Karlsruhe Institute of Technology

Grundbegriffe der Informatik - Tutorium 21

Christian Jülg

Wintersemester 2012/13

18. Dezember 2012

http://gbi-tutor.blogspot.com

Übersicht

Übersicht

Zum Warmwerden...

Graphen...

1. ... lassen sich nicht immer als Matrix repräsentieren
2. ... haben im Allgemeinen viele isomorphe Darstellungen
3. ... erlauben manchmal einen effizienteren Zugriff über die Adjazenzlistendarstellung

Sei W eine Wegematrix...

1. ... dann entspricht W der Erreichbarkeitsrelation $E^* = \bigcup_{i=0}^{n-1} E^i$
2. ... dann ist die Hauptdiagonale nicht voll besetzt
3. ... dann gilt $W = W^*$

Bei der Aufwandsanalyse...

1. ... zählen einfache Rechenoperationen stärker als Zuweisungen
2. ... sind alle atomaren Operationen gleich
3. ... sind die Schleifen das Wichtigste

Zum Warmwerden...

Graphen...

1. ... lassen sich nicht immer als Matrix repräsentieren
2. ... haben im Allgemeinen viele isomorphe Darstellungen
3. ... erlauben manchmal einen effizienteren Zugriff über die Adjazenzlistendarstellung

Sei W eine Wegematrix...

1. ... dann entspricht W der Erreichbarkeitsrelation $E^* = \bigcup_{i=0}^{n-1} E^i$
2. ... dann ist die Hauptdiagonale nicht voll besetzt
3. ... dann gilt $W = W^*$

Bei der Aufwandsanalyse...

1. ... zählen einfache Rechenoperationen stärker als Zuweisungen
2. ... sind alle atomaren Operationen gleich
3. ... sind die Schleifen das Wichtigste

Zum Warmwerden...

Graphen...

1. ... lassen sich nicht immer als Matrix repräsentieren
2. ... haben im Allgemeinen viele isomorphe Darstellungen
3. ... erlauben manchmal einen effizienteren Zugriff über die Adjazenzlistendarstellung

Sei W eine Wegematrix...

1. ... dann entspricht W der Erreichbarkeitsrelation $E^* = \bigcup_{i=0}^{n-1} E^i$
2. ... dann ist die Hauptdiagonale nicht voll besetzt
3. ... dann gilt $W = W^*$

Bei der Aufwandsanalyse...

1. ... zählen einfache Rechenoperationen stärker als Zuweisungen
2. ... sind alle atomaren Operationen gleich
3. ... sind die Schleifen das Wichtigste

Zum Warmwerden...

Graphen...

1. ... lassen sich nicht immer als Matrix repräsentieren
2. ... haben im Allgemeinen viele isomorphe Darstellungen
3. ... erlauben manchmal einen effizienteren Zugriff über die Adjazenzlistendarstellung

Sei W eine Wegematrix...

1. ... dann entspricht W der Erreichbarkeitsrelation $E^* = \bigcup_{i=0}^{n-1} E^i$
2. ... dann ist die Hauptdiagonale nicht voll besetzt
3. ... dann gilt $W = W^*$

Bei der Aufwandsanalyse...

1. ... zählen einfache Rechenoperationen stärker als Zuweisungen
2. ... sind alle atomaren Operationen gleich
3. ... sind die Schleifen das Wichtigste

Übersicht

Aufgabenblatt 7

Blatt 7

- Abgaben: 18 / 19
- Punkte: Durchschnitt der abgegeben Blätter: 13, 7 / 20

häufige Fehler

7.3 ein Beispiel ist kein Beweis

Aufgabenblatt 8

Blatt 8

- Abgaben: 19 / 19
- Punkte: Durchschnitt der abgegeben Blätter: $15,2$ / 19

häufige Fehler

8.2c wenn eine Begründung gefordert ist, gebt eine an

Aufgabenblatt 8

Blatt 8

- Abgaben: 19 / 19
- Punkte: Durchschnitt der abgegeben Blätter: $15,2$ / 19

häufige Fehler

8.2c wenn eine Begründung gefordert ist, gebt eine an

8.3 Warshall Alorithmus: W_1 enth. !nicht! einfach alle Wege der Länge 1

Übersicht

Aufgabenblatt 9

Blatt 9

- Abgabe: 21.12.2012 um 12:30 Uhr im Untergeschoss des Infobaus
- Punkte: maximal 20

Themen

- Effizienzklassen
- Effizienz von Algorithmen

Übersicht

Wegematrix I

Darstellung von Relationen

So wie die Adjazenzmatrix Relationen zwischen Knoten darstellt, können auch weitere Relationen als Matrix dargestellt werden. Ein Beispiel ist die Wegematrix, die eine Darstellungsform der Erreichbarkeitssrelation $E^* = \bigcup_{i=0}^{n-1} E^i$.

Für die Wegematrix gilt

$$W_{ij} = \begin{cases} 1, & \text{falls es in G einen Pfad von i nach j gibt} \\ 0, & \text{falls es in G keinen Pfad von i nach j gibt} \end{cases}$$

Aufwand

Zählweise

Beim Vergleich verschiedener Algorithmen in Bezug auf den Aufwand, sucht man nach einem Maß für die Anzahl der Rechenoperationen für eine Aufgabe der Größe n.

Beispiel

Aufwand

Zählweise

Beim Vergleich verschiedener Algorithmen in Bezug auf den Aufwand, sucht man nach einem Maß für die Anzahl der Rechenoperationen für eine Aufgabe der Größe n.

Beispiel

Summe aller Zahlen von 1 bis n:
$\sum_{i=0}^{n} i =$

Aufwand

Zählweise

Beim Vergleich verschiedener Algorithmen in Bezug auf den Aufwand, sucht man nach einem Maß für die Anzahl der Rechenoperationen für eine Aufgabe der Größe n.

Beispiel

Summe aller Zahlen von 1 bis n:
$\sum_{i=0}^{n} i = n * (n+1)/2$

Algorithmus I

```
1  // Matrix A sei die Adjazenzmatrix
2  // Matrix W wird am Ende die Wegematrix enthalten
3
4  // Matrix M wird benutzt um A^i zu berechnen
5  W ← 0 // Nullmatrix
6  for i ← 0 to n - 1 do
7     M ← Id // Einheitsmatrix
8     for j ← 1 to i do
9        M ← M · A // Matrixmultiplikation
10    od
11    W ← W + M // Matrixaddition
12 od
13 W ← sgn(W)
```

Algorithmus II

```
1  // Matrix A sei die Adjazenzmatrix
2  // Matrix W wird am Ende die Wegematrix enthalten
3
4  // Matrix M wird benutzt um A^i zu berechnen
5  W ← 0 // Nullmatrix
6  M ← Id // Einheitsmatrix
7  for i ← 0 to n - 1 do
8    W ← W + M // Matrixaddition
9    M ← M · A // Matrixmultiplikation
10 od
11 W ← sgn(W)
```

Algorithmus III

$log_2(n) \cdot n^3$

Wir verwenden einen Trick:

Algorithmus III

$log_2(n) \cdot n^3$

Wir verwenden einen Trick:

- $F = E^0 \cup E^1 = Id_V \cup E$

Algorithmus III

$log_2(n) \cdot n^3$

Wir verwenden einen Trick:

- $F = E^0 \cup E^1 = Id_V \cup E$
- dann $F^2 = (E^0 \cup E^1) \circ (E^0 \cup E^1) = E^0 \cup E^1 \cup E^1 \cup E^2 = E^0 \cup E^1 \cup E^2$

Algorithmus III

$log_2(n) \cdot n^3$

Wir verwenden einen Trick:

- $F = E^0 \cup E^1 = Id_V \cup E$
- dann $F^2 = (E^0 \cup E^1) \circ (E^0 \cup E^1) = E^0 \cup E^1 \cup E^1 \cup E^2 = E^0 \cup E^1 \cup E^2$
- und $E^* = \bigcup_{i=0}^{\infty} E^i = \bigcup_{i=0}^{2^m} E^i = F^{2^m}$

Algorithmus III

$log_2(n) \cdot n^3$

Wir verwenden einen Trick:

- $F = E^0 \cup E^1 = Id_V \cup E$
- dann $F^2 = (E^0 \cup E^1) \circ (E^0 \cup E^1) = E^0 \cup E^1 \cup E^1 \cup E^2 = E^0 \cup E^1 \cup E^2$
- und $E^* = \bigcup_{i=0}^{\infty} E^i = \bigcup_{i=0}^{2^m} E^i = F^{2^m}$
- wobei $m = \lceil \log_2 n \rceil$

Algorithmus III

$log_2(n) \cdot n^3$

Wir verwenden einen Trick:

- $F = E^0 \cup E^1 = Id_V \cup E$
- dann $F^2 = (E^0 \cup E^1) \circ (E^0 \cup E^1) = E^0 \cup E^1 \cup E^1 \cup E^2 = E^0 \cup E^1 \cup E^2$
- und $E^* = \bigcup_{i=0}^{\infty} E^i = \bigcup_{i=0}^{2^m} E^i = F^{2^m}$
- wobei $m = \lceil \log_2 n \rceil$

Aufwandsvergleich

Wenn man in einem bestimmten Zeitraum mit dem n^5 Algorithmus gerade noch die Problemgröße $n = 1000$ schafft:

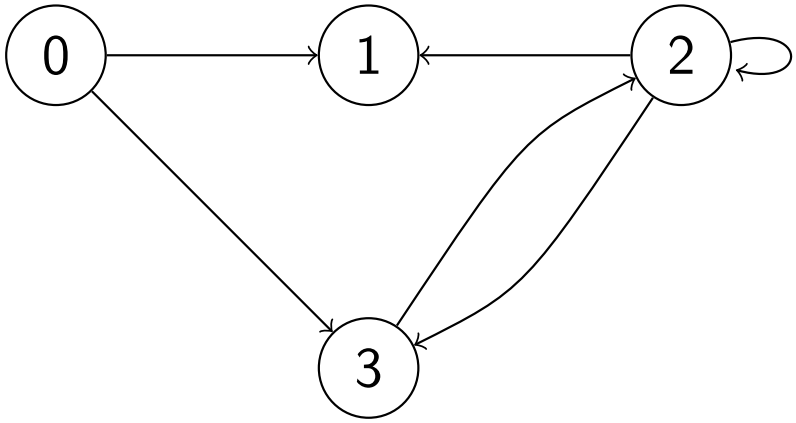
- Wie große Probleminstanzen schafft man mit dem n^4 Algorithmus?

Aufwandsvergleich

Wenn man in einem bestimmten Zeitraum mit dem n^5 Algorithmus gerade noch die Problemgröße $n = 1000$ schafft:

- Wie große Probleminstanzen schafft man mit dem n^4 Algorithmus?
- ... oder mit dem $log_2(n) \cdot n^3$ Algorithmus?

Wegematrix

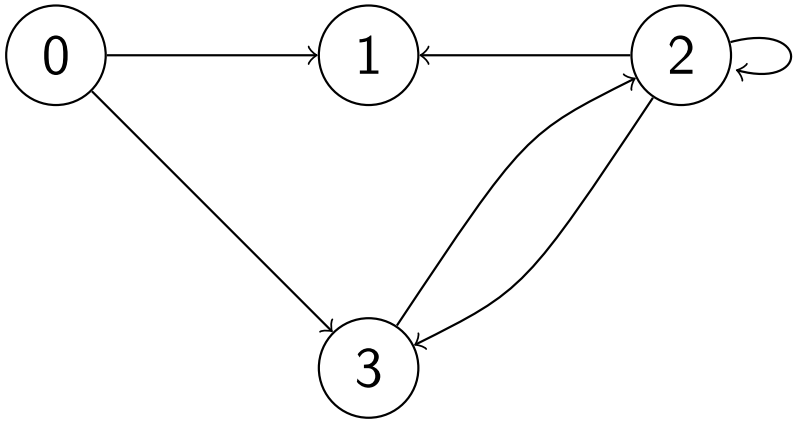


$$\begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Ihr seid dran...

- Wie sieht die Wegematrix zum oben gezeigten Graph aus?

Wegematrix

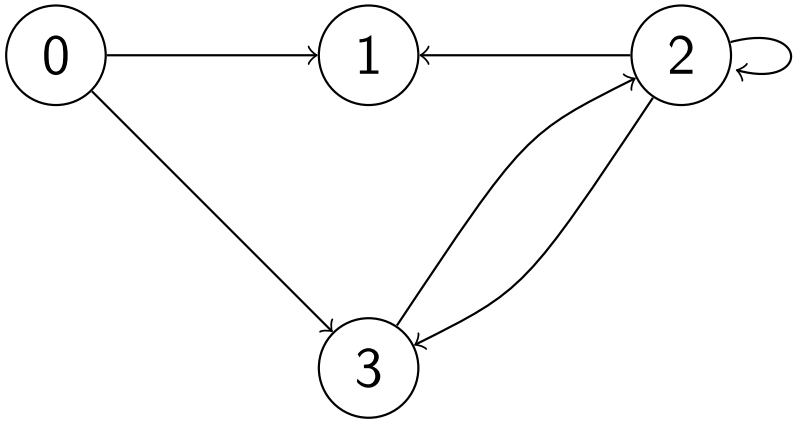


$$\begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Ihr seid dran...

- Wie sieht die Wegematrix zum oben gezeigten Graph aus?
- Wie sieht die Wegematrix für eine vollständig mit 1en gefüllte Matrix aus?

Wegematrix



$$\begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Ihr seid dran...

- Wie sieht die Wegematrix zum oben gezeigten Graph aus?
- Wie sieht die Wegematrix für eine vollständig mit 1en gefüllte Matrix aus?
- Wann gilt allgemein $W = A$? Wann gilt $E^1 = A$?

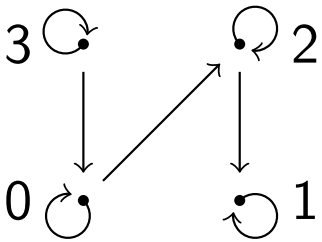
Übersicht

Reflexive Transitive Hülle nach Warshall

Gegeben sei eine reflexive Relation R über einer Eckenmenge $V = \{0, \ldots, n-1\}$.

$$\sigma^{(k)} = \{(i,j) \in V \times V | \exists \text{Weg } i \to e_1 \to \cdots \to e_l \to j \text{ mit } l \leq k+1, e_r \in \{0, \ldots, k\} \text{ für } 1 \leq r \leq l\}$$

Das entspricht allen Wegen im Graph, die über höchstens $k+1$ „Zwischenstationen“ bei Knoten aus $\mathbb{G}_{k+1}$ möglich sind.



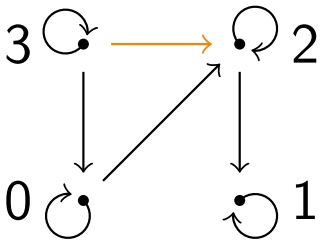
$$\sigma^{(0)} : \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

Reflexive Transitive Hülle nach Warshall

Gegeben sei eine reflexive Relation R über einer Eckenmenge $V = \{0, \ldots, n-1\}$.

$$\sigma^{(k)} = \{(i,j) \in V \times V | \exists \text{Weg } i \to e_1 \to \cdots \to e_l \to j \text{ mit } l \leq k+1, e_r \in \{0, \ldots, k\} \text{ für } 1 \leq r \leq l\}$$

Das entspricht allen Wegen im Graph, die über höchstens $k+1$ „Zwischenstationen“ bei Knoten aus $\mathbb{G}_{k+1}$ möglich sind.



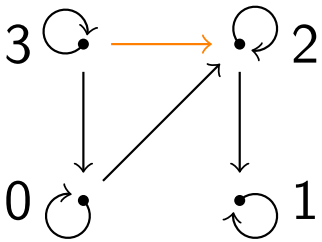
$$\sigma^{(0)} : \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

Reflexive Transitive Hülle nach Warshall

Gegeben sei eine reflexive Relation R über einer Eckenmenge $V = \{0, \ldots, n-1\}$.

$$\sigma^{(k)} = \{(i,j) \in V \times V | \exists \text{Weg } i \to e_1 \to \cdots \to e_l \to j \text{ mit } l \leq k+1, e_r \in \{0, \ldots, k\} \text{ für } 1 \leq r \leq l\}$$

Das entspricht allen Wegen im Graph, die über höchstens $k+1$ „Zwischenstationen“ bei Knoten aus $\mathbb{G}_{k+1}$ möglich sind.



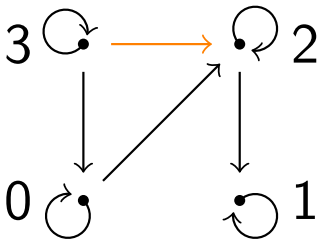
$$\sigma^{(0)}: \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \qquad \sigma^{(1)}: \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

Reflexive Transitive Hülle nach Warshall

Gegeben sei eine reflexive Relation R über einer Eckenmenge $V = \{0, \ldots, n-1\}$.

$$\sigma^{(k)} = \{(i,j) \in V \times V | \exists \text{Weg } i \to e_1 \to \cdots \to e_l \to j \text{ mit } l \leq k+1, e_r \in \{0, \ldots, k\} \text{ für } 1 \leq r \leq l\}$$

Das entspricht allen Wegen im Graph, die über höchstens $k+1$ „Zwischenstationen“ bei Knoten aus $\mathbb{G}_{k+1}$ möglich sind.



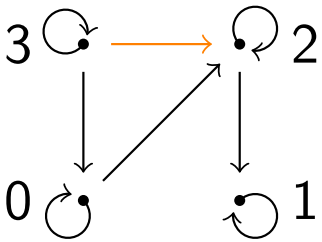
$$\sigma^{(0)}: \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \qquad \sigma^{(1)}: \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

Reflexive Transitive Hülle nach Warshall

Gegeben sei eine reflexive Relation R über einer Eckenmenge $V = \{0, \ldots, n-1\}$.

$$\sigma^{(k)} = \{(i,j) \in V \times V | \exists \text{Weg } i \to e_1 \to \cdots \to e_l \to j \text{ mit } l \leq k+1, e_r \in \{0, \ldots, k\} \text{ für } 1 \leq r \leq l\}$$

Das entspricht allen Wegen im Graph, die über höchstens $k+1$ „Zwischenstationen“ bei Knoten aus $\mathbb{G}_{k+1}$ möglich sind.



$$\sigma^{(0)}: \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

$$\sigma^{(1)}: \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

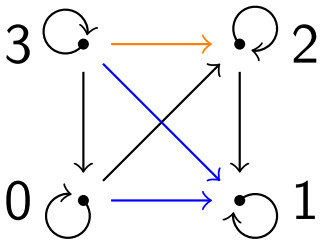
$$\sigma^{(2)}: \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

Reflexive Transitive Hülle nach Warshall

Gegeben sei eine reflexive Relation R über einer Eckenmenge $V = \{0, \ldots, n-1\}$.

$$\sigma^{(k)} = \{(i,j) \in V \times V | \exists \text{Weg } i \to e_1 \to \cdots \to e_l \to j \text{ mit } l \leq k+1, e_r \in \{0, \ldots, k\} \text{ für } 1 \leq r \leq l\}$$

Das entspricht allen Wegen im Graph, die über höchstens $k+1$ „Zwischenstationen“ bei Knoten aus $\mathbb{G}_{k+1}$ möglich sind.



$$\sigma^{(0)}: \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \qquad \sigma^{(1)}: \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

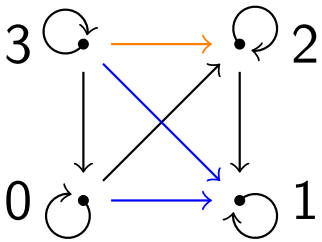
$$\sigma^{(2)}: \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Reflexive Transitive Hülle nach Warshall

Gegeben sei eine reflexive Relation R über einer Eckenmenge $V = \{0, \ldots, n-1\}$.

$$\sigma^{(k)} = \{(i,j) \in V \times V | \exists \text{Weg } i \to e_1 \to \cdots \to e_l \to j \text{ mit } l \leq k+1, e_r \in \{0, \ldots, k\} \text{ für } 1 \leq r \leq l\}$$

Das entspricht allen Wegen im Graph, die über höchstens $k+1$ „Zwischenstationen“ bei Knoten aus $\mathbb{G}_{k+1}$ möglich sind.



$$\sigma^{(0)}: \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \qquad \sigma^{(1)}: \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

$$\sigma^{(2)}: \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix} \qquad \sigma^{(3)}: \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Der Warshall-Algorithmus

Anforderungsbeschreibung

Eingabe: Adjazenzmatrix A einer Relation σ

Ausgabe: Adjanzenzmatrix S von σ^*

Der Warshall-Algorithmus

Anforderungsbeschreibung

Eingabe: Adjazenzmatrix A einer Relation σ

Ausgabe: Adjanzenzmatrix S von σ^* (entspricht der Erreichbarkeitsrelation)

Der Warshall-Algorithmus

Anforderungsbeschreibung

Eingabe: Adjazenzmatrix A einer Relation σ

Ausgabe: Adjanzenzmatrix S von σ^* (entspricht der Erreichbarkeitsrelation)

Der Algorithmus

```
1  S := A
2   for i = 0,...,n-1 set s_ii := 1
3
4   for k = 0,...,n-1
5     for i = 0,...,n-1
6       for j = 0,...,n-1
7         if (s_ij + s_ik * s_kj) ≥ 1 set s_ij := 1
```

Übersicht

Effizienzberechnung

Problem

- Ist ein Algorithmus besser als ein anderer?
- Gilt das auch auf einer anderen Rechenmaschine?
- Gilt es auch, wenn die Datenmenge weiter wächst?

Effizienzberechnung

Karlsruhe Institute of Technology

Problem

- Ist ein Algorithmus besser als ein anderer?
- Gilt das auch auf einer anderen Rechenmaschine?
- Gilt es auch, wenn die Datenmenge weiter wächst?

Lösungsansatz

Wir abstrahieren die Effizienz von Algorithmen:

- unabhängig von der Rechenmaschine
- in Abhängigkeit der Eingabelänge (meist n)
- mathematisch fundiert und beweisbar

So kann anhand der oberen Schranke eines Algorithmus z.B. gesagt werden, dass er im schlimmsten Fall (Worst-Case) signifikant langsamer ist, als ein anderer.

O-Kalkül

Karlsruhe Institute of Technology

Definition

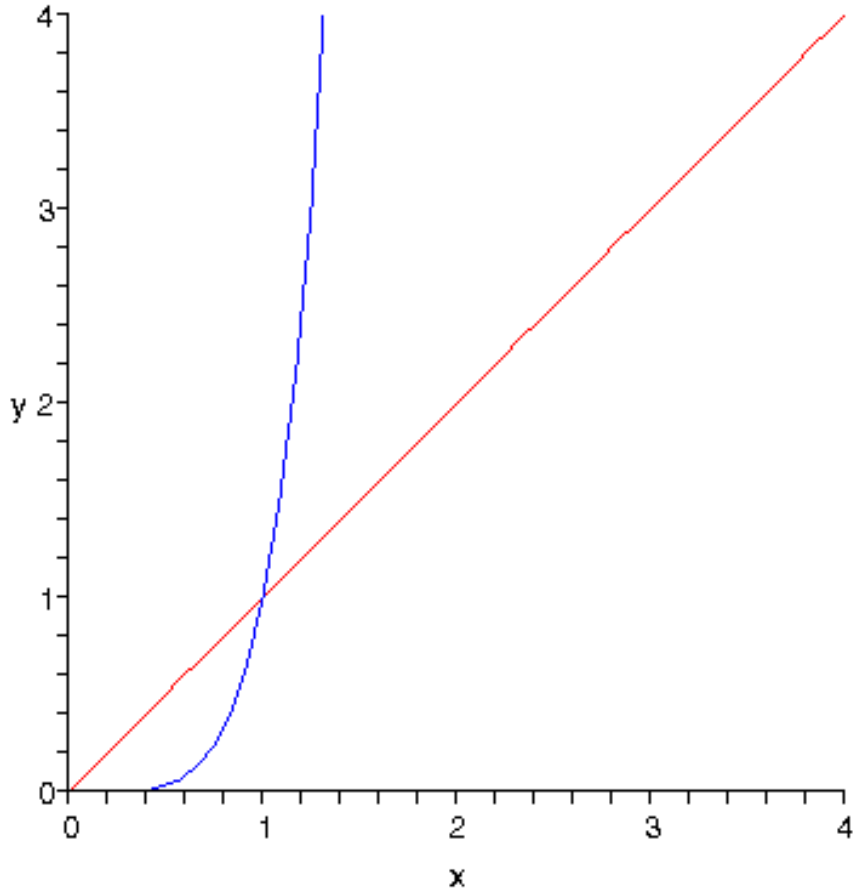
$$O(g(n)) = \{f(n) | \exists c > 0\ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 : 0 \leq f(n) \leq c \cdot g(n)\}$$

Umgangssprachlich

$O(g(n))$ enthält alle nicht-negativen Funktionen, die höchstens so schnell wie $g(n)$ wachsen.
Dabei kümmern wir uns nicht

- darum, was am Anfang passiert ($\exists n_0 \in \mathbb{N} \ldots \forall n \geq n_0$).
- um einfache Faktoren ($\exists c \in \mathbb{R} \ldots c \cdot g(n)$).

O-Kalkül



Rechnen mit dem O-Kalkül

Man kann aus der obigen Definition ein paar Regeln ableiten, mit denen der Umgang deutlich einfacher wird.
(Eigentlich rechnet man hier mit Mengen, und auch sonst ist die Schreibweise nicht mathematisch korrekt, aber das wird meistens unter den Teppich gekehrt.)

Faustregeln

- $O(1) + O(23) + O(4) = O(1)$
- $O(f(n)) + O(f(n)) = O(f(n))$
- $O(a \cdot f(n)) = O(f(n)) \ \forall a \in \mathbb{R}$
- $O(a \cdot n^2 + b \cdot n + c) = O(n^2) \ \forall a, b, c \in \mathbb{R}$

„Der Stärkere gewinnt!“

Aufwandsklassen

Fallunterscheidung: Aufwandsklassen

O-Kalkül	Obere Schranke, die der Algorithmus erreichen, aber nicht überschreiten kann
Ω-Kalkül	Untere Schranke und ein "Mindestaufwand", den der Algorithmus hat
Θ-Kalkül	Vereinigung der Betrachtung aus $\Omega(n)$ und $O(n)$. Es entsteht eine Art Funktionsbereich, den der Algorithmus nie verlässt.

Aufwandsklassen

Obere asymptotische Schranke
$O(g(n)) = \{f(n) \mid$
$\exists c \in \mathbb{R}^+, n_0 \in \mathbb{N} \, \forall n > n_0 : 0 \leq f(n) \leq c \cdot g(n)\}$

Untere asymptotische Schranke
$\Omega(g(n)) = \{f(n) \mid$
$\exists c \in \mathbb{R}^+, n_0 \in \mathbb{N} \, \forall n > n_0 : 0 \leq c \cdot g(n) \leq f(n)\}$

Asymptotisch scharfe Schranke
$\Theta(g(n)) = \{f(n) \mid$
$\exists c_1, c_2 \in \mathbb{R}^+, n_0 \in \mathbb{N} \, \forall n > n_0 : 0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)\}$

Beachte:
Alle Kalküle geben eine **Menge** von Funktionen an. $f(n) = O(n^2)$ bedeutet also eigentlich $f(n) \in O(n^2)$!

Rechenregeln

Reflexivität

- $f(n) \in O(f(n))$
- $g(n) \in \Omega(g(n))$
- $h(n) \in \Theta(h(n))$

Symmetrie

Hier gilt nur: $f(n) \in \Theta(g(n)) \Leftrightarrow g(n) \in \Theta(f(n))$

asymptotisches Wachstum

- $O(n^2 + n + log(n)) = O(n^2)$
- $\Omega(n^2 + n + log(n)) = \Omega(n^2) \subset \Omega(log(n))$

Logarithmen

Karlsruhe Institute of Technology

Achtung!

Logarithmen haben alle asymptotisch das gleiche Wachstum:

Logarithmen

Karlsruhe Institute of Technology

Achtung!

Logarithmen haben alle asymptotisch das gleiche Wachstum:

- z.z.: $\log_2(n) \in \Theta(\log_8(n))$

Logarithmen

Achtung!

Logarithmen haben alle asymptotisch das gleiche Wachstum:

- z.z.: $\log_2(n) \in \Theta(\log_8(n))$
- $n = 8^{\log_8 n} = (2^3)^{\log_8(n)} = 2^{3\log_8(n)}$, also gilt für alle $n \geq 1$: $\log_2(n) = 3\log_8(n)$ und $\log_8(n) = \frac{1}{3}\log_2(n)$
- allgemein:

Logarithmen

Achtung!

Logarithmen haben alle asymptotisch das gleiche Wachstum:

- z.z.: $\log_2(n) \in \Theta(\log_8(n))$
- $n = 8^{\log_8 n} = (2^3)^{\log_8(n)} = 2^{3\log_8(n)}$, also gilt für alle $n \geq 1$:
  $\log_2(n) = 3\log_8(n)$ und $\log_8(n) = \frac{1}{3}\log_2(n)$
- allgemein:
  $\log_b(n) \in \Theta(\log_a(n))$, denn

$$b^{\log_b(n)} = n = a^{\log_a(n)} = (b^{\log_b(a)})^{\log_a(n)} = b^{\log_b(a)\cdot\log_a(n)}$$

Logarithmen

Achtung!

Logarithmen haben alle asymptotisch das gleiche Wachstum:

- z.z.: $\log_2(n) \in \Theta(\log_8(n))$
- $n = 8^{\log_8 n} = (2^3)^{\log_8(n)} = 2^{3\log_8(n)}$, also gilt für alle $n \geq 1$:
  $\log_2(n) = 3\log_8(n)$ und $\log_8(n) = \frac{1}{3}\log_2(n)$
- allgemein:
  $\log_b(n) \in \Theta(\log_a(n))$, denn

$$b^{\log_b(n)} = n = a^{\log_a(n)} = (b^{\log_b(a)})^{\log_a(n)} = b^{\log_b(a)\cdot\log_a(n)}$$

$\log_b(n) = \log_b(a) \cdot \log_a(n)$
also für $c' = c = \log_b(a)$ und alle $n \geq 1$ gilt:
$c\log_a(n) \leq \log_b(n) \leq c'\log_a(n)$

Logarithmen

Achtung!

Logarithmen haben alle asymptotisch das gleiche Wachstum:

- z.z.: $\log_2(n) \in \Theta(\log_8(n))$
- $n = 8^{\log_8 n} = (2^3)^{\log_8(n)} = 2^{3\log_8(n)}$, also gilt für alle $n \geq 1$:
  $\log_2(n) = 3\log_8(n)$ und $\log_8(n) = \frac{1}{3}\log_2(n)$
- allgemein:
  $\log_b(n) \in \Theta(\log_a(n))$, denn

$$b^{\log_b(n)} = n = a^{\log_a(n)} = (b^{\log_b(a)})^{\log_a(n)} = b^{\log_b(a)\cdot\log_a(n)}$$

  $\log_b(n) = \log_b(a) \cdot \log_a(n)$
  also für $c' = c = \log_b(a)$ und alle $n \geq 1$ gilt:
  $c\log_a(n) \leq \log_b(n) \leq c'\log_a(n)$
- Man kann also einfach $\Theta(\log n)$ ohne Basis schreiben

Berechnung von Potenzen

```
Lese Zahl x ein;
Lese Zahl n ein;
Setze ergebnis = 1;
Setze i = 1;
Solange i<n {
    ergebnis = ergebnis · x
    Erhöhe i um 1
}
Gebe Zahl ergebnis aus;
```

Berechnung von Potenzen

```
Lese Zahl x ein;                    O(1)
Lese Zahl n ein;
Setze ergebnis = 1;
Setze i = 1;
Solange i<n {
    ergebnis = ergebnis · x
    Erhöhe i um 1
}
Gebe Zahl ergebnis aus;
```

Berechnung von Potenzen

```
Lese Zahl x ein;                    O(1)
Lese Zahl n ein;                    O(1)
Setze ergebnis = 1;
Setze i = 1;
Solange i<n {
    ergebnis = ergebnis · x
    Erhöhe i um 1
}
Gebe Zahl ergebnis aus;
```

Berechnung von Potenzen

```
Lese Zahl x ein;                    O(1)
Lese Zahl n ein;                    O(1)
Setze ergebnis = 1;                 O(1)
Setze i = 1;
Solange i<n {
    ergebnis = ergebnis · x
    Erhöhe i um 1
}
Gebe Zahl ergebnis aus;
```

Berechnung von Potenzen

```
Lese Zahl x ein;                    O(1)
Lese Zahl n ein;                    O(1)
Setze ergebnis = 1;                 O(1)
Setze i = 1;                        O(1)
Solange i<n {
    ergebnis = ergebnis · x
    Erhöhe i um 1
}
Gebe Zahl ergebnis aus;
```

Berechnung von Potenzen

```
Lese Zahl x ein;                    O(1)
Lese Zahl n ein;                    O(1)
Setze ergebnis = 1;                 O(1)
Setze i = 1;                        O(1)    i...(n-1)
Solange i<n {
    ergebnis = ergebnis · x
    Erhöhe i um 1
}
Gebe Zahl ergebnis aus;
```

Berechnung von Potenzen

```
Lese Zahl x ein;                    O(1)
Lese Zahl n ein;                    O(1)
Setze ergebnis = 1;                 O(1)
Setze i = 1;                        O(1)     i...(n-1)
Solange i<n {
    ergebnis = ergebnis · x                  O(1)
    Erhöhe i um 1                            O(1)
}
Gebe Zahl ergebnis aus;
```

Berechnung von Potenzen

```
Lese Zahl x ein;                    O(1)
Lese Zahl n ein;                    O(1)
Setze ergebnis = 1;                 O(1)
Setze i = 1;                        O(1)      i...(n-1)
Solange i<n {
    ergebnis = ergebnis · x                   O(1)
    Erhöhe i um 1                             O(1)
}
Gebe Zahl ergebnis aus;             O(1)
```

Berechnung von Potenzen

```
Lese Zahl x ein;                    O(1)
Lese Zahl n ein;                    O(1)
Setze ergebnis = 1;                 O(1)
Setze i = 1;                        O(1)      i...(n-1)
Solange i<n {
    ergebnis = ergebnis · x                   O(1)
    Erhöhe i um 1                             O(1)
}
Gebe Zahl ergebnis aus;             O(1)
```

$$O(1+1+1+1+\sum_{i=0}^{n-1} 2+1)$$

$$= O(n)$$

Suchen in einer aufsteigend sortierten Liste

```
Lese gesuchtes Element x ein;
Setze Suchbereich s auf gesamte Liste;
Solange (AnzahlElemente(s) > 0) {
    Wähle Mitte des Suchbereichs als Pivotelement p;
    Falls (p == x) gib WAHR zurück;
    Falls (p > x) Setze s auf linkeHälfte(s);
    Falls (p < x) Setze s auf rechteHälfte(s);
}
Schritt 4: Gib FALSCH zurück;
```

Hinweis: Operationen wie Suchbereich setzen, Pivotelement wählen, Anzahl der Elemente bestimmen verursachen einen Aufwand von $O(1)$. Nehmen Sie an, dass die Liste n Elemente enthält.

Suchen in einer aufsteigend sortierten Liste

```
Lese gesuchtes Element x ein;                                    O(1)
Setze Suchbereich s auf gesamte Liste;
Solange (AnzahlElemente(s) > 0) {
    Wähle Mitte des Suchbereichs als Pivotelement p;
    Falls (p == x) gib WAHR zurück;
    Falls (p > x) Setze s auf linkeHälfte(s);
    Falls (p < x) Setze s auf rechteHälfte(s);
}
Schritt 4: Gib FALSCH zurück;
```

Suchen in einer aufsteigend sortierten Liste

```
Lese gesuchtes Element x ein;                                    O(1)
Setze Suchbereich s auf gesamte Liste;                           O(1)
Solange (AnzahlElemente(s) > 0) {
    Wähle Mitte des Suchbereichs als Pivotelement p;
    Falls (p == x) gib WAHR zurück;
    Falls (p > x) Setze s auf linkeHälfte(s);
    Falls (p < x) Setze s auf rechteHälfte(s);
}
Schritt 4: Gib FALSCH zurück;
```

Suchen in einer aufsteigend sortierten Liste

```
Lese gesuchtes Element x ein;                                  O(1)
Setze Suchbereich s auf gesamte Liste;                         O(1)    i...log(n)
Solange (AnzahlElemente(s) > 0) {
    Wähle Mitte des Suchbereichs als Pivotelement p;
    Falls (p == x) gib WAHR zurück;
    Falls (p > x) Setze s auf linkeHälfte(s);
    Falls (p < x) Setze s auf rechteHälfte(s);
}
Schritt 4: Gib FALSCH zurück;
```

Suchen in einer aufsteigend sortierten Liste

```
Lese gesuchtes Element x ein;                                  O(1)
Setze Suchbereich s auf gesamte Liste;                         O(1)     i...log(n)
Solange (AnzahlElemente(s) > 0) {
    Wähle Mitte des Suchbereichs als Pivotelement p;                    O(1)
    Falls (p == x) gib WAHR zurück;                                     O(1)
    Falls (p > x) Setze s auf linkeHälfte(s);                           O(1)
    Falls (p < x) Setze s auf rechteHälfte(s);                          O(1)
}
Schritt 4: Gib FALSCH zurück;
```

Suchen in einer aufsteigend sortierten Liste

```
Lese gesuchtes Element x ein;                                  O(1)
Setze Suchbereich s auf gesamte Liste;                         O(1)     i...log(n)
Solange (AnzahlElemente(s) > 0) {
    Wähle Mitte des Suchbereichs als Pivotelement p;                    O(1)
    Falls (p == x) gib WAHR zurück;                                     O(1)
    Falls (p > x) Setze s auf linkeHälfte(s);                           O(1)
    Falls (p < x) Setze s auf rechteHälfte(s);                          O(1)
}
Schritt 4: Gib FALSCH zurück;                                  O(1)
```

Suchen in einer aufsteigend sortierten Liste

```
Lese gesuchtes Element x ein;                                  O(1)
Setze Suchbereich s auf gesamte Liste;                         O(1)     i...log(n)
Solange (AnzahlElemente(s) > 0) {
    Wähle Mitte des Suchbereichs als Pivotelement p;                    O(1)
    Falls (p == x) gib WAHR zurück;                                     O(1)
    Falls (p > x) Setze s auf linkeHälfte(s);                           O(1)
    Falls (p < x) Setze s auf rechteHälfte(s);                          O(1)
}
Schritt 4: Gib FALSCH zurück;                                  O(1)
```

$$O(1) + O(1) + O(log(n)) + O(1)$$

$$= O(log(n))$$

Beispiel

Warum gilt...?

$$O(f(n)) \cdot O(g(n)) = O(f(n) \cdot g(n))$$

$$\begin{aligned}
& O(f(n)) \cdot O(g(n)) \\
\Rightarrow\ & (\forall h_1(n) \in O(f(n)) \exists c_1, n_0 \in \mathbb{N}\, \forall n > n_0 : 0 \leq h_1(n) \leq c_1 \cdot f(n)) \wedge \\
& (\forall h_2(n) \in O(g(n)) \exists c_2, n_0 \in \mathbb{N}\, \forall n > n_0 : 0 \leq h_2(n) \leq c_2 \cdot g(n)) \\
\Rightarrow\ & \forall h_1(n) \cdot h_2(n) \exists c_3 = c_1 \cdot c_2, n_0 \in \mathbb{N}\, \forall n > n_0 : \\
& 0 \leq h_1(n) \cdot h_2(n) \leq c_3 \cdot f(n) \cdot g(n) \\
=\ & O(f(n) \cdot g(n))
\end{aligned}$$

Wunschbeispiel

Warum gilt...?

$n^2 + n \in O(n^2)$

$$n^2 + n \in O(n^2)$$

$$\Leftrightarrow \exists c \in \mathbb{R}^+, n_0 \in \mathbb{N} \, \forall n > n_0 : 0 \leq n^2 + n \leq c \cdot n^2$$

Vermutung: $c = 2, n_0 = 2$

Induktionsanfang: $n = 2 : 0 \leq n^2 + n = 2^2 + 2 = 6 \leq 8 = 2 \cdot 2^2 = 2n^2$

Induktionsvoraussetzung: $0 \leq n^2 + n \leq 2n^2$

Induktionsschluss $n \to n + 1$:

$0 \leq (n+1)^2 + (n+1) = n^2 + 3n + 2 \leq 2n^2 + 4n + 2 =$
$2(n+1)^2$ qed.

Aufgaben

- Welche Funktionen gehören in welche Klassen?

	$O(n^2)$	$\Omega(n^2)$	$O(\log n)$	$\Theta(n)$
$n^2 + n$				
$n \cdot \log(n)$				
$2 \cdot n + 1$				
n^3				
5				

Aufgaben

- Welche Funktionen gehören in welche Klassen?

	$O(n^2)$	$\Omega(n^2)$	$O(\log n)$	$\Theta(n)$
$n^2 + n$	j	j	n	n
$n \cdot \log(n)$	j	n	n	n
$2 \cdot n + 1$	j	n	n	j
n^3	n	j	n	n
5	j	n	j	n

Übersicht

Zum Schluss...

Was ihr nun wissen solltet!

Zum Schluss...

Karlsruhe Institute of Technology

Was ihr nun wissen solltet!

- Wie sind die O, Ω und Θ Kalküle definiert?

Zum Schluss...

Karlsruhe Institute of Technology

Was ihr nun wissen solltet!

- Wie sind die O, Ω und Θ Kalküle definiert?
- Wozu braucht man sie überhaupt?

Zum Schluss...

Karlsruhe Institute of Technology

Was ihr nun wissen solltet!

- Wie sind die O, Ω und Θ Kalküle definiert?
- Wozu braucht man sie überhaupt?
- Welche Schreibweise sollten wir verwenden?

Zum Schluss...

Karlsruhe Institute of Technology

Was ihr nun wissen solltet!

- Wie sind die O, Ω und Θ Kalküle definiert?
- Wozu braucht man sie überhaupt?
- Welche Schreibweise sollten wir verwenden?
- Was ist eine geschlossene Formel zu einer Rekursion?

Zum Schluss...

Karlsruhe Institute of Technology

Was ihr nun wissen solltet!

- Wie sind die O, Ω und Θ Kalküle definiert?
- Wozu braucht man sie überhaupt?
- Welche Schreibweise sollten wir verwenden?
- Was ist eine geschlossene Formel zu einer Rekursion?
- Was ist eine monoton steigende Funktion?

Zum Schluss...

Karlsruhe Institute of Technology

Was ihr nun wissen solltet!

- Wie sind die O, Ω und Θ Kalküle definiert?
- Wozu braucht man sie überhaupt?
- Welche Schreibweise sollten wir verwenden?
- Was ist eine geschlossene Formel zu einer Rekursion?
- Was ist eine monoton steigende Funktion?

Ihr wisst was nicht?

Stellt **jetzt** Fragen!

Ende


DINGE, DIE MAN NICHT AN DER TÜR VERKAUFEN KANN.
EINEN WUNDERSCHÖNEN GUTEN TAG, MEIN HERR. HÄTTEN SIE WOMÖGLICH INTERESSE AN EINER...
OH, ICH SEHE, SIE HABEN BEREITS EINE.
WWW.NICHTLUSTIG.DE