Kap. 2 Graphen

Kap. 2.1 Netze

Beispiele: MVV, Autobahnen, WIN, E/R, LH-Flüge, Stützgraphen, Petri-Netze, SIPs, ISA-Hierarchien, Operator-Graphen, Wartegraphen für DB-Transaktionen

Abstraktion von Netz = Graph

- Knoten: $k_1, k_2, ..., k_i, ... k_n$
- Kanten zwischen je 2 Knoten

  Edge $e = (k_1, k_2) \in E$: binäre Relation

1

Varianten von Netzen und Graphen (Komplikationen)

1. Richtung: gerichtete bzw. ungerichtete Kanten

$\overrightarrow{(k_1, k_2)}$ $\overline{(k_1, k_2)}$

Fall 1: — Kurzschreibweise für $\rightleftharpoons$

d.h. E ist symmetrisch:

$(k_1, k_2) \in E$ $(k_2, k_1) \in E$ (implizit)

Fall 2: — ist ungerichtete Kante

$\overline{(k_1, k_2)}$

d.h. E ist symmetrisch mit zusätzlicher Äquivalenzrel. $\equiv_s$: $(k_1, k_2) \equiv_s (k_2, k_1)$,

Kanten sind $E / \equiv_s = \overline{E}$

2

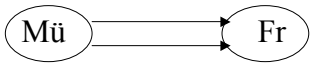
2. Bewertung von Kanten

für gerichtete und ungerichtete Kanten, z.B.:

- Transportkapazität: Baud, m^3/s
- Länge
- Geschwindigkeit
- Reisezeit
- Mautkosten, Flugkosten, Fahrkartenpreis
- allg.Prädikate und Strukturen

3. Multikanten:

z.B. Flüge, Züge mit mehrfachen Markierungen,
i.a. beliebig komplexe Zusatzinfo:



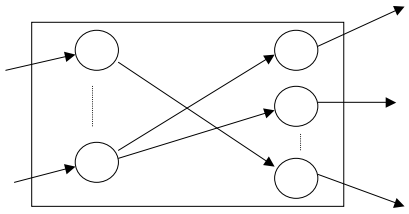
Flugges.	Flug#	Abfl.	Gate	Ank.	Masch.	Kap.
LH	451	7:45	A3	8:30	A310	240
LH	461	8:45	A4	9:30	737	130

3

4. Markierung von Knoten

Umsteigezeit: national 45 min.
international 70 min.

5. Hypernetze



z.B. zur Repräsentation von Verbindungsbedingungen für Wegekonstruktion, Komposition von Petri-Netzen

4

Kap. 2.2 Repräsentation von Graphen

1. Zeichnung, Bild: siehe Beispiele

2. Adjazenz matrix:

für spezielle Probleme bei gerichteten und ungerichteten Graphen, z.B.:

- Stützgraph: Zyklenfreiheit (Warshall)
- Komm. Netz: Erreichbarkeit
- Verkehrsnetze: kürzeste Wege

Knotennamen: 1, 2, ..., n oder Hilfsstruktur
Boolsche Edge-Matrix EM [1:n, 1:n]

EM[i,j] = true $\Leftrightarrow$ e = $(k_i, k_j) \in E$

5

Algorithmen:

Varianten der Matrixmultiplikation, z.B.

Erreichbarkeit: $k_i \rightarrow k_j \rightarrow k_l$

$$WM_2[i,l] := EM[i,l] \vee \bigvee_{j=1}^{n} EM[i,j] \wedge EM[j,l]$$

Struktur wie $\sum_{j=1}^{n} EM[i,j] * EM[j,l]$

Wegematrix WM_l

für Wege der Länge $\leq l$:

$WM_2 := EM + EM^2$
$WM_4 := WM_2 + WM_2^2$

$WM_{2^i} := WM_{2^{i-1}} + WM^2_{2^{i-1}}$

6

Wege sind höchstens von Länge n, d.h.
$\log_2 n$ Matrix-Multiplikationen der Komplexität n^3 für n Knoten.

Zeitkomplexität: $O(n^3 \log n)$
Laufzeit für $n = 10^6$ (Internet)?
Optimierung:
$WM_2[i,l]$:= **if** EM [i,l] **then true**

else $\bigvee_{j-1}^{n} EM[i, j] \wedge EM[j, l]$

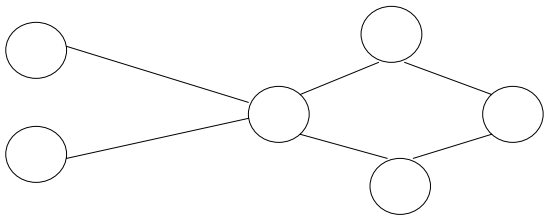
Platzkomplexität: $O(n^2)$
für $n = 10^6$ 125 GB

7

3. Knoten- und Kantenmengen:
Knoten = (k_i, Knoteninfo) oder
(i, Knoteninfo)
mit Zugriffsstruktur über k_i

Kante = (e_j, k_{j1}, k_{j2}, Kanteninfo)
mit Zugriffsstruktur über (e_i, k_{j1}, k_{j2})

4. Hybride Repr. nach Bedarf
z.B. für Spannbaum-Berechnung



8

Knotenzustände:  unbearbeitet ~ 0

 erreicht ~ 1

 bearbeitet ~ 2

mit charakteristischen Funktionen, Mengen, Listen, etc.

Notation: K_u : unbearbeitete Knoten
K_e : erreichte Knoten
K_b : bearbeitete Knoten

9

Alg. für Spannbaum: mit Knotenmenge K
{alle Knoten unbearbeitet}

$K_u := K$; $K_e := K_b := \varnothing$
select $k \in K_u$; $K_u := K_u \setminus \{k\}$;
$K_e := K_e \cup \{k\}$; {k ist erreicht}
while $K_e \neq \varnothing$ **do**
select $k \in K_e$; $K_e := K_e \setminus \{k\}$;
$K_b := K_b \cup \{k\}$; {k ist bearbeitet}
verfolge alle Kanten e von K aus:
if Endknoten k`von $e \in K_u$
then begin $K_u := K_u \setminus \{k`\}$;
$K_e := K_e \cup \{k`\}$;
e zu Spannbaum
end
od.

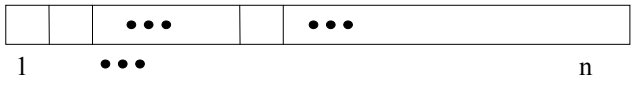
10

Operationen:

auf K_e : **select, delete, insert**
auf K_u : **select, delete, find**
auf K_b : **insert**
auf Kantenmenge: **find** e_j mit k_j, geg.
delete e_j oder bearbeite jede Kante zweimal

11

Variante 4.1: Charakteristische Funktion
für K_u, K_e, K_b
Knotenfeld für Zustände 0, 1, 2:



Analyse: **while** n-mal durchlaufen

select $k \in K_e$ ist O(n)

Alg. mindestens von $O(n^2)$

12

Variante 4.2: Wie 4.1, aber zusätzlich

Liste für K_e:

select $k \in K_e$ ist jetzt O(1)

Komplexität dominiert durch
Kantenverarbeitung, z.B.
pro Knoten eigene Kantenliste
- Sortierung der Kantenmenge oder
- Zugriffstrukturen für Kantenmenge für **find**
  O (m log m) bei m Kanten

13

Variante 4.3: Ku als Baum, z.B. AVL;
K_e als Liste
Knotenbearbeitung: ***O*** (n log n)
Kantenbearbeitung: wie 4.2

Lemma: Spannbaum hat n-1 Kanten

Bew: Übung

Offenes Problem in Alg.:
Darstellung des Spannbaumes;
hier nur als Kantenmenge

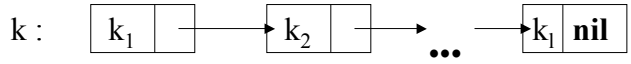
14

Platzkomplexität: bei Varianten

3 und 4 dominiert durch Kantenzahl, praktisch $\boldsymbol{O}(n)$

z.B. Kanten als Listen von Folgeknoten:

Für jeden Knoten k:



Falls $(k, k_1), (k, k_2), ..., (k, k_e) \in E$

i.e. Zeilenrepr. in dünn besetzter Adj.Matrix. Erforderliche Listenoperationen?

Speicherbedarf bei d Kanten pro Knoten: $n \cdot d \cdot 8$ Bytes

für $n = 10^6$, $d = 3$: 24 MB

Vergleich zu Adjazenzmatrix:

$$\frac{125\ \text{GB}}{24\ \text{MB}} \approx 5000 \text{ Verbesserungsfaktor}$$

15

Ausgesparte Probleme:

- Flüsse
- Wegesuche
- Planarität
- Färbung

} Siehe Spezialvorlesung Optimierung Graphentheorie

16