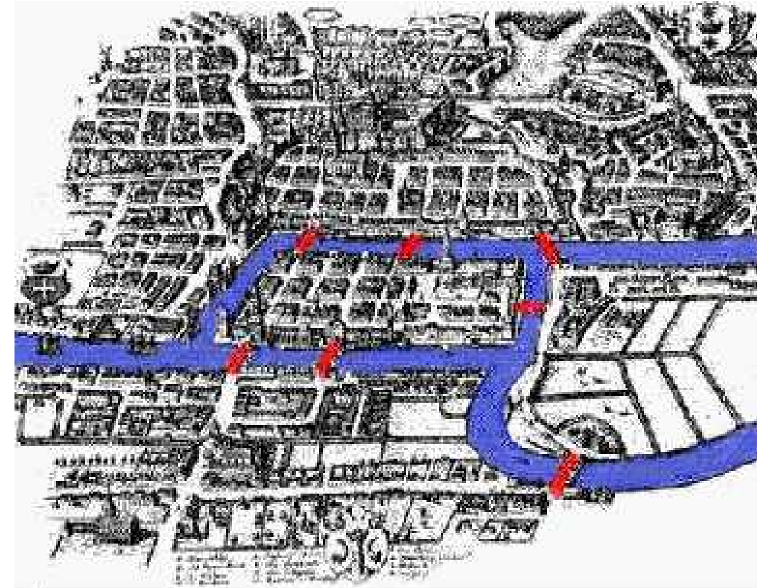
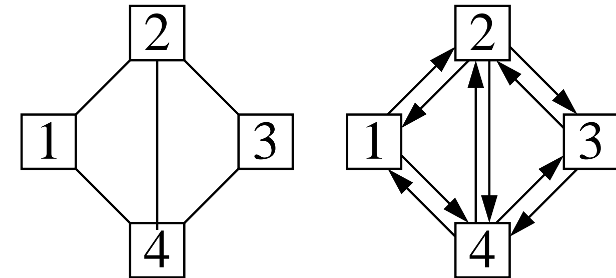
8 Graphrepräsentation



- 1736 fragt L. Euler die folgende "touristische" Frage:
- Straßen- oder Computernetzwerke
- Zugverbindungen (Raum und Zeit)
- Soziale Netzwerke (Freundschafts-, Zitier-, Empfehlungs-,. . .)
- Aufgabenabhängigkeiten $\rightsquigarrow$ scheduling Probleme
- Werte und arithmetische Operationen $\rightsquigarrow$ Compilerbau
- . . .

Graphrepräsentation



Meist repräsentieren wir

ungerichtete Graphen durch bigerichtete Graphen

⇝ wir konzentrieren uns auf gerichtete Graphen

- Was zählt sind die Operationen
- Eine triviale Repräsentation
- Felder
- Verkettete Listen
- Matrizen
- Implizit
- Diskussion

Operationen

Ziel: $\mathrm{O}(1)$ für alle elementaren Operationen

Zugriff auf assoziierte Information: Felder ($V = 1..n$), Hashing

Navigation: Gegeben v, finde ausgehende Kanten.

(ggf. auch eingehende Kanten.)

Kantenanfragen: $(u, v) \in E$? Adjanzenzmatrizen, Hashtabellen.

Umkehrzugriff: Gegeben (u, v) finde (v, u) Hashing, Kantenobjekte

Konstruktion, Konversion und Ausgabe ($\mathrm{O}(m+n)$ Zeit)

Update: Knoten/Kanten einfügen und löschen – der schwierige Teil.

$n =$ #Knoten, $m =$ #Kanten

Kantenfolgenrepräsentation

Folge von Knotenpaaren (oder Tripel mit Kantengewicht)

+ kompakt

+ gut für I/O

− Fast keine nützlichen Operationen ausser alle Kanten durchlaufen

Beispiele: isolierte Knoten suchen, Kruskals MST-Algorithmus

Konvertierung.

Adjazenzfelder

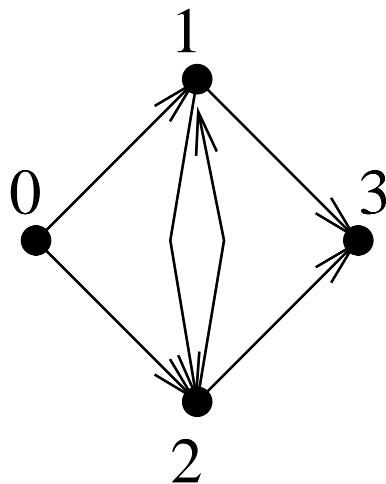
$V = 0..n-1$

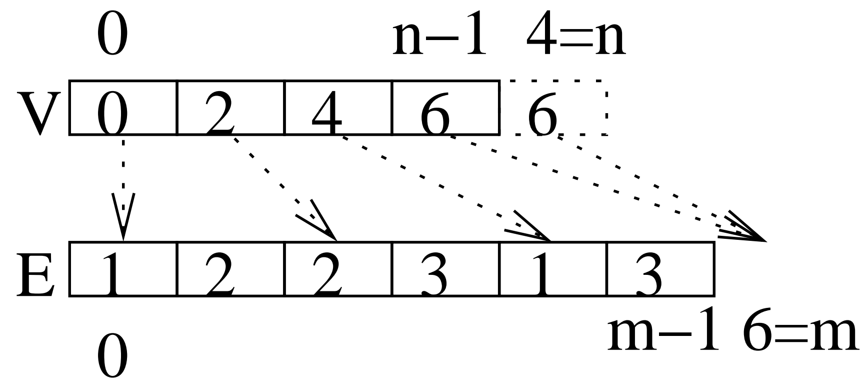
Kantenfeld E speichert Ziele

gruppiert nach Startknoten

V speichert Index der ersten ausgehenden Kante

Dummy-Eintrag $V[n]$ speichert $m+1$





Beispiel: Ausgangsgrad$(v) = \mathtt{V}[v+1] - \mathtt{V}[v]$

Kantenliste $\rightarrow$ Adjacency Array

Zur Erinnerung: Ksort

Function adjacencyArray(EdgeList)

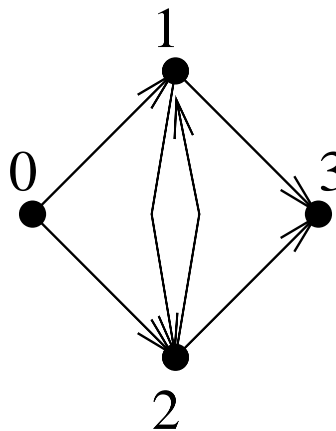
$V=\langle 0,\ldots,0\rangle$: **Array** $[0..n]$ **of** $\mathbb{N}$

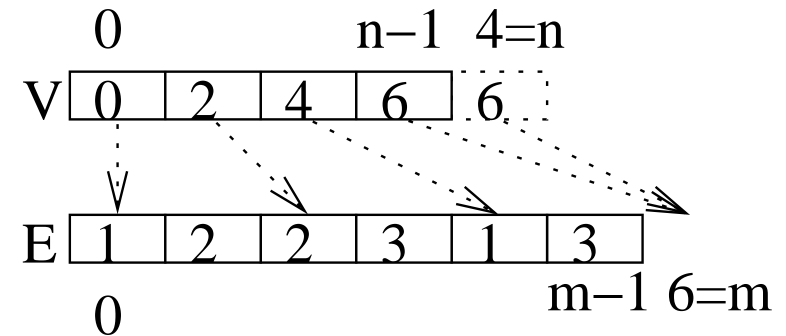
foreach $(u,v) \in$ EdgeList **do** $V[u]{+}{+}$ // count

for $v := 1$ **to** n **do** $V[v] += V[v-1]$ // prefix sums

foreach $(u,v) \in$ EdgeList **do** $E[--V[u]] = v$ // place

return (V,E)





Operationen für Adjanzenzfelder

Navigation: einfach

Kantengewichte: E wird Feld von Records (oder mehrere Felder)

Eingehende Kanten: umgedrehten Graphen speichern

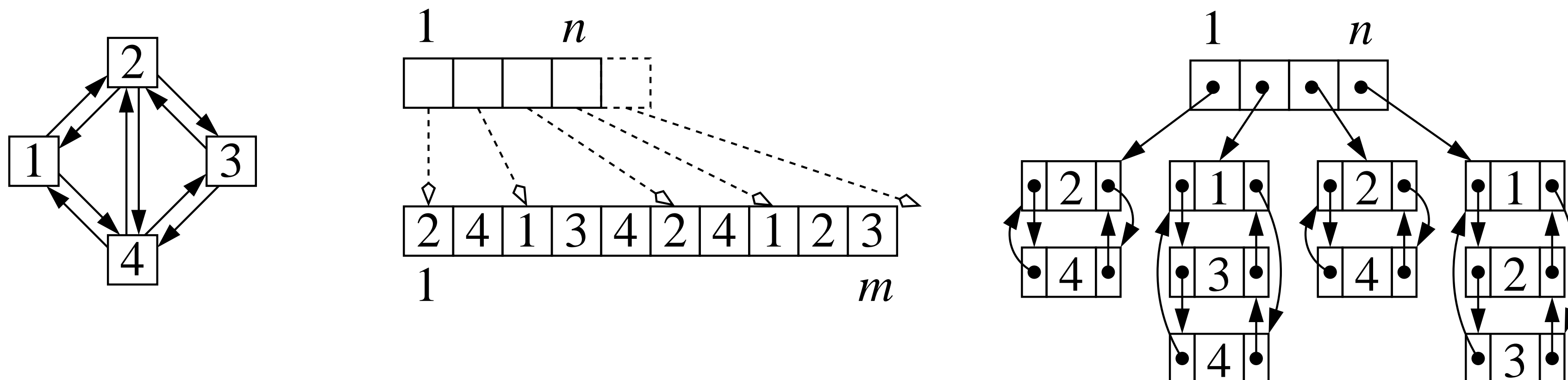
Kanten löschen: explizite Endindizes

Batched Updates: neu aufbauen

Adjazenzlisten

speichere (doppelt) verkettete Liste adjazenter Kanten für jeden Knoten.

+ einfaches Einfügen von Kanten
+ einfaches Löschen von Kanten (ordnungserhaltend)
− mehr Platz (bis zu Faktor 3) als Adjazenzfelder
− mehr Cache-Misses



Enhancing Adjacency Lists

For reverse edge access or edge weight updates in undirected graphs:
explicit edge objects

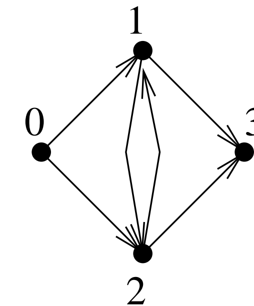
- stores weights
- pred/succ-pointers for all adjacency lists containing this edge
- reverse pointer for directed graphs
- cute trick: store only `xor` of incident nodes

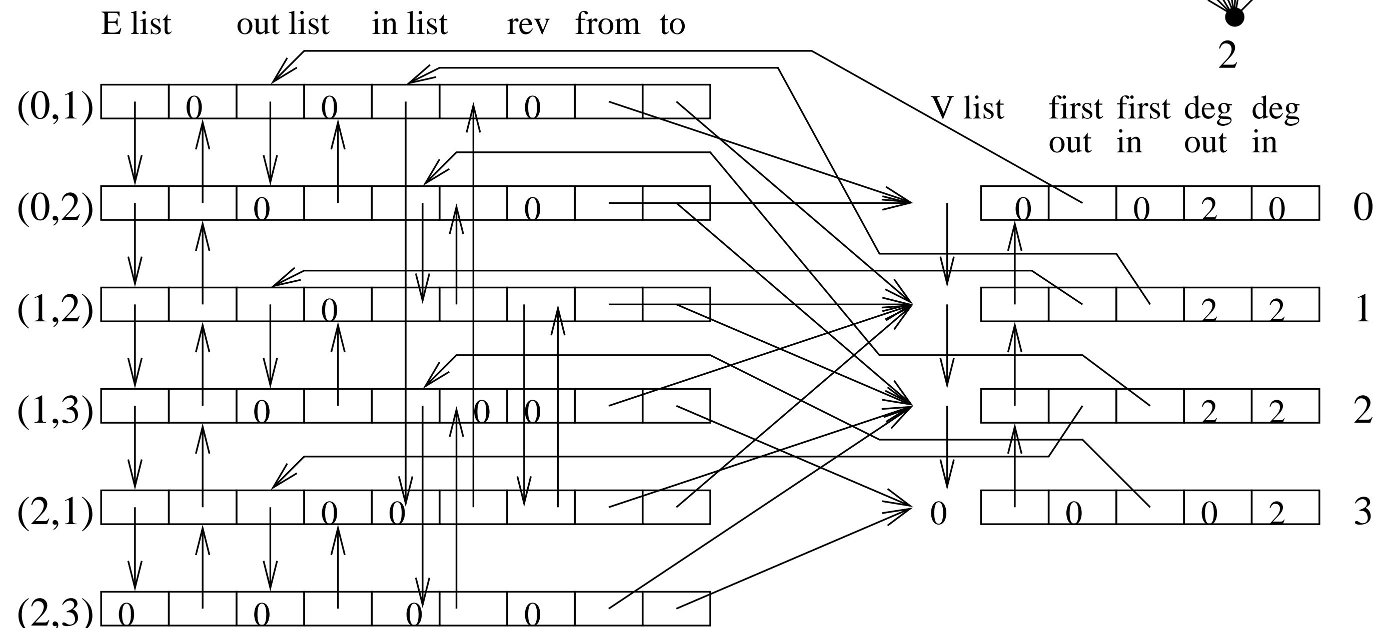
Node insertion/deletion

mark as deleted $\vee$ swap in node n $\vee$ list of nodes

$\Rightarrow$ node handles are pointers.

Problem: graphs with common node set





$\approx 8\times$ more space than vanilla adjacency arrays

Customization

Customize data structure for application for maximum speed/compactness.

Sofware Engineering nightmare

Seperating algorithm from their representation may be a way out

Adjazenz-Matrix

$A \in \{0,1\}^{n \times n}$ with $A(i,j) = [(i,j) \in E]$

$+$ platzeffizient für sehr dichte Graphen

$--$ platzineffizient sonst. Übung: was bedeutet "sehr dicht" hier?

$+$ einfache Kantenanfragen

$-$ langsame Navigation

$++$ verbindet lineare Algebra und Graphentheorie

Beispiel: $\mathbf{C} = \mathbf{A}^k$. $\mathbf{C}_{ij} =\#$ k-Kanten-Pfade von i nach j

Übung: zähle Pfade der Länge $\leq k$

Wichtige Beschleunigungstechniken:

$\mathrm{O}(\log k)$ Matrixmult. für Potenzberechnung

Matrixmultiplikation in subkubischer Zeit, z.B., Strassens Algorithmus

Example where graph theory helps LA

Problem: solve $\mathbf{Bx} = \mathbf{c}$ Consider $G = (1..n, E = \{(i,j) : B_{ij} \neq 0\})$

Assume G has two connected components $\Rightarrow$

we swap rows and cols such that

$$\begin{pmatrix} \mathbf{B}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{B}_2 \end{pmatrix} \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix} = \begin{pmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \end{pmatrix} .$$

Exercise: What if G is a DAG?

Implizite Repräsentation

Kompakte Repräsentation möglicherweise sehr dichter Graphen

Implementiere Algorithmen direkt mittels dieser Repr.

Beispiel: Intervall-Graphen

Knoten: Intervalle $[a,b] \subseteq \mathbb{R}$

Kanten: zwischen überlappenden Intervallen

Zusammenhangstest für Intervallgraphen

$V = \{[a_1, b_1], \ldots, [a_n, b_n]\}$

$E = \{\{[a_i, b_i], [a_j, b_j]\} : [a_i, b_i] \cap [a_j, b_j] \neq \emptyset\}$

Idee: durchlaufe Intervalle von links nach rechts. Die Anzahl überlappender Intervalle darf nie auf Null sinken.

Function isConnected(L : SortedListOfIntervalEndPoints) : $\{0, 1\}$

  remove first element of L; overlap := 1

  foreach $p \in L$ **do**

    if overlap$= 0$ **return** 0

    if p is a start point **then** overlap$++$

    else overlap$--$ // end point

  return 1

$O(n \log n)$ Algorithmus für bis zu $O(n^2)$ Kanten!

Übung: Zusammenhangskomponenten finden