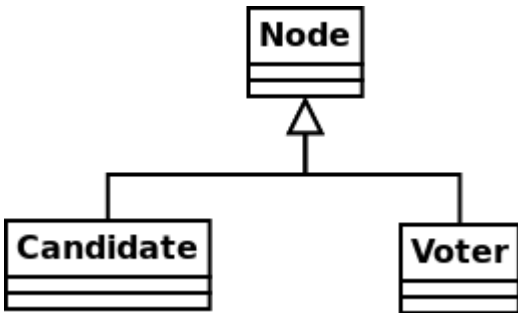
README – ÜBUNG 2 Viktor Werle 3603083

Systemvoraussetzungen:

- mindestens Python 3.5
- ZEROMQ Library

Umsetzung

Die Umsetzung der Übung 2 baut auf der Übung 1 auf. Der Kandidat-Knoten und der Wähler-Knoten erben von dem normalen Knoten und erweitern diesen um die jeweils spezifische Funktionen.



Des Weiteren wurde aus dem Python-Skript node_message.py eine Klasse Message gemacht. Der Protokoll wurde so verändern, dass man diesen nun um beliebig viele Inhalte erweitern kann. Die Reaktion auf die Nachrichten erfolgt nun anhand der Nachrichtentypen, welche in der Klasse MsgType zusammengefasst wurden.

Aufgabe 1

Die Vektorzeit stellt eine Liste der Länder n (n = Anzahl der Knoten im Netzwerk) dar. Beim Start wird die Liste mit 0 – Werten initialisiert. Die Anpassung der Vektorzeit erfolgt jeweils beim Senden oder Empfangen einer Nachricht und wird jeweils in der Klasse Submitter und Receiver gesetzt (sonst nirgends…). Somit wir gewährleistet dass bei jeder „Aktion“ eine korrekte Vektorzeit übermittelt bzw. empfangen wird.

Aufgabe 2

Auf **Wähle** mich gehen ich hier nicht näher ein, da trivial.

Echo – Algorithmus
Der Echo-Algorithmus nutzt zum Übertragen die Nachrichten-Typen:

- ECHO_EXPLORER
- ECHO_ECHO

Anhand des Nachrichtentypes wissen die Knoten, wie auf diese Nachricht zu reagiert ist.
Sobald der Knoten zum ersten mal einen Explorer bekommt, speichert er die ID vom Sender als FirstLink und schickt an alle seine Nachbarn (außer FirstLink) einen Explorer. Da wir mehrere Kandidaten haben, wird FirstLink anhand der Kandidaten ID unterschieden (so können bei mehreren Kandidaten gleichzeitig mehrere Campaign ablaufen.).

Danach wartet der Knoten auf ECHOs von den Nachbarn. Sobald er alle hat, schickt er an den FirstLink selbst ein ECHO, löscht diesen und setzt den ECHO_COUNTER zurück.
Hat der Kandidat alle ECHOs von seinen Nachbarn empfangen, dann weißt er, dass die Campaign zu ende ist.

Aufgabe 3

Der Observer wurde als Klasse Observer realisiert und wird in einem gesonderten Prozess gestartet. Jeder Knoten kennt den Observer und kann mit Ihm über vordefinierte Nachrichtentypen kommunizieren.

Abweichungen von der Aufgabenstellung
Ich habe folgende Abweichungen von der Aufgabenstellung vorgenommen:

- Der Vektorzeitstempel wird nicht durch den Observer an die Knoten übermittelt, sondern in einer Konfigurationsdatei festgelegt. Grund – passt besser in meinen Konzept.
- Beim Erreichen des Vektorzeitstempels schicken die Knoten das Endergebnis nicht an den Observer! Grund - es ist nicht immer gegeben, dass alle Knoten den Vektorzeitstempel erreichen (auf Grund der Confidence-Levels). Und ich fand es nicht sauber genug einfach die Dummy Nachrichten zu schicken, damit wirklich ALLE Knoten die Vektorzeit erreichen.
  Die Endergebnisse können bei mir mittels Snapshots am Ende eingesammelt werden und sind somit immer konsistent.

Umsetzung Snapshot

Der Manager kann den Snapshot initiieren. Sobald ein Knoten eine Init-Nachricht oder eine markierte Nachricht zum ersten mal bekommen hat, setzt er seinen Status auf markiert und schickt seinen Zustand (Confidence Level) an den Observer (beginnt mit dem Record). Ab diesem Zeitpunk verschickt der Knoten nur noch markiert Nachrichten. Danach macht er einen Broadkast an alle seine Nachbarn und zählt anhand empfangenen Nachrichten wie viele Nachbarn markiert sind. Sobald alle Nachbarn markiert sind (von allen wurde eine markierte Nachricht empfangen) schickt er an den Observer den Status READY.
Sollte der Knoten zwischen <der Init-Nachricht oder ersten markierten Nachricht> und dem <READY> unmarkierte Nachrichten empfangen, so schickt er diese Nachricht an den Observer. Der Observer kann dann die Confidence – Levels entsprechend verrechnen.

Sobald der Observer von allen Knoten den READY – Status hat, gibt er das Ergebnis des Snapshots aus.

Konfiguration und Programmablauf bzw. Startart

Die Konfiguration erfolgt analog der Übung 1 über die Settings-Datei, welche im Verzeichnis **conf/** zu finden ist.
Hier können

- die Kandidaten festgelegt werden
- die Vektorzeit für die Terminierung eingestellt werden
- die Anzahl der Knoten und Kanten festgelegt werden
- die Obergrenze für den Response
- … etc. …

Programmstart
Zum Starten werden folgende Skripte ausgeführt:

- manager.py (zum Starten vom Manager)
- node_starter.py (zum Starten aller Knoten)
- observer.py (zum Starten des Observers)

Wichtig: die Skripte MÜSSEN aus dem Verzeichnis src/ ausgeführt werden, sonst stimmen die Pfade nicht!

Momentane Konfiguration

- 100 Knoten
- 180 Kanten
- Terminierungszeit 800
- Kandidaten 7 und 12
- Response-Wert 40 (ab wann neue Camapign oder Wähle mich gestartet wird)

Manager
Der Manager steuert die Knoten und den Observer.

```
******************************
* 1 : Print all avalable Nodes *
* 2 : Shutdown node           *
* 3 : Shutdown all nodes      *
* 4 : Send message            *
* 9 : Start vote for me       *
* 10: Start campaign          *
* 11: Snapshot                *
* 12: Reset observer          *
* 13: Unmark nodes            *
* 0 : Exit manager            *
******************************
```

Aktuell ist alles auf Aufgabe 3 eingestellt. Sobald der Observer und die Knoten laufen, kann die Verbreitung mit 9. gestartet werden. Ab diesem Zeitpunk schicken die Kandidaten random WÄHLE_MICH oder KAMPAGNE Nachrichten.
Mit 11. kann ein Snapshot gemacht werden (dies sollte relativ früh passieren, sonst sind die Confidence Levels ziemlich gleich mit dem Endergebnis). Berechnung kann einige Zeit in Anspruch nehmen!
Sobald das Ergebnis feststeht, warten Sie bis alle Nachrichten terminieren (es werden keine Nachrichten mehr verschickt).
Jetzt kann mit 12. und 13. der Zustand vom Observer resetet werden und die Knotenmarkierung gelöscht werden. Danach können erneut mit 11. die Endergebnisse gesammelt werden und vom Observer ausgegeben werden.

Um den Ablauf zu wiederholen, reicht es den Observer zu Reseten und alle Knoten neu zu starten!