Laboratorio di Algoritmi e Strutture Dati
Progetto "Piastrelle digitali"

Angelica Ruiz Preciado - 12064A - Luglio 2024

1 Introduzione

Questo documento presenta le scelte fatte riguardo alla soluzione del problema "Piastrelle digitali". Dopo una breve presentazione della struttura del programma, un primo capitolo si centra sulle scelte fatte riguardo alla rappresentazione e la implementazione degli oggetti del programma, mentre un secondo è dedicato all'analisi dell'implementazione delle operazioni richieste.

2 Struttura del programma

2.1 La definizione dei tipi

La prima parte del codice è dedicata alla definizione dei tipi, è divisa in una sezione di definizione dei tipi esplicitamente richiesti, quindi il tipo piano, e una sezione di definizione di tutti gli altri tipi. Per rappresentare la configurazione del sistema è stato scelto di definire un tipo punto, equivalente ad una posizione nel piano, un tipo piastrella, un tipo regola (nel codice regolaType), un tipo condizione, utile a comporre le regole, e un tipo contenitore di regole (nel codice regole).

2.2 La definizione delle funzioni

La definizione delle funzioni è anche essa partizionata in modo da distinguere le funzioni di segnatura fissa, colora, spegni, regola, stato e stampa, da quelle strumentali alle operazioni richieste o alle prime funzioni, aggiungiCirconvicine (funzione di appoggio per la creazione delle piastrelle), blocco, contains e containsPointer (funzioni strumentali alla ricerca in array dinamici), bloccoOmog, visitaIntesita (funzione che gestisce la visita del grafo per le funzioni blocco e bloccoOmog), propaga, propagaBlocco, verificaRegola(funzione strumentale alle due funzioni di propagazione), ordina, mergeSort e merge (di cui ordina è una funzione wrapper), pista, lung, visitaPistaBreve (funzione che gestisce la visita del grafo per la funzione lung), esegui e checkInputLength(funzione di controllo per gli input).

3 Strutture dati del problema

3.1 Il piano

La rappresentazione del piano corrisponde ad una hash table che associa ad un punto la corrispondente piastrella purché questa sia accesa.

Posizione	**Piastrella**
(x,y)	Piastrella(x,y)

Table 1: Hash table che associa una posizione ad una piastrella.

Contemporaneamente la relazione tra le piastrelle circonvicine è rappresentata da un grafo nel quale l'insieme dei nodi rappresenta l'insieme delle piastrelle accese e l'insieme degli archi corrisponde alle relazioni di vicinanza tra piastrelle, cioè, la presenza di un arco tra due nodi implica che le due piastrelle corrispondenti sono circonvicine tra di loro, come rappresentato nella figura 1.

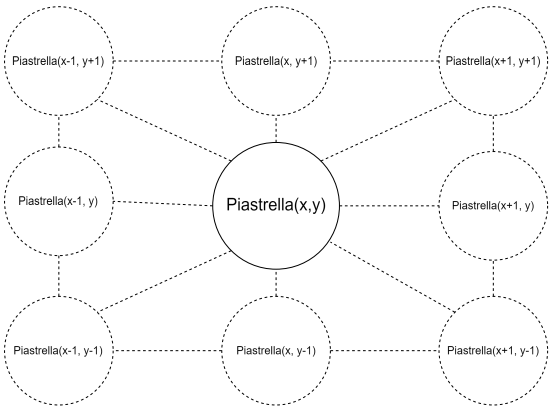

Figure 1: Grafo delle piastrelle circonvicine a Piastrella(x,y)

3.1.1 Implementazione del piano

L'implementazione è stata fatta attraverso una mappa o dizionario che associa ad una struttura tipo punto, composta di una coordinata x e una coordinata y, un puntatore ad una struttura tipo piastrella; dentro alla mappa sono presenti tutte e sole le piastrelle accese. Questa implementazione presenta dei vantaggi per quanto riguarda gli accessi ad una piastrella di date coordinate, che vengono fatti in tempo costante, O(1). Per quanto riguarda il grafo l'implementazione è gestita dalla struttura delle piastrelle (tratta nella sezione 3.2). Oltre alla mappa, il piano contiene anche un puntatore ad una struttura tipo regole contenente le regole di propagazione inserite nel sistema.

3.2 Piastrelle

Le piastrelle sono rappresentate come delle strutture che contengono la posizione che le identifica, il loro colore, la loro intensità, e la corrispondente lista dei nodi adiacenti, cioè, delle piastrelle circonvicine.

3.2.1 Implementazione delle piastrelle

Ogni piastrella è implementata tramite una struct, con un campo posizione di tipo punto, un campo colore di tipo string, un campo intensità di tipo intero e un puntatore a un array dinamico di puntatori a piastrella, che corrisponde alla lista di adiacenza di quella piastrella. La lista di adiacenza permette di svolgere le principali operazioni in tempo O(n) con n nodi, mentre per quanto riguarda lo spazio di memorizzazione ogni array dinamico ha grandezza $m \leq 8$.

3.3 Regole

Le singole regole sono rappresentate da delle strutture che memorizzano il colore risultante dalla loro applicazione, il loro consumo e le condizioni necessarie per l'applicazione. Per quanto riguarda l'insieme delle regole inserite nel sistema questo è rappresentato da una coda che inizialmente mantiene l'ordine d'inserimento e che può essere riorganizzata come una coda di priorità sul parametro consumo.

3.3.1 Implementazione delle regole

L'implementazione di ogni regola è fatta tramite il tipo struct regolaType che conta con un campo beta di tipo string che rappresenta il colore risultante, un campo consumo di tipo int, e un campo condizioni che corrisponde ad un array dinamico di elementi di tipo condizione, composto da un campo colore di tipo string e un campo k di tipo intero che memorizza la quantità di piastrelle di quel colore necessarie per applicare la regola. Il controllo delle condizioni di una

regola avviene dunque per scorrimento di questo array dinamico di lunghezza $l \leq 8$ sul intorno di una piastrella, composto da un massimo di 8 piastrelle.

La coda delle regole del sistema, invece, viene implementata da un altro array dinamico di puntatori a elementi tipo regolaType, che favorisce particolarmente l'inserimento, O(1), mentre la ricerca viene fatta in modo lineare in tempo O(n) con n regole. Per quanto riguarda la conversione in coda di priorità viene applicato un mergeSort sul campo consumo di ogni regola del array, come descritto nella sezione operazioni; questo, però, non cambia la gestione degli inserimenti né della ricerca, che continuano a funzionare come per la coda senza priorità.

4 Operazioni del problema

4.1 Funzioni principali

4.1.1 Colora

```
func colora(p piano, x int, y int, alpha string, i int)
```

La funzione colora parte da un controllo sulla piastrella associata nel piano al punto di coordinate x,y data in input, se trova un riscontro già presente nel piano ne cambia il colore e l'intensità ai valori dati in input, mentre se il punto dato non esiste nella mappa del piano questo viene aggiunto creando la corrispondente piastrella. Questa piastrella prende i parametri dati in input per quanto riguarda il colore e l'intensità, dopodiché vengono aggiunte le sue piastrelle circonvicine tramite un apposita funzione (spiegata nella sezione di funzioni secondarie).

In entrambi casi l'accesso alla posizione nel piano ha un costo unitario per quanto riguarda il tempo; nel caso di creazione della piastrella bisogna considerare oltre all'accesso la visita al grafo per la creazione della sua lista di adiacenza (come si vedrà, questa ha complessità $\theta(1)$).

4.1.2 Spegni

```
func spegni(p piano, x int, y int)
```

La funzione spegni è implementata in modo da cancellare dalla mappa dei punti l'elemento associato al punto di coordinate x,y date in input, questo avviene tramite la funzione predefinita delete. L'accesso all'elemento avviene con complessità O(1) grazie alla struttura tipo dizionario. Ulteriormente la funzione cancella i riferimenti alla piastrella dalla lista di adiacenza di tutte le sue piastrelle circonvicine. Per fare questo itera sulle piastrelle circonvicine, che sono al massimo otto, e per ognuna cerca la posizione della piastrella da cancellare, tra un massimo di otto piastrelle, per rimuovere l'elemento tramite subslicing. Questa operazione impiega un tempo O(1) e non occupa memoria aggiuntiva.

4.1.3 Regola

```
func regola(p piano, r string)
```

La funzione regola aggiunge la regola data in input a traverso la string r, facendola diventare l'ultima regola nella coda. Si parte dal dividere l'input sulle sue parole (cioè, seguendo gli spazzi) e creando un array dinamico ausiliario, in seguito crea la regola con consumo zero e beta uguale al colore risultante fornito, per poi aggiungere le condizioni della regola con un ciclo che svolge n inserimenti totali per n condizioni della regola. La complessità, dunque, è O(n), ma sapendo che la quantità di condizioni è limitata da un massimo di 8 condizioni per regola si può considerare O(1). Durante il ciclo di creazione della regola si utilizza anche una variabile intera per controllare che la quantità di piastrelle coinvolte dalle condizioni della regola non superi le 8 piastrelle che conformano un intorno; nel caso questo avvenga viene stampato un messaggio

di errore e la funzione ritorna il flusso di controllo senza aggiungere alla coda delle regole il puntatore alla regola.

4.1.4 Stato

```
func stato(p piano, x int, y int) (string, int)
```

La funzione stato stampa e restituisce il colore e l'intensità della piastrella associata al punto di coordinate x,y nel piano tramite un accesso al corrispondente elemento nella mappa del piano, se questo esiste, altrimenti non fa nulla. Impiega dunque una quantità di tempo unitaria (O(1)) e spazio aggiuntivo nullo.

4.1.5 Stampa

```
func stampa(p piano)
```

La funzione stampa svolge due cicli annidati per stampare tutte le regole inserite fin'ora nel sistema nell'ordine attuale. Il primo ciclo scandisce le regole linearmente mentre il ciclo interno scandisce le condizioni della singola regola, per cui svolge al massimo 8 iterazione, ottenendo così una complessità pari a O(8n) = O(n) con n regole.

4.1.6 Blocco e blocco omogeneo

Queste due funzioni si basano sul concetto di blocco, rappresentato da una componente connessa del grafo delle piastrelle sulla quale si può imporre il vincolo del colore per ottenere un blocco omogeneo. Per trovare l'intensità del blocco o del blocco omogeneo contenente una data piastrella basta svolgere una visita, in ampiezza o in profondità indifferentemente, partendo dalla piastrella indicata e percorrendo tutti i nodi del sottografo connesso corrispondente.

```
func blocco(p piano, x int, y int)
```

La funzione blocco utilizza la funzione visitaIntensita per calcolare l'intensità del blocco a cui appartiene la Piastrella(x,y). Infatti, questa funzione si incarica di passare in input il piano e le coordinate della piastrella accompagnate da una variabile booleana settata a false che indica che la visita non deve essere limitata dal colore in quanto il blocco non è segnato come omogeneo. La complessità di questa funzione corrisponde a quella della funzione visitaIntensita, calcolata nella sezione 4.2.

```
func bloccoOmog(p piano, x int, y int)
```

Questa funzione, analogamente alla funzione blocco, funziona da wrapper per la funzione visitaIntensita, a cui passa come parametri il piano, le coordinate della piastrella, e una variabile booleana settata a true per indicare che la visita deve tener conto del colore delle piastrelle in modo da esplorare unicamente nodi che corrispondono a piatrelle dello stesso colore e che dunque compongono un blocco omogeneo. Anche in questo caso la complessità corrisponde a quella della funzione secondaria.

4.1.7 Propaga

```
func propaga(p piano, x int, y int)
```

La funzione propaga parte da un controllo sul fatto che la piastrella di posizione data in input sia accesa o meno, questo in quanto una piastrella spenta deve essere almeno momentaneamente accesa per svolgere il controllo delle condizioni sulle sue circonvicine, alla fine della funzione se

la piastrella era originalmente spenta e non incontra una regola applicabile questa deve tornare ad essere spenta.

La funzione svolge un ciclo sull'insieme delle regole, impiegando O(n) con n regole in quanto l'acceso viene fatto ad una struttura indicizzata, e su ogni regola applica una funzione che svolge la verifica di applicabilità e che, come si vedrà, svolge al massimo 72 confronti, e impiega un tempo O(1).

4.1.8 Propaga blocco

```
func propagaBlocco(p piano, x int, y int)
```

Analogamente alla funzione propaga, la funzione propaga blocco controlla che la piastrella sia accesa, ma diversamente dalla prima questa ritorna senza svolgere nessuna operazione se trova una piastrella spenta in quanto si considera che la piastrella spenta non appartenga a nessun blocco.

Se la piastrella è accesa la funzione svolge una esplorazione in ampiezza nella quale per ogni piastrella del blocco scorre la coda delle regole applicando la funzione di verifica e quando trova una regola applicabile utilizza una struttura di supporto, un dizionario che contiene una coppia fatta dal puntatore alla piastrella sulla quale si deve applicare la regola e il puntatore alla regola che si deve applicare. Finito il ciclo sul blocco inizia un ciclo sul dizionario appena popolato che applica i cambiamenti lì registrati. In questo modo la funzione svolge O(n) iterazioni con n piastrelle nel sistema, e per ognuna svolge O(m) con m regole la funzione di verifica che lavora in tempo costante, per una complessità totale di O(n*m), più O(n) iterazioni di applicazione delle regole, risultando in un tempo O(n*m).

4.1.9 Ordina, merge e merge sort

```
func ordina(p piano)
```

La funzione ordina funziona come un wrapper per la funzione mergeSort. Si incarica di passare in input le regole da ordinare e di cambiare l'insieme delle regole del sistema con quelle risultanti dal mergeSort. L'ordinamento prende una complessità temporale di $O(n \log n)$.

```
func mergeSort(arr []*regolaType) regole
```

Per eseguire il sort, la funzione mergeSort divide a metà la slice da ordinare, questo non costruisce strutture secondarie dato che il linguaggio gestisce queste situazioni (subslicing) tramite i puntatori. Dopo la divisione, la funzione chiama se stessa su ogni metà e restituisce il risultato della funzione merge sul risultato di queste due chiamate.

```
func merge(left, right []*regolaType) []*regolaType
```

La funzione merge ordina gli elementi delle due slice passate in input restituendo una nuova slice fatta dagli elementi tipo regola di entrambe slice ordinati in modo non decrescente sul parametro consumo. Per fare questo svolge un ciclo nel quale visita contemporaneamente un elemento di ogni metà confrontandoli tra di loro e aggiungendo al risultato il minore dei due, poi portando l'altro al prossimo confronto. Dopo confrontare tutti gli elementi di una metà aggiunge gli elementi rimanenti al risultato (supponendo che questi siano già stati ordinati da una chiamata precendente della funzione mergeSort).

4.1.10 Pista

```
func pista(p piano, x1, y1 int, s []string)
```

Questa funzione verifica se la sequenza di direzioni data in input nel array di string s definisce una pista partendo dalla Piastrella(x1,y1), cioè, se la sequenza di direzioni corrisponde a una sequenza di archi. In caso positivo stampa la pista come sequenza di piastrelle percorse. Per farlo, la funzione parte dal controllo sulla Piastrella(x,y), che deve essere accesa, e svolge un ciclo sul array contenente le direzioni, nel quale ogni iterazione memorizza le coordinate della seguente piastrella e controlla che anche questa sia accesa, memorizzando in un array ausiliario le coordinate della piastrella controllata. Se durante questo ciclo si va incontro ad una piastrella spenta, la sequenza non definisce una pista e la funzione ritorna senza fare niente, altrimenti un secondo ciclo stampa la sequenza di piastrelle le cui coordinate sono state memorizzate nel array ausiliario. Questa funzione svolge un massimo di n+1 confronti per n direzioni contenute nel array dato in input, insieme a n+1 iterazioni del ciclo di stampa, per cui la sua complessità temporale è O(n).

4.1.11 Lunghezza

```
func lung(p piano, x1, y1, x2, y2 int) int
```

Questa funzione cerca la pista più breve tra le due piastrelle di coordinate fornite in input, cioè, il percorso che collega i due nodi corrispondenti composto dalla minor quantità di nodi possibile. Parte da controllare che entrambe le piastrelle siano accese e in caso positivo procede a invocare la funzione ausiliaria visitaPistaBreve. La funzione ausiliare svolge una visita in ampiezza partendo dalla Piastrella(x1,y1) attraverso la quale trova la distanza, intesa come numero di archi da attraversare, tra una piastrella e tutte le altre, fermandosi quando visita la seconda piastrella data in input. Da questa chiamata si ottiene la distanza tra le due piastrelle, oppure -1 se queste sono scollegate. La funzione lung esegue un controllo sul valore ottenuto e, se le due piastrelle sono collegate, incrementa il risultato di uno per contare anche la piastrella di partenza, in quanto anche questa fa parte della pista, per poi stamparlo e restituirlo. Se le piastrelle non sono collegate viene dato come risultato -1 e non viene stampato niente. La complessità di questa funzione è pari a quella della funzione ausiliaria, che come si vedrà nella sezione 4.2 è pari a O(V+E) con V nodi ed E archi presenti nel grafo.

4.2 Funzioni secondarie

4.2.1 Aggiungi piastrelle circonvicine

```
func (p *piano) aggiungiCirconvicine(x, y int)
```

Questa funzione è strumentale alla colorazione delle piastrelle, permette di riempire il campo delle piastrelle circonvicine svolgendo una visita al grafo per ottenere i puntatori delle piastrelle circonvicine accese. Per fare questa visita la funzione è composta da due cicli annidati sui valori (-1, 0 , 1), in modo da interrogare la mappa un massimo di 9 volte, ottenendo una complessità temporale O(1).

4.2.2 Visita sull'intensità

```
func visitaIntensita(p piano, x int, y int, omogeneo bool) int
```

La funzione visitaIntensita svolge una visita in ampiezza del grafo delle piastrelle, partendo dalla piastrella di coordinate date in input e considerando che la variabile booleana omogeneo indica se le piastrelle da visitare devono essere unicamente quelle che coincidono in colore con quella di partenza o meno. Ogni volta che viene visitato un nodo si aggiunge l'intensità della corrispondente piastrella alla variabile da ritornare.

Per fare questo la funzione impiega un tempo O(V+E), con V piastrelle (nodi) che fanno parte del blocco ed E archi. La funzione impiega delle strutture di supporto, degli array dinamici,

per memorizzare i nodi osservati e i nodi da osservare; queste strutture costano uno spazio addizionale di grandezza O(n). Su queste strutture viene applicata la funzione contains, di complessità temporale O(n), che svolge un controllo sul fatto che un nodo sia contenuto o meno dentro a questi array. La complessità totale della funzione è O(n), e come risultato si ottiene la somma delle intensità di tutte le piastrelle visitate, cioè di tutte le piastrelle che compongono il blocco o il blocco omogeneo.

4.2.3 Contains e Contains Pointer

```
func contains(p []piastrella, e piastrella) bool
```

Questa funzione esegue un ciclo sugli elementi della slice data in input e ritorna true se trova la piastrella data in input. Questa funzione impiega tempo O(n) con n piastrelle perché svolge al massimo n confronti, spesso quando viene applicata da altre funzioni la grandezza dell'input è limitata come nel caso delle piastrelle circonvicine che possono essere al massimo 8.

```
func containsPointer(p []*piastrella, e *piastrella) bool
```

Questa funzione è del tutto analoga alla funzione contains ma prende in input una slice di puntatori tra cui cerca uno, questo semplicemente per facilitare il funzionamento di altre funzioni.

4.2.4 Verifica regola

```
func VerificaRegola(p piano, v piastrella, regola *regolaType) bool
```

Questa funzione svolge la verifica di applicabilità di una regola su una piastrella data in input. Questo viene fatto attraverso un primo ciclo sulla lista di piastrelle circonvicine, cioè, la lista di nodi adiacenti, della piastrella data in input che, con un massimo di 8 iterazioni, crea una struttura di supporto di tipo mappa nella quale si associa al colore di una circonvicina la sua frequenza in questo insieme, cioè la quantità di circonvicine di quel colore. Dopo di che, si inizializza una variabile flag a true, e un secondo ciclo itera sulle condizioni della regola data in input, con un massimo di 8 iterazioni anche questa, e verifica che al colore indicato nella condizione sia associato nella struttura di supporto almeno il numero necessario (indicato nella condizione) di frequenze per applicare la regola. Se non ci sono abbastanza circonvicine del colore corrispondente alla condizione, la regola non è applicabile, per cui viene settata a false la variabile booleana che funziona da flag e si conclude il ciclo. Se, invece, tutte le condizioni sono soddisfate, si arriva alla fine del ciclo senza cambiare il valore della variabile flag, che dunque viene data in output con il valore true indicando che la regola è applicabile su quella piastrella.

4.2.5 Visita sulla pista più breve

```
func visitaPistaBreve(p piano, x1, y1, x2, y2 int) int
```

La funzione visitaPistaBreve è ausiliare alla funzione lung, svolge una visita in ampiezza del grafo delle piastrelle per trovare, se esiste, il percorso composto da meno archi tra due nodi, cioè, la pista più corta tra due piastrelle. Riceve in input il piano e le coordinate della piastrella di partenza e della piastrella di arrivo e ritorna un intero con la lunghezza della pista che collega queste due piastrelle oppure un valore che rappresenta il fatto che le due piastrelle non sono collegate da nessuna pista (il valore -1). Prima di svolgere la visita la funzione controlla che le due piastrelle coincidano, se è così finisce dando come output il valore 0, altrimenti inizializza le strutture di supporto. Queste strutture corrispondono a una mappa che associa ad ogni punto presente nel piano un booleano che rappresenta se la corrispondente piastrella è stata visitata o meno, una seconda mappa che associa ad ogni punto un intero che rappresenta la distanza dalla piastrella di partenza alla piastrella corrispondente al punto, e un array dinamico che

implementa una coda nella quale vengono salvati i punti da visitare. Le due mappe vengono inizializzate tramite un ciclo che assegna ad ogni punto esistente nel piano una distanza pari a -1, rappresentando che inizialmente vengono considerate scollegate dalla piastrella di partenza, e il valore falso per la mappa delle visite. Inoltre, la distanza alla piastrella di partenza viene assegnata come 0. L'array viene inizializzato caricandone la piastrella di partenza.

La visita avviene tramite un ciclo sul array di punti da visitare, del quale verranno eseguite un massimo di n iterazioni con n nodi esistenti nel grafo. In ogni iterazione si prende il primo elemento del array, cioè, si esegue un'operazione dequeue sulla coda per prendere il suo primo elemento, e lo si confronta con le coordinate della piastrella di arrivo. Se corrisponde alla piastrella di arrivo la funzione emette in output l'intero ad essa associato nella mappa delle distanze, altrimenti si svolge un ciclo interno. Il ciclo interno itera sui nodi adiacenti a quello corrente ottenuto dalla dequeue, se trova un nodo non visitato si assegna come distanza a questo nodo la distanza dal nodo di partenza al nodo corrente più uno, dopodiché si aggiorna anche la mappa dei visitati segnando il nodo adiacente come visitato, e lo si inserisce alla fine del array (cioè, svolge la funzione enqueue). Questo ciclo interno viene eseguito al massimo 8 volte a causa del limite delle piastrelle circonvicine.

Se viene svuotata la coda, essendo che la funzione non ha incontrato la piastrella di arrivo, viene ritornato -1 come indicatore del fatto che i due nodi del grafo sono sconnessi, e dunque non esiste una pista che colleghi le due piastrelle. Questa funzione impiega uno spazio O(n) per n piastrelle nel piano, questo in quanto le tre strutture di supporto possono aver grandezza n nel peggior caso. Anche il tempo è O(n), risultato della somma dell'inizializzazione, che svolge n iterazioni, e il ciclo principale, che esegue altre n iterazioni in cui il ciclo interno svolge al massimo 8 iterazioni.

4.2.6 Esegui e check input length

```
func checkInputLength(op []string, length int) bool
```

Data una slice di tipo string e un numero che rappresenta la lunghezza attesa, questa funzione fa un semplice controllo sulla lunghezza e restituisce false nel caso l'input ne differisca, altrimenti true. Questa funzione è stata inclusa per facilitare il testing e debuging.

```
func esegui(p piano, s string)
```

Questa funzione gestisce il menu delle operazioni, indirizzando ai comandi dati in input le corrispondenti funzioni, verificando la correttezza degli input e manipolandoli per dare il formato giusto ad ogni funzione.

5 Esempi di esecuzione

5.1 Test 1, 2 3

Per i primi test è stata proiettata una situazione iniziale rappresentata nel diagramma della figura 2.

Le regole inserite inizialmente sono

$$1c + 1r \to v \tag{1}$$

$$1v + 1b \to l \tag{2}$$

$$2r \to s \tag{3}$$

$$1b + 1r \to v \tag{4}$$

$$2b + 1s \to g \tag{5}$$

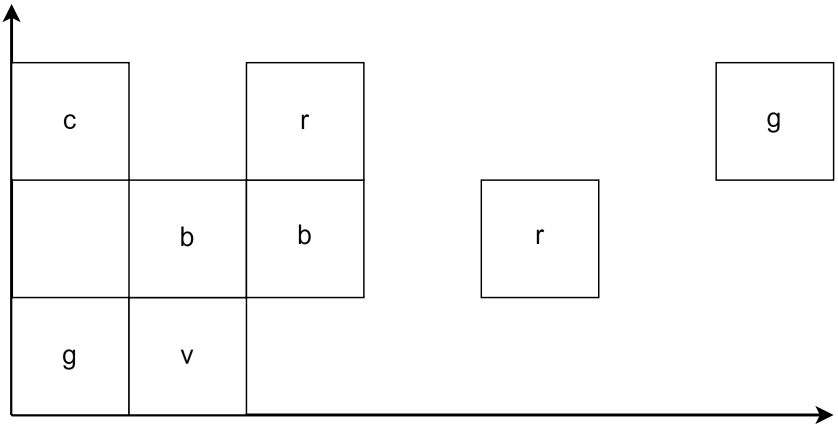

Figure 2: Piano della situazione iniziale per i primi test

Il primo test è indirizzato alla prova di correttezza della funzione propaga e propaga blocco. Dopo l'inserimento delle regole e la colorazione delle piastrelle, chiama la funzione propaga su Piastrella(0,0) e su Piastrella(3,2) e ne controlla lo stato risultante, la prima cambia colore ad l in quanto gli viene applicata la regola (2), e la seconda, originalmente spenta, ad s dopo applicare la regola (3). Il test procede propagando sul blocco della Piastrella(0,0) e interrogando tutte le piastrelle del blocco. Da queste interrogazioni si vedono le variazioni sulla Piastrella(1,1) al colore v per la regola (1), sulla Piastrella(2,1) al colore v per la regola (4) e sulla Piastrella(2,2) al colore g per la regola (5); le altre piastrelle rimangono invariate, si noti che la Piastrella(3,2) subisce la propagazione per via della regola (3), che però risulta nel colore s che aveva già prima. Il test prova anche che l'applicazione dell'operazione propaga blocco su una piastrella isolata, la Piastrella(6,2) e su una piastrella spenta, la Piastrella(6,0), non ha alcun effetto sul sistema.

Il secondo test vuole verificare il funzionamento dell'operazione ordina. Questo test è analogo al primo per quanto riguarda la situazione iniziale e le prime propagazioni singole, dopo le quali viene applicata l'operazione ordina che cambia la coda delle regole ottenendo la sequenza (1), (4), (5), (2), (3). A questo punto si fa la propagazione sul blocco della Piastrella(0,0) e si ottiene un risultato identico a quello del primo test tranne per la Piastrella(2,3) a cui ora si applica la regola (4) ottenendo il colore v. Per concludere il test si chiama la funzione che stampa le regole che rispecchia il nuovo ordine appena illustrato.

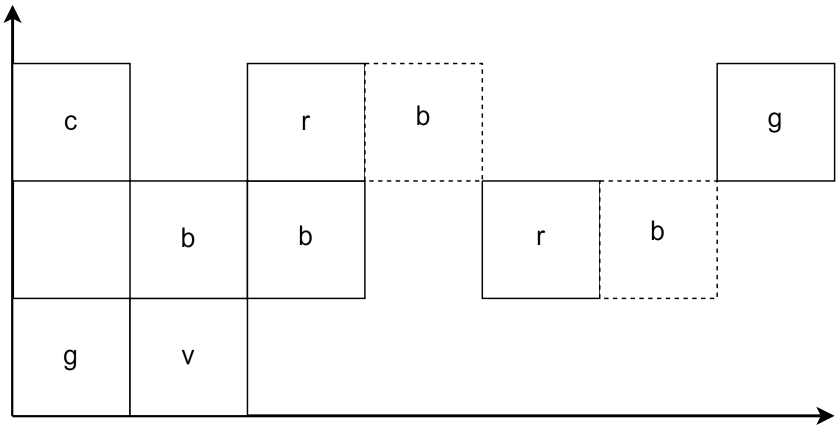

Figure 3: Piano della situazione iniziale per il terzo test

Il terzo test è dedicato alle funzioni di blocco e blocco omogeneo. La situazione iniziale, riportata nella figura 3, è simile a quella dei primi due test ma differisce da questa per quanto riguarda le regole, che in questo caso non vengono inserite. Le piastrelle sono state poste con luminosità pari a 1, facendo in modo che la funzione blocco restituisca la quantità di piastrelle che fanno parte del blocco e la funzione blocco omogeneo la quantità di piastrelle che fanno

parte del blocco omogeneo, e che dunque sono dello stesso colore. Il test parte chiamando la funzione blocco sulla Piastrella(1,1), da cui ottiene l'intensità del blocco che è pari a 6, dopo di che, viene chiamata la funzione blocco omogeneo e si ottiene 2. Il test aggiunge la Piastrella(3,2) usando la funzione colora con b come colore e intensità 1 come per le altre, come riportato nella figura 3 con linee tratteggiate. A questo punto vengono chiamate nuovamente le operazioni di calcolo dell'intensità del blocco e del blocco omogeneo trovando un blocco d'intensità 8 e un blocco omogeneo di intensità 3, visto che questa piastrella appena inserita è dello stesso colore della Piastrella(1,1) e collega il suo blocco alla Piastrella(4,1). Per collegare questo blocco appena creato alla Piastrella(6,2) si esegue l'operazione colora sulla Piastrella(5,1), anche questa riportata in schema con linee tratteggiate, con intensità 1 e colore b, e si eseguono nuovamente le due funzioni di blocco, ottenendo per il blocco un intensità complessiva di 10, pari alla quantità di piastrelle che formano il blocco, e per il blocco omogeneo un'intensità ancora pari a 3, in quanto la piastrella appena inserita permette di collegare tutte le piastrelle del sistema ma non è circonvicina a nessuna delle piastrelle del blocco omogeneo della Piastrella(1,1).

5.2 Test 4 e 5

I test 4 e 5 condividono una situazione iniziale diversa da quella dei primi test. Le piastrelle inizialmente accese sono quelle mostrate nello schema della figura 4, mentre le regole inserite sono solo le due riportate qui sotto e numerate (6) e (7).

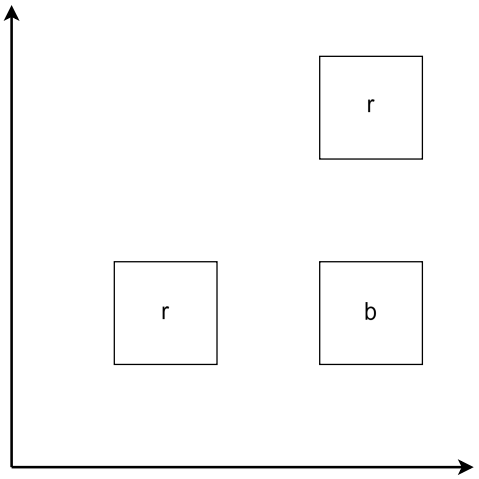

Figure 4: Piano della situazione iniziale per il quarto e quinto test

$$1r + 1b \rightarrow g \tag{6}$$

$$1r + 1g \rightarrow b \tag{7}$$

Il quarto test è puntato a provare la correttezza della funzione spegni, utilizzando la funzione propaga. A questo fine si parte dal propagare il colore sulla Piastrella(2,1), interrogando posteriormente il sistema risulta evidente che questa viene colorata con il colore g per via della regola (6). Si propaga poi sulla Piastrella(2,2) che applica la stessa regola e ottiene il colore g. Il test procede spegnendo la Piastrella(3,1) e la Piastrella(2,2), sulla quale si propaga un'altra volta; questa volta la piastrella viene colorata del colore b applicando la regola (7).

Il quinto test vuole provare tutte le operazioni del programma. Partendo dalla stessa situazione iniziale del test 4, si propaga sulla Piastrella(2,1) che prende come colore g applicando la regola (6), si spegne la Piastrella(3,1), si propaga sulla Piastrella(2,2) che ottiene il colore b per la regola (7), e poi anche sulla Piastrella(1,2) che applica la regola (6) e prende il colore g. A questo punto si esamina il blocco della Piastrella(2,1), e si ottiene come valore di intensità 5, e poi il suo blocco omogeneo, che ha intensità 2. A questo punto si ordinano le regole usando la funzione ordina, e si propaga nuovamente sulla Piastrella(1,2) che questa volta ottiene il colore

b grazie al cambiamento di ordine delle regole. Propagando invece sulla Piastrella(3,1) la regola che si applica è la (6) e il colore propagato è g. Il test va avanti colorando la Piastrella(3,1) del colore b e la Piastrella(3,1) del colore g, per poi propagare sul blocco della Piastrella(3,0) e interrogare le piastrelle alterate da questa propagazione. Le piastrelle alterate sono la Piastrella(2,1), la Piastrella(2,2), e la Piastrella(1,2) che diventano tutte del colore b grazie alla regola (7). Per concludere si interroga il sistema sull'intensità del blocco omogeneo della Piastrella(3,1) che ha ottenuto il valore 4 dopo l'ultima propagazione.

5.3 Test 6

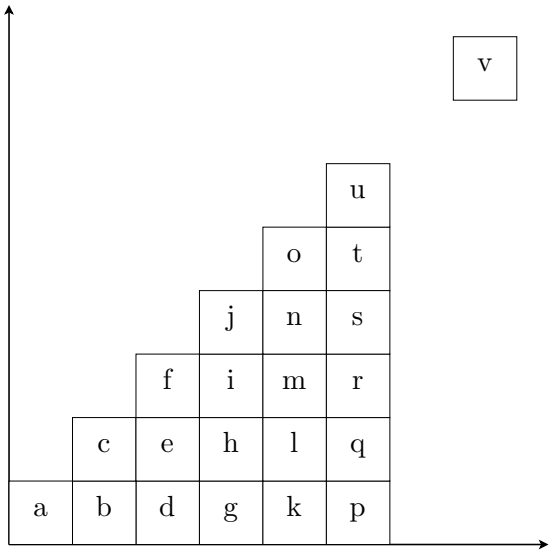

Figure 5: Piano della situazione iniziale per il sesto test

Il sesto test parte dalla situazione iniziale riportata nello schema della figura 4. Questo test vuole dimostrare la correttezza delle operazioni pista e lung. Prima colora le piastrelle coinvolte, poi il test esegue la funzione lung diverse volte per trovare la lunghezza della pista che collega la Piastrella(0,0) alla Piastrella(5,5), poi la lunghezza della pista tra la Piastrella(5,0) e la Piastrella(0,0), dimostrando che l'ordine delle piastrelle non altera il risultato, dopo di che cerca la pista dalla Piastrella(0,0) alla Piastrella(5,0), mostrando che l'operazione funziona correttamente tra piastrelle connesse. Fatto ciò, viene cercata la pista tra la Piastrella(0,0) e la Piastrella(0,7), quella tra la Piastrella(0,1) e la Piastrella(1,1), quella tra la Piastrella(0,0) e la Piastrella(0,1), e quella tra la Piastrella(0,1) e la Piastrella(1,2), per mostrare il comportamento della funzione se le piastrelle sono sconnesse o se una delle due piastrelle è spenta. Viene trovata anche la lunghezza della pista tra la Piastrella(0,0) e la Piastrella(5,3) e di quella tra la Piastrella(0,0) e la Piastrella(3,1) come prova di funzionamento con piste più corte.

Una volta provata l'operazione lung, il test svolge l'operazione pista partendo dalla Piastrella(0,0) e svolgendo delle sequenze prima più corte e poi più lunghe, tutte definite tranne per l'ultima. La stessa operazione viene provata partendo dalla Piastrella(5,5) su una pista definita, e dalla Piastrella(3,2) su una pista non definita; poi l'operazione viene chiamata su due piste definite, una dalla Piastrella(5,0) e una dalla Piastrella(0,0). Il test conclude con una chiamata alla operazione dando in input una piastrella spenta.