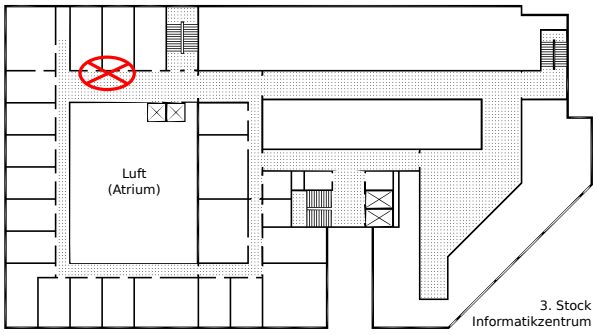
Abteilung Algorithmik **Winter 2014/15**
Institut für Betriebssysteme und Rechnerverbund
TU Braunschweig

Prof. Dr. Sándor P. Fekete
Dr. Christian Scheffer

Algorithmen und Datenstrukturen
Übung 2 vom 28. 11. 2014

Abgabe der Lösungen bis zum Donnerstag, den 11. 12. 2014 um 14:30 im Hausaufgabenrückgabeschrank.

Bitte die Blätter zusammenheften und vorne deutlich mit eigenem Namen, Matrikel- und Gruppennummer, sowie Studiengang versehen!



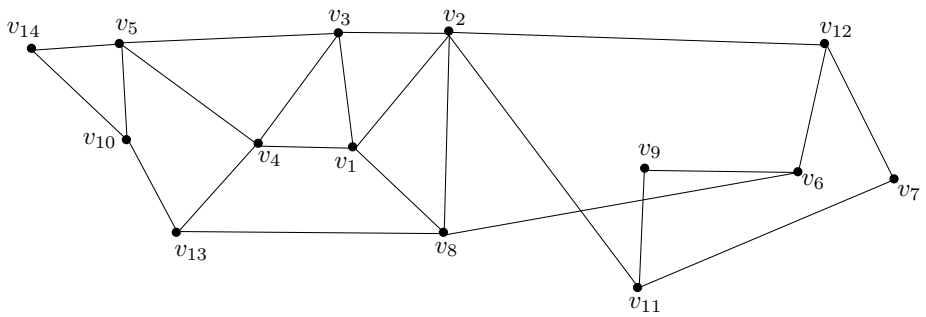


Abbildung 1: Der Graph G

Aufgabe 1 (Breiten- und Tiefensuche):

a) Wende Breitensuche mit Startknoten v_1 auf den Graphen G aus Abbildung 1 an.

b) Wende Tiefensuche mit Startknoten v_1 auf den Graphen G aus Abbildung 1 an.

c) Gib die Adjazenzliste für G an.

Kommt bei a) oder b) zu einem Zeitpunkt mehr als ein Knoten für den nächsten Schritt in Frage, wähle denjenigen mit dem kleinsten Index. Gib nach jeder Veränderung den Stack bzw. die Queue an und zeichne am Ende den gefundenen Baum. **(8+8+4 Punkte)**

Aufgabe 2 (Suche in Labyrinthen): Um einen Ausweg aus einem Labyrinth wie in Abbildung 2 zu finden, kann eine Tiefensuche verwendet werden.

Dabei modelliert man Kreuzungen und Sackgassen in dem Labyrinth als Knoten und Verbindungen (Wege) zwischen den Kreuzungen als Kanten. Start- und Zielpunkt sollen jeweils auch als Knoten modelliert werden.

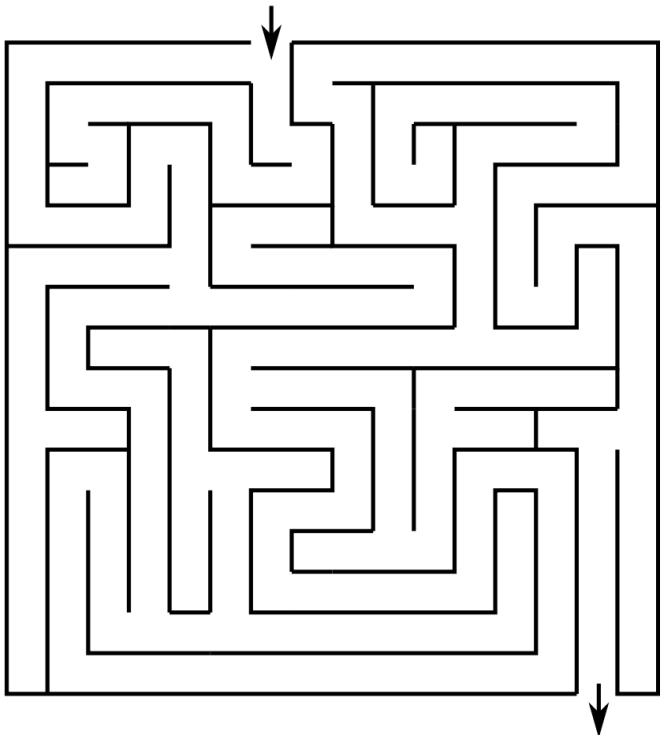


Abbildung 2: Ein Labyrinth.

Schwierig für einen Algorithmus ist, dass ihm nicht das ganze Labyrinth bekannt ist, sondern immer nur eine lokale Sicht. Das heißt, der Algorithmus sieht zu jedem Knoten die anliegenden Kanten, weiß aber nichts über die Knoten, zu denen sie führen. Das Durchlaufen einer Kante kostet in unserem Modell jeweils einen Schritt.

a) Gib den Graphen zum Labyrinth in Abbildung 2 an, kennzeichne den Startknoten s und den Zielknoten t. Markiere die von Dir verwendeten Knoten in Abbildung 2. (Bemerkung: Achte darauf, dass Deine Modellierung konsistent darin ist, was ein Knoten ist und was nicht.)

b) Wir benutzen Tiefensuche, um einen Weg vom Start- zum Zielknoten in einem (Labyrinth-)Graphen zu finden.

   Dabei bricht der Algorithmus ab, sobald er den gesuchten Zielknoten betritt (also ihn zu R hinzufügt, Zeile 2.3). Eine Kante wird durchlaufen, wenn ihr Endknoten w von Algorithmus 3.7 zu R hinzugefügt wird (Zeile 2.3), sowie wenn w wieder verlassen wird, weil es dort keine unerkundeten, ausgehenden Kanten mehr gibt (Zeile 2.2). Genau die durchlaufenen Kanten werden in T gespeichert.

   Zeige, dass bei einer Tiefensuche so keine Kante mehr als zweimal durchlaufen wird (egal, wie das Labyrinth aussieht).

c) Benutze die Aussage aus b), um zu zeigen, dass in einem zusammenhängenden Graphen mit insgesamt n Knoten (inklusive Start- und Zielknoten) das Ziel in höchstens $2n - 3$ Schritten gefunden wird.

Damit hat man gezeigt, dass man mit dem Tiefensuche-Algorithmus in einem Labyrinth *höchstens* (d. h. im worst-case) $2n - 3$ Schritte braucht um den Ausgang zu finden.

Um zu beweisen, dass die Tiefensuche tatsächlich eine bestmögliche Strategie ist, wollen wir jetzt zeigen, dass es für jeden beliebigen Algorithmus einen Graphen gibt, bei dem der Algorithmus $2n - 3$ Schritte benötigt.

(Hinweis: Der Graph wird dabei „maßgeschneidert" für einen unbekannten Algorithmus. Das heißt: Es kann angenommen werden, dass man bei jedem Knoten weiß, welche Kanten von dort aus in welcher Reihenfolge durchlaufen werden, und den Graphen entsprechend so konstruieren, dass er eine lange Laufzeit erzwingt. Es darf hingegen *nicht* davon ausgegangen werden, dass es sich um einen bestimmten Algorithmus, wie z. B. Tiefensuche handelt!)

d) Gib einen Graphen mit $n = 6$ Knoten an, so dass es für jeden Suchalgorithmus einen Knoten gibt, der erst nach $2n - 3 = 9$ Schritten erreicht wird.

e) Gib eine Konstruktionsvorschrift an, die für beliebige $n \geq 2$ einen Graphen mit n Knoten (inklusive Start- und Zielknoten) erzeugt, so dass ein beliebiger Algorithmus mindestens $2n - 3$ Schritte braucht, um den Zielknoten zu finden.

(Hinweis: Konstruiere zunächst wie in Aufgabenteil d) einen Graphen, bei dem es für jeden Algorithmus einen Knoten gibt, der erst nach $2n-3$ Schritten besucht wird, definiere dann den Zielknoten in Abhängigkeit vom Suchalgorithmus.)

(8+3+3+3+3 Punkte)

Aufgabe 3 (Das Orakel von Kevin Bacon): Dem *Orakel von Kevin Bacon* liegt der Schauspielergraph S zugrunde: Schauspieler sind durch Knoten repräsentiert. Zwei Schauspielerknoten sind durch eine Kante verbunden, wenn sie gemeinsam in einem Film gespielt haben. Der Knoten von Kevin Bacon hat die *Kevin-Bacon-Zahl* (KBZ) 0; die KBZ eines anderen Schauspielers ist die Länge eines kürzesten Weges im Schauspielergraphen S zu Kevin Bacon. Beispielsweise hat Tom Hanks die KBZ 1, da er mit Kevin Bacon in *Apollo 13* gespielt hat. Falls kein verbindender Weg existiert, ist die KBZ des betrachteten Schauspielers als unendlich definiert.

Das Orakel ist im Web verfügbar: `http://oracleofbacon.org/`. Die zugrundeliegenden Filmdaten sind der *Internet Movie Database* entnommen: `http://www.imdb.com/`.

a) Wir betrachten einen Pfad für die KBZ z eines Schauspielers A als einen Pfad in S, der Kevin Bacon und A verbindet und der aus $z + 1$ Knoten besteht. Gib einen Schauspieler mit mindestens KBZ 4 und einem entsprechenden Pfad an.

b) Bei der Suche nach einer möglichst hohen KBZ hat man als Nutzer keinen direkten Zugriff auf die gesamten Daten der IMDb, sondern kann sich nur mit Links durch IMDb-Webseiten klicken. Zusätzlich kann man ein weiteres Fenster einsetzen, in dem man per Kevin-Bacon-Orakel für konkrete Schauspieler die KBZ abfragt. Beschreibe eine systematische Strategie, die Schauspieler mit größtmöglicher (endlicher) KBZ findet. Wie verhält sie sich zu Tiefensuche? Welche Rolle spielt die Breitensuche?

c) Wenn man einen Server wie das Kevin-Bacon-Orakel betreibt, muss man damit rechnen, dass in kurzer Zeit sehr viele Anfragen hereinkommen, die jeweils schnell beantwortet werden müssen. Deshalb lohnt es sich, zwischen Verfahren zu unterscheiden, die eine Aufgabenstellung nur einmal lösen, und solchen, die (nach einem

gewissen Aufwand für "Preprocessing" zur Erstellung einer geeigneten Datenstruktur) dieselbe Frage immer wieder neu für unterschiedliche Anfragen ("Queries") beantworten können. Bei einer Anfrage soll nun nicht nur die KBZ des betrachteten Schauspielers sondern auch ein entsprechender Pfad ausgegeben werden, siehe Aufgabenteil a). Wie kann man es als Betreiber eines Orakels vermeiden, dass man für jede Anfrage eine neue Breitensuche ausführen muss, ohne dass man gigantische Datenmengen vorhalten muss?

(4+10+6 Punkte)