Aufgabe 2: VS (WiSe 13/14)

Implementieren eines verteilten Algorithmus zur Bestimmung des minimalen Spannbaums

Definitionen

Gewicht eines Graphen

Ein gewichteter ungerichteter Graph (G,w) ist ein ungerichteter Graph G=(V,E) zusammen mit einer Gewichtsfunktion $w: E \rightarrow R$

Ist H=(U,F) ein Teilgraph von G mit $U \subseteq E \; und \; F \subseteq V$, dann ist das Gewicht von H definiert als $w(H) = \sum_{e \in F} w(e)$.

Minimaler Spannbaum

Ein Teilgraph H eines ungerichteten Graphen G heißt Spannbaum von G, wenn H ein Baum auf den Knoten von G ist.

Ein Spannbaum S eines gewichteten ungerichteten Graphen G heißt minimaler Spannbaum von G, wenn S minimales Gewicht unter allen Spannbäumen von G besitzt.

Eindeutigkeit

Wenn die Gewichte der Kanten eindeutig sind, dann ist auch der minimale Spannbaum eindeutig. (Bei eindeutigen Kantengewichten gibt es genau eine Lösung für den minimalen Spannbaum.)

Anwendung in verteilten Systemen

Berechnung eines Overlay-Trees für effizientes anwendungsbasiertes Multicast in verteilten Systemen. Inhalt der Vorlesung über Kommunikation.

Der verteilte Algorithmus zur Berechnung eines minimalen Spannbaums nach Gallager, Humblet, Spira

Der Algorithmus ist nachrichtenbasierter Ansatz mit einer bottom up Konstruktionsvorschrift. N Knoten eines Graphen berechnen den minimalen Spannbaum durch Verschicken von Nachrichten nach einem wohldefinierten Protokoll.

Die Kernidee des Algorithmus lautet:

Gegeben G=(V,E) ein ungerichteter Graph und T1, T2 minimale Spannbäume mit disjunkten Teilmengen von Knoten V1 und V2, e eine Kante mit minimalem Gewicht, die die Bäume T1 und ***T2*** verbindet. Dann ist der Subgraph (***T1***,***T2***,***e***) ein minimaler Spannbaum auf den Knoten $V1 \cup V2$. Teilgraphen werden auch Fragments genannt.

Zu Beginn ist jeder Knoten ein Fragment der Ebene L=0. Wiederholte Anwendung des Verbindens zweier Teilbäume ***T1*** und ***T2*** erzeugt den minimalen Spannbaum auf G.

Problem:

Wie bestimmen die Knoten eines Fragments die Kante mit dem geringsten Gewicht, über die die Verbindung zu einem anderen Fragment hergestellt wird?

Antwort: Jedes Fragment hat einen Koordinator, der die Suche initiiert und die ausgehende Kante selektiert.

Problem:

Wie stellt ein Knoten in ***T1*** fest, ob die Kante ***e***, über die die Verbindung hergestellt wird, zu einem Knoten in einem anderen Baum ***T2*** oder in den eigenen Baum ***T1*** führt?

Lösung: Knoten desselben Fragments müssen den gleichen Namen annehmen, bevor verschiedene Fragmente zusammengeführt werden können. Fragmentnamen leiten sich aus dem Gewicht der Kante ab, über die Fragments zusammengeführt werden. Die Kante muss zu einem Fragment mit anderem Namen führen.

Konstruktionsvorschrift

Jedes Fragment sucht asynchron nach der ausgehenden Kante mit minimalem Gewicht. Eine ausgehende Kante ist eine Kante, die nicht zum Fragment selbst gehört. Über diese Kante wird die Verbindung zu einem Fragment am „anderen“ Ende der Kante hergestellt. Wie und wann diese Verbindung hergestellt wird, hängt von den Ebenen (levels) der beiden Fragments ab. Wenn Fragment F mit Ebene L und Fragment F‘ mit Ebene L‘ kombiniert werden, dann gibt es zwei Fälle zu unterscheiden:

1. L < L‘: Fragment F wird sofort von Fragment F‘ absorbiert. Das erweiterte Fragment hat den Level L‘.
2. L=L‘: Fragment F und F‘ werden zu einem neuen Fragment F‘‘ kombiniert. Der Level von F‘‘ ist L+1 und die Name von F‘‘ ergibt sich aus dem Gewicht der verbindenden Kante („core“) genannt

Wenn keine Kombination möglich ist, oder L > L‘, dann wartet Fragment F, bis F‘ einen Level erreicht, der die Kombination nach den Regeln erlaubt.

Zum Verständnis der nachfolgenden Ausführung:

Jeder Knoten hat 3 Zustände:

1. *sleeping*: Zu Beginn ist jeder Knoten in diesem Zustand.
2. *find*: der Zustand während der Suche nach einer ausgehenden Kante mit minimalem Gewicht
3. *found*: weder *sleeping* noch *find*. Insbesondere werden im Zustand *found* Zustand keine *test* Nachrichten über ausgehende Kanten verschickt.

Suchen nach der ausgehenden Kante mit minimalem Gewicht (akmg)

Fall 1: Fragments mit Level L=0 (Fragments mit nur einem Knoten)

Zu Beginn ist jeder Knoten im Zustand s*leeping.* Wird ein Knoten „aufgeweckt“ oder erhält er in diesem Zustand eine Nachricht eines anderen Knoten, dann

1. wählt der Knoten die *akmg*
2. markiert *akmg* als *branch*
3. schickt eine *connect* Nachricht über die Kante
4. geht in den Zustand *found*, um wartet auf eine Antwort des Fragments auf der anderen Seite.

Fall 2: Knoten in Fragments mit Level >0

Ausgangspunkt: Ein Fragment F mit Level L wurde gerade durch Kombination von Fragments der Ebene L-1 mit derselben *akmg* geformt. *akmg* wird zum *core* von F. Das Gewicht von *akmg* ist der Name von F.

Die beiden Knoten an den Enden der Kante starten einen neuen Suchzyklus, indem sie eine *initiate* Nachricht an alle Knoten ihres Fragments (der Ebene L-1) senden. Die *initiate* Nachricht hat 3 Parameter: Level von F, Name von F und die Zustandsinformation *find*.

Wenn Knoten die *initiate* Nachricht erhalten, starten diese die Suche nach einer *akmg*. Das Problem ist, wie Knoten eine ausgehende Kante erkennen können. Dazu werden die Kanten klassifiziert in:

1. *branch*: Kanten, über die Fragmente verbunden werden / wurden.
2. *rejected*: Kanten in einem Fragment, die keine branches sind, aber als verbindende Kanten in einem Fragment identifiziert wurden.
3. *basic*: weder *branch* noch *rejected*. Zu Beginn sind alle Kanten *basic*

Um die *akmg* zu bestimmen, wählt ein Knoten die *basic* Kante mit minimalem Gewicht aus und schickt eine *test(L,N)* Nachricht über die Kante.

Wenn ein Knoten in Fragment F‘ mit Level = L‘ und Name = N‘ eine *test* Nachricht erhält, reagiert er abhängig von den Parameterwerten:

1. N‘=N: der Knoten befindet sich im selben Fragment weist den Test zurück
    a. markiert die Kante als *rejected*
    b. schickt eine *reject* Nachricht über die Kante. **Ausnahme**: Der Knoten schickt und empfängt eine test Nachricht mit gleichem Fragmentnamen über die gleiche Kante, dann wird keine Nachricht zurückgeschickt.
2. N‘!=N und L‘>= L: der empfangene Knoten befindet sich in einem anderen Fragment, sein Fragment Level ist größer oder gleich dem Level des Sendenden, dann sendet der empfangende Knoten die Nachricht *accept* über die Kante.
3. N‘ != N und L‘ < L: der empfangende Knoten verzögert die Antwort, bis sein Fragment Level die Bedingungen unter 1.) oder 2.) erfüllt. Grund: der Empfangene Knoten könnte schon Bestandteil eines Fragments einer höheren Ebene sein, da sein Fragment gerade dabei ist eine Kombination mit einem anderen einzugehen.

Bestimmen der gemeinsamen akmg eines Fragments

Alle Knoten eines Fragments müssen nun die *akmg* des Fragments bestimmen. Dazu schicken alle Blatt-Knoten eine *report(W)* Nachricht über den innenliegenden *branch* Kanten. *W* ist das Gewicht der *akmg* eines einzelnen Knoten. Die innenliegenden Knoten warten bis sie Antwort auf ihre Testnachrichten erhalten haben und von den Vorgängern die *akmg* über *report* mitgeteilt bekommen. Sie berechnen und merken sich die lokal beste Kante und reichen das minimale Gewicht über die innenliegenden *branch* Kanten weiter. Knoten, die eine *report* Nachricht schicken, gehen in den Zustand *found* über. Irgendwann tauschen auch die beiden Knoten rechts und links von *core report* Nachrichten aus, so dass die Seite, in der die *akmg* liegt bestimmt werden kann. Wenn die Knoten rechts und links von *core report* Nachrichten ausgetauscht haben, dann erlaubt die Information über die lokal besten Kanten, die Pfade zu der neuen *akmg* zu bestimmen. Dann wird die Nachricht *changeroot* über diese Pfade geroutet.

Erreicht die Nachricht *changeroot* die *akmg*, dann sind alle innenliegenden Knoten auf den neuen *core* ausgerichtet. Jetzt wird eine Nachricht *connect(L)* über diese Kante geschickt, dabei ist L das Level des neuen Fragments.

Verbinden zweier Fragments mit connect(L)

Fall 1: Zwei Fragments mit gleichem Level

Jedes Fragment sendet eine *connect(L)* Nachricht über die *akmg.* Damit wird die Kante zum *core* des neuen Fragments der Ebene L+1 und wird als *branch* markiert. Mit dem *connect* wird ein erneuter Zyklus über die *initiate(L+1,core,find)* Nachrichten eingeleitet.

Fall 2: Zwei Fragments mit unterschiedlichem Level

Eine *connect(L)* Nachricht eines Knotens in einem Fragment mit Namen F erreicht einen Knoten n‘ des Fragments F‘ des Level L‘ mit L < L‘. Dann schickt n‘ eine initiate(L‘,F‘) an den Knoten n. Wenn n‘ noch keine *report* Nachricht verschickt hat, dann tritt F dem Fragment F‘ bei und beteiligt sich an der Suche für das *akmg* des Fragments F‘. Wenn n‘ bereits eine *report* Nachricht verschickt hat, dann ist das *akmg* von F‘ kleiner als das von F und F muss sich an der Suche nach dem *akmg* nicht beteiligen. Die beiden Fälle werden durch den Zustandsparameter in der *initiate* Nachricht gesteuert. Knoten in F übernehmen den Zustand der *initiate* Nachricht. Testnachrichten werden nur im Zustand *find* versendet.

Nachrichten

{initiate,Level,FragName,NodeState,Edge}

{test,Level,FragName,Edge}

{accept,Edge}

{reject,Edge}

{report,Weight,Edge}

{changeroot,Edge}

{connect,Level,Edge}

Dabei ist

NodeState : sleeping | find | found

Level: Ebene des Fragments eine ganze Zahl

FragName: Name des Fragments, das sich aus dem Gewicht des Cores ergibt

Edge = {Weight, Nodex, Nodey} eine Kante zwischen den Knoten Nodex und Nodey mit Gewicht Weight. Nodex und Nodey sind logische Namen von Knoten, die systemweit registriert sein müssen.

Implementierung

1. Implementieren Sie den Algorithmus in Erlang. Folgende Einschränkungen sind zu beachten:
    a. Jeder Knoten kennt nur seine lokale Umgebung, d.h. nur die von ihm ausgehenden Kanten. Diese werden in einer Konfigurationdatei *node.cfg* mit einer Zeile proausgehender Kante wie folgt definiert.

        weight,nodename

        weight eine Zahl, das Gewicht der Kante

        nodename der globale Name eines Knotens, mit dem dieser Knoten über die Kante verbunden ist.

    b. Alle Knoten registrieren sich global unter ihrem Namen

2. Protokollieren Sie jeden Nachrichtenaufruf und die Teilergebnisse, d.h. die minimalen Spannbäume aller Ebenen auf der Konsole.

Quellen und Hilfen

Foliensatz mit Grafiken zur Veranschaulichung des Algorithmus

A *Distributed Algorithm* for Minimum-Weight *Spanning Trees*

Originalveröffentlichung zum Algorithmus: Studium diese Originals wird explizit empfohlen, sehr hilfreich für die Lösung.

http://dl.acm.org/citation.cfm?id=357200

Beispielkonfiguration