Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Bakalárska práca

Inteligentné vyhľadávanie v systéme na evidenciu skautských družinových hier

Smart searching in system for managing group games

2014 Richard Dvorský

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Bakalárska práca

Inteligentné vyhľadávanie v systéme na evidenciu skautských družinových hier

Smart searching in system for managing group games

Študijný program:	Informatika
Študijný odbor:	9.2.1. Informatika
Číslo študijného odboru:	2508
Školiace pracovisko:	FMFI.KI - Katedra informatiky
Školiteľ:	RNDr. Michal Rjaško PhD.

2014 Richard Dvorský

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Richard Dvorský
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: 9.2.1. informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský

Názov: Inteligentné vyhľadávanie v systéme na evidenciu skautských družinových hier
Smart searching in system for managing group games

Cieľ: Navrhnúť a naprogramovať inteligentné vyhľadávanie požadovanej družinovej hry na základe vety napísanej v prirodzenom jazyku. Analyzovať efektivitu takéhoto vyhľadávania.

Vedúci: RNDr. Michal Rjaško, PhD.
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: doc. RNDr. Daniel Olejár, PhD.

Dátum zadania: 23.10.2013

Dátum schválenia: 24.10.2013 doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Čestné prehlásenie

Čestne prehlasujem, že túto bakalársku prácu som vypracoval samostatne s použitím uvedených zdrojov.

.....
Richard Dvorský

Poďakovanie

Chcem sa poďakovať vedúcemu bakalárskej práce RNDr. Michalovi Rjaškovi PhD. za cenné rady a pripomienky, rodine a blízkym priateľom za pomoc a morálnu podporu.

Richard Dvorský

Abstrakt

Cieľom mojej bakalárskej práce bolo vytvoriť program inteligentného vyhľadávania v systéme na generovanie skautských družinoviek.
Program inteligentného vyhľadávania dokáže analyzovať a predvídať vstup dát v prirodzenom jazyku a na základe toho vygenerovať program na skautskú družinovku.

Kľúčové slová: generátor družinoviek, inteligentné vyhľadávanie, PHP, JavaScript

Abstract

The goal of my thesis was to create the program of smart searching in system for managing scout group games. This program is able to detect and foretell the entrence of dates v natural language and thanks to this ability it knows how to create scout group games.

Keywords: scout group games, smart search, PHP, JavaScript

Obsah

Úvod

Slovenský skauting je mládežnícka nezisková organizácia, ktorá má okolo 6000 členov. Takmer 1000 dobrovoľníkov sa stará o výchovu, neformálne vzdelávanie a zmysluplné využitie volného času detí a mladých ľudí v 120 mestách a obciach na Slovensku. (13)

Každý týždeň majú skauti a skautky stretnutia, ktoré vždy prichystajú skautský vedúci. Na uľahčenie vymýšľania programu na skautské stretnutia vznikol projekt Generátor družinoviek, ktorý dokáže zostaviť program podľa vybraných parametrov.

Moja bakalárska práca rozširuje a uľahčuje komunikáciu užívateľa s Generátorom družinoviek tak, že užívateľ môže generovať program napísaním vety v slovenskom jazyku. Program dokáže takú vetu analyzovať a následne pochopiť, čo užívateľ chce vygenerovať. Inteligentné vyhľadávanie sa učí používaním a dokonca vie predpovedať vstup, ktorý užívateľ chce napísať.

Bakalárska práca je rozdelená do siedmich kapitol.

1. V prvej kapitole sa venujeme základným pojmom, ktoré je potrebné poznať, nakoľko sa budú používať v ďalších častiach bakalárskej práce.

2. V druhej kapitole sa zaoberáme matematickou teóriou, ktorú využívame v inteligentnom vyhľadávaní.

3. V tretej kapitole sa venujeme technológiám, ktoré boli použité na realizáciu bakalárskej práce

4. V štvrtej kapitole zoznamujem čitateľa s ročníkovým projektom - Generátorom družinoviek.

5. V piatej kapitole rozoberáme fungovanie inteligentného vyhľadávania

6. V šiestej kapitole opisujeme kompletnú realizáciu inteligentného vyhľadávania od databázy, až ku častiam zdrojového kódu.

7. V siedmej kapitole rozoberáme problémy, ktoré bolo treba vyriešiť

8. V ôsmej kapitole rozoberáme efektivitu inteligentného vyhľadávania

Všeobecné pojmy

V nasledujúcich častiach úvodu si zadefinujeme základné pojmy, ktoré budem ďalej v práci uvádzať a sú nevyhnutné k správnemu porozumeniu ďalších častí.

Skautská družina

Družina je základná stavebná častica skautskej organizácie. Združuje deti vo veku 11 až 18 rokov. Spravidla ju tvorí jeden radca, jeden zástupca radcu a členovia. Väčšinou má družina okolo 6 až 8 členov.

Družinovka

Pod pojmom družinovka sa rozumie skautská schôdzka, ktorá sa koná raz do týždňa. Každá družinovka má program, ktorý sa skladá z viacerých hier a náučných činností.

Generátor družinoviek

Generátor družinoviek je nástroj na generovanie programu družinoviek, ktorý som vytvoril ako ročníkový projekt. Funguje na rozsiahlej databáze hier, náučných činností, ktoré obsahujú spoločné atribúty, podl'a ktorých vie generátor generovať skautské družinovky.

Aktivita

Pod pojmom aktivita rozumieme rôznorodú činnosť, ktorú obsahuje Generátor družinoviek v databáze.

Typ aktivity

Aktivity v databáze majú rôzny typ. Napríklad to môže byť: Hra, Intro, Zamyslenie …

Parametere družinovky

Každá družinovka obsahuje niekol'ko parametrov. Medzi bežné parametre patria: Miesto, Čas, Zameranie, Veková skupina …

Teória inteligentného vyhľadávania

Markovský rozhodovací proces

MDP (Markov Decision Process) je pomenovaný po ruskom matematikovi Andreji Markovovi. Poskytuje matematický rámec na modelovanie rozhodnutí v situáciach, kde následky sú z časti náhodné a z časti pod kontrolou užívateľa. (5) (6) (10)

Formálne môžme MDP zapísať takto:

Štvorica (S, A, R, T)*:*

- $S = \{S_0, S_1, \ldots, S_n\}$ je konečná množina stavov, kde $s \in S$
- $A = \{A_0, A_1, \ldots, A_n\}$ je konečná množina akcií, kde $a \in A$ (Alebo A_s je konečná množina akcií, ktoré sú možné zo stavu S)
- $P_a(s,s') = Pr(s_{t+1} = s' \mid s_t = s, a_t = a)$ je pravdepodobnosť, že ak v stave s a čase t vykonáme akciu a, tak potom sa v čase $t+1$ dostaneme do stavu s'
- $R_a(s,s')$ je okamžitá, alebo očakávaná odmena po prejdení zo stavu s do stavu s' s pravdepodobnosťou $P_a(s,s')$

V Markovskom rozhodovacom procese je proces v každom časovom okamihu v nejakom stave s.
V tomto stave si užívateľ môže vybrať ľubovolnú akciu a, ktorá je dostupná v stave. Pravdepodobnosť že sa dostane do stavu s', záleží od pravdepodobnostnej funkcie $P_a(s,s')$ a od akcie užívateľa. Po presunutí do stavu s', užívateľ dostáva odmenu $R_a(s,s')$

Markovov reťazec

DTMC (discrete-time Markov chain) je náhodný proces s diskrétnou množinou stavov, pre ktorý platí, že pravdepodobnosť prechodu do ďalšieho stavu, záleží iba na súčasnom stave a nie na predchádzajúcich stavov. (3) (11)

Formálne môžme DTMC zapísať takto:

Markovov reťazec s diskrétnym časom je postupnosť náhodných premenných

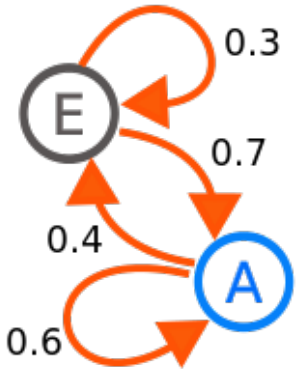
$X_0, X_1, \ldots, X_n \; kde \; X_n \in S$

pre ktoré platí:

$Pr(X_{n+1} = s \mid X_1 = s_1, X_2 = s_2, \ldots, X_n = s_n) = Pr(X_{n+1} = s \mid X_n = s_n)$

$S = \{0, 1, , \ldots, n\}$ je množina n stavov.

Budúci stav Markovho reťazca časte $t+1$ zavisí iba od stavu v čase t a nie od predchádzajúcich stavov, čo sa niekedy označuje ako *Markovská vlastnosť.*



Obr. 1: Stavový diagram Markovovho reťazca (11)

Markovov reťazec si vieme znázorniť stavovým diagramom, kde z každého stavu vychádzajú hrany možných prechodov do ďalších stavov s určitou pravdepodobnosťou. Takýto systém vieme reprezentovať deterministickým konečným automatom.

Deterministický konečný automat

DFA (deterministic finite automaton) je teoretický model, ktorý popisuje jednoduchý počítač. Automat sa môže nachádzať v jednom stave z množiny možných stavov. Medzi stavmi vie prechádzať prechodovou funkciou. Množina stavov obsahuje začiatočný stav, kde automat začína a potom množinu (akceptačných) konečných stavov, kde automat končí. (4) (12)

Formálne môžme DFA zapísať takto:

Pätica $(K, \Sigma, \delta, q0, F)$*:*

K je konečná množina stavov,

Σ je konečná vstupná abeceda,

$q0 \in K$ je začiatočný stav,

$F \subseteq K$ je množina akceptačných (koncových) stavov,

$\delta : K \times \Sigma \to K$ je prechodová funkcia.

Medzi stavmi vie prechádzať prechodovou funkciou. Množina stavov obsahuje začiatočný stav, kde automat začína a potom množinu (akceptačných) konečných stavov, kde automat končí.

Levenshteinova vzdialenosť

LD (Levenshtein distance) sa veľmi často používa, keď chceme zistiť mieru podobnosti dvoch reťazcov. Táto metóda meria minimálny počet potrebných úprav na zmenu jedného reťazca na druhý reťazec. Používa tri rôzne druhy úprav: (7)

- vymazanie znaku (remove)
- nahradenie znaku (substitute)
- vloženie znaku (insert)

Formálne môžme LD pre dva reťazce A, B zapísať takto:

$$LD(A,B) = \min\{r(j) + s(j) + i(j)\}$$

Reťazec B dostaneme z reťazca A pomocou $r(j)$ vymazaní, $s(j)$ substitúcii, $i(j)$ vložení jednotlivých znakov. Existuje nekonečne veľa kombinácií vymazania, substitúcie a vloženia. My potrebujeme nájsť kombináciu s minimálnu vzdialenosťou. To riešime pomocou dynamického programovania.

V inteligentnom vyhľadávaní používame váhovú Levenshteinovú vzdialenosť (weighted Levenshtein distance - WLD), ktorá umožňuje lepšie porovnávať reťazce.

Formálne môžme WLD pre dva reťazce A, B zapísať takto:

$$WLD(A,B) = \min\{a*r(j) + b*s(j) + c*i(j)\}$$

Kde koeficienty a,b,c sú váhové konštanty, ku jednotlivým operáciám.

Technológie v inteligentnom vyhľadávaní

PHP

Webová aplikácia Generátor družinoviek je napísaný v objektovom jazyku PHP, preto aj inteligentné vyhľadávanie budem písať v jazyku PHP.

PHP (Hypertext Preprocessor) je skriptovací OpenSource jazyk na strane servera, ktorý sa veľmi často používa na vývoj internetových aplikácii. Je rozsiahly a obsahuje veľa knižníc. Na zvýšenie rýchlosti PHP, budem používať PHP cache, ktorý si ukladá už skompilované PHP skripty, takže sa nemusia kompilovať pri každom načítaní stránky, čo značne zvyšuje rýchlosť aplikácie. (2) (8)

Použitie jazyka PHP na Inteligentné vyhľadávanie spolu s Generátorom družinoviek má niekoľko výhod.

- Môžem využívať frameworky, ktoré už obsahuje Generátor
- Môžem vytvárať triedy, ktoré sú potomkami tried v Generátore

Javascript a jQuery

Inteligentné vyhľadávanie sa bude snažiť spracovávať vstupy, ako bude užívateľ písať. Preto budem využívať Javascriptový framework jQuery, cez ktorý viem veľmi jednoducho odosielať ajaxové požiadavky na server.

Javascript je skriptovací jazyk, ktorý sa spúšťa na strane klienta, v internetovom prehliadači. jQuery je rozsiahla knižnica funkcií, ktorá umožňuje lepšiu interakciu medzi HTML a Javascript. (1) (9)

MySQL

Inteligentné vyhľadávanie sa vie učiť a zdokonalovať v jazyku a preto je potreba ukladať tieto dáta. Na ukladanie využívame MySQL databázový systém.

MySQL je databázový systém, ktorý používa jazyk SQL, ktorý má menšie odlišnosti. od ostatných SQL jazykov. Veľkou výhodou MySQL je jej rýchlosť a licencia, ktorá ju umožňuje voľne šíriť.

Generátor družinoviek

Úvod

Generátor družinoviek je nástroj vytvorený pre potreby zostavovania programu na skautských družinovkách.
Vedúci družinovky (radca) si vždy pred družinovkou musí zostaviť program aktivít, ktoré bude na družinovke robiť.

Tento proces zostavovania programu na družinovku býva náročný a vyžaduje veľa času. Preto vznikol nápad vytvoriť nástroj, ktorý by výrazne zjednodušil prípravu družinoviek.

Priebeh družinovky

Bežná družinovka pozostáva z viacerých typov aktivít, ktoré sú rovnomerne začlenené do dvojhodinového programu.

I. **Intro**
je krátka úvodná aktivita na začiatku družinovky

II. **Bodovací závod**
je aktivita zameraná na vedomosti

III. **Teória**
je aktivita zameraná na učenie poznatkov

IV. **Hra**
je klasická aktivita na družinovke

V. **Zamyslenie**
je aktivita, ktorá rozvíja človeka po duchovnej stránke

VI. **Prax**
je aktivita, ktorá rozvíja praktické zručnosti

Každá aktivita rozvíja človeka po inej stránke, čo je aj cieľom skautských družinoviek -všestranný rozvoj človeka. Avšak pri zostavovaní programu sa dosť často nezahrnú všetky typy aktivít. Preto družinovky generované Generátorom družinoviek riešia tento problém a snažia sa vygenerovať program čo najpestrejší.

Parametre družinovky

Každá družinovka obsahuje rôzne parametre ktoré ju charakterizujú. Na základe týchto parametrov vieme s určitou pravdepodobnosťou danú družinovku vygenerovať v Generátore družinoviek.

Medzi základné typy parametrov družinovky patria:

1. **Čas** - koľko má v priemere družinovka trvať. Napr: 2 hodiny, 1,5 hodiny ...
2. **Rozvoj** - ktorý osobnostný rozvoj má družinovka rozvíjať. Napr: telesný, duchovný, charakterový
3. **Zameranie** - ktorým programovým zameraním sa má družinovka uberať. Napr: stromy, uzly, mapa ...
4. **Typ hry** - ktoré typy hier majú byť použité v družinovke. Napr: ekohra, kreativita, tímbilding ...
5. **Pre koho** - pre akú vekovú kategóriu je družinovka. Napr: vĺčatá (6 - 10 rokov), skauti (11 - 15 rokov), roveri (16 - 19 rokov)
6. **Miesto** - kde sa má väčšina programu odohrávať. Napr: klubovňa, les, lúka ...
7. **Obdobie** - ročné obdobie vhodné na program. Napr: jar, leto, zima
8. **Počet ľudí** - koľko ľudí sa priemerne zúčastní na programe

Medzi doplnkové typy parametrov družinovky patria:

9. **Autor** - autor ktorý vymyslel hru Napr: časopis Skaut, webstránka skauting.sk...
10. **Veci** - aké predmety sú potrebné na aktivitu Napr: kameň, papier, nožnice ...
11. **Postavy** - zoznam postáv ktoré sú potrebné na aktivitu Napr: ježibaba, pirát, policajt
12. **Typ aktivity** - aký typ aktivity potrebujeme Napr: hra, intro, družinovka …
13. **Kľúčové slová** - tie dodatočne charakterizujú aktivitu

Obr. 2: Uživatel'ské rozhranie Generátora družinoviek pre výber parametrov

Generovanie družinovky

Keď zostavujeme program na skautskú družinovku, tak si najprv určíme parametre, ktoré by našu pripravovanú družinovku mali charakterizovať. Potom sa pustíme do vymýšľania alebo hľadania takých hier, ktoré sa aspoň s niektorými parametrami zhodujú. Tento druhý krok hľadania a vymýšľania hier už za nás spraví Generátor družinoviek.

V grafickom rozhraní Generátora družinoviek si vyberieme parametre, ktoré by mala vaša družinovka obsahovať. Potom stlačíme tlačítko na vygenerovanie a generátor zobrazí vygenerovanú družinovku.

Ako funguje Generátor družinoviek?

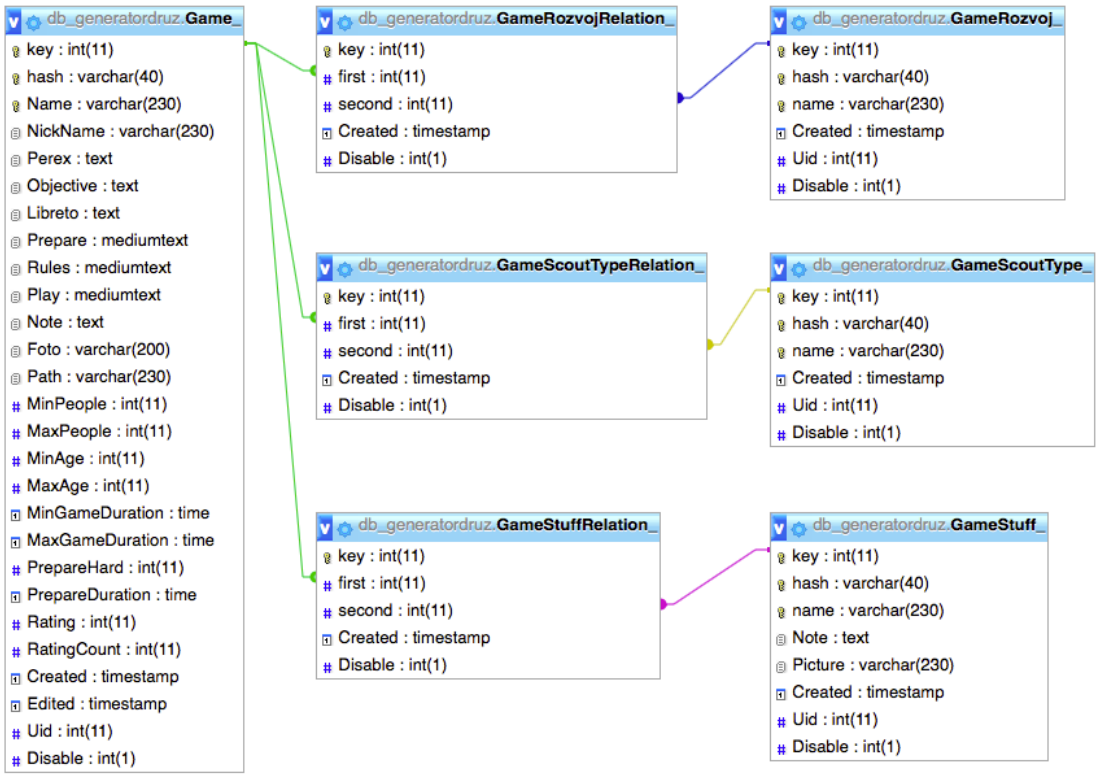
Generátor družinoviek využíva dve základné veci. Efektívny algoritmus generovania družinoviek a robustnú databázu aktivít.

Databáza aktivít a parametrov

Pre každý typ aktivít je v Generátore vytvorená tabuľka, ktorá je prispôsobená na mieru danému typu aktivít. Pre každý typ parametra je tiež vytvorená tabuľka, ktorá obsahuje hodnoty parametra. Ďalej sú vytvorené tabuľky na relácie medzi typmi aktivít a

parametrami. V podstate každá aktivita je spárovaná s 13 typmi parametrov v ktorých sú uložené konkrétne informácie o hre.

Obrázok nižšie ilustruje ako sú prepojené hry s parametrami. Každý z trinástich parametrov má relačnú tabul'ku s hrou. Pre jednoduchosť uvádzam iba tri typy parametrov.



Obr. 3: Náčrt vzťahu hry a typmi parametrov v databáze Generátora družinoviek

Algoritmus generovania skautskej družinovky

Na začiatku si užívatel' vyberie rôzne parametre, ktoré má obsahovať družinovka. Algoritmus si v databáze nájde všetky aktivity, ktoré obsahujú aspoň jeden vybraný parameter. Každý typ parametra má svoje skóre, ktoré sa odvíja od jeho dôležitosti. Napríklad počet l'udí je dôležitejší parameter, ako ročné obdobie. Takto pre každú aktivitu vypočítame skóre, podl'a toho kol'ko obsahuje vybraných parametrov. Tým získame zoznam najlepších aktivít.

Teraz nasleduje kombinácia najlepších aktivít. Tu vstupuje do algoritmu váhová pravdepodobnosť, aby sa zaručilo, že výsledná vygenerovaná družinovka bude vždy trochu odlišná, od inej s rovnakými parametrami. Aktivita s najlepším skóre má najväčšiu

pravdepodobnosť, že sa dostane do výslednej družinovky. Takto dostaneme pravdepodobnosťou vybrané aktivity, ktoré budú v družinovke.

Posledná časť algoritmu usporiada vybrané aktivity do poradia, v akom ich budú robiť na skautskej družinovke. Na to používame šablóny, ktoré určia aký typ aktivity má nasledovať. Napríklad na začiatku družinovky vždy máme máme intro, potom nasleduje zamyslenie, hra, atď. Algoritmus si náhodne vyberie šablónu zo zoznamu šablón a dosadí vybrané aktivity, tak aby nepresiahli časový rámec družinovky.

Obr. 4: Grafické zobrazenie algoritmu generovania skautskej družinovky

Návrh riešenia inteligentného vyhľadávania

Cieľ inteligentného vyhľadávania

Cieľom inteligentného vyhľadávania je analýza vety v prirodzenom spisovnom jazyku, jej spracovanie, ale aj predvídanie, čo chce užívateľ napísať.

Obr. 5: Užívateľské rozhranie Generátora družinoviek pre vyhľadávanie

Vstupné a výstupné dáta

Vstupné dáta bude tvoriť veta v slovenskom jazyku. To znamená, že môže existovať veľa rôznych variant vety s rovnakým významom. Môže byť napísaná s diakritikou alebo aj bez diakritiky a slovosled môže byť tiež poprehadzovaný.

Dôležité je aby vyhľadávací algoritmus dokázal rozpoznať podstatné údaje v obidvoch odlišných vetách, ale s rovnakým významom, ako môžme vidieť na bližšie uvedenom príklade.

vstup:

Družinovka zameraná na telesný rozvoj, ktorá je pre piatich skautov v klubovni a približne na dve hodiny.

- alebo -

Vygeneruj družinovku ktorá je na 2 hodiny v klubovni s piatimi skautmi a telesným rozvojom

výstup:

***Družinovka** zameraná na **telesný** rozvoj, ktorá je pre **piatich skautov** v **klubovni** a približne na **dve hodiny**.*

- alebo -

*Vygeneruj **družinovku** ktorá je na **2 hodiny** v **klubovni** s **piatimi skautmi** a s **telesným** rozvojom.*

Z týchto dvoch viet budeme chcieť získať zvýraznené slová, ktoré sú parametrami družinovky napísané v slovenčine. To bude mať nastarosti vyhľadávací algoritmus. Ostatné slová, ktoré nie sú parametrami, sú zase dôležité pri následnom predpovedaní vety, ktoré bude robiť predpovedací algoritmus

Obr. 6: Zoznam zistených parametrov z vety

Základná myšlienka vyhľadávacieho algoritmu

Program dostane na vstupe vetu, ktorú si rozdelí na jednotlivé slová. Začne prechádzať tieto slová po jednom a bude sa snažiť zistiť ich informačnú hodnotu pre Generátor družinoviek.

1. Skontroluje či slovo patrí do kategórie slov, ktoré neobsahujú v sebe informáciu a nachádzajú sa pred následujúcim parametrom. Budem ich nazývať Prefix. Tieto slová sú dôležité, z hľadiska vetnej syntaxe.

*Vygeneruj **družinovku** ktorá je na **2 hodiny***

Z tejto vstupnej vety medzi Prefixy patria slová *"Vygeneruj"* a *"ktorá je na"*. Ak ich algoritmus nenašiel v databáze, tak ide ďalej na bod dva.

2. Skontroluje či slovo patrí do kategórie slov, ktoré neobsahujú v sebe informáciu a nachádzajú sa za parametrom. Budem ich nazývať Sufix. Tieto slová sú tiež dôležité, z hľadiska vetnej syntaxe.

   > *Vygeneruj* ***družinovku*** *ktorá je na* ***2 hodiny*** *a na* ***citový*** *rozvoj.*

   Z tejto vety medzi Sufixy patrí slovo *"rozvoj"*, ktoré sa viaže na predchádzajúci parameter. Väčšinou Sufixy sú podstatné mená, keď je parameter (***citový***) prídavné meno, ako v tomto prípade. Ak ich algoritmus nenašiel v databáze, tak ide ďalej na bod tri.

3. Skontroluje či je slovo parameter družinovky. Na to používam tri metódy. Od najmenej náročnej až po najnáročnejšiu vzhľadom na čas. Všetky metódy sa snažia prezerať rôzne typy parametrov, ako napríklad čas, miesto, zameranie, atď. Poradie v akom kontroluje parametre, pomáha určovať Markovský rozhodovací proces [Kapitola].

   a. Prvá metóda zisťuje či slovo napr ***družinovku*** sa už nachádza v databáze medzi vyskloňovanými slovami parametrov, ktoré majú odkaz na parameter ***družinovka***.

   b. Druhá metóda zisťuje či slovo je rovnaké ako parameter. V našom prípade nie je. ***družinovku*** != ***družinovka***

   c. Tretia metóda porovnáva rôzne podreťazce zo slova s parametrom pomocou LIKE príkazu SQL. Napríklad zo slova ***družinovku*** vytvorí tieto podreťazce ***družinovku%, družinovk%, družinov%, družino%.*** Podreťazcov je presne jedna tretina dĺžky slova.
      Následne sa vrátené slová porovnávajú zo vstupným slovom pomocou Levenshteinovej vzdialenosti.

Ak našiel v slovách parametre, tak ich zobrazí zvýraznené.
Za každé slovo užívateľ získava skóre (odmeny za akciu v Markovskom rozhodovacom procese), ktorým sa hodnotí veta, ktorú užívateľ napísal.

V skratke za známe slová a dobrú vetnú skladbu, získa užívateľ vyššie skóre ako za nové slová z nezvyčajnou vetnou skladbou. Ak užívateľ napísal dokonca spam, tak dostáva až záporné skóre. Problematikou spamu a zlou vetnou skladbou sa zaoberám v kapitole Problémy inteligentného vyhľadávania.

Základná myšlienka predpovedacieho algoritmu

Program snaží nielen pochopiť, ale aj predpovedať vstup, čo užívateľ chce napísať. Na predpovedanie vstupu budeme používať Markovskú reťaz.

Algoritmus si najprv zistí, aké typy parametrov už užívateľ použil vo vete.

*Vygeneruj **družinovku** ktorá je na **2 hodiny** a na **citový** rozvoj.*

V našom prípade máme 3 typy parametrov:

1. ***družinovku*** to je parameter ***družinovka***, ktorý patrí do *aktivít*
2. ***2 hodiny*** to je parameter ***120 min***, ktorý patrí do *času*
3. ***citový*** to je parameter ***citový***, ktorý patrí do *rozvoja*

Zoberie si posledný typ parametru, takže v našej vete je to *rozvoj* a zistí z Markovského procesu, aký ďalší typ parametru je najpravdepodobnejší. Následne vyberie najpravdepodobnejší parameter. Potom pre najpravdepodobnejší parameter, zistí aký je najpravdepodobnejší prefix a sufix.

*Vygeneruj **družinovku** ktorá je na **2 hodiny** a na **citový** rozvoj.*

*Vygeneruj **družinovku** ktorá je na **2 hodiny** a na **citový** rozvoj a pre **skautov***

*Vygeneruj **družinovku** ktorá je na **2 hodiny** a na **citový** rozvoj v **meste** pre **skautky***

V prvom príklade najpravdepodobnejší typ parametru je *Veková kategória*, z ktorej doplnilo najpravdepodobnejší parameter ***skauti***, z ktorého vybralo najpravdepodobnejšiu verziu skloňovania ***skautov***. Pre verziu skloňovania ***skautov***, zistilo že najpravdepodobnejší prefix je *“a pre”*. Sufix v tomto prípade nie je.

V druhom príklade bol najpravdepodobnejší typ parametru *Miesto, z ktorého* si vybralo parameter ***mesto***, ktorý vyskloňovalo na ***meste***. Najpravdepodobnejší prefix bol *“v”*. Sufix tiež v tomto prípade nie je.

Následne sa posunulo v Markovskom procese na ďalší stav, v ktorom vyšiel najpravdepodobnejší typ parametru *Veková kategória.* Zvyšný postup je rovnaký ako v prvom príklade.

Takýmto spôsobom vieme vygenerovať vetu, ktorý by chcel užívateľ napísať.

Proces učenia

Je dôležité aby vyhl'adávací algoritmus bol rýchly, preto je potrebné do inteligentného vyhl'adávania implementovať učiaci proces

Vo fáze, keď užívatel' dopísal a odoslal rozpoznanú vetu Generátoru družinoviek, sa spúšťa učiaci proces.

Učiaci proces pozostáva zo štyroch fáz.

1. V prvej fáze program zráta skóre, ktoré dosiahol užívatel' písaním vety. To ohodnocoval Markovský rozhodovací proces. Ak je skóre vel'mi nízke, tak znamená, že veta obsahuje vel'a nových slov a divný slovosled. Takáto veta bude mať vel'mi nízku váhu pre učenie. Ak bude skóre dokonca záporné, napríklad v dôsledku spamu, tak takou vetou sa už ďalej nezaoberáme.
Naopak, ak veta bude mať vysoké skóre, bude mať vysokú váhu pre učenie. To znamená, že si silnejšie zapamätá slová, parametre a slovosled vety.

2. V druhej fáze zistíme či veta obsahuje neznáme slová. Ak áno, tak ich rozdelíme na Prefixy a Sufixy. Detekovaniu Sufixov sa venujem v kapitole Problémy inteligentného vyhl'adávania.

3. V tretej fáze ukladáme do databázy nové Prefixy, Sufixy, nové gramatické verzie parametrov. Ak sa už v databáze nachádzajú, tak upravíme ich váhu našim skóre z Markovského rozhodovacieho procesu.

4. V štvrtej fáze sa zameriavame na posilňovanie váh (pravdepodobností) v prechodoch medzi stavmi potrebných pre Markovský rozhodovací proces a Markovský reťazec. Hodnoty zväčšujeme tiež podl'a získaného skóre a poradia parametrov vo vete.

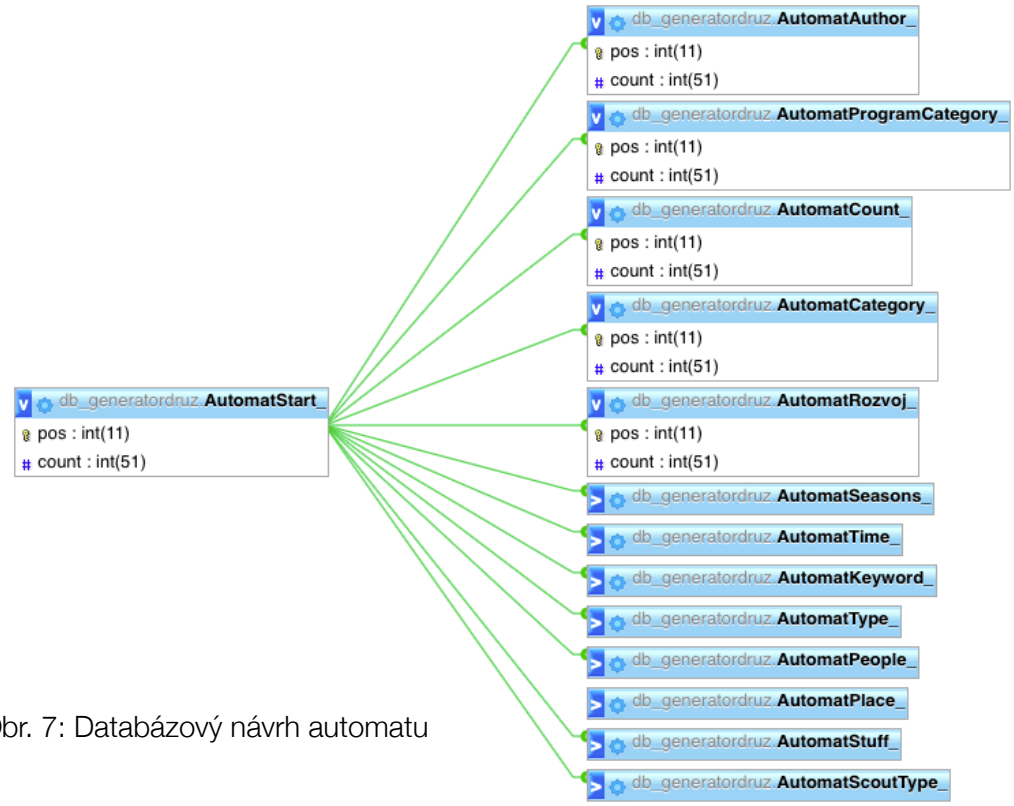
Implementácia inteligentného vyhľadávania

V tejto kapitole opíšem ako sme implementovali inteligentné vyhľadávanie. Kvôli lepšej prehľadnosti na úvod uvediem schému databázy a schému tried, ktoré potom dopodrobna rozoberiem v ďalších častiach tejto kapitoly.

Prehľad databázy inteligentného vyhľadávania

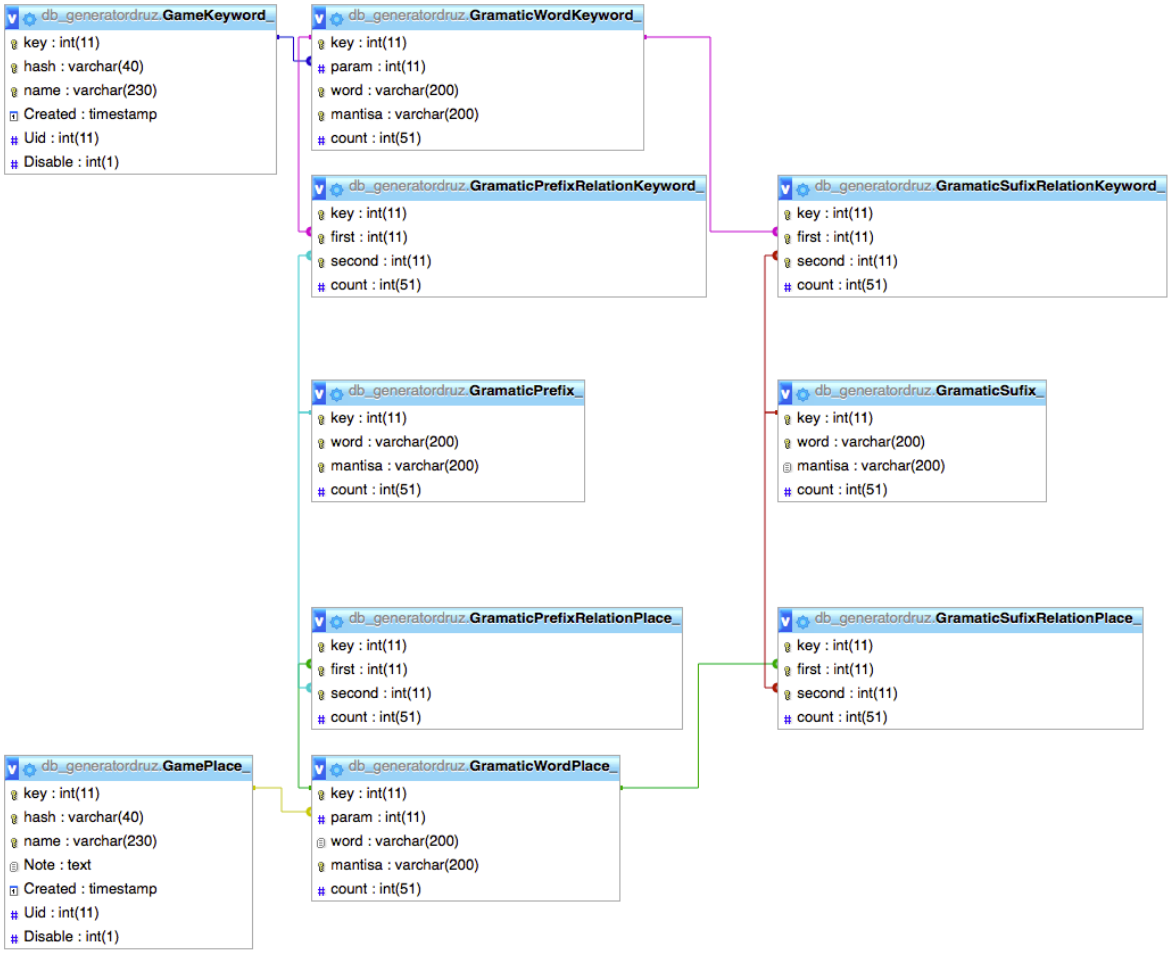
Schému databázy rozdelím na dve časti. V prvej časti budeme opisovať databázové riešenie automatu v Markovových reťazcov a Markovských rozhodovacích procesov. V druhej časti budeme opisovať ako narábame s parametrami a s naučenými slovami.

1. Každý stav v automate má svoju tabuľku. Spolu máme 13 tabuliek, kde každý typ parametra má svoj stav. Potom máme jednu tabuľku na začiatočný stav automatu AutomatStart. Každý stav automatu obsahuje 13 odkazov na ďalšie stavy. Pre jednoduchosť uvádzam iba odkazy zo začiatočného stavu. Podrobne sa tomuto venujem v ďalších častiach tejto kapitoly.



Obr. 7: Databázový návrh automatu

2. Kvôli jednoduchosti rozoberiem následujúcu databázovú štruktúru pre konkrétny typ parametra Keyword. Vo vstupnej vete sme detekovali slovo, ktoré patrí medzi parametre kľúčových slov. Ak to slovo ešte nemáme v databáze tak ho uložíme do tabuľky GramaticWordKeyword spolu s odkazom na originálne slovo z parametrov GameKeyword.
Ak sme pri slove detekovali nejaké predložky (Prefix) alebo príslovky (Sufix), tak ich uložíme do príslušných tabuliek Prefix a Sufix. Následne vytvoríme reláciu v relačnej tabuľke GramaticPrefix(Sufix)RelationKeyword, ktorá spája naše slovo s Prefixom, alebo Sufixom. Takúto štruktúru obsahuje všetkých 13 parametrov. Podrobne sa tomuto venujem v ďalších častiach tejto kapitoly.

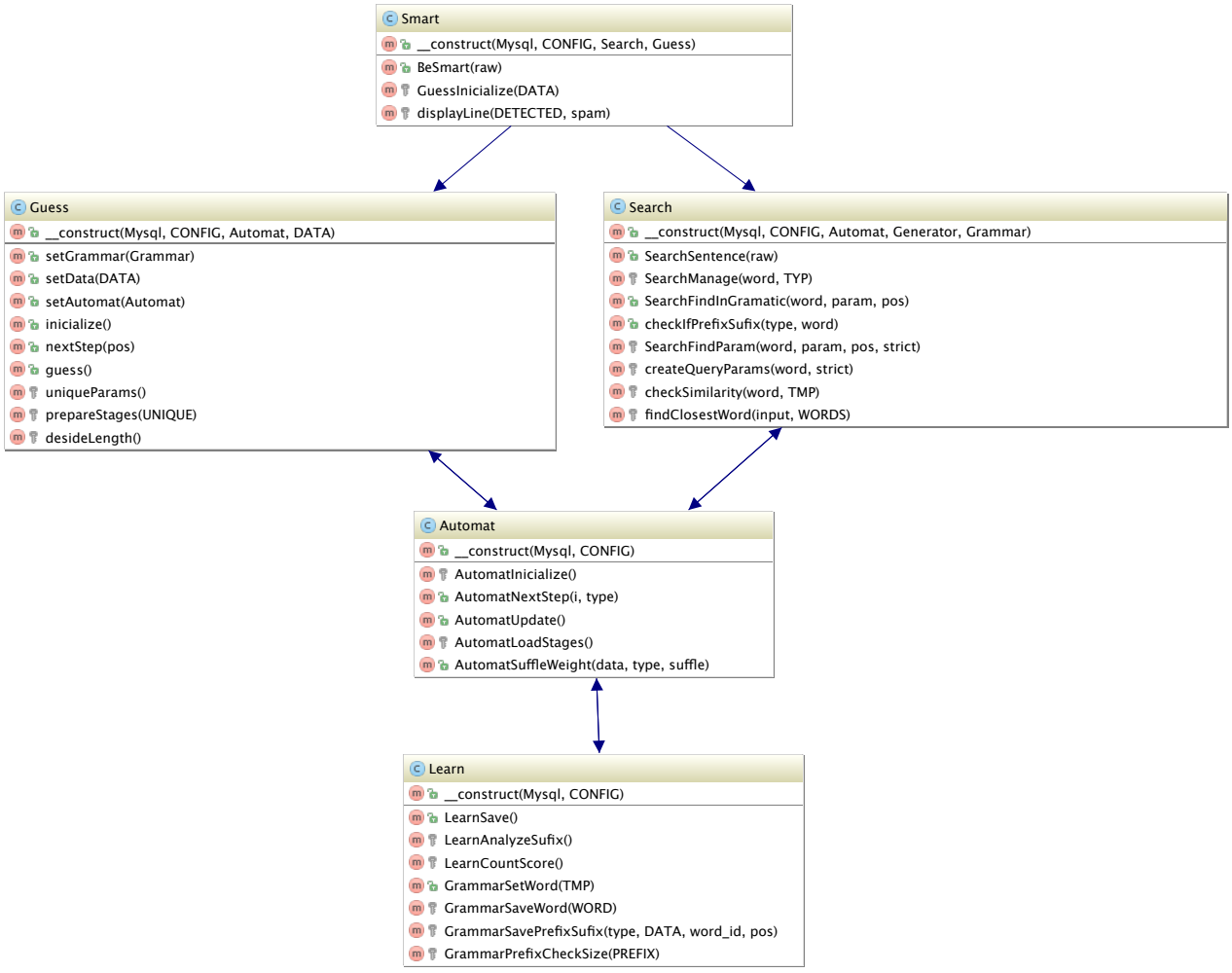


Obr. 8: Databázové riešenie pre typy parametrov Keyword a Place

Prehľad tried inteligentného vyhľadávania

V následujúcej schéme tried máme jednu hlavnú triedu Smart.php, ktorá zastrešuje inteligentné vyhľadávanie Search.php a inteligentné predpovedanie vstupnej vety Guess.php. Obidve tieto triedy komunikujú s triedou Automat.php, ktorá má nastarosti riadenie stavov v Markovských procesoch. Posledná trieda je Learn.php, ktorá

obsluhuje ukladanie analyzovaných slov zo vstupnej vety. Pri ukladaní tiež využíva triedu Automat.php.



Obr. 9: Prehl'ad tried Inteligentnom vyhl'adávaní.

Návrh automatu na Markov rozhodovací proces a Markovove reťazce

V Markovskom rozhodovacom procese a Markovových reťazcoch je chovanie bezpamäťové, to znamená, že si nepotrebujeme pamätať minulosť, ale stačí vedieť iba aktuálny stav. Takýto systém vieme vyjadriť konečným automatom.

Automat pozostáva zo 13 + 1 stavov, ktoré charakterizujú typy parametrov, plus začiatočný stav.

1. S_0 stav je **začiatočný**, ktorý určuje pravdepodobnosti akého typu bude prvý parameter vo vete.

Nasledujúcich 13 stavov sú tieto parametre

1. S_1 stav je **typ aktivity**
2. S_2 stav je **rozvoj**
3. S_3 stav je **zameranie**
4. S_4 stav je **typ hry**
5. S_5 stav je **veková kategória**
6. S_6 stav je **miesto**
7. S_7 stav je **obdobie**
8. S_8 stav je **čas**
9. S_9 stav je **počet l'udí**
10. S_{10} stav je **autor**
11. S_{11} stav je **veci**
12. S_{12} stav je **postavy**
13. S_{13} stav je **kl'účové slová**

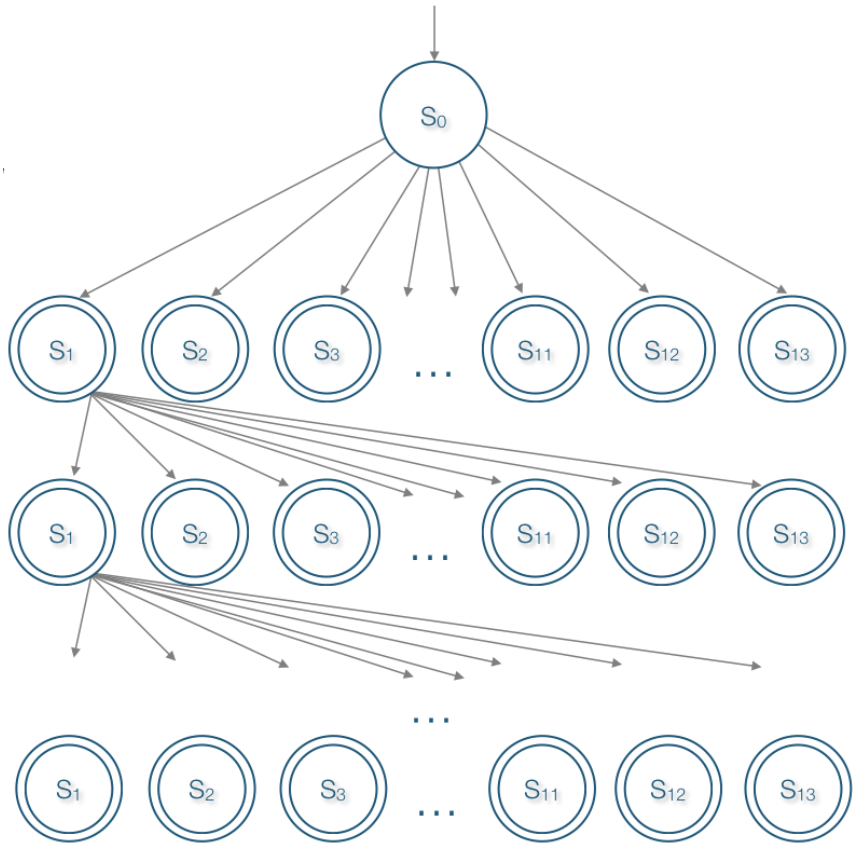
Zo stavu S_0 sú prechodové funkcie do všetkých ostatných stavov S_1 až S_{13} . Zo stavov S_1 až S_{13} sú prechodové funkcie do stavov S_1 až S_{13}, čiže obsahujú aj prechodovú funkciu na seba. Lebo užívatel' môže zadať vo vete aj dva rovnaké parametre za sebou napríklad **skauti** a **skautky**. Obidva tieto parametre patria do rovnakého typu - veková kategória, pre ktorú prislúcha stav S_5.

Akceptačné (konečné) stavy sú vo všetkých stavov S_1 až S_{13}. Nakoľko vo vete môže byť l'ubovolný parameter posledný.

*Vygeneruj **družinovku** ktorá je na **2 hodiny** a na **citový** rozvoj.*

Napríklad poradie detekovaných parametrov v tejto vete je **typ aktivity** (družinovku), **čas** (2 hodiny) a **rozvoj** (citový). Takže v automate sme prešli cez stavy S_0, S_1, S_8 a skončili sme v akceptačnom stave S_2.

Automat môžme znázorniť následujúcim stavovým diagramom. Pre lepšiu prehl'adnosť som znázornil iba prechody zo stavu S_0 a zo stavu S_1. Vo zvyšných stavoch idú prechodové funkcie tiež do všetkých stavov ako v S_1.

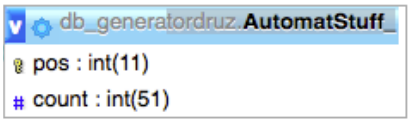


Obr. 10: Stavový diagram automatu

V diagrame nie sú znázornené pravdepodobnosti prechodu a tiež nie sú znázornené odmeny za akcie pre MDP, nakoľko budú závisieť až používateľských dát, ktoré sa menia každým generovaním družinovky.

Databázové riešenie automatu

Jednotlivé stavy automatu budem reprezentovať 13 + 1 tabuľkami. Pre jednoduchosť, uvediem príklad iba jednej tabuľky, nakoľko ostatné majú rovnakú štruktúru. Prechody medzi stavmi budú jednotlivé záznamy v tabuľke. Stĺpce v tabuľke sú `pos` - id stavu automatu do ktorého ide prechod a `count` - hodnota, ktorá charakterizuje silu prechodu do daného stavu. Primárny kľúč je `pos`. Stĺpec `count` má aj index, aby sme vedeli rýchlo hľadať hodnoty.



Trieda Automat.php

Táto trieda má na starosti obsluhu automatu pre Markovský reťazec a Markovský rozhodovací proces. Rozoberiem najdôležitejšie metódy tejto triedy. Metódy budú skrátené. Zachovám iba najdôležitejšie časti kódu.

AutomatInicialize

Na začiatku je dôležité inicializovať automat na začiatočný stav S_0, ktorý uložíme do histórie. Následne načítame všetky možné prechodové funkcie zo stavu S_0 metódou AutomatLoadStages()

```
protected function AutomatInicialize(){
        $this->pos=0;
        $this->HISTORY[] = $this->pos;
        $this->AutomatLoadStages();
}
```

AutomatLoadStages

Metóda `AutomatLoadStages()` získa všetky prechodové funkcie od triedy MySQL, ktorá má na starosti komunikáciu z databázou. Následne získané prechodové funkcie zoradíme pomocou metódy `AutomatSuffleWeight()`.

Táto metóda zoraďuje prvky pola podľa ich váhy. Čím majú prvky vyššiu váhu, tým je väčšia pravdepodobnosť, že sa ocitnú na prvých miestach.

Zatriedené prechodové funkcie vložíme do internej premennej, `AUTOMAT`.

```
protected function AutomatLoadStages(){
        $TMP=$this->Mysql->SelectAll("","",$table_,"ORDER BY `count` DESC ,`pos` ASC");
        $TMP=$this->AutomatSuffleWeight($TMP,"many", 1);
        $this->AUTOMAT[$this->pos]=$TMP;
}
```

AutomatNextStep

Táto metóda je len upravená inicializačná metóda `AutomatInicialize()`. Jej úlohou je presunúť sa do ďalšieho stavu Automatu a získať jeho prechodové funkcie.

```
public function AutomatNextStep($i){
        $this->pos = $i;
        $this->HISTORY[] = $i;
        $this->AutomatLoadStages();
}
```

AutomatUpdate

Hlavná funkcia tejto metódy je posilňovanie váh pre jednotlivé prechodové funkcie. Metóda prechádza históriu prechodov medzi stavmi v automate a zvyšuje ich váhu.

```
public function AutomatUpdate($weight){
        for($i=0; $i<(count($this->HISTORY)-1); $i++){
        $table_ = $this->CONFIG["mysql"]["smart"]["automat"][$this->HISTORY[$i]];
        $this->Mysql->Exec("UPDATE ".$table_." t SET count = count +".$weight." WHERE
         t.pos = '".$this->HISTORY[($i+1)]."'");

        }
}
```

Databázové riešenie tvary parametrov, prefixy a sufixy

Inteligentné vyhľadávanie sa snaží rozpoznávať gramatické tvary parametrov a následne ich ukladať do databázy. Napríklad pre parameter **skauti**, budem mať tieto tvary: **skautom**, **skautov**, atď. Každý typ parametra má svoju tabuľku. Preto budem potrebovať 13 tabuliek.

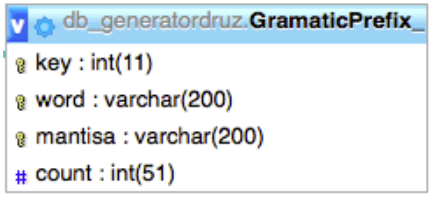
Tabuľka pre tvary parametrov

V tabuľke si pamätám jednotlivé hodnoty: `id` je jedinečný kľúč daného slova. `param_id` je odkaz na id parametra, ktorému toto slovo patrí. Tvar slova aj s gramatikou je uložený v hodnote `word`. Slovo bez diakritiky a bez veľkých písmen je uložené v `core`, aby sme zabránili duplicitným slovám s rôznou diakritikou. Preto je aj unique. Nakoniec máme hodnotu `count`, ktorá si pamätá váhu.

Tabuľka pre Prefixy a Sufixy

Pre rýchle rozpoznanie slov vo vete, ale predpovedanie vety, si musíme pamätať slová a znaky, ktoré nie sú parametrami. Rozdelené sú do dvoch kategórii: slová, ktoré sú pred parametrom - prefixy a slová za parametrom - sufixy.

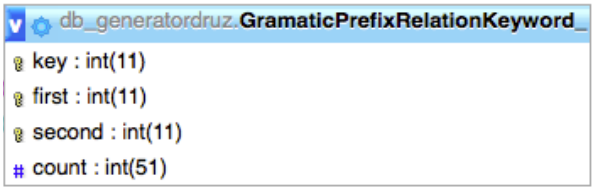
Tabuľka pre prefix a sufix je rovnaká. Pamätám si v nej následujúce hodnoty: `id` je jedinečný kľúč prefixu, `word` je prefix s diakritikou, slovo bez diakritiky a bez veľkých písmen je uložené v `core`, aby sme zabránili duplicitným slovám s rôznou diakritikou. Nakoniec máme `count`, ktorý si pamätá koľko krát sa slovo použilo.



Tabuľka relácie medzi tvarmi parametrov a Prefixami alebo Sufixami

Táto jednoduchá tabuľka priraďuje prefixy alebo sufixy, jednotlivým vyskloňovaným tvarom parametrov. Na každý typ parametru potrebujeme relačnú tabuľku pre prefix a sufix, preto budeme potrebovať 2 * 13 tabuliek. Uvediem príklad relačnej tabuľky pre prefix. Sufixová relačná tabuľka sa líši iba v tom, že v hodnote `first` je odkaz na sufixovú hodnotu.

V tabuľke si pamätáme `id`, čo je jedinečný kľúč, potom odkaz na prefix cez jeho id `first` a odkaz na id gramatického tvaru slova `second`. Posledná hodnota je váha `count`. Ešte máme uložený aj jedinečný zložený kľúč z `first` a `second` aby sme zabránili duplicitným hodnotám.



Trieda Search.php

Trieda Search má na starosti identifikáciu slov vo vete. Nakoľko táto trieda je rozsiahla, uvediem iba niekoľko najdôležitejších metód. Metódy budú skrátené o pomocné funkcie, premenné a o časti ktoré komunikujú s frameworkom Generátora družinoviek. Zachovám iba najdôležitejšie časti kódu, ktoré nesú hlavnú myšlienku.

SearchSentence

Táto public metóda dostane na vstup vetu od užívateľa, ktorá je očistená od nežiadúcich znakov a duplicitných medzier.

Rozdelíme si vetu na jednotlivé slová a v časti switch zisťujeme informácie o slove. Ak slovo sa rovná niektorej podmienke, tak už nekontrolujeme ďalej a rovno ho uložíme do dočasnej premennej $TMP. Ako prvé sa pozeráme či je slovo čiarka. Ak nie tak zisťujeme či je to sufix, alebo prefix. Inak kontrolujeme či je to parameter. Prvá metóda zisťovania parametru je skombinovaná z dvoch podmetód “gramatic” a “strict". Metóda “gramatic” kontroluje či slovo je vyskloňovaný tvar parametera, ktorý už pozná. Druhá metóda “strict" zisťuje, či je slovo presne rovnaké, ako parameter.

Ak ani jedna z týchto dvoch podmetód neurčila parameter, tak prichádza posledná metóda "all", ktorá je najnáročnejšia na výpočet. Pretože zisťuje, či sa rôzne podreťazce daného slova podobajú na parameter. Ak sme našli parameter, tak sa posunieme do ďalšieho stavu v automate. Keď prejdeme všetky slová, tak vrátime pole všetkých slov, ktoré prešli analýzou.

```
public function SearchSentence($raw)
{
        $this->INPUT = explode(" ",$raw);

        foreach($this->INPUT as $word){
                switch(true)
                {
                        case ($word == ","):
                                $TMP = array("word"=>$word, "comma"=>"1");
                        break;

                        case ($this->checkIfPrefixSufix("sufix",$word)):
                                $TMP= array("word"=>$word, "sufix"=>"1");
                        break;

                        case ($this->checkIfPrefixSufix("prefix",$word)):
                                $TMP= array("word"=>$word, "prefix"=>"1");
                        break;

                        default:
                          $TMP = $this->SearchManage($word, array("gramatic","strict"));

                          if((!isset($TMP["name"]))&&(strlen($word)>4)){
                                $TMP = $this->SearchManage($word, array("all"));
                          }
                        break;
                }
        $this->nextStep($TMP);
        }
return $this->Grammar->GrammarGetDetected();
}
```

checkIfPrefixSufix

Metóda, ktorá zisťuje, či slovo je prefix, alebo sufix. Na’vstup dostane hodnotu $type, ktorá určuje čo ideme zisťovať a ešte slovo $word, ktoré ideme kontrolovať. Pozrie sa do databázy či sa tam slovo nachádza a podľa toho vráti true alebo false.

```
protected function checkIfPrefixSufix($type,$word){
      $query="SELECT t.word FROM ”.ucfirst($type)." t WHERE LOWER( CONVERT(t.word
       USING utf8) ) LIKE '".$word."'";

      $OUT = $this->Mysql->Fetch($query);


      if(count($OUT)>0){ return 1;}
      else{ return 0;}
 }
```

SearchManage

Je dôležitá metóda, ktorá obrazne povedaná menežuje hľadanie parametrov v slovách. Na vstup dostane slovo, ktoré ide analyzovať $word a pole $TYP, ktoré určuje v akom poradí má používať metódy na detekovanie parametru.

Metóda prechádza cez jednotlivé stavy automatu $this->STAGES, ktoré získala od triedy Automat.php Prechádza ich vzostupne od najpravdepodobnejších až po najmenej pravdepodobné. Zistí si, aký typ parametru prislúcha danému stavu automatu. Následne používa hľadacie metódy v poradí, akom ich dostal na vstupe v poly.

Z hľadania dostane výstup do pola $TMP, ktorý potom skontroluje, či je podobný zo vstupným slovom. Ak hľadacia funkcia našla viac podobných slov, tak vyberie to najbližšie pomocou váhovej Levenshteinovej vzdialenosti. Ak nenašlo podobné slovo, tak pokračuje v hľadaní ďalšou metódou. Nové slovo vráti vtedy, keď vyskúšal všetky hľadacie metódy a prešiel všetkými stavmi v automate. Nakoniec výstup vrátime v poly $OUT.

```
protected function SearchManage($word, $TYP){
     foreach($this->STAGES as $stage){
            $param_type  = $this->Automat->AutomatGetParamType($stage);
            for($i=0; $i<count($TYP); $i++)
               {
                   switch($TYP[$i]){
                      case "gramatic":
                         $TMP = $this->SearchFindInGramatic($word, $param_type);
                      break;

                      case "strict":
                         $TMP = $this->SearchFindParam($word, $param_type, "strict");
                      break;

                      case "all":
                         $TMP = $this->SearchFindParam($word, $param_type, "all");
                      break;
                   }
                /* ak nic nenaslo */
                if(count($TMP)==0){continue;}
                if(count($TMP)==1){
                   $PREPARED = $this->checkSimilarity($word,$TMP);
                }
                if(count($TMP)>1){
                   $PREPARED = $this->findClosestWord($word, $TMP);
```

```
                $PREPARED = $this->checkSimilarity($word,$PREPARED);
            }
                /* ak sa slová nepodobali */
            if(count($PREPARED)==0){continue;}
            $OUT=$PREPARED;
            break;
        }
        /* ak naslo, tak vyskocime z cyklu */
        if(count($OUT)>0){break;}
    }
$OUT["word"] = $word;
return $OUT;
}
```

SearchFindInGramatic

Hlavný účel tejto metódy je zistiť, či sa slovo nachádza už medzi naučenými slovnými tvarmi parametrov. Na vstupe dostane slovo $word, ktoré ide zisťovať a premennú $param_type, ktorá určuje, že v akom type parametrov má hľadať. Skontroluje, či slovo je v databáze a vráti výstup, ktorý našlo.

```
protected function findInGramatic($word, $param_type){
        $query="SELECT t.param, t.word FROM ".$param_type." t WHERE LOWER( CONVERT(
        t.word USING utf8) ) LIKE '".$word."'";
      $OUT = $this->Mysql->Fetch($query);
      return $OUT;
}
```

SearchFindParam

Metóda dostáva na vstupe slovo $word, ktoré ide pozerať a premennú $param_type, ktorá určuje, v akom type parametrov má hľadať. Nakoniec dostane v premennej $metod aký spôsob hľadania má použiť.

Spôsoby hľadania sa líšia len databázovým dotazom. Spôsob "all" prehľadáva podreťazce vstupného slova. Uberá počet znakov jednej tretiny dĺžky slova. Napríklad pre slovo **skautov** pozerá v databáze parametrov tieto podreťazce **skautov**%, **skauto**%, **skaut**%.

Avšak spôsob hľadania "strict" pozerá v databáze parametrov iba vstupné slovo "**skautov**".

Databázový dotaz nám vytvorí pomocná funkcia `createQueryParams()`. Následne vygenerovaný dotaz dáme spracovať databáze a ona nám vráti parametre.

```
protected function findParam($word,$param_type,$metod){
        $query = $this->createQueryParams($word,param_type,$metoda);
        $OUT = $this->Mysql->Fetch($query);
      return $OUT;
}
```

checkSimilarity

Táto jednoduchá metóda dostane na vstup dva reťazce slov. Slová porovnáva pomocou internej php funkcie `similar_text()`. Ak sú slová podobné menej ako 70 percent, vráti prázdne pole, inak vráti pole so slovom, ktoré sme porovnávali.

findClosestWord

Metóda vyberie z pola slov $TMP, také, ktoré sa najviac podobajú vstupnému slovu $word. Pole slov porovnáva so slovom $word pomocou váhovej Levenshteinovej vzdialenosti. Váhy sú nastavené takto. Ak sú slová podobné na menej ako 70 percent, tak vráti prázdne pole, inak vráti pole so slovom

Trieda Learn.php

V trieda Learn má nastarosti proces učenia. Na začiatku dostane trieda od serveru pomocou ajaxu vetu, ktorá obsahuje už detekované parametre, prefixy, sufixy a nové slová.

Najprv spočíta skóre vety, ktoré dosiahla pomocou Markovského ohodnocovacieho procesu. Toto skóre nám hovorí, ako veľmi sa veta líši od najpravdepodobnejšej vstupnej vety. Ak je skóre malé, tak generátor sa ju naučí s menšou váhou, ako keď je skóre veľké.

Uloží postupnosť parametrov vo vete do automatu, kde upraví váhu daného stavu podľa skóre. Ak veta obsahuje nové tvary parametrov, tak ich uloží, ako nové slová do databázy. Ak ich už pozná, tak iba zaktualizuje počet, koľko krát bol tvar použitý. Tento postup platí aj pre prefixy a sufixy.

Metódy budú skrátené o pomocné funkcie, premenné a o časti ktoré komunikujú s frameworkom Generátora družinoviek. Zachovám iba najdôležitejšie časti kódu, ktoré nesú hlavnú myšlienku.

LearnCountScore

Metóda dostane pole stavov automatov v poradí, ako boli napísané parametre vo vete. Následne začne prechádzať stavmi a spočíta skóre. Potom skóre porovná so skóre najpravdepodobnejšej vety a vráti výsledok v percentách.

LearnSave

Metóda, ktorá riadi ukladanie použitých slov vo vete do databázy. Na vstupe má pole $this->DETECTED, ktoré obsahuje všetky informácie o vete. Pred ukladaním každého slova, ešte zistí či veta obsahuje nové sufixy. Potom začne prechádzať postupne jednotlivými slovami a ukladá ich do databázy kde patria.

Ako prvé ukladá parametre aj s prefixom. Pri prefixe ešte kontroluje ich dĺžku, či ich tam nie je viac ako tri. Potom uloží prefix. Druhé ukladá sufixy, ktoré majú odkaz na id tvaru parametra, ku ktorému patria. Posledné dva prípady sú iba pridávanie známych a nových slov do pola prefixov, ktoré potom uložíme v prvom bode.

```
public function LearnSave()
{
    $this->LearnAnalyzeSufix();
    foreach($this->DETECTED as $key => $WORD)
    {
       switch(true){
          case (isset($WORD["param"])):
             $id = $this->GrammarSaveWord($WORD);
            $PREFIX = $this->GrammarPrefixCheckSize($PREFIX);
             if(count($PREFIX)>0){
               $this->GrammarSavePrefixSufix("prefix",$PREFIX, $id, $WORD["pos"]);
             }
             break;
             case (isset($WORD["sufix"])):
               $this->GrammarSavePrefixSufix("sufix",  $WORD["word"],  $this->
DETECTED[($key-1)][“gramatic_id"], $this->DETECTED[($key-1)][“pos"]);

             break;
             case (isset($WORD["prefix"])):
               $PREFIX[]=$WORD["word"];
             break;
              default:
                $PREFIX[]=$WORD["word"];
             break;
        }
    }
}
```

LearnAnalyzeSufix

Táto metóda hľadá sufix vo vete. Podrobne sa tejto funkcii venujem v kapitole Problémy inteligentného vyhľadávania.

GrammarSaveWord a GrammarSavePrefixSufix

Sú dve metódy, ktoré majú nastarosti ukladanie slov, prefixov a sufixov do databázy. Ešte aj pripočítavajú hodnotu, ktorú dosiahla veta v Markovskom ohodnocovacom procese.

Trieda Guess.php

V triede Guess riešime predpovedanie vety, ktorú užívatel' chce napísať. Na predpovedanie využívame Markovské reťazce, ktorými zisťuje najpravdepodobnejšiu vstupnú vetu.

Metódy budú skrátené o pomocné funkcie, premenné a o časti ktoré komunikujú s frameworkom Generátora družinoviek. Zachovám iba najdôležitejšie časti kódu, ktoré nesú hlavnú myšlienku.

Inicialize

Metóda inicializuje premenné, ktoré sú potrebné na rozbehnutie predpovedania vstupnej vety. Pole $this->MISSING obsahuje zoznam jednotlivých typov parametrov, ktoré užívatel' ešte nezadal. V premmenej $this->best je uožená informácia o najpravdepodobnejšiom následujúcom stave, ktorú sme získali pomocou Markovských reťazcov.

```
function inicialize(){
      $this->MISSING  = $this->uniqueParams();
      $this->best     = $this->Automat->AutomatSuffleWeight($this->MISSING, "one");
}
```

GuessSentence

Je hlavná metóda, ktorá riadi predpovedanie vety. Predpovedá iba tú časť, ktorú užívatel' ešte nenapísal. Z metódy `GrammarGetWordComplete()` dostane slovo aj s prefixom a sufixom, ktoré vyšlo najpravdepodobnejšie v danom stave $this->best["pos"]. Ak metóda vrátila prázdne pole, tak nenašla vhodné slovo pre danný stav. Následne, zisťujeme či slovo obsahuje prefix, ak hej tak hprefix uložíme do metódou `GuessSetWord()`. Potom uložíme samotné slovo a za ním uložíme sufix, ak ho obsahuje. A posunieme sa do ďalšieho pravdepodobného stavu. Po skončení while cyklu vrátime predpovedanú vetu.

```
public function GuessSentence(){
        $i=0;
        while($i<count($this->MISSING)){
```

```
            $ARRAY = $this->GuessGetWordComplete($this->best["pos"]);

            if(count($ARRAY)==0){ return array();}
            if(count($ARRAY["prefix"])>0){
                $ARRAY["prefix"]["prefix"]=1;
                $this->GuessSetWord($ARRAY["prefix"]);
            }

            $this->GuessSetWord($ARRAY["word"]);

           if(count($ARRAY["sufix"])>0){
                $ARRAY["sufix"]["sufix"]=1;
                $this->GuessSetWord($ARRAY["sufix"]);
            }

            $this->nextStep($this->best["pos"]);
            $i++;
        }
        return $this->GuessGetDetected();
    }
```

Problémy inteligentného vyhľadávania

Rozpoznávanie nových sufixov

V generátore družinoviek máme skoro všetky parametre podstatné mená. Až na jednu výnimku - Rozvoj. Všetky hodnoty Rozvoju sú prídavné mená a to je pre nás problém. Pretože prídavné mená sa v slovenčine väčšinou spájajú s podstatnými menami a tie sú vo vete až za nimi, čiže to podstatné meno sa dostane do prefixu následujúceho parametra.

Zoberme si napríklad túto vetu..

Družinovka *zameraná na* ***telesný*** *rozvoj a pre* ***skautov***

Ideme ju analyzovať, ale bez detekcie sufixov. Táto veta obsahuje tri parametre. Prefix pre parameter ***telesný*** je "*zameraná na*", čo je v poriadku, ale pre parameter ***skautov*** je prefix "*rozvoj a pre*". To je už zle lebo slovo "*rozvoj*" sa viaže s predošlým parametrom a slová "*a pre*" sa viažu s nasledujúcim parametrom.

Riešenie

Na začiatok som si napísal veľa viet a snažil som sa ich zanalyzovať. Označil som si jednotlivé slová písmenami podľa toho čo sú, takže parameter - R, prefix - P, sufix - S a nové slovo - N. V našej vete budem považovať slovo "*rozvoj*", ako nové slovo, teda že ho nemá ešte v databáze. Veta po označení by vyzerala takto RPRNPR.

Všimol som si, že postupnosť znakov RNP alebo RN. (na konci vety) okolo nového sufixu bola rovnaká vo viacerých vetách. Vždy pred sufixom bol parameter, potom nasledoval samotný sufix a niektoré slová z prefixu. Toto fungovalo za predpokladu, že už dáme v databáze uložených asi päťdesiat najčastejších prefixov, lebo inak by sme nevedeli určiť prefix v postupnosti a tým pádom by sme dostali postupnosť RNNR, čo sú iba nové slová medzi parametrami.

Nakoniec som spravil metódu `LearnAnalyzeSufix()`, ktorá si označí slová a potom hľadá výskyt postupnosti, ktorý charakterizuje sufix.

Nezmyselné slová na vstupe - spam

Neviem vylúčiť možnosť, že užívatel' začne písať nezmyselné slová a znaky do vety. Tieto slová a znaky nám zbytočne zaťažujú server, nakol'ko nie sú uložené v databáze a vyhl'adávací algoritmus používa na takéto slová časovo náročnejšie metódy.

Preto je ideálne spam už filtrovať v prehliadači, aby nám nezaťažoval server. Na spamovú ochranu som spravil funkciu v javascripte, ktorá sa snaží zachytiť slová, ktoré sa nepodobajú na slovenčinu.

Riešenie

Funkcia sa skladá z viacerých častí. Kontrola vety a kontrola slov. V kontrole slov zisťujem pomer dĺžky slova ku jedinečným znakom vo slove. Ak je pomer menší ako jedna tretina, tak slovo považujem za spam.

Vo vete porovnávam počet znakov ku medzerám. Ak je pomer blízky jednotke, tak je to spam. Ak je veta dlhšia ako 26 znakov, tak spravím frekvenčnú analýzu znakov a porovnám ju so slovenčinou. Tiež ak to bude malá zhoda, tak veta bude obsahovať spam.

Táto kontrola sa spúšťa zakaždým, keď užívatel' dopísal slovo. Lebo pri písaní slova sa stávalo, že nedokončené slovo bolo detekované ako spam. Takýmto opatrením sa mi podarilo odstrániť ten hrubý spam, avšak ak niekto napodobňuje slovenčinu, tak je to viac menej nemožné. Ešte jedna spamová ochrana je taká, že inteligentné vyhl'adávanie je prístupné, až po prihlásení do Generátora družinoviek, takže vieme zistiť, kto písal spam.znamená

Analýza efektivity vyhľadávania

Testovanie presnosti inteligentného vyhľadávania

Snažili sme sa zistiť, či vyhľadávací algoritmus správne zanalyzuje vetu. To znamená, že detekuje všetky parametre, prefixy, sufixy a nové slová.

Na testovanie presnosti inteligentného vyhľadávanie sme použili 30 rôznych viet. Každá veta obsahovala minimálne štyri parametre a mala priemernú dĺžku 10 slov. Parametre boli rôzne vyskloňované, tak aby ich nebolo jednoduché detekovať.

Výsledky testu dopadli nad naše očakávania. Inteligentnému vyhľadávaniu sa podarilo detekovať všetkých 156 parametrov v 30 vetách. Najdlhšie trvalo detekovanie nových tvarov parametrov, ktoré ešte inteligentné vyhľadávanie nepoznalo. Pri detekcii prefixov, sufixov a nových slov sme nezaznamenali žiadny problém. Všetky spoľahlivo detekovalo.

Výhody pamätania si prefixov a sufixov

Na jednoduchom prípade illustrujem, ako nám pomáha, keď si v databáze pamätáme použité prefixy a sufixy. Zoberme si napríklad túto vetu.

Družinovka *zameraná na* ***telesný*** *rozvoj a pre* ***skautov***

V tejto vete chceme detektovať slová, ktoré nie sú parametre. Na to aby sme to zistili, musíme sa pozrieť do tabuľky prefixov, alebo sufixov či sa tam slovo nachádza. Ak ich v tabuľke nenájdeme tak potom skontrolujeme či náhodou slovo nepatrí do niektorej z 13 typov parametrov.

Bez pamätania si prefixov a sufixov v databáze:

Keď si v databáze nepamätáme prefix, alebo sufix, tak nám stačí iba zisťovať, či slová patria do jednej z 13 typov parametrov. Všetko ostatné budú napríklad prefixy. To znamená, že pri každom slove sa najviac 13 krát pozrieme do databázy aby sme to zistili. Pre vetu dĺžky osem slov to robí až 104 pozretí do databázy!

S pamätaním si prefixov a sufixov v databáze:

Keďže už máme databáze zapamätaných niekoľko prefixov a sufixov, tak nám stačí si každé slovo skontrolovať maximálne dvakrát pozretím do tabuľky prefixov a sufixov. Takto si vo vete dĺžky osem slov oddelíme parametre od prefixov a sufixov iba na 16 dotazov do databázy.

Výhody použitia Markovových reťazcov

Na vysvetlenie použijem tiež predchádzajúcu vetu s tým, že už vieme, ktoré slová sú prefixy a sufixy.

Bez Markovových reťazcov:

Pri každom slove ktoré je parameter, sa vo vete musíme pozrieť najviac 13 krát do databázy aby sme prešli všetky typy parametrov. Takto zistíme aký parameter reprezentuje dané slovo. Pre vetu, ktorá obsahuje tri parametre sa pozrie do tabuliek parametrov priemerne až 33 krát.

S Markovovými reťazcami:

Markovove reťazce nám určujú pravdepodobnosť následujúceho typu parametra. Takto vieme najprv pozrieť v databáze tie najpravdepodobnejšie a potom tie menej pravdepodobné typy parametrov. Napríklad v databáze máme niektoré pravdepodobnosti aj na 98 percent. Takýmto postupom sa nám zmenší priemerný počet pozretí do databázy na 5 krát, pre tri parametre.

Výhody učenia sa nových gramatických tvarov parametrov

Budem porovnávať efektivitu inteligentného vyhľadávania s učením sa a bez učenia sa nových tvarov parametrov. Bez učenia nemôžeme využívať hľadacia metódu “gramatic”, ale iba “strict” a “all”.

Bez učenia:

Keby sme nemali uložené gramatické tvary parametrov, tak by sme museli kontrolovať, či má slovo presne tvar parametru. Alebo by sme zisťovali či rôzne podreťazce slova sa aspoň z časti podobajú na daný parameter. A to je náročné nakoľko ešte by sme museli počítať podobnosť slova s parametrom, ale bo vyberať z množiny parametrov, ktoré sa najviac podobajú vstupnému slovu.

S učením:

Ako som už spomenul s učením môžme využívať hľadaciu metódu “gramatic”, ktorá kontroluje či slovo je rovnaké ako nejaký gramatický tvar parametra, ktorý už má v databáze. A to je veľmi rýchle, lebo už nemusí robiť ďalšie kontroly slova, ktoré sú napríklad potrebné pri metóde “all”. Samozrejme, ak slovo nenájde metódou “gramatic”, tak musí použiť ostatné dve metódy, ktoré nevyužívajú učenie.

Záver

Inteligentné vyhl'adávanie pre Generátor družinoviek sa mi podarilo úspešne dokončiť, natrénovať a nasadiť do prevádzky. Každým používaním sa inteligentné vyhl'adávanie zlepšuje a spresňuje predpovedanie vstupnej vety.

Vzhl'adom na krátku dobu v ostrej prevádzke, nemáme dostatočné údaje o spokojnosti užívatel'ov s inteligentným vyhl'adávaním. Avšak užívatelia, ktorí testovali beta verziou, boli nadmieru spokojní s presnosťou inteligentného vyhl'adávania.

Užívatel'om je prístupné na stránke Generátora družinoviek po prihlásení. Väčšina užívatel'ov uvítala inteligentné vyhl'adávanie, pretože im ul'ahčilo a spresnilo generovanie skautských družinoviek.

Zdroje

Literatúra

(1) Nicholas C. Zakas, Jeremy McPeak, Joe Fawcett: **Ajax Profesionálně** Zoner Press, Brno, 2007

(2) Timothy Boronczyk, Elizabeth Naramore, Jason Gerner, Yann Le Scouarnec, Jeremy Stolz: **PHP 6, MySQL, Apache** Computer Press, Praha, 2009

(3) Jíří Demel: **Operační výskum**, 2006-2011 [strana 123]
http://kix.fsv.cvut.cz/~demel/ped/ov/ov.pdf

(4) prof. RNDr. Branislav Rovan, PhD: **Formálne Jazyky a Automaty**, 2013 [strana 13]
http://foja.dcs.fmph.uniba.sk/materialy/skripta.pdf

(5) Tomáš Malík, Diplomová práca: **Neurálny model integrácie jazykových a motorických znalostí simulovaného agenta**. 2007 [strana 16 - 17]
http://cogsci.fmph.uniba.sk/~farkas/theses/tomas.malik.dip11.pdf

(6) Viliam Dillinger, Bakalárska práca: **Trénovanie neurónových sietí s echo stavmi pomocou posilňovania**. [strana 9 - 10]
http://www.dcs.fmph.uniba.sk/bakalarky/obhajene/getfile.php/BAKApraca.pdf?id=121&fid=227&type=application%2Fpdf

(7) Martin Lipták, Bakalárska práca: **Automatizované čistenie verejných dát**. 2012 [strana 22] *https://martinliptak.files.wordpress.com/2012/05/vsetko.pdf*

(8) www.php.net

(9) www.jquery.com

(10) http://en.wikipedia.org/wiki/Markov_decision_process

(11) http://en.wikipedia.org/wiki/Markov_chain

(12) http://en.wikipedia.org/wiki/Deterministic_finite_automaton

(13) http://skauting.sk/verejnost