Algoritmi per grafi

Emanuele Casciaro

July 10, 2021

Contents

1 Introduzione

In questo articolo verranno presentati degli algoritmi per grafi, in particolare la Depth First Search e l'implementazione di Prim dello Strongly Connected Component, valutandone le prestazioni in base all'estensione del grafo.

2 Cenni teorici

Vediamo adesso qualche definizione sui grafi.

2.1 Grafo

Un grafo può essere definito come un insieme di vertivi e di archi $G = (V, E)$, dove V è l'insieme dei vertici E è l'insieme degli archi, ossia di coppie di vertici. Ad ogni nodo A e B, se collegati da un nodo $\exists(A, B) \in E$.
Un grafo è detto *orientato* se la direzione degli archi è rilevante, pertanto se non lo è $(A, B) \in E \Rightarrow (B, A) \in E$. Un gruppo di *Strongly Connected Component* è un insieme di vertici tale che per ogni coppia di vertici esiste un percorso che li collega.

$$\forall v_i, v_j \in S \subseteq V \exists v_i \rightarrow v_j$$

3 Depth First Search

È un algoritmo di esplorazione dei grafi, che consente di visitare ogni nodo del grafo, restituendo inoltre le informazioni circa i tempi di inizio e fine esplorazione di ogni nodo. L'algoritmo assegna ad ogni nodo tre colori:

- **Bianco** per i nodi ancora non esplorati
- **Grigio** per i nodi la quale espolrazione è ancora in corso
- **Nero** per i nodi già esplorati

Occorre però chiarificare cosa significa per un nodo essere **grigio**: esplorare un nodo vuol dire esploarare tutti i suoi figli; allora un nodo diventerà grigio nel momento in cui inizierà la sua esplorazione, e diventerà nero una volta che non avrà più figli da esplorare. Ciò allora ci da un criterio di arresto dell'esplorazione: si parte da un nodo detto *radice* arbitrario e si inizia ad esplorare in modo ricorsivo i suoi figli, interrompendo ogni volta che si incontra un nodo non bianco.

3.1 Tempi di esplorazioni

Come detto in precedenza, l'algoritmo assegna ad ogni nodo i tempi di inizio e fine espolrazione, usate successivamente. Per com'è fatto l'algoritmo, è garantita la seguente proprietà.

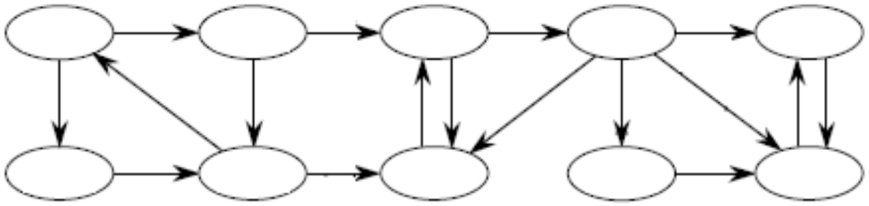
Per ogni coppia di vertici v_i, v_j, i tempi di inizio e fine esplorazione non si incrociano, ossia solo una delle seguenti è vera:

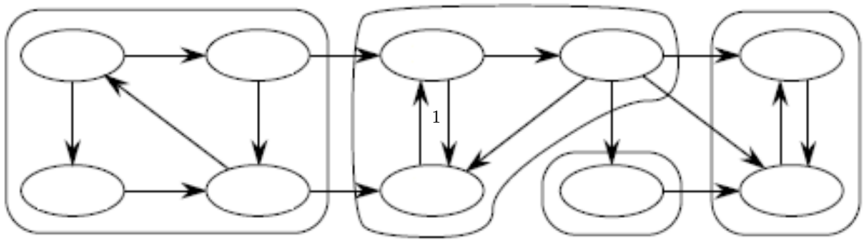
- $[v_i.begin, v_i.end] \cap [v_j.begin, v_j.end] = [v_i.begin, v_i.end]$
- $[v_i.begin, v_i.end] \cap [v_j.begin, v_j.end] = [v_j.begin, v_j.end]$
- $[v_i.begin, v_i.end] \cap [v_j.begin, v_j.end] = \emptyset$

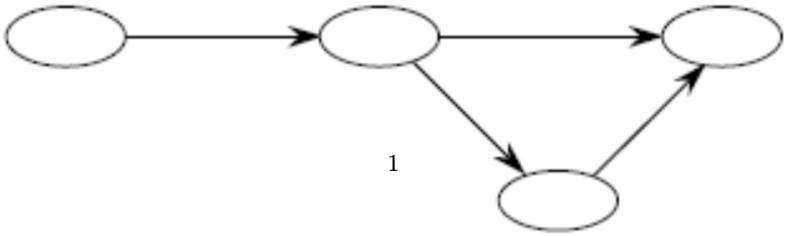
Questo teorema, detto teorema delle parentesi, è un risultato fondamentale che ci permetterà di formulare un algoritmo per il calcolo delle *Strongly Connected Component*

4 Strongly Connected Component

L'algoritmo si basa sul fatto che sia un grafo G che la sua trasposta G' condividono le stesse SCC, cosa di cui ci si può convincere abbastanza facilemente osservando l'esempio sottostante.







In particolare, anche $G^{SCC} = (V^{SCC}, E^{SCC})$ è un grafo orientato.

4.1 Algoritmo

1. Calcola $DFS(G)$ per ottenere $v.f \forall v \in V$
2. Calcola G^T

3. Calcola $DFS(G^T)$ considerando in ordine decrescente rispetto a $u.f$
4. Genera l'output ad ogni iterazione come un singolo nodo contenente ogni nodo esplorato durente l'iterazione

Chiamando $DFS(G^T)$ con questo ordine si ha infatti che ad ogni iterazione possono essere esplorati soltanto i nodi appartententi all'SCC locale, oltre che a i nodi già esplorati, che verranno quindi ignorati. Ciò è appunto garantito in quanro considerando che il nodo a $v.f$ più alto nel grafo G ha intuitivamente pochi figli in G^T, perchè avendo molti figli nel grafo di partenza, invertendo gli archi ne ha sicuramente pochi. E dato che le SCC sono le stesse a quell'iterazione esplorerà tutti e soli i nodi di tale SCC (non può uscire, in quanto gli archi sono invertiti).All'iterazione successiva, si inizierà da un nodo all'interno del SCC collegata a quella appena esplorata, e così via. A questo punto non resta che raccogliere i nodi esplorati ad ogni iterazione e unirli in un unico nodo, mantenendo gli archi verso le altre SCC.

5 Minimum Spanning Tree

Dato un grafo non orientato pesato, l'albero di connessione minima è l'albero che collega tutti i nodi nel modo più efficiente possibile.
Si ha quindi un peso $w(u,v) \forall (u,v) \in E$, che permette di formalizzare il problema in questo modo:

Trovare $T^* \subseteq E$, con $T^* = arg\ \min_T\ w(T) = \sum_{(u,v)\in T} w(u,v)$

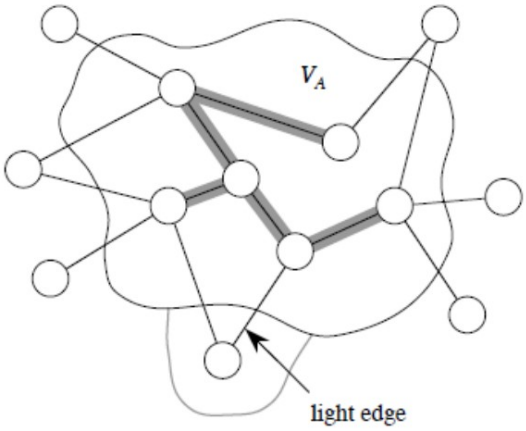
Qualche definizione utile:

- un *taglio di* V (V_A, V_S) è una partizione di V di insieme disgiunti V_A e V_S
- un *arco* $(u,v) \in E$ *attraversa* un taglio se $u \in V_A$ e $v \in V_S$ (o viceversa)
- un arco è *leggero* per un taglio se ha *peso minimo* rispetto a tutti gli archi che attraversano il taglio

Intuitivamente costruire un albero di soli archi leggeri porta ad avere un MST.

6 Algoritmo di Prim

Costruisce l'MST a partire da un nodo casuale, aggiungendo ad ogni iterazione un *arco sicuro* all'albero, fino a raggiungere tutti i nodi. Per fare ciò si usa la struttura dati della coda di priorità, dove ogni nodo è ordinato (in modo crescente) in base al suo costo, definito come il peso minimo degli archi che lo collegano al sottoalbero V_A: così facendo il pop dalla coda si prende sempre un arco sicuro.



7 Test

Ci sono due test principali: il primo testa l'algoritmo di Prim generando un grafo *non orientato*, assegnando agli archi pesi casuali, per poi cercarne l'MST. Il secondo test invece riguarda la ricerca delle componenti fortemente connesse, in particolare si cerca una possibile relazione tra dimensione di un grafo e il suo numero di SCC: vengono creati una serie di grafi orientati, nei quali gli archi sono collocati in modo casuale, e si calcolano le loro SCCs, per ogni dimensione di grafo vengono calcolate le SCCs di più grafi, raccogliendone i dati sulla loro presenza, riportati poi in un grafico.

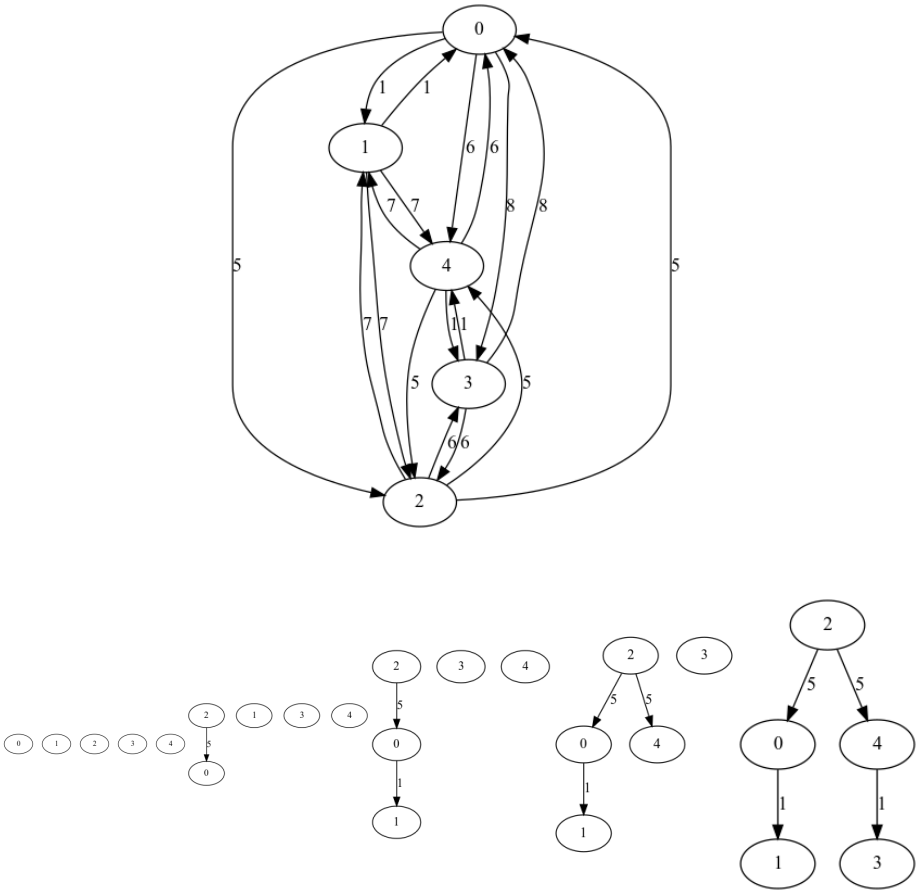
8 Risultati attesi

Data la casualità nell'assegnazione degli archi, mi aspetto che ci sia una relazione di proporzionalità diretta tra dimensione del grafo ed il suo numero di SCCs.

9 Risultati e conclusioni

9.1 Prim

L'algoritmo di Prim ha dimostrato di poter generare un MST: a seguire un esempio della sua esecuzione



9.2 Strongly Connected Components

Ecco qualche esempi di funzionamento: in ordine, grafo iniziale, grafo inverso, grafo SCCs.

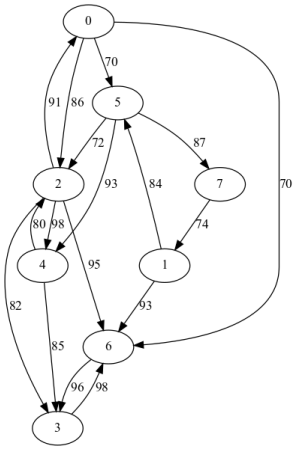


Figure 1: Original Graph

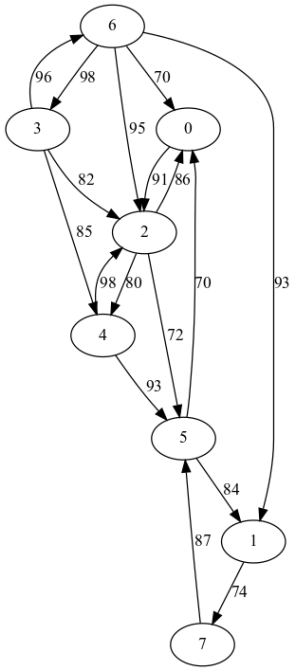


Figure 2: Inverted Graph

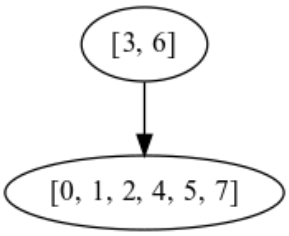


Figure 3: Result

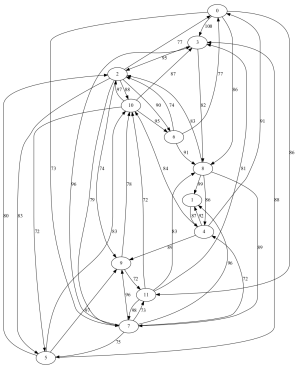


Figure 4: Original Graph

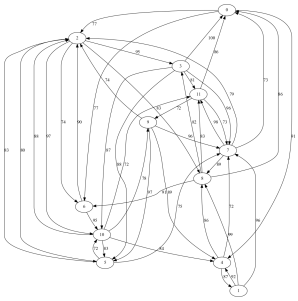


Figure 5: Inverted Graph

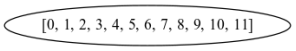


Figure 6: Result

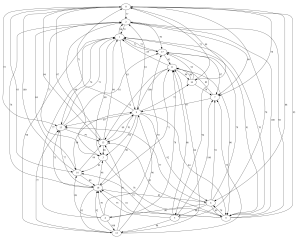


Figure 7: Original Graph

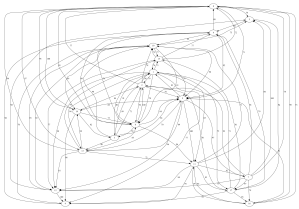


Figure 8: Inverted Graph

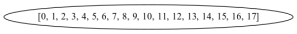
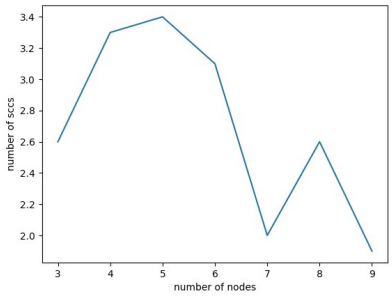
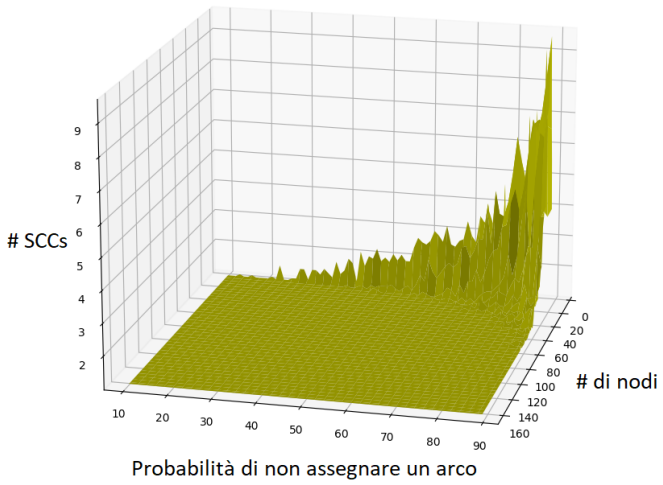


Figure 9: Result

I risultati in questo caso sono differenti dalle aspettative, in quanto in media, per grafi grandi si ha solo una SCC. Questo probabilmente accade perchè dato che gli archi sono assegnati casualmente, al crescere del numero dei nodi cresce anche il numero degli archi, con maggiori probabilità di unire più SCCs.



Allora sono state eseguite simulazioni aggiuntive, per evidenziare come il numero di nodi e la quantità di archi influenzano il numero di SCCs.


SCCs
9
8
7
6
5
4
3
2
10
20
30
40
50
60
70
80
90
Probabilità di non assegnare un arco
0
20
40
60
80
100
120
140
160
di nodi