Assignment 2
Advanced Algorithms & Data Structures PS

Christian Müller 1123410
Daniel Kocher, 0926293

March 15, 2016

Theorieaufgabe

Für den Divide-&-Conquer Algorithmus zur Berechnung von Segmentschnittpunkten werden zuerst zwei Datenstrukturen definiert, welche verwendet werden um die Elemente der Liste S darzustellen.

Endpunkt Stellt einen Endpunkt eines horizontalen Segments dar. Jeder Endpunkt besitzt mindestens drei Felder:

- x ... repräsentiert die x-Koordinate des Endpunktes.
- y ... repräsentiert die y-Koordinate des Endpunktes.
- ass ... Pointer zum zugehörigen Partner-Endpunkt.

Beispiel: Das horizontale Segment mit den Endpunkten $(1, 2)$ und $(4, 2)$ wird durch zwei **Endpunkte** $e_{1,2}$ repräsentiert. Die Felder von $e_{1,2}$ sind wie folgt belegt:

e_1	$.x = 1$	$.y = 2$	$.ass = e_2$
e_2	$.x = 4$	$.y = 2$	$.ass = e_1$

VertSegment Stellt ein vertikales Segment dar. Jedes vertikale Segment besitzt mindestens drei Felder:

- x ... repräsentiert die x-Koordinate des vertikalen Segments.
- y_{unten} ... repräsentiert die untere y-Koordinate des vertikalen Segments.
- y_{oben} ... repräsentiert die obere y-Koordinate des vertikalen Segments.

Beispiel: Das vertikale Segment mit den Endpunkten $(1, 6)$ und $(1, 1)$ wird durch ein **VertSegment** v repräsentiert. Die Felder von v sind wie folgt belegt:

v	$.x = 1$	$.y_{unten} = 1$	$.y_{oben} = 6$

Im folgenden sollen mit mit $L(S_i)$, $R(S_i)$ und $V(S_i)$ die verwendeten einfach verkettete Listen bezeichnet werden. $L(S_i)$ enthält die y-Koordinaten aller linken Endpunkte in S_i, deren rechter Partner (ass) nicht in S_i enthalten ist. Analog beinhaltet $R(S_i)$ die y-Koordinaten aller rechten Endpunkte in S_i, deren linker Partner (ass) nicht in S_i liegt. $V(S_i)$ hingegen enthält die y-Intervalle (y_{unten} bis y_{oben}) der vertikalen Segmente in S_i.

$L(S_i)$ und $R(S_i)$ sind nach steigenden y-Koordinaten sortierte, einfach verkettete Listen. $V(S_i)$ sind nach steigenden unteren Endpunkten (y_{unten}) sortierte, einfach verkettete Listen.

Seien l, l_i einfach verkettete Listen und d ein Wert gespeichert in einer einfach verketteten Liste. Solche Listen haben mindestens folgende Felder:

- $head$ ... Erster Knoten einer Liste.

Weiters stellen sie folgende Funktionen zur Verfügung:

- l_1.insertAll(l_2) ... Fügt alle Element von Liste l_2 zu Liste l_1 hinzu.
- l.search(d) ... Durchsucht die Liste l nach einem Element d und gibt dessen Knoten zurück (falls gefunden).
- l.insert(d) ... Fügt das Element d zu Liste l hinzu.
- l.delete(d) ... Entfernt das Element d aus Liste l.

Knoten einer solchen Liste stellen mindestens die folgenden Felder bereit:

- $next$ ... Nächster Knoten.

Input : Liste S bestehend aus vertikalen Segmenten (**VertSegment**) und Endpunkten (**Endpunkt**) von horizontalen Segmenten.
S sei nach horizontalen Koordinaten sortiert (Algorithmus ist leicht zu adaptieren falls nicht).
Listen $L(S)$, $R(S)$ und $V(S)$, welche beim ersten (initialen) Aufruf der Funktion leer sind.
Im Laufe der Rekursion werden diese drei Listen nach und nach mit Elementen befüllt.

Output: Alle Schnittpunkte von vertikalen Segmenten mit horizontalen Segmenten

```
1  Function ReportCuts(S, L(S), R(S), V(S))
2      if |S| ≤ 0 then return;
3      if |S| = 1 then
           // Initialisierung von L(S), R(S) und V(S)
4          Sei s das einzige Element in S;
5          if s ist linker Endpunkt then L(S) ← {s}, R(S) ← ∅, V(S) ← ∅;
6          else if s ist rechter Endpunkt then L(S) ← ∅, R(S) ← {s}, V(S) ← ∅;
7          else if s ist vertikales Segment then L(S) ← ∅, R(S) ← ∅, V(S) ← {s};
8          return ;                                                  // Rekursionsende
9      end
10     if S ist nicht bezüglich x-Koordinaten sortiert then Sortiere S bezüglich x-Koordinaten;
       /* Divide Schritt.
          m sei ein Integer (repräsentiert die Mitte der Liste S).
          S1 und S2 seien Listen von vertikalen Segmenten und Endpunkten von horizontalen
          Segmenten, sortiert bezüglich ihrer x-Koordinaten.                                  */
11     if |S| ist gerade then m ← |S|/2;
12     else m ← ⌈|S|/2⌉;
13     S1 ← Liste der ersten m Elemente aus S;
14     S2 ← Liste der letzten (|S| − m) Elemente aus S;
       // Conquer Schritt
15     ReportCuts(S1, L(S1), R(S1), V(S1));
16     ReportCuts(S2, L(S2), R(S2), V(S2));
       /* L(Si), R(Si), V(Si) für i = 1, 2 bekannt ⇒ Merge Schritt
          Berichte Segmentschnittpunkte (Paare (h, v))                                        */
17     h1 ← R(S2);
18     DeletePartnersOf(h1, L(S1));
19     IntersectAndReport(h1, V(S1));
20     h2 ← L(S1);
21     DeletePartnersOf(h2, R(S2));
22     IntersectAndReport(h2, V(S2));
       // Aktualisiere L(S), R(S) und V(S) für S = S1 ∪ S2
23     L(S).insertAll(h2));
24     L(S).InsertAll(L(S2));
25     R(S).insertAll(h1);
26     R(S).InsertAll(R(S1));
27     V(S).insertAll(V(S1));
28     V(S).InsertAll(V(S2));
```

Input : Zwei einfach verkettete Listen l_1 und l_2.
$l_{1,2}$ beinhalten nur Endpunkte von horizontalen Segmenten.
Output: Eine Liste, welche alle Elemente aus l_1 beinhaltet, die keinen Partner in l_2 besitzen.

```
1  Function DeletePartnersOf(l_1, l_2)
2      if l_2 ist leer then return l_1;
3      current ← l_2.head;
4      while current ≠ null do
5          if l_1.search(current.ass) hat einen Partner gefunden then l_1.delete(current) ;
6          current ← current.next;
7      end
8      return l_1;
```

Input : Zwei einfach verkettete Listen h und v.
h enthält Endpunkte von horizontalen Segmenten.
v enthält vertikale Segmente.
h und v sind aufsteigend gemäß y bzw. y_{unten} sortiert.
Output: Berichtet alle Schnittpunkte zwischen Elementen aus v und Elementen aus h.

```
1  Function IntersectAndReport(h, v)
       // Keine Schnittpunkte möglich
2      if h ist leer oder v ist leer then return;
3      currHor ← h.head;
4      currVert ← v.head;
5      while currHor ≠ null und currVert ≠ null do
6          if currHor.y > currVert.y_oben then currVert = currVert.next;
7          if currHor.y ≥ currVert.y_unten then
8              print(currVert.x, currHor.y);
9              tmpHor ← currHor.next;
10             while tmpHor ≠ null und tmpHor.y < currVert.y_oben do
11                 print(currVert.x, tmpHor.y);
12                 tmpHor ← tmpHor.next;
13             end
14             currVert ← currVer.next;
15         else
16             if currVert ≠ null then currHor ← currHor.next;
17         end
18     end
```