Übungsblatt 13

Fröhlich, Tenten, Lehmann

13.1: Triangulationen ineinander überführen

Seien T_1 und T_2 zwei beliebige Transformationen von P. Wie in der Vorlesung gezeigt wurde, lässt sich jede Triangulation von Punkten in allgemeiner Lage in eine eindeutige Delauney-Triangulation überführen. Da die Punkt aus P in allgemeiner Lage sind, gilt dies auch für T_1 und T_2.

Seien die Folgen $(e_{1,1}, e_{1,2}, \ldots, e_{1,j})$ und $(e_{2,1}, e_{2,2}, \ldots, e_{2,k})$ die Kanten, welche für T_1 bzw. T_2 geflipt werden müssen, um die Delauney-Triangulation T_d zu erreichen. Da ein beliebiger Flip durch den Flip derselben Kante wieder rückgängig gemacht werden kann, lässt sich auch eine Folge von Flips wieder Rückgängig machen, indem die Kanten in der umgekehrten Reihenfolge geflipt werden.

Demnach ist die Folge $(e_{1,1}, e_{1,2}, \ldots, e_{1,j}, e_{2,k}, e_{2,k-1}, \ldots e_{2,1})$ eine Transformation von T_1 nach T_2. $\square$

13.2: Knotengrad in Triangulation

Betrachte die eulersche Polyederformel ohne die äußere Fläche. Da $c = 1$, gilt $v + f = e + 1$.

Für eine Triangulation T ist die Anzahl der zu einer inneren Fläche inzidenten Kanten genau gleich 3.

Da eine Kante zu höchstens zwei inneren Flächen inzident ist und mindestens 3 Kanten an der äußeren Fläche liegen, gilt $3f \leq 2e - 3 \iff f \leq \frac{2}{3}e - 1$

Also gilt

$$\begin{aligned} & v = e - f + 1 \\ \iff & v \geq e - \left(\frac{2}{3}e - 1\right) + 1 \\ \iff & v \geq \frac{1}{3}e \end{aligned}$$

Das heißt, es gibt höchstens dreimal so viele Kanten wie Knoten.
Durch jede vorhandene Kante erhöht sich die Summe aller Out-Grade um genau 2.

Das heißt:

$$\frac{1}{v}\sum_{i=1}^{v} d_i = \frac{2e}{v} \leq \frac{2 \cdot 3v}{v} = 6.$$

$\square$

13.3: Warenlager

```
def Warenlager(Höhe h, Produkttypen p):

        G[j] := minimales Gewicht für Höhe j
        G[i < 0] := inf
        T[j] := welchen Produkttypen soll als erstes bei Höhe j gewählt werden

        G[0] = 0
        for (j from 1 to h):
                G[j] = inf

         for (i from 1 to n):
                for (j from 1 to h):
                        if (G[j - p[i].h] + p[i].w < G[j]):
                                G[j] = G[j - p[i].h] + p[i].w
                                T[j] = i


        if (G[h] = inf):
                return "Keine Lösung möglich"

        return (G, T)
```

Rekonstruktion der Lösung:

```
def Reconstruct(Höhe h, Produkttypen p, Typ-Map T):
        j = h
        while (j > 0):
                Nehme Kiste von Typ p[T[j]]
                j -= p[T[j]].h
```

Laufzeit:

- Erste For-Schleife: $\mathcal{O}(h)$
- Zweite For-Schleife: $\mathcal{O}(nh)$ (inneres if-Statement: $\mathcal{O}(1)$)
- Rest: $\mathcal{O}(1)$
  $\rightarrow$ Laufzeit gesamt: $\mathcal{O}(nh)$

Der Algorithmus orientiert sich am Zuschnittproblem aus der Vorlesung. Insbesondere gilt folgende Abwandlung des Lemma 2.10:
$\forall j \in \{1, \ldots, h\} : G[j] = \min\{p[i].w + G[j - p[i].h] \mid i \in \{1, \ldots, n\}\}$

Wie genau man für diese iterative Lösung eine Rekursionsgleichung definiert, weiß ich leider nicht.