Grundlagen der Programmierung

Vom Problem zum Algorithmus:

Modelle ♦ Graphen ♦ Brute Force

Algorithmisches Denken: Vom Problem zur Lösung

1. Identifizieren des Problems
2. Formulieren des Problems
3. Entwurf des Algorithmus
4. **Implementierung des Algorithmus**
5. **Anwendung des Algorithmus**

Vom Problem zum Algorithmus

→ Problemlösung

Beispiele

- **größter gemeinsamer Teiler (ggT):**
  Welche Zahl ist ggT von zwei natürlichen Zahlen?

- **größtes Listenelement:**
  Welches ist die größte Zahl in einer Liste ganzer Zahlen?

- **Freundschaftsproblem:**
  Wie oft kommt es vor, dass sich Freundschaften als transitive Beziehung erweisen?

Beispiel: größtes Listenelement

1. **Spezifikation** des Algorithmus:
   Name: größtes Listenelement
   Eingabe: Liste *L* ganzer Zahlen
   Ausgabe: größte Zahl in der Liste

2. Liste als Folge von Elementen (Zahlen),
   die durch die Nummer ihrer Position (**Index**) aufgefunden werden: *L*[1], *L*[2], *L*[3], ...
   → **indizierte Liste**

3. Formulieren der algorithmischen Idee z.B. in **Pseudocode**

Beispiel: Freundschaftsproblem

- *Wie oft kommt es vor, dass sich Freundschaften als transitive Beziehungen erweisen?*

- Repräsentation der Beziehungen als **Graph**:
  - **Knoten** („Punkte“) für Individuen
  - **Kanten** (Verbindungen) für Beziehungen

- **Definition.** Ein **ungerichteter Graph** ist ein Paar $G = (V, E)$, wobei V eine endliche Menge von Knoten und E eine Menge von *ungeordneten* Paaren (Zweiermengen) von Knoten ist.

Freunschaftsproblem (1)

- Zwei Knoten u und v heißen **adjazent** gdw. es eine Kante gibt, die u und v verbindet: $\{u,v\} \in E$.
- *Erster Versuch*
  Name: Freundschaftsproblem
  Eingabe: ungerichteter Graph $G=(V,E)$
  Ausgabe: Anzahl transitiver Adjazenzbeziehungen

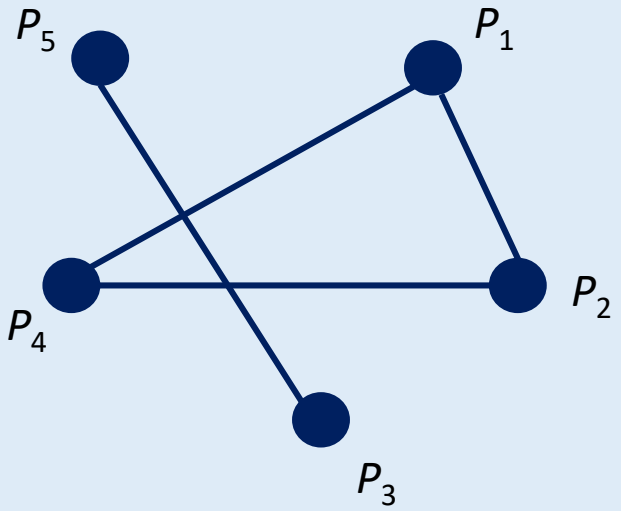
```
x ⟵ 0
für alle i  in  V
|       für alle j adjazent mit i
|       |       für alle k ≠ i adjazent mit j
|       |       |       falls k adjazent mit i
|       |       |       |       x ⟵ x + 1
gib x aus
```

Zählen wir nicht dieselben Beziehungen mehrfach?

($\{u,v\} = \{v,u\}$)

Repräsentation von Graphen

1. **Adjazenzlisten-Repräsentation**
   Für jeden Knoten *u* eine Liste **adj[*u*]** der zu *u* adjazenten Knoten



u	adj[*u*]
P_1	$[P_2,P_4]$
P_2	$[P_1,P_4]$
P_3	$[P_5]$
P_4	$[P_1,P_2]$
P_5	$[P_3]$

Repräsentation von Graphen

1. **Adjazenzlisten-Repräsentation**
   Für jeden Knoten *u* eine Liste **adj[*u*]** der zu *u* adjazenten Knoten

2. **Adjazenzmatrix-Repräsentation**
   Matrix A_G (Rechteckschema) von $n \times n$ Zahlen,
   wobei n die Anzahl der Knoten ist ($n = |V|$),

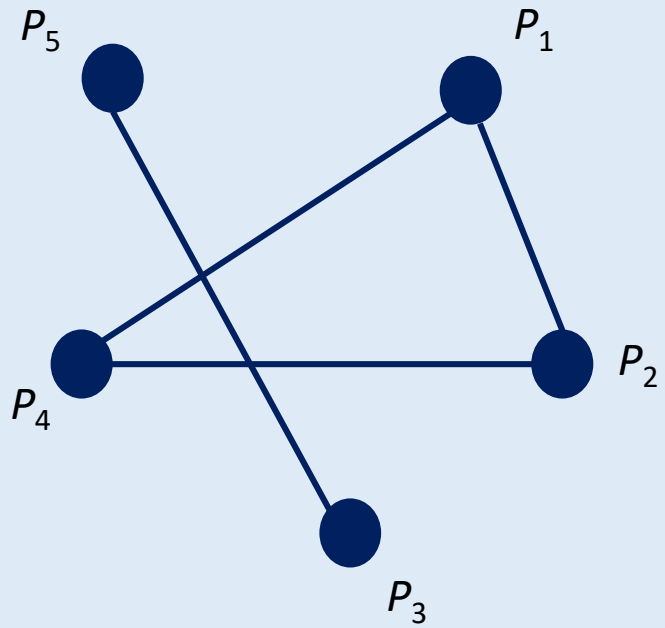
$$\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix} \quad \text{mit} \quad a_{ij} = \begin{cases} 0 & \text{falls } \{u_i, u_j\} \notin E \\ 1 & \text{falls } \{u_i, u_j\} \in E \end{cases}$$

   (*darstellbar als Liste von n Listen der Länge n*)

Repräsentation von Graphen

1.

2.



	P_1	P_2	P_3	P_4	P_5
P_1	0	1	0	1	0
P_2	1	0	0	1	0
P_3	0	0	0	0	1
P_4	1	1	0	0	0
P_5	0	0	1	0	0

(*darstellbar als Liste von n Listen der Länge n*)

Freundschaftsproblem (2)

Name: Freundschaftsproblem
Eingabe: Adjazenzmatrix A eines ungerichteten Graphen $G = (V,E)$
Ausgabe: Anzahl transitiver Adjazenzbeziehungen

$x \longleftarrow 0$
für $i \longleftarrow 1$ **bis** $|V|$
| **für** $j \longleftarrow i + 1$ **bis** $|V|$
| | **für** $k \longleftarrow j + 1$ **bis** $|V|$
| | | **falls** $a_{ij} = 1$ **UND** $a_{jk} = 1$ **UND** $a_{ik} = 1$
| | | | $x \longleftarrow x + 1$
gib x **aus**

Modelle beim Algorithmenentwurf

- Modelle repräsentieren einen **Realitätsausschnitt**.
  - *soziale Netzwerke*
- Beschränkung auf relevante **Aspekte** der Realität
  - *Freundschaftsbeziehungen*
- **Abstraktion**
  - *Individuen als Knoten, Beziehungen als Kanten*
- Modelle dienen immer einem **Zweck**.
  - *Untersuchung des Freundschaftsproblems*

Abstraktion

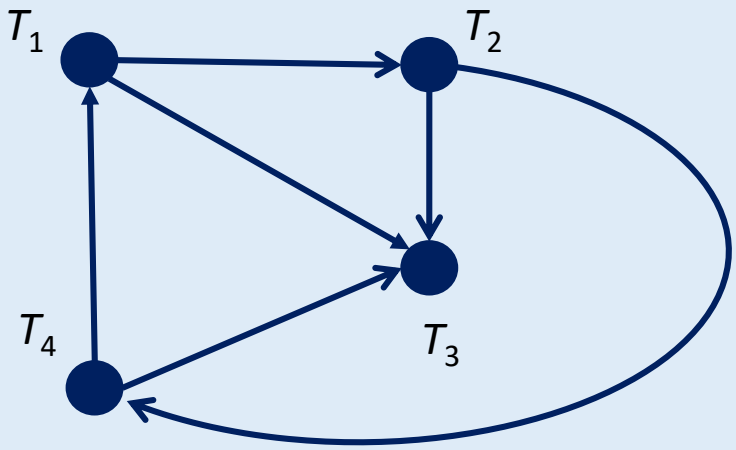
- Abbildung, meist viele-zu-eins (Klassenbildung)
- *Beispiele:*
  - Mengen von Zahlen *i* mit $a \leq i \leq b$ auf Intervall [*a*,*b*]
  - Geschwindigkeiten (Vektoren) auf ihren Betrag (in m/s)
  - Individuum auf Knoten eines Graphen
- entspricht Begriffsbildung (z.B. „Tier“)
- Bild (Klasse/Begriff) wird im Modell stellvertretend für die Originale verwendet

Modelle beim Algorithmenentwurf

- Abstraktion erlaubt oft vielfache Verwendung.
  - Freundschaften transitiv?
  - andere Transitivitätsprobleme
  - Kleine-Welt-Problem
  - Routenplanung (z.B. kürzeste Flugverbindungen oder Internet-Routing, ...)

- Reichen in allen Fällen ungerichtete Graphen?

Routenplanung und gerichtete Graphen



$V = \{T_1, T_2, T_3, T_4\}$

$E = \{(T_1,T_2), (T_1,T_3),$
$(T_2,T_3), (T_2,T_4),$
$(T_4,T_1), (T_4,T_3)\}$

Paare

transitiv?

$T_1 - T_2 - T_3$ *vs.* $T_1 - T_2 - T_4$

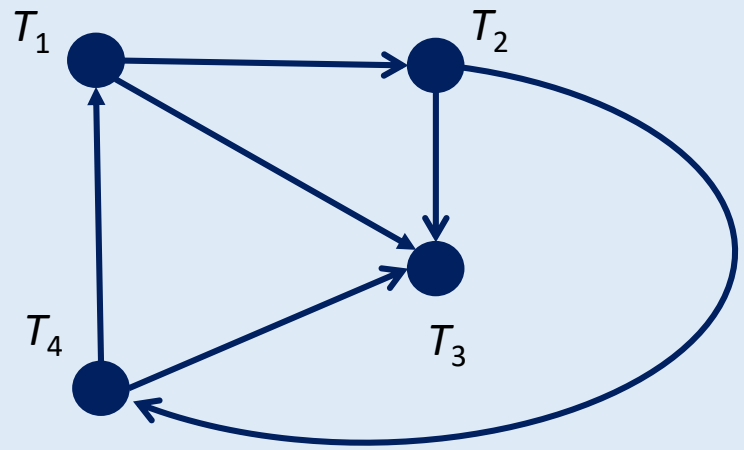
- **Definition.** Ein **gerichteter Graph** ist ein Paar $G = (V,E)$, wobei V eine endliche Menge von Knoten und E eine Menge von geordneten Knotenpaaren ist: $E \subseteq V \times V$.

Graph $G = (V,E)$: Grundbegriffe (1)

- *Adjazenzlisten und -matrix entsprechend anpassen!*
  - v in adj[u] gdw. $(u,v) \in E$
  - $a_{ij} = 1$ gdw. $(u_i,u_j) \in E$
- Für einen Knoten u ist
  - $d(u) = |\{ \{u,v\} \in E : v \in V \}|$ sein **Grad** (ungerichtet),
  - $d^+(u) = |\{ (u,v) \in E : v \in V \}|$ sein **Ausgangsgrad**,
  - $d^-(u) = |\{ (v,u) \in E : v \in V \}|$ sein **Eingangsgrad**.
- Kante (u,v) (bzw. $\{u,v\}$) heißt **Schlinge** gdw. $u = v$.
- $G = (V,E)$ heißt **schlingenfrei**, falls keine Kante eine Schlinge ist.

Graph $G = (V,E)$: Grundbegriffe (1)

- *Adjazenzlisten und -matrix entsprechend anpassen!*
  - v in adj[u] gdw. $(u,v) \in E$
  - $a_{ij} = 1$ gdw. $(u_i,u_j) \in E$
- E
-
-



	T_1	T_2	T_3	T_4	
T_1	0	1	1	0	$d^+(T_1) = 2$
T_2	0	0	1	1	$d^+(T_2) = 2$
T_3	0	0	0	0	$d^+(T_3) = 0$
T_4	1	0	1	0	$d^+(T_4) = 2$
$d^-(T_i)$:	1	1	3	1	

Graph $G = (V,E)$: Grundbegriffe (2)

- Seien u und v Knoten. Ein **Pfad** von u nach v ist eine Folge $v_0, v_1, ..., v_k$ von Knoten, wobei $v_0 = u$, $v_k = v$ und für alle $0 \leq i < k$ gilt, dass $(v_i , v_{i+1}) \in E$ (bzw. $\{v_i , v_{i+1}\} \in E$) .

- Die Zahl ***k*** heißt **Länge** des Pfades.

- Der Pfad ist ein **Zyklus**, falls $v_0 = v_k$ gilt.

- Ein Zyklus ist ein **Kreis**, falls außer $v_0 = v_k$ kein Knoten in der Folge mehrfach auftritt, also falls $v_i \neq v_j$ für alle $i \neq j$ mit $1 \leq i \leq k-1$ und $0 \leq j \leq k$.

Graph $G = (V,E)$: Grundbegriffe (3)

- Der Graph G heißt **(stark) zusammenhängend**, falls für jedes Paar von Knoten (u,v) ein Pfad von u nach v existiert.

- Ist G gerichtet, dann heißt G **schwach zusammenhängend**, falls der ungerichtete Graph $G' = (V, \{ \{u,v\} : (u,v) \in E \})$ zusammenhängend ist .

- Seien u und v Knoten. Der **Abstand $D(u,v)$** von u nach v ist die Länge des kürzesten Pfades von u nach v.
  Falls kein Pfad existiert, ist der Abstand ∞ .

  → *Kleine-Welt-Problem*

Kleine-Welt-Problem

Eingabe: Freudschaftsbeziehungen in einem sozialen Netzwerk, repräsentiert als ungerichteter, schlingenfreier Graph $G = (V,E)$

Offenbart G das Kleine-Welt-Phänomen?

→ *Welche Abstände von Knotenpaaren treten wie häufig auf?*

→ Algorithmus zur Berechnung des Abstands zweier Knoten **?!**

Abstand von Knoten

1. Existiert Pfad der Länge 1 (also eine Kante)? ... Sonst:
2. Existiert ein Pfad der Länge 2? ... Sonst:

⋮

→ **Brute Force Algorithmus**
(probiert systematisch alle Möglichkeiten durch)

Terminiert? (Was, wenn *G* nicht zusammenhängend ist?)

Theorem (Maximaler Abstand): *Für alle Knoten u, v gilt:*
Wenn ein Pfad von u nach v existiert, dann gilt $D(u,v) \leq |V| - 1$.

Abstand von Knoten

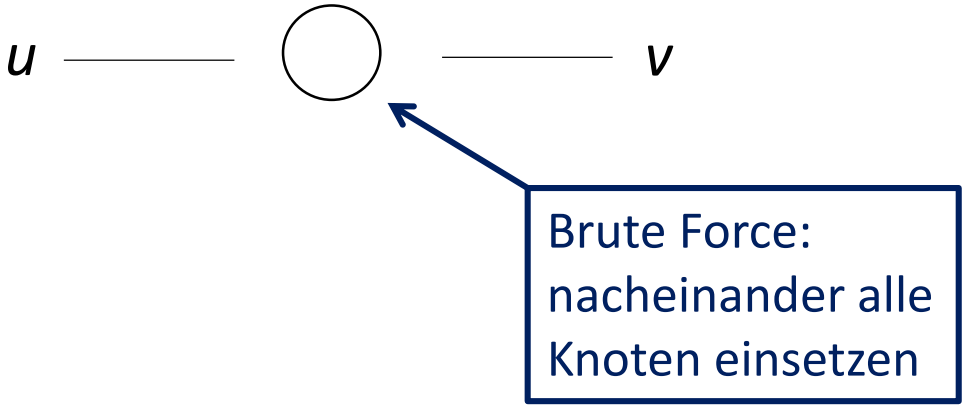
Name: Abstand von Knoten (Brute Force)
Eingabe: ungerichteter, schlingenfreier Graph $G = (V,E)$,
$u, v \in V, u \neq v$
Ausgabe: $D(u,v)$

für $k \longleftarrow 1$ **bis** $|V| - 1$
falls Pfad der Länge k von u nach v existiert
gib k **aus**
STOP

→ *benötigen Algorithmus, der für zwei Knoten u und v und eine positive ganze Zahl k feststellt, ob ein Pfad der Länge k von u nach v existiert*

Algorithmische Idee (Brute Force)

1. Idee für $k = 1$: $\{u,v\} \in E$?
2. Idee für $k = 2$:



3. Idee für beliebiges k:
   Probieren aller Teilmengen von V der Größe $k - 1$
   und aller Anordnungen von deren Elementen

Abstand: Brute Force Algorithmus

```
k ⟵ 1
solange k < |V|
|       u_0 ⟵ u
|       u_k ⟵ v
|       für jede Teilmenge V' ⊆ V mit k – 1 Elementen
|       |       für jede Permutation u_1,u_2, ..., u_{k-1} ihrer Elemente
|       |       |       istPfad ⟵ 1
|       |       |       für j ⟵ 0 bis k – 1
|       |       |       |       falls {u_j,u_{j+1}} ∉ E
|       |       |       |       |       istPfad ⟵ 0
|       |       |       falls istPfad = 1
|       |       |       |       gib k aus
|       |       |       |       STOP
|       k ⟵ k + 1
gib ∞ aus
```

Brute Force Algorithmen

- oft erste Idee,
  direkt an Definition des Modells orientiert
- systematisches Durchprobieren aller Möglichkeiten
- einfach zu beschreiben
- *aber oft sehr ineffizient* (später in diesem Kurs!)

Kleine-Welt-Problem

Eingabe: Freudschaftsbeziehungen in einem sozialen Netzwerk, repräsentiert als ungerichteter, schlingenfreier Graph $G = (V,E)$

Offenbart G das Kleine-Welt-Phänomen?

→ *Welche Abstände von Knotenpaaren treten wie häufig auf?*

→ Algorithmus zur Berechnung des Abstands zweier Knoten ✓

Kleine-Welt-Problem

Name: Abstandsverteilung
Eingabe: ungerichteter, schlingenfreier Graph $G = (V,E)$
Ausgabe: Liste mit Häufigkeiten von Knotenabständen in G

```
für D ⟵ 1 bis |V|
|       H[D] ⟵ 0
für i ⟵ 1 bis |V| - 1
|       für j ⟵ i + 1 bis |V|
|       |       D ⟵ Abstand der Knoten u_i und u_j
|       |       falls D = ∞
|       |       |       H[|V|] ⟵ H[|V|] + 1
|       |       sonst
|       |       |       H[D] ⟵ H[D] + 1
gib H aus
```