Algorithmen und Datenstrukturen (Master) WiSe 19/20

Benedikt Lüken-Winkels

February 6, 2020

Contents

1 Wörterbuchproblem

Menge S mit n Schlüssln aus einem Universum U. Operationen: INSERT (darauf achten, dass die Balance nicht verloren geht), DELETE, LOOKUP (Im Baum runterlaufen, bis das Element gefunden wurde)

Situationen

1. U linear geordnet, also existiert ein $\leq$-Test $\Rightarrow$ Suchbäume
2. U ist ein Intervall $\{0, ..., N-1\}$ der gesamten Zahlen $\Rightarrow$ Hashing

1.1 zu 1:

Randomisierte Suchbäume Idee: Benutze Zufallszahlen zur Balancierung eines binären Suchbaums

Binärer Suchbaum (Knoten-Orientiert) Schlüssel werden in den n Knoten eines binären Baums gespeichert, sodass im linken Unterbaum des Knotens mit Schlüssel x alle Schlüssel $< x$ **und** im rechten Unterbaum alle $> x$. Balanciert $\Rightarrow H\ddot{o}he(T) \leq logn$. Degeneriert $\Rightarrow H\ddot{o}he(T) = O(n)$

1.2 Definition: Randomized Search Tree (RST)

Sei $S = \{x_1, ..., x_n\}$ eine Menge von n Schlüsseln. Jedem x_i wird eine zusätzlich eine Zufallszahl (auch Priorität genannt) $prio(x_i)$ zugeordnet. $prio(x_i)$ sind gleichverteilte reelle Zufallszahlen $\in [0, 1]$ (Implementierung wären int-Zahlen, zB 32-bit).
Ein RST für S ist eine binärer Suchbaum für die Paare $(x_i, prio(x_i), 1 \leq i \leq n$, sodass

1. normaler Knoten-orientierter Suchbaum für die Schlüssel $x_i, ..., x_n$
2. Maximumsheap bzgl der Prioritäten. dh $prio(v) \geq prio(u)$, falls v Parent. ((u,v) sind Knoten in einem Baum). $\Rightarrow$ Wurzel enthält maximale Priorität.

Existenz durch Algorithmus zum Aufbau (rekursiv).

- Wurzel einthält (x_i, p_i) mit $p_i = prio(x_i)$ maximal
- Linker Unterbaum: RST für $\{(x_i, p_i) | x_j < x_i\}$
- Rechter Unterbaum: RST für $\{(x_k, p_k) | x_k > x_i\}$

Beispiel: $S = \{1, ..., 10\}$

- Schreibe Tabelle mit Prioriäten und Werten.
- Teile die Tabelle beim Maximum und schreibe es in die Wurzel. Wiederhole, bis alle Elemente geschrieben.

$\Rightarrow$ Wenn sich die Prioritäten genauso oder umgekehrt, wie die Schlüssel verhalten, erhält man einen degenrierten Baum. (bzgl $\leq$). zB $prio(x_i) = x_i$. Dieser Fall ist sehr unwahrscheinlich, wenn sich bei der Priorität um gleichverteilte Zufallszahlen handelt.

1.3 Operationen

- Lookup(x): normale suche in binärem Baum. Kosten $O(Höhe(T))$
- Insert(x): Füge einen neuen Knoten v als Blatt $(x, prio(x))$ gemäß des Schlüssels in den binären Baum ein, wobei $prio(x)$ neue Zufallszahl (kann die Prio-Ordnung zerstören). Dann: Rotiere v nach oben, bis die Heap-Eigenschaft gilt, also $prio(v) \leq prio(parent(v))$. Kosten: O(#Rotationen) = O(Höhe(T)). Alternativ: normales einfügen in binären Baum in absteigender Reihenfolge der Prioritäten.
- DELETE(x): Sei v der knoten mit Schlüssel x (v = Lookup(x)). Kosten: O(#Rotationen) = O(1 + |L| + |R|)
  1. Rotiere v nach unten, bis v ein Blatt ist. R = linkes Rückgrat des rechten Unterbaums von v. L = rechtes Rückgrat des linken Unterbaums.
  2. Entferne das Blatt.
- Split(y) $\rightarrow S_1 = \{x \in S | x \leq y\}, S_2 = \{x \in S | x \geq y\}$ (Teile den Baum, indem y mit maximaler Priorität zur Wurzel rotiert wird)
  1. Insert$(y + \epsilon)$ mit Priorität ∞
  2. Entferne die Wurzel
- Join(T_1, T_2): $S \leftarrow S_1 \cup S_2$. T_1 RST für S_1 und T_2 RST für S_2
  1. Konstruiere T (Füge y zwischen Max(S_1) und Min(S_2) ein. Voraussetzung: Max(S_1) < Min(S_2)
  2. Lösche die Wurzel (Durch runterrotieren des eingefügten Knotens y)

1.4 Analyse des RST

Wir analysieren die erwarteten Kosten einer Delete-Operation (Insert $\rightarrow$ umgekehrtes Delete). Seit T ein RST für die Menge $\{x_1, ..., x_n\} mit x_1 < x_2 < ... < x_n$ der durch Inserts aufgebaut wurde. Bertrachte die Operation Delete(x_k) für eine $k, 1 \leq k \leq n$. Für einen Knoten x_k im Baum T mit Suchpfad P_k, L_k rechtes Rückgrad von T_l und R_k linkes Rückgrad von T_r. Kosten $O(|P_k| + |L_k| + |R_k|)$. Wir schätzen die Erwartungswerte

1.5 Lemma 1:

- a) E$(|P_k|) = H_k + H_{n-k+1} - 1$

$$k - te\ HarmonischeZahl = H_k = \sum_{i_1}^{k} \frac{1}{i}\ H_k \leq ln(x) + 1$$

- b) E$(|L_k|) = 1 - \frac{1}{k}$
- c) E$(|R_k|) = 1 - \frac{1}{n-k+1}$

Beweis Betrachte eine Permutation $\pi : [1..n] \to [1..n]$ (bijektive Abbildung), die die Schlüssel absteigend nach ihren Prio Werten sortiert. Dann gilt:

1. Jede Permutation π ist gleichwahrscheinlich (Wahrscheinlichkeit $\frac{1}{n!}$), da die Prioritäten gleichverteilte Zufallszahlen sind.
2. Man erhält den selben binären Baum durh einfügen der Schlüssel in einen unbalancierten Baum in der Reihenfolge, die π angibt. $\to$ gleiches Vehalten, wie ein zufälliger binärer Baum.
3. Baum wächst nur an den Blättern.

Trick: arbeite ab jetzt mit zufälliger Permutation statt den Prioritäten. $\to$ normaler Binärbaum mit zufälliger Einfügereihenfolge.

Teil a) des Lemmas P_k ist Suchpfad für Knoten x_k. Seien P'_k und P''_k Teilfolgen von P_k mit: $\forall v \in P'_k, key(v) \leq x_k$ und $\forall u \in P''_k, key(u) \geq x_k$.

Proof. Beobachtungen:

1. $|P_k| = |P'_k| + |P''_k| - 1$ (x_k in beiden Teilfolgen)
2. P'_k = Menge der knoten v mit:
   - Wenn v eingefügt wird, gilt key(v) ist maximal mit key(v)$\leq x_k$
3. P''_k = Menge der knoten u mit:
   - Wenn u eingefügt wird, gilt key(u) ist minimal mit key(u)$\geq x_k$

Wir zeigen

1. E($|P'_k|$) = H_k
2. E($|P''_k|$) = H_{n-k+1}

zu 1) K mögliche Kandidaten für $P'_k\{x_1, ..., x_k\}$. Spiel: Ziehe zufällig Schlüssel aus $\{x_1, ..., x_k\}$. E($|P'_k|$) = Erwartungswert, wie of ein Kandidat gezogen wird, der $\geq$ als alle vorher gezogenen ist (neues Maximum). $A^k = E(|P'_k|)$ (Spiel A)

$$A^k = \sum_{i=1}^{k} \frac{1}{k} \cdot (1 + A^{k-i})$$

Im Zug x_i schließt $x_1...x_i$ au. Dann gleiches Spiel mit K-i Kandidaten.

$$A^k = \frac{1}{k}(k + \sum_{i=1}^{k} A^{k-i})$$

$$= 1 + \frac{1}{k} \sum_{i=1}^{k} A^{k-i}$$

Wir zeigen durch Induktion über k, dass $A^k = H_k$

IA

$$= 1 + \frac{1}{k} \sum_{i=0}^{k} H_i$$

Eigenschaften der harmonischen Zahlen:

1.
$$\sum_{i=0}^{k} H_i = k \cdot (H_k - 1)$$

2.
$$H_k \leq 1 + lnk$$

aus 1) folgt:

$$\begin{aligned} A^k &= 1 + \frac{1}{k} \cdot k \cdot (H_k - 1) \\ &= 1 + H_k - 1 \\ &= H_k \end{aligned}$$

Der Erwartungswert ist ist gleich der k-ten Harmonischen Zahl. $E(P'_k) = H_k$. Abschätzung von $E(P''_k) =: B^k$. (Spiel B) kandidaten $\{x_k ... x_n\}$: Zähle, wie of ein neues Minimum gezogen wird. Dann sieht man leicht, dass

$$B^k = H_{n-k+1}$$

Beweis: symmetrisch.

$$\begin{aligned} E(|P_k|) &= E(|P'_k|) + E(|P''_k|) - 1 \\ &= A^k + B^k - 1 \\ &= H_k + H_{n-k+1} - 1 \end{aligned}$$

□

Teil b) des Lemmas

Proof. L_k und R_k Seien $L_k = v_1, ..., v_l$ und $R_k = u_1, ..., u_m$

Erwartungswerte Spiel C: Ziehe zufällig Elemente aus $\{x_1 ... x_n\}$. Sobald x_k gezogen wird: ??? Trigger ??? Sei C^k der Erwartungswert dieses Spiels, dh $C^k = E(|L_k|)$.

$$C^k = \frac{1}{k} \cdot A^{k-1} + \sum_{i=1}^{k-1} \frac{1}{k} \cdot C^{k-i}$$

im 1. Zug x_k, dass Spiel mit k-1 Kandidaten (alle kleiner, als x_k).

$$C^k = \frac{1}{k} (\cdot H_{k-1} + \sum_{i=0}^{k-1} \cdot C^i)$$

Trick: Schätze die Differenz zweier aufeinanderfolgender C^is $= \delta_j = C^{j+1} - C^j$ ab.

$$\Rightarrow C^k = \sum_{j=1}^{k} \delta_j + C^0$$

Beatrachte:

$$(j+1) \cdot C^{j+1} - j \cdot C^j$$

$$= (j+1)\frac{1}{j+1}(H_j + \sum_{i=0}^{j} C^i) - j\frac{1}{j}(H_{j-1} + \sum_{i=0}^{j} C^i)$$

$$= H_j + \sum_{i=0}^{j} C^i - (H_{j-1} + \sum_{i=0}^{j} C^i)$$

$$= H_j - H_{j-1} + C^i$$

$$= \frac{1}{j} + C^j$$

Wir wissen nun, dass

$$(j+1) \cdot C^{j+1} - j \cdot C^j = \frac{1}{j} + C^j$$

$$\frac{1}{j} = (j+1)C^{j+1} - (j+1) \cdot C^j$$

$$\frac{1}{j(j+1)} = C^{j+1} - C^j = \delta_j$$

$$\Rightarrow \delta_j = \frac{1}{j(j+1)} = \frac{1}{j} - \frac{1}{j+1}$$

$$C^k = \sum_{j=1}^{k-1} \delta_j = \sum_{j=1}^{k-1} (\frac{1}{j} - \frac{1}{j+1})$$

$$= 1 - \frac{1}{k}$$

□

Teil c) des Lemmas Spiel $D^k = E(|R_k|)$: Wie oft wird ein neues Minimum größer x_k gezogen, nachdem x_k gezogen wurde (Trigger).

Proof. symmetrisch:

$$D^k = \frac{1}{n-k+1} \cdot B^{k-1} + \sum_{i=1}^{k-1??} \frac{1}{n+k-1} D^{i-k??}$$

...

$$D^k = 1 - \frac{1}{n-k+1}$$

□

Satz Sie T ein RST für eine Menge von n Schlüsseln. Dann gilt:

1. Die erwartete Laufzeit fpr Insert, Delete und Lookup ist O(logn)
2. Die erwartete Zahl der Rotationen bei Delete ist < 2

Beweis

1. Kosten von Lookup $= O(|P_k|)$, Insert und Delete $= O(|P_k| + |L_k| + |R_k|)$ Kosten: $O(H_k + H_{n-k+1} + 1 - \frac{1}{k} + 1 - \frac{1}{n-k+1}) = O(H_n) = O(lnn) = O(logn)$
2. Erwartete Zahl der Rotationen: $E(|L_k|) + E(|R_k|) < 2$

2 Hashing

2.1 Hashing mit offener Adressierung

Tafel T $[0, ..., m-1], m \leq n, |S| = n$ Verwende die Folge von Hashfunktionen h_0, h_1

$$h_i(x) = f(x) + i \cdot g(x), i = 0, 1, ...$$

Häufig verwendet wird

$$h_i(x) = (x \cdot modm + i) \cdot modm$$

→ Linear Probing.
Idee

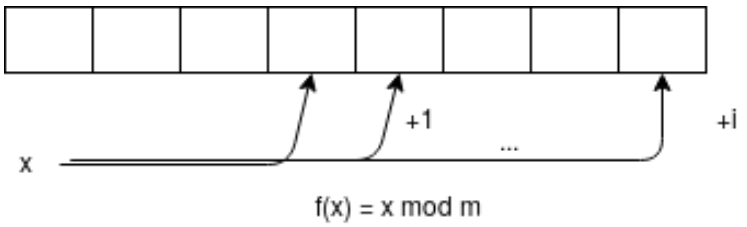

Figure 1: Idee: Offene Adressierungs-Tabelle

Operationen Tafelposition T[i] belegt oder frei.

- Init: alle frei
- Insert(x): betrachte die Tafelpositionen $T[h_0(x)], T[h_1(x)], T[h_2(x)]$ bis $T[h_i(x)]$ frei und speicher x dort ab. $T[h_i(x)] \leftarrow x$ und markiere $T[h_i(x)]$ als belegt. Voraussetzungen:
  1. $m \leq |S| = n$

2. $h_i(x)$ frei $i = 0, 1, 2, ...$ muss alle Tafelpoitionen durchlaufen

- Lookup(x): Teste die Tafelpoition $T[h_i(x)]$ für $i = 0, 1, 2, ...$ bis entweder $T[h_i(x)] = x$ erfolgreich oder $T[h_i(x)]$ ist frei. **Terminiert nicht**, wenn m=n und die Tafel voll ist und das gesuchte Element nicht vorhanden ist. Daher idR $m \geq n$
- Delete(x): (Idee 1):
  1. $j \leftarrow Lookup(x)$
  2. $T[j] \leftarrow frei$, dann sind auf j folgende Elemente nicht mehr erreichbar.

  (Idee 2):
  1. Dritter Zustand: 'gelöscht' (Details: Übung)

2.2 Perfektes Hashing

Situation: Statische Menge S von n Schlüsseln aus $[0, .., N-1]$. **Ziel:** Speichere S in einer Tafel der Größe O(n), sodass Lookup in Zeit O(1) realisiert werden kann (N » n, N sehr viel größer, als n). **Andere Formulierung:** Finde einer Hashfunktion $h : [0, ..., N-1] \rightarrow [0, ..., S-1]$ mit

1. S = Größe der Tafel und S = O(n)
2. h injektiv auf S

Zur Konstruktion oder Auswahl einer solchen Funktion Hashfunktion verwenden wir ein probabilitisches Verfahren (Zufallsverfahren).

Idee 2-stufiges Hashing-Schema (Hashing-Verfahren)

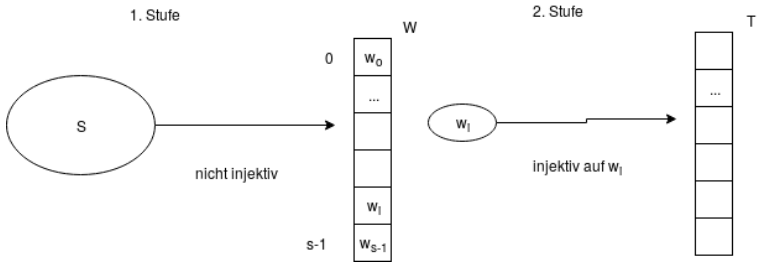

Figure 2: Idee: Perfekt Hashing Tabellen

Sei p eine Primzahl mit p > N und $S \in \mathbb{N}$ (Tafelgröße). Betrachte folgende Hashfunktionen:

$$h_k : [0, ..., N-1] \rightarrow [0, ..., S-1]$$

mit $h_k(x) = ((k \cdot x) mod p) mod s$ für alle $1 \leq k \leq p-1$ (Modulo Primzahl ergibt einen Restkörper, zB Inverses der Multiplikation). Diese Funktionen sind im Allgemeinen nicht injekt, dh h_k verteilt die Menge S auf s Buckets W_0^k, W_1^k, W_{s-1}^k.

$$\Rightarrow W_i^k = \{x \in S | h_k(x) = i\}$$

h_k injektiv auf $S \Leftrightarrow |W_i^k| \leq 1$ für $0 \leq i \leq s-1$

Lemma 1: Für jede Menge $S \subseteq \{0, ..., N-1\}, |S| = n$ gilt $\exists k, 1 \leq k \leq p-1 mit$

$$\sum_{i=0}^{s-1} \binom{|W_i^k|}{2} < \frac{n^2}{s}$$

(Anzahl der Kollisionen $< \frac{n^2}{s}$).

Beweis zunächst: Behauptung.

$$\sum_{k=0}^{p-1} \sum_{i=0}^{s-1} \binom{|W_i^k|}{2} < (p-1)\frac{n^2}{s}$$

Daraus folgt das Lemma (indirekt). Annahme, das Lemma 1 gilt nicht, dh $\forall 1 \leq k \leq p-1$:

$$\sum_{i=0}^{s-1} \binom{|W_i^k|}{2} \geq \frac{n^2}{s}$$

$$\Rightarrow \sum_{i=0}^{s-1} \binom{|W_i^k|}{2} \geq \frac{n^2}{s}$$

Widerspruch zur Behauptung!

Beweis zur Behauptung.

$$(1) \sum_{k=0}^{p-1} \sum_{i=0}^{s-1} \binom{|W_i^k|}{2}$$

= Anzahl der Paare $(k, \{x, y\}$, $x \neq y$ und $h_k(x) = h_k(y)$ (Anzahl der Kollisionen).
Wir schätzen zunächst den Betrag für 2 feste Werte $x \neq y$ zur Behauptung ab.

Sei also $x \neq y$:
(1) = Anzahl aller Ks mit $(k \cdot x \cdot mod p) mod s = (k \cdot y \cdot mod p) mod s$

$$\Leftrightarrow ((k \cdot x \cdot mod p) - (k \cdot y \cdot mod p)) mod s = 0$$

$$k \cdot x \cdot mod p - k \cdot y \cdot mod p = i \cdot s \text{ für ein } i \in \mathbb{Z}$$

$$k \cdot (x-y) mod p = i \cdot s$$

Es gibt maximal $\frac{2(p-1)}{s}$ mögliche Lösungen. Da p eine Primzahl ($\mathbb{Z}_p$ ist ein Körper) hat jede dieser Gleichungen höchstens 1 Lösung für k. $\Rightarrow$ Beitrag eines festen Paares $x \neq y$ zu (1) ist maximal $\frac{2(p-1)}{s}$

$$\begin{aligned} \Rightarrow (1) &\leq \binom{n}{2} \cdot \frac{2(p-1)}{s} \\ &= \frac{n(n-1)}{2} \cdot \frac{2(p-1)}{s} \\ &< \frac{n^2(p-1)}{s} \end{aligned}$$

□

Folgerung 1 (aus Lemma 1)
Für s=n (dh Tafel der Größe n):

$$\exists k, 1 \leq k \leq p-1 \text{ mit } \sum_{i=0}^{n-1} |W_i^k|^2 < 3n$$

Beweis für Folgerung 1. Betrachte Lemma 1 für s=n

$$\exists k : \sum_{i=0}^{n-1} \binom{|W_i^k|}{2} < n \tag{1}$$

$$\sum_{i=0}^{n-1} \frac{|W_i^k| \cdot (|W_i^k|-1)}{2} < n \tag{2}$$

$$\sum_{i=0}^{n-1} |W_i^k| \cdot (|W_i^k|-1) < 2n \tag{3}$$

$$\sum_{i=0}^{n-1} |W_i^k|^2 < 2n + \sum_{i=0}^{n-1} |W_i^k| (= n) \tag{4}$$

$$< 3n \tag{5}$$

□

Folgerung 2 Für $s = n^2$ dh quadratische Tafelgröße.
$\exists k' : 1 \leq k \leq p-1$, sodass die Hashfunktion

$$k_{k'} : x \to (k'x \cdot mod p) mod s$$

injektiv auf S ist dh $|W_i^k| \leq 1$ für $0 \leq i \leq s-1$.
$\Rightarrow$ Für Tafeln mit quadratische Größe existiert eine perfekte (injektive) Hashfunktion.

Beweis für Folgerung 2. Betrachte Lemma 1 mit $s = n^2$

$$\exists k' : \sum_{i=0}^{n^2-1} \binom{|W_i^k|}{2} < 1 \tag{6}$$

$$\Rightarrow |W_i^k| \leq 1 \tag{7}$$

$$\Rightarrow h_{k'} \text{ ist injektiv auf S} \tag{8}$$

(zu 6): dh Keine Kollisionen, bzw Doppeltbelegung in den W_i^k

□

Folgerung 2 zeigt Perfektes Hashing mit quadratischem Platz. Vermeidung des quadratischen Platzbedarfs durch ein 2-stufigen Hashing-Schema:

- Stufe 1: Wähle ein k gemäß Folgerung 1, dh Tafelgröße s=n und Hashfunktion h_k mit $\sum_{i=0}^{n-1} |W_i^k| < 3n$
- Stufe 2: Für jedes nicht-leere Bucket W_i^k der ersten Stufe verwende iene Tafel der Größe $s_i = |W_i^k|^2 (0 \leq i \leq n-1)$ und wähle ein k_i gemäß Folgerung 2, dh h_{k_i} ist injektiv auf W_i^k

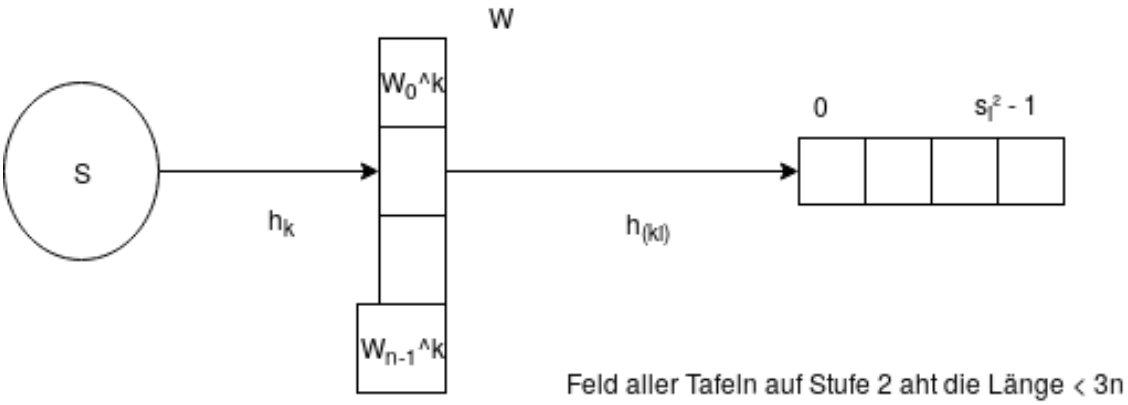

Figure 3: Konzept 2-Stufen Hashing

2.2.1 Datenstruktur

4 Felder + Variable k

- Variable k: (h_k der 1. Stufe)
- k[0,...,n-1]: k[i] ist k_i der 2. Stufe

- Size[0,...,n-1]: Size[i] = $|W_i^k|$ Bucketgrößen der 1. Stufe
- Ptr[0,...,n-1]: Pointer auf die Hashtafeln der 2. Stufe. Ptr[i] zeigt auf die Tafel B[0,...,Size[i]^2-1]
- T[0,...,3n]: Gesamtspeicherplatz aller B-Tafeln der 2. Stufe

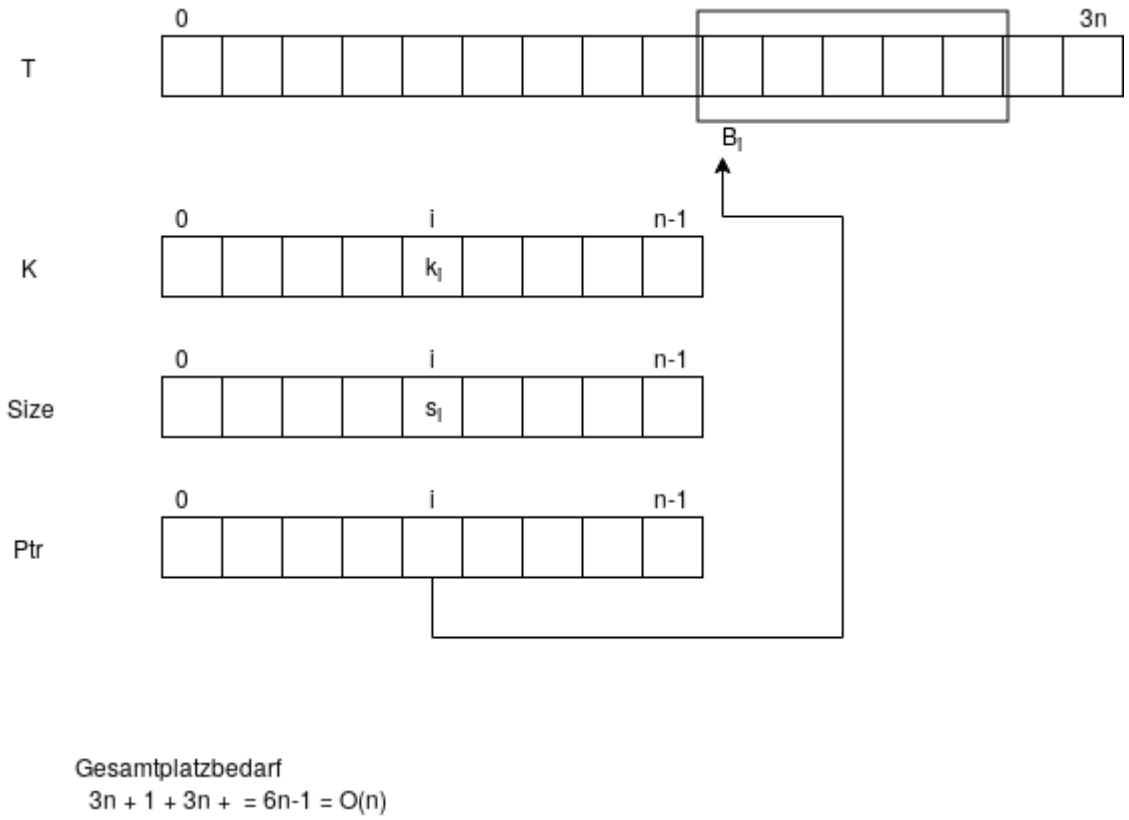

Figure 4: Datenstruktur: Perfekt Hashing

Abspeichern eines Elements x

$$i \leftarrow (k \cdot xmodp)modn \tag{9}$$
$$k' \leftarrow K[i] \tag{10}$$
$$s' \leftarrow Size[i] \tag{11}$$
$$j \leftarrow (k' \cdot xmodp)mods' \tag{12}$$
$$Ptr[i][j] \leftarrow x \tag{13}$$

2.2.2 Aufbauzeit der Datenstruktur

Wir findet man die k bzw k' Werte der ersten und zweiten Stufe (gemäß Folgerung 1 und 2). Aus Folgerungen 1 und 2 folgt: Es existiert immer mindestens ein Wert für k (dh eine geeignete Hashfunktion h_k). **Aufbaualgorithmus:** Teste alle k-Werte. Auf 1. und 2. Stufe

---

def *Aufbau***:**
  for $k = 1$ **to** $p - 1$ **do**
    1.Stufe: Teste ob $\sum_{i=0}^{n-1} |W_i^k|^2 < 3n$;

Kostet im worst case $O(p \cdot n) = O(n \cdot N)$

2.Stufe für jedes Bucket W_i^k

---

def *Aufbau***:**
  for $k' = 1$ **to** n **do**
    Teste ob $h_{k'}$ injektiv auf W_i^k;

Kostet für $W_i^k O(p \cdot |W_i^k|$ für alle Buckets $W_i^k (0 \leq i \leq n)$. Gesamtlaufzeit:

$$O(\sum_{i=0}^{n-1} p \cdot |W_i^k|) \tag{14}$$

Eine genauere Analyse zeigt, dass es viele k-Werte mit den geforderten Eigenschaften gibt.

Folgerung 3 (aus Lemma 1)
Für mindestens die Hälfte aller k, $0 \leq k \leq p - 1$, gilt

$$\sum_{i=0}^{n-1} |W_i^k|^2 < 5n (s = n)$$

Folgerung 4 (aus Lemma 1)
Für mindestens die Hälfte aller k', $0 \leq k' \leq p - 1$, gilt

$$h_{k'} : x \to (k' \cdot x \ mod p) mod 2 \cdot n^2$$

ist injektiv auf S (angedwandt: $W_i^k s$. Beweis analog zum Beweis der Folgerung 1 und 2.

Änderungen in der Datenstruktur Auf 2.Stufe Tafelgrößen verdoppeln, dh jeweils Größe $2 \cdot |W_i^k|^2$. Platzbedarf der 2.Stufe:

$$\sum_{i=0}^{n-1} 2 - |W_i^k|^2 = 2 \sum_{i=0}^{n-1} |W_i^k|^2 < 10n$$

Insgesamt: Platz 13n = O(n).

Aufbau Stufe 1: Wähle ein zufälliges $k \in \{1, ..., p-1\}$ bis Folgerung 3 erfüllt. Wahrscheinlichkeit daür ist jeweils $\geq \frac{1}{2}$. **Frage:** Wie hoch ist der Erwartungswert für die Anzahl der Tests. Analog zum Münzwurf: Wie viele Würfe, bis eine bestimmte Seite erscheint? Erwartungswert für diese Zahl (Integrierende Reihe):

$$\sum_{i=0}^{\infty} \frac{1}{2^i} \cdot i = \frac{0.5}{(1-0.5)^2} = 2$$

Erwartete Laufzeit für Stufe 1 ist dann $O(2 \cdot |S|) = O(n)$. Analog für Stufe 2: Erwartete Zahl der Tests = 2. $\Rightarrow$ Gesamtlaufzeit O(n).

2.2.3 Zusammenfassung

Man kann eine Menge S von n Schlüsseln aus $[0, ..., N-1]$ so abspeichern, dass gilt

1. Platubedarf ist O(n)
2. Erwartete Aufbauzeit O(n)
3. Zugriffszeit (Lookup) O(1) worst-case

Dynamisierung ist möglich (Dynmaic Perfect Hashing). Idee: Zeigen, dass die gewählten k-Werte mit großer Wahrscheinlichkeit für weitere Schlüssel funktionieren.

3 Planare Graphen

Literatur: Nishizekim und Chiba (Planar Graphs). Graph kann in die Ebene gezeichnet werden, ohne dass sich Kanten kreuzen. Wir betrachten ungerichtete Graphen G=(V,E).

3.1 Definition

1. Ein Graph G=(V,E) ist planar, wenn G eine planare Zeichnung hat.
2. Eine planare Zeichnung von G ordnet jedem Knoten v einen Punkt $p \in \mathbb{R}^2$ (Ebene) zu, $p = pos(x)$ heißt die Position von v, und jeder Kante (v, w) eine stetige Kurve im $\mathbb{R}^2$ mit den Endpunkten $pos(v)$ und $pos(w)$ zu, sodass sich diese Kurven paarweise nicht schneiden, außer in ihren Endpunkten.

G planar $\Leftrightarrow \exists$ planare Zeichnung.

Beispiele

nicht-planar $K_{3,3}$ bipartiter vollständiger Graph mit jeweils 3 Knoten auf jeder Seite. K_5 vollständiger Graph mit 5 Knoten. Ist ein Graph nicht planar, kann einer der beiden nicht-planaren Graphen gefunden werden.

3.2 Planaritätstest

Frage: ist der Graph planar?

1. Falls planar: planare Zeichnung
2. Falls nicht: möglichst kleiner nicht-planarer Teilgraph ($K_{3,3}$ oder K_5)

Beobachtung

- G ist planar $\Leftrightarrow \exists$ planare Zeichnung auf einer Kugeloberfläche
- Oberflächenstruktur von Polyedern kann durch planare Graphen dargestellt werden. (zB Würfel)

3.3 Wichtige Begriffe und Definitionen

3.3.1 Knotenzusammenhang

Der Zusammenhand eines Graphen (genauer Knotenzusammenhang).

- G heißt einfach zusammenhängend, wenn es für jedes Paar (v, w) von Knoten einen Pfad zwischen v und w gibt. Zusammenhangskomponenten sind zusammenhängede Teilgraphen. Der Knoten, der den Graph zerfallen lässt, nennt man Artikulationspunkt (cut-vertex)
- G heißt zweifach zusammenhänged (biconnected), falls $G\backslash\{v\}$ für jeden Knoten $v \in V$ zusammenhängend ist. **Splitgraphen** G_1 und G_2 durch Entfernung eines Separationspaares, erweitert um Kopieen des Entferten Paares. Kante innerhalb des Separationspaares kann existieren oder nicht.
- Ein zweifach zusammenhängeder Graph G heißt dreifach zusammenhänged, wenn für beliebige Knoten v, w gilt, $G\backslash\{v, w\}$ ist zusammenhängend. Die Knoten, die den Graph zerfallen lassen nennt man Separationspaar.
- Allgemein k-fach zusammenhängend kann man beliebige k-1 Knoten entfernen, sodass G zusammenhängend bleibt.

1-fach, 2-fach und 3-fach kann mit DFS in Zeit O(n+m) gelöst werden.

3.3.2 Beobachtung

Ein nicht-zweifach zusammenhängender Graph G ist genau dann planaer, wenn seine zweifach Zusammenhangskomponenten (Blöcke) planar sind.

Idee Konstruiere für jeden Block eine planare Zeichnung, sodass alle Cut-Vertices außen liegen und kleben diese zusammen.

3.4 Planare Einbettung

(Plane Graph) Abstrakte planare Zeichnung eines Graphen:

- Keine Positionen (Koordinaten) für die Kanten
- Angabe der Flächen (Faces) in einer mögichen planaren Zeichnung

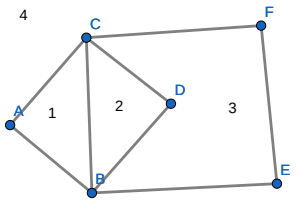

Figure 5: Die Faces sind die Flächen, die durch die Kanten eingeschlossen werden.

Alternativ Festlegung der Reihenfolge der Nachbarknotes jedes Knotens (Adjazenzliste) in einer möglichen planaren Zeichnung gegen den Uhrzeigersinn. Wir arbeiten meistens mit der 2. Alternative. Die beiden Definitionnen sind äquivalent.

Planaritätstest bzw Einbettung besteht dann in der Aufgabe die Adjazenzlisten in eine Reihenfolge zu sortieren, so dass diese eine planare Einbettung definiert (falls möglich). Im Allgemeinen besitzt ein planarer Graph verschiedene panare Einbettungen (Hinweis: Einbettung ist eindeutig für 3-fach zusammenhängende Graphen). Beispiel für nicht-eindeutige Einbettungen (siehe Graphik). Die Faces sind die Flächen, die durch die Kanten eingeschlossen werden.

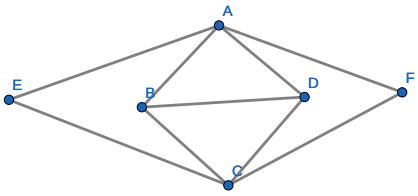

Figure 6: Graph mit nichteindeutiger Einbettung: Rotation des unteren Teils (B und D)ergibt verschiedene Faces

Unterteilung (Subdivision) Sei G ein ungerichteter Graph. G' heißt Unterteilung oder Subdivision von G, wenn G' aus G durch ersetzen von Kanten durch Pfade entsteht (Platzieren neuer (Unterteilungs-)Knoten auf bestehende Kanten).

3.5 Die Euler-Formel

Sei G eine zusammenhängende planare Einbettung mit n Knoten, m Kanten und f Faces. Dann gilt

$$n - m + f = 2$$

Beweis Euler-Formel. Induktion über m
$\underline{m = 0:}$ Graph besteht aus einem isolierten Knoten: n=1, f=1
$\underline{m > 0:}$ Sei G eine planare Einbettung mit m Kanten, n Knoten und f Faces
$\underline{IA}$ Für alle Einbettungen mit m-1 Kanten gilt die Formel
$\underline{Fall1}$ G ist azyklisch (dh: Baum)

1. f = 1
2. G besitzt Knoten v mit $deg(v) = 1$ (Blatt)

Die planare Einbettung $G\backslash\{v\}$ ist zusammenhängend und hat n-1 Knoten und m-1 Kanten.

$$IA : (n-1) - (m-1) + f = 2$$

$\underline{Fall2}$ G ist kein Baum dh besitzt einen Kreis (f>1). Sei e eine beliebige Kante auf dem Kreis. Betrachte die planare Einbettung $G\backslash\{e\}$

- $G\backslash\{e\}$ ist zusammenhängend
- $G\backslash\{e\}$ hat m-1 Kanten, n Knoten und f-1 Flächen

$$IA : n - (m-1) + (f-1) = 2$$

□

3.5.1 Folgerung

Sei G ein planarer Graph mit $n \geq 3$ Knoten und m Kanten, dann gilt $m = 3n - 6$. dh $m = O(n)$, also linear viele Kanten.

Proof. Ein maximal planarer Graph ist ein planarer Graph, der durch Hinzufügen einer Kante $(v, w) \notin E$ nicht-planar wird. Beobachtung: Alle Faces in jeder planaren Einbettung von G sind Dreiecke (Triangulierung). Jedes Face in einer Triangulierung hat 3 Rand-Kanten und jede Kante liegt am Rand von 3 Faces.

$$\Rightarrow 3f = 2m$$

Einstetzen in Euler-Formel

$$n - m + \frac{2}{3}m = 2$$
$$m = 3n - 6$$

$m \leq 3n - 6$ für beliebige planare Graphen □

3.5.2 Folgerung

Sie G ein **bipartiter** planarer Graph Dann gilt $m \leq 2n - 4$. Beweis: Keine Kreise ungerader Länge in bipartiten Graphen. Kleinstmögliche Fläche in einem bipartiten Graphen ist ein Viereck.

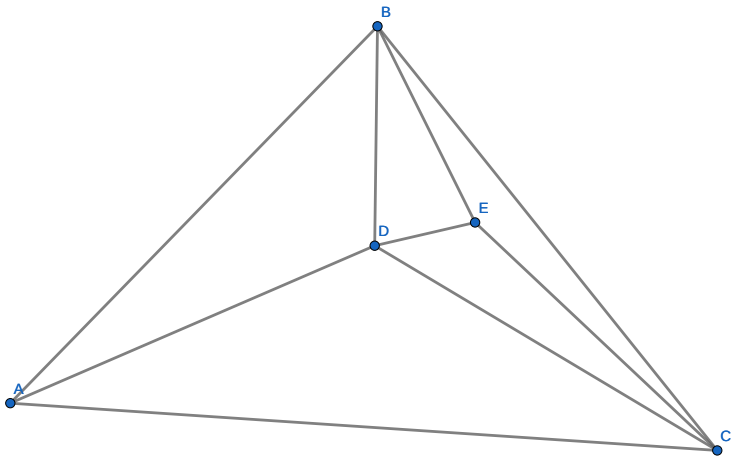

Figure 7: Beispiel für maximal planaren Graphen

3.5.3 Folgerung

$K_{3,3}$ und K_5 sind nicht planar. Jeder planare Graph besitzt einen Knoten v mit $deg(v) \leq 5$.

Proof. Annahme $\forall v \in V \ deg(v) \geq 6, \ m = \sum_{v \in V} \frac{deg(v)}{2} \geq \frac{6n}{2} = 3n$ □

3.6 Das Färbungsproblem für planare Graphen

Knotenfärbung: k Farben $\{1, ..., k\}$. Finde eine Abbildung $f :\rightarrow \{1, ..., k\}$, sodass $f(v) \neq f(w)$ für alle Kanten $(v, w) \in E$. **Frage:** Wie viele Farben k sind notwendig? (minimales k):

- k=n geht immer
- notwendig k=n für K_n

3.6.1 Vierfarbensatz

k=4 für planare Graphen. Ursprung in der Darstellung von Landkarten beim Einfärben der Länder, sodass benachbarte Länder verschiedene Farben haben.

Dualer Graph $G = (V, E)$ Für jedes Land (Fläche) einen Knoten $v \in V, (v, w) \in E \Leftrightarrow$ Länder v und w haben eine gemeinsame Grenze.

1. G ist planar
2. Knotenfärbung von G $\Leftrightarrow$ Färbung der Karte

Einfacher, als der Vierfarbensatz ist der Fünffarbensatz.

Proof. Induktion über n
IA für alle Grapfen mit $n \leq 5$.
Sei G ein planarer Graph mit $n > 5$. Aus Folgerung 3: G besitzt einen Knoten mit $deg(v) \leq 5$.
Fall 1 G besitzt einen Knoten v mit $deg(v) \leq 4$. Nach IA $G\backslash\{v\}$ besitzt eine 5-Färbung. Betrachte die r Farben den Nachbarn von v in G, dann gilt $r \leq 4$ ($deg(v) \leq 4$) $\Rightarrow$ mindestens eine Farbe frei. Diese erhält v.
Fall 2 Alle Knoten haben mindestens Grad 5. Aus Folgerung 3: Sei $v \in V$ mit $deg(v) = 5$. Beobachtung: v besitzt 2 Nachbarnknoten x, y mit $(x, y) \notin E$ (dh x,y unabhängig), sonst totaler Graph K_5. Betrachte $G' = G\backslash\{v\}$: Planare Einbettung von G und entferne v. G" = Veschmelze die unabhängigen Knoten x und y zu einem Knoten z. G" ist planar. G" hat nur noch n-2 Knoten.
Aus IA: G" kann mit 5 Farben gefärbt werden. Sei c die Farbe von z in dieser Färbung. Expandiere Z wieder zu x, y mit $color(x) = color(y) = c$. $\Rightarrow$ Die 5 Nachbarn von v belegen höchstens 4 Farben und eine Farbe ist frei. □

Andere Anwendung Finde eine große unabhängige Knotenmenge (Independent Set). Beobachtung: 5-Färbung $\rightarrow$ Farbklasse (Menge von Knoten der gleichen Farbe) mit mindestens $\frac{n}{5}$ Knoten. Alle Knoten einer Farbe bilden ein Independent Set.

3.7 Satz von Kuratowski

Notation Verschmelzung (Kontraktion) entlang einer Kante $e = (v, w)$: $G|e \rightarrow$ Graph den man durch Verschmelzen der Endknoten v und w. G planar $\Rightarrow G|e$ planar.

Satz Ein Graph G ist genau dann planar, wenn er keine Unterteilung (Subdivision) des K_5 oder $K_{3,3}$ enthält.

Proof. Satz von Kuratowski
$\Rightarrow$ trivial
$\Leftarrow$ (Jeder Graph ohne eine Unterteilung des K_5 oder $K_{3,3}$ ist planar) Induktion über n
Sei G ein Graph mit n Knoten, der keine Unterteilung des K_5 oder $K_{3,3}$ enthält.
IB $n \leq 5$ (dh, kein K_5)
IA Für alle Graphen mit weniger als n Knoten gilt die Behauptung
Fall 1 G ist nicht 3-fach zusammenhängend
Angenommen, G ist 2-fach zusammenhängend$\Rightarrow$ G bestitzt ein Separationspaar (x,y),

das G in die Blöcke G_1 und G_2 zerlegt, mit jeweils weniger, als n. Offensichtlich enthalten G_1 und G_2 keinen K_5 oder $K_{3,3}$. Nach IA sind G_1 und G_2 planar. Betrachte jeweils eine planare Einbettung von G_1 und G_2 mit x und y (bzw Kanze (x,y)) außen liegt. Dann kann man diese Eingbettungen leicht zu einer Einbettung von G zusammenfügen.
$\Rightarrow$ G ist planar.
Fall 2 G ist 3-fach zusammenhängend

Lemma 1 Die planare Einbettung aus 2-fach zusammenhängenden Graphen G ist eindeutig, genau dann wenn G eine Unterteilung eines 3-fach zusammenhängenden Graphen.

Corollar Die planare Einbettung eines 3-fach zusammenhängenden Graphen ist eindeutig.

Lemma 2 Sei G ein 3-fach zusammenhängender Graph mit mindestens 5 Knoten. Dann enthält G eine Kante e, sodass $G|e$ 3-fach zusammenhängend ist.

Lemma 3 Sei e eine beliebige Kante in G. Falls $G|e$ eine Unterteilung des K_5 oder $K_{3,3}$ enthält, dann gilt dies auch für G. Dh Kontraktion einer Kante erzeugt keinen K_5 oder $K_{3,3}$.
IS Sei G ein 3-fach zusammenhängender Graph mit n Knoten ohne K_5 oder $K_{3,3}$. Aus Lemma 2 folgt $\exists$ Kante e, sodass $G|e$ (=G') 3-fach zusammenhängend ist. Aus Lemma 3 folgt G' hat weder K_5 noch $K_{3,3}$. Nach IA ist G' planar. Aus Lemma 1 folgt, G' hat eine eindeutige planare Einbettung. Betrachte diese Einbettung in der Umgebung von z (=Knoten, der aus der Knotraktion von x,y entsteht). Seien $x_1, ..., x_k$ die Nachbarknoten von z, angeordnet im Uhrzeigersinn, gemäß der planaren Einbettung. Ersetze z wieder durch die Kante (x,y) und Konstruiere eine planare Einbettung für G. Sei $x'_1, ..., x'_l$ die Teilfolge der Knoten $x_1, ..., x_k$, die in G benachbart zu x sind.
Fall 1 Alle Nachbarn von y liegen auf einem Face-Segment zwischen x'_i und x'_{i+1} zyklisch und inklusive Ränder. $deg(y) \geq 3$, da G 3-fach zusammenhängend. Dann kann man leicht eine planare Einbettung für G' konstruieren.
Fall 2 Nicht alle Nachbarn von y (außer x) liegen im selben Segment (x'_i, x'_{i+1})
Fall 2.1 y hat ≥ 3 Nachbarn in $x'_1, ..., x'_l$. G enthält eine Unterteilung des K_5 definiert durch die Kanten.

Fall 2.2 y hat einen Nachbarn u in einem Segment $P_i = (x'_i, x'_{i+1}) \backslash (x'_i, x'_{i+1})$ und einem anderen u $\notin P_i$. G enthält eine Unterteilung des $K_{3,3}$

Fall 2.3 y hat 2 Nachbarn in x'_1 und x'_j mit $j \neq i+1$ und $i \neq j+1$ (zyklisch). G enthält eine Unterteilung des $K_{3,3}$

$\Rightarrow$ Es kann nur Fall 1 vorkommen, also ist G planar. $\square$

3.7.1 Beweise der Lemmata

Lemma 1 Die Einbettung eines zweifach planaren Graphen G ist eindeutig $\Leftrightarrow$ G ist eine Unterteilung eines 3-fach zusammenhängenden Graphen.

Proof. Für A $\Leftrightarrow$ B zeige:

1. $\overline{B} \Rightarrow \overline{A}$
2. $\overline{A} \Rightarrow \overline{B}$

1) Sei G 2-fach zusammenhängend, aber keine Unterteilung eines 3-fach zusammenhängenden Graphen. Beobachtung Dann existiert ein Separationspaar x, y mit Split-Graphen G_1 und G_2, sodass weder G_1 noch G_2 ein Pfad ist. Annahme Für alle Separationspaare (x,y) sind diese Splitgraphen Pfade $\Rightarrow$ G ist eine Unterteilung eines 3-fach zusammenhängenden Graphen **Widerspruch zur Annahme**. Betrachte eine planare Einbettung von G und das Separationspaar (x, y) gemäß Beobachtung. Durch Drehung bzw Spiegelung von G_1 oder G_2 in dieser Einbettung erhält man eine neue Einbettung. Ist nur eine Seite eine Pfad, existiert nur eine Einbettung.

2) Sei G 2-fach zusammenhängend und planar mit einer nicht-eindeutigen Einbettung. Dh, es existieren mindestens 2 verschiedene Einbettungen E(G) und E'(G). $\Rightarrow$ es existiert ein Face Zyklus C in E, der nicht in E' existiert. Betrachte E mit C als äußeres Face. C zerlegt G in mindestens 2 Komponenten, die wahlweise auf der einen oder anderen Seite von C eingebetter werden können. Dann existieren 2 Knoten x,y auf C die die beiden Komponenten trennt. $\Rightarrow$ (x,y) ist ein Separationspaar, sodass beide Split-Graphen keine Pfade sind. $\Rightarrow$ G ist keine Unterteilung eines 3-fach zusammenhängenden Graphen. $\square$

Lemma 2 Sein G ein 3-fach zusammenhängender Graph mit mindestens 5 Knoten, dann existiert eine Kante e in G, sodass $G|e$ 3-fach zusammenhängend ist.

Proof. (indirekt)
Annahme Für jede Kante e in G: $G|e$ ist nicht 3-fach zusammenhängend. $\Rightarrow$ $G|e$ enthält ein Separationspaar (x', z). Sei e=(x,y) mit x' als Resultat der Kontraktion von (x,y), Im Originalgraphen G ist $\{x, y, z\}$ eine Separationsmenge. Wähle die Kante e=(x,y) und den Knoten z so, dass eine möglichst große Komponente Komponente H entsteht. Sei H' der Rest der Grpahen (ohne H, x, y, z). Dann betrachte eine Kante e'=(z,v) mit u in H' (existiert, denn sonst wäre (x,y) ein Sep-Paarin G). Auch $G|e'$ ist nicht 3-fach zusammenhängend (nach Annahme) $\Rightarrow$ Es existiert ein Knoten v mit $\{x, y, z\}$ ist Separationsmenge von G. Für v gibt es 3 Möglichkeiten:

1. $v \in H'$
2. v = x. Dann existiert Komponente, die mindestens H und y enthält. Widersprch zur Maximalität con H.
3. v = y. Symmetrisch zu Fall 2 (Widerspruch)

Das sind alle möglichen Fälle. Dh $v \notin H$

3 Für jede Kante e: $G|e$ enthält eine Unterteilung des K_5 oder $K_{3,3} \Rightarrow$ G enhält eine solche Unterteilung.

□

4 Maximal Matchings im bipartiten Graphen

Definition Sei G = (V, E) ein ungerichteter Graph

1. $M \subseteq E$ heißt Matching von G, falls keine 2 Kanten aus M einen gemeinensamen Endpunkt (Knoten) haben.
2. Falls $e \in M$, dann heißt e paarned. Ein Knoten v heißt gepaart, falls er einen Endpunkt einer paarenden Kante ist.
3. Ein Matching M heißt maximal, falls $|M| \geq |M'|$ für alle Matchings M'.

Beobachtung $|M| \leq \lfloor \frac{n}{2} \rfloor \Rightarrow$ Falls $M = \lfloor \frac{n}{2} \rfloor$, dann ist M maximal.

Idee für Algorithmus Konstruiere $M_i, ..., M_l$ von Matchings mit $|M_i| < |M_{i+1}|$ und M_l maximal. Schritt (Erhöhung/Erweiterung) $M \to M'$ mit $|M'| = |M| + 1$.

Definition Ein einfacher Pfad $P = (v_1, v_2), (v_2, v_3), ..., (v_{2k-1}, v_{2k})$ (ungerade Zahl von Kanten) heißt erweiternder Pfad (augmenting path) für ein Matching M, falls

1. v_1 und v_2 sind frei ($\Rightarrow (v_1, v_2), (v_{2k-1}, v_{2k}) \notin M$)
2. Kanten von P sind abwechselnd in M und $E \backslash M$

Beobachtung: Man kann M um 1 vergrößern durch vertauschen von paarenden und nicht paarenden Kanten von P. Für $M = \emptyset$ gilt jede Kante (v,w) ist erhöhender Pfad.

4.1 Lemma 1

Sei M ein Matching und P ein erweiternder Pfad für M, dann ist $M \oplus P = (M \cup P) \backslash (M \cap P)$ (symmetrische Differenz) ein Matching und $|M \oplus P| = |M| + 1$.

Algorithmus 1 : Grundalgorithmus

```
M ← ∅;
while ∃ ein Pfad P für M do
    M ← M ⊕ P;
end
```

Beobachtung: Knotendusjunkte erweiternde Pfade $P_1, ..., P_k$ kommen gelichzeitig behandelt werden: $M \leftarrow M \oplus (P_1 \oplus P_2 \oplus ... \oplus P_H$. Problem: Finde möglichst viele Knotendisjunkte Pfade für ein Matching M in einem Schritt.

4.1.1 Satz 1

Seien M und N Matchings mit $|M| = r$ und $|N| = s$ und $s > r$. Dann enhält $M \oplus N$ mindestens s-r Knotendisjunkte erweiternde Pfade für M.

Proof. $M \oplus N = (M \backslash N) \cup (N \backslash M)$ dh Kanten die entweder in M oder in N sind. Sei $\overline{G} = (V, M \oplus N)$. Da M und N Matchings sind, ist jeder Knoten in $\overline{G}$ Endpunkt höchstens einer Kante aus $M \backslash N$ und höchstens einer Kante aus $M \backslash n \Rightarrow$ Für alle Knoten in $\overline{G}$ gilt, $outdeg_{\overline{G}}(c) \leq 2$ dh $outdeg_{\overline{G}}(c) \in \{1, 2, 3\} \Rightarrow \overline{G}$ zerfällt in Zusammenhangskomponenten der folgenden Form

1. isolierter Knoten
2. Pfade, die abwechselnd Kanten aus $M \backslash N$ und $N \backslash N$ enthalten.
3. Kreis wie 2).

Korollar 1 M ist maximales Matching $\Leftrightarrow$ es existiert kein erweiternder Pfad für M $\square$

Lemma 2 Sei M ein Matching mit $|M| = r$ und s die Größes eines maximalen Matchings mit $s > r$ (dh M ist nicht maximal). Dann gibt es für M einen erweiternden Pfad der Länge $\leq 2 \cdot \lfloor \frac{r}{s-r} \rfloor + 1$.

Proof. Sie N ein max Matching ($|N| = s$)
Aus Satz 1 $\Rightarrow M \oplus N$ enthält mindestends s-r Knotendisjunkte erweiternde Pfade für M. Alle diese Pfade enthalten Zusammen $\leq r$ Kanten aus M (da $|M| = r$).
Verteile r Dinge auf s-r Pfade

$\Rightarrow \exists$ ein Pfad mit $\leq \lfloor \frac{r}{s-r} \rfloor$ Kanten aus M

$\Rightarrow \exists$ ein Pfad P mit $|P| \leq 2 \cdot \lfloor \frac{r}{s-r} \rfloor + 1$

$\square$

Ein erweiternder Pfad P für M heißt kürzester erweiternder Pfad, falls $|P| \leq |P'|$ für alle erweiternden Pfade P' für M.

4.1.2 Satz 2

Sei M ein Matching, P ein kürzester erweiternder Pfad für M und P' ein beliebiger erweiternder Pfad für $M \oplus P$. Dann gilt

$$|P'| \geq |P| + |P \cap P'|$$

Proof. Satz 2
$M \to_P M \oplus P \to_{P'} \underbrace{M \oplus P \oplus P'}_{N}$. N ist ein Matching, $|N| = |M| + 2$. Aus Satz 1 $\Rightarrow M \oplus N$ enthält 2 Knotendisjunkte erweiternde Pfade P_1, P_2 für M.

$$M \oplus N = M \oplus M \oplus P \oplus P' = P \oplus P'$$

$$\Rightarrow |P \oplus P'| = |M \oplus N| \geq |P_1| + |P_2|$$

Außerdem $|P_1| \geq P, |P_2| \geq P$, da P ein kürzester erweiternder Pfad für M ist.

$$\Rightarrow \underbrace{|P \oplus P'|}_{=|P|+|P'|-|P \cap P'| \geq 2 \cdot |P|} \geq |P_1| + |P_2| \geq 2 \cdot |P|$$

$$\Rightarrow |P'| \geq |P| + |P \cap P'|$$

□

4.1.3 Neue Idee für Algorithmus

Berechne Folge von Matchings $M_0, M_1, ..., M_i$ mit

1. $M_0 = \emptyset$
2. $M_{i+1} \leftarrow M_i \cdot P_i$ mit P_i ist ein erweiternder Pfad für M_i

Korollar 2 $|P_i| \leq |P_{i+1}|$, dh die Länge der kürzesten erweiternden Pfade ist in der Folge $P_0, P_1, ...$ monoton wachsend

Korollar 3 Für alle i,j mit $i \neq j$ und $|P_i| = |P_j|$ gilt P_i, P_j sind knotendisjunkt.

4.1.4 Maximal Cardinality Matching Algorithmus

Situation: Folge der kürzesten erhöhenden Pfade: $\underbrace{P_0, P_1}_{\text{Abschnitte gleicher Länge}}, \underbrace{..., ...}, \underbrace{..., P_s}$ (Abschnitt = Phase des Algorithmus)

1. In jedem Abschnitt sind die Pfade knotendisjunkt
2. Anzahl der Abschnitte $\leq 2\lfloor\sqrt{s}\rfloor + 2$, dh $s = O(\sqrt{n})$

Effizienter Algorithmus:

1. Behandlung aller Pfade in einem Abschnitt (einer Phase) in Zeit $O(n + m)$
2. Das muss für $O(\sqrt{n})$ Abschnitte gemacht werden

   → Laufzeit $O(\sqrt{n}nm) = O(\sqrt{n}m)(m \leq n) = O(n^{2,5}(m \leq n^2))$

Im Gegensatz zum trivialen Algorithmus, der jeden Pfad getrennt durch Exploration des Graphen (Tiefen oder Breitensuche) betrachtet. Das führt zu einer Laufzeit von $O(s(n+m)) = O(nm) = O(n^3)$

Zwei Knotenmengen A, B (Seiten des Graphen). Darstellung eines Matchings M in bipartiten Graphen:
Idee $\rightarrow$ gerichteter Graph.

Falls $(v, w) \in M$, dann Richtung von B nach A

Falls $(v, w) \in M$, dann Richtung von A nach B

Beobachtung (in der gerichteten Darstellung): Ein Pfad P ist genau dann ein ehöhender Pfad, wenn er in einem freien Knoten von A startet und in einem freien Knoten von B endet.

Erhöhung $M \oplus P$ in der Darstellung:

1. Drehe alle Kanten von P um
2. Endpunkte als gepaar markieren (nicht mehr frei)

Algorithmus sucht nach maximal vielen kürzesten erhöhenden Pfaden (gleicher Länge) und dreht alle Kanten dieser Pfade um. Wichtig dafür: Pfade sind Knotendisjunkt. Wiederhole, bis keine Pfade mehr existieren.

Kürzeste Pfade: Teile den Graphen in Schichten ein (Levels) ein.

Level 1: alle freien Knoten in A

Level 2: alle von Level 1 über eine Kante erreichbar

Level i: alle von Level i-1 über eine Kante erreichbaren, die nicht in Level i...i-1 enthalten sind.

$\Rightarrow$ Level i $\subseteq$ A, falls i ungerade, Level i $\subseteq$ B, falls i gerade.

Konstruiere den geschichteten Graphen $\overline{G}$

1. füge 2 Knoten s,t hinzu
2. Kanten von s zu allen freien Knoten in A; von allen freien in B zu t.

Dann repräsentiert jeder Pfad von s nach t in $\overline{G}$ einen erhöhenden Pfad in G (1. und letzte Kante weglassen).
Level-Einteilung: Breitensuche die in s startet. Berechnet für jeden Knoten v $dist[v]$ mit $dist[v] = 1 \Leftrightarrow v$ in Level i

Einschub: Breitensuche Startknoten s:

- Felder:
  - dist über die Knoten
  - Init $dist[v] = -1$ für alle $v \in V$
  - Schlange u (FIFO-Queue)

BFS(s)

Algorithmus 2 : BFS(s)

```
dist[s] ← 0 ;
u.append(s);
while not u.empty() do
    v ← u.pop();
    for w ∈ V mit (v, w) ∈ E do
        if dist[w] = −1 then
            dist[w] ← dist[v] + 1;
            u.append(w);
    end
end
```

Laufzeit O(n+m) (n = $\sum_{v \in V}(1 + outdeg(v))$)

Einschub: Tiefensuche DFS zur Berechnung von Pfaden von s aus

- $besucht[v]$ = false für alle $v \in V$
- $pred[v]$ = null

Algorithmus 3 : DFS(v)

```
besucht[v] ← true ;
for w ∈ V mit (v, w) ∈ E do
    if not besucht[w] then
        pred[w] ← (v, w) dfs(w);
end
```

(Aufruf DFS(s)) Darstellung der Pfade: $s \to v$ duch Vorgängerverweise. $pred[v]$: die Kante, (u,v) über die DFS den Knoten v besucht hat. → Pfade können adnn rückwärts durchlaufen werden.

4.1.5 Algorithmus von Hopcroft und Karp

Maximale Matchings in bipartiten Graphen in Zeit $O(\sqrt{n}m)$. Bipartiter Graph G (mit Seiten A, B) → $\overline{G}$

- Füge Knoten s,t ein und verbinde s mit allen Knoten in A und t mit allen Knoten in B. Alle Knoten sind von links nach rechts, A nach B, gerichtet $\Rightarrow M = 0$.
- Jeder Pfad P von s nach t ist ein erhöhender Pfad. $M \oplus P$
  1. Lösche erste und letzte Kante
  2. alle anderen (inneren) Kanten umdrehen
- Kürzeste erweiternde Pfade: Breitensuche **BFS**(s, level)
  - Führt BFS mit Startknoten s aus und berechnet Levels ($dist[v]$, auch als $level[v]$ bezeichnet)
  - $dist[t]$ Länge der aktuell kürzesten erweiternden Pfade
- Ausführung des eigentlichen Algorithmus: **DFS**

Algorithmus

1. Initialisierung: (Konstruiere $\overline{G}$)
   a) Richte alle Kanten von A nach B
   b) Füge die 2 neue Knoten s und t hinzu
   c) Füge Kanten (s,v) für alle $v \in A$ und (u,t) für alle $u \in B$
2. Wiederholte Pfaderhöhung BFS(s, level):
   a) t ist von s aus erreichbar $\Rightarrow \exists$ erhöhende Pfade
   b) Bestimme maximale Menge von Knotendisjunkten Pfaden von s nach t, sodass für alle Kanten (v,w) auf einem Pfad gilt: $level[v] = level[w] - 1$ (Dies sind stets **kürzeste erweiternde Pfade**. Nur Kanten, die)
   c) Sei S die Menge dieser Pfade (jeweils eine Phase):
      i. streiche erste und letzte Kante ($M \oplus P$)
      ii. drehe die Richtung aller anderen Kanten um

Algorithmus 4 : Hopcroft und Karp

```
Richte alle Kanten von A nach B ;
Füge die 2 neue Knoten s und t hinzu ;
Füge Kanten (s,v) für alle v ∈ A und (u,t) für alle u ∈ B;
while level[t] ≠ −1 do
    Bestimme maximale Menge von Knotendisjunkten Pfaden von s nach t,
     sodass für alle Kanten (v,w) auf einem Pfad gilt: level[v] = level[w] − 1;
    for Pfade P ∈ S do
        streiche erste und letzte Kante;
        drehe die Richtung aller anderen Kanten um;
    end
end
```

Für die innere Schleife (Max Menge von Pfaden von s nach t) brauchen wir eine Variante von DFS, die Kanten (v,w) ausfiltert (dh nicht benutzt) mit $level[v] \neq level[w] - 1$.

Algorithmus 5 : dfs-Variante (Kanten im geschichtenten Graphen)

```
besetzt[v] ← true;
for w ∈ V mit (v, w) ∈ R do
    if not besucht[w] ∧ levle[w] = level[v] + 1 then
        pred[w] ← (v, w);
        dfs(w);
end
```

Die Funktion berechnet stet kürzeste Pfade.

Algorithmus 6 : Maximale Menge von kürzesten Pfaden

```
dfs(s, pred);
while pred[t] ≠ null do
    //Iteration über die Kanten (rückwärts);
    u ← t;
    while u ≠ s do
        (x, y) ← pred[u];
        u ← x;
    end
    dfs(s, pred);
end
```

Algorithmus in Zeit $O(n + m)$

Laufzeitanalyse

- Hauptschleife: solange t von s erreichbar: wird höchstens $2\lfloor\sqrt{s}\rfloor+2$ mal ausgefürht. Mögliche verschiedener Läangen kürzester erweiternder Pfade
- Inner Schleife $O(n + m)$: BFS und DFS zur Behandlung aller Pfade

Satz (Hopcroft/Karp) In bipartiten Graphen kann man ein max Matching in Zeit $O(\sqrt{n}m)$ berechnen (Annahme $(m \geq n)$)

Bemerkungen

1. Der Algorihmus funktioniert nicht bei allgemeinen Graphen, da diese zyklisch sein können und erfordert einen komplizierteren Algorithmus.
2. Maximales Matching $M_max \leftrightarrow$ Minimales Vertex Cover C_min.
   - Vertex Cover $C_min \subseteq V$ mit: für alle Kanten $(v, w) \in E$ gilt $v \in C_min$ oder $w \in C_min$
   - Einfache Beobachtung $|C_min| \geq |M_max|$ kein Knoten kann zwei Matching-Kanten überdecken

Satz von König In bipartiten Graphen ist $|M_max| = |C_min|$

5 Übungen

Übung 1:

Übung 2:

Übung 3:

1) Durch entfernen von Kanten soll der Graph zerlegt werden. (Unions in umgekehrter Reihenfolge)

2) Zu zeigen:

$$a(z,n) \leq \lfloor \frac{4m}{n} \rfloor \ f\text{ür } z = \alpha(m,n)$$

Definition von a und α

$$a(z,n) = min\{j | A(z,j) > logn\}$$

$$\alpha(m,n) = min\{i | A(i, \lfloor \frac{4m}{n} \rfloor) > logn\}$$

Behauptung:

$$a(\alpha(m,n),n) \leq \lfloor \frac{4m}{n} \rfloor$$

Beweis: indirekt. Annahme:

$$a(\alpha(m,n),n) > \lfloor \frac{4m}{n} \rfloor$$

$$\Rightarrow A(\alpha(m,n), \lfloor \frac{4m}{n} \rfloor) \leq logn$$

Widerspruch zur Definition von α , denn

$$A(\alpha(m,n), \lfloor \frac{4m}{n} \rfloor) > logn$$

3.a) Union-Split-Find. (van Emde-Boas aht Datenstruktur mit log log n für Union-Split-Find.) Gegeben ist eine Array

- Split(i): Markiere i
- Find(x): Finde nächste Markierung
- Union(x): Lösche Markierung x

Balancierter (blatt-orientierter) Baum zur Speicherung der markierten Elemente. Einfügen der markierten Elemente als Blätte rdes Baums

- Split = Insert
- Union = Delete
- Find = Locate

Platz = #Intervalle, Zeit O(logn)

3.b)

- Insert = Split
- Delete = Union
- FindMin = Find(1)

Übung 4:

1) Rekursive Funktion zum Aufbau eines RST
A[1,...,n] von Schlüsseln. P[1,...,n] Prioritäten

---

for $1 \leftarrow 0$ **to** n **do**
  $P[i] \leftarrow random()$

Rekursice Funktion RST(A,P,l,r) baut einen RST für A[l,...,r] und liefert Pointer auf die Wurzel.

```
class rst_node {
    key
    prio
    left,right: rst_node
}
```

---

def *RST(A,P,l,r)*:
  if $l > r$ **then**
    return null;
  $i \leftarrow max[P, l, r]$;
  $q \leftarrow$ new rstnode ;
  $q.key \leftarrow$ A[i] ;
  $q.prio \leftarrow$ P[i] ;
  $q.left \leftarrow$ RST(A,P,l,i-1) ;
  $q.right \leftarrow$ RST(A,P,i+1,r) ;
  return q;

---

2) Spiel B symmetrisch zu Spiel A.

3) Implementiertung von Hashing mit Verkettung. Idee: Tafelgröße s beliebig. Hashfunktion $h(x) = x \cdot mods$

---

def *Insert(x)*:

def *Lookup(x)*:

def *Delete(x)*:

---

4) Belegungsfaktor $\beta = \frac{n}{m}$ m = Tafelgröße. Bei Hashing mit Verkettung ist β = erwartete Länge einer Liste. Laufzeit für eine Operation $O(1+\beta) = O(1)$ für $\frac{1}{2} \leq \beta \leq 2$

Rehash Die Tabelle muss in eine größere Liste kopiert werden

---

def *Insert(x)*:
- ... ;
- $n \leftarrow n + 1$;
- **if** $\frac{n}{m} > 2$ **then**
  - $m_0 \leftarrow m$;
  - $m \leftarrow 2m$;
  - $T' \leftarrow new\ int[m]$
- Kopiere alte Tabelle in neue ;

def *Delete(x)*:
- $n \leftarrow n - 1$ **if** $\frac{n}{m} < \frac{1}{2}$ **then**
  - ...;
  - $m \leftarrow \frac{m}{2}$;
  - Kopiere alte Tabelle in neue ;

5.1 Übung 5

5.2 Übung 6

5.3 Übung 7

5.4 Übung 8

2) Belegungsfaktor = $\beta = \frac{n}{k}$. Ziel ist $\frac{1}{2} \leq \beta \leq 2$. Erwartete Laufzeit pro Operation $O(1 + \beta)$. Rehashing: Test nach Inerst/Delete. Wenn β außerhalb des Zielintervalls, Rehash mit $k \leftarrow n$.
Potentialmethode: $pot : D \rightarrow \mathbb{R}_0^+$. Folge von Insert, Rehash mit pot = 2 · Anzahl der Markierten

1. Rehash löscht alle Markierungen
2. Insert markiert neues Elemenent

pot = $2 \cdot$ Anzahl der Eingefügten + $1 \cdot$ (Anzahl der Gelöschten seit dem letzten Rehash)

- $T_{amontisiert}(INSERT) = O(1+\beta) + 2$
- $T_{amontisiert}(INSERT) = O(1+\beta) + 1$
- $T_{amontisiert}(INSERT)$
  - Verdopplung: $O(n) - 2 \cdot neue$
  - Halbierung: $-n$

3) Darstellungen von planaren Einbettungen:

- Face Zyklen. Auflistung der Knoten eines Faces
- Nachbarn jedes Knotens. Sortierung der Adjazenslisten

Implementierung als doppelt verkettete Listen.

5.5 Übung 9

1) G planarer Graph mit $n \geq 3$. Sei d der maximale Grad eines Knotens und n_i = Anzahl von Knoten vom Grade i. Dann gilt

$$5n_1 + 4n_2 + 3n_3 + 2n_4 + n_5 \geq n_7 + 2n_8 + ... + (d-6)n_d + 12$$
$$0 \geq 5n_1 - 4n_2 - 3n_3 - 2n_4 - n_5 + n_7 + 2n_8 + ... + (d-6)n_d + 12$$
$$\sum_{i=1}^{d} (i-6) \cdot n_i + 12 \leq 0$$
$$\underbrace{\sum_{i=1}^{d} i \cdot n_i}_{2m} - \underbrace{\sum_{i=1}^{d} 6n_i}_{6n} + 12 \leq 0$$
$$m - 3n + 6 \leq 0$$
$$m \leq 3n - 6$$

2) Für jede Kante e=(x,y) gilt: $G|e$ enthält eine Unterteilung des K_5 oder $K_{3,3}$, dann enthält auch G eine solche Unterteilung. Sei H die Unterteilung des K_5 bzw $K_{3,3}$ in $G|e$.
Fall 1: $z \in H$ und $deg_H(z) = 0$, dann existiert H auch in G.
Fall 2: $z \in H$ mögliche Grade $deg_H(z) \in \{2, 3, 4\}$
Fall 2.1: $deg_H(z) = 2$ (z liegt auf einer Kante) G $\Rightarrow$ G enthält H der gleichen Art
Fall 2.2: $deg_H(z) = 3$ z ist Ende eines $K_{3,3}$ Fall 2.3: $deg_H(z) = 4$ z ist Ende eines K_5

1. Hat x eine Kante in H, so ist x ein Subdivision-Knoten (symmetrisch für y)
2. Hat x 2 Kanten in H, so bilden x und y einen $K_{3,3}$

6 Allgemeines

6.1 Einschub: Erwartungswerte

Situation: n Ereignisse, die mit einer gweissen Wahrscheinlichkeit prob(i) auftreten. Jedes Ereignis besitzt einen Wert val(i).

$$E(val) = \sum_{i_1}^{n} prob(i) \cdot val(i)$$

Spezialfall: Gleichverteilung: $prob(i) = \frac{1}{n} f\ddot{u}r\ 1 \leq i \leq n$. Dann gilt:

$$E(val) = \frac{1}{n} \sum_{i=1}^{n} val(i) = Mittelwert$$

6.2 Integrierende Reihe

$x \leq 1$

$$\sum_{i=0}^{\infty} x^i \cdot i = \frac{x}{(1-x)^2}$$

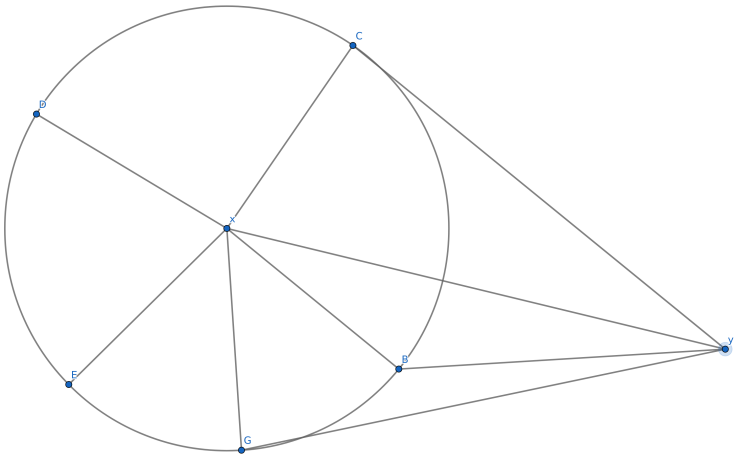

Figure 8: Beweis Satz von Kuratowski: Fall 2.1

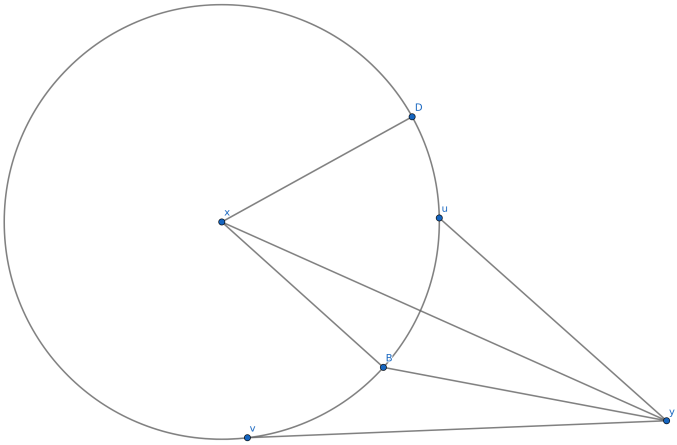

Figure 9: Beweis Satz von Kuratowski: Fall 2.2

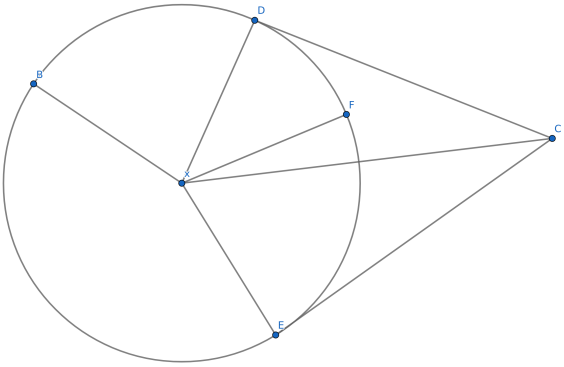

Figure 10: Beweis Satz von Kuratowski: Fall 2.3