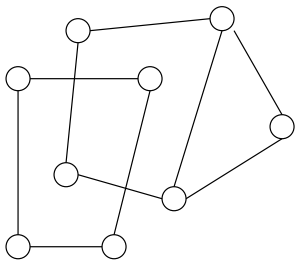
1 Inhalt und Motivation des Praktikums

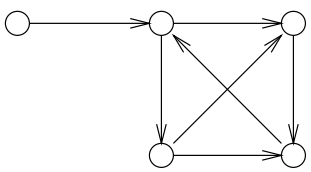
Im Rahmen des Praktikums Algorithmen-Entwurf werden wir eine Reihe von effizienten Algorithmen zur Lösung von Problemen in Graphen in C++ implementieren und ihre Arbeitsweise durch Animation visualisieren. Es werden sowohl grundlegende Algorithmen wie z.B. der Kürzeste-Wege-Algorithmus von Dijkstra als auch speziellere Algorithmen für kompliziertere Problemstellungen behandelt. Bei den Implementierungen werden wir einerseits auf Effizienz achten, d.h. auf Einhaltung der besten erzielbaren asymptotischen Worst-Case-Laufzeit, und andererseits auf eine anschauliche Visualisierung, die die Arbeitsweise der Algorithmen gut nachvollziehen läßt. Die Teilnehmer(innen) sollen durch das Praktikum unter anderem die Vorzüge des Einsatzes höherer Datenstrukturen zur Effizienzsteigerung kennenlernen und ein besseres Verständnis für die Arbeitsweise typischer Graphenalgorithmen entwickeln. Nebenbei wird sich sicher auch Routine bei der Programmierung in C++ einstellen. Aspekte des Software-Engineering und des guten Designs objekt-orientierter Software werden wir im Praktikum dagegen kaum berücksichtigen.

2 Graphen

Ein Graph $G = (V, E)$ besteht aus einer Knotenmenge V und einer Kantenmenge E. Jede Kante $e \in E$ verbindet zwei Knoten aus V miteinander. Dabei unterscheidet man gerichtete und ungerichtete Graphen. Bei gerichteten Graphen ist jede Kante als geordnetes Paar (u, v) von Knoten gegeben; u ist der Quellknoten der Kante, v der Zielknoten. Gerichtete Kanten werden üblicherweise mit einem Pfeil gezeichnet. Bei ungerichteten Graphen haben die Kanten keine Richtung und lassen sich daher als zweielementige Teilmengen $\{u, v\}$ von V repräsentieren. Ungerichtete Graphen lassen sich meist auch als gerichtete Graphen mit je zwei antiparallelen Kanten zwischen benachbarten Knoten auffassen.



ungerichteter Graph



gerichteter Graph

Die Zahl der Knoten wird mit $|V| = n$ bezeichnet, die der Kanten mit $|E| = m$. Wir werden die Laufzeit von Algorithmen in Abhängigkeit von n und m ausdrücken. Algorithmen mit Worst-Case-Laufzeit $O(n + m)$ nennen wir *linear*.

In einem ungerichteten Graphen heißen die maximalen Teilmengen der Knotenmenge, innerhalb derer jeder Knoten von jedem anderen über einen Pfad erreichbar ist, seine Zusammenhangskomponenten. Der linke Graph in obigem Beispiel hat z.B. zwei Zusammenhangskomponenten.

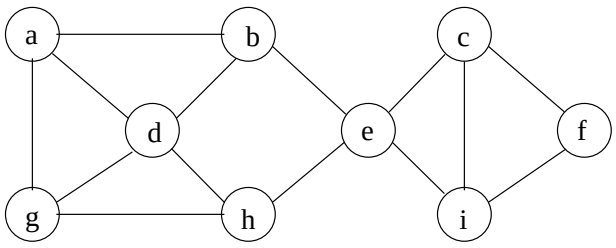
Die Datenstruktur, die üblicherweise zur Speicherung eines Graphen in einem Programm verwendet wird, besteht grob gesagt aus einer Liste aller Knoten, einer Liste aller Kanten, und zu jedem Knoten einer Liste der zu dem Knoten benachbarten (inzidenten) Kanten. Bei gerichteten Graphen werden bei den Knoten getrennte Listen für eingehende und ausgehende Kanten gespeichert. Weiterhin sind die Knoten und Kanten i.d.R. durchnumeriert und können über ihre Nummer direkt zugegriffen werden. Eine alternative Darstellung verwendet eine Adjazenz-Matrix, wobei für einen Graphen mit n Knoten aber immer Speicherplatz $\Omega(n^2)$ benötigt wird.

Eine Vielzahl von in der Praxis auftretenden Problemen lassen sich sehr natürlich durch Graphen modellieren und auf Graphenprobleme zurückführen, etwa Wegeprobleme, Zuordnungsprobleme und Spiele.

3 Tiefen- und Breitensuche

Eine grundlegende Aufgabenstellung ist das Durchsuchen eines gegebenen Graphen. Die häufigsten Varianten sind die Tiefensuche (DFS, depth-first-search) und die Breitensuche (BFS, breadth-first-search). Beide Varianten beginnen jeweils bei einem beliebigen unbesuchten Knoten des Graphen und besuchen dann alle von diesem Knoten über Kanten erreichbaren weiteren Knoten des Graphen (im gerichteten Fall werden Kanten nur in Vorwärts-Richtung durchlaufen). Falls dann noch unbesuchte Knoten existieren, wird die Tiefen- oder Breitensuche an einem beliebigen unbesuchten Knoten fortgesetzt.

Bei der Tiefensuche wird der Graph so durchlaufen, daß beim Auffinden eines unbesuchten Nachbarknotens w des aktuellen Knotens v die Suche sofort bei diesem Nachbarknoten fortgesetzt wird. Die Suche kehrt erst zu v zurück, wenn der über w erreichbare Teilgraph komplett abgearbeitet ist. Die Breitensuche dagegen besucht zuerst alle Knoten, die vom Startknoten über eine einzige Kante erreichbar sind, dann diejenigen, die über zwei Kanten erreichbar sind, usw. Numeriert man die Knoten in der Reihenfolge, in der sie bei einer DFS bzw. BFS besucht werden, so erhält man eine sog. DFS- bzw. BFS-Numerierung, die i.a. aber nicht eindeutig bestimmt ist.



Wenn man in diesem Beispiel den Knoten e als Startknoten wählt, so könnte eine DFS die Knoten in der Reihenfolge e,h,g,a,d,b,i,f,c besuchen, eine BFS in der Reihenfolge e,i,c,b,h,f,d,a,g.

Zur Implementierung von DFS wird i.d.R. eine rekursive Funktion verwendet, zur Implementierung von BFS eine Queue (Warteschlange, FIFO).