Wiederholung

- Spannbaum
  - Algorithmisch: entferne Kreiskanten
  - Anzahl markierter Spannbäume, Prüferkode
- Darstellung von Graphen
  - Adjazenzmatrix
  - Adjazenzliste
- Datenstruktur Warteschlange
- BFS

Breitensuche (Breadth First Search)

Algorithmus BFS

Eingabe: G=(V,E) in Adjazenzlistendarstellung mit Startknoten s $\in$ V

1. Für alle v $\in$ V
   1. if (v=s) then d[v] $\leftarrow$ 0 else d[v] $\leftarrow \infty$
   2. pred(v) $\leftarrow$ nil;
2. Q $\leftarrow$ new Queue;
3. Q.Enqueue(s);
4. while (Q.Isempty $\neq$ TRUE)
   1. v $\leftarrow$ Q.Dequeue(Q).
   2. Für alle u $\in \Gamma$(v)
      1. if (d[u]=∞) then
         1. d[u] $\leftarrow$ d[v]+1; pred[u] $\leftarrow$ v;
         2. Q.Enqueue(u);

Ausgabe: Arrays d[v], pred[v] für v $\in$ V.

Kürzeste s-v-Pfade

Satz: Sei G=(V,E) mit |V|=n, |E|=m. Bei Eingabe s $\in$ V berechnet Alg. BFS für jeden Knoten v $\in$ V einen kürzesten s-v-Pfad in Zeit $\mathcal{O}$(n+m).

Laufzeit:

- Schritt 1: Laufzeit $\mathcal{O}$(n)
- Sei s in der Zusammenhangskomponente V‘ $\subseteq$ V.
- Für jeden Knoten v $\in$ V‘ werden alle Knoten u $\in \Gamma$(v) betrachtet:

$$\sum_{v \in V'} |\Gamma(v)| \leq \sum_{v \in V} \deg(v) = 2|E|.$$

Korrektheit kürzester s-v-Pfad:

- p=(v, pred[v], pred[pred[v]], …, s) ist ein v-s-Pfad der Länge d[v].
  - Pfadeigenschaft: Kein Knoten wird zweimal in Q eingefügt.
  - Länge: d[pred[v]]=d[v]-1, d[pred[v]]=d[v]-2, …, d(s)=0
    $\Rightarrow$ p hat Länge d[v]
- Annahme: p‘=(v, v_1, …, v_k=s) hat Länge k < d[v].
  - Für jede Kante {u,v} in E gilt: d[v] $\leq$ d[u]+1.
    $\Rightarrow$ d[v] $\leq$ d[v_1]+1 $\leq$ d[v_2]+2 $\leq$ … $\leq$ d[s]+k = k (Widerspruch zu k < d[v])

Spannbaum mit kürzesten s-v-Pfade

Satz: Sei G=(V,E) ein zusammenhängender Graph. Dann berechnet Alg. BFS in Zeit $\mathcal{O}$(m) einen Spannbaum T=(V,E'),

$$E' = \{ \{v, pred(v)\} \in V \setminus \{s\} \times V \}$$

mit kürzesten s-v-Pfaden.

Nach Beweis auf letzter Folie gilt:

- Alle kürzesten v-s-Pfade in G haben die Form
  (v, pred[v], pred[pred[v]], …, s)
- Für alle u,v ∈ V:
  - ∃ kürzester u-s-Pfad p_u=(u, pred[u]=u_1, pred[pred[u]]=u_2, …, s)
  - ∃ kürzester v-s-Pfad p_v=(v, pred[v]=v_1, …, s)
- Fall1: p_u, p_v sind bis auf s knotendisjunkt
  - (u, u_1, u_2, …, s, …, v_2, v_1, v) ist u-v-Pfad
- Fall 2: Sei u_i ein Knoten mit minimalem i, so dass $u_i = v_j$ für ein j.
  - (u, u_1, …, u_i, v_{j-1}, v_1, v) ist ein u-v-Pfad

  - D.h. T ist zusammenhängend und |E'| = n-1
    ⇒ T ist ein Baum.

Zusammenhangskomponenten

Algorithmus Full-BFS

Eingabe: G=(V,E)

1. $i \leftarrow 1$
2. while $V \neq \emptyset$
   1. BFS(s) für beliebigen Startknoten $s \in V$
   2. $V_i \leftarrow \{v \in V | d[v] < \infty\}$
   3. $V = V \setminus V_i$; $G \leftarrow G[V]$;
   4. $i \leftarrow i+1$

Ausgabe: ZHK V_1, …, V_k

Laufzeit für ZHK

Satz: Algorithmus Full-BFS berechnet die Zusammenhangskomponenten in Zeit $\mathcal{O}$(n+m).

Korrektheit:

- Folgt aus den Sätzen zuvor.

Laufzeit:

- Jeder Knoten wird genau einmal in Q eingefügt:
  - Laufzeit: $\mathcal{O}$(n).
- Jede Kante {u,v} wird zweimal betrachtet:
  - für Γ(u) und für Γ(v)
  - Laufzeit: $\mathcal{O}$(m)

Datenstruktur Stack

Stack S

- Generieren eines Stacks S:
  - S $\leftarrow$ new Stack
- Einfügen von u in S:
  - S.Push(u)
- Leertest:
  - S.Isempty() $\neq$ TRUE $\Leftrightarrow$ S enthält mindestens ein Element
- Entfernen eines Elements:
  - S.Pop() liefert das zuletzt eingefügte Element zurück

Stack ist LIFO-Datenstruktur (last in, first out)
Annahme: Push, Isempty und Pop in Laufzeit $\mathcal{O}(1)$ durchführbar

Tiefensuche DFS (Depth First Search)

Algorithmus DFS

Eingabe: G=(V,E) in Adjazenzlistendarstellung, Startknoten s $\in$ V

1. Für alle v $\in$ V
   1. pred[v] $\leftarrow$ nil
2. S $\leftarrow$ new Stack; S.Push(s);
3. While (S.Isempty() $\neq$ TRUE)
   1. v $\leftarrow$ S.Pop();
   2. if ($\exists$ u $\in \Gamma$(v)\{s} mit pred[u] = nil) then
      1. S.Push(v); S.Push(u);
      2. pred[u] $\leftarrow$ v

Ausgabe: pred[v] für alle v $\in$ V

Rekursive Variante von DFS

Algorithmus DFS

Eingabe: G=(V,E) in Adjazenzlistendarstellung, Startknoten s $\in$ V

1. Für alle v $\in$ V
   1. pred[v] $\leftarrow$ nil
2. DFS-r(s);

Funktion DFS-r(v)

1. while ($\exists$ u $\in \Gamma$(v)\{s} mit pred[u] = nil)
   1. pred[u] $\leftarrow$ v
   2. DFS-r(u)

Ausgabe: pred[v] für alle v $\in$ V

Rekursive Aufrufe simulieren den Stack S.

DFS berechnet Spannbaum

Satz: Sei G=(V,E) zusammenhängend. Dann berechnet Alg. DFS in Zeit $\mathcal{O}$(m) einen Spannbaum T=(V,E'), E'={ {v,pred(v)} $\in$ V\{s}$\times$V}.

Korrektheit:

- Sei $pred^{(i)}[v]$ die i-malige Anwendung von pred[.] auf v.
- Für jeden Knoten v $\in$ V\{s} gilt:
  - $\exists$ i, 0<i<n: $pred^{(i)}[v]=s$
- D.h. $\forall$ u,v $\in$ V:
  - $\exists$ u-s-Pfad $p_u=(u, pred[u]=u_1, pred[pred[u]]=u_2, \ldots, s)$
  - $\exists$ v-s-Pfad $p_v=(v, pred[v]=v_1, \ldots, s)$

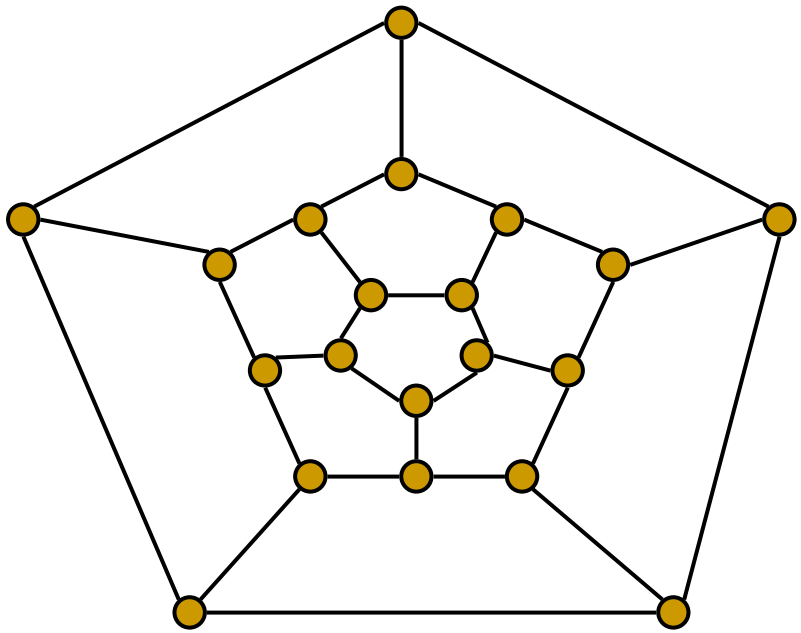
  Folgern analog zu Analyse bei BFS: $\exists$ u-v-Pfad
- Damit ist T zusammenhängend und hat |E'|=n-1 Kanten

  $\Rightarrow$ T ist ein Spannbaum.

Laufzeitanalyse: Analog zu BFS.

Hamiltonscher Kreis

Hamilton‘s Gesellschaftsspiel „Around the World“ (1859):

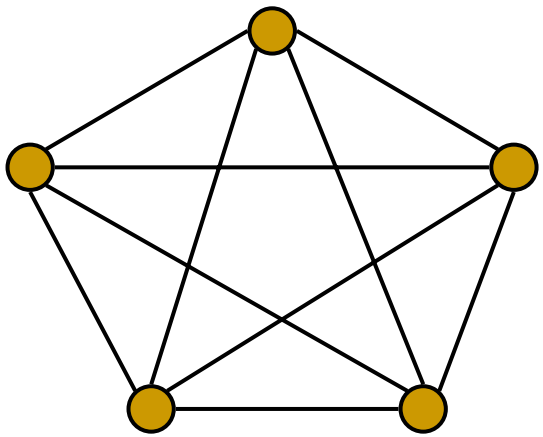


Finde einen Kreis, der alle Knoten enthält (Hamiltonkreis).
Graphen mit Hamiltonkreis nennt man hamiltonsch.

Modellierung durch Hamiltonkreis

Problem:

- 5 Personen an rundem Tisch
- Gibt es 2 Konstellationen bei denen jeder andere Nachbarn hat?



- Konstellationen=kantendisjunkte Hamiltonkreise des K_5.
  - 2 kantendisjunkte Kreise: äußerer Kreis und innerer Stern

Existenz eines Hamiltonkreises

Sei L = {G | G enthält Hamiltonkreis}.

Gegeben G=(V,E).

- Es ist algorithmisch schwer zu entscheiden, ob $G \in L$ oder $G \notin L$.
- Vermutlich kein Algorithmus mit Laufzeit polynomiell in (n,m).

Einfach: Algorithmus mit Laufzeit n!*n

- Für jede Permutation π_n: [n] $\Rightarrow$ [n]
  - Teste ob ($\pi_n(1)$, $\pi_n(2)$, …, $\pi_n(n)$) ein Kreis in G ist.

Hamiltonkreis in dichten Graphen

Satz: Sei G=(V,E) mit
$\deg(u)+\deg(v) \geq n$ für alle $u,v \in V$. **(*)**
Dann enthält G einen Hamiltonkreis.

Ann: G=(V,E) mit (*) und max. |E| ohne Hamiltonkreis

- Für alle $\{u,v\} \notin E$ gilt: G'=(V, $E \cup \{u,v\}$) enthält Hamiltonkreis C.
- Sei C=($u=v_1, v_2, \ldots, v_{n-1}, v=v_{n-1}$). Definieren
  - $S = \{v_i \in C \mid \{u, v_i\} \in E\}$
  - $T = \{v_i \in C \mid \{v, v_{i-1}\} \in E\}$
- Es gilt $|S|+|T| = \deg(u)+\deg(v) \geq n$, aber $|S \cup T| \leq |\{v_2,\ldots,v_n\}| = n-1$
  - Schubfachprinzip: $\exists$ j mit $v_j \in S$ und $v_j \in T$
  - D.h. $\{u, v_j\} \in E$ und $\{v, v_{j-1}\} \in E$
- C' = $(u, v_j, v_{j+1}, \ldots, v_{n-1}, v, v_{j-1}, \ldots, v_2)$ ist Hamiltonkreis in G (Widerspruch: G enthält keinen Hamiltonkreis.)

Konstruktion von Hamiltonkreis

Algorithmus Hamilton

Eingabe: G=(V,E) mit deg(u)+deg(v)$\geq$ n für alle u,v $\in$ V

- $c=(c_0,c_1,\ldots,c_{n-1}) \leftarrow (1,2,\ldots,n)$
- while ($\exists \{c_i,c_{i+1 \bmod n}\} \notin E$)
  - Wende auf c Konstruktion aus Beweis zuvor an.

Ausgabe: Hamiltonkreis c.

Terminierung:

- In jedem Schritt:
  - Ersetzung von nichtexistenter Kante durch Kanten aus E.
- Maximal n Iterationen.

Zusammenfassung

- Breitensuche BFS mit Startknoten s
  - Berechnet kürzeste s-v-Pfade
  - Berechnet Spannbaum
  - Zusammenhangskomponenten
  - Laufzeit $\mathcal{O}$(n+m)
- Tiefensuche
  - Berechnet Spannbaum
  - Laufzeit $\mathcal{O}$(n+m)
- Hamiltonsche Kreis
  - Berechenbar für sehr dichte Graphen