MST: Kruskal e Prim

Ilda Serjanaj

Aprile 2022

Indice

1 Introduzione al problema **2**

2 Caratteristiche teoriche di algoritmi e strutture utilizzate **2**
2.1 Differenze tra Kruskal e Prim 2

3 Esperimenti **3**

4 Risultati **3**
4.1 Tempo id esecuzione all'aumentare della densità della matrice di adiacenza 3
4.2 Tempo di esecuzione al variare del numero di Vertici con Densità archi Bassa 4
4.3 Tempo di esecuzione al variare del numero di Vertici con Densità archi Alta 5

1 Introduzione al problema

Il problema analizzato è quello della ricerca di un **MST: Minimum Spannin Tree** per un grafo. Dato un grafo connesso, uno Spanning tree è un sottografo che collega tutti i vertici. Un singolo grafo può avere diversi Spanning Tree, ma tra tutti questi l'albero con il peso minore è quello che chiamiamo MST (il peso si uno Spanning Tree).
Per trovare un **MST** ci sono vari algoritmi, quelli che andremo a comparare sono **Kruskal** e **Prim**.

2 Caratteristiche teoriche di algoritmi e strutture utilizzate

Kruskal Algorithm:
Questo algoritmo parte ordinando tutti i bordi dal meno costoso al più costoso. Sceglie lo spigolo più piccolo e controlla se non genera un ciclo con lo Spanning Tree formato finora, se non si forma viene aggiunto al bordo, altrimenti viene scartato. Ripetere questo passaggio fino a quando un numero di bordi pari a (V-1) nell'albero spanning.

Prim Algorithm:
Come l'algoritmo di Kruskal, anche l'algoritmo di **Prim** è un **algoritmo Greedy**(o Goloso, sono quegli algoritmi che compiono, ad ogni passo, la scelta migliore in quel momento piuttosto che adottare una strategia a lungo termine).
Si inizia con uno Spanning Tree vuoto. Prendiamo due insiemi, uno che contiene tutti i vertici inclusi nell'MST e nell'altro quelli non ancora inclusi. Ad ogni passo, si considerano tutti i bordi che collegano i due insiemi e si sceglie il bordo con il peso minore tra tutti.

2.1 Differenze tra Kruskal e Prim

Entrambi sono algoritmi golosi ma con differenze sostanziali.

- **Prim**
  - Inizia a costruire il Minimum Spanning Tree da qualsiasi vertici del Grafo.
  - Ha una complessita temporale di $\Theta(ElogV)$
  - Funziona solo su grafo connesso
  - Funziona efficacemente su Grafi Densi
- **Kruskal**
  - Inizia a costruire il Minimum Spanning Tree dal vertice che porta il peso minimo del grafo
  - Attraversa un nodo una sola volta
  - complessità temporale è $\Theta(ElogV)$
  - Può generare Foreste e può funzionare su componenti disconnesse
  - Funziona più velocemente su Grafi sparsi

3 Esperimenti

Effettuiamo i test sull'aumentare ad ogni ciclo in numero di vertici V e la percentuale di densità (quindi il numero di E) della matrice, per vedere il comportamento dei due algoritmi. Partendo da un numero di nodi di 50 aumentando ad ogni ciclo di 100 fino a 450.

4 Risultati

4.1 Tempo id esecuzione all'aumentare della densità della matrice di adiacenza

Come si vede dal grafico 3 l'algoritmo di **Prim** è più efficace su Grafi Densi rispetto a Kruskal.

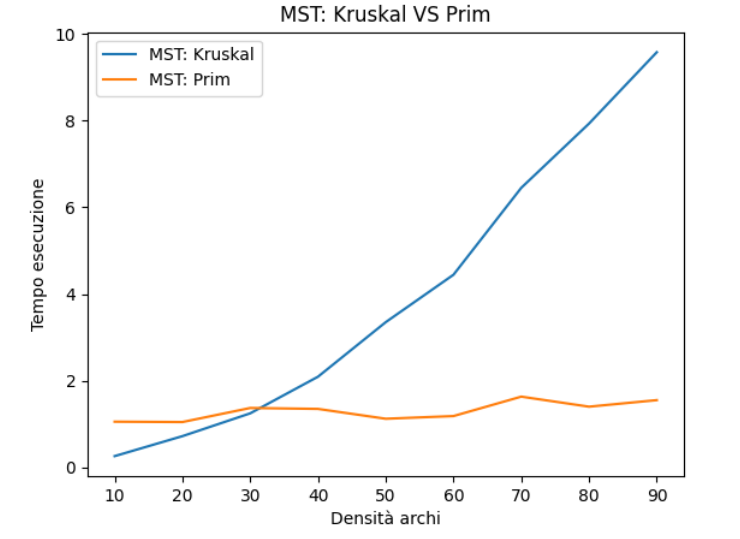

Figura 1: Tempo di esecuzione al crescere della percentuale della densità della matrice

4.2 Tempo di esecuzione al variare del numero di Vertici con Densità archi Bassa

Da questi due grafici 4 possiamo vedere l'andamento temporale nel caso di aumento del numero dei Nodi con grafi sparsi, più diminuisce la densità più **Kruskal risulta efficace**

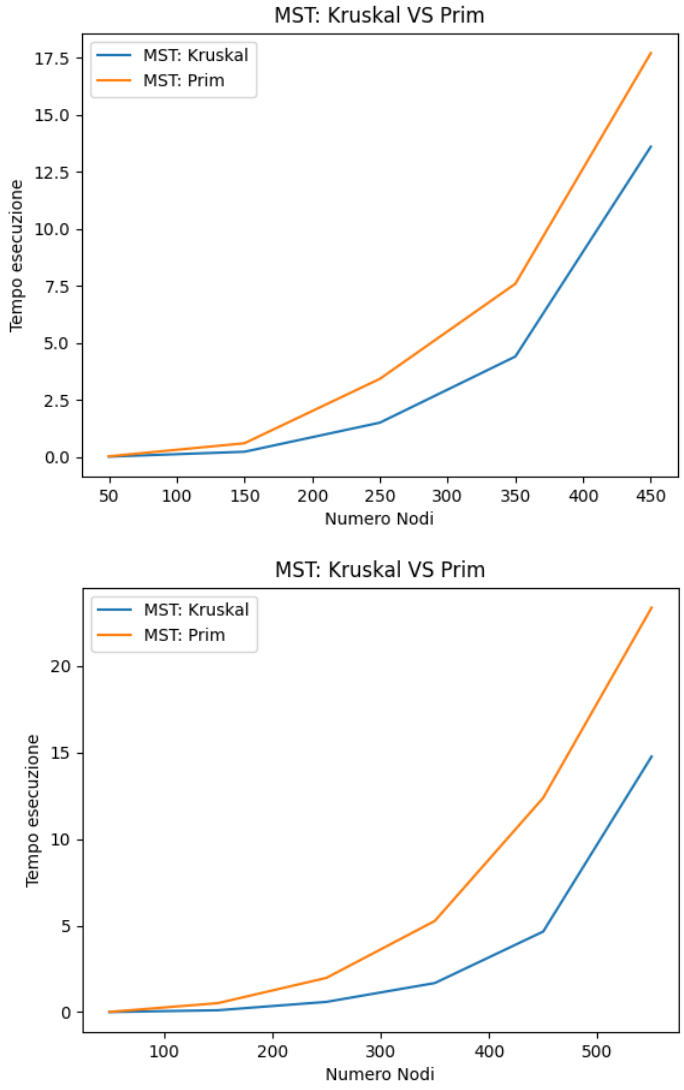

Figura 2: Aumento Nodi con percentuale Densità 20 e 10

4.3 Tempo di esecuzione al variare del numero di Vertici con Densità archi Alta

Al contrario di quanto visto sopra, ovvero al crescere della densità aumenta l'efficenza di **Prim** rispetto a **Kruskal**, fino ad arrivare nel caso di Percentuale densità completa (100) a impiegare un minuto per trovare l'MST con 350 nodi.

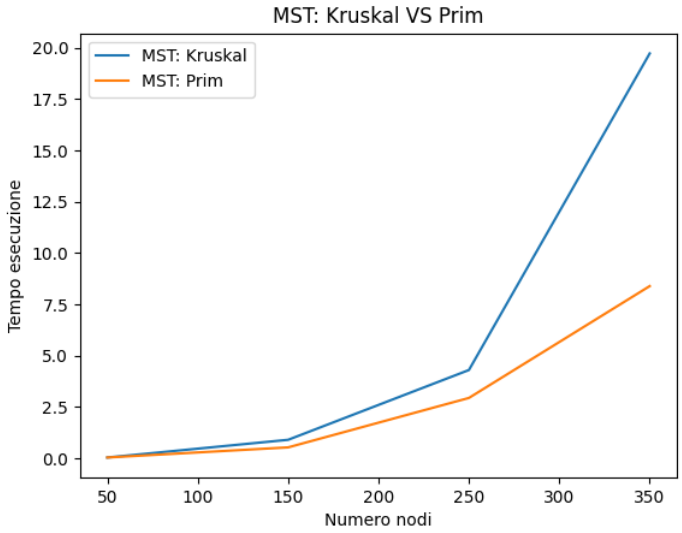

Figura 3: Aumento numero nodi con percentuale Densità 80

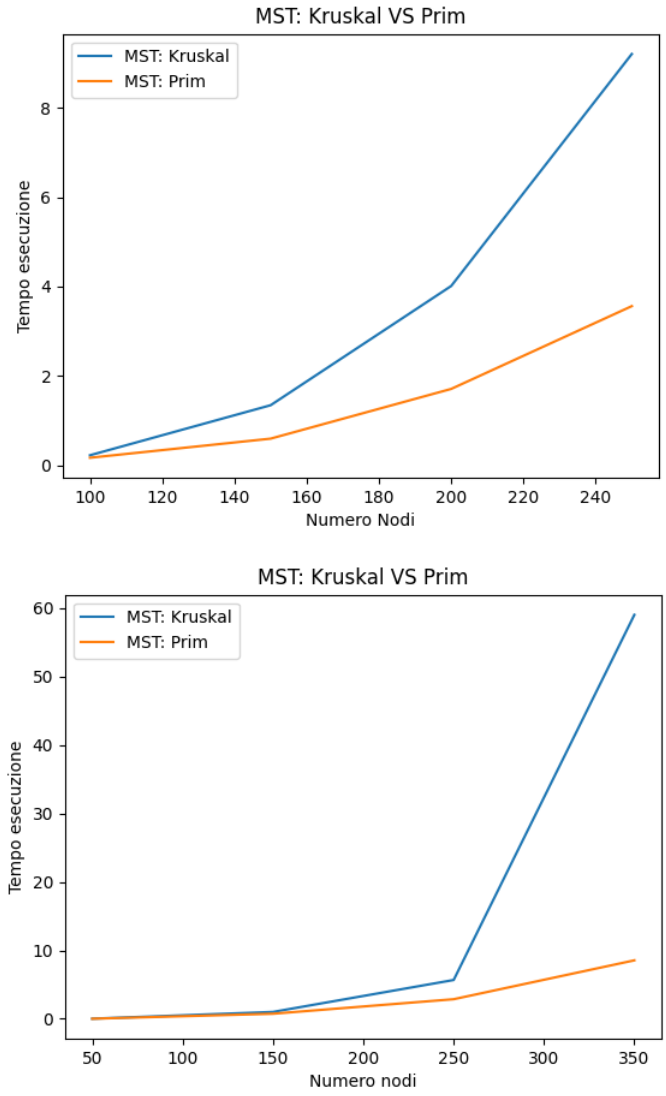

Figura 4: Percentuale densità 100