C23 – Laborübung 5

Kontext

Ein Taxiunternehmen benötigt ein Programm zur Erfassung und Abrechnung von Fahrten. Das Grundprogramm dafür ist in Labor 1 bis 3 erstellt worden; jetzt soll es um eine Fahrtenplanung erweitert werden.

Fahrtenplanungsprobleme lassen sich gut mit Hilfe von Graphen modellieren und lösen. Als Vorbereitung auf die Modellierung von Graphen wurden in Labor 4 die Klassen „Node“ und „Edge“ erstellt.

Jetzt werden diese Klassen gemeinsam mit einer neuen Klasse „Graph“ in eine statische Bibliothek eingebracht und erweitert. Anschließend wird auf dieser Basis wird eine Fahrtenplanung als Ergänzung für das Taxiverwaltungsprogramm implementiert.

Aufgabenstellung

Aufgabe 1 – Statische Bibliothek

a) Erstellen Sie ein neues Projekt für eine statische Bibliothek „Graphlib“.
b) Überführen Sie die Quelltexte der Klassen Node, Edge und Graph (Anlage zur Aufgabenstellung, als .zip-Datei auf Moodle) in das „Graphlib“-Projekt, und erstellen Sie die statische Bibliothek graphlib.lib.

Hinweis: In den Quelltexten sind folgende Änderungen und Erweiterungen vorgenommen worden:

- *Die Klassen Node und Edge sind um einige Funktionen erweitert.*
- *Eine neue Klasse Graph wurde eingeführt, die die Node- und Edge-Objekte zentral verwaltet*
- *Node- und Graph-Objekte verwalten Referenzen auf andere Objekte unter Nutzung der sequentiellen Container der Standardbibliothek (list, deque).*

Hinweis: Der vorliegende Quelltext ist nicht vollständig, und muss von Ihnen gemäß der Aufgaben 2 bis 4 erweitert werden.

c) Nutzen Sie nun die statische Bibliothek Graphlib.lib im Programm „Taxi“.

Aufgabe 2 – Erweiterung der Klasse Edge

a) Erweitern Sie die Konstruktoren von Edge so, dass die neu erstellte Edge in die Liste der Eingangs- und Ausgangskanten der verbundenen Knoten eingefügt wird.
b) Erweitern Sie den Destruktor von Edge so, dass die zu löschende Edge aus der Liste der Eingangs- und Ausgangskanten der verbundenen Knoten entfernt wird.
c) Implementieren Sie die Funktion isConnectedTo() der Klasse Edge.

Aufgabe 3 – Erweiterung der Klasse Graph

a) Implementieren Sie die Funktion findNode(string id) so, dass ein Knoten mit dem Bezeichner id gesucht wird und dessen Adresse zurückgegeben wird. Falls der Knoten nicht gefunden wird, soll ein Nullpointer zurückgegeben werden.
b) Implementieren Sie die Funktion addNode(Node* pNewNode). In dieser Funktion soll geprüft werden, ob bereits ein Knoten mit gleichem Bezeichner im Graph vorhanden ist. Falls ja, soll eine Exception ausgelöst werden, anderenfalls soll der neue Knoten in den Graphen eingefügt werden.

c) Implementieren Sie die Funktion addEdge(Edge* pNewEdge). Sie soll die neue Edge in den Graphen einfügen. Testen Sie dabei, ob die mit der Edge verbundenen Knoten bereits im Graphen enthalten sind, und fügen Sie sie bei Bedarf hinzu (nutzen Sie dafür die Funktion addNode(…)).
d) Vervollständigen Sie den Destruktor von Graph so, dass alle mit dem Graphen assoziierten Nodes und Edges gelöscht werden (mit delete).
e) Implementieren Sie die Funktion remove(…) für Knoten und Kanten. Beim Entfernen von Knoten müssen diese aus der Knotenliste des Graphen entfernt werden. Die verbundenen Kanten müssen ebenso entfernt und gelöscht werden, und dann der Knoten selbst gelöscht werden. Beim Entfernen von Kanten müssen diese aus der Kantenliste des Graphen entfernt und gelöscht werden (mit delete).
f) Vervollständigen Sie die Funktion findEdges(…), die alle Kanten mit vorgegebenen Start- und Zielknoten als Vektor von Zeigern auf Kanten zurückgibt.

Aufgabe 4 – Dijkstra-Algorithmus (Kürzester Pfad)

a) Implementieren Sie den Algorithmus zum Finden des kürzesten Pfades in einem Graphen nach Dijkstra. Nutzen Sie den Pseudocode im Quelltext als Hilfestellung, oder geeignete Informationen im Web.

Aufgabe 5 – Graph

a) Definieren Sie in der Taxi-App eine Klasse Route als von Edge abgeleitete Klasse (lassen Sie die Graphlib-Bibliothek ab jetzt unverändert). Route soll die Entfernung zwischen zwei Knoten als Member-Variable speichern. Für die Klasse Route soll die Berechnung des Kantengewichts überladen werden; die Entfernungsinformation soll dabei Kantengewicht zurückgegeben werden.
b) Legen Sie in der Taxi-App einen Graphen mit folgenden Knoten (Orte) und Kanten (Routen) an:
    - Alexanderplatz -> Funkturm: 10,2km
    - Funkturm -> Strandbad Wannsee: 11,2km
    - Strandbad Wannsee -> Brandenburger Tor: 18,0km
    - Brandenburger Tor -> Alexanderplatz: 2,3km
    - Alexanderplatz -> Ostkreuz: 6,2km
    - Ostkreuz -> Regattastrecke Grünau: 14,6km
    - Regattastrecke Grünau -> Grenzallee: 12,8km
    - Grenzallee -> Alexanderplatz: 11,2km
    - Grenzallee -> Funkturm: 14,9km
    - Funkturm -> Zitadelle Spandau: 8,5km

Aufgabe 6 – Taxi-App mit Fahrtenplanung

a) Erweitern Sie die Taxi-App so, dass beim Buchen einer Fahrt der Start- und Zielort angegeben werden kann.
b) Erweitern Sie das Taxi-Programm so, dass mit Hilfe des Dijkstra-Algorithmus günstigste (=kürzeste) Route ermittelt wird. Dies Route soll als Liste der Orte und Entfernungen am Bildschirm ausgegeben werden. Wenn die Fahrt durchgeführt werden kann (also der Tankinhalt ausreicht), dann soll die entsprechende Fahrtbuchung durchgeführt werden, ansonsten soll ein Fehler ausgegeben werden.