Relazione Progetto Algoritmi e Strutture Dati

Fernando Henrique Gavezzotti

a.a. 2024-2025

Indice

1 Organizzazione dei File

Il file `solution.go` contiene tutto il codice del progetto, compreso il `main`.

2 Contesto Generale

Il progetto consiste in un programma che simula il movimento di automi su un piano infinito. Gli automi possono essere posizionati in qualsiasi punto del piano e possono muoversi solo in orizzontale o in verticale. Il piano è costituito da ostacoli, che sono rettangoli definiti da due punti: il primo in basso a sinistra e il secondo in alto a destra. Gli automi non possono attraversare gli ostacoli.

Nel piano possono essere emessi dei punti di richiamo, che sono posizioni specifiche che gli automi possono raggiungere. Il percorso tra l'automa e il punto di richiamo deve corrispondere alla distanza minima tra i due punti.

La distanza minima tra due punti a e b è data dalla formula: $|x_b - x_a| + |y_b - y_a|$

3 Modellazione del Problema

3.1 Strutture Dati Utilizzate

Il problema è stato modellato come un piano infinito con:

- **Automi**: rappresentati come punti definiti da coordinate intere $(x,\ y)$.
- **Ostacoli**: rettangoli definiti dai vertici in basso a sinistra e in alto a destra; i vertici sono definiti da coordinate intere $(x,\ y)$.

Per rappresentare le coordinate è stata definita una struttura `coordinates` che contiene:

- `x int`: coordinata x del piano;
- `y int`: coordinata y del piano.

In particolare per rappresentare il piano, gli automi e gli ostacoli è stata definita una struttura `Piano` che contiene:

- `automata map[string]coordinates`: mappa che associa ad ogni nome di automa le sue coordinate, in modo da poter accedere velocemente alle coordinate di un automa dato il suo nome;
- `numberOfAutomata map[coordinates]int`: mappa che associa ad ogni coordinata il numero di automi presenti in quella posizione, in modo da poter controllare più velocemente se una posizione $(x,\ y)$ è occupata da un automa;
- `obstacles [][2]coordinates`: slice di coppie di coordinate che rappresentano ciascun ostacolo.

Inoltre, per poter seguire le specifiche fornite, è stato definito un tipo `piano` che è un puntatore a `Piano`, in modo da poter passare il piano per riferimento e modificarlo all'interno delle funzioni senza doverlo restituire.

3.2 Funzioni Utilizzate e Spiegazioni

- `newPiano() piano`: restituisce un nuovo piano vuoto, con `automata`, `numberOfAutomata` e `obstacles` inizializzati a valori vuoti. Tempo: costante.
- `func obstacle(p piano, point1 coordinates, point2 coordinates)`: aggiunge un ostacolo di coordinate `point1` (punto in basso a sinistra) e `point2` (punto in altro a destra) alla slice di ostacoli del piano `p`. Se nelle coordinate di `point1` o `point2` sono presenti automi, l'ostacolo non viene aggiunto. Tempo: costante;
- `automaton(p piano, point coordinates, name string)`: aggiunge un automa di coordinate `point` e di nome `name` alle due mappe del piano `p`. In particolare viene aggiunto il nome e le coordinate alla mappa `automata` e viena aggiornata la chiave di `numberOfAutomata` per quella posizione. Se in `point` è presente un ostacolo o l'automa con quel nome esiste già, non viene aggiunto al piano. Tempo: $O(n)$, dove n è il numero di ostacoli presenti sul piano perché è necessario iterare gli ostacoli per controllare che il punto non sia all'interno di un ostacolo;
- `printPiano(p piano)`: stampa gli ostacoli e gli automi presenti sul piano `p`. Per stamparli è necessario fare un ciclo `for` sulla mappa contenenti gli automi e sulla slice contenente gli ostacoli. Tempo: $\Theta(n+m)$ dove n è il numero di automi presenti sul piano e m il numero di ostacoli;
- `positions(p piano, prefix string)`: stampa tutti gli automi presenti sul piano `p` che hanno il prefisso `prefix` nel nome. Per poter effettuare la stampa è necessario iterare sulla mappa contenente gli automi del piano. Tempo: $O(n \cdot m)$ dove n è il numero di automi presenti sul piano e m è la lunghezza del prefisso `prefix`;
- `pointIsInObstacle(p piano, point coordinates) bool`: controlla se il punto `point` si trova all'interno di un ostacolo. Per poter fare ciò, è stato necessario iterare la slice contenente gli ostacoli del piano, e controllare che le coordinate del punto fossero comprese tra le coordinate dei vertici dell'ostacolo. Tempo: $O(n)$ dove n è il numero di ostacoli presenti sul piano;
- `pointIsInAutomaton(p piano, point coordinates) bool`: controlla se il punto `point` si trova all'interno di un automa utilizzando la mappa `numberOfAutomata` che associa ad ogni coordinata il numero di automi presenti in quella posizione. Tempo: costante;
- `state(p piano, point coordinates)`: restituisce lo stato di un automa. Lo stato è controllato utilizzando le funzioni `pointIsInObstacle` e `pointIsInAutomaton`. Tempo: $O(n)$, dove n è il numero di ostacoli presenti sul piano;
- `distance(point1 coordinates, point2 coordinates) int`: calcola e restituisce la distanza tra due punti `point1` e `point2`. Tempo: costante;
- `step(x int) int`: calcola la direzione in cui muoversi per poter andare avanti di un passo sul piano. Se `x` è maggiore di 0 ci si trova nel quadrante 1 o 4, quindi il movimento sarà verso destra o in altro (+1), se `x` è minore di 0 ci si trova nel quadrante 2 o 3, quindi il movimento sarà verso sinistra o in basso (-1). Nel caso in cui `x` sia 0 non si muove. Tempo: costante;
- `findNextX(p piano, start, goal coordinates) coordinates`: calcola la prossima posizione in cui il punto `start` si sposterà per raggiungere il punto `goal` muovendosi esclusivamente lungo l'asse `x` del piano. In particolare, una volta calcolata la direzione di movimento tramite lo step, viene iterata la slice degli ostacoli del piano in modo da poter controllare se nel percorso che va da un punto all'altro siano presenti ostacoli. In caso affermativo, la nuova posizione di `x` di `start` sarà quella immediatamente precedente all'ostacolo. Altrimenti, `start` potrà raggiungere la coordinata `x` di `goal` direttamente. Tempo: $\Theta(n)$, dove n è il numero totale di ostacoli presenti sul piano;

- `findNextY(p piano, start, goal coordinates) coordinates`: calcola la prossima posizione in cui il punto `start` si sposterà per raggiungere il punto `goal` muovendosi esclusivamente lungo l'asse `y` del piano. In particolare, una volta calcolata la direzione di movimento tramite lo step, viene iterata la slice degli ostacoli del piano in modo da poter controllare se nel percorso che va da un punto all'altro siano presenti ostacoli. In caso affermativo, la nuova posizione di `y` di `start` sarà quella immediatamente precedente all'ostacolo. Altrimenti, `start` potrà raggiungere la coordinata `y` di `goal` direttamente. Tempo: $\Theta(n)$, dove n è il numero totale di ostacoli presenti sul piano;
- `findPath(p piano, start coordinates, goal coordinates, visited map[coordinates]bool) bool`: implementa un algoritmo ricorsivo di ricerca percorso tra il punto iniziale `start` e il punto finale `goal` sul piano `p`. Il caso base della ricorsione è quando le coordinate del punto iniziale sono uguali a quelle del punto finale. Per poter controllare che la lunghezza del percorso sia uguale alla distanza tra i due punti, è bastato limitare il movimento a due direzioni. Ad esempio, se il punto iniziale si trova ad un'altezza minore e alla sinistra del punto di arrivo, il movimento effettuato dal punto sarà limitato solamente a destra e in alto. Lo sviluppo dell'algoritmo è stato ispirato a una DFS, ma adattato all'utilizzo su un piano. Il movimento viene eseguito una direzione alla volta e la nuova posizione per ogni asse viene calcolata utilizzando le funzioni `findNextX` e `findNextY`. Una volta calcolate quest'ultime, per ogni nuova posizione viene innanzitutto controllato che non sia già stata visitata: per fare ciò è stata utilizzata la mappa `visited`, che associa ad ogni coordinata un valore booleano; la mappa aggiorna le posizioni visitate all'inizio della funzione. Se la posizione non è stata visitata, viene effettuata una chiamata ricorsiva con la nuova posizione come punto di partenenza. Nel caso in cui la ricerca del percorso in quella determinata direzione non abbia avuto successo, è stato implementato un backtracking che torna indietro di un passo alla volta a partire dalla nuova posizione fino alla posizione precedente. Nel tornare indietro viene controllato che quel nuovo punto non sia già stato visitato e in caso affermativo viene nuovamente richiamata ricorsivamente la funzione con la nuova posizione. Nel caso neanche con il backtracking venga trovato un percorso per la posizione corrente e per le successive, vuol dire che non esiste un percorso. Tempo: $O(bh(n+b))$ oppure $O(bh(n+h))$ dove n è il numero di ostacoli totali, b è la base del rettangolo che si forma tra il punto iniziale e quello di arrivo e h è l'altezza del relativo rettangolo; $b+h$ equivale alla distanza tra i due punti. Lo spazio occupato dalla mappa è di $O(b \cdot h)$;
- `existsPath(p piano, point coordinates, name string)`: controlla che esista un percorso valido con la distanza pari alla distanza tra il punto `point` e le coordinate dell'automa `name`. Per fare ciò, è stata utilizzata la funzione `findPath` che restituisce un valore booleano. La funzione stampa "NO" se l'automa specificato non esiste, se il punto specificato si trova all'interno di un ostacolo (grazie all'utilizzo della funzione `pointIsInObstacle`), se il punto passato come parametro coincide al punto in cui si trova l'automa o se il percorso non esiste. Altrimenti stampa "SI". Tempo: $O(bh(n+b)+n)$ oppure $O(bh(n+h)+n)$ dove n è il numero di ostacoli totali, b è la base del rettangolo che si forma tra il punto iniziale e quello di arrivo e h è l'altezza del relativo rettangolo; $b+h$ equivale alla distanza tra i due punti;
- `recall(p piano, prefix string, point coordinates)`: sposta tutti gli automi con il prefisso specificato nel punto di richiamo `point` che hanno la distanza minima dal punto di richiamo e se il percorso tra i due punti esiste (utilizzando la funzione `findPath`). Per poter trovare tutti gli automi con prefisso `prefix` e con distanza minima dal punto di richiamo è stata usata una variabile per tenere traccia della distanza minima e una slice per tenere traccia di tutti i nomi di automi che hanno distanza minima. Infine è stata iterata proprio questa slice contenente solo gli automi interessati e sono state aggiornate le relative posizioni; di conseguenza viene aggiornata anche la mappa `numberOfAutomata` del piano decrementando il valore attuale di automi presenti nella vecchia posizione e, nel caso dovesse corrisponde a zero, viene cancellato il valore dalla mappa. Infine viene aggiornata la mappa `automata` con la nuova posizione e viene incrementato il numero di automi presenti nella nuova posizione per la mappa `numberOfAutomata`. Tempo: $O(nsbh(o+b))$ oppure $O(nsbh(o+h))$ dove n è il numero totali di automi, s è il prefisso della stringa `prefix`, b è la base del rettangolo che si forma tra il punto iniziale e quello di arrivo, h è l'altezza del relativo rettangolo e o è il numero di ostacoli totali;
- `esegui(p piano, s string)`: esegue le operazioni specificate nella stringa `s` passata come parametro.

4 Esempi di Esecuzione

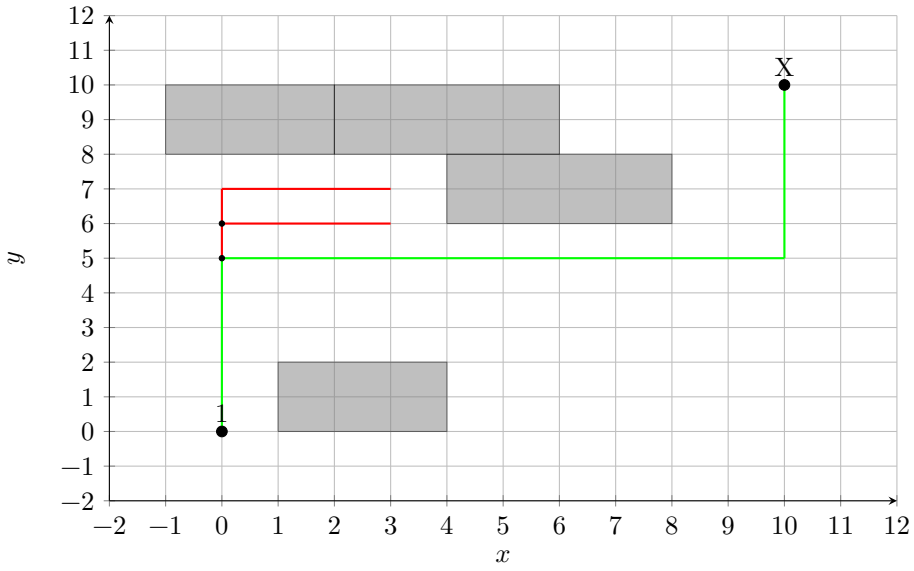
4.1 Backtracking

```
$ go run solution.go

c
a 0 0 1
o -1 8 2 10
o 1 0 4 2
o 4 6 8 8
o 2 8 6 10
e 10 10 1
f

output

SI
```



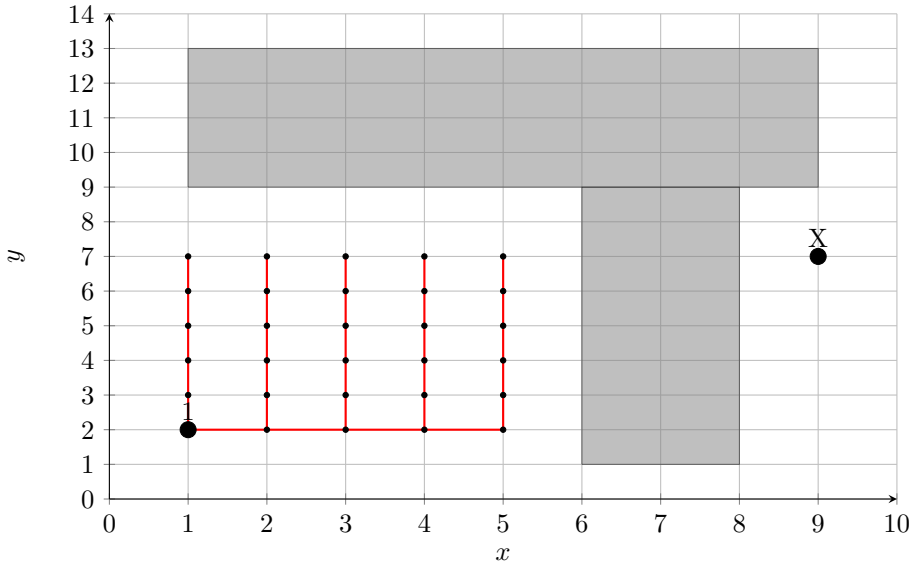
4.2 Nessun Percorso

```
$ go run solution.go

c
a 1 2 1
o 6 1 8 9
o 1 9 9 13
e 9 7 1
f

output

NO
```



Nel caso in cui questo input dovesse essere riprodotto su scala maggiore, i costi di esecuzione aumenterebbo notevolmente, in quanto il backtracking dovrà essere eseguito per ogni possibile percorso.

4.3 Ulteriori Esempi

```
$ go run solution.go

c
a 1 1 1
o 10000 -10000 10001 5000
e 20000 10000 1
f

output
SI
```

```
$ go run solution.go

c
a 2 2 101
a 2 8 00
a 6 8 11
a 12 4 10
o 1 3 5 7
r 10 7 1
s 3 4
S
f

output

O
(
101: 2,2
00: 2,8
```

```
11: 10,7
10: 10,7
)
[
(1,3)(5,7)
]
```

```
$ go run solution.go

c
a 5000 5000 0
a 10000 10000 1
a 7000 8000 10
a 15000 15000 11
o 6000 6000 9000 7000
o 12000 12000 14000 14000
o 8000 9000 8500 9500
e 9500 9500 1
e 4500 4500 0
r 7500 8500 10
p 1
s 5000 7000
f

output

SI
SI
(
1: 10000,10000
10: 7500,8500
11: 15000,15000
)
E
```