Componenti connesse e Minimum Spanning Tree

Andrea Spitaleri

July 8, 2018

1 Introduzione

Ci si propone di testare le differenze tra i tempi di esecuzione di algortimi per creare matrici di adiacenza di grafi casuali, algoritmi per creare strutture dati Union-Find partendo da queste matrici, e algoritmi per ricavare Minimum Spanning Tree da grafi indiretti e memorizzarli in strutture dati Union-Find (Algoritmo di Kruskal). Tutto ciò verrà testato al variare del numero di nodi di tali grafi, e al variare della percentuale di interconnessioni tra nodi, al fine di capire quale tra questi paramentri incide maggiormente nei tempi di esecuzione di tali algoritmi.

2 Generazione di matrici di adiacenza di grafi causuali

La funzione implementata nei file Python allegati è in grado di generare matrici di adiacenza di grafi casuali, in cambio di parametri quali il numero di nodi che tale grafo deve avere, il fatto che si voglia generare un grafo diretto o indiretto, il peso massimo che un arco può avere (impostando il parametro a 1 semplicemente genera grafi non "pesati"), e ultimo, ma non per importanza, la percentuale massima di variazione sul peso degli archi. Quest' ultimo parametro influisce direttamente su quante connessioni verranno create tra i vari nodi. Valori bassi di questo parametro può generare grafi composti da più componenti connesse non collegate tra loro. Se impostato a zero, semplicemente genera un grafo composto da soli nodi, senza archi tra loro.
Nei grafi indiretti si ha che la relativa matrice di adiacenza e simmetrica rispetto alla diagonale. L'algoritmo, in questo caso, non spreca tempo a compilare un'intera matrice per poi specchiarla (buttando via la metà del tempo totale eventualmente richiesto per compilarla), ma genera la matrice simmetrica in maniera intelligente ed efficiente, senza sprechi di tempo.

3 Creazione di Union-Find a partire da matrici di adiacenza

Una Union-Find è una struttura dati atta a ospitare informazioni circa le (eventualmente più di una) componenti connesse che costituiscono un grafo. Le componenti connesse di un grafo sono insiemi disgiunti tra loro, e la Union-Find si dimostra essere una struttura dati efficace non solo nel tenere separate le varie componenti connesse, ma anche ad agevolare l'accorpamento di nodi che appartengono a una stessa componente connessa in un insieme, tramite l'algoritmo Connected-Components, il quale a sua volta si serve di altri tre fondamentali algoritmi: Make-Set, Find-Set, e Union.
Data la matrice di adiacenza di un qualsiasi grafo, non possiamo sapere "a occhio" quali sono le componenti connesse che compongono tale grafo. Ecco allora che l'algoritmo Connected-Components entra in gioco. Infatti, inizialmente identifica i vari nodi a partire dalla matrice di adiacenza, crea un insieme per ognuno di essi grazie all'algoritmo Make-Set, e li aggiunge alla struttura Union-Find. A questo punto, la Union-Find mostrerà di avere tante componenti connesse quanti sono i nodi, ma l'algoritmo è appena iniziato. Adesso identifica tutti gli archi che connettono i vari nodi (ancora una volta l'algoritmo è intelligente: su matrici di adiacenza di grafi indiretti, quindi matrici simmetriche rispetto alla diagonale, non sta a identificare ogni arco come diretto una volta per ogni direzione (raddoppiando i tempi di analisi), ma li identifica come "archi a doppio senso" una volta sola per ogni arco). Per ogni arco identificato controlla se nella Union-Find, il nodo sorgente e quello destinazione fanno parte dello stesso insieme grazie all'algoritmo Find-Set. In caso negativo fa diventare i due insiemi uno solo nella Union-Find, grazie all'algoritmo Union. Union fa un' unione "pesata", cioè è attento ad aggiungere sempre la lista di nodi più corta a quella più

lunga, al fine di ottimizzare i tempi necessari per l'operazione.
Ciò che otteniamo alla fine, è una Union-Find che incorpora tanti insiemi quante sono le componenti connesse del grafo. Ognuno di questi insiemi contiene i nodi che la relativa componente connessa contiene.

4 Minimum Spanning Tree con l'algoritmo di Kruskal

L'algoritmo di Kruskal si propone di identificare, in un grafo pesato indiretto, un Minimum Spanning Tree, o Albero di Connessione Minimo. Un MST è un albero (grafo aciclico e connesso), sottoinsieme del grafo di partenza (con lo stesso numero n di nodi, ma soltanto n-1 archi) che connette tutti i nodi del grafo in modo tale da avere costo minimo. Per costo si intende la somma dei pesi del MST.
A tal fine l'algoritmo di Kruskal da me implementato può generare, dato un grafo indiretto, un MST, e restituirlo sottoforma di Union-Find (giusto a titolo di test, dato che abbiamo sempre le medesime componenti connesse), oppure sottoforma, di nuovo, di matrice di adiacenza (ad un costo leggermente maggiore. Questo è interessante nel caso in cui si voglia vedere effettivamente quali archi sono stati scelti per far parte del MST, o per rappresentazioni grafiche). Questo algoritmo è davvero molto simile a Connected-Components, ma a differenza di quest'ultimo esegue la fase di ricerca e unione partendo dagli archi con peso minimo, e via via cresendo. Salva in un vettore a parte soltanto gli archi che sono stati oggetto di unione. Questo sottoinsieme di archi va propio a definire il MST che verrà poi restituito.

5 Sperimentazione

Verranno condotti 10 sessioni di test sugli algoritmi presentati, e verranno presentati i tempi medi di esecuzione. In particolar modo, in ogni sessione verranno testati i tempi per la creazione di un grafo diretto casuale, i tempi per creare una struttura dati Union-Find a partire da tale grafo diretto, i tempi per la creazione di un grafo indiretto casuale, per creare una struttura Union-Find a partire da tale grafo indiretto, e per eseguire l'algoritmo di Kruskal (con memorizzazione su struttura Union-Find) su tale grafo indiretto. Sulle 10 sessioni, 5 eseguiranno gli algoritmi presentati su grafi di dimensione crescente, dove per dimensione si intende il numero di nodi del grafo, e percentuale massima di variazione sul peso degli archi fissa all' 80%. Avremo inizialmente grafi di un solo nodo, poi di 10 nodi, poi di 100, e infine di 1000 nodi. Le altre 5 sessioni invece eseguono gli stessi algoritmo, ma con numero di nodi dei grafi fisso a 1000, e percentuale massima di variazione sul peso degli archi crescente, da 0% a 100%. I risultati ottenuti sono i segenti:

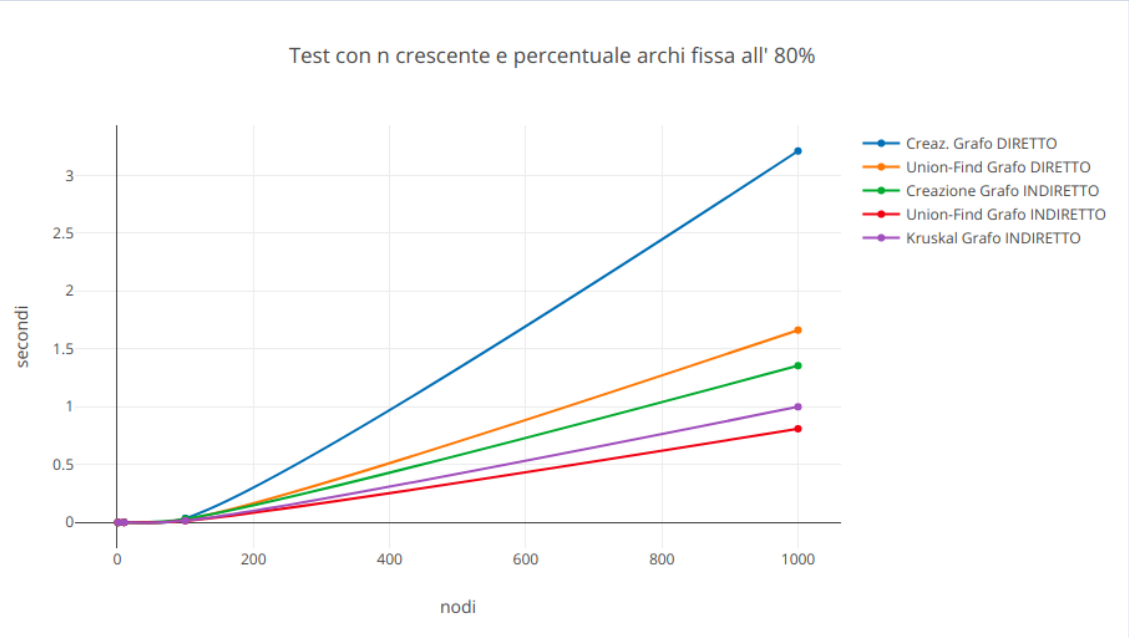

Table 1: n crescente

	Creaz. G DIR	Uni-Find DIR	Creaz. G INDIR	Uni-Find INDIR	Kruskal INDIR
1	0.0000409120000	0.00003740600004	0.00008986099987	0.00010916499991	0.00002492399994
10	0.0004596999999	0.00034614100013	0.00072479799996	0.00023236200013	0.00022664600010
100	0.0362459559999	0.02424546299994	0.03083025200021	0.01464583799997	0.01533432799988
1000	3.2102681429998	1.66232122299993	1.35474095400013	0.80963657899997	0.99910421699996

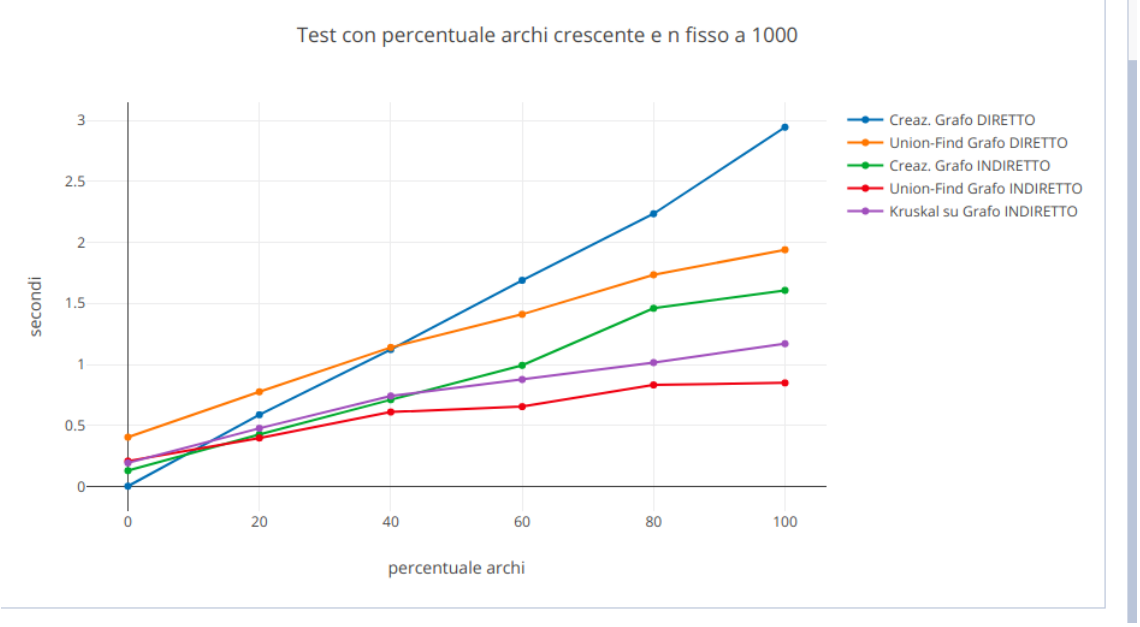

Table 2: percentuale crescente

	Creaz. G DIR	Uni-Find DIR	Creaz. G INDIR	Uni-Find INDIR	Kruskal INDIR
0	0.0019845369999	0.40365771999995	0.1294898230000853	0.2078824850000273	0.1928667390000100
20	0.5872671330000	0.77552619499988	0.4256242090000341	0.3969187860000147	0.4765137279998725
40	1.1216889209999	1.13950331099999	0.7101766969999517	0.6099592549999215	0.7411713859999054
60	1.6894644800001	1.41066877699995	0.9922115469998971	0.6540355300001011	0.8763893299999381
80	2.2347083889999	1.73442262499997	1.4600537569999688	0.8319208660000186	1.0149514760000784
100	2.9443280380000	1.93982146299981	1.6069517659998382	0.8485179840001820	1.1697204070001135

Notiamo che, nel caso dei test con numero n di punti crescente, il tempo di tutte le operazioni cresce in maniera ragionevole e attendibile. L'operazione più costosa di tutte risulta essere la creazione di un grafo diretto casuale. Ci mette più tempo rispetto alla generazione di un grafo indiretto casuale perché rispetto a quest' ultimo deve generare righe della matrice di adiacenza tutte lunghe n, dove invece per il grafo indiretto genera righe di dimensione a partire da n fino ad arrivare a righe di un solo elemento. Inoltre per la generazione di un grafo diretto si ha che vengono creati l'80% di archi per righe lunghe tutte quante n, invece per i grafi indiretti vengono creati l'80% di archi per righe lunghe da n a 1. Quindi effettivamente tra i due algoritmi, quello che genera grafi diretti ha molto di più da fare, e via via di più al crescere di n. Non al variare di n, ma bensì della percentuale di interconnessione tra i nodi, la creazione di un grafo diretto casuale passa da essere l' operazione meno costosa, a essere l'operazione più costosa. Questo perché per percentuali basse, gli archi aggiunti sono molto pochi, per cui ci mette meno tempo ad aggiungerli. Via via aumentando la percentuale vengono fatte molte più variazioni su ogni riga della matrice di adiacenza, e questo porta a far lievitare i costi. Se invece il grafo da generare casualmente è indiretto vale lo stesso discorso fatto prima: devo fare quasi la metà delle operazioni.
Il costo che incide maggiormente sull' algortimo Union-Find, sia questo applicato su grafi diretti o indiretti, è il numero di archi del grafo. Ma questo numero dipende sia dal numero n di nodi, sia dalla percentuale di interconnessione tra i nodi. Si ha però che per grafi diretti deve considerare tutti gli archi, ma su grafi indiretti, quindi matrici simmetriche rispetto alla diagonale, non sta a

considerare ogni arco come diretto una volta per ogni direzione (raddoppiando i tempi di analisi), ma li identifica come "archi a doppio senso" una volta sola per ogni arco. Quindi a parità di numero di archi, la Union-Find da me implementata fa quasi la metà delle operazioni per grafi indiretti, rispetto a grafi diretti.
L'algoritmo di Kruskal, utilizzabile solo su grafi indiretti, fa esattamente quello che fa Union-Find, con la sola differenza che prima di iniziare a considerare gli archi, li ordina in ordine non decrescente di peso. Si nota infatti, dal grafo, che non si discosta molto, come andamento, dall' algoritmo che genera Union-Find.

6 Conclusioni

Complessivamente, dai grafici si vede che al crescere di n, o al crescere della percentuale di interconnessione tra nodi, gli andamenti delle varie operazioni rimangono pressoché invariati. Infatti, guardando l'ultima campionatura (n = 1000 per la tabella con n crescente, e percentuale = 100 per la tabella con la percentuale crescente) si ha addirittura lo stesso ordine relativo tra i costi considerati. Si evince perciò che i parametri variati in queste sessioni di test incidono sui tempi di esecuzione circa in egual maniera.
Andrea Spitaleri