ALGRAPH

Algraph, progetto nato dal lavoro di Barbone Roberto, Iazzetta Luigi Ferdinando e Monacchi Massimo, prende forma dal famoso algoritmo di Edsger Dijkstra per la risoluzione della ricerca dei cammini minimi in un grafo. A questo proposito si è deciso di creare un programma che permettesse la generazione immediata di grafi con la consecutiva illustrazione step-by-step di come l'algoritmo funzioni nel dettaglio.
Al fine di realizzare tale programma ci siamo avvalsi dell'utilizzo dell'applicazione Java FX, fondamentale per la fase dimostrativa del progetto.
Entrando nello specifico, Algraph si suddivide in quattro diversi packages: DataStructure, Graphic, EventMouse e Dijkstra, e separato da essi, si trova ovviamente la classe Main, fautrice del coordinamento di tutte le altre classi.

Partendo da DataStructure, possiamo contare quattro classi al suo interno: *Vertex*, *Edge*, *Graph* e *PriorityQueue*. Per agevolare la comprensione e l'utilizzo dei metodi si è deciso di operare con le funzioni get e set.
Vertex rappresenta i singoli nodi dei possibili grafi, composti quindi tutti da una stringa identificativa, contenuta nel costruttore, la cui posizione viene settata della funzione **setText**, dal cerchio fisico che viene creato (**setCircle**), da un colore che varia in base alla funzionalità del nodo all'interno dell'algoritmo, e infine un testo che indica il valore del peso del nodo stesso, che può quindi essere compreso fra 0 e infinito(**setTextDijkstra**).
In seguito troviamo la classe *Edge*, la quale opera attingendo da quella precedente, infatti tra i suoi campi rientrano due Vertex, uno di partenza e uno di arrivo. Per quanto concerne le sue funzioni compaiono quelle necessarie a settare il peso dell'arco (**setPeso**) e della la linea, formata da quattro valori, le coordinate iniziali e quelle finali (**setLine**); in aggiunta la funzione **setText** permette di stampare sopra la linea il testo indicante i due nodi collegati da una freccia e il peso dell'arco che li unisce e infine **getHash** ritorna il nome del nodo di partenza u e di quello di arrivo v, e tale metodo serve a controllare che non ci siano archi con lo stesso nome al momento della loro creazione.
Graph ingloba le due classi appena descritte creando con esse un hashmap, in cui le chiavi sono i Vertex e i valori collegati alle chiavi gli insiemi di archi: tale scelta deriva dall'impossibilità di avere archi con lo stesso nome all'interno di un grafo, di conseguenza la scelta del set all'interno dell hashmap si rivela ottimale. Altro campo privato di Graph è un booleano, stateDijkstra, la cui unica funzionalità è di controllare se l'algoritmo è attivo o meno. Tra i metodi è possibile vedere quelli fondamentali di aggiunta e rimozione rispettivamente di un nodo e di un arco, **checkIfEdgeIsPresent** che prendendo in input un arco controlla prima se il set di archi non è vuoto e successivamente li scorre tutti per verificare se in quel set è già presente l'arco passato, **getInvertedEdge** lavora analogamente alla funzione prima verificando se è presente l'arco inverso, il modo da poter permettere la sua creazione e per ultimo **getFromSet** il quale si assicura che data una stringa sia presente il vertice ad essa corrispondente, e di conseguenza lo ritorna.
L'ultima classe del package, *PriorityQueue*, si occupa di creare una coda di priorità che verrà poi disegnata sulla classe Dijkstra. Tale coda è composta da un arraylist di vertici, e su questo opera la funzione deleteMin, la quale scorrendolo con un for cancella il nodo con il peso minore e lo restituisce in modo da trovare gli archi ad esso connessi.

Continuando l'analisi del programma, troviamo il package Graphic, al cui interno sono presenti le classi *ButtonLayout* e *LineAndTextLayout,* responsabili della creazione dell'aspetto grafico del programma.
La prima, *ButtonLayout*, si caratterizza di otto bottoni:

- Open file : crea il grafico da file di testo caricato dall'utente, prendendo i nodi e disponendoli circolarmente secondo una precisa logica: il numero di nodi totale fa da divisore ai 360° del cerchio e del risultato si trova il seno e il coseno, moltiplicandoli per il raggio in modo da trovare le coordinate X e Y. Tale bottone prende in input solamente file .txt e sono impostati in modo tale che i nodi, formati da una stringa di un solo char, vengano letti finché la funzione .**read** non legge la configurazione dell'arco, ossia una linea con char del vertice di partenza, freccia e char del vertice di arrivo e un'altra con indicato il peso dell'arco.
- Save file: permette di salvare il proprio grafo. Sia per Open file che per Save file al momento della pressione si apre una shell con le directory da cui prendere o su cui salvare il grafo.
- Delete: Pulisce il pannello del grafo, graficamente e strutturalmente a livello di codice. Graficamente con un for elimina prima gli archi, poi con lo stesso costrutto i nodi, mentre strutturalmente si utilizza la funzione **clear** la quale cancella l'oggetto G della classe *Graph.*
- Random: apre un pannello che chiede all'utente di quanti nodi vuole che il grafo sia composto (massimo 15) e quale sia il peso massimo degli archi (da 0 a 50), quindi crea un grafico pseudo casuale. Tale creazione viene eseguita secondo le modalità precedentemente descritte per open file. Per i nomi dei nodi si utilizza un array di char che va da 0 a 14 con le lettere dell'alfabeto. Per quanto riguarda la probabilità di generazione degli archi, essa diminuisce o aumenta in base al numero dei nodi (più nodi meno archi). Scelto un nodo, se il rand, che va da uno a cento, supera la variabile probabilità, ossia 100 – (n° nodi * 6), allora l'arco viene creato. Per rendere il disegno più bello esteticamente vengono disegnate prima le linee e poi i cerchi sopra, in modo da evitare che le linee coprano il testo dei nodi.
- Reset: Pulisce le operazioni fatte dal bottone Dijkstra, in modo da ripartire con i grafo allo stato originario. Ciò viene fatto con un ciclo for che setta tutti i valori dei nodi a infinito colorandoli anche di nero ed elimina tutti i nodi dalla lista di priorità.
- Help: Apre una finestra in cui spiega le possibili operazioni da eseguire.
- Dijkstra: Esegue l'algoritmo in due diverse modalità, le quali verranno spiegate nel paragrafo relativo alla classe di appartenenza, per l'appunto *Dijkstra*.
- End: Termina l'operato dell'algoritmo.

Alcuni di questi bottoni possono disabilitarne altri, per esempio non è possibile eseguire il comando di Open file o di Random due volte di fila prima di aver avviato l'operazione di Delete.
L'altra classe del package, *LineAndTextLayout*, si occupa invece di gestire tutte le linee e le parti di testo del programma: possiamo quindi trovare la funzione **setAlert**, che si manifesta nel momento in cui l'utente commette un errore di digitazione o simile all'interno del programma, **setEdgeTextLabel** che organizza l'impostazione del testo degli archi per renderlo allineato con essi, **CreateLine**, la quale crea una lista di linee utilizzate in vari ambiti del programma, come per esempio nella separazione dei bottoni e per ultima **setChbox**, funzione che dà la possibilità all'utente di spuntare una casella in grado di nascondere tutto il testo sovrastante gli archi, in modo da migliorare il grafo esteticamente senza troppe scritte.
Terzo package del nostro Algraph è EventMouse, comprendente la sola classe omonima *EventMouse*, la quale, come suggerisce il nome, concerne tutti gli eventi correlati dall'azione del mouse, elencati qui di seguito. Elemento importante di questi è la funzione **selectPressed**, il cui compito è quello di trovare, date le coordinate x e y, la presenza di un vertice del grafo all'interno del pannello in cui si trova il grafo stesso, permettendo così l'interazione col grafo.

- **setEventClicked**: premendo e rilasciando subito il mouse, la funzione controlla che venga usato il tasto destro(quello sinistro non ha utilità in questo caso) e con la funzione **selectPressed** trova il nodo con le sue coordinate, e successivamente si apre un menù in cui è possibile scegliere tra eliminare un nodo, e quindi tutti gli archi da e a esso connessi,

aggiungere un arco, indicando il peso, eliminare un arco, indicando il nodo di arrivo, e cambiare il peso dell'arco, determinando in tutti e tre i casi il nodo di arrivo.

- **setEventPressed**: sempre avvalendosi di **selectPressed** la funzione ritorna semplicemente il vertice sul cerchio premuto.
- **setEventDragged:** grazie a setEventPressed, il nodo viene trovato all'interno del grafo. In seguito per ogni minuscolo movimento del mouse viene richiamata la funzione, la quale individua gli archi connessi e in base alla posizione svolta del nodo, se di arrivo o di partenza, nel grafo sposta e setta nuovamente la coordinata iniziale o finale della linea.
- **setEventMoved**: Nel momento in cui il mouse va a localizzarsi sopra un vertice, in basso viene visualizzato il nome del nodo e gli archi uscenti dal nodo stesso; in quel momento il nodo diventa temporaneamente blu.

Altra funzione da notare è la booleana **isInt**, il cui obiettivo consta nel verificare che data una stringa essa possa diventare un intero, utilizzata largamente nei controlli di corretto inserimento dell'utente; tale funzione si avvale del costrutto try and catch, ritornando cosi un valore vero o falso. Piccolo dettaglio in più che possiamo riscontrare nella classe, riguarda la possibilità di gestire i bottoni dalla tastiera, attraversi quelli che sono i JEvent, i quali attraverso determinate combinazioni di tasti, interagiscono con la scena richiamando gli stessi compiti dei bottoni visibili graficamente.

Quarto e ultimo package altro non è che il fulcro del programma, ossia quello che contiene la classe Dijkstra, esecutrice della voluta ricerca dei cammini minimi.
Avvalendosi dell'omonimo bottone, nel momento in cui esso viene premuto appare un menu che chiede da quale nodo l'utente vuole far partite l'algoritmo. A questo punto la risoluzione può avvenire secondo due modalità diverse:

1) Step-by-step: permette di visualizzare graficamente ogni passaggio dell'algoritmo in ogni sua fase. Nel concreto, setta tutti i nodi tranne quello scelto come punto di partenza ad infinito, e quello con il valore 0 viene inserito all'interno della coda di priorità con il colore nero. Un istante dopo si richiama la funzione delete min, la quale elimina il nodo con il valore minore ritornandolo; in questa fase nella coda di priorità viene visualizzato in giallo ad indicare che l'algoritmo sta lavorando con il nodo eliminato e ritornato. Quindi controlla e calcola il peso di tutti i possibili percorsi colorando di verde il nodo nella coda ad indicare che su quel nodo è già stata effettuata una fase di lavoro. Tutto il procedimento viene effettuato ciclicamente fino a quando i nodi nella coda di priorità saranno solo di due colori, verde per i nodi raggiungi e rosso per quelli mai raggiunti.
2) Immediata: vengono eseguite tutte le fasi appena descritte senza che l'utente possa vederle, facendo visualizzare solamente i nodi rossi o verdi nel grafo e nella coda di priorità.