Kingdom

Marco Kabelitz

19. Dezember 2014

1 Problembeschreibung und Interpretation

Das Reich des Königs Kong besteht aus 2 Städten und $N < 150$ Dörfern, welche durch Straßen direkt miteinander verbunden sein können. Die Straßen sind in beiden Richtungen begehbar und überkreuzen sich nicht. Da sich der illegale Handel mit Ziegenkäse zwischen den zwei Städten wachsender Beliebtheit erfreut, möchte der König seine $G < 353535$ Soldaten so auf den Straßen seines Reiches platzieren, dass man auf jedem Weg von einer Stadt in die andere mindestens einen Soldaten passieren muss. Die Soldaten dürfen zwar weder direkt in Städten noch in Dörfern platziert werden, dafür aber in beliebiger Anzahl auf jedem Punkt einer Straße (und somit auch beliebig nah an einer Stadt oder einem Dorf).

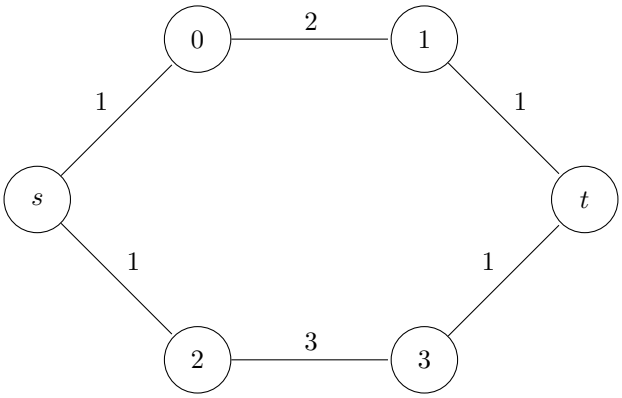
Das Problem an der ganzen Sache ist, dass jede der beiden Städte ständig Gefahr läuft, von fremden Armeen angegriffen zu werden. Um seine Soldaten möglichst schnell in der betroffenen Stadt sammeln zu können, muss der König sie zusätzlich so platzieren, dass die Zeit, bis alle Soldaten in der betroffenen Stadt eingetroffen sind, minimiert wird. Dabei assoziieren wir mit jeder Straße eine Zeit $\phi < 1000$, welche benötigt wird, um die gesamte Länge dieser Straße abzulaufen.

Die Frage ist nun: Wie lange brauchen die Soldaten mindestens, um sich in einer der Städte zu sammeln?

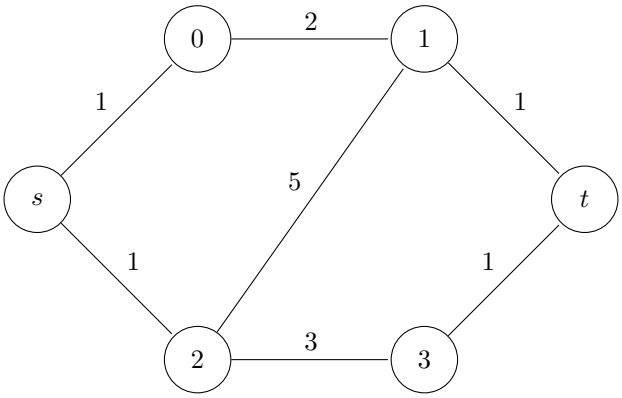
Wir formulieren diese Beschreibung als Graph-Problem: Gegeben sei ein ungerichteter Graph U mit mindestens 2 und höchstens 151 Knoten. Mit s und t bezeichnen wir die Knoten, welche die beiden Städte darstellen. Jede Kante in U besitzt ein ganzzahliges Gewicht $\phi < 1000$. Eine Soldat teilt eine Kante des Gewichts ϕ in zwei Gewichte $p \cdot \phi$ und $(1-p) \cdot \phi$ für ein reellwertiges $p \in [0, 1]$. Wir bezeichnen das Maximum aus der Länge des kürzesten Weges eines Soldaten zu s und eines kürzesten Weges zu t als *Antwortzeit* des Soldaten. Gesucht ist nun die minimale Zeit t_{opt}, für die eine Verteilung von $G < 353535$ Soldaten auf den Kanten von U existiert, sodass

1. kein Weg zwischen s und t existiert, der nicht mindestens eine mit einem Soldaten versehene Kante enthält und
2. das Maximum aller Antwortzeiten gleich t_{opt} ist.

Betrachten wir ein Beispiel. Es sei der folgende Graph U gegeben:



Wenn G kleiner als 2 sein sollte, ist das Problem nicht lösbar, da wir mindestens 2 Soldaten benötigen, um die 2 möglichen Wege von s nach t abzudecken. Gilt hingegen $G \geq 2$, so können wir mindestens einen Soldaten auf der Mitte der Kante $(0, 1)$ platzieren und mindestens einen Soldaten auf der Mitte der Kante $(2, 3)$. Die Antwortzeit des Soldaten auf $(0, 1)$ zu s und zu t beträgt 2.0, während die Antwortzeit des Soldaten auf $(2, 3)$ bei 2.5 liegt. 2.5 ist gleichzeitig das optimale Ergebnis auf diesem Graphen. Wir modifizieren den Graphen für ein zweites Beispiel wie folgt:



Wieder ist das Problem für $G < 2$ nicht lösbar. Für $G \geq 2$ ist die optimale Lösung 3.0, z.B. bei der Platzierung mindestens eines Soldaten beliebig nah an 1 auf der Kante $(1, t)$ und mindestens eines Soldaten auf der Mitte der Kante $(2, 3)$.

2 Lösungsalgorithmus

Unser Algorithmus besteht aus mehreren Stufen. Zuerst überprüfen wir, ob wir über eine ausreichende Anzahl an Soldaten verfügen, um alle Wege von s nach t in U abzudecken. Hierfür erstellen wir einen zu U äquivalenten Graphen A und setzen alle Kantengewichte in A auf 1. Nun suchen wir mit Hilfe des Ford-Fulkerson-Algorithmus einen maximalen Fluss in A, wobei s als Quelle und t als Senke dient.

Da der maximale Fluss dem minimalen Schnitt des Graphen entspricht und alle Kantengewichte 1 sind, können wir das Ergebnis als Anzahl der Kanten interpretieren, welche wir mindestens abdecken müssen. Sollte der maximale Fluss höher sein als G, so ist das Problem nicht lösbar und der Algorithmus terminiert.

Nachdem die Existenz einer Lösung festgestellt wurde, bestimmen wir als nächstes die Länge des kürzesten Weges von jedem Knoten in U zu jedem anderen Knoten. Dies kann entweder durch einen Aufruf des Dijkstra-Algorithmus (SSSP, *single-source shortest path*) auf jedem Knoten des Graphen oder mit Hilfe des Floyd-Warshall-Algorithmus (APSP, *all-pairs shortest path*) durchgeführt werden. Die Länge des kürzesten Weges zwischen zwei Knoten i und j bezeichnen wir im Folgenden als d_{ij}.

Nun errechnen wir für jede Kante (i, j) die optimale Antwortzeit eines Soldaten auf dieser Kante. Wir bezeichnen dieses Optimum als r_{ij}. Zu dessen Bestimmung müssen folgende Fälle betrachtet werden:

- $d_{si} \leq d_{sj}$ und $d_{it} \leq d_{jt} \Rightarrow r_{ij} := \max\{d_{si}, d_{it}\}$
- $d_{si} > d_{sj}$ und $d_{it} > d_{jt} \Rightarrow r_{ij} := \max\{d_{sj}, d_{jt}\}$
- $d_{si} \leq d_{sj}$ und $d_{it} > d_{jt} \Rightarrow r_{ij} := \max\left\{\max\left\{\frac{d_{si} + d_{ij} + d_{jt}}{2}, d_{si}\right\}, d_{jt}\right\}$
- $d_{si} > d_{sj}$ und $d_{it} \leq d_{jt} \Rightarrow r_{ij} := \max\left\{\max\left\{\frac{d_{it} + d_{ij} + d_{sj}}{2}, d_{it}\right\}, d_{sj}\right\}$

Wir schreiben alle r_{ij} in eine Liste L und sortieren diese in aufsteigender Reihenfolge. Da ein optimales Ergebnis gleich der optimalen Antwortzeit einer Kante des Graphen sein muss, muss L das gesuchte Ergebnis enthalten. Daher führen wir nun eine binäre Suche auf L durch, wobei wir für jede gewählte Antwortzeit $L(x)$ überprüfen, ob der Graph sich für diese Antwortzeit mit G Soldaten abdecken lässt.

Diese Überprüfung findet folgendermaßen statt: Wir erstellen einen zu U äquivalenten Graphen F und setzen die Gewichte aller Kanten (i, j), für die r_{ij} echt größer ist als die aktuell gewählte Antwortzeit $L(x)$, auf ∞. Das Gewicht aller anderen Kanten wird auf 1 gesetzt. Nun führen wir den Ford-Fulkerson-Algorithmus auf F aus, erneut mit s als Quelle und t als Senke. Ist der maximale Fluss in F nicht größer als G, so ist $L(x)$ eine gültige Lösung und die binäre Suche wird entweder terminieren oder ein kleineres Element aus L wählen und die nächste Iteration starten. Ist der Fluss in F hingegen größer als F, so ist L(x) keine gültige Lösung und eine größere Antwortzeit aus L muss gewählt werden.

Wieso ist die Bestimmung des maximalen Flusses in F hilfreich? F stellt für jede Kante dar, ob für eine Antwortzeit $L(x)$ ein Soldat auf einer Kante stehen darf (induziert durch Gewicht 1) oder ob dies nicht möglich ist (induziert durch Gewicht ∞). Ist $L(x)$ größer als eine optimale Lösung, so wird der maximale Fluss in F immer kleiner oder gleich G sein. Ist $L(x)$ hingegen kleiner als eine optimale Lösung, so enthält F mehr Wege von s nach t als wir Soldaten zur Verfügung haben, was bedeutet, dass der minimale Schnitt (und somit der maximale Fluss) größer als G sein muss.

Sobald die binäre Suche für ein $L(x)$ terminiert, ist dieses $L(x)$ eine optimale Lösung des Problems.

3 Laufzeitabschätzung

Sei n die Zahl der Knoten, m die Zahl der Kanten und f der maximal Fluss eines Graphen.

Die Überprüfung der Existenz einer validen Lösung beinhaltet einen Aufruf des Ford-Fulkerson-Algorithmus, dessen Komplexität bei $\mathcal{O}(f \cdot m)$ liegt.

Die Bestimmung der kürzesten Distanzen zwischen alle Knotenpaaren erfordert sowohl mit Hilfe des Floyd-Warshall- als auch mit Hilfe des Dijkstra-Algortihmus einen Aufwand von $\mathcal{O}(n^3)$.

Der Zeitaufwand für die Bestimmung der optimalen Antwortzeiten ist linear in der Zahl der Kanten, also $\mathcal{O}(m)$.

Die binäre Suche auf L hat $\mathcal{O}(\log m)$ Iterationen, wobei in jeder Iteration der Ford-Fulkerson-Algortihmus Anwendung findet. Der Aufwand beträgt also $\mathcal{O}(f \cdot m \cdot \log m)$ und gibt zugleich die Laufzeitabschätzung für den gesamten Algorithmus an.

4 Anmerkungen

- Problem G des NCPC 2005 - Im Wettbewerb ungelöst, 0 Submissions
- Java-Implementierung: 1.7 Sekunden (Time Limit liegt bei 2.0 Sekunden)
- Äquivalente C++-Implementierung benötigt 0.9 Sekunden