README.md

SpanningTreeSimulation

Aufgabe des Netztechnik Labors in 2021/2022. Ziel ist einen Spanning Tree Algorithmus auf Layer 2 zu implementieren.

Aufbau der Simulation

Für jede komponente des Netzwerks wurde ein Objekttyp erstellt. Für jeden Knoten des Netzwerks ist ein Thread vorgesehen. Diese kommunizieren über definierte Links. Dabei findet die Kommunikation über Listen statt. Wo die beiden beteiligten Threads ihre "gesendeten" Messages speichern und die "empfangenen" Nachrichten abrufen. Der Link stellt dabei ein shared media (zwischen Threads) dar und wird über ein Semaphor abgesichert.

Eingabe- und Ausgabeformat

Eingabeformat

Die Eingabe findet über folgendes definiertes Format in form einer Textdatei statt. Die Datei hat folgenden Aufbau. Gestartet wird mit folgendem:

```
Graph mygraph {
```

Anschließend werden die Nodes definiert: Bezeichnung = ID;. Hier ein paar Beispiele:

```
// Node-IDs
 A = 5;
 B = 1;
 C = 3;
 D = 7;
 E = 6;
 F = 4;
```

Folgend werden die Links folgendermaßen definiert: KnotenA - KnotenB : Kosten;. Beispiele:

```
// Links und zugeh. Kosten
 A - B : 10;
 A - C : 10;
```

```
      B - D : 15;
      B - E : 10;
      C - D : 3;
      C - E : 10;
      D - E : 2;
      D - F : 10;
      E - F : 2;
}
```

Abgeschlossen mit '}'.

Ausgabeformat

Der Spanning Tree wird folgendermaßen ausgegeben:

```
Spanning-Tree of mygraph {
    Root: B;
     A - B;
     C - D;
     D - E;
     E - B;
     F - E;
}
```

Der Root wird mit Root gekennzeichnet, die nächsten Hops von den Knoten immer so: Knoten - Hop;.

Benutzung
Benötigt:

- Python
- pip

1. In Verzeichnis navigieren und requirements (tabulate zur darstellung des spanning trees) installieren.

```
pip install -r requirements.txt
```

2. Aufrufen des Skripts (-help für Argumente). Argumente:

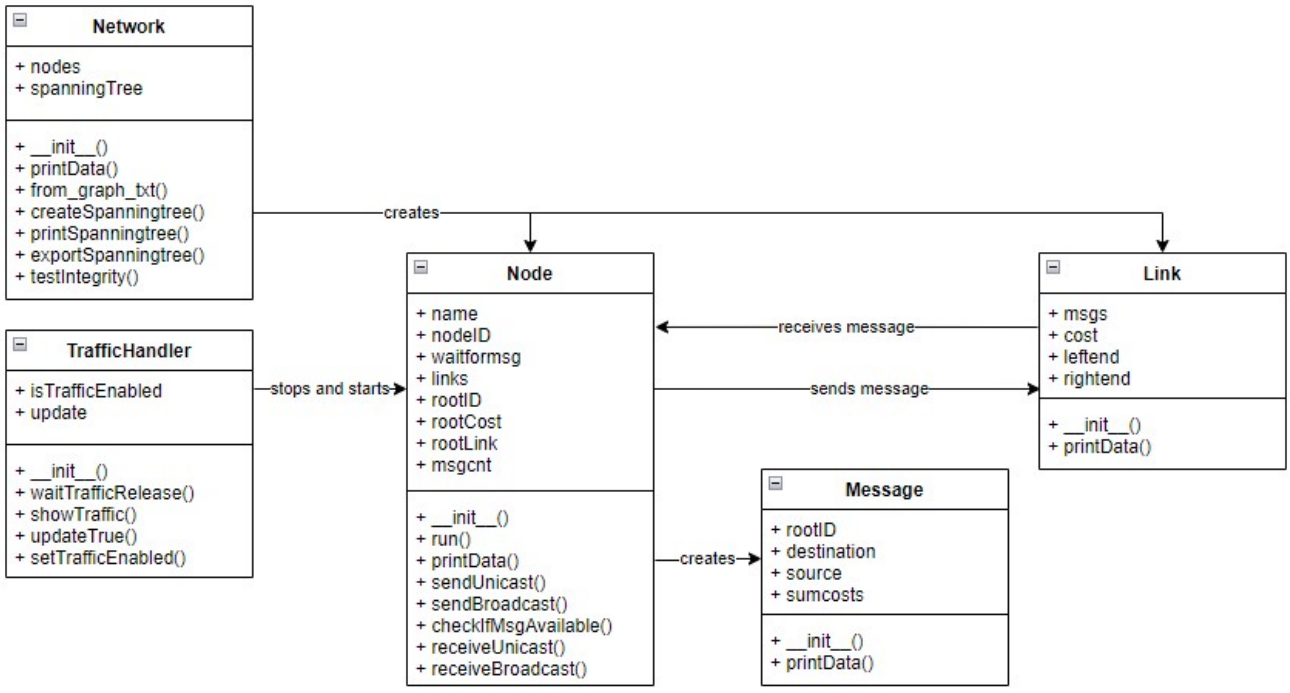
- "-f", "--filepath": Path to the .txt file that contains the Network specifications (default=./Inputdateien/graph.txt).
- "-amsg", "--waitformsg": Amount of empty messages before stopping (default=5).
- "-t", "--showtraffic": Shows every interaction. Slows down routing process (default=False).
- "-mit", "--maxitems": Max amount of items (default=50).

- "-mco", "--maxcost": Max cost (default=50).
- "-mid", "--maxid": Max ID (default=50).

Funktion

Simulation

Um die Netzwerkkomponenten nachzubilden wurde für jede Komponente eine Klasse erstellt. Die Netzwerk Klasse enthält dabei alle Netzwerk Knoten (Nodes) und Kanten (Links). Es dient dazu aus den verschiedenen unabhängigen Nodes den Spanning Tree zu erzeugen. Außerdem dient die Klasse dazu die Netzwerkkonfiguration aus der graph.txt datei einzulesen und die Nodes und Links zu erstellen. Die Node Klasse stellt dabei die Kanten des Netzwerks dar. Das besondere an der Klasse ist, dass diese von der Thread Klasse erbt. Somit laufen alle Nodes als selbstständige Threads parallel. Die Knoten bekommen bei ihrer Erstellung ein oder mehrere Objekte der Klasse Link übergeben. Über diese kommuniziert der Knoten mit seinen benachbarten. Der Link ist dabei als shared media zu betrachten. Der Zugriff wird über ein Semaphor geschützt. Die eben erwähnte Link Klasse dient zur Kommunikation zwischen zwei Knoten. Jeder Link hat Kosten sowie die IDs der zwei verbundenen Knoten. Außerdem existiert eine Liste in der die Knoten ihre Nachrichten abspeichern ("senden"). Die Form der Nachrichten ist durch die Message Klasse definiert und besteht aus der neuen root ID, den Kosten zu dieser und den IDs des Senders sowie des Empfängers. Die Traffic Handler Klasse dient zur visualisierung des Datenverkehrs. Sobald ein Knoten Nachrichten gesendet hat wird ein Flag gesetzt, dass den Traffic Handler anweist das Bild der Konsole zu aktualisieren. Somit unterbricht der Handler die Knoten (diese warten solange), aktualisiert das Bild und gibt die Knoten anschließend wieder frei.



Algorithmus

Der Algorithmus wird solange ausgeführt wie das exit-Flag gesetzt wird. Der Algorithmus überprüft als erstes ob eine Nachricht vorhanden ist. Wenn dieses Kriterium erfüllt ist und zusätzlich das Start-Flag gesetzt ist, dann wird die Nachricht abgerufen. Anschließend wird überprüft ob die root id kleiner oder gleich der momentan abgespeicherten ist und ob die Kosten zu dieser ID geringer sind. Falls das der Fall ist, dann wird die abgespeicherte Root ID sowie die Kosten aktualisiert. Die neue Root ID mit den Kosten wird an alle Links gesendet. Ansonsten wird in die Hauptschleife zurückgekehrt. Falls keine Nachricht vorhanden ist wird ein Counter hochgezählt welcher angibt wie viele Iterationen ohne Nachricht vergangen sind. Wenn dieser Counter einen bestimmten Wert erreicht (in den Input Argumenten festgelegt), verfällt der Knoten in einen Schlaf Modus. In diesem wird dem mit der inkrementierung einer Variable dem main-Thread signalisiert, dass der Knoten sich im Schlaf Zustand befindet. Falls alle Knoten diese Variable inkrementieren wird der Algorithmus beendet. Knoten, die sich im Schlafzustand befinden überprüfen in regelmäßigen Abständen ob entweder eine neue Nachricht anliegt oder das exit Flag gesetzt wurde. Falls eine neue Nachricht anliegt wird die end Variable dekrementiert und der message-Counter auf null zurückgesetzt. Der Knoten kehrt in die Hauptschleife zurück. Wenn das exit Flag gesetzt ist, wird der Algorithmus beendet.