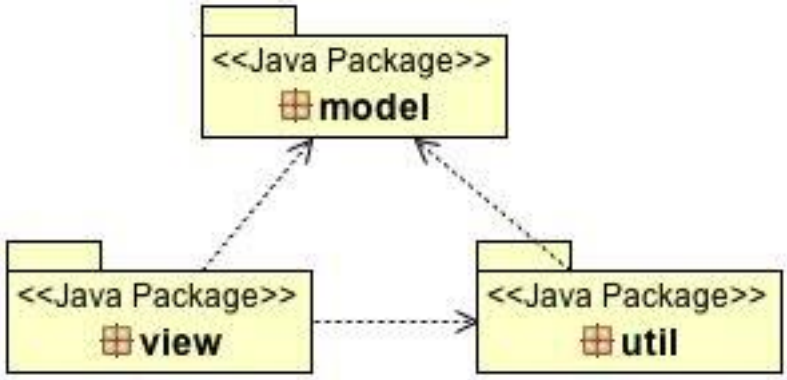
Ingegneria del Software

Progetto: Critical Path Method

Il Critical Path Method ha come obiettivo la realizzazione del grafo delle attività di un eventuale progetto con relativa ricerca del percorso critico. Si vuole dunque effettuare una distinzione tra quelle attività che devono assolutamente rispettare dei tempi di consegna da quelle attività che possono ritardare senza influire sul ritardo complessivo. Il cliente avrà la possibilità di inserire dei nodi (punti chiave del progetto) e degli archi pesati (attività) e il software calcolerà e restituirà il percorso critico trovato.

Il progetto è stato strutturato secondo il seguente package diagram:



Il package **view** fornisce le classi per l' interfaccia grafica. Il package **model** contiene le classi relative all'implementazione del grafo ed infine il package **util** contiene classi di utilità indispensabili al calcolo del percorso critico.

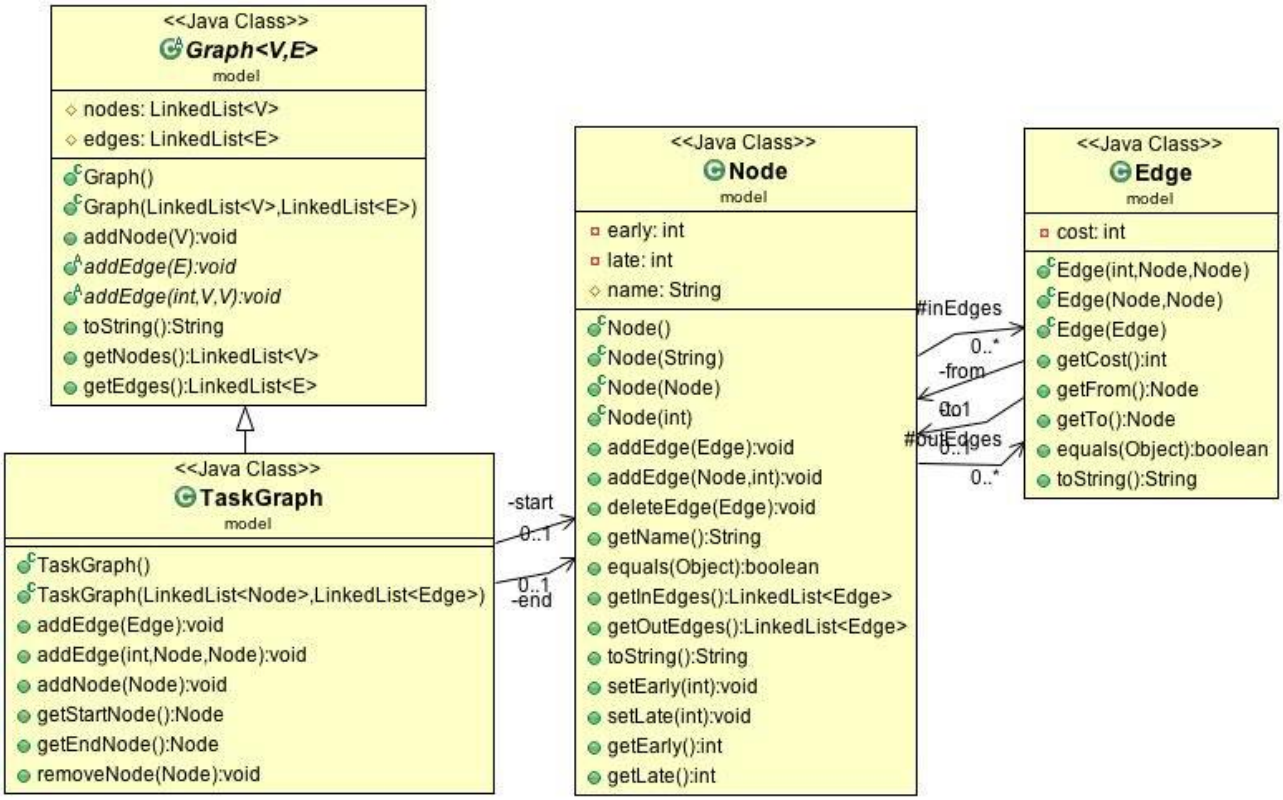
Uno dei casi d'uso del sistema è il seguente:

1. L' utente inserisce nodi ed archi.
2. Chiede al sistema di risolvere lo schema.
3. Visualizza il risultato calcolato dal sistema.
4. Azzera i dati inseriti e ne inserisce di nuovi.

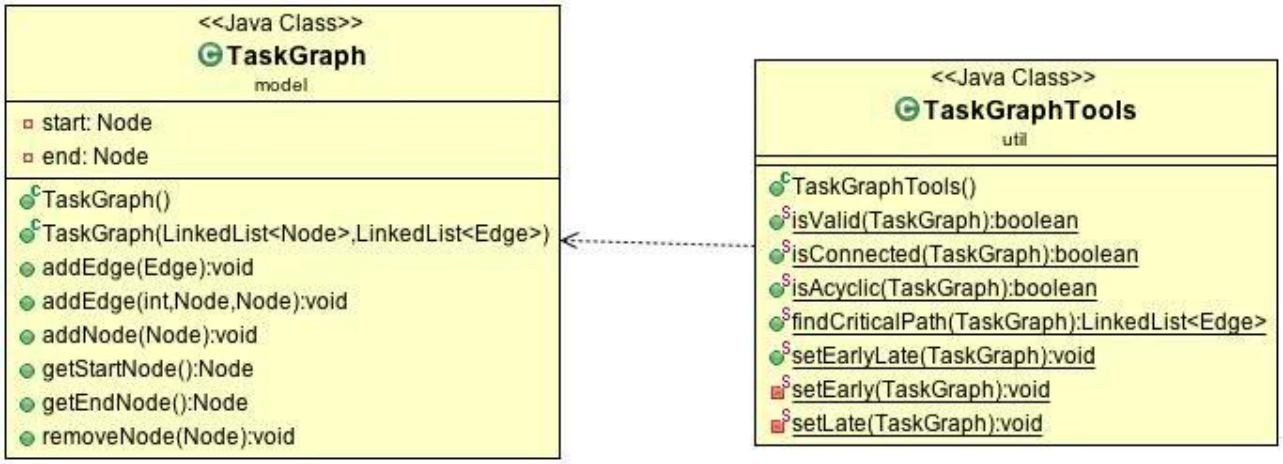
Per la realizzazione del grafo è stata dapprima realizzata la classe astratta **Graph<V,E>** parametrica nei nodi e negli archi. Sono stati concretizzati quanti più metodi possibili e successivamente è stata estesa dalla classe specializzata **TaskGraph** che implementa i metodi astratti relativi all'aggiunta degli archi e introduce nuovi metodi utili alla gestione del grafo delle attività. Le classi **Node** e **Edge** rappresentavo rispettivamente i nodi e gli archi del grafo. Oltre alla propria etichetta, i nodi racchiudono al loro interno informazioni riguardanti il T-Early ed il T-Late. Un TaskGraph tiene traccia di nodi e

archi ed ogni singolo nodo tiene traccia dei suoi archi entranti ed uscenti. Questo facilita alcune operazioni come l'analisi dei nodi adiacenti.

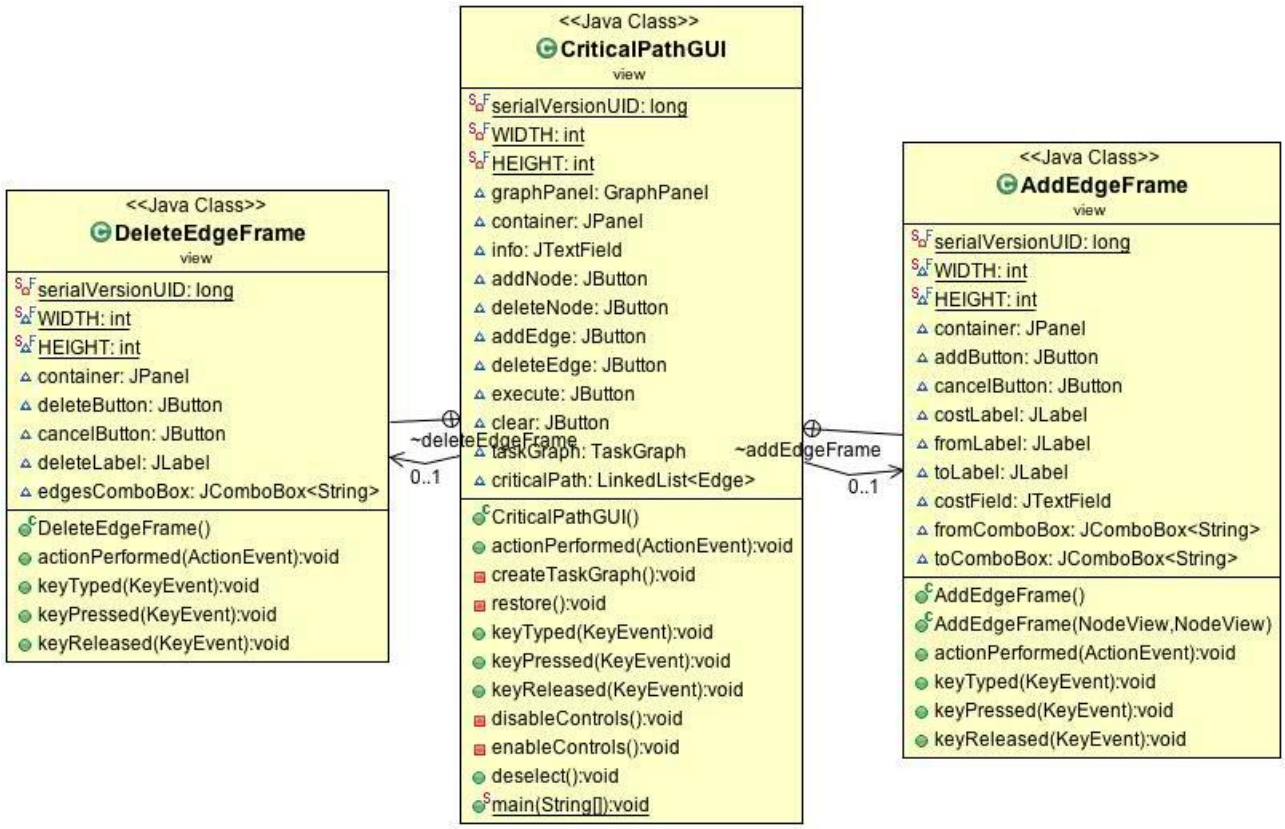
Di seguito il diagramma delle classi che modellano il grafo:



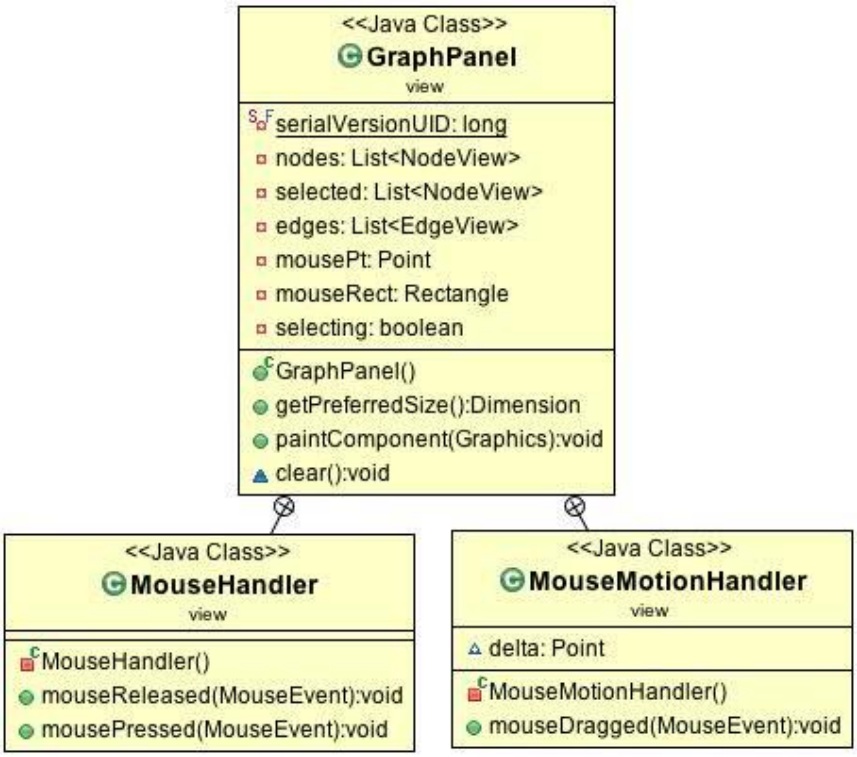
La classe **TaskGraphTools**, contenuta nel package util, è una classe di utilità fondamentale al raggiungimento del nostro obiettivo. Oltre ai metodi per i calcolo dei tempi T-Early/T-Late e l'individuazione del percorso critico, questa classe contiene metodi utili a verificare che il grafo delle attività sia connesso e aciclico. Per la verifica dell'assenza di cicli nel grafo si è fatto uso, ovviamente, dell'ordinamento topologico.



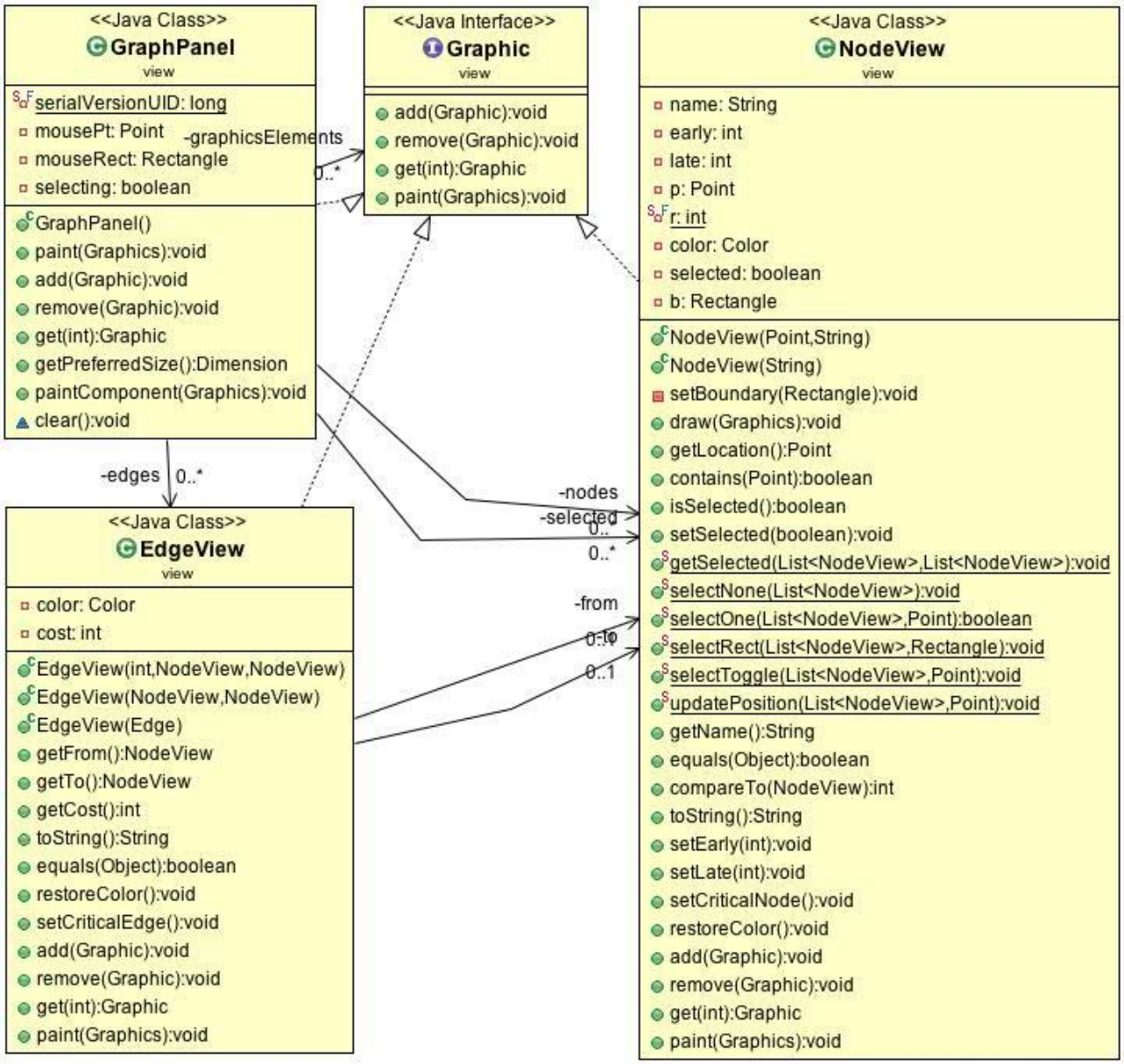
Per l'interfaccia grafica e i relativi comandi si è realizzato la classe **CriticalPathGUI** che al suo interno contiene, come inner-class, le classi **AddEdgeFrame** e **DeleteEdgeFrame**.



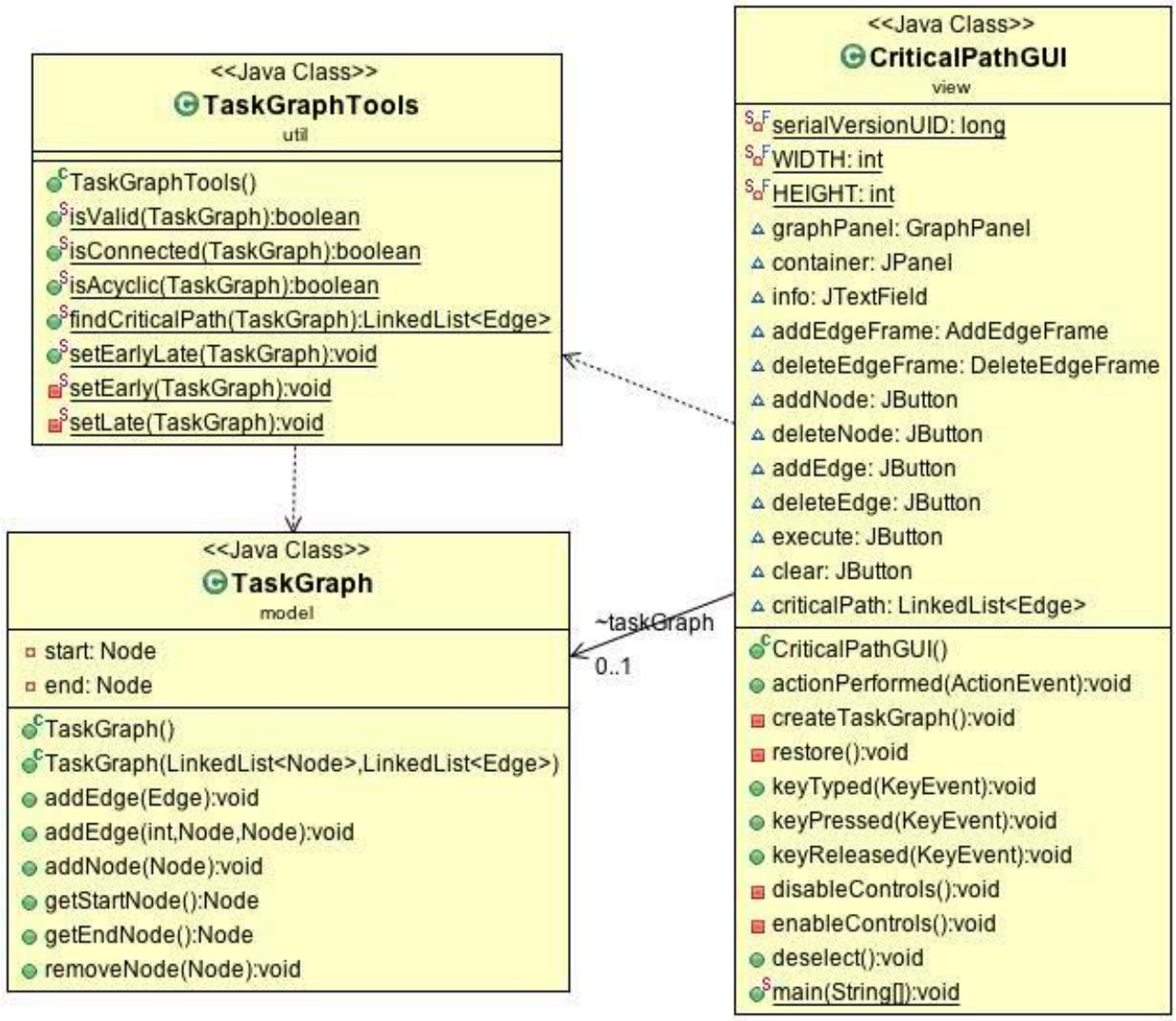
Per quanto riguarda la visualizzazione nel grafo si è scelto di realizzare, sempre come inner-class di CriticalPathGUI, la classe **GraphPanel**. Questa classe, che estende JComponent, ha lo scopo di essere facile ed intuitiva per il cliente che utilizzando il mouse è in grado di spostare e selezionare i nodi del grafo.



Le classi **NodeView** ed **EdgeView** servono a rappresentare graficamente rispettivamente i nodi e gli archi. Essendo elementi grafici, è venuto naturale utilizzare il **pattern Composite** per la loro rappresentazione. In particolare le classi GraphPanel, NodeView ed EdgeView implementano l'interfaccia **Graphic** la quale richiede di implementare il metodo astratto *paint(Graphics g)* cosicché richiamando questo comando dalla classe GraphPanel, essa a sua volta lo richiamerà ricorsivamente per tutti i suoi componenti.



Si è scelto di usare una variante del pattern **Model-View-Controller** per disaccoppiare la logica applicativa dalla sua rappresentazione grafica.



Infine si è fatto uso del framework di **JUnit** per il testing della classe TaskGraph. Si è dunque creata un'istanza delle classe e si sono testati metodi quali *isAcyclic(TaskGraph g)*, *isConnected(TaskGraph g)* e i metodi delegati al calcolo dei tempi T-Early/T-Late e all'individuazione del percorso critico, rispettivamente *setEarlyLate(TaskGraph g)* e *findCriticalPath(TaskGraph g)*.