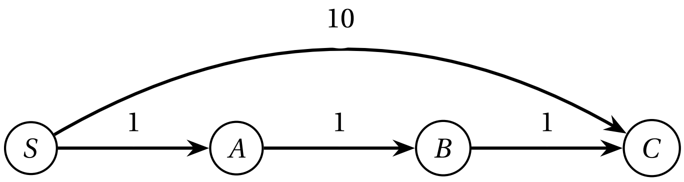
Algorithmik Blatt 5 Teil 2

Mtr.-Nr. 6329857

Universität Hamburg — 21. November 2019

Aufgabe 14

Teil 1



Wir hätten gerne alle kürzesten Pfade der Länge $k = 1$.

Reihenfolge der Relaxion: $(S, A), (A, B), (B, C), (S, C)$. Nach einer Iteration von Bellman-Ford ist von S nach C bereits der wirklich kürzeste Pfad Gefunden. Allerdings hat dieser Pfad 3 Kanten lang. Wir hatten uns aber nur für Pfade der Länge 1 interessiert. Was Bellman-Ford korrekt bestimmt hat, ist die Länge der k langen Prefixe der insgesamt kürzesten Pfade.

Teil 2

Das Problem ist, dass bei zu guter Wahl der Reihenfolge der Relaxion nach der n-ten Iteration bereits Pfade gefunden wurden, die länger sind als n. Wir wolle aber nach der n-ten Iteration nur Entfernungen bestimmt haben, die weniger als n Kanten entfernt liegen. Dann können wir sicher sein, dass die Knoten, deren Entfernung wir kennen, nur über Pfade der Länge n erreicht wurden — somit die bekannten Entfernungen sich auch nur auf diese Pfade beziehen.

In jeder Iteration darf die Kantenzahl der längsten bekannten Pfade nur um 1 erhöhnt werden. Ausgehende Kanten von Knoten, die selbst in der aktuellen Iteration erst erreicht wurden (Distanz wurde von ∞ auf eine Zahl gesetzt), müssen bis zum Ende der Iteration selbst von der Relaxion ausgeschlossen werden.

Aufgabe 15

```
1: procedure ClosestTreeLeafs(V, E, W)          // Knoten, Kanten, Gewichte
2:     (E', V') = Kruskal(E, V)          // O(|E| log(|V|))
3:     queue = initQueue()
4:     bestLeafs = 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:          bestLeafs[V[i]] = V[i]
9:          bestDistance[V[i]] = 0
10:         insertQueue(V[i], queue, bestDistance[V[i]])            // 𝒪(1), weil bestDistance[i] = 0
11:     else            // Inner Node
12:         bestLeafs[V[i]] = nil
13:         bestDistance[V[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|))
    return bestLeafs
```

Argumentation

Wir nehmen an, dass sich aus der Fernreisekarte in linearer Zeit ein Graph konstruieren lässt, in welchem jede Stadt und jeder Hafen einen von einem Knoten dargestellt wird und genau jede Kante einer Straße entspricht die die jeweiligen Städte mit einander bzw mit den erreichbaren Häfen verbindet.

Wir gehen davon aus, dass dieser Graph G zusammenhängend ist und 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 Häfen Blattknoten im Spannbaum.

Als nächtes durchlaufen wir den Baum Ebene für Ebene beginnend bei den Blättern (Häfen) und weisen dabei jedem Knoten den, nächsten Blattknoten, sprich den besten Vorgänger (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 Distanz 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 Distanz 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.