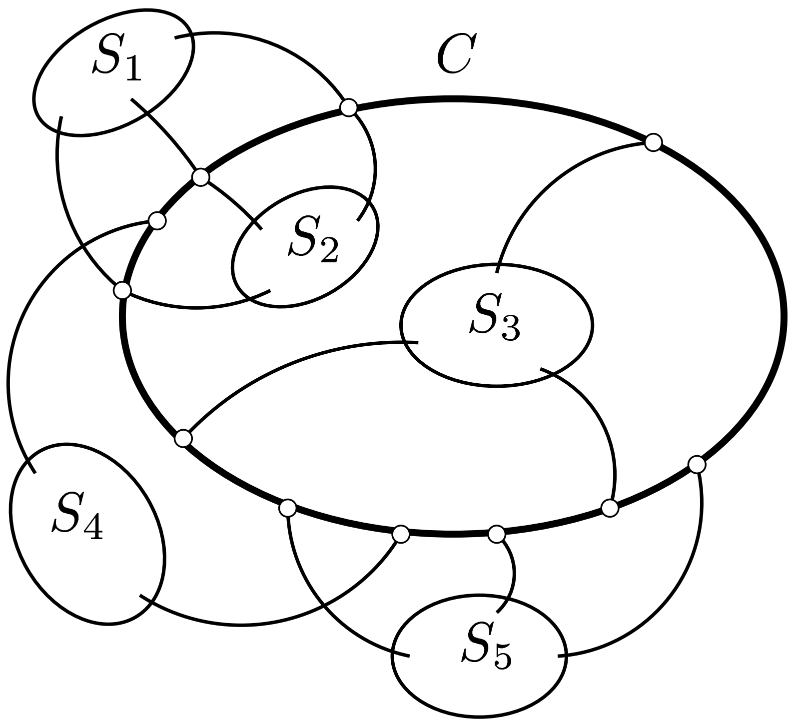


S_1 und S_2 überlappen,

S_3 und S_5 sowie S_4 und S_5 überlappen,

S_1 und S_3 sowie S_1 und S_5 überlappen nicht

Idee für einen Planaritätstest

- Wähle Kreis C beliebig, bestimme die Segmente S_i und zugehörige Pfade P_i.
- Verifiziere rekursiv, daß $S_i + P_i$ (hat weniger Kanten als G!) planar ist und es eine Einbettung gibt, in der P_i außen liegt.
- Bestimme den Überlappungsgraphen UG der Segmente S_i:

Knoten von UG	$\hat{=}$	Segmente S_i
Kanten von UG:		S_i und S_j werden genau dann durch eine Kante verbunden, wenn S_i und S_j überlappen.

- Verifiziere, ob UG bipartit ist. Ist dies der Fall, so ergeben sich aus den beiden bipartiten Knotenmengen von UG die Segmente, die innen bzw. außen eingebettet werden müssen, und G ist damit planar.

Verbesserte Vorgehensweise

Idee: Sorge dafür, dass Segmente in der “richtigen” Reihenfolge ausgegeben werden.

- Bestimme einen DFS-Baum für G und berechne dabei auch die Felder

$$\begin{aligned}
\text{dfs}[v] &= \text{dfs-Nummer, wie bisher} \\
\text{low}[v] &= \min\{\text{dfs}[v],\ \text{dfs}[w] \mid w \text{ durch} \geq 0 \text{ Baumkanten abwärts und genau eine} \\
&\qquad \text{Rückwärtskante von } v \text{ aus erreichbar}\} \\
\text{low2}[v] &= \min\{\text{dfs}[v],\ \text{dfs}[w] \mid \text{dfs}[w] \neq \text{low}[v], w \text{ durch} \geq 0 \text{ Baumkanten abwärts} \\
&\qquad \text{und genau eine Rückwärtskante von } v \text{ aus erreichbar}\}
\end{aligned}$$

$\Rightarrow$ Laufzeit $\mathcal{O}(n)$ [wegen $m \leq 3n-6$ ist $\mathcal{O}(m) = \mathcal{O}(n)$ bei planaren Graphen]

- Richte Kanten:
  - Baumkanten abwärts
  - Rückwärtskanten aufwärts
- Gewichte die Kanten folgendermaßen:

$$c(v,w) = \begin{cases} 2 \cdot \text{dfs}[w] & \text{falls } (v,w) \text{ Rückwärtskante} \\ 2 \cdot \text{low}[w] & \text{falls } (v,w) \text{ Baumkante und low2}[w] \geq \text{dfs}[v] \\ 2 \cdot \text{low}[w] + 1 & \text{falls } (v,w) \text{ Baumkante und low2}[w] < \text{dfs}[v] \end{cases}$$

- Erzeuge für alle $v \in V$ eine Adjazenzliste der ausgehenden Kanten, *aufsteigend sortiert* nach deren Gewicht $c(.)$.

  $\Rightarrow$ Laufzeit $\mathcal{O}(n)$ [Beweiseidee: Buckets!]

- Markiere jede Kante als “neu” (d.h. unbetrachtet).

- Bestimme einen Kreis C:

  $v :=$ ⟨Wurzel des DFS-Baumes⟩;
  $w :=$ erster Nachbar von v in der Adjazenzliste von v;
  markiere (v, w) als “benutzt”; $C := \{(v, w)\}$;
  while ($w \neq$ ⟨Wurzel des DFS-Baumes⟩)
  $\{$
  $\quad v := w$;
  $\quad w :=$ Endknoten der ersten “neuen” Kante in Adjazenzliste von v;
  $\quad$ markiere (v, w) als “benutzt”; $C := C + (v, w)$;
  $\}$
  Gib Kreis C aus;

$\Rightarrow C$ besteht aus Baumkanten und genau einer Rückwärtskante (Beweis ÜA).

- Als nächstes werden die Segmente mit folgendem Algorithmus in “kleinen Teilen” ausgegeben, die im weiteren als *Pfade* bezeichnet werden sollen:

$v :=$ ⟨Knoten mit größter dfs-Nummer in C⟩;

// dies ist Knoten v am Ende des vorigen Algorithmus

while ($v \neq$ ⟨Wurzel des DFS-Baumes⟩)

    while (es gibt “neue” Kanten in Adjazenzliste von v)

        $w :=$ Endknoten der ersten “neuen” Kante

            in Adjazenzliste von v;

        markiere (v, w) als “benutzt”; $P := \{(v, w)\}$;

        while (dfs$[v] <$ dfs$[w]$)

            // (v, w) ist eine Baumkante

            $v := w$;

            $w :=$ Endknoten der ersten “neuen” Kante in

                Adjazenzliste von v;

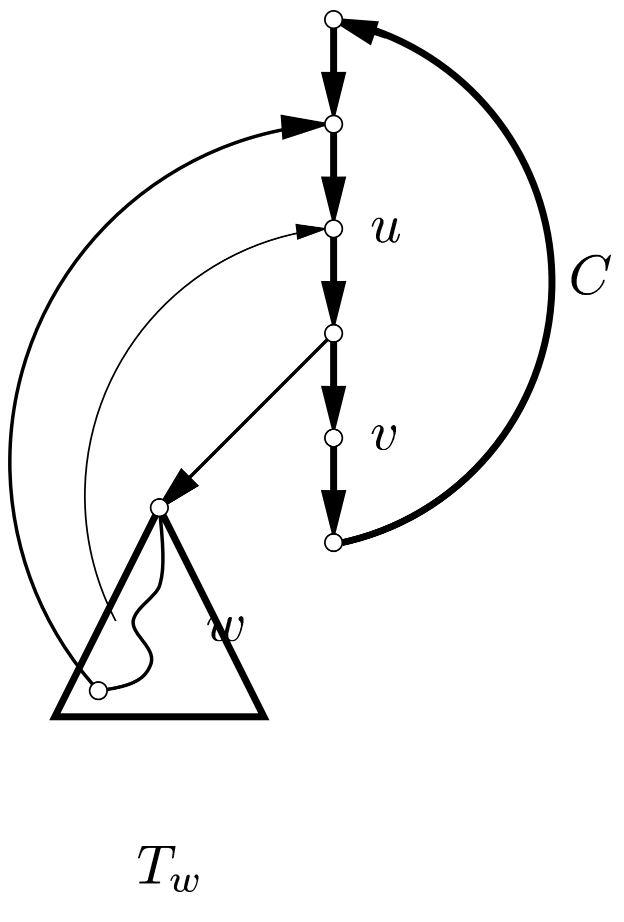
            markiere (v, w) als “benutzt”; $P := P+(v, w)$;

        Gib Pfad P aus;

    $v :=$ Vorgänger von v;

Bemerkung: Sei S ein Segment bezüglich C und v eine Verankerung von S mit größter DFS-Nummer Dann treffen folgende Beobachtungen zu:

- S wird über eine Baumkante (v, w) von v aus betreten und vollständig ausgegeben, bevor der Algorithmus S wieder verläßt.
- S besteht aus dem Subbaum T_w des DFS-Baumes und allen von dort ausgehenden Rückwärtskanten.
- Sei u der Endknoten des ersten ausgegebenen Pfades. Dann enthält S nur Verankerungen aus P_{uv}, wobei P_{uv} der Pfad des DFS-Baumes (bzw. des Kreises C) ist, der u und v verbindet.



Lemma: Seien $S_1, \ldots, S_\ell$ die ersten ℓ Segmente, die vom Algorithmus ausgegeben werden und $P_\ell = (v, \ldots, u)$ der erste Pfad von S_ℓ, der ausgegeben wird. Weiter sei eine Einbettung von $C + S_1 + \cdots + S_{\ell-1}$ gegeben. Dann gilt:

S_ℓ läßt sich genau dann nicht außen (innen) hinzufügen, wenn es ein außen (innen) eingebettetes Segment S_i mit $i \in \{1, \ldots, \ell - 1\}$ gibt, das eine Verankerung zwischen u und v besitzt.

Beweis: siehe z.B. Skript SS97

Verbesserte Vorgehensweise (Fortsetzung)

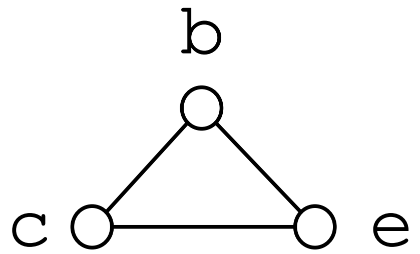
Der Algorithmus “erkennt”, wenn der erste Pfad eines Segmentes ausgegeben wird. Daraufhin schickt er den derzeitigen Stand (die derzeitige Einbettung) auf einen Stack und macht rekursiv mit dem gerade ausgegebenen Pfad zusammen mit einem Teil des alten Kreises als neuem Kreis weiter.

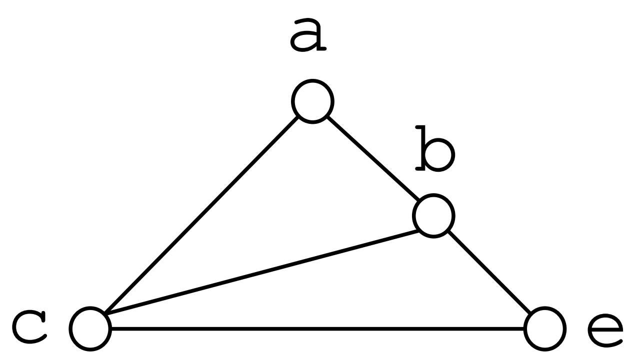
Es werden folgende Datenstrukturen verwendet:

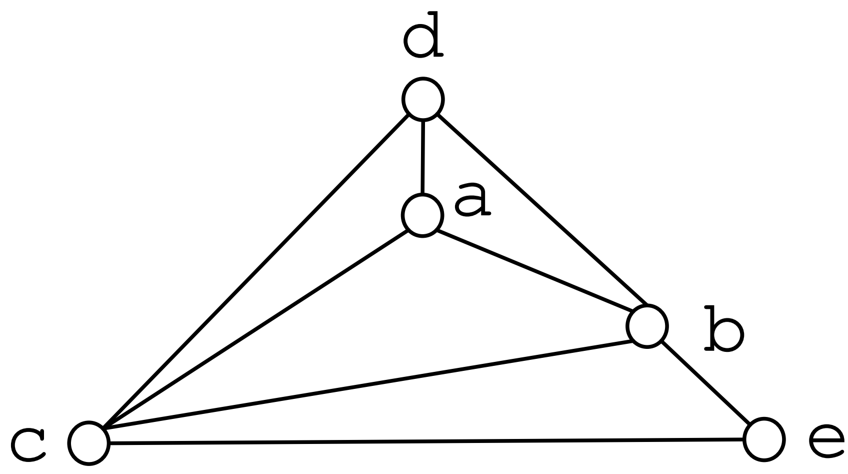
L_i : Liste der Pfade, die innen eingebettet werden

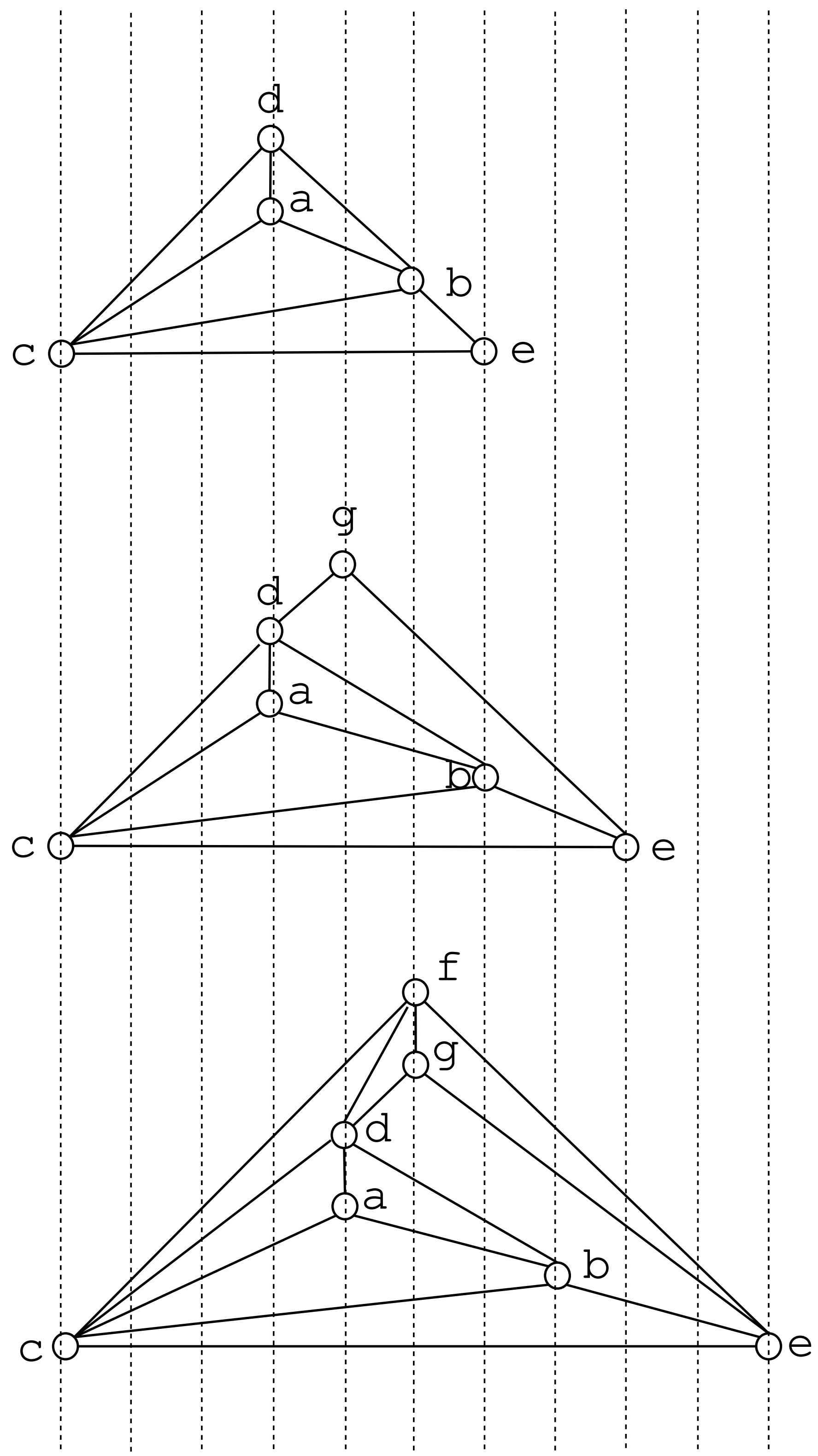
L_o : Liste der Pfade, die außen eingebettet werden

Diese Listen sind aufgeteilt in Blöcke, für die die Einbettung eines Pfades bereits die Einbettung aller übrigen vorschreibt. Durch diese Form der Organisation kann für jeweils eine ganze Gruppe von Pfaden das Ändern des Ortes der Einbettung von innen nach außen und umgekehrt (“flippen”) durch Pointer-Manipulationen in konstanter Zeit bewerkstelligt werden. Details siehe z.B. Mehlhorn, *Data structures and algorithms 2: Graph algorithms and NP-Completeness*, kanonische Ordnung: c,e,b,a,d,g,f








d
a
b
c
e
g
d
a
b
c
e
f
g
d
a
b
c
e