e (x_1, y_1) .
- `avanza(Campo piano, p *punto, Sorgente *punto, passi int) (*punto, int)` : determina il percorso minimo e controlla gli ostacoli.
- `(p *punto)posizioneOstacoloVerticale(Campo piano, y int) (bool, *punto)` : verifica se esistono ostacoli verticali lungo uno specifico asse x di un punto.
- `(p *punto)posizioneOstacoloOrizzontale(Campo piano, x int) (bool, *punto)` : verifica la presenza di ostacoli orizzontali lungo uno specifico asse y di un punto.
- `(Campo *Piano)forwardingX(start *punto, destination *punto) *punto` : determina il miglior percorso orizzontale evitando ostacoli. Tramite questo metodo viene controllanto l'avanzamento dell'automa in modo che non rimanga incastrato in caso esista un percorso percorribile.
- `(Campo *Piano)forwardingY(start *punto, destination *punto) *punto` : determina il miglior percorso verticale evitando ostacoli. Analogamente alla `forwardingX` questa funzione controlla che l'automa avanzando non si

incastri quando vi sono percorsi di distanza minima D disponibili.

> **Digressione su funzione `avanza` e metodi `forwarding`**
>
> La funzione `avanza` implementa la ricerca del percorso minimo fra un punto e un segnale di richiamo e controlla il suo spostamento. In base alla distanza che il punto ha dagli ostacoli sul suo asse delle x e sul suo asse delle y, al momento del controllo, effettua l'avanzamento dove questa è maggiore. Lo spostamento è implementato dai metodi `forwardingX` e `forwardingY`.
>
> I metodi `forwardingX` e `forwardingY` sono usate per effettuare un salto condizonato senza che l'automa resti bloccato da un'ostacolo.
>
> - `forwardingX` controlla la presenza di ostacoli lungo il suo asse (x) e effettua uno spostamento sulla coordinata x, successiva o antecedente al vertice dell'ostacolo, più vicino alla sorgente del segnale. In caso non ci siano osatcoli sull'asse, l'automa, si muove sulle x fino a quando non trova un'ostacolo o fino a quando non arriva alla coordinata x del richiamo. Viene fatto un'ulteriore controllo sul nuovo asse x, per verificare la presenza di ostacoli. Se non ce ne sono si ferma. Altrimenti si sposta sulla coordinata x, successiva o antecedente al vertice dell'ostacolo trovato, più vicina al punto di partenza dell'automa. Questo metodo permette spostamenti solamente sull'asse orizzontale.
> - `forwardingY` controlla la presenza di ostacoli lungo il suo asse (y) e effettua uno spostamento sulla coordinara y, successiva o precedente al vertice dell'ostacolo, che si avvicina di più alla sorgente del segnale. In caso non vi siano ostacoli, l'automa, si sposta su y fino a quando non arriva sulla coordinata y del richiamo, o fino a che non viene trovato un'ostacolo sul suo percorso. Dopodichè controlla se sull'asse orizzontale del punto appena raggiunto vi sono ostacoli. In caso affermativo, viene fatto uno spostamento sul punto y, successivo o precedente al vertice dell'ostacolo, più vicino al punto di partenza dell'automa. Altrimenti si ferma. Questo metodo permette spostamenti solamente sull'asse verticale.
>
> Entrambi i metodi utilizzano informazioni sulle coordinate e sulla distanza degli ostacoli per determinare il punto più vicino e accessibile, se presente, ottimizzando il percorso.

Scelte implementative e analisi delle prestazioni

L'uso di una lista doppiamente concatenata consente di mantenere basso l'uso di memoria. La gestione delle operazioni principali avviene con complessità ottimizzata:

Si consideri n come il numero totale di elementi nel piano.

Si consideri a come il numero totale di automi nel piano.

Si consideri m come il numero totale di ostacoli nel piano.

Si consideri d come la distanza minima da un punto al segnale.

- **Inserimento di automi**: tempo medio $\Theta(m)$. Vengono controllati tutti gli ostacoli per verificare che l'automa non occupi l'area di un'ostacolo.
- **Inserimento di ostacoli**: tempo medio $\Theta(a)$. Vengono controllati tutti gli automi in modo da sapere se occupano l'area dell'ostacolo da inserire.
- **Verifica di percorsi liberi**: tempo peggiore $O(d \cdot m)$ nel caso peggiore. Si controlla per ogni punto della distanza se fa parte dell'area di un'ostacolo.

- **Gestione dei richiami**: usa una pila per determinare gli automi che si spostano con complessità $O(a)$. Nel caso che tutti gli automi del piano siano candidati per raggiungere il richiamo. Il richiamo sfrutta la verifica dei percorsi liberi, quindi il tempo d'esecuzione nel caso peggiore è $O(a \cdot d \cdot m)$.

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
```

Esempio 3: Richiamo degli automi

```
c
a 1 1 101
a 3 3 110
o 2 2 4 4
r 5 5 1
p 1
```

Output atteso:

```
(
110: 5,5
101: 1,1
)
```

Esempio 4: Operazioni multiple

```
c
a 5 6 101
s 6 5
o 2 4 8 6
S
e 2 1 101
r 2 1 1
p 1
f
```

Output atteso:

```
E
(
101: 5,6
)
[
]
SI
(
101: 2,1
)
```

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
)
```

L'automa non si muove perché circondato da ostacoli da ostacoli

Caso limite 2: Verifica con molti ostacoli

```
c
a 20 7 101
a 1 1 101
a 10 10 110
o 4 7 5 9
o 3 1 6 4
o 4 8 7 10
o 2 1 8 5
o 5 4 6 9
o 2 2 9 6
o 0 4 7 6
e 10 10 101
```

Output atteso:

```
NO
```

Caso limite 3: Grandi distanze

```
c
a 1 9 100
o 1000 30 50000 5500
o 800000 -400 3000000 94
e 20000000 1 100
r 20000000 1 1
S
```

Output atteso:

```
SI
(
100: 20000000,1
)
[
(800000,-400)(3000000,94)
(1000,30)(50000,5500)
]
```

Conclusione

Il progetto implementato è conforme alle specifiche e fornisce un'efficace gestione degli automi e degli ostacoli nel piano. Ulteriori miglioramenti potrebbero includere ottimizzazioni sugli algoritmi di percorso per ridurre ulteriormente il tempo di esecuzione in scenari complessi.