Componenti Connesse e MST-Kruskal

Francesco Baiocchi

Febbraio 2023

Indice

1 Introduzione

La presente relazione ha come obiettivo quello di illustrare i risultati di esperimenti su algoritmi per la ricerca di **componenti connesse** e dell'**MST**, Minimum Spanning Tree, applicati a grafi pesati generati casualmente.

2 Algoritmi e Strutture Dati utilizzati

Per la comprensione degli argomenti trattati in seguito, vengono brevemente richiamati i concetti principali della teoria dei grafi che verranno utilizzati. Successivamente vengono presentati algoritmi e strutture dati utilizzati nel progetto.

2.1 Richiami dalla Teoria dei Grafi

Di seguito alcune delle nozioni più rilevanti usate nel progetto.

- **Componente Connessa** : in un grafo non orientato, una componente connessa rappresenta un sottoinsieme di nodi che sono tutti collegati tra loro tramite archi del grafo.
- **MST** : Un Minimum Spanning Tree è un albero ricoprente di un grafo dove la somma dei pesi degli archi ha valore minimo.

2.2 Strutture dati utilizzate

Le strutture dati utlizzate nel progetto sono:

- **Matrice di Adiacenza** : utilizzata per la memorizzazione di grafi pesati. La posizione [i,j] della matrice rappresenta l'arco che connette il nodo i-esimo con il nodo j-esimo. Se il valore dell'elemento [i,j] è 0 allora non esiste un arco tra i rispettivi nodi, altrimenti esiste e il peso è dato da quel valore.
- **Union-Find** : utilizzata nell'algoritmo per trovare le componenti connesse e nell'algoritmo di Kruskal. Questa consente di gestire una collezione di insiemi disgiunti dinamici.

2.3 Specifiche della piattaforma di test

L'insieme dei test è condotto su un elaboratore con le seguenti caratteristiche:

- **Processore** : 1,2 GHz Dual-Core Intel Core m3
- **RAM** : 8 GB 1867 MHz LPDDR3
- **Sistema Operativo** : macOS Ventura 13.2.1

3 Descrizione metodi implementati

Prima di vedere l'implementazione, in linguaggio Python, del progetto descriviamo brevemente gli algoritmi analizzati.

3.1 MST-Kruskal

MST-Kruskal : algoritmo "greedy" per la ricerca del *Minimum Spanning Tree* in un grafo pesato e connesso.
Questo algoritmo prende in input un grafo connesso, non orientato e pesato e ne trova un albero di connessione minimo. L'idea che utilizza è quella di isolare inizialmente tutti i vertici del grafo definendo una foresta di alberi disgiunta formati da ogni singolo vertice. A questo punto attraverso un approccio iterativo considera gli archi del grafo in ordine decrescente di peso, verifica che i due nodi che questi connettono non siano nello stesso albero e in quel caso unisce i due alberi di cui questi fanno parte. Per fare ciò si appoggia sulla struttura Union-Find. L'algoritmo termina quando la foresta è costituita da un solo albero.

Il costo computazionale è asintodico a $\mathcal{O}(E \log(V))$, dove V ed E sono rispettivamente l'insieme dei vertici e degli archi del grafo.

4 Documentazione Codice Python

In questa sezione viene descritta l'implementazione dei metodi più importanti del progetto.

4.1 Classe GraphGenerator

- **generateGraph** : questo metodo presente nel file **graphGeneretor.py** riceve come input la size del grafo che si vuole creare e la probabilità di presenza di archi tra vertici e restituisce una matrice di adiacenza associata al grafo creato.
- **randomGraphGenerator** : fa la stessa cosa generando casualmente un grafo pesato

4.2 Classe Union-Find

- **makeSet(x)**: crea una nuova lista contenente solo il nodo x.
- **findSet(y)**: restituisce il nodo rappresentante della lista che contiene il nodo y.
- **union(x, y)**: effettua l'unione delle due liste cui i nodi x e y appartengono; controllando, attraverso i relativi rappresentanti, che non siano già nella stessa lista.

4.3 ConnectedComponents

- **findConnectedComponents** : metodo che riceve in input una matrice di adiacenza di un grafo, ne trova le componenti connesse, utilizzando la struttura dati Union-Find, e ne restituisce il numero.

4.4 Kruskal

- **MST-Kruskal** : metodo che riceve in input una matrice di adiacenza di un grafo pesato, ne trova l'MST e ne restituisce una lista di archi che compongono l'MST.

5 Test effettuati

In questa sezione vengono presentati quindi i test condotti, analizzando i risultati ottenuti. Ogni test presenta graficamente l'andamento dei risultati al crescere della dimensione dei grafi e della probabilità di presenza di archi al loro interno. I risultati presentati rappresentano una media ottenuta da un totale di 100 esperimenti.

5.1 Test componenti connesse

Analizziamo il comportamento dell'algoritmo **findConnectedComponents()** all'aumentare del numero di nodi,da 10 nodi fino a 100,con incremento costante di 1 e probabilità di presenza di archi fissata ad 1%.

La figura 2 mostra come nelle condizioni sopra descritte il tempo di esecuzione dell'algoritmo preso in esame aumenti al crescere dei nodi. Inoltre, possiamo notare come nella figura 1 il numero di componenti connesse aumenti tenendo ferma la probabilità di presenza di archi.

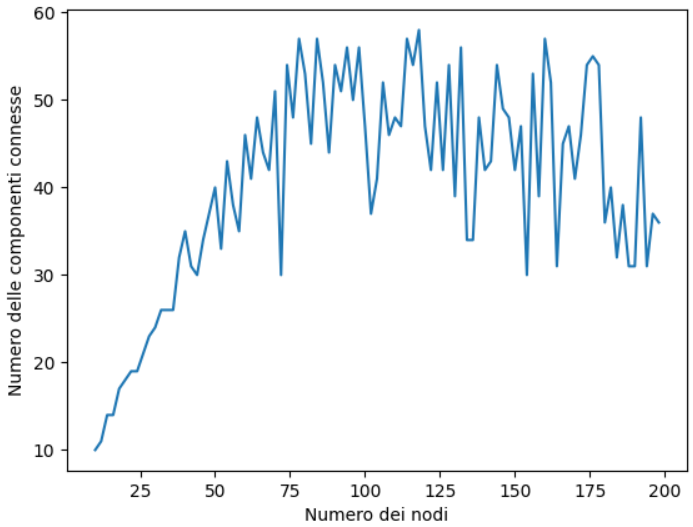

Figura 1: Componenti connesse trovate all'aumentare del numero di nodi

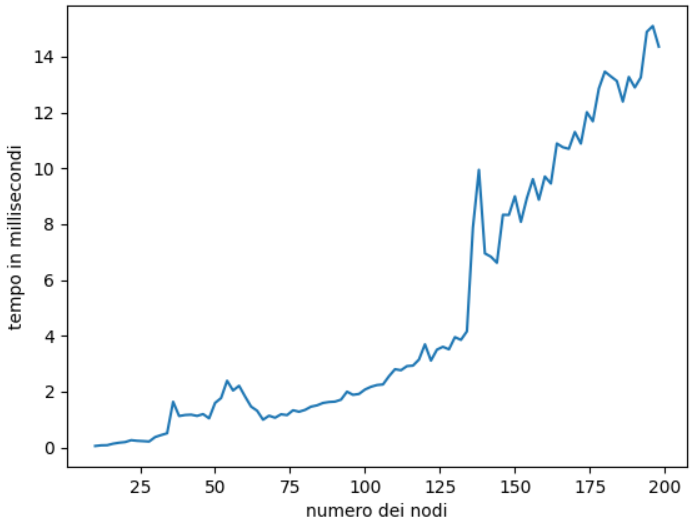

Figura 2: Tempo esecuzione algoritmo findConnectedComponents() al crescere dei nodi e con probabilità di presenza di archi fissata

Viene fatta variare la probabilità di presenza degli archi all'interno del grafo da 0% a 100% con incremento dell'1%, tenendo fisso il numero dei nodi. Dalle figure seguenti si può notare come il numero delle componenti connesse dimuisce sempre di più fino ad arrivare fissa ad 1 mentre il tempo d'esecuzione dell'algoritmo aumenta.

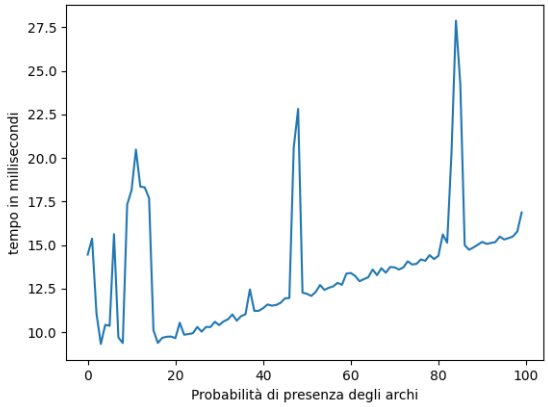

Figura 3: tempo impiegato dall'algoritmo per trovare le componenti connesse all'aumentare della probabilità di presenza tra gli archi

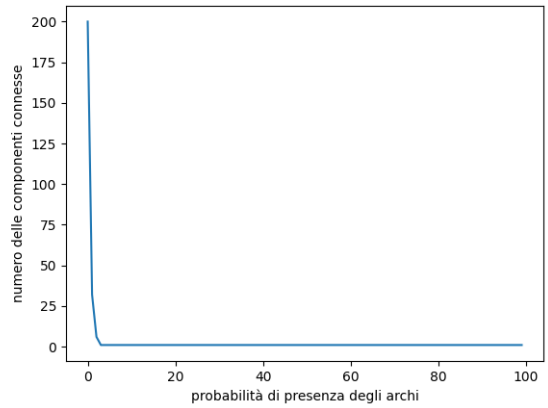

Figura 4: numero delle componenti connesse trovate all'aumentare della probabilità di presenza tra gli archi

5.2 MST-Kruskal

Partendo da 100 nodi fino a 1000,con step di 10 nodi e una probabilità di presenza di archi fissata al 1%, come mostrato in figura 5.

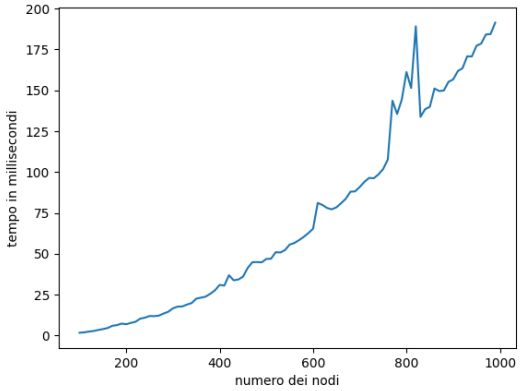

Figura 5: tempo impiegato dall'algoritmo di Kruskal all'aumentare del numero di nodi

In quest'ultimo test si è variata la probabilità da 0 a 100% con incremento costante dell'1% , tendendo fissato il numero dei nodi a 30.

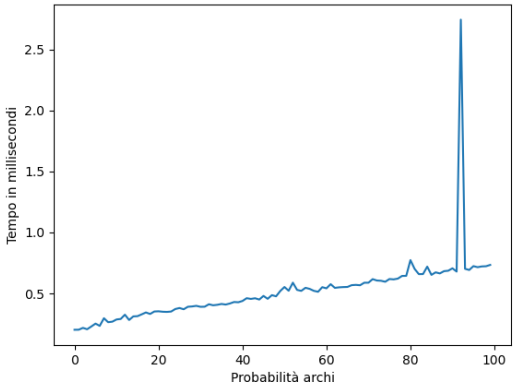

Figura 6: tempo impiegato dall'algoritmo di Kruskal all'aumentare della probabilità di presenza degli archi

6 Conclusioni

6.1 Conclusioni test componenti connesse

Dai risultati si può osservare che le aspettative pre-test vengono rispettate. Dalle figure associate ai relativi test emerge che sia all'aumentare della probabilità di presenza degli archi e sia all'aumentare dei nodi il tempo di esecuzione aumenti mentre il numero di componenti connesse trovate diminuisca. Questo è in linea con i concetti inizialmente espressi dalla teoria dei grafi, aumentando gli archi le componenti connesse sono sempre meno e più grandi.

6.2 Conclusioni test MST-Kruskal

Allo stesso modo dei test sulle componenti connesse, i test relativi all'algoritmo di Kruskal confermano ciò che ci si aspettava, il tempo di esecuzione dell'algoritmo aumenta sia al crescere degli archi che dei nodi. E' inoltre in accordo che la crescita sia più ripida all'aumentare degli archi piuttosto che dei nodi essendo la complessità computazionale dell"algoritmo $\mathcal{O}(E \log(V))$