Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Nový portál Programátorská liaheň

Bakalárska práca

2014

Mária Mrocková

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Nový portál Programátorská liaheň

Bakalárska práca

Študijný program: Informatika
Študijný odbor: 2508 Informatika
Katedra: FMFI.KI - Katedra informatiky
Vedúci: RNDr. Michal Forišek, PhD.

Bratislava, 2014
Mária Mrocková

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Mária Mrocková
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: Nový portál Programátorská liaheň

Cieľ: Cieľom práce je pomocou moderných technológií od základu vyvinúť novú verziu portálu Programátorská liaheň. Portál slúži na samostatnú výučbu riešenia algoritmických úloh. Súčasťou práce je aj návrh úvodných sád úloh, ich príprava a integrácia do portálu.

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

Dátum zadania: 21.03.2014

Dátum schválenia: 26.03.2014 doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Venovanie

Túto prácu venujem svojmu krstnému synovi Jankovi pri príležitosti jeho prvých odlezených krôčkov.

Poďakovanie

Touto cestou by som chcela úprimne poďakovať svojmu vedúcemu RNDr. Michalovi Foriškovi, PhD. za odborné usmernenie a za ochotu konzultovať v ľubovoľnej časti dňa.

Ďalej ďakujem svojmu priateľovi, rodine a kamarátom za podporu, ktorú som od nich celý čas cítila.

Špeciálne poďakovanie patrí mojim súrodencom, ktorí ma v čase intenzívneho písania trpezlivo zobúdzali.

Abstrakt

V práci predstavujeme novú implementáciu portálu Programátorská liaheň. Ide o webový projekt, ktorý sa zameriava na riešenie algoritmických úloh systémom samoštúdia. Zahŕňa úlohy, ktoré sú tematicky rozčlenené do sád a majú určenú svoju nadväznosť. Využívame princíp automatického testovania riešení na vopred pripravených, ale nezverejnených vstupoch. Liaheň poskytuje vzorové riešenia úloh a študijné materiály. Projekt obsahuje aj administrátorské rozhranie pre pohodlné pridávanie a upravovanie úloh. Používame jazyk Python s frameworkom Django.

Kľúčové slová: výučba programovania, algoritmické úlohy, automatické testovanie, Django

Abstract

In this work, we introduce the new implementation of internet portal Programming hatchery (Programátorská liaheň) which offers self-study system for improving in solving programming tasks with algorithmic nature. It contains several tasks which are divided into several sets (usually one set contains tasks with similar topic) and have recommended solving order. We test solutions automatically on several inputs which are prepared in advance but are not public. Hatchery also contains example solutions and texts to study. Project also has admin interface for convenient adding and editing tasks. Project is written in Python using Django framework.

Key words: teaching of programming, algorithmic tasks, automatic evaluation, Django

Obsah

Úvod

Programátorská liaheň je úplne nová verzia rovnomenného portálu. Jej cieľom je precvičovanie riešenia algoritmických úloh, ako aj získavanie nových poznatkov zo študijných materiálov.

Už prešlo 10 rokov odvtedy, keď ju Michal Forišek, vedúci tejto práce, spolu s ďalšími organizátormi Korešpondenčného seminára z programovania (KSP) vymysleli a vytvorili. Ja sama mám na ňu veľmi pekné spomienky, keďže nejednu noc som počas strednej školy strávila riešením úloh z Liahne.

O novej a lepšej verzii sa v komunite KSP uvažovalo už dlhší čas. Preto som sa rozhodla, že vrámci svojej bakalárskej práce zhrniem naše doterajšie skúsenosti a položím základy novej Liahne. Dôležité je, aby bola napísaná prehľadne a ľahko rozšíriteľná o nové nápady.

Súčasný stav podobných projektov

Vo svete už existujú projekty so zameraním na algoritmické riešenie problémov. Naši susedia v Českej republike majú projekt *Programátorská džungle*[1], „v němž si mohou změřit své síly i ti, kdož neoplývají nadšením pro řešení těžkých teoretických úloh“. Vo svete je známy americký *USACO Training Program*[2], ktorý je úzko spojený s americkou aj medzinárodnou olympiádou v informatike (USACO a IOI). Výsledkom americko–ruskej spolupráce je pomerne významná platforma *Rosalind*[3], ktorá sa cez riešenie problémov zameriava na výučbu bioinformatiky.

Hoci vo svete nie sme prví, pokiaľ je nám známe, v slovenčine žiadny podobný projekt neexistuje. Vzhľadom na zahraničné programy ostáva Programátorská liaheň špecifická svojou štruktúrou úloh a tiež systémov achievementov, preto sme presvedčení, že naše dielo má zmysel rozvíjať.

[1] `https://ksp.mff.cuni.cz/dzungle/`

[2] `http://cerberus.delos.com:790/usacogate`

[3] `http://rosalind.info/problems/locations/`

Členenie práce

V tejto práci popíšeme novú Liaheň z pohľadu didaktika, používateľa a programátora.

Teórii o zostavovaní a testovaní úloh je venovaná prvá kapitola. V druhej kapitole popisujeme používateľské rozhranie. Najväčšia časť tretej kapitoly je stručným prehľadom filozofie frameworku Django. Najdôležitejšou kapitolou je štvrtá, ktorá slúži ako dokumentácia pre ďalšiu prácu v projekte, pretože v nej podrobne vysvetľujeme programátorské rozhranie.

Kapitola 1

Prehľad problematiky

V Programátorskej liahni je základným prvkom riešenie programátorských úloh. Najdôležitejšie otázky, na ktoré si musíme položiť odpoveď sú:

- Aký typ a postupnosť úloh by sme mali zvoliť?
- Ako hodnotiť riešenia?

V tejto kapitole budú hlavným motívom.

1.1 Analýza možností

1.1.1 Prehľad typov úloh

Zaoberáme sa úlohami so zameraním na algoritmické riešenie problémov. Ragonis [Rag12] rozoznáva pri vyučovaní programovania 12 typov zadaní. Uvádzame ich aj s príkladmi:

Vývoj riešenia Napíš metódu, ktorá dostane číslo n a vráti jeho delitele.

Vývoj riešenia s použitím zadaných modulov Napíš metódu, ktorá na vstupe dostane celé číslo n a vráti celé číslo medzi 1 a n s najväčším počtom deliteľov. Použi funkciu $numberOfDivisors(n)$, ktorá dostane celé číslo a vráti počet jeho deliteľov.

Trasovanie zadaného riešenia Vytvor trasovaciu tabuľku, ktorá znázorňuje beh danej metódy. V tabuľke by mal byť stĺpec pre každú premennú a pre výstup kódu.

Analýza behu programu Máme zadaný kód obsahujúci cyklus. Preskúmaj ho a odpovedz na otázky typu: „Pre ktoré hodnoty x a y sa cyklus vykoná práve raz?“ „Pre ktoré nikdy neskončí?“

Hľadanie významu daného riešenia Zisti, aký problém rieši zadaná metóda.

Skúmanie správnosti zadaného riešenia Máme zadaný zdrojový kód metódy, ktorá rieši zadaný problém. Je toto riešenie správne?

Dopĺňanie zadaného riešenia Máme zadaný zdrojový kód metódy a vieme, aký problém rieši. Niektoré jej časti sú vynechané, doplň ich.

Manipulácia s príkazmi Máme zadaný zdrojový kód metódy, ktorá rieši pomenovaný problém. Odpovedz na otázky a vysvetli odpoveď; otázky sú typu: „Ovplyvní sa správnosť algoritmu, ak odstránime inštrukcie z riadku (4) a obsah dvoch polí použitých v metóde sa vymení vždy?“

Odhad efektívnosti Odhadni časovú aj pamäťovú zložitosť zadaného riešenia. Vysvetli svoju analýzu.

Konštrukcia úloh Vymysli úlohu, ktorej riešenie využíva metódu na nájdenie najčastejšie sa vyskytujúceho prvku v poli. Taká úloha by mohla byť napr.: Napíš metódu, ktorá dostane na vstupe známky z písomky zo slovenčiny od každej triedy v maturitnom ročníku školy a vypíše najčastejšiu známku.

Úlohy týkajúce sa programátorského štýlu Pozri si zadané viaceré správne riešenia rovnakého problému. Ktoré je podľa teba najlepšie? Prečo?

Transformácia riešenia z jednej reprezentácie do druhej Zadaná metóda je implementovaná využívajúc rekurziu. Prepíš ju pomocou $while - cyklu$ tak, aby robila to isté.

1.1.2 Prehľad systémov hodnotenia

Navrhnúť systém hodnotenia algoritmických úloh ani zďaleka nie je tak ľahký problém, ako by sa mohlo zdať. Forišek [For06] spomína napr. tieto prístupy:

Čierna krabička (Black-box testing) Plne automatický spôsob. Riešenie sa automaticky skompiluje a spravidla niekoľko krát sa spustí. Na testovacom serveri sú pripravené vstupné dáta a checker (*kontrolór*). Ak sú výstupné dáta jednoznačné, môžu nahrádzať jednoduchý checker. Každému zo vstupov prislúcha istý počet bodov a tie sa udeľujú za správne vyriešenie úlohy v danom časom i pamäťovom limite. Výsledný počet bodov sa sčíta za každý vstup.

Rozšírenie čiernej krabičky (More extensive black-box testing) V predchádzajúcom prístupe by sa mohlo stať, že nekorektné riešenie, ktoré generuje náhodné výstupy získa neúmerné množstvo bodov. Prevenciu je taký prístup k testovaniu,

ktorý vstupy zoskupuje do sád a body za sadu udeľuje len v prípade, že je celá správne vyriešená.

Pero a papier (Pen-and-paper evaluation) Celé riešenie sa píše na papier bez prístupu k počítaču. Vyhodnocujú ho živí ľudia, ideálne s bohatými skúsenosťami.

Dôkaz správnosti (Supplying a proof) Okrem zdrojového kódu sa vyžaduje aj formálny dôkaz správnosti riešenia.

Recenzia kódu - biela krabička (Code review – white-box testing) Riešitelia si kontrolujú svoje kódy navzájom a úlohou je nájsť vstup, kde algoritmus nefunguje. Tento prístup je ukážkou, že často býva ľahšie dokázať nekorektnosť algoritmu ako naopak.

Vopred zadané vstupy (Open-data tasks) Testovacie vstupy sú známe a cieľom je pre každý z nich vyrobiť správny výstup. Nehodnotí sa spôsob riešenia, len korektnosť výstupu.

1.1.3 Vzdelávacie paradigmy

Na popis edukačného prostredia sa okrem iného používa aj klasifikácia podľa vzdelávacej paradigmy, teda istého súboru princípov. Podľa Kemmisa a spol. [KWA77] rozlišujeme:

Inštruktívnu paradigmu (Instructional paradigm) Systém má v sebe zahrnuté informácie, ktoré postupne predkladá používateľovi na základe doterajších pokrokov. Každá časť obsahuje úlohu po ktorej vyriešení sa odkryje ďalšia sada informácií a úloh. Systém plní úlohu trpezlivého kontrolóra, ktorý na rozdiel od živého človeka nikdy nie je nervózny.

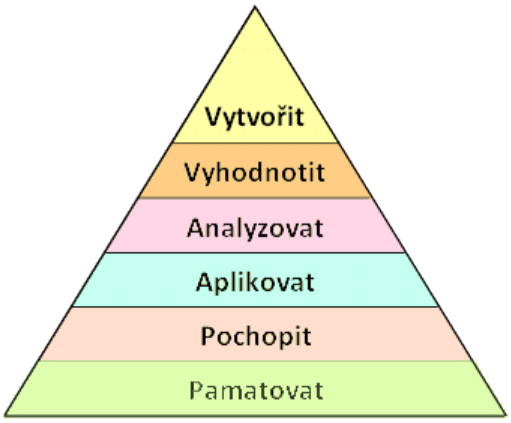
Paradigmu hypotéz (Conjectural paradigm) Táto paradigma zahŕňa prostredie, v ktorom si užívateľ môže overiť svoju hypotézu, v našom prípade poskytuje možnosť kontroly správnosti riešenia príkladu.

Objaviteľskú paradigmu (Revelatory paradigm) Užívateľ skúma predložený model či simuláciu, zatiaľ čo ich konštrukcia ostáva skrytá a odkrýva sa až postupne.

Oslobodzujúcu paradigmu (Emancipatory paradigm) Snaží sa obmedziť pracovné úsilie spojené so získavaním informácií, ktoré samo o sebe nemá vzdelávaciu hodnotu. Patria sem napr. programy na spracovanie informácií.

1.1.4 Bloomova taxonómia

Návrh systému úloh môžeme skúmať aj z hľadiska novej Bloomovej taxonómie [AKB01], ktorá rozlišuje intelektuálne úrovne v procese učenia. Zvyknú sa znázorňovať ako pyramída (obr. 1.1), zdôrazňujúc nutnosť postupu zdola nahor. Na najnižšej úrovni sa nachádza sloveso *Pamätať*, zatiaľ čo pre najvyššiu úroveň je charakteristické sloveso *Vytvoriť*.



Obr. 1.1: Bloomova taxonómia výukových aktivít. *Zdroj:* `http://spomocnik.rvp.cz/clanek/12573/`

1.1.5 Postupnosť krokov

Okrem typu úloh je dôležité ich vhodné zaradenie. Spomeňme dve hľadiská:

- **Postup od konkrétneho k všeobecnému** vs. **Postup od všeobecného ku konkrétnemu** – Prvý prístup spočíva v tom, že riešiteľ najprv nazbiera konkrétne skúsenosti, ktoré sa neskôr „upracú“ do všeobecného poznatku. V druhom prípade je najprv vysvetlená všeobecná teória a tá sa následne dopĺňa konkrétnymi príkladmi.
- **Lineárny program** vs. **Vetvený program** – V lineárnom programe sú úlohy predkladané podľa dopredu určeného poradia. Vetvený program umožňuje istú mieru rozhodovania; niekedy poskytuje riešiteľovi možnosť výberu úlohy.

1.2 Náš prístup

My pracujeme na báze *inštruktívnej paradigmy* a *paradigmy hypotéz*. Naše úlohy sú cielene navrhnuté tak, aby sa náročnosť stupňovala. Okrem toho ich riešiteľ nevidí

všetky naraz, ale postupne, po vyriešení predchádzajúcich úloh sa mu odkrývajú nové a nové zadania. Riešiteľ si môže overovať správnosť riešenia automatickým testovaním na konkrétnych vstupoch. Po vyriešení úlohy sa mu navyše sprístupní naše vzorové riešenie a niekedy aj nový študijný materiál. Úlohy sú logicky rozčlenené do sád podľa obsahu. Snažíme sa voliť úlohy, ktoré majú viacero možných postupov vedúcich k správnemu riešeniu.

V poradí riešenia úloh ponechávame určitú mieru voľnosti. Ak sa dá, ponúkame možnosť *vetvenia*. Sme presvedčení, že v Liahni tento spôsob motivuje riešiteľa oveľa viac ako lineárne prechádzanie úlohami. V prvom rade mu dodáva pocit miernej kontroly nad situáciou. V druhom rade pôsobí ako prevencia proti „uviaznutie“ na jednej úlohe, ktorú sa mu nedarí vyriešiť. Zaseknutie v jednom bode by dokonca nemuselo byť chybou riešiteľa, ale didakticky nevhodným zaradením úlohy.

Pri zostavovaní poradia úloh do Liahne je dôležité, aby zadania postupovali *od konkrétnych k všeobecným*. Vychádzame pri tom z našich praktických skúseností s učením.

Z pohľadu Lomovej taxonómie kombinujeme vrstvy vysokej aj nízkej úrovne. V Liahni získava riešiteľ nové vedomosti primárne týmito spôsobmi:

1. Riešením úloh
2. Skúmaním vzorového riešenia
3. Porovnávaním svojho riešenia s ostatnými riešiteľmi
4. Štúdiom učebných materiálov

1.2.1 Submitovateľné úlohy

V Liahni sa primárne zameriavame na tzv. *submitovateľné úlohy (z angl. submit = odovzdať)*, kde treba *vymyslieť a implementovať riešenie*. Z hľadiska Bloomovej taxonómie sú na najvyššej úrovni. Testujeme štýlom *black-box*: automaticky, na pripravených vstupných dátach, za prítomnosti checkera. Na rozdiel od „ručnej“ kontroly sa nemôže stať, že opravovateľ hodnotí zaujato. Na druhej strane je nutné zdôrazniť, že automatickým testovaním nedokážeme korektnosť riešenia zaručiť. My ju iba predpokladáme na základe dostatočne veľkého počtu pripravených vstupov. Tie testujú všetky nám známe okrajové a potencionálne problémové prípady. Dôraz kladieme na korektnosť aj efektívnosť riešenia.

Korektnosť riešenia

Riešiteľov vedieme v prvom rade k písaniu *správnych* riešení, preto vyžadujeme *správnu odpoveď na všetkých vstupných dátach*. Tento postup sa nám osvedčil aj z praxe. Už sme spomenuli, že testovanie správnosti je samé o sebe ťažkým problémom. Aj v prípade, že požadujeme korektnosť programu pre všetky vstupy by sa mohlo stať, že „omylom“ na základe „slabých“ testovacích vstupov uznáme nesprávne riešenie za správne. O čo nespravodlivejší by bol systém v prípade, že by sme udeľovali čiastkové body čiastočne fungujúcim úlohám. V niektorých úlohách by totiž napr. mohlo „náhodné“ riešenie získať neúmerne veľa bodov.

Efektívnosť riešenia

Správne riešenie neimplikuje optimálne riešenie. Efektívnosť riešenia posudzujeme postupne podľa:

- Časovej zložitosti
- Pamäťovej zložitosti

Každá úloha má určený časový limit, po ktorom sa ukončí testovanie programu. Definovaný je aj pamäťový limit, ktorý sa nesmie presiahnuť.

Nároky na efektívnosť riešenia problému môžeme rozdeliť na tri stupne. Podľa nich stačí implementovať:

Ľubovoľné správne riešenie Tu vyžadujeme jedine správnosť výsledku, hoci by sa počítal veľmi neefektívne. Napr. úloha o triedení porovnávaním by mohla byť implementovaná hoci aj skúšaním všetkých možností usporiadania.

V praxi dostatočne efektívne riešenie Odovzdané riešenie nemusí byť najlepšie, no v praxi stále rozumne použiteľné. Pokračujúc v príklade by sme mohli uznať aj kvadratické riešenie

Optimálne riešenie Vzorové a odovzdané riešenie musia patriť do rovnakej triedy zložitostí. V spomínanej úlohe by sme uznali jedine riešenie s pamäťovou zložitosťou $O(N \lg N)$

V našich úlohách nároky na efektívnosť stupňujeme s pokrokmi riešiteľa. V úvodných úlohách sú minimálne, pretože prvoradá je správnosť. Týmto spôsobom nepriamo učíme riešiteľa hneď od prvých úloh rozlišovať priority.

Optimálne riešenie mnohokrát nie je triviálne ani pre skúseného programátora, tobôž pre začiatočníka. Pre nás je dôležité, aby na základe svojich doterajších vedomostí

sám vymyslel funkčné riešenie. Po vyriešení úlohy sa dozvie optimálne riešenie a odvtedy v ho v podobných úlohách už budeme vyžadovať.

Čo sa týka časových zložitostí riešení, uvedomujeme si, že v praxi sa na testovacích dátach ťažko odhaduje presná zložitosť. Zoberme si len rozdiely medzi programovacími jazykmi, okrem toho je napr. ťažké odlíšiť konštantný faktor od logaritmického. Konkrétny časový limit býva zväčša dvoj– až trojnásobok reálneho času, ktorý potrebuje naše vzorové riešenie.

Na pamäťovú zložitosť príliš nedbáme, stanovujeme dosť veľký konštantný limit (v súčasnosti už povoľujeme $1GB$). Zvedavý čitateľ si iste pokladá otázku, prečo je to tak. Nuž, aby riešenie spotrebovalo veľa pamäte, musí ju aj naplniť hodnotami. Väčšinou prekročeniu pamäťového limitu predchádza prekročenie limitu časového.

1.2.2 Vzorové riešenie

Sme presvedčení, že pre riešiteľov rast je ukážka vzorového riešenia nesmierne užitočná. Kľúčom k úspechu je vzbudiť oň záujem. Je dosť možné, že sa nad riešením úlohy dlho trápil a po veľkom množstve neúspechov sa mu ju konečne podarilo vyriešiť. Presne v tomto momente mu ukážeme vzorové riešenie. Často sa stáva, že je efektívnejšie a trikové. Naším cieľom je, aby mohol v budúcnosti *aplikovať* toto riešenie v podobných úlohách. Je dôležité, aby sme ho poskytli ihneď po vyriešení úlohy, kým si zreteľne pamätá zadanie aj svoj postup riešenia. Z psychologického hľadiska nám pomáhajú aj emócie sprevádzané úspešným submitom. Je známe, že emócie pomáhajú lepšiemu zapamätaniu si.[1]

1.2.3 Študijné materiály

Schopnosť samoštúdia je ďalšou oblasťou, ktorú Liaheň rozvíja. Okrem submitovateľných úloh rozlišujeme ešte jeden typ: *študijné úlohy*[2]. Tie pozostávajú z multimediálneho obsahu a ich cieľom je zrozumiteľne podať riešiteľovi nové informácie v správnom čase. Považujeme za dôležité, aby boli predkladané postupne, v závislosti od úrovne riešiteľa. Snažíme sa, aby nové učivo najprv *pochopil* a následne *aplikoval* v nových úlohách. Okrem toho, opäť v ňom chceme vzbudiť záujem o danú problematiku. Pred každú študijnú úlohu sa snažíme zaradiť aspoň jednu submitovateľnú, ktorá sa problematiky dotýka, a na jej riešenie postačujú doterajšie poznatky. *Po* nazbieraní *konkrétnych skúseností* môžeme v budúcnosti vytvoriť *všeobecný poznatok*.

[1] Odhliadnuc od extrémnych prípadov: ak by bola emócia príliš silná, môže zapamätanie dokonca potlačiť. Takéto prípady ale v Liahni nepredpokladáme.

[2] Teda aj čítanie študijných materiálov budeme v ďalšom texte považovať za riešenie úlohy.

1.2.4 Porovnávanie riešení

Všetci úspešní riešitelia úlohy majú prístup k svojim zdrojovým kódom navzájom. Tento krok považujeme za významný, pretože rozširuje už popísané výhody vzorového riešenia. Hlavným cieľom je poskytnúť materiál na *analýzu*. Cudzie riešenie môže priniesť inšpiráciu pre vlastné. Po čase si napr. riešiteľ môže obľúbiť štýl iného riešiteľa a cielene analyzovať jeho kódy.

1.2.5 Ukážka zadania úlohy

Uvedieme úlohu, ktorá má triviálne riešenie, no z pohľadu učiaceho procesu je veľmi dôležitá. S menšími úpravami sme ju prevzali z pôvodnej Liahne, kde bola zaradená ako úplne prvá submitovateľná úloha.

Nasleduje zadanie úlohy:

Popolvár najväčší na svete

Pred kráľom stoja v rade siedmi mládenci. Kráľ by rád spoznal, ktorý z nich je hrdina Popolvár. To nie je až taká ťažká úloha, keďže o Popolvárovi sa vie, že je na svete najväčší.

Zadanie

Napíš program, ktorý načíta výšky mládencov a rozhodne, ktorý z nich je Popolvár.

Formát vstupu

Vstup obsahuje práve sedem riadkov. V každom riadku je jedno prirodzené číslo, menšie ako 3000 – výška jedného z mládencov v milimetroch. Tvoj program by mal tieto čísla načítavať zo štandardného vstupu („z klávesnice“), použi teda štandardné funkcie na načítanie – `readln`, `scanf`, `cin`, a pod.

Formát výstupu

Tvoj program by mal vypísať jediný riadok a v ňom jediné celé číslo – Popolvárovu výšku v milimetroch. Tvoj program by mal toto číslo vypísať na štandardný výstup („na obrazovku“). Použi teda štandardné funkcie na výpis – `writeln`, `printf`, `cout`, a pod. Nevypisuj nič iné, iba toto jedno číslo (a znak konca riadku).

Príklad

Vstup	Výstup
1880	2143
2143	
1722	
2080	
1929	
1789	
2142	

⋆ ⋆ ⋆

Vstup zámerne *nie je zadaný všeobecne, ale konkrétne*. Ukazuje sa totiž, že pochopiť princíp opakovania s dopredu neurčeným počtom opakovaní je ťažké a musia mu predchádzať úlohy s konkrétnym počtom opakovaní.

K správnemu riešeniu vedie viacero ciest. Riešiteľ – začiatočník pravdepodobne použije copy-paste a konkrétne podmienkové príkazy. V úvodnom štádiu učiaceho procesu je však aj takéto riešenie výborné, pretože je správne.

Kapitola 2

Používateľské rozhranie

Z viacerých dôvodov je Liaheň realizovaná ako webový portál. Na účely testovania riešení potrebujeme pripojenie na internet. Pridávanie nových príkladov a achievementov[1] je jednoduchšie cez centrálnu databázu ako vydávaním nových verzií projektu. Tiež chceme podporovať aspoň minimálnu interakciu medzi užívateľmi. Výhodou je, že nie je potrebná žiadna inštalácia. Vyhýbame sa aj problémom s kompatibilitou na rôznych platformách.

V tejto kapitole popíšeme tú časť Liahne, s ktorou užívateľ priamo komunikuje. Predstavíme GUI, vysvetlíme používateľské role a v závere uvedieme graf užívateľského pohybu po stránkach.

2.1 GUI

Naše GUI *(angl. Graphical User Interface = grafické používateľské rozhranie)* je navrhnuté v minimalistickom dizajne, aby sme upriamili pozornosť na podstatné. Používame stĺpcový layout*(usporiadanie)* a tak sa v zobrazovaní obsahu prispôsobujeme rozličným typom zariadení bez nutnosti vývoja odlišných verzií webu.

2.1.1 Úlohy

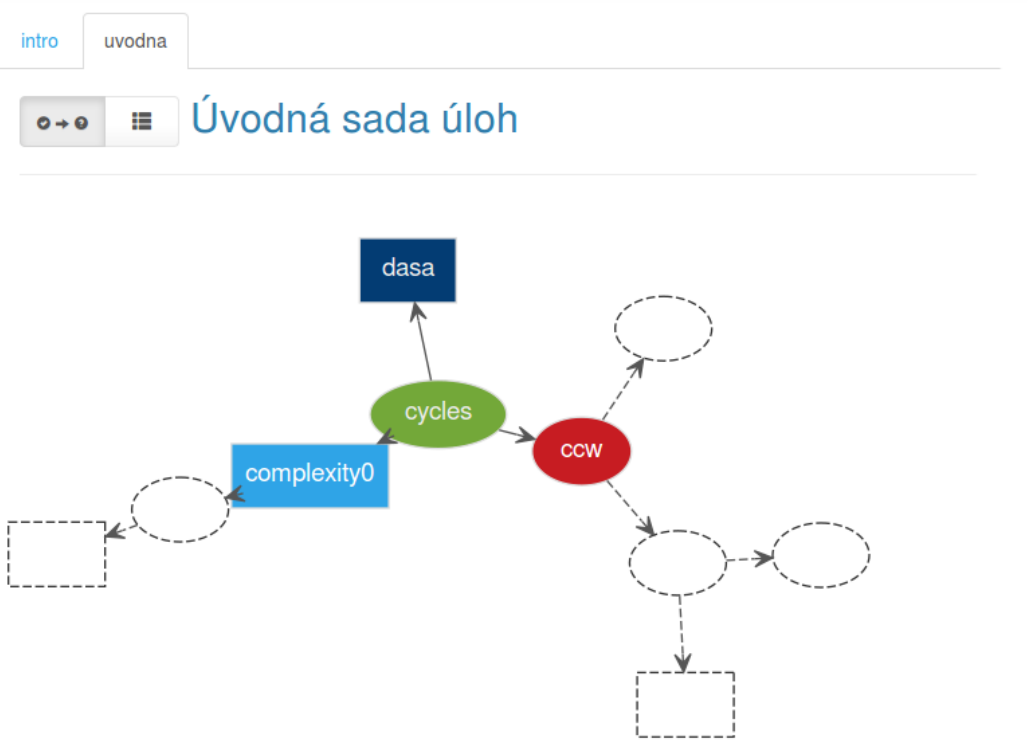
Úlohy sú kvôli prehľadnosti rozčlenené do tematických sád. Po kliknutí na položku *[Úlohy]* v hlavnom menu sa zobrazí tá sada, ktorú užívateľ práve rieši. Zmeniť ju môže pomocou záložiek situovaných v hornej časti stránky. Zobrazenie úloh v sade je dostupné v dvoch režimoch. Prepínajú sa ikonkami v ľavej hornej časti:

Graf Je to vizualizácia orientovaného acyklického grafu *(directed acyclic graph)*. Jeho vrcholy sú názvy úloh a hrany predstavujú závislosti medzi nimi – určujú poradie

[1] Priebežne ich plánujeme dopĺňať.

riešenia. Po vyriešení úlohy sa odkrývajú nové zadania; no už od začiatku sú naznačené všetky úlohy, aj neobjavené. Poskytnutím informácie o ich existencii dávame riešiteľovi prehľad o veľkosti sady. Je pre nás dôležité, aby vedel odhadnúť, v akej fáze jej riešenia sa práve nachádza. Tento spôsob je výborným prehľadom riešiteľovho pokroku v danej sade.

Zoznam Úlohy sú rozdelené do štyroch kategórií: Aktuálne úlohy, Vyriešené úlohy, Prečítané texty, Neprečítané texty. Všetky úlohy jedného typu sú zobrazené spolu. K submitovateľným úlohám, zobrazujeme aj podrobnosti, napr. počet riešiteľov. V tomto štýle nie je potrebné uvádzať existenciu neviditeľných úloh. Kvôli prehľadnosti nezobrazujeme ani kategórie bez úlohy.



Obr. 2.1: Sada úloh zobrazená ako graf. Hranaté vrcholy predstavujú študijné úlohy, oválne zas submitovateľné. Farebne sa odlišujú vyriešené a aktuálne úlohy oboch typov. V celom portáli máme definovaných niekoľko farebných konštánt, ktoré používame podľa ich účelu. Farby grafu sú zámerne zvolené podľa tejto schémy, a tak celkový dizajn pôsobí celistvejšie. O neviditeľných úlohách riešiteľ zatiaľ nevie podrobnosti.

Zadanie úlohy

V oboch štýloch zobrazenia sa k zadaniu konkrétnej úlohy dostaneme veľmi jednoducho: kliknutím na linku s názvom sa ukáže text zadania.

Pri **submitovateľných úlohách** zadanie spravidla pozostáva z:

- motivácie
- samotného zadania
- presného popisu formátu vstupu a výstupu, ako aj obmedzení
- príkladu vstupu a výstupu
- formulára na odovzdanie riešenia
- zoznamu doterajších riešiteľových pokusov v danej úlohe
- záložky s odkazom na vzorové riešenie (iba u vyriešených úloh)

Študijné úlohy tvorí multimediálne spracovaná téma. V súčasnosti využívame iba text, linky a obrázky. Do budúcna uvažujeme aj o videu a animácii.

Riešenie úlohy

Submitovateľné úlohy sú tou zaujímavejšou časťou. Po prečítaní zadania má užívateľ napísať skompilovateľné riešenie v ľubovoľnom programovacom jazyku, ktorý podporuje náš testovač. V súčasnosti sú to: $Pascal, C, C++, Python, Python3, Java, C\#, Haskell$. Program má vypísať pre každý testovací vstup – obmedzený podmienkami zo zadania – správny výsledok. Na prácu s dátami vstupu aj výstupu sa nemá používať súbor, ale štandardný konzolový vstup a výstup.

Nasleduje odovzdanie riešenia[2], čiže upload práve jedného súboru. Ten sa automaticky posiela na testovač, ktorý sa ho pokúsi skompilovať a spustiť na dopredu pripravených vstupných testovacích dátach. Testovač posiela späť finálny výsledok, ako aj podrobný záznam. Zahŕňa čas behu programu i čiastkové výsledky podľa jednotlivých vstupov. Testovač vždy rozhodne o jednom z nasledujúcich výsledkov:

OK Program sa podarilo skompilovať a v stanovenom časovom limite vypísal správny výsledok pre všetky vstupné dáta.

Compilation error Program sa nepodarilo skompilovať. Chyba sa bližšie špecifikuje podrobnom zázname z testovania.

Runtime exception Nastala chyba počas behu programu, môže ísť napr. o prekročenie pamäťového limitu.

[2]V ďalšom texte budeme k slovnému spojeniu *odovzdanie riešenia* často používať výraz *submit* ako synonymum.

Security exception Program chcel vykonať nedovolený príkaz, ako napr. `fork()`.

Time limit exceeded Program bežal dlhšie ako stanovený limit.

Wrong answer Program nefunguje správne: naše a riešiteľove výstupy sa nezhodujú.

Internal error Chyba nie je na strane riešiteľa, ale nášho testovača.

Počet pokusov na submity neobmedzujeme. Dôležité je, aby riešiteľ mohol overovať správnosť svojho riešenia a aby sám dokázal opraviť prípadné chyby.

Študijná úloha sa považuje za vyriešenú ihneď po prečítaní, technicky je to po zobrazení jej obsahu.

2.1.2 Submity

Ďalšou položkou v hlavnom menu sú *[Submity]*. Odkazuje na stránky so zoznamom submitov, ktoré sa dajú filtrovať podľa niekoľkých kritérií. Submity, ku ktorým má užívateľ prístup [3] sa dajú rozkliknúť pre viac detailov. V budúcnosti plánujeme pridať aj štatistiky, napr. priemerný počet submitov potrebný na vyriešenie úlohy.

2.1.3 Achievementy

[Achievementy] (angl. achievement = úspech) sú treťou položkou v hlavnom menu. Táto sekcia je predbežne vymyslená, ale zatiaľ nie je implementovaná. Vo voľnom čase ju máme v úmysle doplniť.

Aby sme užívateľov ešte viac motivovali riešiť úlohy, umožníme im zbierať achievementy. Tie budú mať formu akoby titulov. Niektoré nápady uvádzame v nasledovnom zozname:

- **Sovička** – vyriešiť úlohu medzi 02:47 a 03:42 *hod.*
- **Lingvistik** vyriešiť úlohu v každom jazyku, ktorý testovač podporuje
- **Killing spree** – vyriešiť aspoň 8 úloh v jeden deň
- **Závislák** – vyriešiť 6 dní po sebe po aspoň jednej úlohe
- **Opravár** – do minúty po neúspešnom submite odovzdať správne riešenie

Achievementy sú prvok, ktorý robí Liaheň čímsi príťažlivou a odlišnou od ostatných podobných projektov. Momentálne pred nami stojí veľká výzva, aby sme tento nápad dobre spracovali.

[3] Práva užívateľov vysvetľujeme v časti 2.2.

2.2 Používateľské role

Liaheň používajú tri základné typy užívateľov:

2.2.1 Anonymný návštevník

Anonymný návštevník je pre nás najmenej zaujímavý [4] typ. Je to neprihlásený užívateľ a má prístup len k statickým stránkam popisujúcim Liaheň a k prihlasovaciemu aj registračnému formuláru. Prvý je určený pre užívateľov s existujúcim kontom, pomocou druhého je možné vytvoriť nové konto. Náš prihlasovací systém podporuje aj autentifikáciu použitím konta z inej prihlasovacej služby. V súčasnosti podporujeme: *facebook*, *Google*, *Launchpad*, *GitHub*, *OpenID*.

2.2.2 Riešiteľ (Bežný užívateľ)

Riešiteľ je najpočetnejšie zastúpený typ, cieľová skupina celého portálu. Hlavnou náplňou jeho akcií je riešenie úloh. K zázname o svojich aktivitách má kompletný prístup. O existencii ostatných užívateľov môže vedieť, no nemá o nich úplnú informáciu. Môže si napr. pozerať ich úspešné zdrojové kódy v úlohách, ktoré má aj on sám vyriešené.

2.2.3 Administrátor

Administrátor je užívateľ s neobmedzenými právami. Napĺňa Liaheň obsahom a odpovedá na prípadné otázky riešiteľov. Stará sa aj o technickú stránku celého portálu.

Na pridávanie obsahu môže použiť grafický interface, ktorý je dostupný cez vysúvacie menu umiestnené v pravom hornom rohu. Na testovacie účely má možnosť označiť (novú) úlohu ako neverejnú. V tom prípade uvidí na stránke jej zadanie bez toho, aby sa k nemu mohli dostať riešitelia. Neverejné úlohy sú v jeho profile graficky odlíšené. V momente, keď je s úpravami spokojný, stačí úlohu označiť ako verejnú a automaticky sa sprístupní aj bežným užívateľom.

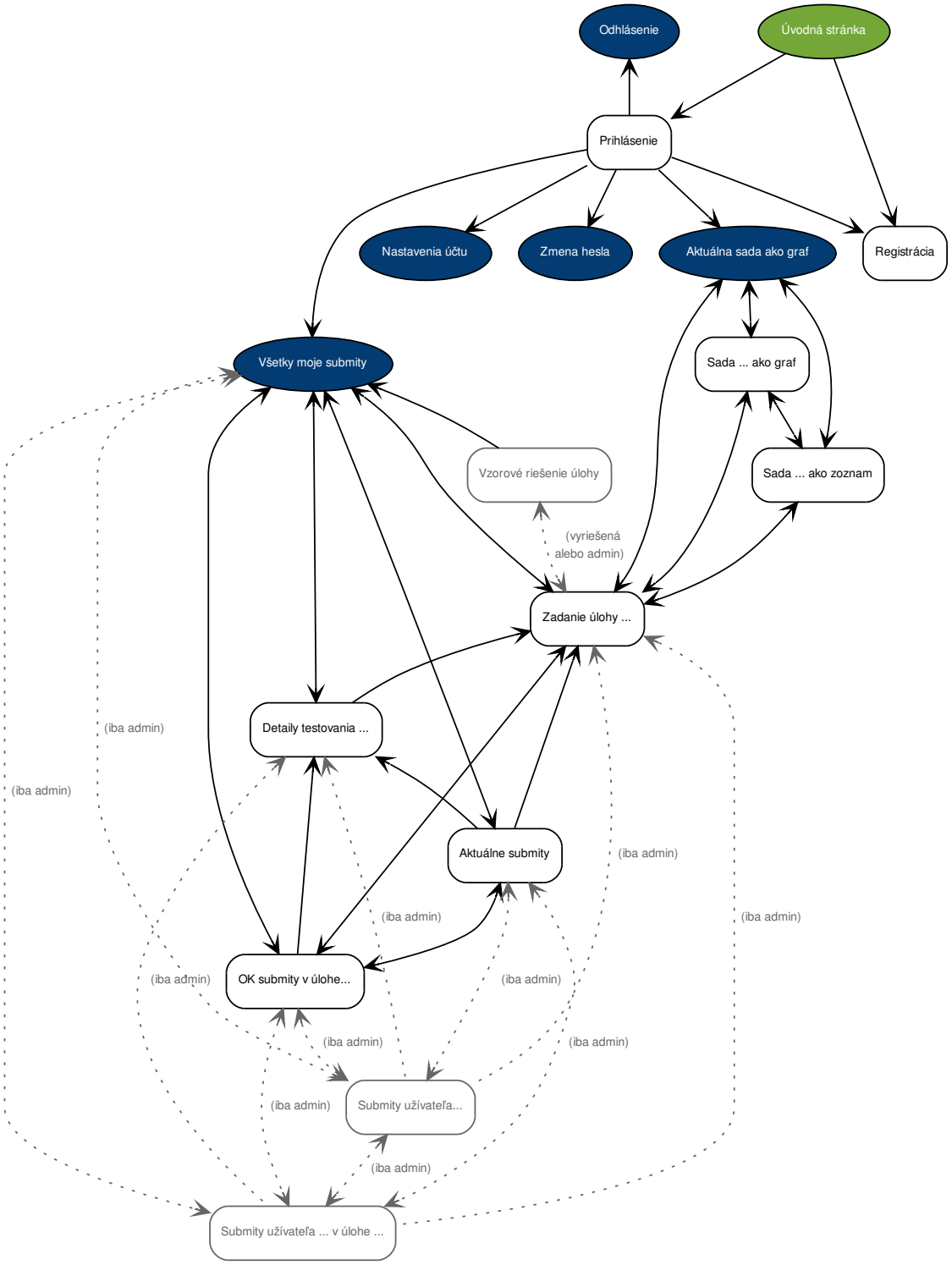
K vzorovým riešeniam má administrátor prístup vždy, hoci by úlohu nemal vyriešenú. Má povolené submitovať ľubovoľnú submitovateľnú úlohu. Vo vizualizácii sady úloh grafovým štýlom sa mu aj v neviditeľných úlohách zobrazujú názvy úloh. Okrem toho má prístup k detailom ľubovoľného submitu, nielen svojho. To platí aj pre úlohy, ktoré sám nemá vyriešené.

[4] Odhliadnuc od možností realizácie v marketingovom smere. O tom, ako zaujať a presvedčiť anonymného návštevníka k registrácii by sa dal spraviť zaujímavý samostatný výskum, čo samozrejme presahuje rámec nielen tejto práce, ale aj nášho študijného odboru.

Administrátorské konto neumožňuje plný zážitok z riešenia a objavovania úloh. V skutočnosti naň ani nie je určené. Má byť pohodlným prostriedkom pre svoje ciele spomenuté vyššie. Obsahuje grafické rozhranie na pridávanie a úpravu úloh. Poskytuje užívateľsky príjemný prístup k všetkým riešeniam pre prípady, keď riešitelia potrebujú radu. Taktiež je priestorom na analýzu náročnosti úloh. Administrátor si môže napr. pozrieť, aké sú najčastejšie chyby v riešeniach konkrétnej úlohy a odvodiť poučenia do budúcnosti. Výhodou je aj možnosť analýzy zaradenia úlohy vzhľadom na poradie ostatných úloh. Pre pohodlnejšiu manipuláciu sú priamo v položke hlavného menu *[Submity]* implementované filtre podľa konkrétneho riešiteľa aj konkrétnej úlohy.

2.3 Pohyb užívateľa po stránkach

Je dôležité, aby štruktúra portálu bola prehľadná a užívateľ si ju ľahko osvojil. Dbáme na to, aby boli jednotlivé časti čo najviac poprepájané. Pomáhame tak užívateľovi rýchlo vyhľadávať informácie. Takisto je dôležité poskytnúť niekoľko „stabilných bodov“. Sú to dôverne známe stránky a dá sa na ne vždy dostať. Sú umiestnené v hornom menu a svoju pozíciu nikdy nemenia. V grafe 2.2 uvádzame podrobnejší prehľad.



Obr. 2.2: Graf znázorňuje možnosti užívateľského pohybu. Na úvodnú stránku sa dá dostať vždy. Na ostatné zvýraznené stránky sa dá dostať vždy po prihlásení. Kvôli prehľadnosti sme nevyznačili všetky z týchto hrán. Vynechané miesta v názvoch značia potrebu konkrétnych hodnôt.

Kapitola 3

Použité technológie a programátorské princípy

V dnešnej dobe nie je umenie vyvinúť web od základu, ale vhodne využiť dostupné frameworky. Nielen, že šetria čas, ale aj „nútia“ programátora písať tak, aby sa práca v budúcnosti ľahko rozšírila, príp. zmenila. V tejto kapitole popíšeme, ktoré frameworky používame my.

3.1 Dynamický a statický web

Liaheň je *dynamický web*, čo znamená, že sa svojím obsahom aj vzhľadom dokáže prispôsobiť každému užívateľovi osobite. Dynamickosť stránky sa zabezpečuje dvomi spôsobmi:

Vykonávaním skriptu na strane servera Stránka sa generuje na serveri. Skript dokáže komunikovať s databázou. Dokáže z nej čítať: na základe dostupných informácií o užívateľovi môže vyprodukovať obsah „na mieru“. Dokáže do nej aj zapisovať: napr. dáta získané od užívateľa z formulára.

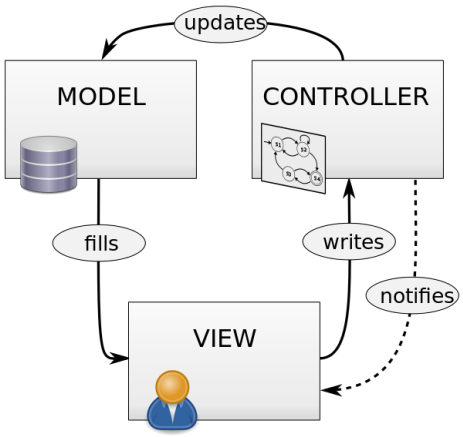
Vykonávaním skriptu na strane užívateľa Vykonávanie prebieha v užívateľovom prehliadači. Interaktívne prvky založené na vstupe z klávesnice alebo pohybe myšky by sa totiž ťažko vyhodnocovali na strane servera. Nechceme, aby sa udalosť na stránke stala po tom, ako niekto pohne myškou servera, ale po udalosti na užívateľskom zariadení. Môže ísť napr. o „vysúvacie“ menu.

My na strane servera používame jazyk *Python* s frameworkom *Django*. Na interaktívne aj statické prvky dizajnu používame framework *Bootstrap*, ktorý vyžaduje dokumenty v *HTML5*.

3.2 Django

Django [HWKM+] je open-source framework pre webové aplikácie napísaný v jazyku Python. V súčasnosti existujú aj iné podobné frameworky, ako napr. Ruby on Rails[1]. Cieľom Djanga je uľahčiť vývoj väčších aplikácií, aby boli znovupoužiteľné, ľahko udržiavateľné a ľahko rozšíriteľné. Dôležitým princípom je heslo „DRY“ (*z angl. Don't Repeat Yourself = neopakuj sa*). Jeho architektúra je navrhnutá v duchu návrhového vzoru MVC (obr. 3.1).

3.2.1 Návrhový vzor MVC



Obr. 3.1: MVC. *Zdroj:* `http://cs.wikipedia.org/wiki/Model-view-controller`

Základnou myšlienkou návrhového vzoru MVC *(z angl. Model-View-Controller = model-pohľad-riadič)* je rozdeliť aplikáciu na tri nezávislé vrstvy tak, aby zmena jednej – v ideálnom prípade vôbec – neovplyvnila inú. Tieto vrstvy sú:

Model Reprezentuje dátový model aplikácie, zahŕňa informácie, s ktorými sa narába.

View Dáta z modelu spracúva do podoby vhodnej pre užívateľa. Ide teda o užívateľské rozhranie aplikácie.

Controller Zabezpečuje riadiacu logiku aplikácie. Reaguje na udalosti vyvolané užívateľom a následne sa stará o zmenu pohľadu, príp. modelu.

[1] `http://rubyonrails.org/`

3.2.2 Štruktúra projektu v Django

Projekty a aplikácie

V Djangu je z hľadiska adresárovej štruktúry na najvyššej úrovni projekt. Ten môže zahŕňať viacero aplikácií. Jedna aplikácia môže byť súčasťou viacerých projektov. Projekt je zbierkou konfigurácií a aplikácii pre nejakú webovú stránku a aplikácia býva väčšinou menšia ucelená časť projektu.

Dôležité súbory a adresáre v aplikácii

Uvádzame prehľad najdôležitejších typov súborov a adresárov, s ktorými budeme v ďalšej kapitole pracovať:

urls.py *(z angl. Uniform Resource Locator = jednotný vyhľadávač zdrojov)* Keď užívateľ zadá do internetového prehliadača adresu, skúsi sa namapovať na niektorú z url definovaných v tomto súbore. V úspešnom prípade sa zavolá priradená funkcia z views.py, inak sa vyhlási chyba. Url sú zadané regulárnym výrazom, a teda môžu obsahovať aj parametre. Tie sú potom vstupom pre zavolanú view–funkciu.

views.py Obsahuje funkcie, ktoré sa volajú z urls.py. Vracajú požiadavku na vygenerovanie webového obsahu. Najčastejšie ňou môže byť render stránky cez zadaný template alebo chybová hláška (napr. HTTP 404).

models.py Definuje dátový model pomocou tried. Ten slúži ako podklad pre štruktúru databázy aj pre automatické vytvorenie administrátorského prostredia.

templates *(z angl. template = šablóna)* Tento adresár obsahuje súbory s príponou .html. Sem sa píše, ako sa majú stránky užívateľom zobraziť. V tejto vrstve sa už nemajú dáta spracúvať (napr. volanie do databázy apod.), iba vypisovať. Django má svoj templatovací systém, v ktorom je zabudovaných dosť veľa filtrov a tagov. Síce vnášajú dynamickosť do statického HTML, no len do takej miery, ako je na výpis dát potrebné.

templatetags Templatetags sú veľmi užitočnou pomôckou. Niekedy sa totiž stane, že by sme chceli používať vlastný tag. Tvorcovia Djanga mysleli aj na to a dávajú nám priestor na dodefinovanie.

3.3 Bootstrap

Bootstrap [OT+] je open-source framework pre vývoj webového frontendu. Jeho výhodou je, že je navrhnutý na podporu responzívneho dizajnu. Je výhodný pre zobrazovanie

stránok na rôznych zariadeniach: od mobilných telefónov až po monitory s veľkým rozlíšením. Pomocou CSS a JavaScriptu má definované štýly pre zobrazovanie tried a tak ich stačí pri písaní HTML vhodne používať.

Kapitola 4

Implementácia

V tejto kapitole podrobne popíšeme štruktúru aplikácie z pohľadu programátora.

4.1 Štruktúra projektu

4.1.1 Adresárová štruktúra

Najprv uvedieme prehľad štruktúry adresárov a súborov a následne vysvetlíme ich funkciu. V nasledujúcom zozname sú kvôli prehľadnosti iba najdôležitejšie dokumenty. Zvýraznené položky sú adresáre, ostatné sú súbory:

```
nova-liahen/
  |--about/
  |--page/
  |--static/
  |--submit/
  |  |--forms.py
  |  |--helpers.py
  |  |--submits/
  |  |  |--janko14/
  |  |  |  |--cycles/
  |  |  |  |  |--103.data
  |  |  |  |  |--103.raw
  |  |  |  |--popolvar/
  |  |--models.py
  |  |--templates/
  |  |  |--submit/
  |  |  |  |--judge.html
```

```
|  |  |  |--protocol.html
|  |  |  |--send_solution.html
|  |  |  |--submits_header.html
|  |  |  |--submits_user_task.html
|  |--templatetags/
|  |  |--items_format.py
|  |--urls.py
|  |--views.py
|--tasks/
|  |--models.py
|  |--templates/
|  |  |--tasks/
|  |  |  |--example_solution.html
|  |  |  |--task_header.html
|  |  |  |--task.html
|  |  |  |--task_set_graph.html
|  |  |  |--task_set_header.html
|  |  |  |--task_set.html
|  |--templatetags/
|  |  |--widgets.py
|  |--urls.py
|  |--views.py
|--templates/
```

Backend

Celý projekt sa nachádza v domovskom adresári **nova-liahen**.

Dôležitý adresár je **page**, ktorý zahŕňa kód platný v celom projekte. Súbor **manage.py** je konzolová utilita *(angl. utility = užitočnosť)*, pomocou nej sa okrem iného spúšťa testovací server.

V projekte sa nachádzajú tri adresáre definujúce vlastné aplikácie: tasks, submit, about. Aplikácia **tasks** má na starosti zobrazovanie úloh. Presnejšie, dokáže zobraziť celú sadu úloh ako graf/zoznam, zadania úloh a vzorové riešenia. Dokáže rozlíšiť, ktoré úlohy sa zobrazujú bežnému užívateľovi a ktoré administrátorovi. Aplikácia **submit** obsluhuje testovanie úloh. Dokáže posielať riešenia testovaču, zobrazuje odosielací formulár, detaily z testovania a rôzne tabuľky so zoznamom submitov podľa implementovaných filtrov. Rozlišuje, ktoré submity môže vidieť bežný užívateľ a ktoré administrátor. Aplikácia **about** je statická a zahŕňa všetky informácie z úvodnej stránky Liahne,

ako napr. Kontakt, O nás, Podmienky používania...

Prihlasovanie do Liahne riešime cez externú aplikáciu **ksp-login** [Pet]. Dokáže zaregistrovať, prihlásiť, odhlásiť užívateľa. Prihlásiť sa dá aj cez existujúce sociálne konto. Aplikácia zvládne spojiť viacero prihlasovacích služieb do jedného konta. Ďalej vie zmeniť užívateľské nastavenia a heslo. Administrátorom ponúka odkaz na správu Liahne.

Frontend

Ďalším adresárom je **static** – obsahuje súbory definujúce štýl stránky (.css a .js) a tiež obrázky, ktoré sa používajú v študijných úlohách či zadaniach. So štýlom stránky súvisí aj adresár **templates**, kde sú definované šablóny platné pre celú stránku a tiež šablóny pre administrátorské rozhranie.

4.1.2 Stručný prehľad aplikácií

Pre tých, čo sa ponáhľajú uvádzame ako prvé mapovanie od slovného opisu stránky cez jej url a view–funkciu až k templatu, ktorý generuje výstupné HTML. V ďalšom texte bude celá tabuľka vysvetlená podrobne.

Aplikácia *tasks*

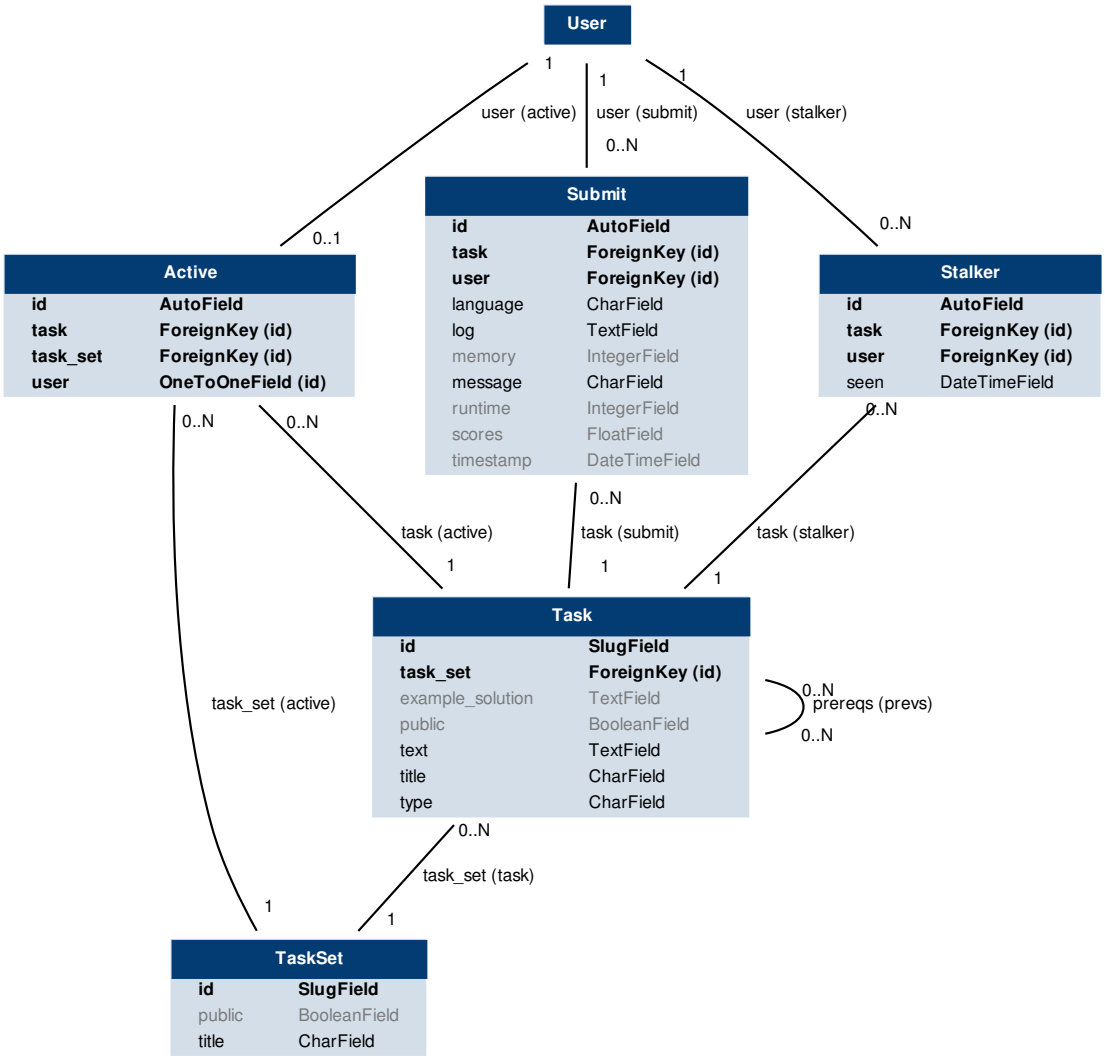
Popis	URL	view	template
Aktuálna sada úloh ako graf	/tasks/	task_set_graph_view	task_set_graph.html
Sada ... ako zoznam	/tasks/set/.../	task_set_view	task_set.html
Sada ... ako graf	/tasks/set/.../graph/	task_set_graph_view	task_set_graph.html
Zadanie úlohy ...	/taks/.../	task_view	task.html
Vzorové riešenie úlohy ...	/tasks/.../ex-sol	example_solution_view	example_solution.html

Aplikácia *submit*

Popis	URL	view	template
Všetky moje submity	/submits/	judge_view	judge.html
Aktuálne submity	/submits/now/.../	judge_view	judge.html
OK submity v úlohe ...	/submits/task/.../	judge_view	judge.html
Submity užívateľa ...	/submits/user/.../	judge_view	judge.html
Submity užívateľa ... v úlohe ...	/submits/user/.../task/...	judge_view	judge.html
Detaily testovania ...	/submits/protocol/.../	protocol_view	protocol.html

4.2 Modely

V tejto časti popíšeme vrstvu modelov. Najprv ukážeme entitno-relačný diagram modelov celého projektu. Následne podrobne vysvetlíme, načo slúžia jednotlivé entity a ich konštanty, atribúty, odvodené atribúty[1] a metódy. Entitu User *(angl. user = užívateľ)* z externého balíčka ksp-login podrobnejšie opisovať nebudeme, stačí nám vedieť o jej existencii.



Obr. 4.1: Entitno-relačný diagram zachytáva vzťahy medzi jednotlivými entitami

[1]Používajú sa ako ostatné atribúty. Získavajú sa však ako výsledok pomocnej metódy, pretože je zbytočné pamätať si dáta duplicitne. Táto metóda býva u nás pomenovaná ako názov vlastnosti s prefixom `_get_`.

4.2.1 Modely definované v *tasks*

TaskSet: Definuje sadu úloh.

- Atribúty
  - **id**: krátky identifikátor pozostávajúci z písmen, čísel a spojovacích znakov medzera a podčiarkovník
  - **title**: celý názov s diakritikou aj medzerami
  - **public**: značka, či je úloha verejná; neverejnú vidia iba administrátori
- Metódy
  - **can_see**: Pre zadaného užívateľa určí, či môže zadanú sadu úloh vidieť. Berie do úvahy atribút public a užívateľskú rolu.

Task – Definuje úlohu. Submitovateľné aj čítacie úlohy patria sem, rozdiel medzi nimi je akurát v atribúte type a v tom, či majú mať vyplnený atribút example_solution.

- Konštanty
  - **TASK_TEMPLATE**: preddefinovaný text do administrátorského prostredia na pridávanie atribútu text
  - **EX_SOL_TEMPLATE**: podobne, ale slúži pre atribút example_solution
  - **ID_HELP**: pomocný text s pokynmi pre administrátorov na vypĺňanie atribútu id
  - **TEXT_HELP**: podobne, pre atribút text
  - **EX_SOL_HELP**: podobne, pre atribút example_solution
  - **SUBMIT, READ, TYPES**: definujú možnosti výberu pre atribút type
- Atribúty
  - **id**: rovnaké ako v sade
  - **title**: rovnaké ako v sade
  - **text**: celý text zadania submitovateľnej úlohy alebo náplň čítacej úlohy. Formátuje sa pomocou markdownu[2] a HTML – v tomto poradí
  - **example_solution**: celý text vzorového riešenia úlohy, ak je submitovateľná, inak je tento atribút nevyplnený; formátuje sa rovnako ako text
  - **public**: rovnaké ako v sade

[2] http://daringfireball.net/projects/markdown/

  - **type**: rozhoduje o type úlohy; povolené hodnoty sú SUBMIT a READ
  - **task_set**: sada, do ktorej úloha patrí
  - **timestamps**: cez triedu Stalker zaznamenáva časy zobrazenia zadania
  - **prereqs**: prerekvizity, čiže zoznam úloh, ktoré je potrebné mať vyriešené pred sprístupnením zadania tejto úlohy
- Odvodené atribúty
  - **is_public**: rozhoduje, či je úloha verejná; od atribútu public sa odlišuje tým, že berie do úvahy aj verejnosť sady, v ktorej sa nachádza
  - **rendered_text**: text zadania, ktorý prešiel cez značkovanie markdown
  - **rendered_ex_sol**: to isté pre text vzorového riešenia
  - **num_solvers**: počet riešiteľov; ak niekto vyriešil úlohu viackrát, zaráta sa iba raz
- Metódy
  - **is_solved**: pre zadaného užívateľa zodpovie, či je úloha vyriešená
  - **is_enabled**: pre zadaného užívateľa zodpovie, či má úloha sprístupnené zadanie, teda, či sú vyriešené všetky jej prerekvizity
  - **can_see**: podobné ako v TaskSet, avšak táto metóda si priberá ešte jeden parameter, podľa ktorého sa rozhoduje o kontexte, v ktorom užívateľ úlohu vidieť môže alebo nie. Možnosti sú tri: 't' – či môže vidieť *t*ext zadania, 's' – či môže vidieť vzorové riešenie (*s*olution), 'g' – či môže vedieť o jej existencii na účely zobrazenia v *g*rafe. Do úvahy sa berie aj atribút public a používateľská rola.

Stalker – *(z angl. stalker = stopár)* Slúži na podrobnosti o zobrazení textu zadania. V budúcnosti sa pomocou neho bude dať do Liahne doplniť výpočet štatistík (napr. ako dlho trvá vyriešenie úlohy od prvého zobrazenia zadania). Tiež bude užitočná pri implementácii niektorých achievementov.

- Atribúty
  - **user**: ktorý užívateľ
  - **task**: ktorá úloha
  - **seen**: čas zobrazenia

Active – Rozširuje atribúty entity User. Slúži na to, aby sme vedeli, ktorú sadu užívateľ práve rieši a tú mu zobrazovali defaultne po kliknutí na položku *[Úlohy]* v menu. Tiež máme informáciu o tom, ktorú úlohu rieši a tú mu zobrazujeme

zas vo filtri podľa úlohy v položke menu *[Submity]*. V budúcnosti sa dá rozšíriť o informáciu o defaultnom štýle zobrazovania sady (graf/zoznam.)

- Atribúty
  - **user**: ktorý užívateľ
  - **task_set**: ktorá sada
  - **task**: ktorá úloha

4.2.2 Modely definované v *tasks*

Submit – Všetky potrebné aj doplňujúce informácie o submitoch sa ukladajú práve sem.

Konštanty

- **STATS, OK, QUEUE**: mapovanie krátkych správ z testovača na ľudsky príjemnejšie; hodnoty OK a QUEUE sú prístupné aj ako samostatné konštanty kvôli príjemnejšiemu narábaniu v ďalších vrstvách
- **LANGUAGE_CHOICES**: mapovanie koncoviek programovacích jazykov na ich celé názvy
- **EXTMAPPING**: niektoré jazyky môžu mať viacero koncoviek pre svoj typ súboru, tu sa mapujú na práve jeden

Atribúty

- **user**: ktorý užívateľ
- **task**: ktorá úloha
- **timestamp**: čas odoslania na testovač; vypĺňa sa automaticky
- **message**: správa z testovača celými slovami, nie skratkou. Kým sa čaká na výsledok, je správa totožná s hodnotou definovanou v konštante QUEUE. V súčastnosti je každá správa iba premapovaním podľa konštanty STATS, v budúcnosti by bolo pekné pre neúspešné submity pridať informáciu o prvom problémovom vstupe.
- **scores**: nepovinný atribút pre úlohy, kde sa nejakým spôsobom meria skóre
- **runtime**: čas behu programu
- **memory**: spotrebovaná pamäť
- **language**: jazyk, v ktorom sa submit testoval; ak sa automaticky určil podľa prípony, uloží sa táto nová hodnota

- **log**: podrobnejší záznam z testovača o jednotlivých vstupoch; momentálne sa posiela vo formáte HTML, v budúcnosti to možno pre programátorsky čistejšie spracovanie zmeníme na iný formát

4.3 Url

Naše url sme starostlivo navrhli tak, aby sa riadili niektorými všeobecne zaužívanými pravidlami pre *pekné* url:

- Len pohľadom na url sa dá uhádnuť obsah stránky.
- Každá stránka má práve jednu url a každá url prislúcha k práve jednej stránke.
- Ak obsahuje parametre, nepredávame si ich pomocou znakov `? & =`.
- Parametre nie sú zadané nejasnými číslami, ale ľudsky ľahko čitateľným identifikátorom.
- Nepoužívame žiadne koncovky ako .php, .html apod.

Tieto pravidlá nie sú len na okrasu, ale sú aj veľmi praktické. Naše linky sú samovysvetľujúce, ľahko zdieľateľné, ľahko prístupné, trvalé, nezávislé od súčasne použitých technológií. Užívateľ môže napríklad poslať kamarátovi odkaz na konkrétnu stránku. Nielen, že sa mu zobrazí rovnaká stránka, ale dokonca ešte pred kliknutím už tuší, čoho sa týka. Môže si ju natrvalo uložiť ako záložku v prehliadači. Ak sa časom zmení naša architektúra, linky ostanú použiteľné aj naďalej.

4.4 Views

Naše viewy sa pokúšajú pre zadané vstupné parametre vyrobiť požiadavku do databázy, spracovať dáta a poslať ich templatu. Ak sa im to nepodarí, vyhlásia chybu HTTP 404. Chyby najčastejšie vznikajú ručnou zmenou url zo strany užívateľa. Tie môžu byť iba obyčajným preklepom a vtedy je problémom neexistujúca položka v databáze. Môžu byť ale aj cieleným útokom, pokusom o prístup k nepovoleným dátam. Útočníci by sa mohli snažiť pozrieť si riešenie nevyriešenej úlohy, zadanie neverejnej alebo neobjavenej úlohy, detaily cudzieho submitu... Na všetky tieto prípady sme mysleli.

4.4.1 Views definované v *tasks*

task_set_view: Má na starosti zobrazovanie sady úloh ako zoznam. Ako vstupný parameter si berie id sady. Vyberie všetky úlohy z danej sady a rozdelí ich na

kategórie: aktuálne úlohy, neprečítané texty, vyriešené úlohy, prečítané texty. Tie pošle templatu.

task_set_graph_view: Zobrazuje sadu úloh ako graf[3]. Generuje sa pomocou balíčka *pygraphviz*[4], ktorý tvorí pythonovú medzivrstvu k programu *Graphviz* [RC] bežiacemu v konzole. Graf sa generuje do formátu .svg, ktorý sa pošle do templatu. Schválne sme zvolili vektorový formát miesto rastrového, aby sme sa lepšie prispôsobili prípadnému zväčšovaniu stránky zo strany užívateľa.

task_view: Prestaví aktívnu úlohu užívateľa na túto. Zapamätá si čas zobrazenia. Pomocou triedy submit.forms.TaskSubmitForm generuje políčka pre submitovací formulár. V prípade submitu posiela riešenie na otestovanie a presmerúva na stránku s detailami z testovania. V súvislosti so submitom ošetruje prípadný výskyt technických chýb. Tie môže spôsobiť: neúspešný upload súboru, neschopnosť uhádnuť programovací jazyk podľa koncovky, zlyhanie pripojenia na testovač. Ak sa niektorá z týchto chýb vyskytla, templatu posúva aj informácie o chybe. Z databázy si vypýta zoznam doterajších submitov užívateľa v tejto úlohe a samozrejme text a nadpis zadania. Zisťuje, či je úloha už vyriešená, aby sa v template mohol prípadne objaviť odkaz na vzorové riešenie. Tieto potrebné informácie posiela templatu.

example_solution_view: Vypýta si informácie o vzorovom riešení a posiela ich templatu.

4.4.2 Views definované v *submit*

judge_view: Stará sa o zobrazovanie zoznamu submitov podľa spomínaných filtrov. Typ filtra sa volí podľa povinnej vstupnej premennej type, ďalšie dve vstupné premenné task a user, sú nepovinné a vypĺňajú sa v závislosti od zvoleného filtra. Metóda vyberie správnu množinu submitov z databázy a tú pošle templatu. Zámerne sme nevytvorili pre každý filter novú metódu. Takto sa totiž ľahko pridávajú nové filtre. Stačí dodefinovať požiadavku na databázu pre nový typ a príp. nastaviť zopár pomocných premenných. Po pridaní filtra treba ešte v template dodefinovať nadpis – názov filtra.

protocol_view: Zabezpečuje zobrazenie podrobností z konkrétneho submitu. Dáta si neberie len z databázy, ale aj z disku. V databáze sú uložené atribúty ohľadom

[3]Graf je podrobne popísaný v 2.1.1

[4]`http://pygraphviz.github.io/`

submitu, ktoré sú vysvetlené v modeloch. Na disku je uložený súbor so submitnutým zdrojovým kódom. Ten sa načíta do premennej a spolu s ostatnými podrobnosťami z databázy sa posiela templatu.

4.5 Pomocné metódy

V tejto časti opisujeme iba proces spracovania submitu, no v budúcnosti by sem patril aj proces spracovania achievementov.

4.5.1 Spracovanie submitu

Ako sme už naznačili, testovanie riešení funguje takto:

1. Užívateľ skúša poslať riešenie na náš server, kde beží Liaheň. Ak sa mu to z technických príčin nepodarí, tento krok sa opakuje.

2. Zistíme, v akom jazyky máme testovať. Ak sa nám to nepodarí, vyhlásime chybu a skončíme.

3. Vytvoríme novú inštanciu triedy Submit a uložíme si zatiaľ známe atribúty do databázy. Výsledok testovania je zatiaľ defaultná hodnota, teda Submit.QUEUE. Id protokolu sa udeľuje automaticky.

4. Súbor s riešením si uložíme k sebe na disk do adresára submit/submits/login/-task_id. Súbor pomenujeme ako submit_id.data . Pozorný čitateľ si iste kladie otázku, prečo súbor neukladáme do databázy. Táto voľba nebola jednoznačná, ale nakoniec sme zvážili, že bude praktickejšie, ak budú mať administrátori možnosť prístupu k jednotlivým riešeniam bez nutnosti pracovať s databázou. Toto sa im môže hodiť napr. pre prípady nešťastných riešiteľov, ktorým sa dlho nedarí poslať správne riešenie.

5. Vytvoríme nový súbor, ktorý sa od predchádzajúceho líši tým, že obsahuje navyše hlavičky, ktoré špecifikujú pomocné informácie pre testvovač. Dôležité je najmä to, aby testovač vedel, na ktorých vstupoch má riešenie testovať a tiež v akom programovacom jazyku je riešenie napísané.

6. Tento súbor si opäť uložíme k sebe na disk, do rovnakého adresára ako predchádzajúci súbor. Tento krát sa bude volať submit_id.raw. Opäť by sme sa mohli pýtať, prečo tento krok. Výrobu tohto súboru vieme zrekonštruovať takmer identicky. Výhoda ukladania aj týchto súborov spočíva v tom, že výnimočne sa stane, že chceme submitnuté riešenia opätovne pretestovať. Stáva sa to buď vtedy, ak

zistíme, že sme mali v našich vstupoch chybu alebo vymyslíme nový vstup (napr. ak sa podozrivé riešenie otestuje s výsledkom OK), na ktorom by sme radi dodatočne otestovali riešenia.

7. Pokúsime sa nadviazať spojenie so serverom, na ktorom sa testujú riešenia. Volá sa Experiment a sídli na adrese 10.0.0.3, my sa pokúšame pripojiť na port 12347. Ak sa nám to nepodarí, vyhlásime chybu a skončíme. Experiment je testovač, na ktorom sa testujú riešenia z viacerých programátorských súťaží, spomeňme napr. *Olympiádu v informatike* a *Korešpondenčný seminár z programovania*. Z predmetov vyučovaných na matfyze sú to: *Rýchlostné programovanie*, *Vybrané kapitoly z teoretickej informatiky* a *Algoritmy a dátové štruktúry*. Pôvodná verzia Liahne sa tiež testuje na tomto testovači.

8. Pošleme testovaču dáta z novovzniknutého súboru .raw.

9. Ukončíme spojenie.

10. Keď nám testovač pošle naspäť výsledky z testovania, z databázy si vypýtame inštanciu daného submitu a aktualizujeme jej atribúty. Finálny výsledok nám testovač posiela v skrátenej úspornej forme. Pomocou mapovania definovanom v konštante Submit.STATS si uložíme výsledok ako príjemnejšie ľudsky čitateľný.

11. Dotestovali sme. Hotovo.

Uvedený postup je implementovaný pomocou týchto funkcií:

submits.helpers.get_lang_from_file: Pomocná funkcia, ktorá zisťuje programovací jazyk podľa koncovky.

submits.helpers.process_submit: Dôležitá funkcia, ktorá zabezpečuje väčšinu z popísaného postupu testovania. Na vstupe dostáva odovzdaný súbor, jazyk, ktorý užívateľ označil, úlohu a užívateľa, ktorých sa submit týka. Pomocou vyššie spomenutej funkcie zisťuje jazyk, vytvára záznam v databáze, ukladá odovzdaný súbor na disk, posiela riešenie aj s hlavičkami na testovač.

submit.views.update_submit: Stará sa o upravenie atribútov po odpovedi testovača.

4.6 Templatetags

4.6.1 Templatetags definované v *tasks*

widgets.py: Obsahuje zatiaľ len jednu položku, konkrétne is_false. Znie to neužitočne, no v testovaní hodnoty pomocou tagu {% if %} sa berie do úvahy, či je premenná nastavená a teda bol problém testovať hodnotu False booleanovských premenných. Príklad použitia: `{% if task_set.public|is_false %}`

4.6.2 Templatetags definované v *submit*

items_format.py: Keďže atribúty submity vypisujeme na viacerých miestach a väčšinou ich chceme naformátované, zadefinovali sme si vlastné štýly výpisu jednotlivých atribútov. Napr. úloha sa vypisuje kliknuteľná na zadanie, správa z testovača je farebne odlíšená podľa toho, či je výsledok dobrý (OK), neutrálny (QUEUE) alebo zlý (všetko ostatné), k informáciám o čase behu a spotrebovanej pamäti pridávame jednotky, v ktorých meriame... Filtre sú pomenované ako atribut_format. Príklad použitia v template, ktorým vypíšeme naformátovaný čas submitu: `{{ submit.timestamp|timestamp_format }}`

4.7 Templaty

4.7.1 Templaty definované v *tasks*

task_set.html: Vykresľuje sadu úloh ako zoznam. Rozlišuje 4 kategórie úloh, v každej kategórii vypisuje zoznam úloh zoskupených spolu. Kategórie sú farebne odlíšené. Kvôli prehľadnosti, kategórie bez úloh nevypisuje. K názvu každej kategórie pridáva aj informáciu o počte úloh v nej. K názvom submitovateľných úloh uvádza aj informáciu o počte riešiteľov.

task_set_graph.html: Vykresľuje sadu úloh ako graf. Štruktúru grafu už dostal z viewu. Vrcholy grafu sú pomenované identifikátormi úloh a odkazujú na ich zadania. Farebne aj tvarovo odlišujeme kategórie úloh. Neviditeľným úlohám sme nastavili prázdny label a tiež sme ich graficky odlíšili. Všetko toto už prišlo vygenerované z viewu.

V budúcnosti by sme do tohto templatu chceli doimplementovať interaktívnejšie tooltips k vrcholom, ako sú defaultné. Presnejšie, keďže kvôli väčšej prehľadnosti sú vo vrcholoch grafu uvedené iba identifikátory, občas užívateľovi nemusí byť celkom jasné o ktorú úlohu ide. Ak dlhšie postojí myškou nad vrcholom, objaví

sa mu aj celé zadanie, no toto nie je okamžité. Je to vecou internetového prehliadača a rýchlosť zobrazovania sa nám nepodarilo ovplyvniť. Nové vylepšenie bude pomocou javaScriptu a udalostí myšky zobrazovať celé názvy okamžite po prejdení myšky cez vrchol. V pôvodnej Liahni sa tieto názvy zobrazovali je jednom konštantnom mieste, v tomto prevedení sa budú zobrazovať priamo pri myške. Je to pre používateľa prirodzenejšie.

task.html: Stará sa o vykreslenie zadania úlohy. Zahŕňa aj formulár na odoslanie riešenia a zoznam doterajších submitov užívateľa v tejto úlohe, ktoré sú definované v aplikácii submit. Ak existuje chyba z predchádzajúceho submitu, vypíše ju.

example_solution.html: Vykresľuje vzorové riešenie úlohy.

task_header.html: Pomocný tepmlate, ktorý definuje hlavičku spoločnú pre zobrazovanie zadania aj vzorového riešenia. Obsahuje záložky s odkazmi na zadanie, riešenie a filter úspešných riešiteľov tejto úlohy. Ak úloha ešte nie je vyriešená, záložka s odkazom je iba naznačená (z motivačných dôvodov), ale nedá sa na ňu kliknúť.

task_set_header.html: Pomocný template s hlavičkami pre zobrazovanie sady úloh ako zoznam aj ako graf. Obsahuje zoznam dostupných sád úloh, pomocou ktorého sa dá zmeniť aktuálne riešená sada. Tiež má v sebe tlačítka na prepínanie medzi štýlmi vizualizácie sady úloh.

4.7.2 Templaty definované v *submit*

judge.html: Zobrazuje zoznam submitov podľa filtrov.

protocol.html: Zobrazuje detaily z testovania. Kým sa nedotestovalo, animuje skutočnosť, že treba čakať. Vypisuje aj submitnutý program, ktorému zvýrazňuje syntax pomocou balíčka *google-code-prettify*[5]. Zatiaľ sa výsledky v prípade nedotestovanej úlohy neobnovujú automaticky, v budúcnosti je to na pláne implementovať.

submits_header.html: Pomocný template s hlavičkami pre zobrazovanie filtrov. Sú v ňom definované nadpisy pre všetky implementované typy. Filter podľa úlohy obsahuje aj príjemný vyhľadávací nástroj podľa častí názvov úloh; stačí začať

[5] `http://code.google.com/p/google-code-prettify/`

písať časť názvu, ktorá ani nemusí byť zo začiatku a automaticky sa upravuje ponuka úloh. Tento vyhľadávací nástroj máme z javascriptovej knižnice *selectize*[6].

send_solution.html: Template, ktorý sa volá z aplikácie tasks pri zobrazovaní zadania úlohy. Popisuje formulár na posielanie riešenia.

submits_user_task.html: Takisto sa volá z tasks a popisuje zoznam všetkých doterajších užívateľových submitov.

[6] http://brianreavis.github.io/selectize.js/

Záver

Cieľom práce bolo implementovať novú verziu Programátorskej liahne. V prílohe sme pripojili funkčné a odladené riešenie, ktoré je pripravené do reálnej prevádzky, aby nahradilo starú verziu. Využili sme moderné technológie, ako frameworky Django a Bootstrap. V textovej časti sme analyzovali didaktické pozadie problému zostavovania a testovania algoritmických úloh. Do práce sme začlenili aj dokumentáciu, ktorá slúži ako základ pre ďalší vývoj projektu.

Budúca práca

Počas vývoja projektu sa vynárali nové nápady, ktoré chceme v budúcnosti doimplementovať.

Zaujímavá bude práca na systéme achievementov. Ďalej plánujeme pridať ukážku Liahne pre neprihlásených užívateľov (interaktívne demo), štatistiky o úlohách a riešiteľoch, prostredie pre pridávanie oznamov. Okrem toho chceme pripojiť diskusné fórum, ktoré budu slúžiť najmä pre nejasnosti ohľadom úloh.

Budeme dbať aj o prvky, ktoré zvýšia užívateľský komfort, napr. automatická obnova výsledku submitu hneď po odovzdaní, krajší dizajn grafového zobrazovania úloh (tooltips) alebo možnosť submitovania pomocou funkcií copy&paste či drag&drop.

V neposlednom rade budeme do Liahne pridávať nové úlohy. V submitovateľných úlohách je výzvou didakticky vhodné usporiadanie. V študijných úlohách sme ešte stále nevyužili potenciál moderného webu, ktorý ponúka široké spektrum na spracovanie učebnej látky pomocou videa, animácie, interaktívneho prostredia na simulácie apod.

Literatúra

[AKB01] Lorin W Anderson, David R Krathwohl, and Benjamin Samuel Bloom. *A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives*. Allyn & Bacon, 2001.

[For06] Michal Forišek. On the suitability of programming tasks for automated evaluation. *Informatics in Education–An International Journal*, (Vol 5_1):63–76, 2006.

[HWKM+] Adrian Holovaty, Simon Willison, Jacob Kaplan-Moss, Wilson Miner, et al. Django – The Web framework for perfectionists with deadlines. web: `https://www.djangoproject.com/`.

[KWA77] Stephen Kemmis, Eleanor Wright, and Roderick Atkin. *How Do Students Learn?: Working Papers on Computer Assisted Learning*. Centre for Applied Research in Education, University of East Anglia, 1977.

[OT+] Mark Otto, Jacob Thornton, et al. Bootstrap – The most popular front–end framework for developing responsive, mobile first projects on the web. web: `http://getbootstrap.com/`.

[Pet] Michal Petrucha. ksp_login – A Django app for both traditional and social authentication used by the Slovak Correspondence Seminar in Programming. web: `https://github.com/Koniiiik/ksp_login`.

[Rag12] Noa Ragonis. Type of questions–the case of computer science. *Olympiads in Informatics*, 6, 2012.

[RC] AT&T Labs Research and Contributors. Graphviz – Graph Visualization Software. web: `http://www.graphviz.org/`.