Zadanie 1: rozproszona wiedza
SKJ (2016)

Wstęp

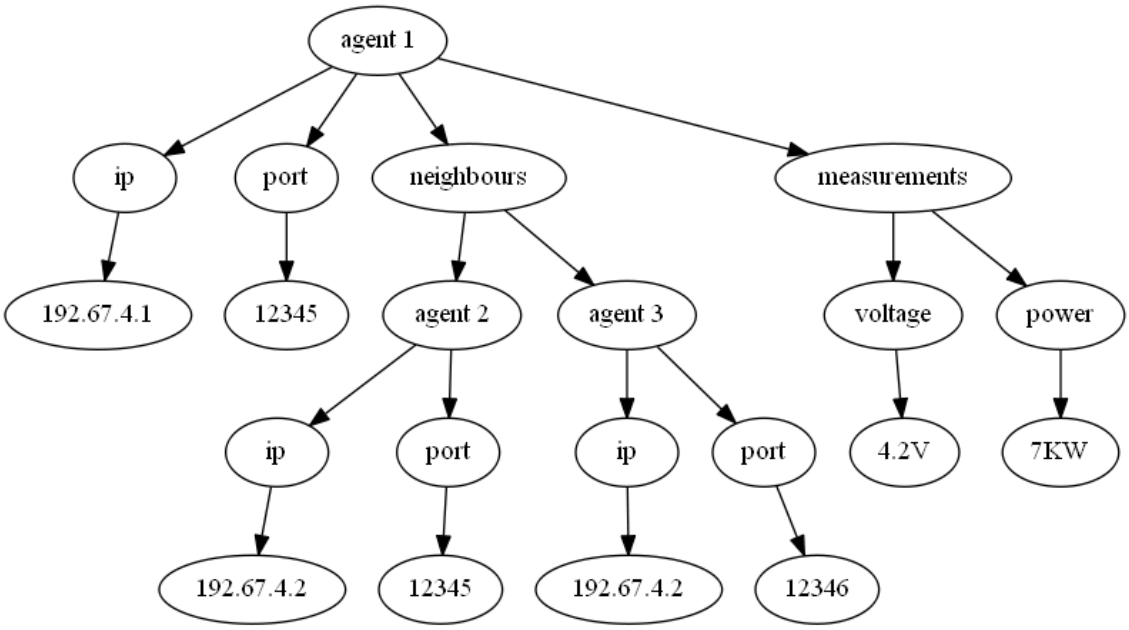
Zadanie polega na zaimplementowaniu systemu do utrzymywania rozproszonej wiedzy pomiędzy agentami.

Agent to program uruchomiony na hoście posiadający pewną wiedzę lokalną i mogący komunikować się z innymi agentami, z którymi jest bezpośrednio połączony. Wiedza lokalna reprezentowana jest przez drzewo, którego węzły etykietowane są napisami. Etykieta w korzeniu takiego drzewa jest nazwą agenta. Zakładamy, że w systemie nazwy agentów są unikalne. Ponadto, korzeń musi posiadać następujące dzieci:

- węzeł z etykietą "ip" — opisuje adres IP, na którym działa agent; jedynym dzieckiem tego węzła jest węzeł zawierający adres ip w formacie "XXX.XXX.XXX.XXX",
- węzeł z etykietą "port" — opisuje port, na którym działa agent; jedynym jego dzieckiem jest węzeł zawierający numer portu,
- węzeł z etykietą "neighbours" — opisuje lokalną wiedzę o bezpośrednich sąsiadach agenta; etykietą każdego dziecka tego węzła jest nazwa sąsiada i węzeł ten posiada woje dzieci, węzły "ip" i "port" z potomkami w formacie jak wyżej.

Na Rysunku 1 przedstawiono przykładową wiedzę lokalną agenta "agent 1". Oprócz wymaganych dzieci "ip", "port", "neighbours", korzeń dodatkowo posiada poddrzewo "measurements" z dziećmi "voltage" i "power". Zakładamy, że żaden z węzłów nie może mieć więcej niż jednego syna z daną etykietą.

Wiedza globalna to las, którego drzewa stanowią wiedzę lokalną agentów działających w systemie. Sąsiedztwo agentów wyznaczane jest przez wiedzę *globalną* — jeżeli agent A ma wpisanego lokalnie B jako sąsiada, lub wie (dowiedział się na skutek wymiany komunikatów), że B ma wpisanego lokalnie A jako sąsiada, to B jest sąsiadem A. Czyli, po rozpropagowaniu wiedzy, relacja sąsiedztwa jest *symetryczna* (jest symetrycznym domknięciem relacji wyznaczanej przez wiedzę lokalną ze wszystkich agentów), ale nie jest *przechodnia* — z tego, że A jest sąsiadem B, a B jest sąsiadem C, nie wynika, że A jest sąsiadem C (sąsiad mojego sąsiada, nie musi być moim sąsiadem). Warto też zwrócić uwagę, że relacja sąsiedztwa staje się symetryczna dopiero po rozpropagowaniu wiedzy — jeżeli A



Rysunek 1: Przykładowa wiedza lokalna agenta.

ma wpisanego B jako sąsiada, a B nie ma wpisanego A jako sąsiada, to B będzie uważał A jako sąsiada dopiero po pozyskaniu wiedzy lokalnej od A.

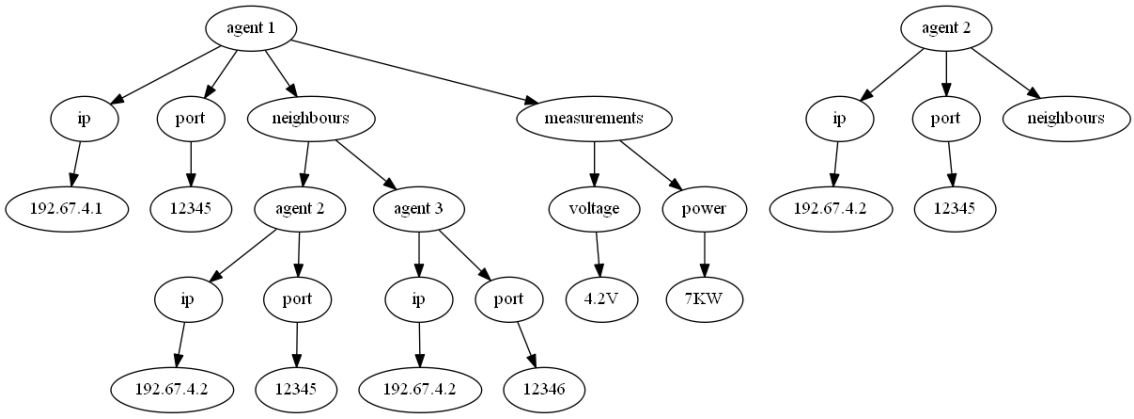
Na Rysunku 2 przedstawiono przykładową wiedzę globalną widzianą z perspektywy agenta “agent 2” działającego w systemie, w którym uruchomiony jest jeszcze dodatkowy agent “agent 1” i “agent 3”. Pomimo, że “agent 2” nie ma wpisanego żadnego agenta jako sąsiada (węzeł “neighbours” jego wiedzy lokalnej jest pusty), uważa on agenta “agent 1” za swojego sąsiada od czasu pozyskania wiedzy agenta “agent 1” (w posiadanej wiedzy globalnej “agent 1” ma wpisanego w węźle “neighbours” agenta “agent 2”).

Należy zaprojektować aplikację agenta, która komunikując się z sąsiednimi agentami potrafi utrzymywać wiedzę rozproszoną. Wymagana jest następująca funkcjonalność:

- wczytanie inicjalnej wiedzy z pliku konfiguracyjnego
- dodawanie/usuwanie/modyfikacja wiedzy lokalnej w agencie
- wymiana wiedzy globalnej pomiędzy bezpośrednimi sąsiadami,
- prezentacja wiedzy lokalnej i globalnej na stronie www danego agenta (można użyć programu Graphviz do wygenerowania obrazków z drzewami wiedzy).

Specyfikacja

Aplikacja agenta powinna być uruchamiana z czterema parametrami: ścieżką do pliku z inicjalną konfiguracją agenta, ścieżką do pliku ze stroną w formacie HTML, na której będą



Rysunek 2: Przykładowa wiedza globalna widziana z perspektywy “agent 2”.

prezentowane informacje o wiedzy globalnej agenta, adresem oraz portem po którym można zarządzać wiedzą lokalną. Plik konfiguracyjny ma opisywać drzewo z wiedzą początkową agenta w następującej składni:

```
drzewo ::= "string"(drzewo, ..., drzewo) | "string"
```

Gdzie *string* to napis z etykietą węzła. Dla przykładu, plik z wiedzą początkową odpowiadającą Rysunkowi 1 ma następującą postać (białe znaki poza napisami w etykietach są nieistotne):

```
"agent 1"(
    "ip"("192.67.4.1"),
    "port"("12345"),
    "neighbours"(
        "agent 2"(
            "ip"("192.67.4.2"),
            "port"("12345")),
        "agent 3"(
            "ip"("192.67.4.1"),
            "port"("12346")),
    "measurements"(
        "voltage"("4.2V"),
        "power"("7KW")))
```

Dodawanie/usuwanie/modyfikacja wiedzy lokalnej agenta ma odbywać się za pomocą komend tekstowych wysyłanych na gniazdo TCP wyspecyfikowane podczas uruchomienia agenta. Komendy mają mieć następującą postać:

```
komenda ::= (ADD | REMOVE | MODIFY) "string" "string" ... "string"
```

Semantyka komend jest następująca:

- `ADD "etykieta 1" "etykieta 2" ... "etykieta n"`

  dodaje do drzewa ścieżkę *"etykieta 1"*, *"etykieta 2"*, ..., *"etykieta n"*, gdzie *"etykieta 1"* jest etykietą korzenia drzewa.

- `REMOVE "etykieta 1" "etykieta 2" ... "etykieta n"`

  usuwa z drzewa całe poddrzewo począwszy od wierzchołka *"etykieta n"* identyfikowanego przez ścieżkę *"etykieta 1"*, *"etykieta 2"*, ..., *"etykieta n"*, gdzie *"etykieta 1"* jest etykietą korzenia drzewa.

- `MODIFY "etykieta 1" "etykieta 2" ... "etykieta n" "nowa etykieta"`

  zmienia etykietę węzła "etykieta n" identyfikowanego przez ścieżkę *"etykieta 1"*, *"etykieta 2"*, ..., *"etykieta n"*, gdzie *"etykieta 1"* jest etykietą korzenia drzewa, na etykietę *"nowa etykieta"*

Dla przykładu, wysłanie komendy:

```
ADD "agent 1" "misc" "circle" "radius" "22"
```

do agenta "agent 1" o drzewie wiedzy lokalnej jak na Rysunku 1, zmieni jego wiedzę do stanu z Rysunku 3, późniejsze wysłanie komendy:

```
MODIFY "agent 1" "misc" "circle" "radius" "diameter"
```

doprowadzi do wiedzy jak na Rysunku 4, a po komendzie:

```
REMOVE "agent 1" "misc" "circle" "diameter"
```

otrzymamy wiedzę jak na Rysunku 5.

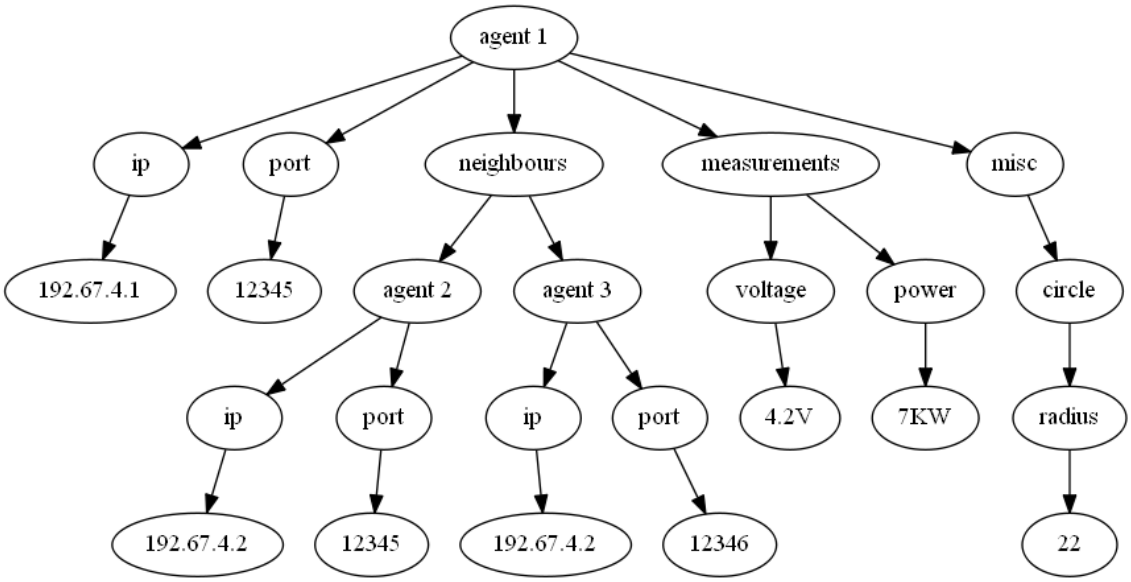
Każdy agent odpowiada tylko i wyłącznie za swoją wiedzę lokalną — nie może bezpośrednio modyfikować wiedzy globalnej.

Protokół wymiany wiedzy pomiędzy agentami powinien odbywać się według następujących zasad:

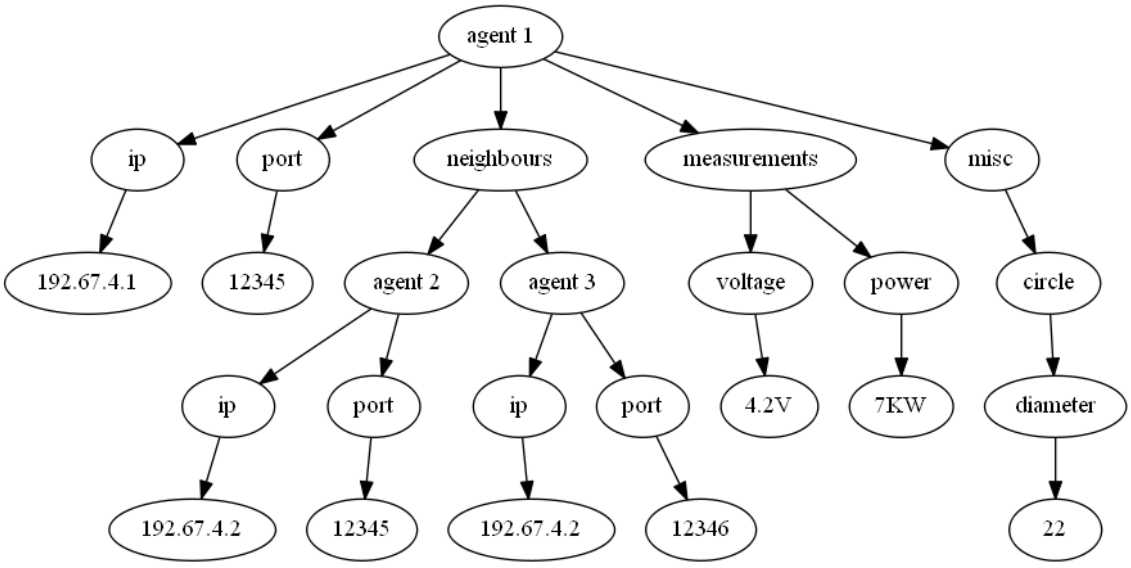
- każde drzewo wiedzy agenta jest rewizjonowane (rewizja wiedzy lokalnej zmienia się poprzez modyfikację wiedzy za pomocą komend ADD/REMOVE/MODIFY, a rewizja pozostałej wiedzy w procesie komunikacji z innymi agentami; każdy agent odpowiada za spójność rewizji swojej wiedzy lokalnej)

- jeżeli agent zmienia rewizje to wysyła informacje do swoich sąsiadów o nowej rewizji

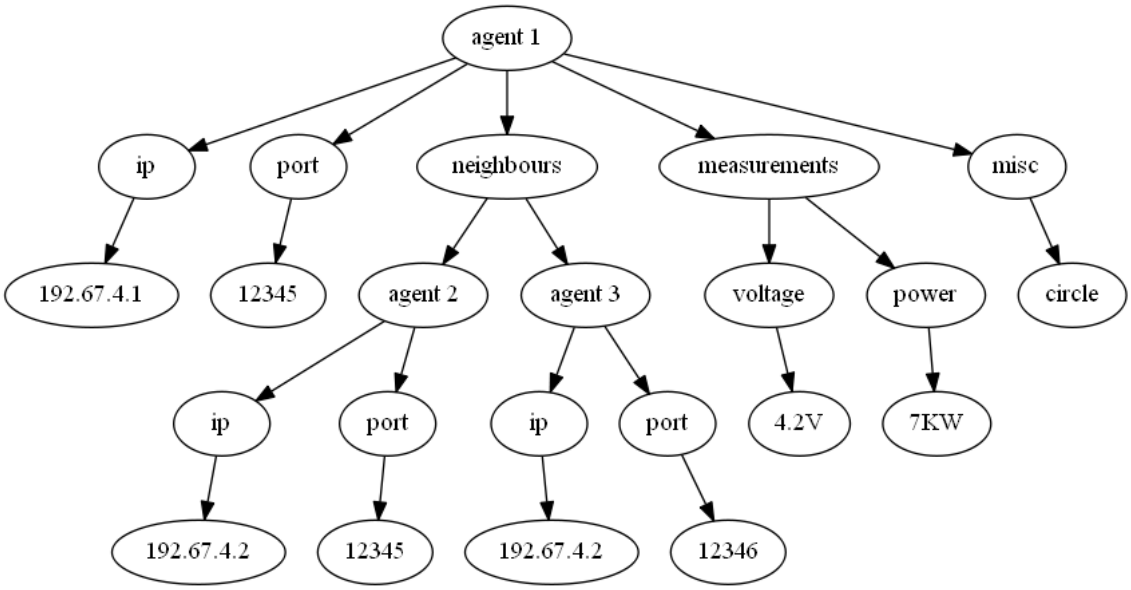
- jeżeli za pomocą komendy REMOVE został usunięty sąsiad danego agenta, to usunięty sąsiad jest także informowany o zmianie rewizji (i może cały czas prosić o dosłanie brakujących danych)



Rysunek 3: Modyfikacja wiedzy komendą ADD.



Rysunek 4: Modyfikacja wiedzy komendą MODIFY.



Rysunek 5: Modyfikacja wiedzy komendą REMOVE.

- jeżeli agent nie może przesłać informacji o rewizji do któregoś ze swoich sąsiadów, to co ustaloną jednostkę czasu ponawia próbę komunikacji
- jeżeli agent A otrzyma informacje o rewizji nowszej, niż posiadana od agenta B, to pobiera od agenta B brakującą wiedzę (tym samym zmieniając rewizję i informując o tym sąsiadów)

Protokół powinien być zoptymalizowany pod względem ilości danych przesyłanych przez sieć. W szczególności nisko punktowane będą projekty, w których agenci po każdej modyfikacji wysyłają na nowo całą swoją lokalną wiedzę. Warto zwrócić uwagę, że graf agentów może w pewnym momencie się rozspójnić, ewoluować osobno, po czym znowu połączyć się w spójny graf. Protokół musi zapewnić spójność wiedzy globalnej także po takim ponownym połączeniu.

Zadanie

1. Poprawny i pełny projekt wart jest 6 punktów (plus 2 dodatkowe za zrealizowanie opcjonalnej funkcjonalności):
   - za funkcjonalność obejmującą utrzymywanie wiedzy lokalnej agenta (wczytanie pliku konfiguracyjnego agenta, utrzymywanie lokalnego drzewa wiedzy za pomocą komend ADD/REMOVE/MODIFY) można otrzymać do **2 punktów**.
   - za prezentowanie wiedzy w formacie strony HTML można otrzymać do **1 punktu**.

  - Jeśli programista zdecyduje się na dodatkową implementację serwera HTTP, za pomocą którego agent będzie udostępniał swoja wiedzę w sieci (po połączeniu na dodatkowy port podany do konfiguracji podczas startu programu agenta), może otrzymać dodatkowo maksymalnie **2 punkty**.

- za prawidłowo i efektywnie zaimplementowany mechanizm rewizjonowania i wymiany wiedzy pomiędzy agentami można otrzymać do **3 punktów**. W tym przypadku sprawdzający bedą zwracali uwagę na informacje, które są przesyłane, sposób ich przesyłania w grafie połączeń (nie tylko kwestię implementacji obsługi sieci, ale również minimalizację liczby przesyłanych komunikatów).

2. Aplikację agenta piszemy w języku Java zgodnie ze standardem Java 8 (JDK 1.8).

3. Projekty powinny zostać zapisane do odpowiednich katalogów w systemie EDUX w nieprzekraczalnym terminie 20.11.2016 (termin może zostać zmieniony przez prowadzącego grupę).

4. Spakowany plik projektu powinien obejmować:

   - pełen opis protokołu komunikacyjnego pomiędzy agentami, który powinien zawierać opis i fomat kominikatów przesyłanych między węzłami (co i w jakiej sytuacji jest przesyłane),
   - pliki źródłowe (dla JDK 1.8) (włącznie z wszelkimi bibliotekami nie należącymi do standardowej instalacji Javy, których autor użył) - aplikacja musi dać się bez problemu skompilować na komputerach w laboratorium w PJA.
   - plik *Readme.txt* z opisem i uwagami autora (co zostało zrealizowane, czego się nie udało, gdzie ewentualnie są błędy, których nie udało się poprawić).

5. Prowadzący oceniać będą w pierwszym rzędzie organizację komunikacji sieciowej i poprawność utrzymywanej wiedzy globalnej, ale na ocenę wpływać będzie także zgodność wytworzonego oprogramowania z zasadami inżynierii oprogramowania i jakość implementacji.

6. JEŚLI NIE WYSZCZEGÓLNIONO INACZEJ, WSZYSTKIE NIEJASNOŚCI NALEŻY PRZEDYSKUTOWAĆ Z PROWADZĄCYM ZAJĘCIA POD GROŹBĄ NIEZALICZENIA PROGRAMU W PRZYPADKU ICH NIEWŁAŚCIWEJ INTERPRETACJI.