FLY, API per grafi

Simone Bisogno

26 luglio 2019

Sommario

Nella presente relazione è riportata l'interfaccia pubblica per il tipo di dato astratto *grafo* per il linguaggio di programmazione FLY, sviluppato all'interno del laboratorio ISISLab del Dipartimento di Informatica dell'Università degli Studi di Salerno.

In due dei linguaggi di programmazione più famosi e utilizzati – Java per la programmazoine orientata agli oggetti, Python per lo scripting – le librerie che rappresentano lo stato dell'arte nell'implementazione e gestione dell'ADT grafo sono, rispettivamente, **JGraphT** e **NetworkX**. In quanto fondato su entrambi questi linguaggi di programmazione, per l'introduzione dell'ADT grafo all'interno del linguaggio è stato necessario selezionare la migliore libreria per grafi per ogni linguaggio e poi metter a fattor comune le loro funzionalità, tenendo conto della diversa natura dei due linguaggi.

Qui è stata creata un'intersezione minimale delle API delle due librerie precedentemente citate per la creazione dell'API FLY per la gestione dei grafi.

Indice

Capitolo 1

API comune delle librerie JGraphT e NetworkX

1.1 Struttura

L'API comune si compone delle seguenti sezioni:

- manipolazione del grafo
- lettura e scrittura del grafo su file
- visita di un grafo
- individuazione componenti connesse
- ordinamento topologico
- albero di copertura minimo

1.2 Creazione

JGraphT La creazione di un grafo in JGraphT può essere realizzata instanziando oggetti di diverse classi per ottenere le possibili tipologie di grafo; il modo più comodo è l'utilizzo della classe `org.jgrapht.graph.builder.GraphTypeBuilder`[1] che permette la concatenazione di metodi per impostare le caratteristiche del grafo desiderato.

NetworkX In NetworkX, un'istanza di grafo è ottenuta tramite la funzione `networkx.Graph()`; non sono necessarie diverse implementazioni di grafo: ove necessario, le informazioni aggiuntive sono salvate in dizionari associati all'entità interessata (per esempio, il peso degli archi).

[1] `GraphTypeBuilder` realizza il design pattern creazionale *builder*.

1.3 Manipolazione

Di seguito sono presentate le funzioni di API che permettono di gestire gli aspetti base di un grafo come l'aggiunta o la rimozione di nodi o di archi, il vicinato di un nodo, eccetera.

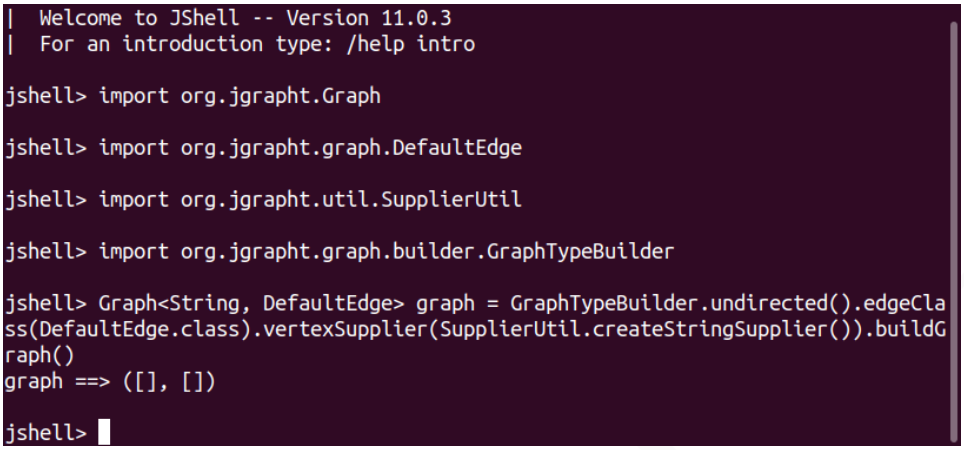

Figura 1.1: Creazione di un grafo con JGraphX all'interno della shell Java *JShell*.

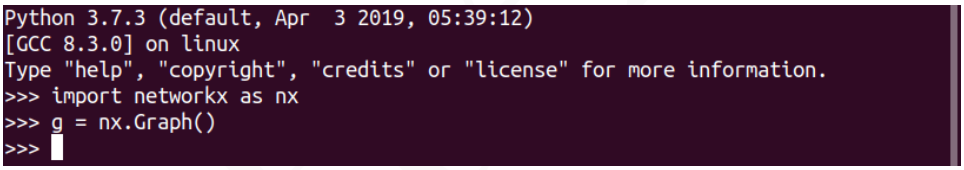

Figura 1.2: Creazione di un grafo con NetworkX all'interno della shell Python.

I metodi di JGraphT, ove non espressamente indicato, sono tutti appartenenti all'interfaccia `org.jgrapht.Graph`.

1.3.1 Aggiunta nodo

JGraphT

- metodo: `addVertex(V)`
- parametri: nodo di tipo `V`
- restituisce: `boolean`, indica il corretto inserimento del nodo

NetworkX

- funzione: `add_node(v)`
- parametri: `v`, oggetto che rappresenta un nodo
- restituisce: nulla

1.3.2 Grado di un nodo

JGraphT

- metodo: `degreeOf(V)`
- parametri: nodo di tipo `V`
- restituisce: il grado del nodo (`int`)

NetworkX

- funzione: `degree(v)`
- parametri: `v` è un oggetto che rappresenta un nodo
- restituisce: il grado del nodo (`int`)

1.3.3 Grado entrante di un nodo

JGraphT

- metodo: `inDegreeOf(V)`
- parametri: nodo di tipo `V`
- restituisce: il grado entrante del nodo (`int`)

NetworkX

- funzione: `in_degree(v)`
- parametri: `v` è un oggetto che rappresenta un nodo
- restituisce: il grado entrante del nodo (`int`)

1.3.4 Grado uscente di un nodo

JGraphT

- metodo: `outDegreeOf(V)`
- parametri: nodo di tipo `V`
- restituisce: il grado uscente del nodo (`int`)

NetworkX

- funzione: `out_degree(v)`
- parametri: `v` è un oggetto che rappresenta un nodo
- restituisce: il grado uscente del nodo (`int`)

1.3.5 Vicinato

JGraphT

- metodo: `edgesOf(V)`
- parametri: nodo di tipo `V`
- restituisce: l'insieme degli archi incidenti (`Set<E>`)

NetworkX

- funzione: `list(g.adj[v])`
- parametri: `g` è un oggetto che rappresenta un grafo, `v` è un oggetto che rappresenta un nodo
- restituisce: una lista di archi incidenti su `v`

1.3.6 Archi entranti di un nodo

JGraphT

- metodo: `incomingEdgesOf(V)`
- parametri: nodo di tipo `V`
- restituisce: l'insieme degli archi entranti (`Set<E>`)

NetworkX

- funzione: `list(g.in_edges(nbunch=v))`
- parametri: `g` è un oggetto che rappresenta un grafo, `v` è un oggetto che rappresenta un nodo
- restituisce: una lista di archi entranti in `v`

1.3.7 Stella uscente di un nodo

JGraphT

- metodo: `outgoingEdgesOf(V)`
- parametri: nodo di tipo `V`
- restituisce: l'insieme degli archi uscenti (`Set<E>`)

NetworkX

- funzione: `list(g.out_edges(nbunch=v))`
- parametri: `g` è un oggetto che rappresenta un grafo, `v` è un oggetto che rappresenta un nodo
- restituisce: una lista di archi uscenti da `v`

1.3.8 Nodi di un grafo

JGraphT

- metodo: `vertexSet()`
- parametri: nessuno
- restituisce: l'insieme di nodi del grafo (`Set<V>`)

NetworkX

- funzione: `list(g.nodes)`
- parametri: `g` è un oggetto che rappresenta un grafo
- restituisce: la lista di nodi del grafo

1.3.9 Quantità nodi

JGraphT

- metodo: `vertexSet().size()`
- parametri: nessuno
- restituisce: la cardinalità dell'insieme dei nodi (`int`)

NetworkX

- funzione: `order()`
- parametri: nessuno
- restituisce: la cardinalità dell'insieme dei nodi (`int`)
- funzione: `number_of_nodes()`
- parametri: nessuno
- restituisce: la cardinalità dell'insieme dei nodi (`int`)

1.3.10 Rimozione nodo

JGraphT

- metodo: `removeVertex(V)`
- parametri: nodo di tipo `V`
- restituisce: `boolean`, indica la corretta rimozione del nodo

NetworkX

- funzione: `remove_node(v)`
- parametri: `v` è un oggetto che rappresenta un nodo
- restituisce: nulla
- note: la rimozione di un nodo non presente nel grafo provoca un `NetworkXError` con il messaggio *"The node e is not in the graph"*

1.3.11 Presenza nodo

JGraphT

- metodo: `containsVertex(V)`
- parametri: nodo di tipo `V`
- restituisce: `boolean`, indica la presenza del nodo nel grafo

NetworkX

- funzione: `has_node(v)`
- parametri: `v` è un oggetto che rappresenta un nodo
- restituisce: `Boolean`, indica la presenza del nodo nel grafo

1.3.12 Aggiunta arco

JGraphT

- metodo: `addEdge(V, V)`
- parametri: due nodi di tipo `V`
- restituisce: l'arco creato (`E`)

NetworkX

- funzione: `add_edge(v1, v2)`
- parametri: `v1` e `v2` sono due oggetti che rappresentano due nodi
- restituisce: nulla

1.3.13 Arco tra due nodi

JGraphT

- metodo: `getEdge(V, V)`
- parametri: due nodi di tipo `V`
- restituisce: l'arco tra i due nodi, se esiste (`E`)

NetworkX

- funzione: `edges[v1, v2]`
- parametri: `v1` e `v2` sono due oggetti che rappresentano due nodi
- restituisce: l'arco tra i due nodi, se esiste

1.3.14 Archi di un grafo

JGraphT

- metodo: `edgeSet()`
- parametri: nessuno
- restituisce: l'insieme degli archi di un grafo (`Set<E>`)

NetworkX

- funzione: `list(g.edges)`
- parametri: `g` è un oggetto che rappresenta un grafo
- restituisce: la lista di archi del grafo

1.3.15 Quantità archi

JGraphT

- metodo: `edgeSet().size()`
- parametri: nessuno
- restituisce: la cardinalità dell'insieme di archi (`int`)

NetworkX

- funzione: `size()`
- parametri: nessuno
- restituisce: la cardinalità dell'insieme di archi (`int`)

1.3.16 Peso di un arco

JGraphT

- metodo: `getEdgeWeight(E)`
- parametri: un arco di tipo `E`
- restituisce: il peso dell'arco (`double`)
- note: il metodo accessore e quello modificatore sono gli unici metodi che richiedono esclusivamente la specifica di un oggetto del tipo dell'arco piuttosto che della coppia di nodi suoi estremi; per utilizzare i due nodi agli estremi, passare come argomento l'invocazione al metodo `getEdge(V, V)`

NetworkX

- espressione: `g[v1][v2]['weight']`
- parametri: `v1` e `v2` sono due oggetti che rappresentano due nodi, `weight` è l'attributo dell'arco associato al peso
- restituisce: il peso dell'arco (`float`)

1.3.17 Modifica peso di un arco

JGraphT

- metodo: `setEdgeWeight(E, double)`
- parametri: un arco di tipo `E`, un peso di tipo `double`
- restituisce: nulla
- note: il metodo accessore e quello modificatore sono gli unici metodi che richiedono esclusivamente la specifica di un oggetto del tipo dell'arco piuttosto che della coppia di nodi suoi estremi; per utilizzare i due nodi agli estremi, passare come argomento l'invocazione al metodo `getEdge(V, V)`

NetworkX

- espressione: `g[v1][v2]['weight'] = new_weight`
- parametri: `v1` e `v2` sono due oggetti che rappresentano due nodi, `weight` è l'attributo dell'arco associato al peso, `new_weight` è il valore del nuovo peso
- restituisce: nulla

1.3.18 Rimozione arco

JGraphT

- metodo: `removeEdge(V, V)`
- parametri: due nodi di tipo `V`
- restituisce: l'arco rimosso (`E`)

NetworkX

- funzione: `remove_edge(v1, v2)`
- parametri: `v1` e `v2` sono due oggetti che rappresentano due nodi
- restituisce: nulla

1.3.19 Presenza arco

JGraphT

- metodo: `containsEdge(V, V)`
- parametri: due nodi di tipo `V`
- restituisce: `boolean`, indica la presenza dell'arco nel grafo

NetworkX

- funzione: `has_edge(v1, v2)`
- parametri: `v1` e `v2` sono due oggetti che rappresentano due nodi
- restituisce: `Boolean`, indica la presenza dell'arco nel grafo

1.4 I/O

I metodi qui descritti sono relativi a classi del pacchetto `org.jgrapht.io` di JGraphT, mentre le funzioni sono definite nel pacchetto `networkx` di NetworkX.

Il formato utilizzato per la serializzazione dei grafi è la lista di archi (*edge list*), una delle rappresentazioni più comuni per i grafi.

1.4.1 Carica grafo da file

Per il caricamento di un grafo da file in JGraphT, è necessario prima creare un oggetto del tipo dell'interfaccia `GraphImporter<V, E>`, dove `V` è il tipo generico del nodo ed `E` quello dell'arco; la classe `CSVImporter<V, E>` realizza i metodi della su citata interfaccia per il caricamento di un grafo in rappresentazione *edge list*.

JGraphT

- metodo: `importGraph(Graph<V, E>, Reader)`
- parametri:
  - un grafo con tipo di nodi `V` e tipo di archi `E`
  - un oggetto `Reader` per l'accesso in lettura a un flusso di caratteri
- restituisce: nulla
- note: il metodo legge la rappresentazione dal flusso di caratteri in input e la salva all'interno del grafo dato

NetworkX

- funzione: `read_edgelist(path, delimiter=' ')`
- parametri:
  - `path` è il percorso del file da cui leggere la lista di archi di un grafo

  - `delimiter` è un parametro opzionale a cui può essere associato come valore il carattere da utilizzare per separare i vertici di un arco all'interno della lista (di default è *None*)

- restituisce: il grafo letto dal flusso
- funzione: `read_weighted_edgelist(path, delimiter=' ')`
- parametri:
  - `path` è il percorso del file da cui leggere la lista di archi di un grafo
  - `delimiter` è un parametro opzionale a cui può essere associato come valore il carattere da utilizzare per separare i vertici di un arco all'interno della lista (di default è *None*)
- restituisce: il grafo pesato letto dal flusso

1.4.2 Salva grafo su file

Per il salvataggio di un graf su file in JGraphT, è necessario prima creare un oggetto del tipo dell'interfaccia `GraphExporter<V, E>`, dove `V` è il tipo generico del nodo ed `E` quello dell'arco; la classe `CSVExporter<V, E>` realizza i metodi della su citata interfaccia per il salvataggio di un grafo in rappresentazione *edge list*.

JGraphT

- metodo: `exportGraph(Graph<V, E>, Writer)`
- parametri:
  - un grafo con tipo di nodi `V` e tipo di archi `E`
  - un oggetto `Writer` per l'accesso in scrittura a un flusso di caratteri
- restituisce: nulla
- note: il metodo scrive il grafo dato all'interno del flusso di caratteri in output come lista di archi

NetworkX

- funzione: `write_edgelist(G, path, delimiter=' ')`
- parametri:
  - `G` è un oggetto che rappresenta un grafo

  - `path` è il percorso del file in cui scrivere la lista di archi di `G`
  - `delimiter` è un parametro opzionale a cui può essere associato come valore il carattere da utilizzare per separare i vertici di un arco all'interno della lista (di default è *None*)

- restituisce: nulla

- funzione: `write_weighted_edgelist(G, path, delimiter=' ')`

- parametri:

  - `G` è un oggetto che rappresenta un grafo
  - `path` è il percorso del file in cui scrivere la lista di archi di `G`
  - `delimiter` è un parametro opzionale a cui può essere associato come valore il carattere da utilizzare per separare i vertici di un arco all'interno della lista (di default è *None*)

- restituisce: nulla

1.5 Visita

I metodi qui descritti sono relativi a classi del pacchetto `org.jgrapht.traverse` di JGraphT, mentre le funzioni sono definite nel pacchetto `networkx` di NetworkX.

Le visite sono eseguite in JGraphT instanziando un iteratore del tipo interfaccia `GraphIterator<V, E>`, mentre in NetworkX è sufficiente l'invocazione di funzioni a cui viene passato il grafo come parametro.

1.5.1 Visita in ampiezza

La classe JGraphT che realizza l'iteratore sul grafo per la visita in ampiezza è `BreadthFirstIterator(Graph<V, E>, V)`.

JGraphT

- metodo: `BreadthFirstIterator(Graph<V, E>, V)`

- parametri:

  - un grafo con tipo di nodi `V` e tipo di archi `E`
  - un nodo di tipo `V` da cui iniziare la visita

- restituisce: un iteratore sui nodi del grafo

NetworkX

- funzione: `list(bfs_edges(G, root))`
- parametri:
  - `G` è un oggetto che rappresenta un grafo
  - `root` è il nodo da cui iniziare la visita
- restituisce: lista di archi appartenenti all'albero di visita
- nota: per ottenere la lista di nodi visitati, è possibile utilizzare l'espressione `nodes = [root] + [v for u, v in edges]`, dove `edges` è la lista di archi visitati

1.5.2 Visita in profondità

La classe JGraphT che realizza l'iteratore sul grafo per la visita in profondità è `DepthFirstIterator(Graph<V, E>, V)`.

JGraphT

- metodo: `DepthFirstIterator(Graph<V, E>, V)`
- parametri:
  - un grafo con tipo di nodi `V` e tipo di archi `E`
  - un nodo di tipo `V` da cui iniziare la visita
- restituisce: un iteratore sui nodi del grafo

NetworkX

- funzione: `list(dfs_edges(G, source=root))`
- parametri:
  - `G` è un oggetto che rappresenta un grafo
  - `source` (opzionale) è il nodo da cui iniziare la visita
- restituisce: lista di archi appartenenti all'albero di visita
- nota: per ottenere la lista di nodi visitati, è possibile utilizzare l'espressione `nodes = [root] + [v for u, v in edges]`, dove `edges` è la lista di archi visitati

1.6 Connettività

I metodi qui descritti sono relativi a classi del pacchetto `org.jgrapht.alg.connectivity` di JGraphT, mentre le funzioni sono definite nel pacchetto `networkx` di NetworkX.

In particolare, la classe che realizza l'ispezione del grafo per la connettività è `ConnectivityInspector<V, E>`, al cui costruttore va passato il grafo da ispezionare; per la connettività forte, la classe da utilizzare è `KosarajuStrongConnectivityInspector<V, E>` mentre non è definita una classe o un metodo per la verifica della connettività debole.

1.6.1 Verifica

JGraphT

- metodo: `isConnected()`
- parametri: nessuno
- restituisce: `boolean`, indica lo stato di connessione del grafo

NetworkX

- funzione: `is_connected(G)`
- parametri: `G` è un oggetto che rappresenta un grafo
- restituisce: `Boolean`, indica lo stato di connessione del grafo

1.6.2 Verifica connettività forte

JGraphT

- metodo: `isStronglyConnected()`
- parametri: nessuno
- restituisce: `boolean`, indica lo stato di connessione forte del grafo direzionato
- nota: il metodo va utilizzato su un ispettore di tipo `KosarajuStrongConnectivityInspector<V, E>`

NetworkX

- funzione: `is_strongly_connected(G)`
- parametri: `G` è un oggetto che rappresenta un grafo direzionato
- restituisce: `Boolean`, indica lo stato di connessione forte del grafo

1.6.3 Componenti connesse

JGraphT

- metodo: `connectedSets()`
- parametri: nessuno
- restituisce: una lista di insiemi in cui ogni insieme contiene i nodi nella stessa componente (`List<Set<V>>`)

NetworkX

- funzione: `connected_components(G)`
- parametri: `G` è un oggetto che rappresenta un grafo
- restituisce: un generatore per una lista di insiemi di nodi, ordinata per cardinalità
- nota: per estrarre le componenti connesse dal generatore, è possibile usare l'espressione `list(set for set in connected_components(G))`; per estrarre i sottografi connessi, è possibile usare l'espressione `list(G .subgraph(set) for set in connected_components(G))`

1.6.4 Quantità componenti connesse

JGraphT

- metodo: `connectedSets().size()`
- parametri: nessuno
- restituisce: il numero di componenti connesse del grafo (`int`)

NetworkX

- funzione: `number_connected_components(G)`
- parametri: `G` è un oggetto che rappresenta un grafo
- restituisce: il numero di componenti connesse del grafo (`int`)

1.6.5 Componente connessa di un nodo

JGraphT

- metodo: `connectedSetOf(V)`
- parametri: nodo di tipo `V`
- restituisce: l'insieme dei nodi che costituisce la componente connessa del nodo dato (`Set<V>`)

NetworkX

- funzione: `node_connected_component(G, v)`
- parametri:
  - `G` è un oggetto che rappresenta un grafo
  - `v` è un oggetto che rappresenta un nodo
- restituisce: l'insieme dei nodi che costituisce la componente connessa del nodo dato

1.6.6 Componenti fortemente connesse

Entrambe le librerie utilizzano l'algoritmo di Kosaraju-Sharir per la determinazione delle componenti fortemente connesse di un grafo.

JGraphT

- metodo: `getStronglyConnectedComponents()`
- parametri: nessuno
- restituisce: la lista di sottografi fortemente connessi del grafo ispezionato (`List<Graph<V, E>>`)
- nota: il metodo va utilizzato su un ispettore di tipo `KosarajuStrongConnectivityInspector<V, E>`
- metodo: `getStronglyConnectedSets()`
- parametri: nessuno
- restituisce: la lista di insiemi di nodi fortemente connessi del grafo ispezionato (`List<Graph<V, E>>`)
- nota: il metodo va utilizzato su un ispettore di tipo `KosarajuStrongConnectivityInspector<V, E>`

NetworkX

- funzione: `kosaraju_strongly_connected_components(G)`
- parametri: `G` è un oggetto che rappresenta un grafo direzionato
- restituisce: un generatore di insiemi di nodi
- nota: per estrarre le componenti fortemente connesse dal generatore, è possibile usare l'espressione `list(set for set in kosaraju_connected_components(G))`; per estrarre i sottografi fortemente connessi, è possibile usare l'espressione `list(G.subgraph(set) for set in kosaraju_connected_components(G))`

1.7 Digrafi aciclici (DAG) e ordinamento topologico

I metodi qui descritti sono relativi alle classi `TopologicalOrderIterator<V, E>` del pacchetto `org.jgrapht.traverse` di JGraphT e `CycleDetector<V, E>` di `org.jgrapht.alg.cycle`, mentre le funzioni sono definite nel pacchetto `networkx` di NetworkX.

1.7.1 Test ciclicità

Il test di ciclicità determina se un grafo direzionato è provvisto di cicli.

In JGraphT viene testata esplicitamente la presenza di cicli in un grafo direzionato con il metodo `detectCycles()` della classe `CycleDetector<V, E>` costruendone un oggetto passando il grafo direzionato come argomento al costruttore; in NetworkX viene testata la proprietà del grafo di essere direzionato e aciclico.

JGraphT

- metodo: `detectCycles()`
- parametri: nessuno
- restituisce: `boolean`, indica la presenza di almeno un ciclo nel grafo direzionato
- note: `detectCycles()` va invocato su un oggetto `CycleDetector<V, E>` costruito con un grafo direzionato (viene lanciata una `IllegalArgumentException` altrimenti)

NetworkX

- funzione: `is_directed_acyclic_graph(G)`
- parametri: `G` è un oggetto che rappresenta un grafo
- restituisce: `Boolean`, indica se il grafo è diretto aciclico
- nota: non è imposto che `G` sia creato con `DiGraph()`

1.7.2 Ordinamento topologico

In JGraphT l'ordinamento topologico viene definito creando un iteratore di tipo `TopologicalOrderIterator<V, E>` (il quale parte da un nodo sorgente qualsiasi del grafo); parimodo, in NetworkX l'ordinamento topologico viene definito con un generatore ottenuto dall'invocazione del metodo `topological_sort(G)`.

In entrambe le librerie, le implementazioni scelgono un nodo sorgente (non è specificato espressamente come) e da lì calcolano l'ordinamento.

JGraphT

- metodo: `TopologicalOrderIterator(Graph<V, E>)`
- parametri:
  - un grafo con tipo di nodi `V` e tipo di archi `E`
- restituisce: un iteratore sull'ordinamento topologico

NetworkX

- funzione: `topological_sort(G)`
- parametri: `G` è un oggetto che rappresenta un grafo direzionato
- restituisce: un generatore sull'ordinamento topologico
- nota: per avere la lista, è sufficiente avvolgere la chiamata a funzione con `list()`

1.8 Albero di copertura minimo

In JGraphT, l'albero di copertura minimo viene realizzato instanziando un oggetto del tipo dell'interfaccia `SpanningTreeAlgorithm<E>` di `org.jgrapht.alg.interfaces`, per esempio `KruskalMinimumSpanningTree<V, E>` oppure `PrimMinimumSpanningTree<V, E>`, entrambi di `org.jgrapht.alg.spanning`; in NetworkX, la funzione per il calcolo dell'albero di copertura minimo è contenuta nel pacchetto `networkx`.

L'algoritmo scelto per l'individuazione dell'albero minimo ricoprente è *Prim*.

1.8.1 Albero

JGraphT

- metodo: `getSpanningTree()`
- parametri: nessuno
- restituisce: l'albero di copertura minimo (`SpanningTreeAlgorithm.SpanningTree<E>`)
- note: il metodo va invocato su un oggetto di tipo `PrimMinimumSpanningTree<V, E>`, al cui costruttore va passato il grafo pesato da cui estrarre l'albero minimo ricoprente

NetworkX

- funzione: `minimum_spanning_tree(G, algorithm='prim')`
- parametri:
  - `G` è un oggetto che rappresenta un grafo pesato
  - `algorithm` indica l'algoritmo da utilizzare per il calcolo dell'albero minimo ricoprente: può essere `kruskal`, `prim` oppure `boruvka`.
- restituisce: un grafo che rappresenta un minimo albero ricoprente oppure una foresta
- nota: per l'algoritmo di Borůvka, è richiesto che tutti gli archi siano pesati e che i pesi siano a due a due distinti; per gli altri algoritmi, in caso di arco non pesato, viene considerato come peso predefinito il valore `1`