Abgabe: **03.07.2016** (bis 23:59 Uhr)

Aufgabe 11.1 (P) Multiple-Choice (alte Klausur-Aufgabe)

Kreuzen Sie in den folgenden Teilaufgaben jeweils die richtigen Antworten an. Es können pro Teilaufgabe keine, einige oder alle Antworten richtig sein.

a) [1] Die zufällige Wahl einer Hashfunktion h aus einer 1-universellen Familie von Hashfunktionen auf $\{0, ..., m-1\}$ garantiert,

☐ dass Kollisionen ausgeschlossen sind.

☐ dass Kollisionen ausgeschlossen sind, solange höchstens die Hälfte der Plätze der Hashtabelle belegt sind.

☐ dass die Wahrscheinlichkeit für eine Kollision zweier beliebiger Elemente x und y mit $x \neq y$ höchstens $\frac{1}{m}$ ist.

b) [1] AVL-Bäume

☐ lösen das Problem der Entartung von Binärbäumen, indem sie beim Einfügen und Löschen von Elementen Balancierungen vornehmen.

☐ balancieren beim Einfügen und Löschen von Elementen derart, dass sich ein idealer unvollständiger Binärbaum ergibt (d.h., ein Binärbaum, der lediglich auf der untersten Ebene unvollständig ist).

☐ sind sehr effizient, da sie beim Einfügen von Elementen höchstens eine Rotation durchführen müssen.

☐ lassen sich, im Gegensatz zu (a,b)-Bäumen, speichereffizient in einem Feld verwalten und weisen daher eine gute Cache-Lokalität auf.

c) [2] Gegeben sei die folgende Rekursionsgleichung:

$$r(n) = \begin{cases} a & \text{falls } n = 1 \\ 2n + 3 * r(n-1) & \text{falls } n > 1 \end{cases}$$

Aus dem vereinfachten Master-Theorem aus der Vorlesung folgt, dass gilt:

☐ $r \in \Theta(n)$

☐ $r \in \Theta(n * log(n))$

☐ $r \in \Theta(n^{log_1(3)})$

☐ $r \in \Theta(n^{log_{\frac{n}{n-1}}(3)})$

☐ Das vereinfachte Master-Theorem lässt sich hier nicht anwenden.

d) [2] Gegeben sei die Funktion $f(n) = c\sqrt[d]{n}(\log_e n)n^g$, wobei c, d, e, g Konstanten mit Werten größer als 1 sind. Welche Konstante kann man durch eine andere Konstante mit einem größeren Wert ersetzen, sodass für die entstehende Funktion f' gilt: $f' \in \Theta(f)$?

☐ c ☐ d ☐ e ☐ g

e) [2] Ein Binärbaum ist *vollständig*, wenn alle seine Blätter auf der selben Tiefe liegen, also den selben Abstand zur Wurzel haben. Es gibt einen vollständigen Binärbaum mit

☐ 6 Blättern.

☐ 7 Knoten.

☐ 8 Blättern.

☐ 9 Knoten.

f) [1] Zur Berechnung kürzester Wege in einem Graphen verwendet man manchmal den Bellman-Ford-Algorithmus (kurz BFA) statt des Dijkstraalgorithmus (kurz DA), weil

☐ die Laufzeit des BFA auf baumförmigen Graphen um einen linearen Faktor besser ist als die des DA.

☐ der BFA auch bei negativen Kantengewichten anwendbar ist.

☐ der BFA bei nicht-zyklischen Graphen kürzere und damit effizientere Wege als der DA findet.

☐ der BFA im Gegensatz zum DA auch mit rationalen Kantengewichten eine logarithmische Laufzeit erreicht.

g) [1] Für eine Eingabemenge der Größe m soll eine Funktion f alle Palindrome der Länge n ausgeben. Beispiel: $f(\{a, d\}, 3) = \{aaa, ddd, ada, dad\}$. Welche Aussage beschreibt die Mindestlaufzeit am besten?

☐ $f \in \Omega(n * m)$ ☐ $f \in \Omega(n * m!)$ ☐ $f \in \Omega(n^2 * m!)$ ☐ $f \in \Omega(n * m^{\frac{n}{2}})$

h) [2] Welche Sortierverfahren können stabil implementiert werden?

☐ Insertionsort ☐ Heapsort ☐ Mergesort ☐ Bubblesort

Aufgabe 11.2 (P) Laufzeit-Analyse

Bestimmen Sie die asymptotische Laufzeit der Breitensuche, wenn

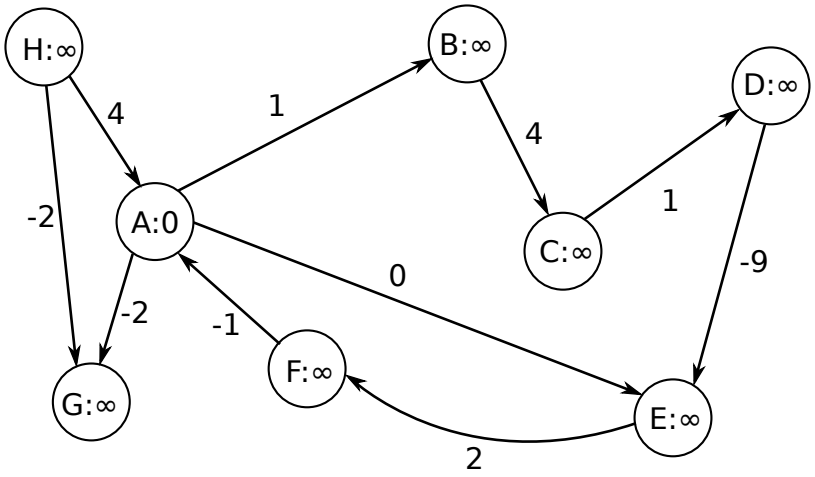
a) die Kanten als Adjazenzliste dargestellt werden.

b) die Kanten als Adjazenzmatrix dargestellt werden.

Die Abfrage nach Kanten innerhalb der Breitensuche soll dabei natürlich jedesmal über die gegebene Datenstruktur erfolgen. Der Graph besitze $k \geq 1$ Knoten und $e \geq 0$ Kanten.

Aufgabe 11.3 (P) Sumhtirogla-Drof-Namlleb

Führen Sie den Bellman-Ford-Algorithmus auf folgendem Graphen aus:



Der Knoten **A** soll der Startknoten sein. Zu Beginn ist die vorläufige Distanz zum Startknoten 0 und zu allen anderen Knoten unendlich (∞). Mit Hilfe des Bellman-Ford-Algorithmus sollen diese Werte nun schrittweise aktualisiert werden.

Aufgabe 11.4 (P) Binomial-Heaps

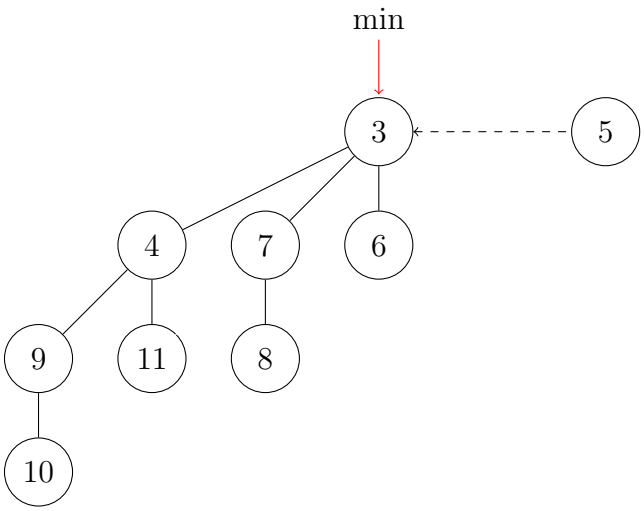
Führen Sie auf einem anfangs leeren Binomialheap folgende Operationen aus und stellen Sie die Zwischenergebnisse graphisch dar:

- build({15, 20, 9, 1, 11, 4, 13, 17, 42, 7}),
- deleteMin,
- deleteMin,
- deleteMin,
- deleteMin

Die build-Operation ist dabei durch iterative Ausführung von der insert-Operation auf den Elementen realisiert. Die insert-Operation wiederum besteht aus einer merge-Operation auf dem aktuellen, anfangs leeren Binomialheap und einem einelementigen Binomialheap, wobei letzterer das einzufügende Element enthält.

Aufgabe 11.5 [3 Punkte] **(H) Binomial-Heaps**

Führen Sie auf dem folgenden Binomial-Heap nacheinander drei deleteMin-Operationen aus. Zeichnen Sie nach jeder deleteMin-Operation den entstandenen Binomial-Heap.



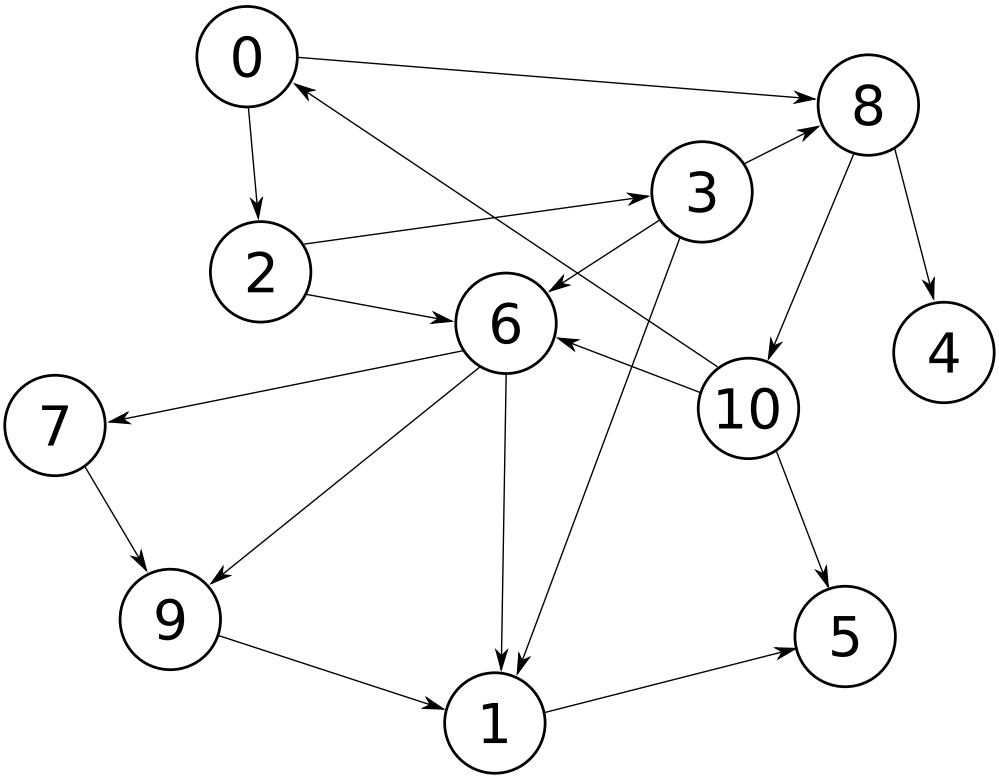
Aufgabe 11.6 [3 Punkte] **(H) Graphalgorithmen**

Gegeben sei der folgende Graph. Bearbeiten Sie mit diesem folgende Teilaufgaben:

a) [4] Nennen Sie die Reihenfolge, in der eine *Tiefensuche*, die bei Knoten 0 gestartet wird, die Knoten des Graphen besucht, wenn die Nachbarn eines Knotens in aufsteigender Reihenfolge entsprechend der natürlichen Zahlenordnung bearbeitet werden.

b) [4] Nennen Sie die Reihenfolge, in der eine *Breitensuche*, die bei Knoten 0 gestartet wird, die Knoten des Graphen besucht, wenn die Nachbarn eines Knotens in aufsteigender Reihenfolge entsprechend der natürlichen Zahlenordnung bearbeitet werden.

c) [4] Kann der Graph *topologisch sortiert* werden? Begründen Sie Ihre Antwort und nennen Sie, falls möglich, die topologische Sortierung der Knoten, die sich aus der Reihenfolge ergibt, in der die von Ihnen in Aufgabe a) durchgeführte Tiefensuche die Knoten fertig bearbeitet hat.



Nutzen Sie folgende Vorlage, um die verschiedenen Knotenreihenfolgen anzugeben.

a) Knotenfolge der Tiefensuche:

b) Knotenfolge der Breitensuche:

c) Knotenfolge der topologischen Sortierung (falls möglich):

Aufgabe 11.7 [4 Punkte] **(H) Topologische Sortierung mit DFS**

In dieser Aufgabe wollen wir ein alternatives Verfahren zur topologischen Sortierung auf einem DAG G betrachten. Dazu nehmen wir an, dass uns weder die üblichen Graphoperationen zum Traversieren einer Kante oder Abfragen der Nachbarschaft eines Knotens, noch irgendwelche „höheren Datenstrukuren“ wie z.B. Queues, Listen, PriorityQueues oder Stacks zur Verfügung stehen.

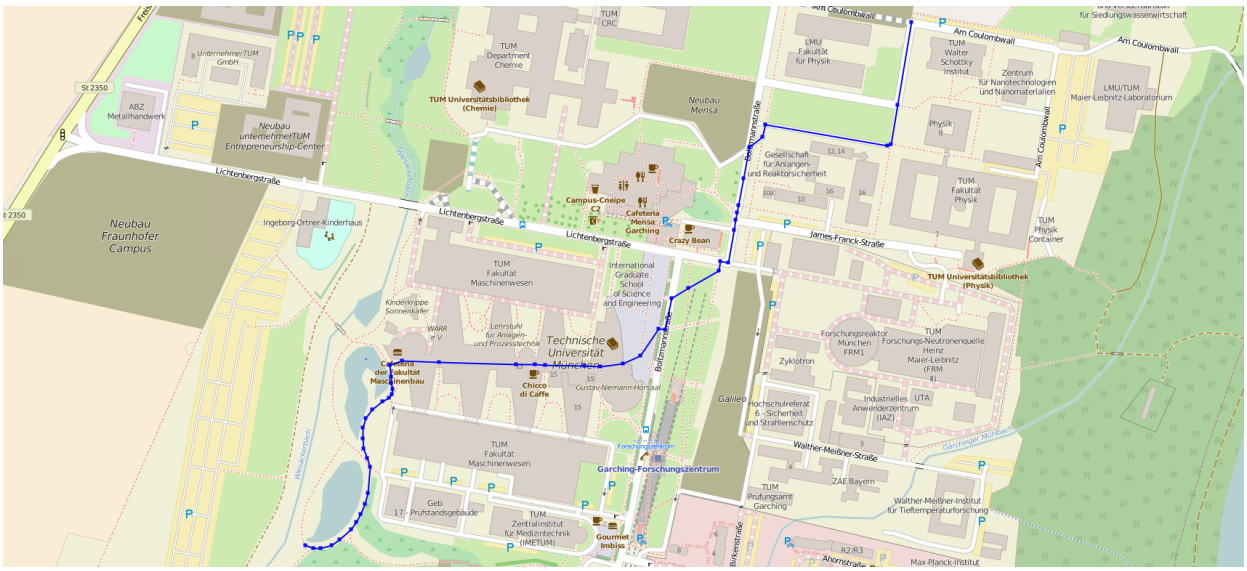
Die einzige Operation, die wir benutzen können ist eine Tiefensuche (DFS), die Arrays `dfs_num` und `finish_num` zurückgibt, die zu den jeweiligen Knoten die entsprechenden Werte beeinhalten. Außerdem stehen uns `for`-Schleifen und die übliche Addition bzw. Subtraktion zur Verfügung.

a) Geben Sie einen Algorithmus (Pseudocode reicht) an, der dieselbe asymptotische Laufzeit erreicht, wie der in der Vorlesung vorgestellte Algorithmus.

b) Begründen Sie die Laufzeit Ihres Algorithmus kurz.

c) Beweisen Sie die Korrektheit Ihres Algorithmus.

Aufgabe 11.8 [10 + 1 Punkte] **(H) Programmieraufgabe: Nogivan**



Ziel dieser Aufgabe ist es, den Algorithmus von Dijkstra in einem praktischen Anwendungsbeispiel einzusetzen. Wir werden dazu eine Route minimaler Länge in einer Karte von OpenStreetMap[1] berechnen. OpenStreetMap ist ein Projekt, welches daran arbeitet, mithilfe der Community eine quelloffene Karte für die gesamte Welt zu erstellen. In Deutschland sind die Kartendaten schon sehr präzise, vielerorts sogar besser als kommerzielle Angebote.

Die Karte wird in mehreren Formaten angeboten. Da wir kürzeste Wege berechnen wollen, können wir mit Bilddaten nichts anfangen; wir benötigen stattdessen den Graphen, dessen Kanten die Straßen repräsentieren. Mit diesem Blatt wird ein kleiner Ausschnitt der Karte vom Campus Garching verteilt. Da die OpenStreetMap-Daten sehr umfangreich sind, müssen Sie sich größere Ausschnitte der Karte selbst herunterladen. Dies können Sie tun, indem Sie auf `http://www.openstreetmap.org` auf *Export* klicken und einen Teil der Karte wählen. Nutzen Sie anschließend den Button *Overpass API*, der auch den Download größerer Ausschnitte erlaubt.

Die Daten befinden sich im OSM-Format, welches den XML-Standard verwendet; der genaue Aufbau wird unter `http://wiki.openstreetmap.org/wiki/OSM_XML` beschrieben. Nutzen Sie einen Texteditor, um sich mit dem Format vertraut zu machen. Wollen Sie sich eine Karte im OSM-Format ansehen bzw. sie verändern, können Sie das Programm JOSM[2] verwenden.

Gehen Sie bei Ihrer Implementierung vor wie folgt:

a) Implementieren Sie zunächst die Methode `parseFile(String fileName)` der Java-Klasse `MapParser`, die eine Instanz der Klasse `MapGraph` aus einer Datei im OSM-Format erstellt. Die Klasse `MapGraph` repräsentiert den Graphen der Straßen und Wege, auf dem Sie später minimale Wege suchen möchten. Sie können zum Einlesen der Datei einen XML-Parser verwenden – ein Tutorial dazu finden Sie unter `http://www.tutorialspoint.com/java_xml/`. Achten Sie unbedingt darauf, den *SAX Parser* zu verwenden (etwas weiter hinten im Tutorial beschrieben), da die OpenStreetMap-Daten extrem umfangreich sind (Oberbayern z.B. ist fast 3GB groß). Achten Sie beim Einlesen darauf, möglichst nicht benötigte Knoten (z.B. solche, die zu Gebäuden gehören) zu ignorieren, um Arbeitsspeicher zu sparen.

b) Implementieren Sie nun die Methode `int distance()` der Klasse `MapPoint`, die den Abstand zweier Punkte auf der Karte berechnet. Eine Beschreibung, die erklärt, wie Sie aus den Werten von Längengrad und Breitengrad den Abstand ermitteln, können Sie z.B. unter `http://www.movable-type.co.uk/scripts/latlong.html` finden.

c) Implementieren Sie die Methode `OSMNode closest (MapPoint p)` der Klasse `MapGraph`. Die Methode ermittelt zu einem gegebenen Kartenpunkt p denjenigen Knoten des Graphens, der dem Punkt am nähesten ist. Gibt es mehrere Knoten mit dem gleichen kleinsten Abstand zu p, so wird derjenige Knoten von ihnen zurückgegeben, der die kleinste Id hat. Implementieren Sie außerdem die Methode `boolean hasEdge(OSMNode from, OSMNode to)`, die zurückgibt, ob eine bestimmte Kante im Graphen vorhanden ist.

d) Implementieren Sie die Methode `Optional<RoutingResult> route (MapPoint from, MapPoint to)` der Klasse `MapGraph`. Nutzen Sie den Algorithmus von Dijkstra, um alle kleinsten Entfernungen von dem Knoten, der dem Kartenpunkt `from` am nähesten ist, zu berechnen. Die Methode gibt ein Objekt des Typs `RoutingResult` zurück, welches die Entfernung der zwischen Start- und Endknoten (entlang des Weges) und den entsprechenden Weg enthält. Testen Sie anschließend Ihre Implementierung mit **kleinen** Beispielen. **Hinweis**: Sie können für die Prioritätswarteschlange z.B. Ihre

[1] `https://www.openstreetmap.org`
[2] `https://josm.openstreetmap.de/`

Implementierung vom letzten Übungsblatt oder die Musterlösung verwenden, die ab Montag verfügbar sein wird.

e) Implementieren Sie die Methode `void write(String fileName, RoutingResult rr)` der Klasse `GPXWriter`, die eine Route im GPX-Format ausgibt. Eine Beschreibung des GPX-Formates lässt sich z.B. unter `https://de.wikipedia.org/wiki/GPS_Exchange_Format` finden. Nutzen Sie Ihre Implementierung anschließend, um eine kürzeste Route in eine Datei zu schreiben und zu betrachten. Sie können eine GPX-Datei z.B. mit dem Programm GpsPrune[3] öffnen.

f) Testen Sie Ihre Implementierung nun mit etwas größeren Beispielen. Sie werden feststellen, dass die Suche nach dem kürzesten Weg unangemessen langwierig ist – insbesondere für nah beieinander liegende Kartenpunkte[4]. Überlegen Sie sich, wie Sie den Algorithmus beschleunigen können. Nutzen Sie dabei folgende Denkanstöße:

- Überlegen Sie sich, wie Sie zusätzlich den Luftlinien-Abstand zweier Punkte sinnvoll nutzen können.
- Ein entscheidender Grund für die schlechte Performance des Algorithmus sind die Aufrufe der `decreaseKey()`-Operation. Versuchen Sie, die Anzahl der Aufrufe zu reduzieren!

Sie sollten nun in der Lage sein, kürzeste Wege auf der Karte von Oberbayern[5] hinreichend effizient zu berechnen.

g) **(Bonusaufgabe)** Wenn Sie Ihre Implementierung mit größeren Beispielen und längeren Routen (z.B. mit einer Route von Garching bis zum Starnberger See) testen, ist die Laufzeit weiterhin nicht befriedigend. Optimieren Sie die Laufzeit deswegen nun aggressiver, wobei Sie in Kauf nehmen, dass ihr Algorithmus nicht mehr mit Sicherheit den kürzesten Weg finden kann.

Bitte kommentieren Sie Ihren Code derart, dass Ihr Tutor Ihren Gedankengang nachvollziehen kann. Die Kommentierung des Codes fließt in die Bewertung mit ein!

[3] `http://activityworkshop.net/software/gpsprune/`

[4] Trennen Sie die Laufzeit für das Einlesen von der Laufzeit für die Pfadsuche. Die Laufzeit für das Einlesen ist in unserem Fall unrealistisch lang, da das OSM-Format nicht auf diesen Anwendungszweck optimiert ist.

[5] `http://download.geofabrik.de/europe/germany/bayern.html`