Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Interaktívny portál na vyučovanie dynamického programovania

Bakalárska práca

2016
Michal Smolík

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Interaktívny portál na vyučovanie dynamického programovania

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	2508 Informatika
Školiace pracovisko:	Katedra informatiky
Školiteľ:	Michal Forišek, PhD

Bratislava, 2016
Michal Smolík

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:
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:

Cieľ:

Literatúra:

Kľúčové slová:

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

Dátum zadania:

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

.....
študent

.....
vedúci práce

Poďakovanie:

Abstrakt

Táto práca sa zaoberá implementáciou interaktívneho portálu na vyučovanie dynamického programovania. Výučba na našom portáli prebieha hlavne riešením implementačných úloh, ktoré automaticky testujeme. Portál má aj kapacitu sledovať postup používateľov a náročnosť úloh. Práca obsahuje aj interaktívnu vizualizáciu rekurzívnych výpočtov a memoizácie.

Portál pozostáva z Django databázového servera a jednoduchého HTML používateľského prostredia.

Kľúčové slová:

Abstract

Keywords:

Obsah

Úvod **1**

1 Špecifikácie projektu **2**
1.1 Popis . . . 2
1.2 Učebný proces . . . 2
1.3 Prihlasovanie a oprávnenia . . . 2

2 Návrh používateľského prostredia **4**
2.1 Proces výučby . . . 4
2.1.1 Postup úlohami . . . 4
2.1.2 Rady . . . 4
2.1.3 Bodovanie a rebríček . . . 5
2.1.4 Hodnotenie úloh . . . 5
2.1.5 Diskusia . . . 5
2.2 Prostredie . . . 6
2.2.1 Základné stránky . . . 6
2.2.2 Detail úlohy . . . 6
2.2.3 Vizualizácia rekurzívnych výpočtov . . . 6

3 Návrh servra **8**
3.1 Použité technológie . . . 8
3.1.1 Django 1.9 . . . 8
3.1.2 REST framework . . . 8
3.2 Testovač . . . 9
3.2.1 Stiahnutie vstupov a poslanie výstupov . . . 9
3.2.2 Testovanie skriptom u používateľa . . . 9
3.2.3 Testovanie na servri . . . 10
3.3 Bezpečnosť . . . 10
3.3.1 Autorizácia . . . 10
3.3.2 Prihlasovanie cez externé portály . . . 11
3.4 Administrácia a zbieranie dát . . . 11

3.4.1 Pridávanie a hodnotenie úloh 11
3.4.2 Zbieranie dát o používateľoch 12
3.4.3 Zbieranie dát o úlohách 12
3.4.4 Oprávnenia používateľa a administrátora 13

4 Dokumentácia používateľského prostredia 14
4.1 Stránky a ich adresy 14
4.1.1 Komunikácia so servrom 14
4.2 Vizualizácia 16
4.2.1 Formát 16
4.2.2 Príprava 17
4.2.3 Tvorba stromu výpočtu 17
4.2.4 Vykresľovanie 18
4.2.5 Vykresľovanie s memoizáciou 20

5 Dokumentácia servra 22
5.1 Modely 22
5.1.1 Lesson - úloha 22
5.1.2 User - používateľ 23
5.1.3 Submit - riešenie 24
5.1.4 Comment - komentár 24
5.1.5 Hint - rada 24
5.1.6 Rating - hodnotenie 25
5.1.7 LessonStat - štatistika úlohy 25
5.1.8 UserStat - štatistika používateľa 26
5.1.9 UserLessonWrapper - obal používateľ-úloha 26
5.2 Pomocné funkcie 26
5.2.1 Testovač 27
5.2.2 Posúvanie úloh 27

Záver 30

Úvod a ciele práce

Dynamické programovanie je pre veľa začínajúcich programátorov zastrašujúce, pretože vyžaduje zmenu spôsobu rozmýšľania o probléme. Potrebujú sa naučiť, ako problém rozdeliť na menšie podproblémy a z ich riešení nájsť aj riešenie pôvodného problému.

Preto vytvárame miesto, kde sa žiaci môžu aj zo svojho domova učiť o základných princípoch a konkrétnych využitiach rekurzívnych výpočtov, memoizácie a dynamického programovania interaktívnym spôsobom. Náš portál je prostredie, kde sa používatelia môžu od základov naučiť riešiť úlohy z dynamického programovania aj bez účasti učiteľa.

Je dôležité najprv spomenúť, že cieľom tejto práce je iba implementácia funkcií portálu, nie výčbového obsahu. Preto budeme rozprávať iba o špecifikáciach, návrhu ich splnenia a konkrétnej implementácii portálu, nie o výučbe ako takej.

Hlavným cieľom práce je teda poskytnutie priestoru pre aktívnu výučbu, pretože programovanie sa najlepšie učí skúšaním. Preto sa výučba na našom portáli zameriava na zadávanie a automatické testovanie implementačných úloh. Ďalšia z hlavných funkcií portálu je nteraktívna vizualizácia rekurzívnych výpočtov a memoizácie, ktorej úlohou je prehľadným spôsobom ukázať túto niekedy náročnú, ale mimoriadne dôležitú časť programovania.

Ďaľším cieľom je zaistenie kvality výučby. Tento cieľ v tejto práci neovplyvníme priamo, pretože samotný učebný obsah nie je v rozsahu tejto práce, ale poskytneme viaceré nástroje, ktorými zjednodušíme jej zlepšovanie.

V prvej kapitole si definujeme špecifikácie a požiadavky tohto projektu. V druhej a tretej kapitole si opíšeme postupne návrh používateľského prostredia a servra a v posledných dvoch aj ich implementáciu v prvej pracovnej verzii projektu.

Kapitola 1

Špecifikácie projektu

Tu si uvedieme špecifické požiadavky na náš portál.

1.1 Popis

Projektom je implementácia jednoduchého internetového portálu slúžiaceho na interaktívnu výučbu dynamického programovania.

Portál bude pozostávať z databázového servra a používateľského prostredia prístupného prehliadačom.

Okrem výučbovej funkcie, tento portál bude implementovať aj zbieranie dát o výkone používateľov.

1.2 Učebný proces

Výučba bude rozdelená do samostatných implementačných úloh, ktoré budú automaticky testované na servri. Tieto úlohy budú rozdelené na do úrovní obtiažnosti. V každej úrovni bude práve jedna povinná a ľubovoľný počet nepovinných úloh. Používatelia budú postupovať po úrovniach riešením povinných úloh.

Projekt bude obsahovať aj interaktívnu vizualizáciu rekurzívnych výpočtov, v ktorej si používatelia budú mocť pozrieť proces výpočtu ľubovoľnej rekurzívnej funkcie.

1.3 Prihlasovanie a oprávnenia

Portál bude poskytovať viacero možností autorizácie: klasickú registráciu na náš server a prihlasovanie cez externé siete ako Facebook.

Kontá v našom portáli budú rozdelené jednoducho na používateľské a administrátorské.

Používateľ bude mať možnosť:

- Prezerať si úlohy ktoré sú na jeho úrovni alebo nižšej
- Posielať riešenia týchto úloh
- Hodnotiť úlohy, ktoré správne vyriešil
- Nechávať komentáre k úlohám
- Prezerať si svoj postup a porovnávať sa s ostatnými používateľmi
- Používať s vizualizáciu rekurzívnych výpočtov

Administrátor bude mať navyše ešte tieto možnosti:

- Upravovať a pridávať úlohy
- Prezerať a upravovať výsledky všetkých používateľov
- Prezerať dáta pozbierané servrom

Kapitola 2

Návrh používateľského prostredia

V tejto kapitole si predstavíme používateľské prostredie a ako bude výučba vyzerať z pohľadu používateľa.

2.1 Proces výučby

2.1.1 Postup úlohami

Hlavný spôsob výučby poskytovaný naším portálom je zadávanie implementačných úloh, ktoré používateľ na svojom zariadení vyrieši a pošle nám riešenie, ktorého správnosť overíme. Ďalšie správanie portálu závisí na správnosti riešenia (podľa implementácie testovača (3.2) sa pod riešením môže rozumieť samotný kód alebo archivované súbory s výstupmi).

Úlohy sú rozdelené na povinné a nepovinné a zoradené do rôznych úrovní. Na každej úrovni je jedna povinná a ľubovoľný počet nepovinných úloh. Ak používateľ správne vyrieši povinnú úlohu, jeho úroveň sa zvýši o 1 a odomknú sa mu nové úlohy. Nepovinné úlohy neodomykajú nové úrovne, ale rátajú sa používateľovi do jeho skóre (2.1.3).

2.1.2 Rady

Ak si používateľ nevie rady s úlohou, bude mať možnosť požiadať o radu. Každá úloha bude mať ľubovoľne veľa rád - podľa uváženia administrátora. Používateľ pri začatí riešenia úlohy nebude mať k dispozícii žiadnu radu, ale môže o ňu kedykoľvek požiadať tlačidlom na stránke úlohy, ktoré mu sprístupní jednu ďalšiu radu. Používateľovi sa vždy pri zobrazení úlohy ukážu aj všetky rady ku ktorým má prístup.

2.1.3 Bodovanie a rebríček

Každý používateľ bude mať úroveň a skóre, ktoré ukazujú jeho postup učebným procesom.

Úroveň používateľa je jednoducho najvyššia úroveň úloh ktoré sa podarilo používateľovi odomknúť.

Skóre je o niečo zložitejšie: všetky správne vyriešené úlohy (povinné aj nepovinné) pridajú používateľovi do skóre svoju úroveň. Napríklad prvá úloha sa ráta za 1 bod a úloha na vyššej, povedzme piatej úrovni pridá 5 bodov.

Používatelia budú mať aj možnosť pozrieť si svoje poradie v rebríčku a porovnať sa s ostatnými používateľmi. Sú dva rôzne rebríčky: rebríček úrovní a rebríček skóre.

Každý používateľ bude môcť vidieť v rebríčkoch seba a všetkých ostatných používateľov ktorí budú súhlasiť so zverejnením svojho skóre (používateľ bude môcť tento súhlas vo svojom profile kedykoľvek zmeniť).

K rebríčkom bude mať prístup aj používateľ, ktorý nesúhlasí so zverejnením svojej úrovne a skóre. Takýto používateľ stále bude vidieť svoje poradie v porovnaní s ostatnými používateľmi tak isto ako ostatní používatelia s tou zmenou, že sa nezobrazí v rebríčkoch ostatným používateľom.

2.1.4 Hodnotenie úloh

Používatelia budú mať môcť po správnom vyriešení úlohu ohodnotiť.

Budú mať možnosť (ale nie povinnosť) ohodnotiť jej obtiažnosť a zaujímavosť na stupnici od 1 do 5. Tieto hodnotenia sa budú priemerovať a ich priemery budú prístupné iba administrátorovi. Každé hodnotenie bude prístupné iba používateľovi ktorý ho vytvoril a administrátorovi. Každý používateľ bude môcť ohodnotiť úlohu iba raz.

2.1.5 Diskusia

Každá úloha bude mať diskusiu, v ktorej bude možné rozoberať úlohu. V momentálnom návrhu je celá diskusia prístupná všetkým používateľom ktorí majú prístup k úlohe, z čoho ale môže vzniknúť problém prezrádzania správnych riešenie ostatným používateľom, ktorí ešte úlohu nevyriešili

Ak tento problém vznikne, budeme musieť rozdeliť diskusiu na dve: pre tých používateľov, čo úlohu vyriešili a pre tých čo ešte nie.

Ak budú stále pretrvávať problémy (napríklad používateľ prezradí riešenie skôr ako ho odovzdá na testovanie), môžme povoliť diskusiu iba pre úspešných riešiteľov.

Toto riešenie ale môže byť v rozpore s účelom diskusie, nakoľko sa môže stať, že používatelia sa po vyriešení úlohy o ňu prestanú zaujímať. Preto riešenie tohto problému

zatiaľ necháme na administrátorovi, aby zmazal všetky diskusné príspevky s radami, ktoré príliš zjednodušujú úlohu a iným nevhodným obsahom.

2.2 Prostredie

Predstavíme jednotlivé stránky ku ktorým bude používateľ mať prístup a aké funkcie poskytujú.

2.2.1 Základné stránky

Portál bude obsahovať stránky na registráciu nových používateľov, prihlásenie, zoznam úloh a používateľov profil. Tieto stránky sú jednoduché a ich funkcia evidentná, preto ich iba spomenieme bez podrobného popisu.

2.2.2 Detail úlohy

Stránka ktorá zobrazí detail jednej úlohu: jej názov, zadanie, rady o ktoré používateľ požiadal, hodnotenie jeho riešení generované testovačom a nasledujúce možnosti:

- stiahnuť vstupné dáta alebo testovací skript - poskytnutie tejto možnosti závisí od implementácie testovača (3.2)
- odovzdať riešenie na otestovanie
- po správnom vyriešení úlohy ju ohodnotiť (2.1.4)
- zobraziť diskusiu a pridať komentár
- požiadať o ďalšiu radu

2.2.3 Vizualizácia rekurzívnych výpočtov

Jedna z najťažších častí učenia sa dynamického programovania je pochopenie rekurzívnych funkcií. Preto náš portál bude poskytovať možnosť si v prehľadnej forme zobraziť výpočet rekurzívnej funkcie zadanej používateľom.

Na stránke vizualizácie si používateľ bude môcť napísať svoju funkciu, ktorej stránka vytvorí a zobrazí strom výpočtu. Každý vrchol bude jedno zavolanie funkcie a bude ukazovať všetky argumenty.

Vrcholy budú vykresľované postupne, podľa poradia zavolania. Návratová hodnota zavolania sa zobrazí hneď, ak funkcia samú seba ďalej nevolá. Ak volanie funkcie zavolá funkciu znova (alebo vrchol volania má deti), výsledok zobrazíme až keď všetci potomkovia jej vrchola v strome majú zobrazenú hodnotu. Týmto sa snažíme o to,

aby používateľ lepšie videl, ako postupoval výpočet a v ktorom momente výpočtu boli spočítané ktoré návratové hodnoty.

Vizualizácia bude vedieť pracovať aj s memoizáciou, čiže pamätaním si návratových hodnôt pre argumenty a odpovedanie z pamäte pri každom ďalšom zavolaní funkcie s rovnakými argumentmi. Pri výpočte s memoizáciou budeme zobrazovať aj zoznam argumentov a ich hodnôt, ktoré už boli vypočítané. V prípade vykreslenia vrcholu s argumentmi, pre ktoré hodnotu už poznáme, ukážeme aj na vrchol kde bola hodnota prvý krát vypočítaná.

Aj pre túto stránku si uvedieme možnosti, ktoré ponúka používateľovi:

- vybrať si, či výpočet bude používať memoizáciu alebo nie
- vyhodnotiť zadanú funkciu a spustiť vizualizáciu jej výpočtu
- zastaviť/znova spoustiť vizualizáciu
- krokovať vizualizáciu dopredu alebo dozadu (pri krokovaní vizualizáciu automaticky zastavíme)
- vykresliť celý strom výpočtu alebo začať vizualizáciu odznova

Uprednostníme počítanie a vytvorenie stromu pred vykreslením oproti počítaniu počas vykresľovania kvôli jednoduchšej implementácii počítania, ale aj vykresľovania samotného.

Je možné implementovať aj spoluprácu s úlohami, kde by sa úloha mohla odkazovať na konkrétny vstup vizualizácie, napríklad formou linku ktorý používateľa presmeruje na stránku vizualizácie kde bude zadaný potrebný kód.

V momentálnom návrhu ale pre túto funkciu nie je potreba, pretože rovnaký efekt docielime jednoducho napísaním potrebného kódu, ktorý používateľovi odporučíme skopírovať a vyhodnotiť vo vizualizácii manuálne. Oba spôsoby docielia rovnaký výsledok a automatické presmerovanie ušetrí iba málo používateľovho času takže ho považujeme za nepotrebné.

Ďalšia možnosť pre budúci vývoj je napríklad ukázať iba strom výpočtu bez kódu, čo môžme využiť ako radu k úlohe alebo úplne nový typ úlohy, kde by používateľ musel nájsť funkciu, ktorá tento strom generuje.

Vizualizácia výpočtu nám zjavne ponúka veľké množstvo možností, ktoré určite v budúcnosti využijeme pre zlepšenie výučbového procesu.

Kapitola 3

Návrh servra

V tejto kapitole sa oboznámime s návrhom a špecifikáciami návrhu vnútorného fungovania servera, bezpečnostných protokoloch a možnosťami poskytovanými administrátorovi.

Je možné, že niektoré z týchto funkcií servra nebudú implementované v prvej pracovnej verzii, ale budú prítomné vo finálnom produkte.

3.1 Použité technológie

3.1.1 Django 1.9

Django je open source framework nad programovacím jazykom Python, ktorý uľahčuje vývoj a údržbu webových aplikácií. Poskytuje funkcionalitu servera na nízkych úrovniach (správa databázy, HTTP komunikácia a iné), čo umožňuje vývojárovi sa sústrediť na tvorbu samotného obsahu. Django takisto ponúka jednoducho implementovateľné možnosti ochrany pred niektorými najbežnejšími útokmi, ako napríklad XSS, SQL injection alebo CSRF.

Server používa framework Django 1.9 pre jednoduchosť vývoja spravovania databáz, autentifikácie popoužívateľov a prostredia pre administrátora.

3.1.2 REST framework

Django REST framework je v projekte využitý na jedoduché a prehľadné zobrazovanie obsahu databázy, odpovedí na HTTP požiadavky a na implementáciu token autorizácie. Umožňuje priamo v prehliadači posielať a zobraziť odpovede na požiadavky, čo zjednodušuje debugovací proces. Uľahčuje aj implementáciu externej OAuth autorizácie (3.3.2)

Takisto obsahuje implementáciu pre_save a post_save signálov ktoré používame pri modeloch ktorých pridanie alebo zmeny priamo menia obsah iných modelov (napríklad

pri automaticko zarovnávaní úloh (5.1.1)).

3.2 Testovač

Testovač je najdôležitejšia časť portálu, pretože overí správnosť používateľovho riešenia. Sú tri možné spôsoby testovania:

3.2.1 Stiahnutie vstupov a poslanie výstupov

Toto je najjednoduchší spôsob implementácie testovača. Používateľ si z nášho portálu si stiahne testovacie vstupy, na ktorých spustí svoj program, výstup zapíše do súbora a súbor pošle na server, ktorý skontroluje správnosť odpovede.

Veľká výhoda tohto prístupu je jeho jednoduchosť, ale má značnú nevýhodu: nezaručuje že používateľ napísal optimálny program. Účelom portálu je učiť používateľov dynamické programovanie, ale tento testovač umožňuje správne vyriešiť úlohu aj hrubou silou.

Tento problém by sa dal obísť používaním úloh, ktoré majú mimoriadne časovo zložité riešenie hrubou silou (napríklad $O(2^n)$), ktoré by na väčších vstupoch vyžadovali niekoľko dní počítania. Toto riešenie je samozrejme nežiadúce keďže veľmi obmedzuje výber úloh. Preto bude toto testovanie používané iba počas procesu vývoja a nie vo finálnom produkte.

3.2.2 Testovanie skriptom u používateľa

Druhý možný spôsob testovania je podobný predošlému s jednou zmenu: používateľ si miesto vstupov stiahne testovací skript, ktorý spustí používateľom zadaný program na vstupoch, ktoré skript stiahne z nášho servera. Potom odošle výstupy a tie sú porovnané so správnymi odpoveďami.

Toto riešenie nám umožňuje merať čas potrebný na beh programu, čo je značné vylepšenie oproti predošlému riešeniu, pretože ak používateľ nijak nezasiahne do skriptu, vieme zistiť či sa mu podarilo úlohu riešiť časovo optimálnym algoritmom.

Používateľ ale môže skript viacerými spôsobmi napadnúť, napríklad manipulovať s časovým obmedzením alebo získať vstupy, vyriešiť ich iným, pomalším algoritmom a vytvoriť program ktorý pre každý vstup vypíše správny výstup zo súboru alebo priamo zapísaný v zdrojovom kóde.

Tieto chyby sa dajú vyriešiť zabezpečením skriptu proti útokom, ale žiadna ochrana nie je stopercentná. Mohli by sme teda napríklad neakceptovať výsledky, ktorých podozrivo vyzerajúce časy (t.j. časy, ktoré sa nesprávajú podľa krivky, časy dlhšie ako

limit, ktoré ale skript sa nezastavil atď.). Toto riešenie ale nie je akceptovateľné, pretože by mohlo stále prepustiť falšované výsledky alebo odmietnuť správne riešenie.

Na ochranu vstupov sa dá použiť procedurálna generácia alebo náhodné vyberanie z väčšej množiny vstupov, ale tie pridávajú prácu administrátorovi a komplikujú pridávanie nových úloh.

3.2.3 Testovanie na servri

Tretí spôsob testovania prebieha na našom serveri. Používateľ pošle zdrojový kód svojho riešenia, ten na serveri skompilujeme a vyhodnotíme s časovým obmedzením behu.

Veľkou výhodou je, že používateľ nemá prístup k testovaným vstupom a správnym výstupom, takže úloha nebude vyhodnotená za správne vyriešenú pri neoptimálnom programe kvôli vyprchaniu časového limitu (závisí od dĺžky vstupu a časového limitu, ktoré zadáva administrátor). Rovnako je oveľa ťažšie napadnúť a upraviť testovač.

Nevýhodou testovača je napríklad zložitá implementácia, pretože musí vedieť kompilovať čo najviac programovacích jazykov a zároveň musíme dávať pozor aby poslaný program nijako nenarušoval bezpečnosť servera (nečítal z pamäťe ktorú nealokoval, nespúšťal žiadne iné programy, nepoužíval niektoré systémové volania a podobne). Ďalšou nevýhodou sú nároky na výpočtový čas servera: keďže proces testovania spúšťa program na serveri, môže server spomaliť.

Nakoľko pri tomto testovači je najťažšie mať správne výsledky v časovom limite a nemať optimálne riešenie, vo finálnom produkte bude implementovaný tento testovač. Ak ale sa zvýši popularita nášho portálu a vznikne problém s rýchlosťou testovania, bude možno treba zvážiť buď vylepšenie hardvéru servera alebo implementáciu predošlého testovača (3.2.2)

3.3 Bezpečnosť

3.3.1 Autorizácia

Portál bude používať token autentifikáciu ktorá z pohľadu používateľa vyzerá nasledovne:

Používateľ pri prihlasovaní pošle prihlasovacie meno a heslo (tento krok je iný pri prihlasovaní cez externú doménu) a ako odpoveď dostane náhodne vygenerovaný reťazec o dĺžke približne 40 znakov. Potom všetky požiadavky, pre ktoré server overuje totožnosť používateľa musia v hlavičke uviesť tento token. Ak tieto požiadavky uvedú neplatný alebo žiadny token, budú zamietnuté. Server si pamätá používateľov token až kým sa používateľ neodhlási alebo mu nevyprší platnosť po dlhšej neaktivite.

Všetky požiadavky musia obsahovať token (buď používateľov alebo CSRF token), čo pomáha používateľa chrániť pred CSRF útokom.

3.3.2 Prihlasovanie cez externé portály

Portál bude mať viacero možností prihlasovania sa cez externé portály. Použijeme prihlasovaciu schému OAuth 2.0, aby komunikácia s externým portálom prebiehala iba pri prihlasovaní po ktorom používateľov prehliadačl komunikuje so serverom ako keby sa prihlásil priamo na náš portál bez využitia externej služby.

Prihlasovanie cez OAuth 2.0 prebieha nasledovne:

1. Používateľ sa prihlási na externý portál a potvrdí našej aplikácii vyžiadané oprávnenia
2. Od externého portálu získa prístupový token na ten externý portál
3. Tento token pošle nášmu serveru
4. Náš server pošle token externému portálu spolu s tajným ID našej aplikácie, aby externý portál vedel, že požiadavka prišla od našej aplikácie
5. Ak externý portál potvrdí požiadavku, náš server vráti používateľovi náš token a ďalej komunikujú iba s týmto novým tokenom.

Pomocou tejto schémy sa bude dať prihlásiť cez Facebook, Google+ a GitHub. Tajné ID použité v kroku 3 je prístupné iba administrátorom.

3.4 Administrácia a zbieranie dát

Administrátorovi poskytneme prostredie, v ktorom bude môcť prezerať, pridávať, upravovať ale aj mazať všetky modely ku ktorým má prístup. Oprávnenie používať toto prostredie bude mať samozrejme iba administrátor.

3.4.1 Pridávanie a hodnotenie úloh

Pridávanie a spravovanie úloh je hlavnou povinnosťou administrátora. Administrátor okrem zadávania úloh musí poskytnúť aj vzorové riešenie, prípadne nejaké rady k riešeniu úlohy a zaradiť ju podľa obtiažnosti.

3.4.2 Zbieranie dát o používateľoch

Náš portál bude okrem vyučovania dynamického programovania schopný aj zbierať údaje o schopnostiach používateľov a ich zlepšovaní sa postupom cez naše vyučovanie. Server bude zbierať viacero možných údajov o používateľskej aktivite:

- Počet navštívených návodových a teoretických stránok na našom portáli
- Frekvencia návštev počas riešenia úloh (ako často používateľ používa naše návody na pomoc s riešením)
- Priemernú frekvenciu a dĺžku návštev nášho portálu
- Čas od prečítania úlohy po jej správne vyriešenie (nie veľmi dôležitý, pretože neberie do úvahy prestávky pri riešení)
- Počet neúspešných pokusov o riešenie
- Rozdiel priemerného a používateľovho hodnotenia úlohy (až keď bude dostatočne veľa hodnotení aby tento údaj niečo znamenal)

Z týchto údajov budeme vedieť zistiť, ako efektívne je vyučovanie (napríklad, žiak sa pravdepodobne zlepšil ak prvé úlohy označoval ťažšie ako priemer a neskoršie ľahšie), použiť spätnú väzbu na zlepšenie kvality výučby a prípadne neskôr implementovať automatické odporúčanie úloh podľa používateľovych schopností.

3.4.3 Zbieranie dát o úlohách

Pre kvalitnejší výučbový proces je nutné vedieť, ktoré úlohy sú populárne a koľko sa na nich žiaci naučia. Preto budeme zapisovať dáta z používateľskej aktivity aj pre úlohy. Medzi tieto dáta bude patriť:

- Priemerné hodnotenie zložitosti a zaujímavosti úlohy
- Priemerný čas od zadania do vyriešenia úlohy
- Priemerný počet nesprávnych riešení pred správnym
- Priemerný počet použitých rád

Tieto dáta bude môcť vidieť iba administrátor a na ich základe by mal meniť úlohy tak, aby používatelia boli s nimi čo najspokojnejší.

Administrátor by sa preto mal snažiť, aby každá úloha mala ideálne používateľské hodnotenie (2.1.4) Je najlepšie, aby každá úloha mala čo najvyššie hodnotenie zaujímavosti, nakoľko zaujímavé úlohy lepšie motivujú používateľov.

S hodnotením zložitosti je to ale inak: žiadna úloha by nemala byť príliš ľahká (aby bola pre používateľa výzvou) ani príliš ťažká (aby ju používateľ zvládol), preto ideálny priemer hodnotenia zložitosti je 3.

Server bude poskytovať administrátorovi možnosť posúvať úlohy vyššie a nižšie v poradí, v ktorom ich používatelia riešia. Takisto bude označovať úlohy, ktoré majú príliš vysokú alebo príliš nízku obtiažnosť vzhľadom na úroveň riešiteľa a navrhovať nové miesto v poradí, kam ich zaradiť.

Podobne bude označovať úlohy, ktoré sú považované za najmenej zaujímavé a bude na administrátorovi aby posúdil, či daná nezaujímavá úloha je dôležitá pre proces výučby, alebo či ju možno zmeniť alebo odstrániť.

3.4.4 Oprávnenia používateľa a administrátora

Bežní používatelia majú prístup iba k učebným materiálom, odomknutým úlohám (vyriešeným aj nevyriešeným) a svojim riešeniam. Ďalšie úlohy sa používateľovi odomknú, iba ak správne vyrieši všetky prerekvizitové úlohy. Všetky dáta zozbierané servrom (3.4.2) budú od používateľa skryté, pretože by mohli spôsobiť frustráciu u pomalšie sa učiacich používateľov.

Administrátor bude mať prístup ku všetkým úlohám, riešeniam, zozbieraným dátam všetkých používateľov a ich priemerom. Bude mať aj možnosť manuálne meniť niektoré hodnotenia a odomykať alebo zamykať používateľom úlohy.

Túto možnosť obchádzať pravidlá administrátorovi dávame pre prípad, že vznikne chyba a používateľovi sa napríklad odomkne úloha ku ktorej nemal mať prístup alebo neodomkne taká, ktorej všetky prerekvizity splnil. Používateľ v prípade takejto chyby bude môcť kontaktovať administrátora, ktorý preverí, či naozaj nastala chyba a bude ju môcť napraviť bez debugovania servera.

Kapitola 4

Dokumentácia používateľského prostredia

V tejto kapitole si priblížime implementáciu prednej časti portálu, s ktorou budú používatelia interagovať. Povieme si komunikácii so servrom ale aj o implementácii vizualizácie rekurzívnych výpočtov.

4.1 Stránky a ich adresy

V prvej pracovnej verzii sú implementované tieto stránky (uvedené s URL adresami)

- registrácia - */user/register/*
- prihlasovanie - */user/login/*
- zoznam úloh - */lessons/*
- detail úlohy - */lesson/id/* (id je unikátne identifikačné číslo úlohy)
- vizualizácia - */visualisation/*
- administrátorské prostredie - */admin/*

Administrátorské prostredie je predvolené prostredie Django frameworku bez modifikácií. Toto prostredie poskytuje všetky požadované možnosti (prezeranie, pridávanie, úprava a mazanie objektov) vo veľmi prehľadnej a jednoduchej forme.

4.1.1 Komunikácia so servrom

Na komunikáciu so servrom používame hlavne knižnicu jQuery, konkrétne asynchronickú metódu AJAX, kvôli jej jednoduchému používaniu a podpore pridania autorizačnej hlavičky. Autorizačná hlavička je údaj v hlavičke HTTP požiadavky ktorý

určuje totožnosť používateľa, v našom prípade napríklad `{'Authorization': 'Token a5b498587e33d1f44a97c2328618dd15373b0705'}`.

Tento token dostaneme ako odpoveď na prihlasovaciu požiadavku a uložíme si ho v lokálnej pamäti prehliadača pomocou zavolania `localStorage`, konkrétne takto: `localStorage.setItem('token', response['token'])`. Potom token môžme kedykoľvek získať pomocou zavolania metódy `getItem()`. Väčšina požiadaviek ktoré posielame na server sú si veľmi podobné, preto uvedieme iba jednu typickú požiadavku:

```
1  $.ajax({
2     url: "../../api/lesson/"+id+"/",
3     type: "GET",
4     headers: {'Authorization':'Token '+localStorage.getItem('
          ↪ token')},
5     success: function(data) {
6       title = document.getElementById("title")
7       title.innerHTML = "Lesson "+data['name']
8       problem = document.getElementById("problem")
9       problem.innerHTML = "Lesson "+data['problem']
10    },
11    error: function(data) {
12      if (data.status==401) {
13        window.location.replace("../../user/login/")
14      }
15    }
16 });
```

Požiadavka na získanie zadania jednej úlohy

(v tomto príklade je požiadavka trochu zmenená oproti implementácii, aby bola v texte prehľadnejšia)

Tento a viacero iných podobných požiadavok posielame vždy pri načítaní stránky prehliadačom, aby sme získali dáta ktoré chceme zobraziť.

Teraz si vysvetlíme niektoré časti tohto kódu:
V druhom riadku používame premennú id, ktorá označuje unikátne identifikačné číslo úlohy. Toto číslo získame z URL zobrazenej stránky ako prvú vec pri načítaní.

Vo štvrtom riadku pridávame autorizačnú hlavičku, podľa spôsobu, aký sme popísali vyššie.

V piatom až desiatom riadku deklarujeme správanie pri úspešnom získaní odpovede od servra, v tomto prípade zobrazujeme názov a zadanie úlohy v na to určených HTML elementoch stránky.

V jedenástom až päťnástom riadku deklarujeme správanie sa pri chybe. Zatiaľ jediná

chyba, na ktorú reagujeme je 401-UNAUTHORIZED ktorú dostaneme ak posielame správu s nesprávnym alebo úplne chýbajúcim tokenom. V tom prípade používateľa presmerujeme na prihlasovaciu stránku. Toto správanie je použité v každej požiadavke, ktorá vyžaduje autentifikáciu.

Odosielanie na server je o trochu iné, pretože používa metódu POST a musíme pridať aj posielané dáta. To najčastejšie robíme pomocou pridania
`data : $("#form").serialize()`, čo nám dáta získa priamo z hodnôt vo formulári.

jQuery má ale obmedzenie, že kvôli bezpečnosti nedokáže zapisovať do súborov na disku. To znamená, že sťahovanie súborov je náročné na implementáciu. Preto na sťahovanie vstupných súborov (pre implementáciu jednoduchého testovača 3.2.1) používame formulár, ktorý pošle GET požiadavku na požadovanú adresu. Tento formulár ale nemôže mať hlavičku s tokenom, preto sťahovanie vstupov nebude autentifikovať používateľa.

To síce znamená, že používateľ môže mať prístup aj k vstupom úloh, ktoré si ešte neodomkol. Tento problém nepovažujeme za závažný, pretože bez zadania úlohy je malá šanca že používateľ túto úlohu vyrieši. Ďalej, vo finálnom produkte testovač nebude vyžadovať sťahovanie dát, čím bude tento problém úplne odstránený.

Rovnako, jQuery má ťažkosti s posielaním súborov (skoršie verzie dokonca neodkážu posielať súbory). Preto na posielanie riešení používame formát XMLHttpRequest, ktorý funguje podobne ako AJAX, preto si ho ďalej približovať nebudeme.

4.2 Vizualizácia

Vizualizácia rekurzívnych výpočtov (ďalej len vizualizácia) má za úlohu z používateľovej funkcie vytvoriť strom jej výpočtu a ten prehľadne vykresliť. Naša implementácia najskôr vygeneruje celý strom a až potom ho používateľovi zobrazí.

Na vyhodnotenie funkcie používame Javascript a na vykresľovanie používame HTML5 objekt canvas, teda všetok kód vizualizácie je v HTML dokumente, čo znamená že vizualizácia nijak nekomunikuje so servrom.

Popis implementácie vizualizácie si rozdelíme na štyri časti. Najskôr si spomenieme formát a obmedzenia zadávateľných funkcií. Ako druhé si opíšeme kroky pred generovaním stromu, potom generovanie vrcholov stromu a nakoniec vykresľovanie.

4.2.1 Formát

Vizualizácia akceptuje iba funkcie písané v jazyku Javascript. Kvôli jednoduchšej implementácii sa vyhodnocovaná funkcia vždy volá `f`. Používateľov vstup budeme načítavať z dvoch textových vstupov: v jednom napíše názvy parametrov a ich počiatočné hodnoty a v druhom samotné telo funkcie.

4.2.2 Príprava

Po stlačení tlačidla Evaluate si uložíme kód do textovej premennej. Parametre si uložím ako reťazec v ktorom sú jednotlivé parametre oddelované čiarkami a argumenty si uložím do jednorozmerného poľa. Potom pomocou Javascriptovej funkcie `eval` si vstupný kód a parametre prevediem na funkciu ktorú si uložím pod menom `originalFunction`.

Pokračujem tým, že si ako `f` uložím modifikovanú zapisovaciu funkciu ktorú si predstavíme v ďalšej sekcii. Tento krok je veľmi dôležitý, pretože odteraz, vždy keď zavoláme funkciu f v používateľovom kóde, v skutočnosti nespustíme funkciu ktorú zadal, ale jej modifikovanú, zapisovaciu verziu. Máme jednu zapisovaciu funkciu pre obyčajný rekurzívny výpočet a druhú pre memoizačný.

Ako posledný krok prípravy túto modifikovanú funkciu spustím na parametroch ktoré používateľ zadal.

4.2.3 Tvorba stromu výpočtu

Vrcholy stromu si ukladáme do poľa a získavame ich iba pomocou indexovania tohto poľa, aby sme zaručili že každý vrchol máme uložený v pamäti iba v jednej kópii.

Počas výpočtu si vrcholy vkladáme do zásobníka, aby sme vedeli určiť, ktorý vrchol je rodičom ktorého. O detailoch ukladania a vyberania si povieme neskôr.

Pri memoizácii si pamätáme ešte aj prvé výskyty argumentov, čiže zoznam vrcholov v ktorých ukladáme do pamäte nový údaj. Tento zoznam potrebujeme, aby sme vedeli ukázať, kedy sme vypočítali hodnotu v pamäti.

Pre každý vrchol stromu výpočtu si pamätáme nasledovné údaje:

- poradie v zozname vrcholov
- poradie rodiča v zozname vrcholov
- pole argumentov
- návratovú hodnotu
- či sme získali hodnotu vrchola z pamäti (iba pri memoizácii)
- zoznam detí (tiež ich poradí v zozname)
- hĺbku v strome - používané na zistenie y-ovej súradnice
- počet listov, ktoré sú potomkom tohto vrchola (1 ak tento vrchol je list) - používané na zistenie x-ovej súradnice
- súradnice stredu pri vykresľovaní

- či ukazujeme výsledok (potrebu tohto údaja si vysvetlíme v časti Vykresľovanie)

Ako prvé si v modifikovanej funkcii určíme poradie vrchola a jeho otca. Číslo je veľkosť zoznamu vrcholov a rodič je vrchol na konci zásobníka. Zároveň pridáme nový vrchol do zoznamu rodičovich detí. Hĺbkouvrchola je dĺžka zásobníka. Vrchol pridáme do zoznamu všetkých vrcholov.

Ak máme zapnutú memoizáciu, na tomto mieste overíme, či nemáme pre argumenty nového vrchola v pamäti už vypočítanú hodnotu. Ak áno, označíme vrchol ako získaný z pamäte, priradíme mu návratovú hodnotu a vrátime ju. Ak nemáme hodnotu v pamäti, pokračujeme ďalej ako bez memoizácie.

Teraz overíme, či vrcholov nie je viac ako 1000. Ak áno, výpočet zastavíme a používateľovi ukážeme chybovú hlášku. Týmto zabránime nekonečnému volaniu a príliš veľkým a pamäťovo náročným simuláciam.

Aby sme zistili návratovú hodnotu, pridáme vrchol do zásobníka a ako hodnotu priradíme návratovú hodnotu z originálnej funkcie na rovnakých argumentoch.

Originálna funkcia pri výpočte buď hodnotu vráti hneď, alebo ďalej volá funkciu f. f ale je naša zapisovacia funkcia. Preto originálna funkcia pri svojom výpočte zavolá zapisovaciu funkciu pre všetky volania samej seba (s rovnakými argumentami, ako by volala samú seba). Naša zapisovacia funkcia pre všetky argumenty vráti tú istú hodnotu ako pôvodná funkcia. To znamená, že všetky volania pôvodnej funkcie budú zapísané našou zapisovacou funkciou ako vrcholy stromu.

V prípade memoizácie uložíme dvojicu argumenty-výsledok do pamäte a pridáme tento vrchol do zoznamu prvých výskytov argumentov.

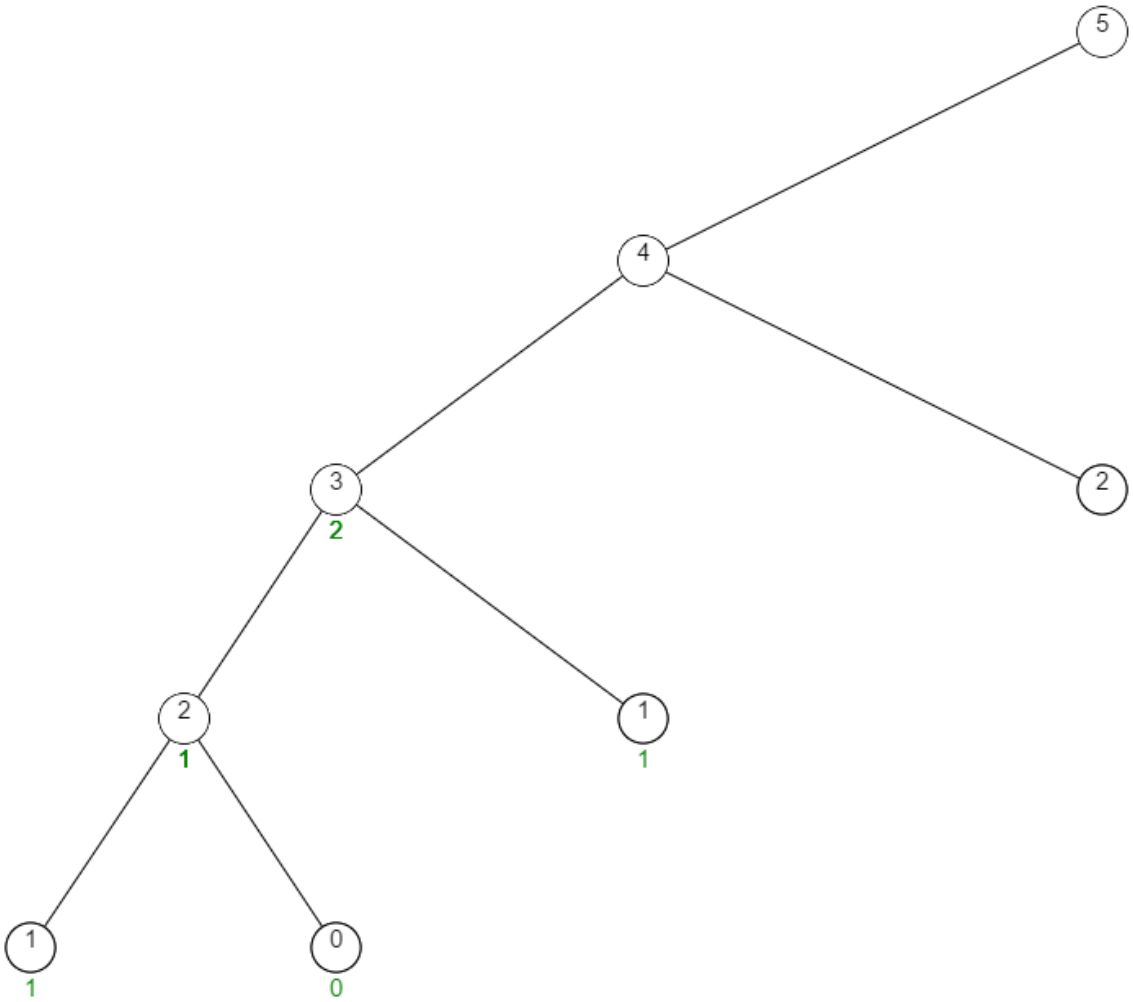
Nakoniec ako počet listov pod terajším vrcholom uložíme sumu počtov listov pod deťmi, odstránime terajší vrchol zo zásobníka a vrátime hodnotu.

4.2.4 Vykresľovanie

V tejto časti si najskôr opíšeme vykresľovanie rekurzie bez memoizácie. Potom si spomenieme, aké zmeny musíme implementovať ak chceme vykresliť aj výpočet s memoizáciou.

V každom kroku buď odhalíme nový vrchol, alebo návratovú hodnotu jedného z už vykreslených vrcholov. Návratovú hodnotu vykreslíme zároveň s vrcholom iba pri listoch. Všetky ostatné vrcholy vykresľujeme bez návratovej hodnoty. Tú odhalíme, iba keď je odhalená návratová hodnota všetkých jeho detí. Tento prístup simuluje výpočet v ktorom poznáme návratovú hodnotu až vtedy, keď vyriešime všetky volania funkcie `f`. Používateľ bude vďaka tomto prístupu mať lepší prehľad o tom, ktoré volanie funkcie ešte prebieha a ktoré je už ukončené návratom hodnoty.

Pri vykresľovaní si pamätáme tri zoznamy: zásobník krokov, zoznam vykreslených a zásobník nevyriešených vrcholov . Vrchol je v zásobníku nevyriešených vrcholov práve vtedy, keď je vykreslený ale jeho hodnota ešte nie je odhalená. Na začiatku simulácie je koreň stromu vykreslený a označený za nevyriešený.



Obr. 4.1: Výpočet piateho Fibonacciho čísla bez memoizácie

Implementujeme dve hlavné funkcie, `nextStep` a `stepBack`:

Krok vpred

Pri zavolaní `nextStep` urobíme krok iba vtedy, keď existuje aspoň jeden nevyriešený vrchol, pretože v opačnom prípade výpočet už skončil. Predpokladajme teda, že existuje aspoň jeden nevyriešený vrchol. Najprv si zoberieme posledný vrchol zo zásobníka nevyriešených a overíme, či sa ho podarilo v minulom kroku vyriešiť. Ak áno, do zásobníka krokov pridáme jeho číslo, odstránime ho z nevyriešených vrcholov a vykreslíme jeho hodnotu.

Ak nie, vykreslíme ďalší vrchol v zozname všetkých vrcholov. Do tohto zoznamu sme každý vrchol zapisovali medzi jeho vytvorením a zistením jeho hodnoty, preto tento zoznam má vrcholy zoradené v poradí, v ktorom boli zavolané.

Pri vykresľovaní ďalšieho vrcholu pridáme do zásobníka krokov -1 a ak je nevyriešený, pridáme ho do zásobníka nevyriešených vrcholov.

Po konci týchto úkonov prekreslíme celý strom.

Krok vzad

Pri kroku vzad ako prvé vyberieme posledný krok zo zásobníka krokov. Ak bol tento krok väčší alebo rovný 0, vieme, že posledný krok bolo odhalenie hodnoty vrchola. Preto tento vrchol pridáme do zásobníka nevyriešených a skryjeme jeho výsledok.

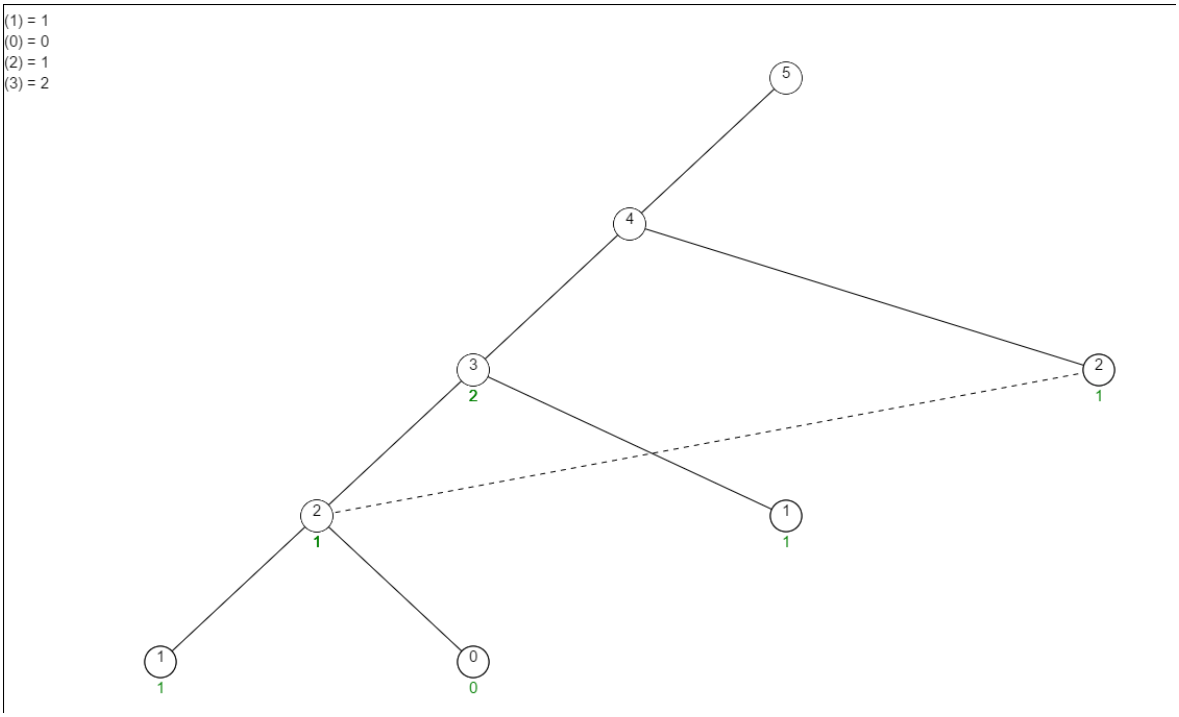
Ak však posledný krok v zásobníku bol -1, musíme skryť posledný odhalený vrchol. Ten teda odstránime zo zoznamu vykreslených vrcholov a ak je nevyriešený, tak aj zo zásobníka nevyriešených.

4.2.5 Vykresľovanie s memoizáciou

Aby sa používatelia lepšie orientovali v memoizačnom výpočte, na ľavej strane vykresľovacieho plátna budeme ukazovať zapamätané hodnoty.

Pri odhalení nového vrchola zistíme, či jeho hodnota bola získaná z pamäti. Ak áno, spojíme ho na jeden krok prerušovanou čiarou s vrcholom, v ktorom sme túto hodnotu vypočítali a uložili. Ak nebol získaný z pamäti, vykreslíme ho ako pri výpočte s memoizáciou.

Pri odhaľovaní hodnoty doteraz nevyriešeného vrchola jeho hodnotu pridáme do vykreslenej pamäti.



Obr. 4.2: Výpočet piateho Fibonacciho čísla s memoizáciou

Kapitola 5

Dokumentácia servra

V tejto kapitole sa pozrieme na prvú pracovnú implementáciu servra. Oboznámime sa s modelmi, pohľadmi a niektorými funkciami, ich možnosťami a použitím.

5.1 Modely

Django servery sledujú návrhový vzor Model-View-Controller (MVC). V prípade Django servrov, model je tabuľka databázy a view je spôsob zobrazenia (napríklad vo forme HTML stránky). Pre pochopenie funkcionality je dôležité vedieť, aké modely a pohľady používame a ako medzi sebou interagujú.
Tu sa oboznámime s dôležitými modelmi, teda tabuľkami databázy. Pri každom modeli spomenieme jeho názov v implementácii a slovenský ekvivalent tohto názvu.

(Každý model obsahuje aj pole id, čo je jedinečné identifikačné číslo automaticky generované Django frameworkom)

5.1.1 Lesson - úloha

Ako prvý si predstavíme asi najdôležitejší model servra, ktorý obsahuje zadanie a testovacie dáta úlohy.

Úlohy majú tieto polia:

- **name**: meno úlohy ktoré sa zobrazuje používateľovi.
- **problem**: znenie úlohy.
- **pub_date**: dátum publikácie úlohy.
- **number**: úroveň úlohy, používateľ musí mať vyriešené všetky povinné úlohy s na nižších úrovniach aby mal k tejto úlohe prístup.
- **optional**: voliteľnosť úlohy. Povinné aj voliteľné úlohy sa odomykajú rovnako, voliteľné ale neodomykajú neskoršie úlohy.

- **inputs**: vstupy na ktorých prebieha testovanie úlohy.
- **correct_solution**: správne riešenie, oproti ktorému sa budú testovať používateľské riešenia.

Pridávanie a upravovanie úloh je jednoduché - stačí v administrátorskom prostredí vyplniť polia a uložiť. Na každej úrovni ale musí byť práve jedna povinná úloha, preto po upravení alebo pridaní novej povinnej úlohy sa poradie upraví tak, aby pridaná alebo zmenená úloha mala po uložení číslo, ktoré jej administrátor zadal. Viac si o spôsobe zarovnania úloh povieme v časti 5.2.2

Na úlohy máme tri rôzne pohľady (views): Jeden ktorý vráti všetky úrovne, voliteľnosti a názvy úloh ku ktorým má používateľ prístup, druhý meno, znenie a úroveň s voliteľnosťou jednej úlohy a tretí iba vstupy alebo skript na stiahnutie (ak používame testovač 3.2.1 alebo 3.2.2)

5.1.2 User - používateľ

Model s osobnými údajmi používateľa.

Polia:

- **username**: prezývka používateľa, nutná na registráciu a prihlásenie, jedinečná
- **email**: emailová adresa používateľa, tiež nutná a jedinečná
- **password**: heslo používateľa, ukladané pomocou hash funkcie poskytovanej REST frameworkom
- **first_name**: prvé meno používateľa, nepovinné
- **last_name**: priezvisko používateľa, taktiež nepovinné

Pri vytvorení nového používateľa sa automaticky vytvorí jeho **UserStat** (5.1.8)
Tu takisto môžem spomenúť model **Token**, ktorý má polia **user** a **key**. **Token** je používaný pri autorizácii (3.3.1). Je generovaný pri každom prihlásení a zmazaný pri každom odhlásení.

5.1.3 Submit - riešenie

Riešenie, ktoré používateľ pošle na server na otestovanie.

Polia:

- **user**: používateľ ktorý riešenie poslal
- **lesson**: úloha ktorú používateľ rieši týmto riešením
- **submittedFile**: súbor s riešením, ktoré bude odovzdané testovaču
- **result**: výsledok - reťazec znakov obsahujúci výpis z testovača

Hneď ako server dostane nové riešenie, spustí testovač (5.2.1) a jeho odpoveď uloží v poli **result**. Ak nie je správne, riešenie iba uložíme a ďalej s ním nič nerobíme. Ak testovač ale vyhodnotí riešenie ako správne (testovač vráti `"OK"`), pred jeho uložením zvýši používateľovu úroveň o 1 (ak je úloha povinná) a označí úlohu ako vyriešenú v modeli **UserLessonWrapper** (5.1.9).

5.1.4 Comment - komentár

Komentár od používateľa na úlohu. Polia:

- **user**: používateľ ktorý vytvoril komentár
- **lesson**: úloha ku ktorej bol komentár pridaný
- **text**: text komentára
- **date**: čas vytvorenia komentára, potrebný na zoraďovanie pri zobrazení

Zatiaľ sú všetky komentáre prístupné všetkým používateľom (ak majú prístup k ich úlohe).

5.1.5 Hint - rada

Rada k úlohe, pre prípad že ju používateľ nevie riešiť.

Polia:

- **lesson**: úloha ku ktorej je táto rada
- **number**: poradové číslo rady
- **text**: text rady

Používateľovi sa pri zobrazení úlohy ukazuje iba toľko rád, o koľko si požiadal. Táto informácia sa ukladá v modeli **UserLessonWrapper** (5.1.9). Používateľ môže kedykoľvek požiadať o ďalšiu radu. Kým existuje nepoužitá rada, zväčšíme číslo použitých rád v **UserLessonWrapper**-i a odpovieme novým, o jednu položku dlhším zoznamom rád.

5.1.6 Rating - hodnotenie

Používateľovo hodnotenie úlohy v kategóriach zložitosti a zábavnosti.

Polia:

- **user**: používateľ, ktorý zaslal toto hodnotenie
- **lesson**: úloha, ktorú hodnotil
- **fun**: hodnotenie zaujímavosti
- **difficulty**: hodnotenie zložitosti

Keďže každý používateľ môže ohodnotiť každú úlohu iba raz, pre každú dvojicu **úloha-používateľ** môže existovať najviac jedno hodnotenie.

Kým používateľ nevyriešil úlohu, používateľské prostredie mu nezobrazí formulár na hodnotenie. Týmto sa snažíme o to aby nemal spôsob, ako poslať hodnotenie úlohy ktorú nevyriešil. Napriek tomu, zavedieme opatrenie proti možnosti manuálne poslať požiadavku a tým obísť podmienku správneho vyriešenia úlohy: pred uložením hodnotenia na servri znova overíme, či používateľ úlohu správne vyriešil. Ak a mu to ešte nepodarilo, odpoveď na poslanie hodnotenia bude chyba 401-UNAUTHORIZED.

Ak sa používateľovi ale podarilo úlohu predtým vyriešiť, vytvoríme nové alebo upravíme existujúce hodnotenie a obnovíme model **LessonStat** (5.1.7), aby mal aktuálne dáta.

5.1.7 LessonStat - štatistika úlohy

Model, ktorý zbiera štatistické dáta o úlohe, ktoré budú neskôr spracované a ukázané administrátorovi. K tomuto modelu má prístup iba administrátor.

Polia:

- **lesson**: úloha ku ktorej štatistika patrí
- **avg_fun**: priemerné hodnotenie zaujímavosti
- **avg_diff**: priemerné hodnotenie zložitosti

- **good_solutions**: počet používateľov ktorí úlohu správne vyriešili (nie počet správnych riešení, pretože účelom tohto poľa je pomôcť zistiť koľko riešení používatelia skúšajú pred správnym vyriešením)
- **bad_solutions**: počet zlých riešení úlohy pred správnym vyriešením úlohy

5.1.8 UserStat - štatistika používateľa

V momentálnej implementácii zbierame iba jeden údaj pre používateľov, ktorý ale používame na zisťovanie, ktoré úlohy používateľovi sprístupniť. Narozdiel od modelu **LessonStat** (5.1.7), táto štatistika je prístupná používateľovi, pretože ide o jeho výsledky.

Tento model bude vo finálnej obsahovať napríklad aj skóre používateľa, ktoré bude podľa priania používateľa zverejnené v rebríčku (2.1.3)

Polia:

- **user**: používateľ
- **progress**: úroveň používateľa - počet správne vyriešených povinných úloh

5.1.9 UserLessonWrapper - obal používateľ-úloha

Model na určovanie postupu používateľa jednou úlohou. Používa sa iba na vnútorné overovanie prístupu používateľa k niektorým modelom. Tento model je unikátny pre každú dvojicu používateľ-úloha

Polia:

- **user**: používateľ
- **lesson**: úroveň používateľa - počet správne vyriešených povinných úloh
- **hints_used**: počet použitých rád k úlohe
- **completed**: či používateľ správne vyriešil úlohu

Tento model sprístupníme administrátorovi na upravovanie, pretože chceme mať možnosť opravovať nepredvídané chyby (napríklad ak používateľ pošle správne riešenie ale úloha sa mu neoznačí za vyriešenú).

5.2 Pomocné funkcie

Teraz si popíšeme implementáciu automatického testovača a zarovnávania pridaných úloh, pretože obe sú dôležité funkcie, ktoré by sme mali ovládať.

5.2.1 Testovač

Testovač je v prvej pracovnej verzii implementovaný podľa najjednoduchšieho prístupu 3.2.1.

Uvedieme si popis kódu testovača a potom aj samotný kód.

Tento testovač otvorí používateľove a správne riešenie úlohy ako .zip súbor. Najprv overí, či počet súborov je rovnaký. Ak nie, vypíše chybovú hlášku `"Wrong file structure"`.

Potom pre všetky súbory zo správneho riešenia skúsi nájsť súbor s rovnakým menom v používateľovom riešení. Ak ho nenájde, tiež vypíše chybu `"Wrong file structure"`. Ak ho nájde, ale obsah súboru sa líši v používateľovom a správnom riešení, vyhodnotí riešenie za nesprávne (hláška `"Wrong answer"`).

Ak všetky súbory z používateľovho riešenia prejdú týmito testami, vieme povedať, že všetky súbory sa zhodovali s tými v správnom riešení a teda riešenie je správne. V tom prípade testovač vráti `"OK"`.

```
1  def compare_files(correctFile, submitFile):
2      correct = zipfile.ZipFile(correctFile)
3      submit = zipfile.ZipFile(submitFile)
4      if len(correct.namelist()) != len(submit.namelist()):
5          return "Wrong file structure"
6      for name in correct.namelist():
7          if name not in submit.namelist():
8              return "Wrong file structure"
9          if submit.read(name) != correct.read(name):
10             return "Wrong answer"
11     return "OK"
```

testovač

Nevýhodou je, že používateľove súbory musia mať rovnaké meno ako tie v správnom riešení, preto je na autorovi úlohy aby v zadaní popísal správny formát riešenia.

My odporúčame ako konvenciu používanie vstupných súborov s príponou .in a výstupov s rovnakým menom, ale príponou .out (napríklad vstupný súbor 01.a.in a korešpondujúci výstupný súbor 01.a.out)

5.2.2 Posúvanie úloh

Každá úroveň úloh musí mať presne jednu povinnú úlohu. Keďže chceme, aby administrátor nemusel úlohy posúvať sám pred pridaním novej alebo pred presunutím existujúcej úlohy na inú úroveň, tento proces zarovnávania si automatizujeme. Funkciu automatického zarovnávania spojíme s modelom **Lesson** pomocou signálu `post_save`.

Tento signál nám zaručí, že pri uložení úlohy sa zavolá zarovnávacia funkcia.

Algoritmus zarovnávania sleduje tieto kroky:

- Všetky povinné úlohy posunieme tak, aby neboli žiadne medzery v poradí.
- všetky povinné úlohy čo majú úroveň väčšiu alebo rovnú úrovni upravovanej/pridanej úlohy posununieme o jednu úroveň vyššie (upravovaná/pridaná úloha nie je posunutá).
- všetky úlohy znovu posununieme tak, aby sme vyplnili medzery (pretože predošlý krok mohol vytvoriť nové medzery ak presúvame existujúcu úlohu na vyššiu úroveň)
- nakoniec pridáme ukladanú úlohu a zarovnáme ju tiež, nakoľko administrátor mohol urobiť chybu a pridať úlohu s číslom väčším ako počet povinných úloh a teda by bola nedosiahnuteľná.

```
1  def PushOtherLessons (sender, instance, *args, **kwargs):
2    if not(instance.optional):
3      rank = instance.number
4      non_optionals = [x for x in Lesson.objects.all().order_by('
           ↪ number')
5        if not(x.optional) and x.id!=instance.id]
6      #fill the gaps
7      for l in non_optionals:
8        while len(Lesson.objects.filter(number = (l.number-1)))==0
             ↪  and l.number>1:
9          Lesson.objects.filter(id=l.id).update(number = l.number
               ↪ -1)
10         l.number -= 1
11         print(l)
12     #push later lessons one level up
13     if len(Lesson.objects.filter(number=rank))>0:
14       Lesson.objects.filter(number__gte = rank, optional=False).
             ↪ exclude(id=instance.id).update(number = F('number')
             ↪ +1)
15     non_optionals = [x for x in Lesson.objects.all().order_by('
           ↪ number') if not(x.optional) and x.id!=instance.id]
16     #fill the gaps again
17     for l in non_optionals:
18       while len(Lesson.objects.filter(number = (l.number-1)))==0
             ↪  and l.number>1:
```

```
19          Lesson.objects.filter(id=l.id).update(number = l.number
               ↪ -1)
20          l.number -= 1
21      #adjust saved lesson
22      while len(Lesson.objects.filter(number = (instance.number-1)
           ↪ ))==0 and instance.number>1:
23        Lesson.objects.filter(id=instance.id).update(number =
             ↪ instance.number-1)
24        instance.number -= 1
25  post_save.connect(PushOtherLessons, sender=Lesson)
```

Posúvanie úloh

Vypĺňanie medzier sa uskutočňuje pomocou `while` cyklu, v ktorom dekrementujeme úroveň zarovnávanej úlohy, až kým nenarazíme na prvú povinnú úlohu ktorá má úroveň o 1 menšiu ako práve zarovnávaná úloha. Vtedy cyklus skončíme a môžme si byť istí, že v každej úrovni menšej alebo rovnej úrovni zarovnávanej úlohy je práve jedna povinná úloha.

Na ukladanie zmien používame metódu `Lesson.update()` miesto `Lesson.save()`. `Lesson.save()` totiž zavolá `post_save` signál, ktorý by spustil proces zarovnávania, ktorý by tiež spustil ďalší proces zarovnávania a funkcia by nikdy neskončila.

Ako nové číslo úlohy v metóde `Lesson.update()` používame pri posúvaní úloh hore výraz F, konkrétne `F('number')`. Tento výraz F nám dovoľuje získať hodnotu poľa práve obnovovaného objektu bez toho, aby sme mu priradili premennú čím skracujeme a sprehľadňujeme kód. V ostatných prípadoch výraz F nie je potrebný, pretože už máme obnovovanú úlohu označenú premennou l alebo instance.

Záver

Literatúra

[1] Django documentation. https://docs.djangoproject.com/en/1.9/. [2015-12-07].

[2] Facebook login for the web with the javascript sdk. https://developers.facebook.com/docs/facebook-login/web. [2015-12-07].

[3] Javascript promise. https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Promise. [2015-12-07].

[4] React documentation. https://facebook.github.io/react/index.html. [2015-12-07].

[5] Daniel Roy Greenfield and Auderey Roy Greenfield. *Two Scoops of Django: Best Practices for Django 1.8*. Two Scoops Press, 2015.