Programmiertechnik II

Graph-Algorithmen

Anwendungsgebiete

- "Verbundene Dinge" oft Teilproblem/Abstraktion einer Aufgabenstellung
  - Karten: Wie ist der kürzeste Weg von Sanssouci nach Kunnersdorf?
  - Hypertext: Welche Seiten sind mit welchen anderen verknüpft
  - Schaltkreise: Gibt es einen Kurzschluss? Kann man eine gegebene Schaltung kreuzungsfrei auf eine Leiterplatte bringen?
  - Programmstruktur: Welche Abläufe sind in einem Programm möglich? Wie wirkt sich eine Variablenzuweisung auf eine bestimmte Stelle im Programm aus?
  - Kommunikationsnetze: Welche Route ist die geeignetste für ein gegebenes Datenpaket? Welche Verbindungen sollten zusätzlich installiert werden?
- Literatur: Sedgewick, Algorithms in Java, Part 5 (Graph Algorithms)

Graphen

- Ein Graph ist eine Menge von Knoten und eine Menge von Kanten, die jeweils verschiedene Knoten miteinander verbinden (mit maximal einer Kante zwischen zwei Knoten)
  - Knoten (engl. *vertex*, pl. *vertices*, oft *node*) nummeriert mit 0..V-1
  - Zahl der Kanten (engl. *edge*, oft *arc, link*) E
  - Menge von Knoten oft implizit durch Angabe aller Kanten gegeben
- Verallgemeinerungen:
  - erlaube parallele Kanten (*multigraph*)
  - erlaube Kanten von Knoten zu sich selbst (*self-loops*)
- Ein Graph mit V Knoten hat höchstens V(V-1)/2 Kanten
- Benachbarte Knoten heißen *adjazent*
- *Grad* eines Knotens: Zahl der Kanten, die zu dem Knoten führen

Graphen (2)

- Teilgraph: Teilmenge der Knoten und Kanten, die wieder einen Graph bilden
  - *induzierter Teilgraph (induced subgraph)*: Gegeben eine Teilmenge der Knoten, füge alle "passenden" Kanten hinzu
- Planarer Graph: Darstellung in der Ebene ist kreuzungsfrei möglich
- Ein *Pfad* ist eine Folge von Knoten derart, dass jeder Knoten adjazent zu seinem Vorgänger ist. Ein *einfacher Pfad* ist ein Pfad, auf dem kein Knoten doppelt vorkommt. Ein *Zyklus* ist ein Pfad, der einfach ist mit Ausnahme des letzen Knotens, der gleich dem ersten ist.
- Ein Graph heißt *azyklisch*, falls es in ihm keinen Zyklus gibt.

Graphen (3)

- Ein Graph ist *zusammenhängend* (*connected*) falls es einen Pfad von jedem Knoten zu jedem anderen gibt.
- Ein nicht-zusammenhängender Graph besteht aus *Zusammenhangskomponenten* (*connected components*), welches die maximal zusammenhängenden Teilgraphen sind.
- Ein azyklischer zusammenhängender Graph heißt *Baum* (*tree*). Eine Menge von Bäumen heißt Wald (*forest*). Ein *aufspannender Baum* (*spanning tree*) ist ein Teilgraph, der ein Baum ist und alle Knoten des Graphs enthält. Ein *aufspannender Wald* (*spanning forest*) ist ein Teilgraph, der ein Wald ist und alle Knoten enthält.

Graphen (4)

- Ein *vollständiger* Graph ist einer, bei der jeder Knoten mit jeder anderen verbunden ist. Das *Komplement* eines Graphen G ist der Graph, den man aus dem vollständigen Graphen mit den Knoten von G erhält, wenn man die Kanten von G entfernt.
- Die *Dichte (density)* eines Graphen ist die Menge der durchschnittliche Knotengrad, 2E/V. Ein *dichter (dense)* Graph ist einer dessen Dichte proportional zu V ist; ein *lichter (sparse)* Graph ist einer, dessen Komplement dicht ist.

Graphen (5)

- Ein Graph heißt *bipartit*, wenn sich die Knoten so in zwei Teilmengen teilen lassen, dass jede Kante Knoten aus verschiedenen Teilmengen verbindet.
- Ein Graph, in dem jede Kante als geordnetes Paar betrachtet wird, heißt *gerichtet (directed).*
  - Neudefinition von Pfad, Zyklus, Komponente, Zusammenhang:
  - Ein *gerichteter azyklischer Graph* ist ein gerichteter Graph, in dem es keine Zyklen unter Beachtung der Richtung gibt (*DAG - directed acyclic graph*)
  - Eine Knotenmenge heißt *schwache Zusammenhangskomponente*, wenn es zu jedem Knoten einen Pfad gibt, und *starke Zusammenhangskomponente*, wenn es von jedem Knoten der Komponente einen Pfad zu jedem anderen gibt
- Ein Graph heißt *gewichtet*, wenn jede Kante mit einem Gewicht versehen ist.

Graph als Abstrakter Datentyp

```
class Graph{
    Graph(int vertices, boolean directed)
    int V()
    int E()
    boolean directed()
    void insert(Edge e)
    void remove(Edge e)
    boolean edge(int from, int to)
    AdjList adjacents(int v)
}
›
```

```
interface AdjList{
    int beg();
    int nxt();
    boolean  end();
}
class Edge{
    int v,w;
    Edge(int v, int w){
        this.v=v; this.w=w;
    }
}
```

Graphen-Algorithmen

- Oft in Form von Bibliotheken
  - Repräsentation von Graphen
  - Realisierung von Algorithmen
- Berechnung von (Mess-)Werten für den Graph
- Berechnung einer Teilmenge von Knoten oder Kanten
- Überprüfung bestimmter Eigenschaften eines Graphen
- Oft unterteilt in Berechnugsphase (*prepare*) und Abfragephase (*query*)
  - z.B. Zusammenhang: Berechnungsphase bestimmt Zusammenhangskomponenten; danach Anfragen über Zahl der Komponenten, Liste der Knoten pro Komponente usw.

Darstellung von Graphen

- Adjazenzmatrix
  - V-mal-V-Matrix von Bits
  - ungerichteter Graph: Halbmatrix ist ausreichend; alternativ Speicherung einer symmetrischen Matrix
  - Berechnung der Adjazenzliste für einen Knoten in O(V)
  - besser geeignet für dichte Graphen
- Adjazenzliste
  - Pro Knoten (verkettete) Liste aller adjazenten Knoten
  - ungerichteter Graph: Doppelte Speicherung jeder Kante erlaubt Ermittlung der Adjazenzliste in O(1)
- Speicherbedarf?
- Gewichtete Graphen?

Bestimmung eines einfachen Pfades zwischen zwei Knoten

- Tiefe-zuerst-Suche (*depth first search*)
  - Lineare Zeit (O(V+E))
  - Vermeidung von Zyklen in der Suche?
  - Vermeidung von Stack-Überläufen durch Rekursion?

Hamilton-Pfad

- Gegeben zwei Knoten, gibt es einen Pfad, der jeden Knoten besucht?
  - rekursive Lösung: Gegeben Anfangsknoten, Endknoten und Zahl der noch zu besuchenden Knoten, finde einen weiteren Knoten, und suche dann rekursiv
  - Algorithmus hat im schlechtesten Fall exponentielle Laufzeit (evtl. sind alle Permutationen zu betrachten)

Euler-Pfad

- Gibt es einen Pfad, der alle Kanten genau einmal durchläuft?
  - Nach L. Euler: Wanderung über die Brücken von Königsberg
- Euler: Eine Euler-Tour (zyklischer Euler-Pfad) existiert genau dann, wenn der Graph zusammenhängend ist und jeder Knoten einen geraden Grad hat.
  - Ein Euler-Pfad existiert, wenn genau maximal zwei Knoten einen ungeraden Grad haben
- Euler-Tour kann in linearer Zeit gefunden werden

Graph-Probleme

- Einfacher Zusammenhang
- Starker Zusammenhang
- Transitive Hülle
- Minimaler Spannbaum (Spannbaum mit minimalem Gewicht)
- Kürzester (nach Gewicht) Pfad von einem Ausgangspunkt
- Planarität
- Einfärbung: kann man die Knoten mit k Farben färben, so dass keine adjazenten Knoten die gleiche Farbe bekommen?
- Graph-Isomorphismus
- Problem des Handelsreisenden (Traveling Salesperson Problem): Kürzester Zyklus, der alle Knoten besucht