Programmierkurs Python II – SS 2013

Übung 2

---

1 Gerichtete Graphen (4 Punkte)

Implementiere eine Datenstruktur für gerichtete Graphen. Du kannst hierfür die in der Vorlesung vorgestellte Klasse `Graph` erweitern. Die folgenden Methoden sollen unterstützt werden:

1. `add_node(self,x)`
   Einen Knoten (ohne Kantenanbindung) hinzufügen, wenn es ihn noch nicht gibt.

2. `add_edge(self,x, y)`
   Eine neue Kante erzeugen; ggf. x und / oder y als neuen Knoten hinzufügen.

3. `has_edge(self,x, y)`
   Existiert eine Kante von `x` nach `y`?

4. `indeg(self,x)` und `outdeg(self,x)` Eingangs- und Ausgangsgrad eines Knotens

5. `nodes(self)` und `edges(self)`
   Listen aller Knoten und Kanten. Kanten sind Paare von Knoten. Anstatt einer Liste kann auch ein Iterator zurückgegeben werden.

6. `dfs(self,x)` und `bfs(self,x)` Ein Iterator über alle vom Knoten `x` aus per Tiefen- bzw. Breitensuche aus erreichbaren Knoten. (Die in der Vorlesung notierte `dfs`-Methode muss zur Lösung der Aufgabe leicht angepasst werden.)

Alle Knoten werden durch ihre Labels repräsentiert, dh. `x` und `y` sind immer Strings (oder auch Integers), und gleiche Labels bedeuten gleiche Knoten.

2 Ungerichtete Graphen (2 Punkte)

Implementiere eine Methode `get_undirected_graph` für die `Graph`-Klasse. Diese Methode soll ein `Graph`-Objekt zurückgeben, in dem die Kanten-Richtungen „entfernt“ wurden. Denkbar wäre folgender Code in einem Test:

```
graph = Graph()
#...[Knoten und Kanten werden hinzugefügt]
undirected = graph.get_undirected_graph()
```

Zwischen allen Knoten, die in `graph` adjazent sind, soll auch Adjazenz in `undirected` bestehen, allerdings soll die ursprüngliche Richtung der Kanten nicht mehr erkennbar sein, wenn man z.B. `has_edge(x,y)` aufruft.

Hinweis: Leite Dir hierfür eine Klasse `UndirectedGraph` von `Graph` ab, die entweder `add_edge` und `__init__` geeignet überschreibt, oder `add_edge` und `has_edge`. (Wenn die Implementierung der anderen Methoden auf die übrige Datenstruktur zurückgreift, und nicht nur auf diese Methoden, muss sie evtl. auch überschrieben werden.)

3 Zyklustest (2 Punkte)

Implementiere eine Methode `has_cycle` für die `Graph`-Klasse. Diese Methode soll testen, ob ein Graph einen Zyklus enthält. Bei korrekter Implementierung liefert diese Methode für die Standard-Graphobjekte als auch die ungerichteten Versionen (siehe Aufgabe 2) das korrekte Ergebnis (also ein Test auf gerichtete und ungerichtete Zyklen gleichermaßen.)

4 Graphen Objektorientiert (Bonus, 6 Punkte)

4 Punkte: Implementiere eine objektorientierte Variante gerichteter Graphen. Definiere eine Klasse für Knoten, die eine Liste ihrer adjazenten Knoten verwaltet, und eine Klasse für Graphen, die die Knoten verwalten. Die Knoten-Klasse funktioniert analog zu der `Tree`-Klasse aus der vorherigen Vorlesung.

Implementiere für diese Klasse alle Methoden aus Aufgabe 1. Die Methoden-Parameter können immer noch Knoten-Label sein; intern soll aber mit komplexen Knoten-Objekten gearbeitet werden.

2 Punkte: Implementiere Aufgabe 2 und 3 für die objektorientierte Graph-Klasse.

Abgabe bis Donnerstag, 09.05.2013, 11:00 Uhr