ALGRAPH

Kruskal's Algorithm Java Implementation

Progetto di: Riccardo Preite e Vittoria Ferri

Introduzione: Algortimo di Kruskal

L'algoritmo di Kruskal si basa sulla tecnica di risoluzione ***Greedy*** per problemi di ottimizzazione. In questo particolare caso applicato ai grafi pesati, calcolando il **minimo albero di copertura** (in inglese Minimum Spanning Tree o **MST**).

Problema: MINIMO ALBERO DI COPERTURA

Definizione:

"Dato un grafo non orientato e connesso G = (V, E), con pesi w(u, v) non negativi sugli archi, trovare un albero di copertura G'=(V, T), per G, cioè un albero avente tutti i nodi di V, ma solo (n-1) degli m archi in E, tale che la somma dei pesi w(u,v) [con (u, v) appartenente a T] sia la più piccola possibile."

(Algoritmi e Strutture di Dati di Bertossi e Montressor)

Il problema di ottimizzazione del Minimo Albero di Copertura trova soluzione nell'algoritmo Kruskal (1956) e nell'algortimo di Prim (1957), basati sul principio greedy (o "principio della scelta ingorda").

Soluzione: ALGORITMO DI KRUSKAL

L'algoritmo di Kruskal genera da un grafo non orientato, pesato e connesso il **minimo albero di copertura** analizzando gli archi in ordine crescente di peso; dopo essere stato analizzato un generico arco viene aggiunto all'albero minimo di copertura T se non forma circuiti con gli altri archi già inseriti. La scelta *greedy* di Kruskal consiste nell'aggiungere a ogni passo un arco con peso minore.

Inizialmente Kruskal esegue un ordinamento in ordine crescente degli archi appartenenti al grafo preso in esame in base al loro peso per poi verificare, una volta scelto l'arco da inserire che non crei circuiti all'interno dell' MST. L'operazione di ordinamento e quella di verifica computazionalmente sono le più costose, infatti l'ordinamento iniziale degli archi costa *O(m log n)* tempo.
Nonostante l'applicazione di possibili "migliorie", quali per esempio il possibile utilizzo della struttura MFSet o l'introduzione della variabile ***c*** per tenere conto del numero di inserimenti in modo da ridurre il costo computazionale dell'algoritmo, **l'ordine di grandezza della complessità dell'algoritmo in ogni caso rimane *O(m log n)*.**

```
SET kruskal(ARCO[] A, integer n, integer m)
    SET T ← Set()
    MFSET M ← Mfset(n)
    { ordina A[1,...,m] in modo che A[1].peso ≤ ··· ≤ A[m].peso }
    integer c ← 0
    integer i ← 1
    while c < n − 1 and i ≤ m do          % Termina quando l'albero è costruito
        if M.find(A[i].u) ≠ M.find(A[i].v) then
            M.merge(A[i].u, A[i].v)
            T.insert(A[i])
            c ← c + 1
        i ← i + 1
```

Pseudocodice di kruskal() - rif. pp. 276 Algoritmi e Strutture di Dati di Bertossi e Montressor

Il progetto: Algraph

L'algoritmo di Kruskal ha ispirato il progetto Algraph, un'applicazione specializzata nella "creazione" del minimo albero di copertura per un grafo ***non orientato, connesso e pesato***. Sviluppata in Java e gestita graficamente con la libreria JavaFX, Algraph offre opzioni di creazione, modifica e molto altro per la struttura inserita in input. Tutte le operazioni effettuabili sono racchiuse e gestite nella barra dei menù. Le opzioni di creazione del grafo, garantite per l'inserimento in input della struttura, sono tre:

1. Nuovo, funzionalità che permette all'utente di creare il proprio grafo nodo per nodo e arco per arco assegnando il valore agli elementi (quando si sceglie la funzione dalla voce di menù Edit "Add Node" e il peso w agli archi, dopo aver selezionato "Add Edge");
2. Apri da file, funzionalità che legge un file .txt già esistente e che contiene tutte le informazioni relative al grafo da elaborare;
3. Genera random, crea un grafo (non orientato e pesato) in modalità randomica semplicemente specificando il numero di nodi di cui si necessita.

Una volta generato il grafo, esso viene implementato nel programma tramite una Hashmap (creata in modo "proprietario", non usando le funzionalità native di Java) nella quale sono contenuti i nodi e i relativi archi tra coppie di nodi dell'intero grafo. Graficamente è possibile visualizzare (passando con il cursore su ognuno dei cerchi, che ad ogni hover del mouse si colorano di blu) le informazioni relative ad ogni nodo in una finestra pop-up in cui sono inserite le coppie di nodi da esso formate con i rispettivi valori, colorati in rosso i nodi collegati con quest'ultimo, il programma in questo caso stamperà:

```
string = string + element + "w" + "<->" + element + "w2";.
```

Alla voce Edit nella barra dei menù troviamo le voci che consentono di creare sia i nodi sia gli archi; infatti le funzionalità di modifica del grafo, qualunque sia il suo metodo di input, sono: Add Node, Remove Node, Add Edge, Remove Edge.

Node: questa voce contiene l'inserimento di nodi all'interno del grafo, usando la funzione **`insertNode()`** che lavora tramite i MouseEvent (racchiusi in una classe a sè stante) facendo si che il programma riconosca ogni click del tasto sinistro e crei il nodo nella parte di schermo scelta e prima di visualizzarlo apra una finestra di dialogo che permetta di inserire il valore da associare (per una scelta di gestione e di leggerezza del programma, tutti i dati inseriti sono letti come stringhe, l'output per l'utente risulterà comunque un numero o una lettera a seconda di ciò che il grafo rappresenta). È anche presente la voce di modifica Set/Change Value per modificare o inserire il valore al nodo già creato, qualora NULL. Una volta generato il nodo è possibile spostarlo a proprio piacimento semplicemente "trascinandolo" e tramite tasto destro su di esso si ha l'opzione di rimozione del nodo e degli archi ad esso connessi.

Edge: creazione, modifica e rimozione di un arco; le due voci di elaborazione degli archi da menù chiamano tutte quante una funzione che chiede all'utente di selezionare i due nodi corrispondenti all'arco in esame, dopodiché in base all'opzione scelta da menù alla funzione viene passato un intero tra 0, 1 e 2. Dopo aver selezionato i nodi, la funzione non fa altro che controllare quale intero gli è stato passato e chiama a sua volta addEdge per lo 0, deleteEdge per l'1 e changeWeight per il 2.

È importante specificare che la gestione del codice del programma è stata strutturata utilizzando classi separate per tutte le parti corpose in modo da ottenere un codice snello e ben navigabile senza appesantire il file Main.java e agevolare una eventuale modifica da altri programmatori.

Dopo aver inserito il grafo, averlo modificato in modo da corrispondere all'input desiderato, si può salvare la visualizzazione grafica sia dell'input sia del risultato ottenuto con l'applicazione dell'algoritmo di Kruskal. Il formato (.txt) in cui vengono salvati i file consente, come in qualsiasi altro programma di editing, di poter riprendere la modifica o l'esecuzione dell'algoritmo sul grafo in esso salvato, tenendo memoria di eventuali cambiamenti già effettuati.

Gestione degli errori

All'interno di Algraph sono presenti "***alert***" che gestiscono la presenza di errori da parte dell'utente, come in particolare quelli riportati qui sotto:

1. Aggiunta di un nodo troppo vicino ai bordi del layout o tentativo di "disegnarlo" sopra un altro;
2. Inserimento di un arco tra due nodi quando già esistente;
3. Nelle funzioni setElement, deleteNode, changeWeight, deleteEdge si genera un errore se si cerca di eliminare qualcosa che non esiste;
4. Apertura file da sistema, nel momento in cui nella sezione Elements non vengano inseriti i valori per ogni nodo, il programma li riempe in modo automatico con la stringa "N.E.";
5. Tentativo di applicare l'algoritmo di Kruskal a un grafo con nodi isolati e/o più componenti connesse disgiunte tra loro.

Ogni volta in cui viene rilevato uno di questi errori, nella schermata del programma comparirà una finestra di avviso all'utente in cui lo si informa dell'errore commesso o rilevato (nel caso dei grafi generati randomicamente per i nodi isolati) in modo che possano essere ricreate le condizioni di stabilità del funzionamento del programma.

Algoritmo di Kruskal

Come già accennato nell'introduzione, l'algoritmo di Kruskal genera da un grafo non orientato, pesato e connesso il **minimo albero di copertura** analizzando gli archi in ordine crescente di peso.
Algraph, una volta ricevuto il grafo in input e scelta l'opzione di esecuzione di Kruskal dalla voce *Option* nella barra dei menù, esegue un controllo tramite ***visita in profondità*** (in inglese Depth First Search o DFS) per verificare la validità delle proprietà della struttura richieste per eseguire l'algoritmo.
La DFS tramite un ciclo **`foreach`** e l'uso di un campo **booleano** *(true o false)* del nodo esegue, partendo da un vertice passato come parametro di input, un controllo su tutte le liste di adiacenza del nodo stesso e di tutti gli altri nodi del grafo per verificare che non ce ne siano di isolati (non connessi a nessun altro) o componenti connesse disgiunte, quindi in pratica verifica che il grafo sia interamente connesso. Nel caso in cui non lo fosse, il programma blocca l'esecuzione dell'algoritmo e mostra sullo schermo un ***alert message*** per informare l'utente, che avrà la possibilità di modificare il grafo in modo da poter calcolare l'MST tramite l'algoritmo di Kruskal.

Implementazione KRUSKAL

L'implementazione dell'algoritmo di Kruskal in Algraph viene eseguita nella classe **kruskal.java**, in cui, come da descrizione teorica iniziale, viene esaminata la lista di archi e in base al risultato della DFS, applica o meno il processo di "creazione" del minimo albero di copertura.
L'algoritmo lavora su una **`LinkedList`** chiamata **`Kruskal_List`** che contiene tutti gli archi e le relative coppie di nodi del grafo inserito e già validato. Nel codice per il calcolo dell'MST si utilizza un vettore di padri, la cui **`length`** viene inizializzata pari al numero di nodi presenti nel grafo di input.
Tramite un doppio ciclo annidato **`foreach`** sulla lista di archi e sulla lista dei nodi, si trova la posizione dei nodi facenti parte dell'arco che si sta esaminando, le quali vengono salvate in due variabili d'appoggio, che avranno la funzione di assegnare l'indice corrispondente al nodo selezionato nel vettore di padri precedentemente inizializzato.

Successivamente, viene effettuato un controllo sui nodi presi in esame per verificare se *"coincidono"*, cioè se si sta cercando di creare un arco nell'albero MST che ritorna allo stesso nodo; se i nodi sono diversi, viene chiamata la funzione **`unionSet()`**, la quale crea l'arco che verrà poi inserito nel minimo albero di copertura.

Il peso di ogni arco dell'MST viene aggiunto, ogni volta che si inserisce un nuovo arco, alla variabile intera **`mst-cost`**, che ha la funzione di accumulatore per ottenere il peso totale dell'albero alla fine del processo. È inoltre presente nel codice di Algraph, un'ArrayList in cui vengono salvati gli archi dell'output che prende il nome di **`MinSpanningTree`**, per non perdere alcun dato relativo alla struttura di dati.

Specifiche dell'output grafico

In Algraph, la classe che implementa Kruskal contiene anche una variabile di tipo booleano, chiamata **`get,`** la quale ha la funzione di dividere il processo applicativo dell'algoritmo in due "*sezioni*", tra cui si passa tramite click del tasto *INVIO*, permettendo così di visualizzare i passaggi del processo di creazione dell'MST; ovviamente nel caso in cui si voglia ottenere direttamente l'output finale, nella barra dei menù è presente l'opzione Skip to End.

Per quanto riguarda le due sezioni di lavoro del processo: la prima ha l'obiettivo di catturare graficamente l'arco preso in esame e colorarlo di giallo ocra per evidenziare il fatto che è in fase di elaborazione, mentre la seconda è fondamentale per la creazione dell'albero vero e proprio dato che in essa si svolge tutta la parte implementativa dell'algoritmo. In questa seconda fase, infatti, se l'arco rispetta i criteri richiesti per essere inserito nell'MST viene colorato di rosso, altrimenti rimane di colore giallo ocra per poi essere "eliminato".

Un'ulteriore funzionalità di questo programma si svolge tramite la chiamata alla funzione **`cleaner()`**, la quale "pulisce" l'output grafico delle linee in eccesso mantenendo solamente sullo schermo la struttura finale dell' albero di copertura.

Infine, una volta generato l'MST è importante puntualizzare che è possibile ritornare alla struttura dati iniziale e poterla modificare ulteriormente ed eventualmente se si vuole, o se ne ha la necessità, ri-applicare l'algoritmo di Kruskal sul grafo modificato.