Laboratorio “Algoritmi e Strutture Dati” - Progetto “Automi e Segnali” - Autore “Ingenito Emiddio”, Matricola 41461A

Struttura della Directory del Progetto

In Questo documento, vengono presentate le scelte implementative e le strutture dati utilizzate per modellare il problema “Automi e segnali”.

Il progetto è stato testato secondo i test forniti dalla professoressa.

Il progetto è organizzato in una struttura a directory:

- **Directory `Soluzione/`**: - Questa directory contiene i file che definiscono la mia implementazione del problema proposto.
- Directory `input-output` - Questa directory contiene una serie di esempi di esecuzione del progetto implementato, evidenziando casi limite o patologici.

Per compilare il progetto, spostarsi da terminale nella directory principale del progetto `Soluzione/` , procedere alla compilazione tramite il comando `go build .`

All’interno della directory `Soluzione/`, sono presenti i seguenti files:

- **`automa.go`**: Definisce l’entità `Automa`.

  La funzione `stampa` stampa il nome e la posizione dell'automa, mentre la funzione a`utoma` consente di creare un automa in una specifica posizione di un `piano`.

- **`coordinate.go`**: Definisce la struttura `coordinate`, che rappresenta una coppia di coordinate X, Y in un piano cartesiano. Include una funzione `nuoveCoordinate` per la creazione di nuove coordinate, e una funzione `distanza`utile a calcolare la distanza Manhattan tra due coordinate.

- **`esegui.go`**: Contiene la funzione `esegui` che interpreta il comando indicato dalla stringa passata come parametro, e applica l'operazione corrispondente al piano in ingresso, come indicato da specifiche

    - c - Operazione di creazione di un nuovo piano
    - S - Stampa dell'elenco degli automi e degli ostacoli presenti nel piano
    - s a b - Indica che entità si trova nelle coordinate definite come \(X=a, Y=b\) (A se un automa, O se un ostacolo, E se la posizione è vuota)
    - a a b ω - Crea o riposiziona un automa dal nome ω, nelle coordinate \(X=a, Y=b\)
    - o a b c d - Crea un nuovo ostacolo, definito come un rettangolo dai vertici in basso a sinistra di coordinate \(X=a, Y=b\), e in alto a destra di coordinate \(X=c, Y=d\)
    - r a b ω - Richiama un automa verso una posizione specificata, a condizione che non ci siano ostacoli tra l'automa e la posizione di destinazione.
    - e a b ω - verifica se esiste un percorso tra l'automa di nome ω, e il punto di coordinate \(X=a, Y=b\)
    - p ω - Stampa le posizioni di tutti gli automi che hanno un nome che inizia con un dato prefisso ω .

- **`ostacoli.go`**: Definisce la struttura `ostacolo` . La funzione `stampa` visualizza le coordinate dei vertici dell'ostacolo.

  La funzione `ostacolo` permetta l'aggiunta di un nuovo ostacolo ad un piano.

- **`piano.go`**: Definisce la struttura `piano`.

  Gestisce le operazioni di creazione di un piano, stampa delle entità che lo popolano (Insieme di automi e ostacoli), verifica dello stato di una coordinata del piano (per verificare la presenza di un automa o di un ostacolo).

  Permette inoltre di emettere segnali di richiamo da una casella di coordinate specificate, e di verificare la presenza di un percorso tra due coordinate.

Modellazione del Problema

Il problema proposto richiede di simulare un piano cartesiano su cui sono posizionati degli automi e degli ostacoli. Ogni automa si muove nel piano e reagisce a segnali di richiamo.

Struttura `coordinate`

La struttura `coordinate` rappresenta una coppia di coordinate (x, y).

Essendo i due campi rappresentati con un tipo primitivo int, lo spazio occupato dalla struttura sara' O(1).

Struttura `ostacolo`

La struttura `ostacolo` rappresenta un ostacolo nel piano sotto forma di un rettangolo, definito da due coordinate: il vertice in basso a sinistra e il vertice in alto a destra.

I campi della struttura sono implementati utilizzando il tipo primitivo `coordinate` per memorizzare i vertici del rettangolo, quindi lo spazio occupato dalla struttura è O(1).

Struttura `automa`

La struttura `automa` rappresenta un automa con un proprio nome e una posizione all'interno del piano.

I campi sono:

- **Nome (String)**
- **Posizione**: Struttura `coordinate`.

Tralasciando la lunghezza della stringa, anche per questa struttura possiamo considerare la complessità spaziale costante.

Struttura `piano`

Il piano è rappresentato dalla struttura `piano`, che contiene:

- **Automi** `map[string]*automa`: una mappa che associa il nome di un automa alla sua istanza .
- **Ostacoli** `*[]ostacolo`: un insieme di ostacoli , ciascuno rappresentato come un rettangolo definito da due coordinate (angolo in basso a sinistra e angolo in alto a destra).
- **Mappa** `map[coordinate][]interface{}`: una mappa che associa ad ogni punto del piano (rappresentato da coordinate), una lista di entità (di tipo `automa` o `ostacolo`) che si trovano in quella posizione. Questo permette di sapere quali entità si trovano in una determinata posizione del piano.

Gestione degli Automi

La presenza degli automi nel piano viene gestita tramite la presenza della mappa `automi` direttamente nel piano. Ogni volta che un automa si sposta o viene creato, la sua posizione viene aggiornata sia nella mappa degli automi che nella mappa `mappa` del piano.

Gestione degli Ostacoli

Gli ostacoli sono rettangoli che impediscono il movimento degli automi.

Quando un ostacolo viene aggiunto al piano, la struttura `Mappa` del piano viene aggiornata per riflettere la sua presenza, aggiornando e marcando ogni punto compreso tra i due vertici del rettangolo, come un ostacolo.

Funzioni implementate

Funzione `newPiano`

Crea una nuova struttura di tipo `Piano`, inizializzando le sue strutture dati:

- **`Automi`**: una mappa (vuota) che assocerà i nomi degli automi (`string`) a puntatori a strutture di tipo `automa`
- **`Ostacoli`**: una lista (vuota) di ostacoli (`*[]ostacolo`).
- **`Mappa`**: una mappa (vuota) che associa le coordinate (`coordinate`) a una lista di entità generiche (`[]interface{}`).

Tempo: **O(1)**.

Funzione `stampa` (Piano)

La funzione `stampa` stampa lo stato del piano, visualizzando la lista degli automi e degli ostacoli presenti.

Tempo: **O(a + o)**, dove `a` è il numero di automi e `o` il numero di ostacoli.

Funzione `stato`

Restituisce lo stato della posizione `(x, y)` nel piano:

- `"A"` se c'è un automa nella posizione.
- `"O"` se c'è un ostacolo nella posizione.
- `"E"` se la posizione è vuota.

Tempo: O(1).

Funzione `automa`

Gestisce l'aggiunta o l'aggiornamento di un automa in una posizione `(x, y)` nel piano:

1. Verifica se la posizione è occupata da un ostacolo. Se sì, non effettua alcuna modifica.
2. Se l'automa è già presente nel piano, rimuove la sua vecchia posizione dalla mappa.
3. Crea un nuovo automa (o aggiorna quello esistente) e lo inserisce nella mappa degli automi e nella mappa del piano, associando la sua posizione con l'automa.

Tempo: O(1).

Funzione `esistePercorso`

Verifica se esiste un percorso tra due punti sul piano, evitando ostacoli. Si tratta di una variante della ricerca in ampiezza (BFS - Breadth-First Search). L'algoritmo esplora il piano a partire dalla posizione iniziale, espandendo progressivamente i possibili movimenti (su, giù, destra, sinistra) finché non si trova il punto di arrivo o si esauriscono tutte le opzioni.

- **Inizializzazione**:

- Si crea una coda (`queue`) che conterrà i nodi (coordinate) da esplorare.
- Si crea una mappa (`visited`) per tenere traccia delle coordinate già visitate. Questo impedisce di esplorare ripetutamente le stesse celle e di entrare in loop infiniti.
- La coordinata di partenza (`start`) viene inserita nella coda e marcata come visitata.

- **Esplorazione delle coordinate del piano**:

  - Finchè la coda non è vuota, l'algoritmo estrae dalla coda la coordinata in prima posizione, e la esplora.
  - Se la coordinata corrente è uguale alla destinazione (`end`), allora il percorso è stato trovato e l'algoritmo restituisce `true` (goal check).
  - Altrimenti, l'algoritmo espande la coordinata nelle 4 direzioni (su, giù, destra, sinistra) e per ciascuna direzione verifica se la nuova posizione:
    - Non è stata visitata.
    - Non è un ostacolo.
    - È all'interno del piano.

  Se una direzione è valida, la coordinata viene aggiunta alla coda per essere esplorata successivamente e marcata come visitata.

- **Termine della Ricerca**:

  - Se l'algoritmo trova una coordinata che corrisponde alla destinazione, restituisce `true`.
  - Se la coda diventa vuota, non è stata trovata alcuna soluzione, restituisce `false`, indicando che non esiste un percorso dal punto di partenza alla destinazione.

Tempo: **O(m*n)**, dove `m` e `n` rappresentano nel caso peggiore, le dimensioni del piano (la complessità corrisponde all'area del piano)

Spazio: **O(m*n)**, pari all'area del piano.

Funzione `richiamo`

La funzione `richiamo` ha lo scopo di gestire il comportamento degli automi in risposta a un segnale di richiamo emesso dalla posizione `(x, y)` nel piano.

Produce effetti solo se la posizione da cui il richiamo viene emesso non è occupata da un ostacolo, e si occupa di spostare gli automi che rispondono al richiamo e sono raggiungibili dalla sorgente.

La funzione opera nel seguente modo:

1. **Controllo della posizione di partenza**: la funzione verifica se la posizione iniziale `(x, y)` è occupata da un ostacolo. Se si, la funzione termina.
2. **Selezione degli automi**: Itera tra tutti gli automi nel piano, selezionando quelli il cui nome inizia con la stringa specificata (`nome`). Per ogni automa che soddisfa questa condizione, viene calcolata la distanza dalla sorgente.
3. **Ordinamento delle distanze**: le distanze calcolate vengono ordinate in ordine crescente, così da considerare prima gli automi più vicini alla sorgente.

4. **Verifica del percorso**: la funzione verifica, per ogni automa a minima distanza, se esiste un percorso libero che consenta all'automa di raggiungere la sorgente. Se il percorso esiste, l'automa viene spostato verso la sorgente, aggiornando la sua posizione tramite il metodo `automa`.
5. **Termine dell'esecuzione**: La funzione termina quando tutti gli automi raggiungibili a distanza minima sono stati spostati verso la sorgente.

Tempo: O(A*n*m), dove `A` è il numero di automi e `n*m` è l'area del piano, dovuto alla chiamata ripetuta della funzione `esistePercorso` per ogni automa.

Spazio: O(A*n*m),, dove `A` è il numero di automi e `n*m` è l'area del piano, dovuto alla chiamata ripetuta della funzione `esistePercorso` per ogni automa.

Funzione `posizioni`

Stampa tutte le posizioni degli automi che hanno un nome che inizia con il prefisso dato come parametro (`prefisso`).

Tempo: **O(n)**, dove `n` è il numero di automi.

Funzione `isOstacolo`

Verifica se una posizione è occupata da un ostacolo.

Tempo: **O(1)**

Funzione `isAutoma`

Verifica se una posizione è occupata da un automa.

Tempo: **O(1)**

Funzione `stampa` (Ostacolo)

Stampa le coordinate dei vertici dell'ostacolo (inferiore sinistro e superiore destro).

Tempo: **O(1)**

Funzione `ostacolo`

Aggiunge un ostacolo al piano, se l'area specificata non contiene automi.

Marca l'area dell'ostacolo sulla mappa.

Tempo: **O(n * m)**, dove `n` è la differenza tra `x1` e `x0`, e `m` è la differenza tra `y1` e `y0`. Per ogni coordinata, la funzione verifica se un automa è presente e aggiorna la mappa.

Spazio: **O(n * m)**, in quanto l'area dell'ostacolo viene memorizzata nella mappa.

Funzione `nuoveCoordinate`

Crea e restituisce una nuova struttura `Coordinate` con le coordinate `x` e `y` fornite come parametri.

Tempo: **O(1)**

Funzione `distanza`

Calcola la distanza di Manhattan tra due coordinate (la somma delle differenze assolute tra le rispettive componenti `X` e `Y`).

Tempo: **O(1)**

Funzione `stampa` (Automa)

Stampa il nome e la posizione di un automa nel formato `Nome: x,y`.

Tempo: **O(1).**

Funzione `automa`

Posiziona un automa nella mappa del piano in una specifica coordinata `(x, y)` con il nome dato. Se la posizione non è occupata da un ostacolo, l'automa viene aggiunto o aggiornato nella mappa e nella lista degli automi.

Tempo: **O(1)**

Esempi di esecuzione

Input

```
c

r -85 -25 1100
o -53 48 -50 52
r 38 -82 11
r 30 -91 101
o -52 -68 -40 -67
r 73 63 11
r 11 83 10
o 31 -55 33 -46
S
f
```

Output

```
(
)
[
(-53,48)(-50,52)
(-52,-68)(-40,-67)
(31,-55)(33,-46)
]
```

Input

```
c
a -30 20 110
a 15 -25 1011
a 42 37 100
a -80 50 1100
o -10 5 -5 10
r -20 15 11
r 50 40 101
o 20 30 25 35
S
r -70 45 110
r 5 -10 1010
o 60 50 65 55
r 15 15 100
S
```

Output

```
(
1100: -80,50
110: -20,15
1011: 50,40
100: 42,37
)
[
(-10,5)(-5,10)
(20,30)(25,35)
]
(
100: 15,15
1100: -70,45
110: -20,15
1011: 50,40
)
[
(-10,5)(-5,10)
(20,30)(25,35)
(60,50)(65,55)
]
```

Input

```
c
a 10 10 111
a 20 20 110
a 30 30 1001
a -10 -10 1011
a -20 -20 1
o 5 5 15 15
o -15 -25 -5 -15
r 12 12 11
S
r -18 -18 1
r 25 25 100
o 50 50 60 60
r 55 55 1011
S
```

Output

```
(
1001: 30,30
1011: -10,-10
1: -20,-20
111: 12,12
110: 20,20
)
[
(-15,-25)(-5,-15)
]
(
1011: -10,-10
1: -18,-18
111: 12,12
110: 20,20
1001: 25,25
)
[
(-15,-25)(-5,-15)
(50,50)(60,60)
]
```