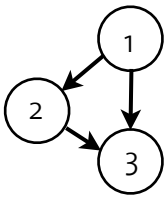
Graphentheorie

Beschreibung eines Graphen



Graph G = (V, E)

V = Menge der *Knoten*, n = |V|

E = Menge der *Kanten*, m = |E|

Wir unterscheiden *gerichtete* und *ungerichtete* Graphen und *ungewichtete* und *gewichtete* Graphen. Um einen Graphen abzuspeichern kennen wir zwei Varianten:

Adjazenz-Matrix

Die Zelle in der Zeile i und Spalte j beschreibt, ob es eine Kante von i nach j gibt. Für unser Beispiel also:

i \ j	1	2	3
1	0	1	1
2	0	0	1
3	0	0	0

Adjazenz-Liste:

Oder wir speichern pro Knoten eine separate Liste mit allen Knoten die von dort aus erreichbar sind:

Liste 1: Knoten 2, Knoten 3

Liste 2: Knoten 3

Liste 3: (leer)

Die leichter zu implementierende Adjazenz-Matrix verwenden wir, falls wir wissen, dass unser Graph sehr viele Kanten hat. In den meisten anderen Fällen nehmen wir die Adjazenz-Liste, da sich viele Operationen damit effizienter realisieren lassen.

Als *Grad (*engl. *degree)* eine Knotens bezeichnen wir die Anzahl ausgehender Kanten.

Graphenalgorithmen

Bei vielen Fragestellungen zu Graphen geht es darum das Netzwerk ausgehend von einem gegebenen Startknoten systematisch anzuschauen. Dafür gibt es 2 typische Algorithmen:

Breitensuche (BFS)

Wir verwenden eine *Queue Q* um uns zu merken welche Knoten wir noch besuchen wollen. Zu Beginn ist nur der Startknoten in Q. Solange *Q* noch Knoten enthält nehmen wir den ersten heraus, markieren diesen Knoten als besucht und fügen alle seine noch nicht besuchten Nachbarn in *Q* ein. Wenn wir uns für jeden Knoten die Anzahl Schritte zum Startpunkt merken, können wir so auch gleich die kürzesten Wege in einem ungewichteten Graphen berechnen.

Tiefensuche (DFS)

Im Gegensatz zu vorhin geht es nun nicht mehr aufsteigend dem Abstand zum Startpunkt nach. Wir beginnen mit einem beliebigen Nachbarn des Startpunktes. Dann fahren wir mit einem beliebigen unbesuchten Nachbarn des Nachbarn weiter und so fort. Haben wir bei einem Knoten alle Nachbarn schon besucht, gehen wir unseren bisherigen Weg zurück und von dort in noch unbesuchte Richtungen weiter.

Beide Traversierverfahren (BFS und DFS) haben eine Laufzeit von O(n+m). Mittels Traversierung lässt sich beispielsweise die Anzahl Komponenten eines Graphen ermitteln.

Topologische Sortierung

Gegeben ist eine Menge von Objekten und Abhängigkeiten der Form „nimm x vor y“. Dazu wollen wir eine Reihenfolge (Sortierung) finden, die keine dieser Abhängigkeiten missachtet. Wir beginnen damit die Anzahl eingehender Kanten (*Indegree*) jedes Knotens zu bestimmen. Falls zu einem Zeitpunkt kein Knoten mit *Indegree* 0 im Graphen vorkommet, sind die Bedingungen nicht erfüllbar, und es gibt eine zyklische Abhängigkeit. Sonst wählen wir nun einen Knoten mit *Indegree* 0 und entfernen ihn aus dem Graphen und fügen ihn in die sortierte Folge ein. Diesen Schritt wiederholen wir nun bis alle Knoten entfernt wurden oder wir auf eine Zyklus stossen.

Weitere Themen

- Kürzeste Wege: Dijkstra und Floyd-Warshall

- Spannbäume: Prim und Kruskal

- Zweifärbbarkeit, Eulertouren, Hamiltontouren

Literaturtipps

[1] Handouts von Sebastian Millius, besonders Handout 11 zu Graphen: http://goo.gl/5DCib

[2] Programming Challenges von Skiena, Revilla