UFoo - Hľadanie podobnosti - Pseudo kód

Nasledujúci pseudo-kód popisuje približné fungovanie algoritmu pre hľadanie podobností jednotlivých requestov - *uFooEntity* v zozname už uložených entít - *allStockItems*. Celá funkcionalita je rozložená do dvoch algoritmov:

- Pvým z nich je *get-nearest-neigbour*, ktorého úlohou je nájsť v zozname *allStockItems* položku z najmenšou vzdialenosťou.
- Pre porovnávanie samotných entít sa používa práve druhý z algoritmov - *getDistance* na určenie vzdialenosti medzi dvoma položkami. Tento algoritmus po rozparsovaní informácií počíta u hlavičiek *Jaccardov index* a u ostatných atribútov porovnáva vzájhomné hodnoty metódou *similarValue*. Všetky hodnoty sú vážené, najväčšiu váhu budú mať *staticHeaders*, následne *unknownHeaders* a ďalej jednotlivé atributy.

get-nearest-neigbour

```
algorithm get-nearest-neigbour is
    input: UFooEntity uFooEntity
           UFooEntity[] allStockItems
    output: Result<int distance, UFooEntity nearestNeigbour>

    minDistance ← 2;
    nearestneigbour ← null;

    for each stockUFooEntity in allStockItems do
        # Compute distance for actual items
        acctualDistance ← getDistance(uFooEntity.staticData, stockUFooEntity.staticData);

        # Addition analysis - manual checks based on Relation Data
        # Possibly can SLIGHTLY afffect distance

        # Remember actual distance and item if it is new minimum
        if minDistance < acctualDistance then
            minDistance ← acctualDistance;
            nearestneigbour ← stockUFooEntity;
        end if
    end do

    return Result<minDistance, nearestneigbour>
```

- Jaccardov index (metóda *jaccardIndex*) bude počítaný vždy pre dve **množiny** štandartným vzťahom: $J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$.
- Metóda *similarValue* nebude porovnávať hodnoty priamo, ale po optimalizáciách (odstránene bielych znakov, *ignoreCase*, …). Na rozdiel od metódy *jaccardIndex* však *similarValue* vráti len 1 pre zhodu, alebo 0 pre rozdielne hodnoty.

getDistance

```
algorithm get-distance is
    input: String actualData
           String stockEntityData
    output: double distance

    if actualData = stockEntityData
        return 0
    end if

    # Compute Jaccard and consider weight (strength) for static headers
    jaccardStaticHeaders ← jaccardIndex(actualData.staticHeaders, stockEntityData.staticHeaders);
    jaccardStaticHeaders ← jaccardStaticHeaders × SH_WEIGHT;

    # Compute Jaccard and consider weight (strength) for unknown headers
    jaccardUnknownHeaders ← jaccardIndex(actualData.unknownHeaders, stockEntityData.unknownHeaders);
    jaccardUnknownHeaders ← jaccardUnknownHeaders × UH_WEIGHT;

    # Compare values of remaining attributes by simple similarValue function
    # Every of this attributes will have sub-weight which will be computed in attributesIndex
    isSameIp ← similarValue(actaulData.IP, stockEntityData.IP) × IP_SUB_WEIGHT;
    isSameCountry ← similarValue(actaulData.country, stockEntityData.country) × COUNTRY_SUB_WEIGHT;
    ...
    isSameLength ← similarValue(actaulData.length, stockEntityData.length) × LENGTH_SUB_WEIGHT;

    # AttributesIndex based on SUM of weighted attributes is computed
    # Additional weight for all attributes is applied
    attributesIndex ← ∑(isSameIp, isSameCountry, ..., isSameLength) / ∑(IP_SUB_WEIGHT, COUNTRY_SUB_WEIGHT, ..., LENGTH_SUB_WEIGHT);
    attributesIndex ← attributesIndex × ATTR_WEIGHT

    # Final similarity value is computed as sum of weighted: staticHeaders, unknownHeaders and attributes
    # Distance as 1 - sim value is returned
    similarity ← ∑(jaccardStaticHeaders,  jaccardUnknownHeaders, attributesIndex) / ∑(SH_WEIGHT, UH_WEIGHT, ATTR_WEIGHT)

    return 1 - similarity;
```