

Einführung in die Informatik 2

– Graphenexploration –

Sven Kosub

AG Algorithmik/Theorie komplexer Systeme
Universität Konstanz

E 202 | Sven.Kosub@uni-konstanz.de | Sprechstunde: Freitag, 12:30-14:00 Uhr, o.n.V.

Sommersemester 2009

Suche im Labyrinth

Aufgabe: Durchsuche ein Labyrinth



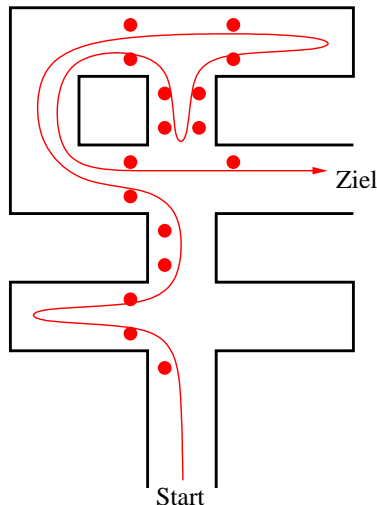
Präzisierung der Aufgabenstellung:

- Jede Kreuzung und Sackgasse soll irgendwann besucht werden
- Gehen von Kreisen soll verhindert werden

Suche im Labyrinth

Lösung:

- Verwende Markierungen an Kreuzungen
- wenn Sackgasse, dann Umdrehen
- wenn Kreuzung, dann beim Betreten Marke setzen und wie folgt verfahren:
 - **keine Kreise:** hat Gang, durch den man gekommen ist, gerade erst Marke bekommen und sind weitere Marken vorhanden, dann zweite Marke in Gang und umkehren
 - **sonst neue Gänge:** wenn Gänge ohne Marken vorhanden, den linken Gang nehmen und Marke setzen
 - **Zurückgehen:** umkehren und den Gang mit nur einer Marke nehmen



- Graphenexploration als Sammlung von Techniken zur „Erforschung“ unbekannter Graphen und Netzen hinsichtlich bestimmter Fragestellungen
- ... mittels Graphtraversierung (in der Vorlesung)
- Graphtraversierung ist das Ablaufen aller Kanten eines Graphen
- Suche im Labyrinth ist Beispiel für Graphtraversierung

Traversierungsarten:

- **Tiefensuche** (engl. *depth-first search*, Abk. *dfs*)
- **Breitensuche** (engl. *breadth-first search*, Abk. *bfs*)

Idee bei der Tiefensuche:

- starte in einem Knoten
- gehe solange wie möglich zu einem benachbarten Knoten, der noch nicht besucht wurde
- falls im aktuellen Knoten alle Nachbarn bereits besucht, dann kehre auf dem gegangenen Weg zurück zum letzten Knoten, der einen noch nicht besuchten Nachbarn hat (**backtracking**)

Tiefensuche produziert zwei Arten von Kanten:

- Baumkanten
- Rückwärtskanten

Algorithmus: DFS(G, v)

Eingabe: (ungerichteter) Graph G , Knoten v in G

Ausgabe: Markierung der Kanten als „Baumkante“ oder „Rückwärtskante“

Markiere Knoten v als „besucht“

for jede mit v inzidente Kante $e = \{u, v\}$

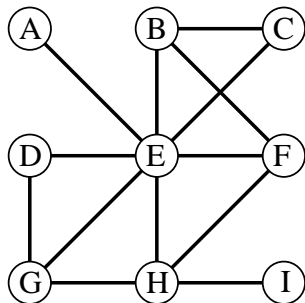
if Gegenknoten u nicht als „besucht“ markiert

 Markiere e als „Baumkante“

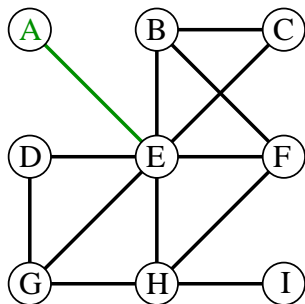
 Markiere u als „besucht“

 DFS(G, u)

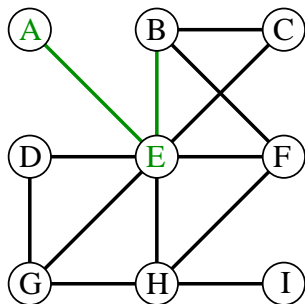
else Markiere e als „Rückwärtskante“



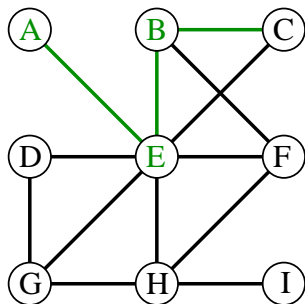
- inzidente Kante nach alphabetischer Sortierung der Nachbarn
- Baumkanten
- Rückwärtskanten



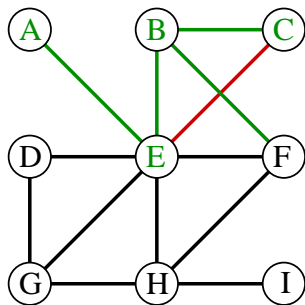
- inzidente Kante nach alphabetischer Sortierung der Nachbarn
- Baumkanten
- Rückwärtskanten



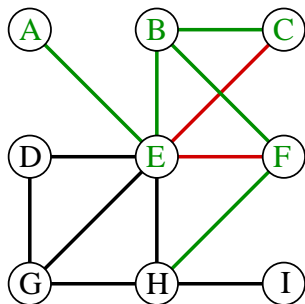
- inzidente Kante nach alphabetischer Sortierung der Nachbarn
- Baumkanten
- Rückwärtskanten



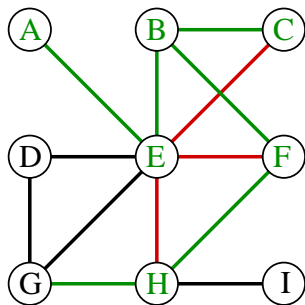
- inzidente Kante nach alphabetischer Sortierung der Nachbarn
- Baumkanten
- Rückwärtskanten



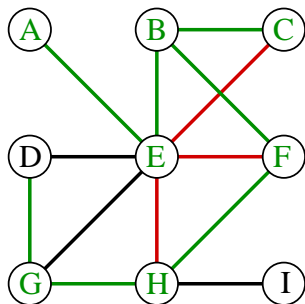
- inzidente Kante nach alphabetischer Sortierung der Nachbarn
- Baumkanten
- Rückwärtskanten



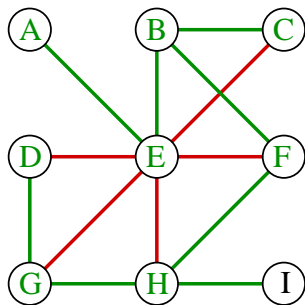
- inzidente Kante nach alphabetischer Sortierung der Nachbarn
- Baumkanten
- Rückwärtskanten



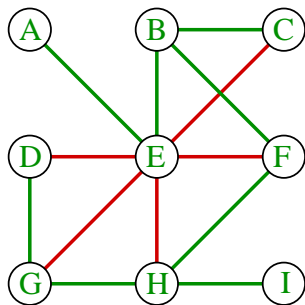
- inzidente Kante nach alphabetischer Sortierung der Nachbarn
- Baumkanten
- Rückwärtskanten



- inzidente Kante nach alphabetischer Sortierung der Nachbarn
- Baumkanten
- Rückwärtskanten

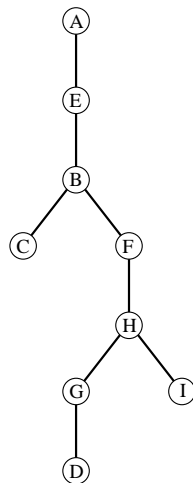


- inzidente Kante nach alphabetischer Sortierung der Nachbarn
- Baumkanten
- Rückwärtskanten



- inzidente Kante nach alphabetischer Sortierung der Nachbarn
- Baumkanten
- Rückwärtskanten

- Baumkanten bilden DFS-Baum
- falls G zusammenhängend, dann ist DFS-Baum ein Spannbaum
- Rückwärtskanten schließen Kreise
- DFS-Baum ändert sich, wenn inzidente Kanten in anderer Reihenfolge behandelt werden



Laufzeit der Tiefensuche:

- mit Adjazenzlisten: $O(n + m)$ (da jede Kante genau zweimal durchlaufen wird)
- mit Adjazenzmatrix: $O(n^2)$ (da jede potenzielle Kante genau zweimal behandelt wird)

(zusätzlicher) Speicherplatz der Tiefensuche:

- proportional zur Höhe des DFS-Baums, d.h. $O(n)$ im schlechtesten Fall

Folgende Probleme sind mit Tiefensuche in Zeit $O(n + m)$ lösbar:

- Test, ob G zusammenhängend ist
- Berechnung eines Spannwaldes von G
- Berechnung eines Pfades zwischen zwei Knoten von G
- Berechnung eines Kreises

Idee bei der Breitensuche:

- starte in einem Knoten
- unterteile iterativ die Knoten in Levels
- ein Level der Ordnung i besteht aus allen Knoten, die Nachbarn im Level $i - 1$ haben
- Startknoten bildet Level 0

Breitensuche produziert zwei Arten von Kanten:

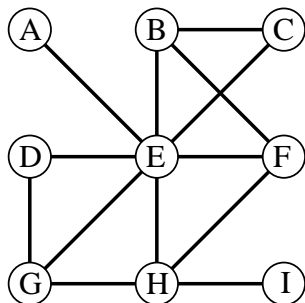
- Baumkanten
- Querkanten

Algorithmus: BFS(G, v)

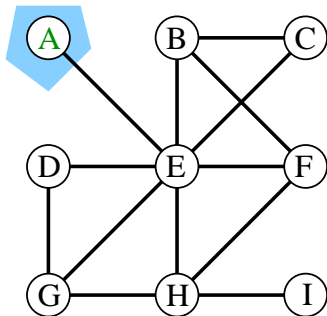
Eingabe: (ungerichteter) Graph G , Knoten v in G

Ausgabe: Markierung der Kanten als „Baumkante“ oder „Querkante“

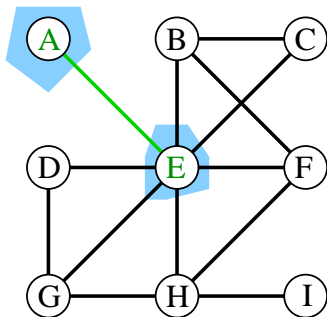
```
i=0
Initialisiere Liste  $L_0$ 
Füge  $v$  in  $L_0$  ein
Markiere  $v$  als „besucht“
while  $L_i$  nicht leer
    Initialisiere Liste  $L_{i+1}$ 
    for jeden Knoten  $v$  in  $L_i$ 
        for jede mit  $v$  inzidente Kante  $e = \{u, v\}$ 
            if Gegenknoten  $u$  nicht als „besucht“ markiert
                Markiere  $e$  als „Baumkante“
                Füge  $u$  in  $L_{i+1}$  ein
                Markiere  $u$  als „besucht“
            else Markiere  $e$  als „Querkante“
    i=i+1
```



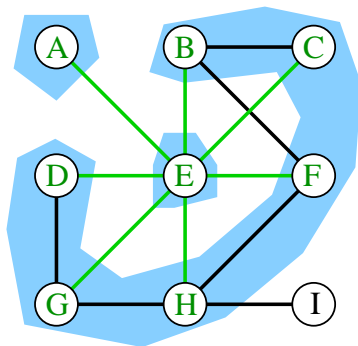
- inzidente Kante nach alphabetischer Sortierung der Nachbarn
- Baumkanten
- Querkanten



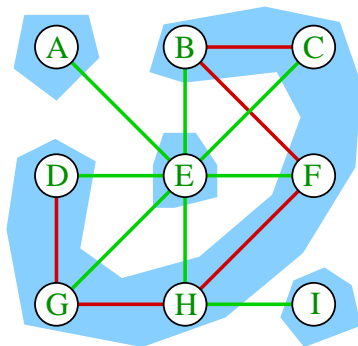
- inzidente Kante nach alphabetischer Sortierung der Nachbarn
- Baumkanten
- Querkanten



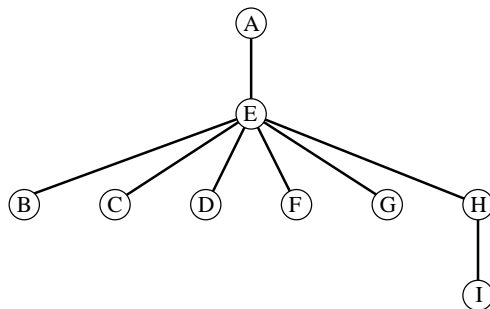
- inzidente Kante nach alphabetischer Sortierung der Nachbarn
- Baumkanten
- Querkanten



- inzidente Kante nach alphabetischer Sortierung der Nachbarn
- Baumkanten
- Querkanten



- inzidente Kante nach alphabetischer Sortierung der Nachbarn
- Baumkanten
- Querkanten



- Baumkanten bilden BFS-Baum
- falls G zusammenhängend, dann ist BFS-Baum ein Spannbaum
- Querkanten schließen Kreise
- Knotentiefen im BFS-Baum entsprechende kürzesten Pfade der Knoten zu A

Laufzeit der Breitensuche:

- mit Adjazenzlisten: $O(n + m)$ (da jede Kante genau zweimal durchlaufen wird)
- mit Adjazenzmatrix: $O(n^2)$ (da jede potenzielle Kante genau zweimal behandelt wird)

(zusätzlicher) Speicherplatz der Breitensuche:

- $O(n)$ im schlechtesten Fall

Folgende Probleme sind mit Breitensuche in Zeit $O(n + m)$ lösbar:

- Test, ob G zusammenhängend ist
- Berechnung eines Spannwaldes von G
- Berechnung eines kürzesten Pfades zwischen zwei Knoten von G
- Berechnung eines Kreises