Algorithmik Blatt 5 Teil 2

Mtr.-Nr. 6329857

Universität Hamburg — 21. November 2019

Aufgabe 14

Aufgabe 15

Zuerst wird ein Graph G konstruiert in welchem die europäischen Städte die Knoten V bilden und die sie verbindenen StraSSen die entsprechenden Kanten E. Die Kanten werden mit der Länger der StraSSen gebildet. AuSSerdem werden auch die Häfen als Knoten hinzugefügt, sowie die StraSSen die von ihnen zu Städten führen als Kanten. Wir nehmen an diese Konstruktion ist in $\mathcal{O}(|E|+|V|)$ realisierbar, da jede Kante und jeder Knoten in eine Datenstruktur mit bereits bekannter maximaler Größe hinzugefügt werden muss.

Wir gehen davon aus, dass der resultierende Graph G zusammenhängend ist und dass er keine negativen Kanten enthält. Daher können wir aus ihm einen minimalen Spannbaum konstruieren. Mit dem Algorithmus von Kruskal lässt sich der minimale Spannbaum in $\mathcal{O}(|E|\log(|V|))$ konstruieren.

Da die Häfen nicht miteinander verbunden sind, jede Stadt aber mit mindestens einem Hafen verbunden ist (Zusammenhang), sind genau die Hafenknoten Blätter im Spannbaum.

Als nächtes durchlaufen wir den Baum Ebene für Ebene beginnen bei den Blättern (Häfen) und weisen dabei jedem Knoten den, nähesten Blattknoten, sprich den besten Vorgänder (Pre), und die Distanz zu diesem zu. Das passiert wie folgt:

1. Alle Blattknoten bekommen sich selbst als besten Vorgänger zugewiesen $\text{Pre}(k) = k$
2. Alle Blattknoten k bekommen die Distanz $D(k) = 0$
3. Alle inneren Knoten k bekommen initial die Distanz $D(k) = \infty$
4. Alle inneren Knoten k bekommen initial als besten Vorgänger $\text{Pre} = \text{nil}$
5. Wird ein Knoten q über eine Kante (p,q) Besucht, prüfe ob $W(p,q)+D(p) < D(q)$, wenn ja aktualisiere $\text{Pre}(q) = p$ und $D(q) = W(p,q)+D(p)$
6. Wenn für einen Knoten der beste Vorgänger und die niedrigste Distanz zu diesem bestimmt ist, kann das Verfahren von diesem Knoten aus wiederholt werden.

Die Intuition hinter diesem Verfahren ist, an jedem Hafen gleichzeitig einen Radius von 0 an mit gleicher Geschwindigkeit wachsen zu lassen und jede Stadt dem Hafen zuzuordnen, dessen Radius sie zu erst erreicht. In einem diskreten Verfahren auf unserem Graphen muss dieses gleichzeitig Wachstum aber explizit sichergestellt werden. Wir wollen zudem sicherstellen, dass jede Kante nur einmal besucht werden muss.

Um das gleichzeitige Wachstum zu erreichen, ordnen wir die bereits erreichten Knoten (zu Beginn alle Häfen) in einer Warteschlange nach ihrer bisher besten Distanz. Zu einem Knoten, der den Kopf der Schlange erreicht, kann es keinen kürzeren Pfad von einem Hafen aus geben, weil alle anderen Knoten in der Schlange bereits eine grösSere Distanz zum nächstens Hafen haben. Alle Knoten die noch nicht in der Schlage sind, haben mindestens eine beste Disanz so groSS wie die Knoten, die schon in der Schlange sind. Daraus ergibt sich, dass wenn wir eine Kante (p,q) Benutzen, wir diese nicht noch ein weiteres mal betrachten müssen, weil sich die beste Distanz von p nicht mehr ändert und diese beste Distanz von q aus Sicht dieser Kante auch nicht mehr verbessern kann. Die beste Disanz von q kann nur dadurch besser werden, dass wir über eine andere Kante (r,q) nach q gelangen. Dafür muss r eine kürzere beste Distanz haben als q, sodass wir auch sicher sein können eine solche Kante benutzt zu haben bevor q an den Kopf der Warteschlange rutscht.

Bei jedem Benutzen einer Kante müssen wir potentiell die beste Distanz des Zielknotens aktualisieren und auch die Position des Knotens in der Warteschlange aktualisieren. Der Aufwand dafür ist in $\mathcal{O}(\log(l))$, wobei l die Länge der Warteschlange, also maximal die Anzahl der Knoten ist. Daraus ergibt sich als Kosten für das Durchlaufen des Baums $\mathcal{O}(|E|\log(|V|))$

Da wir uns in einem Baum bewegen, und von jedem Knoten aus alle nicht benutzten Kanten benutzen, können wir sicher sein, am Ende alle Kanten benutzt und alle Knoten besucht zu haben. Also hat auch jeder Knoten seinen besten Hafen und die kürzeste Distanz zu diesem zugewiesen bekommen.

```
1:  procedure ClosestTreeLeafs(E, V, W)
2:      (E', V') = Kruskal(E, V)            // O(|E|log(|V|))
3:      queue = initQueue()
4:      bestLeaf = InitArray(|V|)
5:      bestDistance = InitArray(|V|)
6:      for n = 0 to |V| do                 // O(|V|)
7:          if deg(V[i]) == 1 then          // Leaf Node
8:              bestLeaf[i] = i
9:              bestDistance[i] = 0
10:             insertQueue(i, queue, bestDistance[i])      // O(1), weil bestDistance[i] = 0
11:         else        // Inner Node
12:             bestLeaf[i] = nil
13:             bestDistance[i] = ∞
14:     while currentNode = extractMin(queue) do        // Zeile 14 und 15 gesamt O(|E|)
15:         while targetNode = selectOutgoingEdge(E', currentNode) do
16:             removeEdge(E', currentNode, targetNode)
17:             newDistance = bestDistance[currentNode] + W((currentNode, targetNode))
18:             if bestDistance[targetNode] == ∞ then
19:                 bestDistance[targetNode] = newDistance
20:                 insertQueue(queue, targetNode, newDistance)      // O(log(|V|))
21:             else if newDistance < bestDistance[targetNode] then
22:                 bestDistance[targetNode] = newDistance
23:                 decreaseKey(queue, targetNode, newDistance)      // O(log(|V|))
```