Componenti connesse e Minimium Spanning Tree di un grafo

Leonardo Toccafondi

February 15, 2022

Introduzione

Grafi

Un grafo G è un insieme di elementi detti **vertici** (o nodi) che possono essere collegati fra di loro tramite linee chiamate **archi**. In particolare, si dice grafo la coppia ordinata di insiemi $G = (V, E)$, dove V è l'insieme dei vertici di G e E è l'insieme degli archi di G. Un grafo si dice **orientato** (o **diretto**) se E è un insieme di archi *orientati*, cioè con una direzione (da sorgente a destinazione). Viceversa, un grafo si dice **non orientato** (o **indiretto**) se gli archi non sono orientati, dunque se la connessione tra due vertici ij ha lo stesso significato della connessione ji. Si definisce come grado di un vertice: numero di archi che incidono nel vertice. Inoltre, ad ogni arco di un grafo può essere assegnato un peso (che di solito corrisponde ad un numero reale, ma può essere anche ristretto agli interi).

I grafi sono utilizzati come strutture dati al fine di rappresentare relazioni tra elementi generici, espresse nel modo più semplice tramite gli archi, che evidenziano le connessioni tra gli elementi.

Rappresentazione Un grafo può essere rappresentato con due tipologie di strutture dati:

- **Lista di adiacenza:** composta da un vettore Adj composto da $|V|$ liste, una per ogni nodo, dove $Adj[u]$ contiene tutti i vertici v t.c. $(u, v) \in E$).

  La lista di adiacenza richiede uno spazio di memoria $\Theta(V + E)$, e un tempo $\Theta(u.degree)$ per determinare se due vertici qualsiasi $(u, v) \in E$. In questo caso $u.degree$ rappresenta il **grado** del vertice (u).

- **Matrice di adiacenza:** si introduce una matrice A di dimensione $|V| \times |V|$, dove ogni elemento (i, j) ha valore 1 se $(i, j) \in E$), cioè se il vertice (j) è adiacente a (i), e valore 0 altrimenti. La matrice di adiacenza richiede uno spazio di memoria $\Theta(V^2)$, quindi peggiore rispetto alla lista, però consente

di determinare se due vertici qualsiasi $(u, v) \in E$) in un tempo $\Theta(1)$, in quanto è sufficiente accedere all'elemento $A[u][v]$ e controllarne il valore. In generale, una matrice di adiacenza è più indicata per descrivere grafi densi e con molti archi.

Per gli esperimenti svolti sono state utilizzate entrambe le strutture dati.

Per estrapolare ulteriori informazioni, si utilizzano le cosiddette componenti connesse e l'albero ricoprente minimo (o Minimum Spanning Tree in inglese), quest'ultimo tramite alcuni algoritmi, tra cui quello di Kruskal. Tuttavia, è necessario prima introdurre una struttura dati per degli insiemi disgiunti, nota anche come *Union-find*

Union find Gestisce una collezione di insiemi dinamici, ciascuno identificato da un rappresentante, ovvero un elemento dell'insieme qualsiasi, a patto che rimanga sempre lo stesso. Su questa struttura dati possiamo operare con tre funzioni:

- MakeSet(x): aggiunge alla collezione un nuovo insieme creato a partire dall'elemento x;
- FindSet(x): restituisce il rappresentante dell'insieme che contiene x;
- Union(x,y): se $x \in S_x, y \in S_y, \Rightarrow S \leftarrow S_x - S_y \bigcup \{S_x \cup S_y)$ Quindi aggiunge alla collezione un nuovo insieme contenente gli elementi degli insiemi di x e y, e poi elimina gli insiemi d'origine. Il rappresentante diventa uno degli elementi di S_x o di S_y

Componenti connesse

Dato un grafo $G = (V, E)$, una componente connessa é un insieme massimale di vertici $C \subset V$ tale che per ogni coppia di nodi esiste un cammino che li collega. Le componenti connesse *partizionano* i vertici in classi di equivalenza secondo la relazione *"è raggiungibile da"*. Un grafo è connesso se ogni coppia di vertici è collegata attraverso un cammino. Sono utilizzate ad esempio per trovare "oggetti" all'interno di immagini, interpretando come nodi i singoli pixel dell'immagine stessa L'algoritmo utilizzato all'interno di questo esperimento si basa sull'utilizzo di union-find come struttura dati.

Albero ricoprente minimo (MST)

Si tratti di un albero di connessione T (sottoinsieme dell'insieme di archi E come un grafo non diretto $G = (V, E)$) in cui la somma dei pesi dei suoi archi:

$w(T) = \sum_{(u,v)\in T} \ w(u, v)$

sia minimo e connetta tutti i vertici. Un MST ha $|V| - 1$ archi, non ha cicli e *può non essere unico.*

Esiste un algoritmo generico che permette di costruire la soluzione e consiste in creare un insieme vuoto di archi A, per poi aggiungerne progressivamente conservando l'invariante di ciclo:

> "Se A é un sottoinsieme di qualche MST, l'arco (u, v) é *sicuro* per A se e solo se $A \cup (u, v)$ é sottoinsieme di un qualche MST."

Da questo è possibile ricavare che al fine di ottenere un albero ricoprente minimo sia necessario aggiungere solamente $N - 1$ archi *sicuri*, con N pari al numero di nodi.

Per ottenere l'algoritmo di Kruskal, che sarà utilizzato per gli esperimenti, è necessario introdurre il seguente *teorema*:

> "Sia A un sottoinsieme di qualche MST, $(S, V - S)$ un taglio che rispetta A e (u, v) un arco leggero che attraversa $(S, V - S)$. Allora (u, v) é sicuro per A."

Anche in questo caso per l'implementazione pratica viene utilizzata la struttura dati *Union-find.*

Implementazione pratica

Il programma è suddiviso in più file:

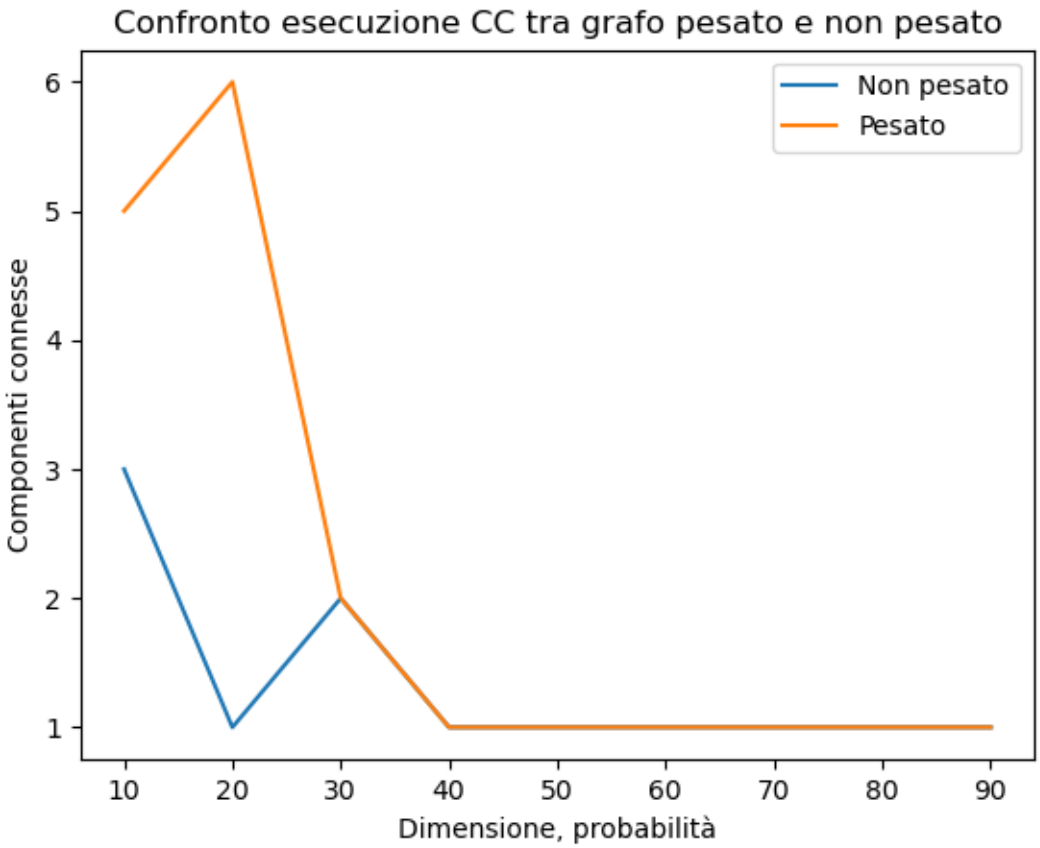
- Graph.py si occupa della definizione dell'oggetto Node, che al suo interno contiene tutti gli attributi necessari, sia l'oggetto Graph, che corrisponde al nostro grafo. Il grafo viene popolato assegnando casualmente (in base ad un valore di probabilità) archi ad ogni nodo, portando ad uno il valore corrispondente nella matrice aggiogata. La matrice può anche essere equivalente alla matrice pesata, nel caso in cui la variabile booleana w sia posta a True al momento della creazione del grafo. Dopodiché viene inserito un vettore composto da [nodo u, nodo v, peso] (il peso nel caso di *default* è posto ad 1). All'interno di questo file è presente anche la funzione sort_edges, utilizzata nell'algoritmo di Kruskal per ordinare in base al peso gli archi.
- linked_list.py: in questo file vi è implementata la lista collegata utilizzata come base per la struttura dati union-find
- union_find.py: in questo file sono stati implementati gli oggetti e le funzioni necessarie al fine di rappresentare insiemi disgiunti dinamici.
- cc.py: implementazione dell'algoritmo delle componenti connesse come trovato sul Cormen
- mst.py: algoritmo di kruskal, sempre basato sull'implementazione di Cormen

Esperimenti

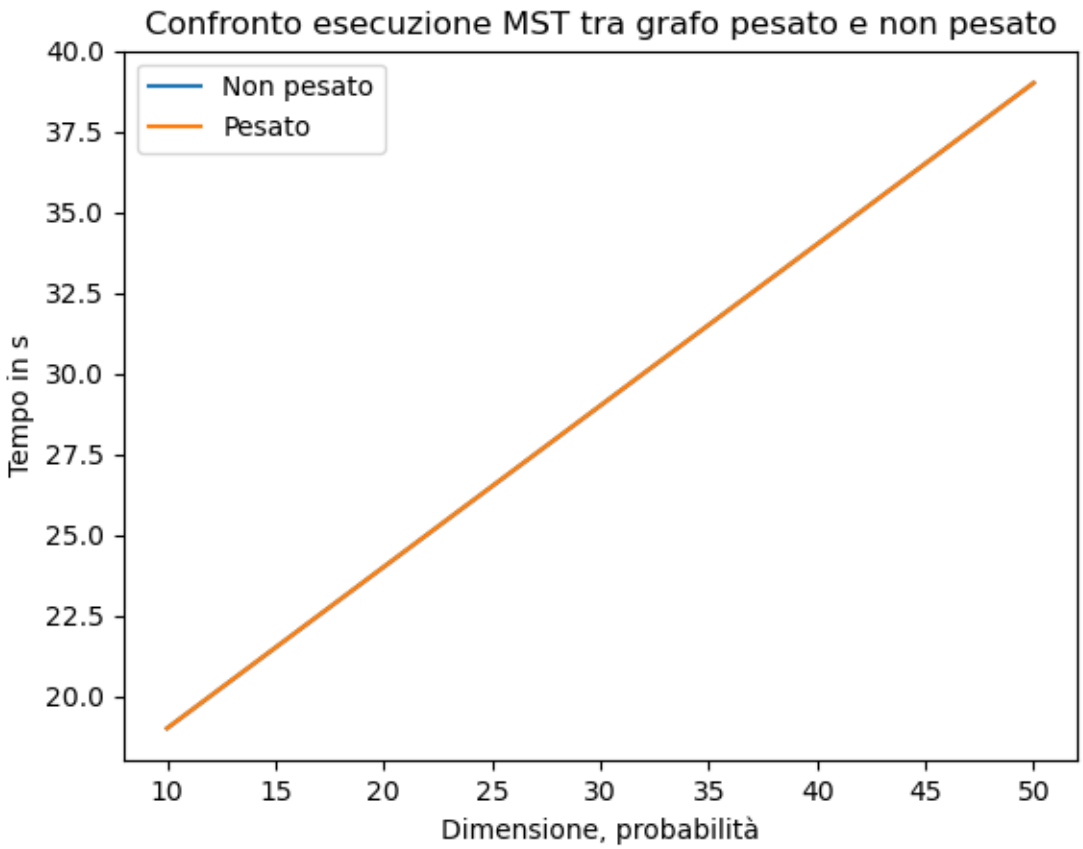
Si andranno a confrontare componenti connesse e MST singolarmente, determinando i loro valori all'aumentare simultaneo della probabilità e del numero di nodi. Inoltre viene misurato, sul grafo pesato il tempo necessario ad eseguire gli algoritmi di Kruskal e delle componenti connesse.

Risultati

Componenti connesse



MST


Confronto esecuzione MST tra grafo pesato e non pesato
Non pesato
Pesato
Tempo in s
40.0
37.5
35.0
32.5
30.0
27.5
25.0
22.5
20.0
10
15
20
25
30
35
40
45
50
Dimensione, probabilità

Confronto tempi