DSAL - 8. Globalübung

Tim Quatmann

19. Juni 2018

Agenda

1. Tiefen- und Breitensuche
2. Anwendung Breitensuche: k-begrenzte Erreichbarkeit
3. Kritische-Pfad-Analyse
4. Bipartite Graphen

Tiefen- und Breitensuche

Tiefensuche: Implementierung

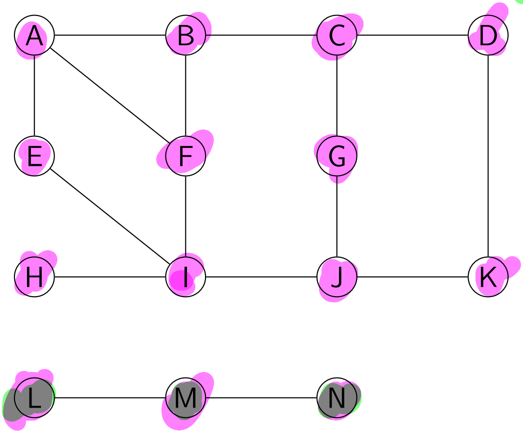
```
void DFS(List adj[n], int start, int &color[n]) {
  color[start] = GRAY; // start ist noch zu verarbeiten
  foreach (w in adj[start]) {
    if (color[w] == WHITE) { // neuer ("ungefundener") Knoten
      DFS(adj, w, color);
    }
  }
  color[start] = BLACK; // start ist abgeschlossen
}

void completeDFS(List adj[n], int n, int start) {
  int color[n] = WHITE; // noch kein Knoten ist gefunden worden
  for (int i = 0; i < n; i++)
    if (color[i] == WHITE) DFS(adj, i, color);
}
```

Untersuche v. Start-Knoten erreichbare Knoten

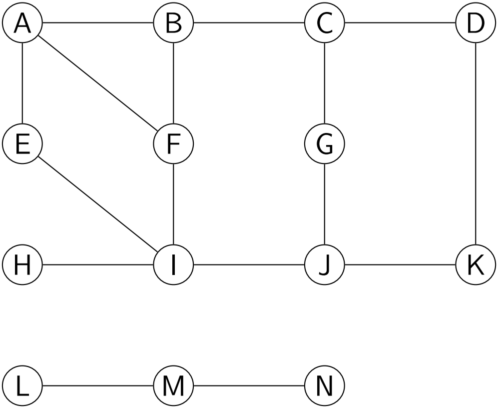
Finde Start-Knoten

Tiefensuche: Beispiel

= Grau Schw.



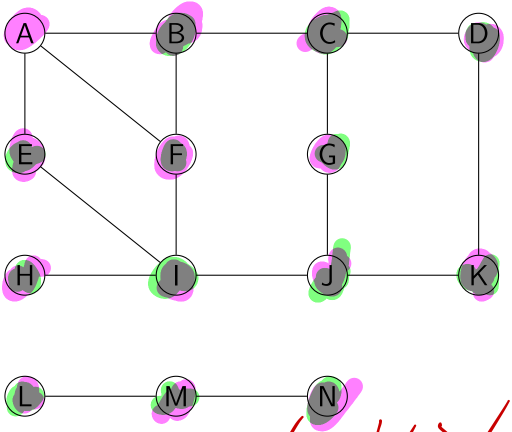
Tiefensuche: Beispiel



Reihenfolge Graufärbung: $A, B, C, D, K, J, G, I, E, F, H, L, M, N$
Reihenfolge Schwarzfärbung: $G, E, F, H, I, J, K, D, C, B, A, N, M, L$

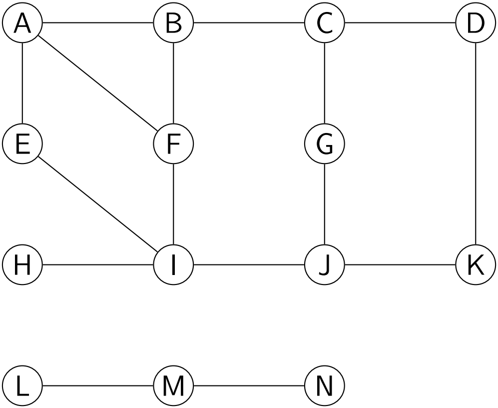
Breitensuche: Implementierung

```
void BFS(List adj[n], int start, int &color[n]) {
  Queue wait; // zu verarbeitende Knoten   FIFO
  color[start] = GRAY; // Knoten start ist noch zu verarbeiten
  wait.enqueue(start);
  while (!wait.isEmpty()) {// es gibt noch unverarbeitete Knoten
    int v = wait.dequeue(); // nächster unverarbeiteter Knoten
    foreach (w in adj[v]) {
      if (color[w] == WHITE) { // neuer ("ungefundener") Knoten
        color[w] = GRAY; // w ist noch zu verarbeiten
        wait.enqueue(w);
      }
    }
    color[v] = BLACK; // v ist abgeschlossen
  }
}

void completeBFS(List adj[n], int n) {
  int color[n] = WHITE; // noch kein Knoten ist gefunden worden
  for (int i = 0; i < n; i++)
    if (color[i] == WHITE) BFS(adj, n, i, color);
}
```



wait: ~~A~~ ~~B~~ ~~E~~, ~~F~~, ~~C~~, ~~I~~, ~~D~~, ~~G~~, ~~H~~, ~~J~~, ~~K~~
~~L~~, ~~M~~, ~~N~~

Breitensuche: Beispiel



Reihenfolge Graufärbung: $A, B, E, F, C, I, D, G, H, J, K, L, M, N$
Reihenfolge Schwarzfärbung: $A, B, E, F, C, I, D, G, H, J, K, L, M, N$

Anwendung Breitensuche: k-begrenzte Erreichbarkeit

Sei $G = (V, E)$ ein (ungewichteter) Graph.

- Ein Pfad von $u \in V$ zu $w \in V$ im Graphen $G = (V, E)$ ist eine Folge $v_0 v_1 \ldots v_{k-1} v_k$ mit $v_i \in V$, $(v_i, v_{i+1}) \in E$, $v_0 = u$, $v_k = w$.
- k ist die Länge des Pfades $v_0 v_1 \ldots v_{k-1} v_k$
- Für zwei Knoten $u, w \in V$ sei $dist(u, w)$ die Länge eines kürzesten Pfades von u zu w, d.h. (bei gew. Graphen ist es anders)

  $$dist(u, w) = \min\{k \in \mathbb{N} \mid \text{Es gibt einen Pfad der Länge } k \text{ von } u \text{ zu } w\}$$

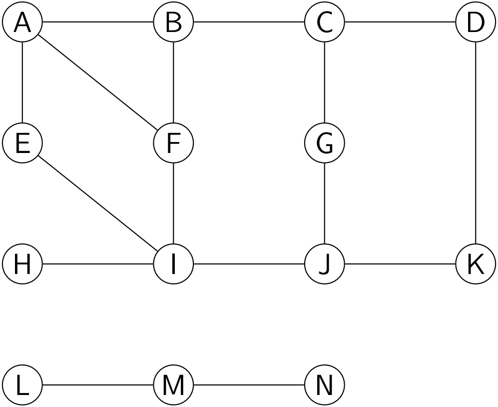
  Außerdem: $dist(u, w) = \infty$, falls w nicht von u erreichbar ist.

$dist(A, J) = 3$

$dist(D, H) = 4$

$dist(H, D) = 4$

$dist(C, M) = \infty$



k-begrenzte Erreichbarkeit

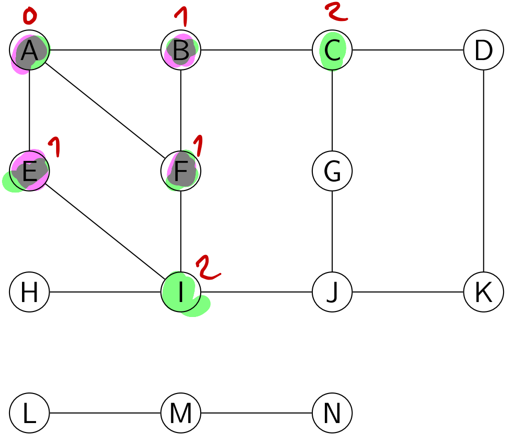
Gegeben: Ein Graph $G = (V, E)$, ein Startknoten $u \in V$, $k \in \mathbb{N}$

Gesucht: Alle Knoten $w \in V$ mit $dist(u, w) \leq k$.

Idee: Nutze Breitensuche!

- Starte bei u
- Merke die aktuelle Suchtiefe
- Stoppe bei Suchtiefe k

Beispiel: Finde alle Knoten $w \in \{A, B, \ldots, N\}$ mit $dist(A, w) \leq 2$:



Ergebnis:

A, B, E, F, I

k-begrenzte Erreichbarkeit: Algorithmus

k-begrenzte Erreichbarkeit: Algorithmus

```
boundedReach(List adj[n], int start, int k) {
    Queue wait; // zu verarbeitende Knoten
    if (k > 0) wait.enqueue(start);
    int dist[n] = -1; // Distanz der gefundenen Knoten
    dist[start] = 0;
    Set result;
    result.insert(start);
    while (!wait.isEmpty()) { // es gibt noch unverarbeitete Knoten
        int v = wait.dequeue();
        foreach (w in adj[v]) {
            if (dist[w] == -1) { // neuer ("ungefundener") Knoten
                result.insert(w);
                dist[w] = dist[v] + 1
                // Untersuche w nur, wenn mit <k Schritten erreichbar
                if (dist[w] < k) wait.enqueue(w);
            }
        }
    }
    return result;
}
```

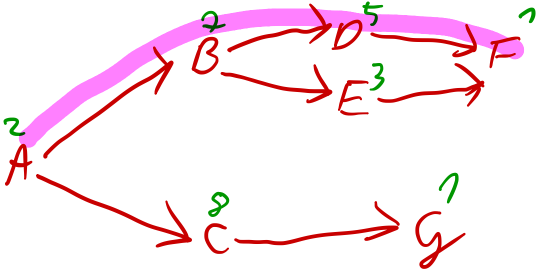
Kritische-Pfad-Analyse

Kritischer-Pfad-Problem

Gegeben: Knotengewichteter DAG $G = (V, E, W)$

Gesucht: Der längste Pfad (bezogen auf das Gesamtgewicht)

Anwendung: Abhängigkeiten von Aufgaben $v \in V$ mit Dauer $W(v)$
$(v, v') \in E \iff$ "Um v zu erledigen muss erst v' erledigt werden"



Kritischer-Pfad-Problem

Gegeben: Knotengewichteter DAG $G = (V, E, W)$

Gesucht: Der längste Pfad (bezogen auf das Gesamtgewicht)

Anwendung: Abhängigkeiten von Aufgaben $v \in V$ mit Dauer $W(v)$
$(v, v') \in E \iff$ "Um v zu erledigen muss erst v' erledigt werden"

Idee: Bestimme *est*/*eft* (earliest start/finish time) für jeden Knoten

$$est(v) = \begin{cases} \max\limits_{(v,v')\in E} eft(v') & \text{, falls } \exists\, (v, v') \in E \\ 0 & \text{, sonst} \end{cases}$$

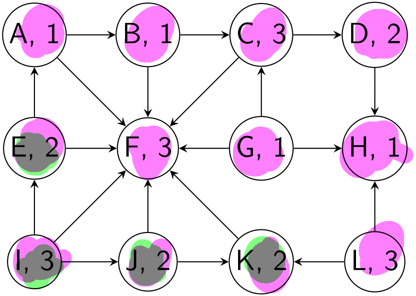
$$eft(v) = est(v) + W(v)$$

Nutze Tiefensuche um die Knoten in "richtigen" Reihenfolge zu bearbeiten.

Kritische-Pfad-Analyse: Implementierung

critical dependency

```
// Knotengewichte in duration.
// Ausgabe: eft, kritischer Pfad kodiert in critDep
void DFS(List adj[n], int start, int &color[n],
               int duration[n], int &critDep[n], int &eft[n]) {
  color[start] = GRAY; critDep[start] = -1; int est = 0;
  foreach (next in adj[start]) {
    if (color[next] == WHITE)
      DFS(adj, next, color, duration, critDep, eft);
    if (eft[next] >= est) {
      est = eft[next]; critDep[start] = next;
    }
  }
  eft[start] = est + duration[start];
  color[start] = BLACK;
}
void critPath(List adj[n], int n,
               int duration[n], int &critDep[n], int &eft[n]){
  int color[n] = WHITE;
  for (int i = 0; i < n; i++)
    if (color[i] == WHITE)
      DFS(adj, i, color, duration, critDep, eft);
}
```



Knoten	A	B	C	D	E	F	G	H	I	J	K	L
est	7	6	3	1	8	0	6	0	10	5	3	5
critDep	B	C	F	H	A	-1	C	-1	E	K	F	K
eft	8	7	6	3	10	3	7	1	13	7	5	8

Gesamtdauer: 13 (max eft)

Kritischer Pfad: I E A B C F

Bipartite Graphen

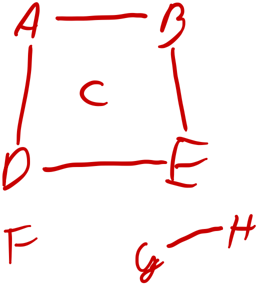
Definition (Bipartiter Graph)

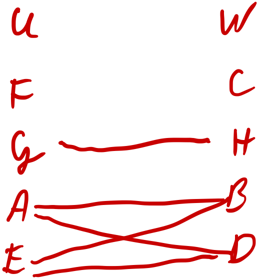
Ein ungerichteter Graph $G = (V, E)$ heißt bipartit, falls es zwei Mengen U und W gibt mit

- $U \cup W = V$
- $U \cap W = \emptyset$

$U \dot\cup W$, $U \uplus W$

- $E \subseteq \big\{\{v, v'\} \mid v \in U \text{ und } v' \in W\big\}$, d.h. es gibt keine Kanten zwischen zwei Knoten in U bzw. zwei Knoten in W.

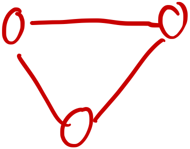




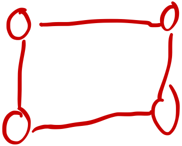
Satz

Ein ungerichteter Graph $G = (V, E)$ ist bipartit genau dann, wenn er keine Zykeln ungerader Länge hat

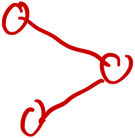
- Die Länge eines Zyklus $v_0 v_1 \dots v_k$ mit $v_0 = v_k$ ist k



nicht bipartit



bipartit



bipartit

"$\Rightarrow$" Sei $G=(V,E)$ bipartit mit $V=U \dot\cup W$

Angenommen G hat einen ungeraden Zykel

$v_0 v_1 \dots v_k$ $(v_0 = v_k,\ v_i \neq v_j$ für $i \neq j)$

OBdA: Sei $v_0 \in U$. Wegen $(v_0, v_1) \in E$ folgt, dass $v_1 \in W$. Daher $v_2 \in U$ ...

Es folgt: $v_i \in U$, falls i gerade
$v_i \in W$, falls i ungerade

Da k ungerade gilt also $v_k \in W$

$\lightning$ Widerspruch, da $v_k = v_0 \in U$

Es folgt, dass G keinen ungeraden Zykel haben kann.

"$\Leftarrow$" G habe keine Zykel ungerader Länge. Wir nehmen an, dass G zusammenhängend ist (ansonsten betrachten wir die Zush. komponenten einzeln).

Sei $u \in V$ ein beliebiger, fest gewählter Knoten. Sei

Länge eines kürzesten Pfades von u nach w

$W = \{v \in V \mid \text{dist}(v,u) \text{ gerade}\}$

$U = \{v \in V \mid \text{dist}(v,u) \text{ ungerade}\}$

Wir zeigen, dass G bipartit mit $U, W \subseteq V$ ist:

- $W \cup U = V$
- $W \cap U = \emptyset$

1. Fall: Falls es eine Kante $(v, v') \in E$ mit $v, v' \in W$ gäbe, betrachte

die Pfade $\underbrace{v_0}_{=v} v_1 \dots \underbrace{v_k}_{=u}$ mit Länge dist(v, u)

$\underbrace{v_0'}_{=v'} v_1' \dots \underbrace{v_{k'}'}_{=u}$ mit Länge dist(v', u)

Sei $l := \min\{m \in \mathbb{N} \mid \exists j \in \{0, \dots, k'\} : v_l = v_j'\}$

(Intuitiv ist v_l der erste Knoten, der auf beiden Pfaden b sucht wird)

Wähle $l' \in \{0, \dots, \}$ mit $v_l = v'_{l'}$

Betrachte den Zykel

$$\underbrace{v_0}_{=v} v_1 \dots \underbrace{v_i}_{=v_i'} v'_{i-1} v'_{i-2} \dots \underbrace{v'_0}_{=v'} v$$

mit Länge $\mathrm{dist}(v_0, v_i) + \mathrm{dist}(v_i', v_0') + 1$

wegen $\mathrm{dist}(v,u) = \mathrm{dist}(v,v_i) + \mathrm{dist}(v_i,u)$

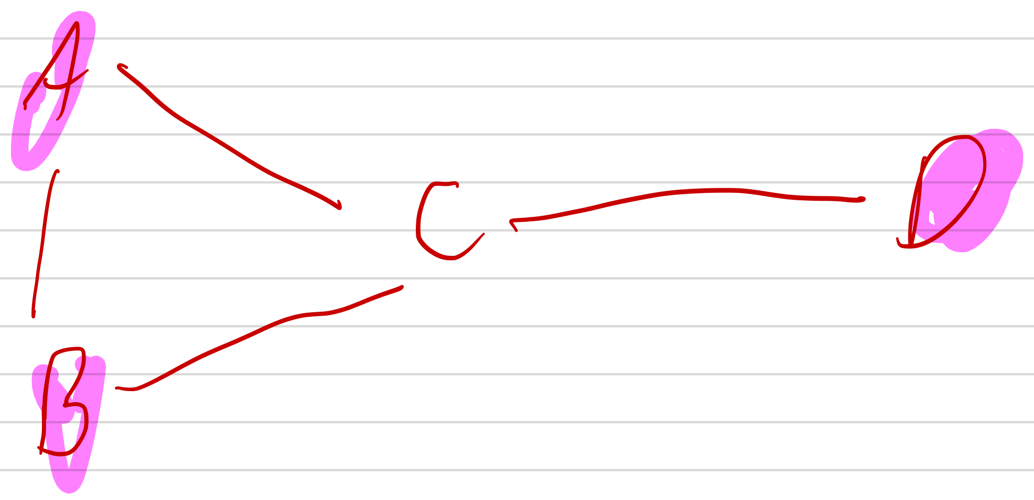
$\mathrm{dist}(v',u) = \mathrm{dist}(v',v_i') + \mathrm{dist}(v_i,u)$ gerade

Gilt: $\mathrm{dist}(v,v_i)$ gerade $\Leftrightarrow$ $\mathrm{dist}(v',v_i')$ gerade

Daher ist $\mathrm{dist}(v_0,v_i) + \mathrm{dist}(v_i',v_0')$ immer gerade. Obiger Zykel hat also ungerade Länge

↯ Widerspruch

2. Fall (Kante zwischen zwei Knoten in W) ist analog



$\mathrm{dist}(A,D) = 2$

$\mathrm{dist}(B,D) = 2$

□

Nächster Termin

Nächste Vorlesung

Freitag, 22. Juni, 13:15 (H01).

Nächste Globalübung

Dienstag, 26. Juni, 14:15 (Aula 1).