Project & Training 2: Aufgaben für den Block 1 Algorithmen und Datenstrukturen.

Abgabe und Zuteilung

Für diesen Aufgabenblock ist eine der sieben Aufgaben zu bearbeiten.

Die Zuteilung der Aufgaben wird mittels einem einfachen Algorithmus vorgenommen, und zwar müssen Sie den (positiven) Restklassewert ihrer Listennummer bei der Division durch sieben bestimmen. Dieser Restklassenwert liefert die zu lösende Aufgabe.

Beispiel: Finden Sie ihren Namen auf der Teilnehmerliste an der Stelle 17, so müssen Sie die Aufgabe 3 lösen, da 17 (mod 6) = 1.

Abgabe der Lösungen ist auf Gitlab (Aufgabe-A&D-Block1).

Bewertung:

Sie können maximal 15 Punkte erreichen. Für die erfolgreiche Bewältigung der Aufgabe müssen Sie mindestens 12 Punkte erhalten.

Die Aufgaben bestehen jeweils aus vier Teilaufgaben. Die Punkte verteilen sich wie folgt: erste Teilaufgabe 4 Punkte, zweite Teilaufgabe 6 Punkte, dritte Teilaufgabe 3 Punkte und vierte Teilaufgabe 2 Punkte.

Die erste Teilaufgabe besteht darin sich in ein bestimmtes Thema bzw. Algorithmenprinzip einzuarbeiten. Für diesen Teil ist nichts abzugeben. Die Punkte werden im Allgemeinen ohne Nachweis gutgeschrieben. Stichprobenartig können jedoch Tests durchgeführt werden, insbesondere bei Verdacht auf Plagiat, bei dem Sie das Beherrschen des erarbeiteten Unterrichtsstoffes nachweisen müssen.

In der zweiten Teilaufgabe müssen Sie einen zu entwerfenden Algorithmus als Pseudocode beschreiben.

Die dritte Teilaufgabe sieht die Anwendung des Verfahrens auf ein Beispielproblem vor.

In vierten Teil gilt es die Komplexität des Verfahrens abzuschätzen.

Bewertungskriterien: Vollständigkeit, Korrektheit, Verständlichkeit der Lösungen.

Referenzen:

Zu den oben angegebenen Probleme finden Sie in vielen Lehrbüchern zum Thema Algorithmen & Datenstrukturen ausführliche Information. Im Folgenden schlage ich hier zwei Lehrbücher vor. Sie können, müssen aber nicht diese Bücher heranziehen.

Algorithmen und Datenstrukturen: Eine Einführung mit Java, Gunter Saake, Kai-Uwe Sattler, 5. Auflage, dpunkt.verlag.

Thomas Ottmann und Peter Widmayer, Algorithmen und Datenstrukturen, Springer, 6. Auflag.

Aufgabe 1: Ähnlichkeit von Zeichenketten

Ähnlichkeitsvergleiche von Zeichenketten sind eine wichtige Technik in vielen Anwendungen, die eine fehlertolerante Suche erfordern. Typische Beispiele hierfür sind die Rechtschreibprüfung in Textverarbeitungsprogrammen oder auch der Vorschlag alternativer Suchbegriffe bei Web-Suchmaschinen.

1. Erarbeiten Sie das Algorithmenprinzip der Dynamischen Programmierung.
2. Geben Sie einen rekursiven Algorithmus zur Bestimmung der minimalen Editierdistanz auf der Basis der Dynamischen Programmierung an (in Pseudocode).
3. Wenden Sie ihr Verfahren an, um die minimale Editierdistanz für das Wortpaar Physiotherapie, Psychiater zu bestimmen.
4. Bestimmen Sie die Speicher und Zeit-Komplexität ihres Verfahrens an (mit Begründung).

Aufgabe 2: Dynamische Programmierung

Dynamische Programmierung ist ein grundlegendes Algorithmenprinzip, das - sofern es anwendbar ist – überraschend effiziente Lösungsverfahren auch für NP-schwierige Probleme liefern kann.

1. Erarbeiten Sie das Algorithmenprinzip der Dynamischen Programmierung.
2. Geben Sie einen rekursiven Algorithmus zum Lösen des Rucksackproblems an (in Pseudocode).
3. Wenden Sie ihr Verfahren auf das folgende Problem an. Gegeben ist ein Rucksack mit der Kapazität $C = 16$ und sechs Objekte mit den Nutzwerten $u_1 = 6, u_2 = 3, u_3 = 5, u_4 = 4, u_5 = 5, u_6 = 2$ und den Gewichten $w_1 = 2, w_2 = 3, w_3 = 7, w_4 = 4, w_5 = 4, w_6 = 2$.
4. Bestimmen Sie die Speicher- und Zeitkomplexität ihres Verfahrens in Abhängigkeit der Anzahl der Objekte n und der Kapazität C.

Aufgabe 3: Dynamische Programmierung

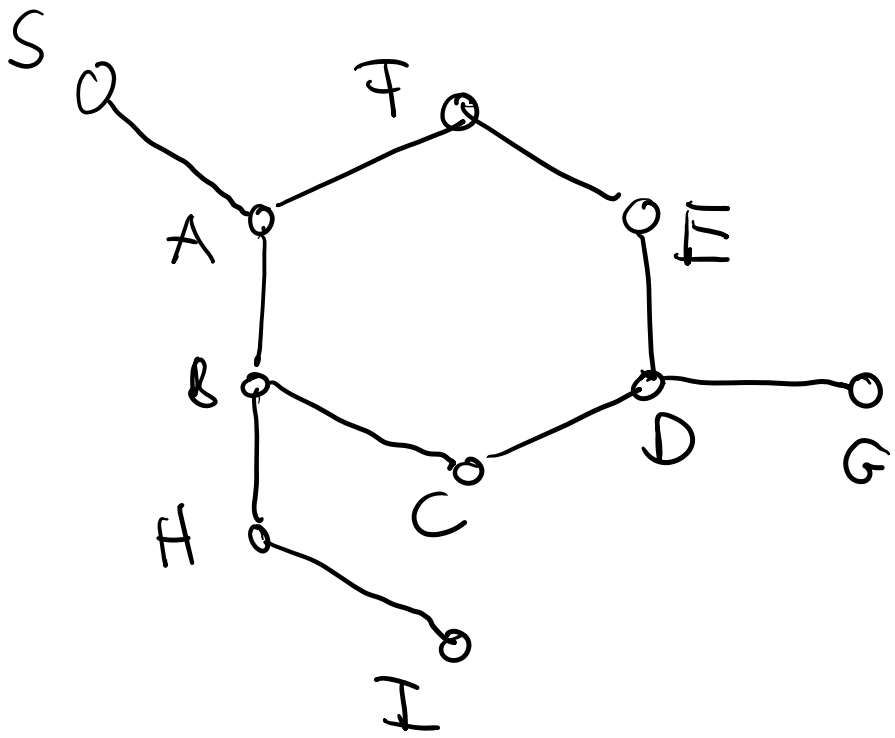
Dynamische Programmierung ist ein grundlegendes Algorithmenprinzip, das - sofern es anwendbar ist – überraschend effiziente Lösungsverfahren auch für NP-schwierige Probleme liefern kann.

1. Erarbeiten Sie das Algorithmenprinzip der Dynamischen Programmierung.
2. Geben Sie den Floyd-Warshall Algorithmus an, ein rekursiver Algorithmus zum Lösen des all-to-all shortest-path Problems(in Pseudocode).
3. Wenden Sie das Verfahren auf den Graphen der Aufgabe 7 an.
4. Bestimmen Sie die Speicher- und Zeitkomplexität dieses Verfahrens in Abhängigkeit der Anzahl der Knoten und der Anzahl der Kanten an.

Aufgabe 4: Sackgassen in Graphen erkennen

Graphen dienen in vielen Fällen dazu, Probleme abstrakt zu modellieren. Für die Problemlösungen kommen dann vielfach Graphsuchverfahren zum Einsatz.

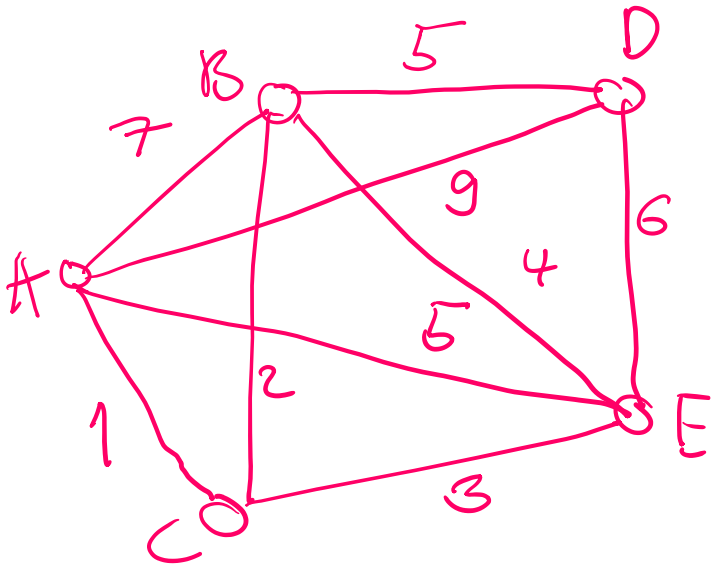
1. Erarbeiten Sie das Konzept des abstrakten Datentyps Graph und eignen Sie sich einige Grafsuchverfahren an, wie Tiefen- und Breitensuche und Dijkstra.
2. Geben Sie auf der Basis der Tiefensuche ein Verfahren an, das einen ungerichteten, zusammenhängenden Graphen durchläuft und dabei jeden Knoten dahingehend klassifiziert, ob dieser in einem Zyklus (keine Zweierzyklen) enthalten ist oder in einer Sackgasse liegt (in Pseudocode). Formulieren Sie dieses Verfahren rekursiv.
3. Wenden Sie dieses Verfahren auf den unten angegeben Graphen mit Startknoten S an.
4. Geben Sie die Speicher- und Zeit-Komplexität ihres Verfahrens an (in Abhängigkeit zur Anzahl der Knoten und Kanten).



Aufgabe 5: Minimal spannender Baum

(Hausinterne) Kommunikationsnetzwerke bilden auf natürliche Weise einen Graphen, deren Endgeräte die Knoten und die Direktverbindungen die Kanten darstellen. Die Kosten für den Aufbau eines solchen Netzwerks werden dem Kunden in Rechnung gestellt. Doch wie sollten diese berechnet werden? Die amerikanische Telefongesellschaft AT&T soll die Gebühren für solche hausinterne Netze von Firmenkunden nach der Länge eines minimal spannenden Baumes aller Direktleitungen – und nicht nach der Länge der tatsächlich verlegten Leitungen – berechnet haben.

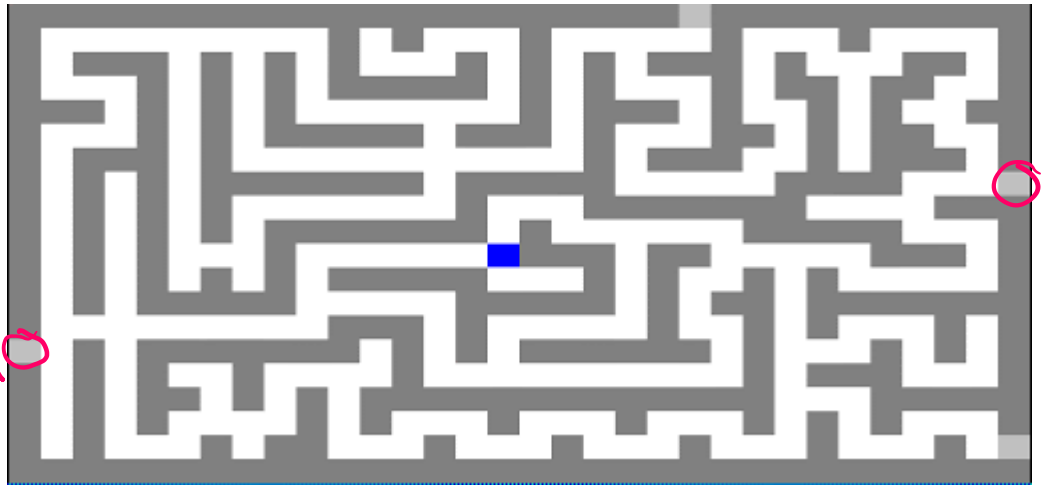
1. Erarbeiten Sie das Konzept des abstrakten Datentyps Graph und das Greedy-Prinzip für Algorithmen.
2. Zur Berechnung eines minimal spannenden Baumes in einem beliebigen ungerichteten, gewichteten, zusammenhängenden Graphen soll das Verfahren von Kruskal angewandt werden. Beschreiben Sie dieses Verfahren in Pseudocode. Eine Priority Queue dürfen Sie als gegeben voraussetzen und deren Primitive in der Algorithmenbeschreibung verwenden.
3. Wenden Sie das Verfahren von Kruskal auf den unten angegebenen Graphen an. Wählen Sie den Knoten A als Startknoten.
4. Geben Sie die Speicher- und Zeitkomplexität dieses Verfahrens (mit Begründung).

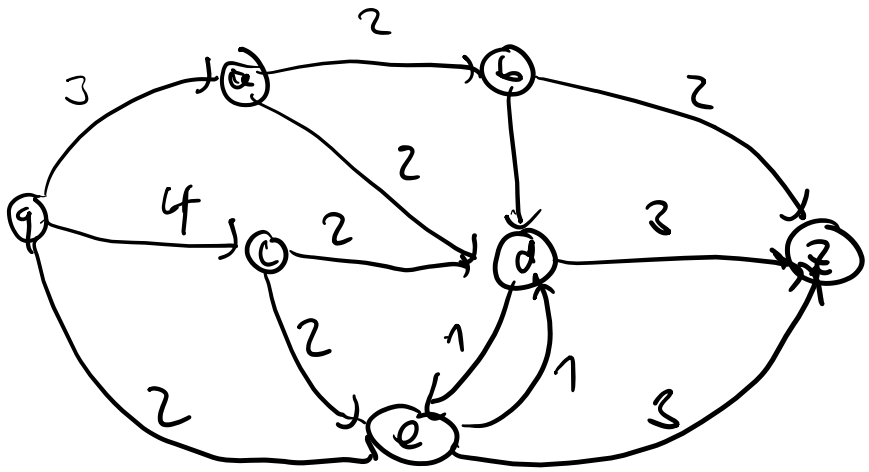


Aufgabe 6: Labyrinthsuche

Viele Probleme lassen sich durch einen Graphen modellieren und mittels eines Graphsuchverfahrens lösen. Viele Probleme lassen sich auf ein Kürzeste-Wege-Problem zurückführen. Ein solches Problem stellt sich, wenn in einem Labyrinth von einem Startpunkt, der Eintrittstelle, ein Weg zu einem Zielpunkt, der Austrittsstelle, zu finden ist.

1. Erarbeiten Sie das Konzept des Abstrakten Datentyps Graph und des Backtrackings.
2. Erläutern Sie, wie sich ein Labyrinth durch einen ungerichteten (gewichteten) Graphen modellieren lässt und dieser mittels der Tiefensuche durchlaufen werden kann. Formulieren Sie die Tiefensuche rekursiv als Pseudocode. Beachten Sie: Grundsätzlich sind in diesem Labyrinth Zyklen möglich. Überlegen Sie sich also eine geeignete Methode, um das Durchlaufen von Zyklen zu erkennen und um eine unendliche Schleife zu vermeiden.
3. Wenden Sie ihre Tiefensuche auf einen Graphen an, der das unten angegebene Labyrinth modelliert. Modellieren Sie das unten angegebene Labyrinth durch einen Graphen und wenden Sie darauf ihre Tiefensuche. Einstiegsposition: A, Zielposition: B.
4. Diskutieren Sie den möglichen Nutzen der Breitensuche bzw. des A*-Verfahrens. Beziehen Sie auch die Zeit- und Speicherkomplexität mit ein.




a
b
c
d
e
f
g
3
2
2
2
4
2
3
2
1
1
2
3