Die Klasse Graph

(Josef Hübl, Triple-S GmbH, 24.7.03)

Die Klasse Graph erlaubt es einen attributierten Graphen zu instanzieren, dessen Attributstruktur zum Zeitpunkt der Instanzierung noch nicht bekannt sein muss.
Der dabei zugrundeliegende Grundgedanke ist der, dass Algorithmen, die auf Graphen arbeiten ihre eigen Knoten- oder Kantenattribute definieren können müssen ohne, dass die Klasse Graph geändert werden muss.

Die verschiedenen Ansätze diesem Ziel gerecht zu werden, sind zum Beispiel:

1. **Eine Basisklasse für attributierte Graphen**
   Letztendlich läuft dieses Konzept darauf hinaus, dass eine Templateklasse AttributedGraph definiert wird, die die Struktur der Attribute als Template-Parameter erhält. Dies funktioniert optimal, solange auf einem Graphen ein einziger Algorithmus ausgeführt werden soll, nicht jedoch wenn des Ergebnis eines Algorithmus für den nächsten wiederverwendet werden soll. Zum Beispiel kann ein Algorithmus eine topologische Sortierung der Knoten für einen Graphen erstellen, die der „Disjoint Path Algorithmus“ (DPA) als Voraussetzung benötigt. Mit der Templateklasse AttributedGraph läßt sich dies nur implementieren, wenn entweder der Algorithmus für die topologische Sortierung innerhalb des DPA nochmals implementiert wird oder dem DPA ein bereits topologisch sortierter Graph mit entsprechend vorhandenen Attributen übergeben werden kann. Arbeiten beide Algorithmen mit gänzlich unterschiedlichen Datenstrukturen, muss der Graph mit Aufwand O(n+m) kopiert bzw. umgewandelt werden, wobei m unter Umständen O(n*n) ist.

2. **Getrennte Verwaltung der Attribute**
   Werden die Attribute in Datenstrukturen verwaltet, die gänzlich losgelöst sind vom Graphen, so kann für jeden Algorithmus eine eigene, von der Graphstruktur unabhängige Attributstruktur definiert werden. Allerdings muss eine Synchronisierung stets dafür sorgen, dass bei Manipulationen am instanziierten Graphen die Zuordnungen Knoten-Attribut bzw. Kante-Attribut korrekt mitgeführt werden.
   Ein direktes Binden der Attribute an die Knoten bzw. Kanten über Pointer, erfordert aber eine gemeinsame (eventuell leere) Basisklasse für die Attributklassen unterschiedlicher Algorithmen, wobei für den Zugriff auf die eigentlichen Attributwerte ein häufiges Casten dieser Pointer notwendig sein wird.
   Alternativ können Knoten-, Kanten- und Attributarray’s gehalten werden, wobei der auf die Elemente über synchron gehaltene Indizes (Integer) erfolgt. Werden Knoten bzw. Kanten eingefügt oder gelöscht, so müssen auch die Attribute dafür eingefügt oder gelöscht werden. Die Implementierung einer solchen Verwaltung kann in einer Attribut-Basisklasse implementiert werden. Jedoch ist der Zugriff auf ein bestimmtes Attribut immer nicht nur von Typ des Attributs sondern auch von dessen Namen abhängig.

3. **Indizierte Attributenamen**
   Bei Verwendung von indizierten Attributennamen wird zunächst davon ausgegangen, dass

einem Graphen nur Attribute mit einem Bezeichner der Art Attributname1, Attributname2, usw. zugeordnet werden können, wobei aus Effizienzgründen nur die Nummer (1,2, usw.) verwaltet wird. Dadurch kann für die Klasse Graph eine Attributverwaltung implementiert werden, die den Namen eines Attributs nicht mehr kennen muss. Innerhalt der Ausführung eines Algorithmus auf dem Graphen, muss der Algorithmus lediglich auf eine konsequente Zuordnung von Attributnummer und Verwendung achten. Die so an einem Graphen von einem Algorithmus erstellten Attribute können von einem anderen Algorithmus wieder verwendet werden, ohne dass sich beide auf einen gemeinsamen Attributnamen abstimmen müssen. Nur über den zugrundeliegenden, meist elementaren Datentyp (z.B. Integer oder Bool) muss Einigkeit herrschen, welches in der Praxis jedoch keine wesentliche Einschränkung darstellen sollte.

Besonderheiten der vorliegenden Implementierung

Die vorliegende Implementierung verwaltet Attributnamen und –werte wie in (3.) skizziert. Der Datentyp eines Attributs bleibt der Klasse Graph jedoch stets verborgen. Aus diesem Grunde sind alle Aktivitäten, die die konkrete Verwaltung der Attributwerte betreffen (z.B. initialisieren, setzen oder auslesen) in die Attributklasse verlagert, die als Template-Klasse implementiert ist.

Die Instanziierung eines attributierten Graphen besteht somit aus der Instanz des eigentlichen Graphen, sowie einer Menge von Attributklassen Instanzen die jede für sich einen Attributvektor enthält. Einem Algorithmus, der auf einem attributierten Graphen ausgeführt werden soll, werden der Graph und die Attributklasseninstanzen als Parameter (Referenzen) übergeben. Da sie auch nach dem Beenden eines Algorithmus vorhanden sind, können sie als Input für einen nächsten Algorithmus wiederverwendet werden.

Da weder dem Graphen noch den Knoten und Kanten die Namen der Attribute bekannt sind, ist ein Zugriff auf die Attribute in der Form `node.MyAttribut = Wert` nicht möglich. Um trotzdem eine einfachen Zugriff auf die Attribute zu haben wurde der Index-Operator [] der Attribut-Templateklasse überladen, so dass der Zugriff über `MyAttribut[node]= Wert` möglich ist. Das heißt nicht das Attributvektorelement ist einem Knoten zugeordnet, sondern der Knoten dem Attributvektorelement. Jedes Attributvektorelement ist genau einem knoten zugeordnet. Umgekehrt können einem Knoten verschiedene Attributvektorelemente zugeordnet sein.

Innerhalb der Implementierung der Klasse Graph wurde auf eine speichereffiziente Verwaltung der Attributindizes geachtet, so dass bereits gebrauchte Indizes nach ihrem Verfall wiederverwendet werden.

Für die Knoten (node) und Kanten (edge) wurde der cast-Operator (bool) so überladen, dass er genau dann den Wert `true` liefert, wenn der Knoten bzw. die Kante einem Graphen zugeordnet ist.