FAKULTA
APLIKOVANÝCH VĚD
ZÁPADOČESKÉ
UNIVERZITY
V PLZNI

Umělá inteligence a rozpoznávání

Expertní genealogický systém

Vypracoval: Zdeněk Janeček

Datum: 14. května 2013

1 Zadání

Vytvořte genealogický expertní systém. Vstupem bude informace databáze základních informací o osobě [id, jméno, matka, otec, pohlaví, partner]. Program bude umět odpovídat na otázky: Kdo je můj pokrevní příbuzný? Kolik má moje prateta sestřenic? Jaký je vtah mezi Petrem a Pavlem?... a další podobné.

2 Analýza

Rodinný strom je z hlediska grafové teorie n-ární strom, protože mám dva rodiče a neomezeně mnoho dětí. Pokud zanedbám manželství a postavím strom jen ze vztahů rodič–potomek, bude požadovaná acykličnost grafu zachována. Manželství není stálé a stačí si ho tedy pouze zaznamenat. Společné děti manželů, také nejsou jisté, tudíž je praktičtější ukládat si potomky u každého rodiče zvlášť.

Jak tedy dobře reprezentovat tento strom aby se dobře prohledával? První co mě napadne je, že každý uzel bude jeden objekt a bude mít odkazy na své potomky a rodiče. Na vyhodnocování zvolím algoritmy jako na každý graf. Mohl jsem ale jít cestou znalostní báze. Ta by shromažďovala všechna známá i nově nalezená fakta. Tento přístup by byl určitě flexibilnější, ale na něj by byl vhodnější jazyk *Prolog*. Měl jsem nápad že by byl vhodný také konečný automat. Nakonec něco na ten způsob jsem také trochu použil. Nemělo cenu zavádět gramatiku.

Abych mohl ukládat rodinný strom, je třeba stanovit jeho kořen. Mám tedy dvě varianty. Kořenem bude jeden společný potomek, a nebo společná matka. Zvolil jsem jako kořenový uzel supermatku, která dostala id 0.

Pro ukládání dat o rodině jsem vytvořil jednoduchý formát souboru. Na obrázku 1 vidíte následující položky:

0. ID
1. Jméno
2. (červeně) id natky
3. (azurová) id otce
4. (zelené) true – muž, false – žena
5. (modře) partner, nemá partnera pak *nil*

Když víme počet položek z prvního řádku, algoritmus je pak jednoduchý:

```
9
[1, Kocourek Petr, 4, 3, true, nil]
[4, Kocourková Ivana, 8, 9, false, 3]
```

Obrázek 1: Formát souboru s rodinnými daty.

1. načti atributy
2. když neexistuje osoba, vytvoř její základ
3. přidej se jako dítě rodiče a nastav partnera. Pokud ještě neexistuji, vytvoř je
4. aktualizuj informace o sobě
5. opakuj krok 1 pro další osobu

Jakmile máme vytvořený strom, lze na něm dělat standardní prohledávání a sledovat, zda jsme dostáli požadovaného výsledku.

3 Vyhodnocování

Při zápisu výstupu byly použity escape sekvence, které umožňují mazat kusy obrazovky a tím dosáhnout určité interaktivity. Budou stačit dva způsoby vstupu a to pro *id osoby* a požadovaný vztah. V hlavičce programu se bude vypisovat formulovaná otázka. Více v sekci 5 na straně 6.

Uvnitř programu zastupuje třída `NTree` rodinný strom. Tato třída je konstruována s parametrem souboru s rodinnými daty. Při vytvoření se soubor načte a vytvoří pole instancí třídy `Node`. Nyní máme k dispozici například metodu `void printTree()`, která vytiskne strom na obrazovku. Každý `Node` představuje jednu osobu. Procházení stromu je pak rychlé, protože můžu využít jak tabulku, tak odkaz na sousední uzel. Veškeré dotazy na uzel se pak provádí v čase $\mathcal{O}(1)$.

Další část je práce s expertními daty. Bylo třeba navrhnout jak reprezentovat rodinný vztah. Využil jsem predikátové logiky, která je dostatečně univerzální pro práci s množinou prvků. Definoval jsem například:

$$P(x,a) \wedge P(a,b) \wedge C(b,c) \wedge N(c,a) \rightarrow c \text{ je sourozenec rodičů} \qquad (1)$$

Vystačil jsem si se třemi funkcemi říkající:

2
[3, prarodič, dědeček, babička]
PxaPab
[4, sourozenec rodičů, strýc, teta]
PxaPabCbcNca

Obrázek 2: Formát souboru s expertními daty.

P(a,b) rodiče termu *a* jsou *b*.

C(a,b) děti termu *a* jsou *b*.

N(a,b) term *a* neobsahuje prvky z *b*.

Soubor jsem navrhl podobně. Stačilo se zbavit závorek a konjunkcí, které budou platit vždy a není tedy nutné je ukládat. Soubor má formát podle obrázku 2. První řádek je opět počet položek v tomto souboru. Každé pravidlo má dva řádky. První z nich obsahuje popis vztahu a na dalším je samotné pravidlo.

Nyní bude následovat popis vnitřních částí třídy `Expert`. Tyto metody volá třída `Starter`, která zobrazuje výsledky otázek.

3.1 Využité nástroje

Užitečná je metoda na nalezení id vztahu, pokud mám zjednodušené pravidlo bez termů jako například PCC místo PabCbcCcx.

```
public int getRealationId(String rule) {
    for (int i = 0; i < relations.length; i++) {
        if (compareRel(relations[i].rule, rule)) {
            return i;
        }
    }
    return -1;
}
```

3.2 Metoda findPerson

Pokud potřebuji odpovědět na otázky typu:

V jakém vztahu jsou Petr a Ivana?

```
/**
 * Searches for some person and returns their relation.
 *
 * @param from Person to search from.
 * @param target Person to search for.
 * @return result of search
 */
public String findPerson(Node from, Node target) {}
```

Prohledáváním do šířky od uzlu `from` přidávám každému nově nalezenému uzlu záznam o jeho cestě. Ukládám si pole `paths`, kde jsou řetězce ve stylu PCC. Do fronty `q` přidávám jak jeho děti, tak rodiče.

Naleznu-li požadovanou osobu, najdu jakému odpovídá vztahu a vrátím řetězec podle vzoru:

> Nalezen [jmeno] ve vztahu [vztah]

3.3 Metoda findPersons

Pokud potřebuji odpovědět na otázky typu:

> Kdo všechno jsou bratranci zadané osoby?

```
/**
 * Finds all persons in specified relation.
 *
 * @param from Person to search from.
 * @param rel wanted relation
 * @param male is male or female
 * @return LinkedList of person instances
 */
public Collection<Node> findPersons(Node from,
    int rel, boolean male) {}
```

Prohledáváním do šířky od uzlu `from` přidávám každému nově nalezenému uzlu záznam o jeho cestě. Ukládám si pole `paths`, kde jsou řetězce ve stylu PCC. Do fronty `q` přidávám jak jeho děti, tak rodiče.

Rezoluce, zda `v` je hledaná osoba, vypadá následovně:

```
if (v.isMale()==male && compareRel(rule, paths[v.getId()])) {
    result.add(v);
}
```

Vzniklý seznam `result` vracím jako návratová hodnota.

3.4 Metoda testRelation

Pokud potřebuji odpovědět na otázky:

Jsou tyto osoby v daném vztahu?

```
/**
 * @param from From which person is done an comparison.
 * @param target To which person is done an comparison.
 * @param rel relation id we want to test
 * @param male true if I look for male, false if it is female
 *
 * @return true if it founds the person and the false in
 *         the other case
 */
public boolean testRelation(Node from, Node target,
    int rel, boolean male) {}
```

V tomto případě se nejedná o slepé prohledávání, ale přesně podle daného pravidla. Vedu si evidenci o každém symbolu, který jsem našel. Mohu se tedy pak vracet k nějaké starší množině lidí. Při provádění pravidla využívá funkcí jak byly definovány ve vzoru 1. První znak je zdroj a druhý je cíl. Na konci se musí hledaná osoba nalézat v poslední zpracované množině.

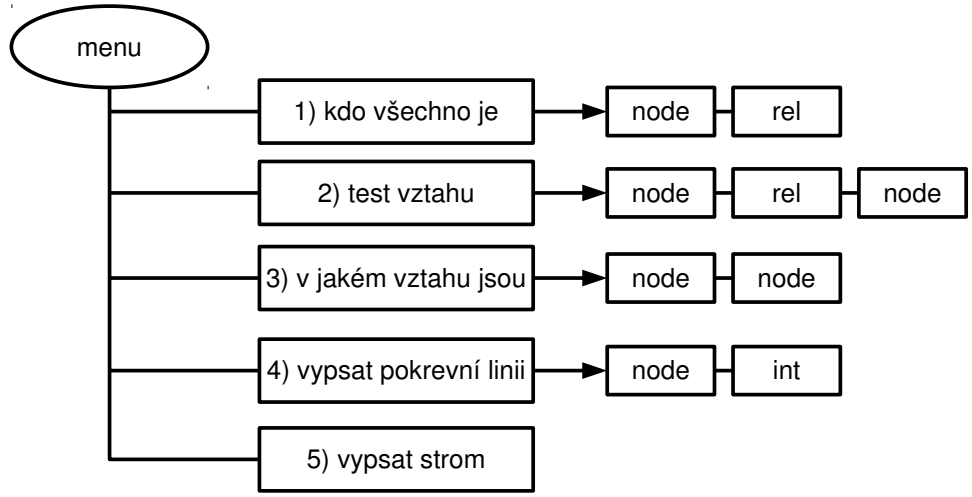
4 Metoda bloodLine

Pokud potřebuji odpovědět na otázku:

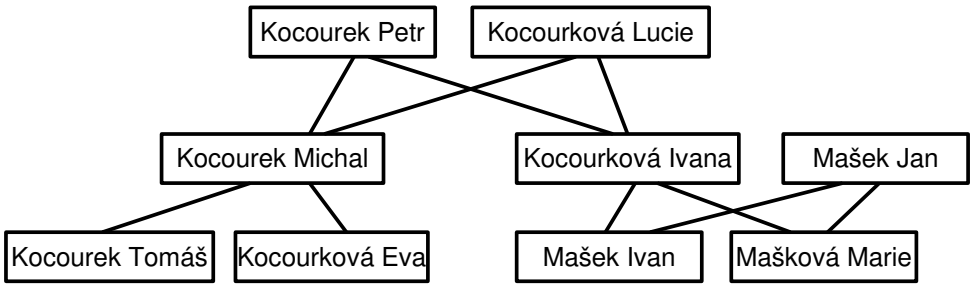
Kdo jsou pokrevní příbuzní dané osoby?

```
/**
 * Get your blood line. They are all parents of parents
 * and so on.
 * @param from Person to search from.
 * @param deep how deep you want to search
 * @return all persons that are my ascendants
 */
public Collection<Node> bloodline(Node from, int deep) {}
```

S použitím dvou stavových proměnných a jedním čítačem je omezeno prohledávání do šířky. Prohledávání se líší od předchozích jen tím, že přidávám do fronty jen rodiče.



Obrázek 3: Typy vstupních dat.



Obrázek 4: Vzorová rodinná data.

5 Interakce s uživatelem

O uživatelské prostředí se stará třída `Starter`. Je tedy nutné používat terminál, který je podporuje ANSI ecape kódy[1].

Vytvořil jsem obrázek 3, na kterím lze snadno vyčíst jaký typ dat je očekáván pro jednotlivé otázky. Blok node otázku na osobu, rel otázku na vztah (např. bratranec) a poslední int načítání celočíselné hodnoty.

Pro účely testování je možné použít soubor familyData, ve kterém je uložen rodokmen z obrázku 4.

[1] `<http://en.wikipedia.org/wiki/ANSI_escape_codes>`

6 Spouštění a obsluha

K dispozici je popisový soubor build.xml. Pro překlad a spuštění stačí spustit:

```
$ ant
$ java -cp bin/ kiv.janecekz.Starter
```

Pokud po aplikace požaduje nějaký vstup, poznáte to podle znaků >>. Vstupem může být číslo, nebo řetězec. To je rozepsáno na obrázku 3.

7 Shrnutí

Při psaní jsem několikrát lehce změnil návrh. V počátku jsem o expertním systému nevěděl vůbec nic, takže jsem použil již ověřené postupy. Aplikace byla napsána pomocí Eclipse IDE a dokumentace vysázena systémem LaTeX. Projekt má svůj Git repositář na adrese `<https://github.com/qwertzdenek/genealogy>`.