Uporaba sistema za avtomatsko preverjanje nalog Projekt Tomo pri učenju programiranja

Gregor Jerše
UL FRI
Večna pot 113
Ljubljana
gregor.jerse@fri.uni-lj.si

Matija Lokar
UL FMF
Jadranska ulica 19
Ljubljana
matija.lokar@fmf.uni-lj.si

Povzetek
V članku [1] je opisana spletna storitev Projekt Tomo, ki je bila razvita na Fakulteti za matematiko in fiziko Univerze v Ljubljani (UL FMF) kot pomoč pri poučevanju in učenju programiranja že leta 2010. Bila je zelo dobro sprejeta, a nekoliko nerodna za uporabo, predvsem s strani učitelja. Zato smo jo leta 2015 [2] posodobili. Nova različica je bila zasnovana na podlagi večletnih izkušenj, pridobljenih z uporabo prve različice pri poučevanju različnih predmetov. V članku bomo predstavili, kako si lahko učitelji programiranja pomagajo s spletno storitvijo Projekt Tomo. Osredotočili se bomo na pripravo učnih gradiv in spremljanje napredka učencev.

Kategorije in predmetne oznake
D.2.4 [**Software Engineering**]: Software/Program Verification: Validation.

K.3.1 [**Computers and education**]: Computer Uses in Education: Computer-ašisted instruction.

K.3.2 [**Computers and education**]: Computer and Information Science Education: Computer science education.

Splošni pojmi
Algorithms, Management, Experimentation, Languages.

Ključne besede
Poučevanje, programiranje, spletna storitev, spremljanje napredka.

1. Uvod
Poučevanje programiranja je zahteven proces. Ker je programiranje veščina, se jo učenci lahko naučijo le z veliko vaje. O tem pričajo številne raziskave, med drugim [3]. Učitelji morajo pripraviti veliko nalog, jih razdeliti učencem, sproti preverjati njihov napredek in jim po potrebi pomagati. To še posebej pride do izraza pri poučevanju programiranja začetnikov, kjer učenci neizbežno naredijo veliko sintaktičnih in semantičnih napak. Pri odkrivanju sintaktičnih napak ni večjih težav, saj sodobna orodja nudijo kar dovršeno pomoč. Večja težava je s semantičnimi napakami. Če pomoč učitelja ni takoj na voljo, to močno upočasni napredek učencev, saj ne vedo, kako naprej. Pogosto se tudi zgodi, da učenci zaradi odsotnosti pomoči nalogo rešijo narobe ali pomanjkljivo in se tega niti ne zavedajo. Naj navedemo preprost primer. Izkušnje kažejo, da bo pri nalogi »*Sestavi program, ki prebere celo število in izpiše njegovo obratno vrednost.*« večina rešitev podobnih rešitvi na sliki Slika 1.

```
n = int(input('Vnesi celo število:  '))
rec = 1/n
print('Obratna vrednost je ', rec)
```

```
Shell
Python 3.5.1
>>> %Run a.py

  Vnesi celo število:  12
  Obratna vrednost je  0.08333333333333333

>>> %Run a.py

  Vnesi celo število:  5
  Obratna vrednost je  0.2

>>> %Run a.py

  Vnesi celo število:  4
  Obratna vrednost je  0.25
```

Slika 1: Primer nepopolne rešitve

Učenci so svojo rešitev preizkusili, dobili pravilne rezultate za nekaj testnih primerov in zaključili, da je naloga pravilno rešena. Le manjši del se jih bo spomnilo, da je vneseni podatek lahko tudi 0. Tukaj bi moral z ustrezno razlago vskočiti učitelj, ampak ter pouk programiranja običajno poteka v večjih skupinah, njegov takojšnji poseg včasih ni mogoč, včasih pa tudi ne smiseln, saj zagotovo obstaja del učencev, ki se je na to spomnila sama. Tukaj se pokaže prava moč sistemov za avtomatsko ocenjevanje, saj učencem takoj podajo povratno informacijo in jih opomnijo na pozabljen robni primer. Seveda pa je glavni še vedno učitelj: če on ne pripravi kakovostnih testnih primerov (tudi on lahko pozabi na robni primer s številom 0), potem nam takšen sistem prav nič ne pomaga.

Velika večina začetniških napak je preprosto rešljivih, le učence je treba usmeriti v pravo smer. Če mora to delati učitelj za vsakega učenca posebej, mu vzame ogromno časa, ki bi ga lahko usmeril drugam. Zato učitelju zelo prav pride orodje, ki mu olajša delo in ga naredi bolj učinkovitega.

S tem namenom je bila leta 2010 razvita prva različica spletne storitve Projekt Tomo. Izkazalo se je, da se kot orodje zelo dobro obnese pri poučevanju. S pridobljenimi izkušnjami smo leta 2015 razvili naslednjo različico, ki je odpravila nekatere pomanjkljivosti prejšnje in učitelju med drugim omogoča lažje spremljanje napredka učencev ter olajša pripravo učnih gradiv. Zakaj smo pri razvoju uporabili določene rešitve, si lahko preberete npr. v [2].

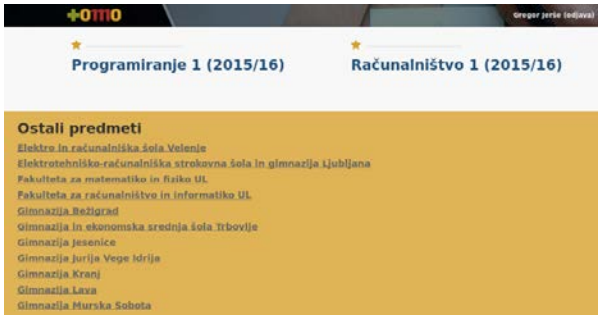
Ta spletna storitev se že vrsto let uporablja pri poučevanju na UL FRI in UL FMF. Uporabljajo jo tudi na več srednjih šolah in pri krožku programiranja na vsaj eni osnovni šoli. V članku bomo predstavili izkušnje, ki smo jih v tem času pridobili pri uporabi orodja Projekt Tomo.

2. Kratka predstavitev spletne storitve Projekt Tomo

Spletna storitev projekt Tomo [4] je prosto dostopna na spletu. Pred uporabo se moramo vanjo prijaviti. Storitev ne podpira ustvarjanja novih uporabnikov, ampak se vanj prijavimo z že obstoječim uporabniškim računom pri Facebook, Google ali ArnesAAI [5]. Možnost računa pri ArnesAAI je za slovenske učitelje in učence zelo zanimiva, saj ima večina šol svoja šolska uporabniška imena že v tem sistemu. Zato lahko do elektronskih storitev na šoli in Projekta Tomo dostopajo z uporabo istega uporabniškega imena in gesla, kar močno olajša delo. To potrjuje tudi spodnje mnenje.

»Odlično je, da za uporabo ni treba ustvarjati novih računov. Vsi naši dijaki so v sistemu upravljanja e-identitet (mdm.arnes.si) in imajo urejen AAI dostop, s katerim so se prijavljali v TOMO. Razen nekaj izjemam je prijava delovala brez težav. Tisti redki pa so uporabili Googlov račun.« (Matej Zdovc, profesor na I. gimnaziji Celje).

Ob prijavi se uporabniku odpre začetna stran, na kateri so našteti vsi predmeti, ki so mu na voljo.



Slika 2: Začetna stran

V zgornjem delu strani uporabnik vidi predmete, na katere je prijavljen, spodaj pa seznam predmetov, na katere se lahko še prijavi. Ti so urejeni po izobraževalnih ustanovah in projektih. Gradiva v predmetu so razdeljena na enote, ki jih imenujemo sklopi.

Slika 3: Predmet, razdeljen na sklope

Vsak sklop je sestavljen iz nalog, vsaka naloga pa vsebuje eno ali več podnalog. Ker je Tomo neodvisen od programskega jezika, je lahko vsaka naloga zastavljena v svojem programskem jeziku. Trenutno Tomo podpira tri programske jezike: Python 3, Octave in R. Dodajanje novega programskega jezika je možno, a zahteva nekaj dela. Trenutno sta v načrtu CSharp in C++.

Slika 4: Naloga, predstavljena v storitvi

Izbrano nalogo učenec s klikom na ustrezno ikono prenese k sebi v obliki tekstovne datoteke. V datoteki ima zapisano vse potrebno za reševanje naloge: besedilo naloge in prostor, kamor vpiše svojo rešitev.

```
# ================================================================================
# Grupa TNT
#
# Za uvod si najprej preberite [nekaj](https://sl.wikipedia.org/wiki/Alan_Ford)
# o stripu Alan Ford ([in še nekaj](https://stripovionline.com/AlanFord.html)).
# ================================================================================@000305=
# 1. podnaloga
# **Lakota**
#
# Broj 1 je z zaskrbljenostjo potresel škatlo. Le nekaj fižolovih zrn je še ostalo.
# Spet lakota v Cvetličarni. Začel je deliti - eno zrno za Alana, eno za Jeremijo, eno
zanj,
# eno za Grunfa. In spet eno zrno za Alana, eno za Jeremijo, eno zanj, eno za Grunfa in
...
# Ko so mu ostala manj kot 4 zrna, je zavzdihnil in vsa pospravil na svoj kupček. Že je
želel
# poklicati k večerji, ko je v kotu zastokal Jeremija in se prebudi. Ah, tudi on mora
dobiti
# svoj delež. Zrna na kup in eno za Alana, eno za Grunfa, ... Seveda, ko je bilo manj
kot pet zrn, gredo spet vsa
# na kupček Broja 1. In ko je bil že skoraj pri koncu, se od nekje pojavita Oliver in
Bob Rock. In spet
# od začetka ...
#
# Da bo šlo hitreje, sestavi funkcijo 'zrna(fižoli, številoOseb)', ki za dano število
zrn in oseb vrne
# koliko zrn dobijo člani skupine TNT in koliko Broj 1. Npr.
#
#         >>>zrna(72, 5)
#         14, 16
# ================================================================================

# ================================================================================@000312=
# 2. podnaloga
```

Slika 5: Besedilo naloge v obliki programa

Datoteko lahko odpre v poljubnem programskem okolju (npr. Python IDLE, Thonny, PyCharm …) in začne z delom. Storitev je zasnovana tako, da ne zahteva učenja novega programskega okolja.

V datoteki se pod razdelkom za pisanje rešitev nahaja koda za testiranje rešitve. Le-ta se izvede ob vsakem zagonu datoteke. Zato učencu za testiranje ni potrebno storiti ničesar drugega, kot samo pognati programsko kodo v datoteki. Ob zagonu se v grobem zgodita dve stvari.

1. Zažene se koda za preverjanje rešitve, ki je lahko poljubno kompleksna. Običajno se rešitev požene na nekaj testnih primerih, ki preverijo, ali učenčeva rešitev vrne pričakovani rezultat. Če rezultat ni ustrezen, učenec dobi takojšnjo povratno informacijo v obliki izpisa v

konzoli (Slika 6). Pomembno je poudariti, da ta korak deluje tudi v primeru, ko učenec ni priklopljen v omrežje. Testni primeri se namreč zaženejo lokalno na njegovem računalniku. Še ena dobra stvar pri tem je, da na takšen način ne obremenjujemo spletnega strežnika, na katerem teče Projekt Tomo. Zato ni težav ne glede na število uporabnikov spletne storitve.

2. Če je spletni strežnik dosegljiv, se nanj shrani učenčeva rešitev. Spletni strežnik hrani vso zgodovino oddaj za posameznega učenca. To zgodovino med drugim uporabimo za olajšanje dela učencem: če so neko nalogo že reševali, se njihova zadnja rešitev samodejno vključi v datoteko za prenos. Tako lahko doma nadaljujejo z delom, ne da bi jim bilo potrebno skrbeti za prenos kode iz šolskih računalnikov na domači. Zgodovina oddaj nam je med drugim prišla prav tudi pri lovljenju goljufov pri pisanju kolokvijev. Glavni namen hranjenja zgodovine pa je ta, da načrtujemo razvoj orodij, s katerimi bomo lahko to zgodovino analizirali. Učitelj bo lahko tako prepoznal tipične napake, morebitne zgrešene koncepte ter tudi grafično predstavil primerjavo posameznih nalog ter skupin učencev.

Ko je naloga v obliki programa prenesena, jo učenec reši. Rešitve napiše v ustrezni vmesni prostor. Ob zagonu programa dobi obvestilo o uspešnosti.

```
Shranjujem rešitve na strežnik... Rešitve so shranjene.
1. podnaloga nima veljavne rešitve.
  - Izraz zrna(3, 7) vrne 0.42857142857142855 namesto (0, 3).
  - Izraz zrna(32, 7) vrne 4.571428571428571 namesto (4, 8).
  - Izraz zrna(35, 7) vrne 5.0 namesto (5, 5).
2. podnaloga je brez rešitve.
3. podnaloga ima veljavno rešitev.

Process finished with exit code 0
```

Slika 6: Uspešnost reševanja naloge

Na sliki Slika 6 vidimo, da je učenec napačno rešil prvo pod nalogo, druge sploh še ni reševal, tretja pa je rešena pravilno.

In če je bil prej »semafor njegove uspešnosti« kot na sliki Slika 7

Slika 7: Še nobena naloga ni bila reševana

je sedaj (potrebno je seveda osvežiti stran v brskalniku) kot na sliki Slika 8.

Slika 8: Uspešnost reševanja

Učenec se lahko loti popravkov te naloge ali pa si prenese novo nalogo in rešuje to.

Na podoben način poteka tudi sestavljanje nalog, kar pa si bomo podrobneje ogledali v razdelku 4: Priprava učnega gradiva.

Zaradi svoje zasnove ima spletna storitev Projekt Tomo naslednje lastnosti [2]:

- Je preprost za uporabo, saj zahteva zelo malo dodatnega dela. Zato omogoča učiteljem in učencem, da se osredotočijo na programiranje.
- Omogoča oddaljeno delo brez stalne povezave z omrežjem.
- Učitelju omogoča preprost pregled nad napredkom skupine, kot tudi posameznih učencev.
- Ker se testni program izvaja na uporabnikovem računalniku, nam ni treba kupovati drage strojne opreme in paziti na škodo, ki jo lahko na strežniku povzroči oddaja zlobne kode.
- Podpira lahko poljuben programski jezik.

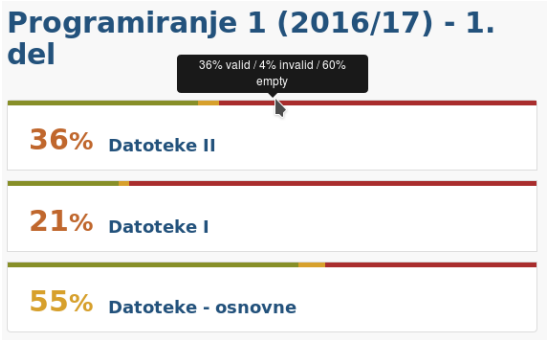
Izvorna koda spletne storitve je na voljo v storitvi Github [6]. Tako jo lahko vsakdo z zadostnim znanjem postavi na svojem strežniku in jo popravi/dopolni po svojih potrebah.

Po naših izkušnjah Projekt Tomo svojo največjo moč kot pomoč učiteljem pokaže predvsem na treh področjih: pri spremljanju napredka učencev, pri pripravi učnega materiala in pri analiziranju dela učencev. V tem prispevku si bomo podrobneje ogledali prva dva.

3. Spremljanje napredka učencev

Pri izgradnji najnovejše različice Projekta Tomo je bilo veliko dela namenjenega izdelavi uporabniškega vmesnika. Želeli smo, da je kar se da preprost in pregleden, kljub temu pa naj posreduje vse pomembne informacije. To smo poskusili doseči z uporabo vizualnih orodij, kot so pasice in tortni diagrami.

Pasice uporabljamo za spremljanje napredka učencev pri sklopih. Kot učitelji na prvi strani pri svojih predmetih vidimo pasico nad sklopi. Ta pasica je sestavljena iz treh barv: rdeče, rumene in zelene. Rdeča barva pomeni, koliko nalog pri sklopu učenci niso niti poskušali reševati, rumena koliko nalog je bilo rešenih narobe in zelena delež sprejetih rešitev.



Slika 9: Skupni uspeh učencev pri posameznem sklopu

Na ta način lahko hitro vidimo, kako uspešni so bili učenci pri določenem sklopu. Če nas zanimajo točni podatki, se z miško

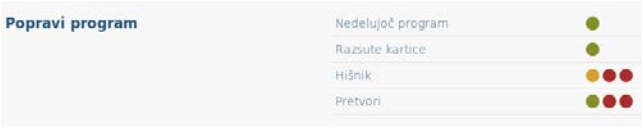
premaknemo nad pasico. Takrat se v oblačku izpišejo točni odstotki.

Ob izbiri predmeta se učitelju enaka informacija ponovi na sklopih, poleg tega pa je na desni strani izpisan še poimenski seznam vseh sodelujočih učencev pri predmetu.

Slika 10: Uspešnost reševanja posamezne naloge - skupno in posamično

Poleg vsakega učenca je izrisan diagram, ki je sestavljen iz istih treh barv kot pasica pri sklopih. Ta diagram nam pove, kako uspešen je določen študent pri reševanju nalog pri našem predmetu. Delež rdeče barve pri tortnem diagramu pomeni odstotek nalog, ki jih učenec še ni oddal, delež rumene predstavlja delež nesprejetih oddaj in delež zelene pomeni, koliko učenčevih oddaj je bilo sprejetih.

Če učitelja napredek posameznega študenta zanima še podrobneje, s klikom na njegovo ime odpre razdelek s podrobnimi informacijami (Slika 11). Na strani je seznam vseh nalog pri predmetu. Poleg vsake je za vsako podnalogo en barvni krog. Krog je zelene barve, če je učenec podnalogo rešil pravilno, rumene, če jo je rešil narobe in rdeče, če je še ni poskušal reševati.

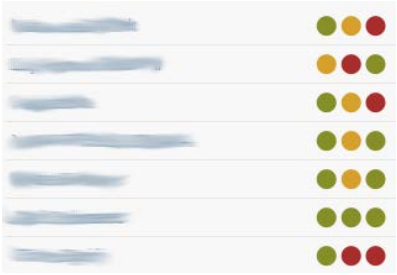


Slika 11: Uspešnost posameznega učenca za sklop

Napredek v posameznem sklopu se učitelju prikaže na strani sklopa. Tudi tu je na desni za vsako nalogo v sklopu izrisanih več tortnih diagramov, po eden za vsako podnalogo, barvno kodiranih na znan način.

Slika 12: Napredek v posameznem sklopu

Tak prikaz se je izkazal kot zelo uspešen pri razvrščanju nalog po težavnosti in ugotavljanju, s katerimi tipi nalog imajo učenci težave. Velikokrat se nam je zgodilo, da smo šele ob pregledu strukture poskusov za posamezno nalogo videli, katero snov je treba še dodatno povaditi. Če nas poimensko zanima, kateri študentje so uspešno rešili nalogo, to dosežemo s klikom na tortni diagram. Odpre se stran, kjer je za vsakega učenca v predmetu nazorno prikazano, katere naloge je rešil.



Slika 13: Uspešnost reševanja posameznih učencev

4. Priprava učnega gradiva

Prvi korak na poti do priprave gradiva (sklopa nalog) je ta, da pri predmetu dobimo status učitelja. To lahko storimo s prošnjo upravljavcem strani (ki po potrebi tudi ustvarijo nov predmet), ali pa nam učiteljske pravice pri obstoječem predmetu podeli eden izmed trenutnih učiteljev tega predmeta. To stori zelo preprosto: na spletni strani predmeta je na desni strani naštet seznam vseh prijavljenih učencev. Izberemo si bodočega učitelja in mu s klikom na ikono kapice (Slika 14) podelimo učiteljske pravice na tem predmetu. Na podoben način lahko učitelju status tudi odvzamemo in ga s tem spremenimo v učenca.

Slika 14: Dodeljevanje pravic učitelja

Ko postanemo učitelj, s klikom na gumb "Dodaj sklop" ustvarimo sklop, v katerega bomo dali naloge. Sedaj imamo na voljo dve možnosti.

1. Naloge prenesemo iz drugih predmetov.
2. Naloge sestavimo sami.

V nadaljevanju razdelka si bomo podrobneje ogledali obe možnosti.

4.1 Prenos gradiv iz drugih predmetov

Kakor hitro imamo v nekem predmetu status učitelja, lahko vanj prenašamo naloge iz drugih predmetov. Izberemo si nalogo, ki jo želimo prenesti. Ob kliku na gumb kopiraj (glej Slika 15) se odpre okno, v katerem izberemo želeni predmet in sklop, v katerega se bo naloga prenesla, ter kliknemo na gumb "Prenesi". Naloga se v novi sklop prenese v celoti, z besedilom nalog in vsemi testnimi primeri. Naredi se kopija naloge, ki ni več povezana z nalogo, ki smo je kopirali. To nam omogoča popravljanje prenesenih nalog, ne da bi pri tem spremenili originale.

Slika 15: Izbira predmeta, kamor bomo kopirali nalogo

Ta možnost je še posebej zanimiva za srednješolske učitelje programiranja, saj je bila v okviru projekta NAPOJ [7] zgrajena tako zbirka nalog iz učbenika Informatika [8] kot tudi zbirka nalog, primernih za dodatne vaje. Kdor ima status učitelja pri vsaj enem od predmetov, lahko naloge iz poljubnega predmeta (torej tudi iz teh dveh zbirk) prosto prenaša v svoje predmete. Tako si lahko ob pomanjkanju časa oz. idej zelo preprosto in hitro pripravi naloge na dano tematiko.

Možnost kopiranja je bila pri učiteljih zelo dobro sprejeta. Naj navedemo nekaj njihovih mnenj.

Kopiranje nalog iz pripravljene učilnice za informatiko (projekt NAPOJ), mi ni predstavljalo nobenega problema. Prav tako ne manjši popravki nalog in tudi sestavljanje nalog z bolj enostavnim preverjanjem rešitev, za zahtevnejše rešitve pa bi potrebovala pa še kakšno uro dodatnega izobraževanja in predvsem časa za testiranje in vajo. Karmen Kotnik, Šolski center Celje, Gimnazija Lava

Profesor lahko enostavno ustvarja nove sklope in dodaja naloge iz skladišča. Naloge v spletni storitvi Tomo niso zaščitene z avtorskimi pravicami in so prosto dostopne. Naloge na določeno temo lahko enostavno kopiram v svoj sklop, kjer jih lahko poljubno prilagajam. Za dodajanje novih nalog pa bi bilo treba pridobiti ustrezna znanja (seminar!?), saj programiranje preverjanja ni enostavno opravilo. Matej Zdovc, profesor na I. gimnaziji Celje.

Fino je, da so naloge razvrščene po tematikah in težavnosti. Učitelji si lahko med seboj naloge izmenjujejo in jih po svoje prirejajo, ne da bi s tem spremenili original. Čeprav je uporaba Toma dokaj preprosta, je nekoliko težji oreh nalogo sestaviti oz. izdelati ustrezno preverjanje rešitev. Jaz bi na tem področju potrebovala še kar nekaj ur usposabljanja. Romana Vogrinčič, profesorica na gimnaziji Murska Sobota

Razgovori z učitelji - uporabniki pa so pokazali, da pogrešajo možnost, da kopiramo le povezavo na nalogo. Če namreč isto nalogo uporabimo pri več predmetih in ugotovimo, da jo moramo kasneje popraviti, moramo sedaj to narediti posebej prav pri vseh predmetih, kjer smo jo uporabili. Pri kopiranju povezave pa bi bilo potrebno spremeniti le original. Seveda to pomeni tudi to, da če avtor originalne naloge to spremeni, se samodejno spremenijo tudi vse kopije.

V eni od prihodnih nadgradenj Projekta Tomo bo torej možno kopirati nalogo kot samostojno in neodvisno od originala, ali pa jo le uporabiti kot povezavo do originala.

4.2 Popravljanje obstoječih gradiv in priprava novih

Če z obstoječo nalogo učitelj ni zadovoljen in bi jo rad popravil, si najprej na računalnik prenese datoteko za urejanje. Odpre jo v svojem najljubšem urejevalniku besedil, popravi besedilo in datoteko zažene (glej Slika 16). Ob zagonu se spremenjeno besedilo prenese na spletni strežnik in spremembe v besedilu naloge so takoj vidne na spletni strani. Besedilo naloge je zapisano v formatu Markdown [9], ki nam omogoča nekaj preprostega oblikovanja.

```
# ================================================================@002908=
# Zapišite funkcijo `geometricna(a, b)`, ki za števili $a$ in $b$ vrne
# geometrično sredino teh dveh števil, tj. $\sqrt{ab}$.
#
# Namig - oglej si [dokumentacijo](https://docs.python.org/3/library/math.html)
# in [zgled](http://stackoverflow.com/questions/8783261/python-math-module)
# ================================================================================
import math # da bomo lahko uporabili vgrajeno funkcijo za korenjenje
def geometricna (a, b):
    '''Funkcija vrne geometrično sredino števil a in b.'''
    return math.sqrt(a*b)

Check.part()
Check.equal("""geometricna(3, 12)""", 6)
Check.equal("""geometricna(75, 12)""", 30)
Check.equal("""geometricna(101, 0)""", 0)

for a in range(1, 10):
    for b in range(2, 8):
        Check.secret(geometricna(a, b), math.sqrt(a*b))
```

Slika 16: Urejanje naloge

Pod besedilom naloge sledi uradna rešitev ter testni primeri, katerim morajo zadoščati rešitve (tudi uradna), da so proglašene za sprejete. Pri tem je uradna rešitev obvezna in je ne smemo izpustiti. S tem ima učitelj na prvi pogled nekaj več dela, ampak prinaša veliko prednosti. Velikokrat namreč šele pri reševanju opazimo težave s formulacijo naloge in se domislimo dobrih testnih primerov zanjo. Poleg tega pa uradno rešitev lahko pokažemo tudi učencem. Zato se splača potruditi in rešitev napisati res dobro, da se učenci ob njenem pregledu naučijo tudi lepega programiranja.

Pri pisanju testnih primerov lahko učitelj pokliče metode, ki jih je sestavil učenec, ter dostopa do njegove izvorne kode. Ker je pisanje testov zamudno opravilo, smo v ta namen sestavili pomožni razred `Check`, ki vsebuje nekaj priročnih metod za lažje testiranje. Verjetno najbolj uporabljena je metoda `equal(izraz, pričakovani_rezultat)`. Z njo preverimo, če nam klic podanega izraza vrne pričakovani rezultat. Če ga ne, to tudi sporočimo uporabniku.

Seveda pa so lahko testni primeri poljubno zahtevni. Denimo, da naloga zahteva izračun produkta lastnih vrednosti matrike, pri čemer ne bi radi, da bi učenec uporabil vgrajeno metodo za izračun determinante. V grobem ta lahko storimo na dva načina.

Na sliki Slika 17 vidimo primer preprostega testa, ki preveri le to, da v učenčevi rešitvi ni vsebovana beseda det. Do izvorne kode rešitve dostopamo z izrazom `Check.current_part['solution']`.

Seveda to ni čisto prav, saj lahko učenec besedo `det` uporabi tudi v komentarju ali kot ime spremenljivke oz. je del daljšega imena.

```
# ======================================================================
# Produkt lastnih vrednosti
#
# Izračunajte produkt lastnih vrednosti podane matrike.
# ======================================================================@002693=
# Sestavite metodo produkt_lastnih_vrednosti (A), ki sprejme matriko A in
# vrne produkt njenih lastnih vrednosti. Pri tem ne smete uporabiti metode det.
#
# Primer:
#
#     >>> produkt_lastnih_vrednosti([1 0, 0 1])
#     1
# ======================================================================
from numpy.linalg import eigvals
from numpy import prod

def produkt_lastnih_vrednosti(A):
    lastne_vrednosti = eigvals(A)
    return prod(lastne_vrednosti)

Check.part()

izvorna_koda = Check.current_part['solution']
if "det" in izvorna_koda:
    Check.error('Metode det ne smete uporabiti')
```

Slika 17: Napredni testi

```
import ast
tree = ast.parse(Check.current_part['solution'])
for node in ast.walk(tree):
    if isinstance(node, ast.Call):
        name = node.func.id
        if name == 'det':
            Check.error('Metode det ne smete uporabiti')

Check.equal('float(produkt_lastnih_vrednosti([[1, 0], [0, 1]]))', 1)
Check.equal('float(produkt_lastnih_vrednosti([[-1, 3], [2, 1]]))', -7)
Check.secret(produkt_lastnih_vrednosti([[2, 1], [0, 1]]))
```

Slika 18: Primer testnega primera z uporabo AST

Na sliki Slika 18 je zgornji testni primer nadgrajen tako, da analizira učenčevo rešitev z uporabo abstraktnih sintaktičnih dreves in preveri, da njegov program nikoli ne pokliče metode z imenom det. Na ta način se izognemo zgoraj opisanim težavam in res preverimo le, da se metoda z imenom det nikoli ne pokliče.

5. Uporaba sistema v praksi

Sistem Projekt Tomo že kar nekaj časa uporabljamo na Fakulteti za matematiko in fiziko, Univerze v Ljubljani pri različnih predmetih. Ti so tako predmeti, namenjeni učenju programiranja, kot tudi predmeti s področja študija podatkovnih struktur in algoritmov, ter kot posledica podpore jezikoma Octave in R tudi s področja numerične in finančne matematike. Izkazal se je kot učinkovit pripomoček pri izvajanju laboratorijskih vaj. Večina teh poteka tako, da študenti samostojno rešujejo naloge iz sistema Tomo s svojim tempom. S tem omogočijo asistentu, da se ukvarja s "pravimi" problemi, torej usmerja učence in jim po potrebi pomaga razložiti kakšen koncept, ne pa da porablja čas za reševanje enostavnih težav. Kot zelo koristna se je tudi pokazala možnost, da je mogoče "na hitro" popraviti nalogo, spremeniti kak test. Že velikokrat smo namreč med samimi vajami ugotovili, da kakšna naloga in dobro formulirana oz. da pri njej manjka kakšen dodaten testni primer. Glede na zasnovo sistema je mogoče nalogo hitro spremeniti in dopolniti, nakar je popravljena takoj na voljo učencem.

Kot smo že omenili, se uporablja tudi na več srednjih šolah, pa tudi pri krožku programiranja v osnovni šoli.

Razmah uporabe v srednjih šolah je Projekt Tomo doživel v letu 2016/17, kar je bila v veliki meri posledica zagona projekta NAPOJ [7] [10] .

Cilj projekta NAPOJ (Načrtovanje poučevanja Algoritmov in Programiranja ter OrganizaciJa skupnosti) je bil vzpostaviti aktivno skupnost učiteljev računalništva in informatike, ter jih opremiti s potrebnimi materiali in orodji. V sklopu projekta se je izvedla vrsta delavnic in aktivnosti, kjer so sodelavci projekta skupaj z izbranimi učitelji v celoto povezali novi e-učbenik za predmet Informatika v gimnazijah, spletno učilnico in spletni sistem za avtomatsko preverjanje nalog Projekt Tomo, v katerega je bilo v okviru projekta NAPOJ dodana obširna zbirka nalog iz programiranja (https://www.projekt-tomo.si/course/19/), tako iz zgornjega učbenika, kot tudi v okviru samostojne zbirke nalog (https://www.projekt-tomo.si/course/20/) .

Ena od dobrih lastnosti storitve Tomo je tudi ta, da prevzame vlogo »sitnega prfoksa«, ki vztraja, da morajo biti rešitve take, kot zahteva besedilo naloge. To nam plastično ponazori naslednji dopis, ki smo ga prejeli od profesorice z ene od slovenskih Gimnazij.

"Kar pa jih je motilo, je bilo dejstvo, da so dobili oranžno ali celo redečo oznako pri rešitvah, ki so dale pravilne rezultate, so pa recimo našli drugo pot rešitve problema, kot je predvidena, ali celo, če so samo drugače poimenovali spremenljivke. Njihov napredek je bil potem napačno zabeležen."

Seveda se nam je to zdelo zelo čudno, saj je cela ideja projekta Tomo ravno ta, da sprejmemo vse rešitve, ki prestanejo testne primere. Zato smo profesorico prosili za pojasnilo, pri kateri nalogi in kdaj se to dogaja. Šlo je za eno od nalog iz novega e-učbenika [8], preneseno v sistem Tomo. Nalogo vidimo na sliki Slika 19.

1. podnaloga

Napiši program, ki prebere celo število in izpiše besedilo DA, če je število večje od 42, sicer pa naj ne izpiše ničesar. Sledijo štirje primeri:

```
>>>Vnesi celo število: 50
DA
>>>Vnesi celo število: 42
>>>Vnesi celo število: 41
>>>Vnesi celo število: 43
DA
```

Slika 19: Težavna naloga

Spodaj si oglejmo uradno rešitev in rešitev učenca. Kot vidimo, se razlikujeta samo v eni črki: dijak je zapisal besedo "vnesi" z malo začetnico, medtem, ko je naloga zahtevala veliko. Seveda bi lahko razpravljali o tem, ali naloga zahteva točno določen pozivnik (torej, ali je v primeru navedena oblika pozivnika zahtevana), a recimo, da je temu tako.

Your solution

```
a = int(input("vnesi celo število: ")) if a >
42: print("DA")
```

Official solution

```
beri = input('Vnesi celo število: ') n =
int(beri) if n > 42: print('DA')
```

Slika 20: Učenčeva in uradna rešitev

Tukaj spet pride do izraza dejstvo, da je Projekt Tomo zgolj orodje v rokah učitelja, ki se mora odločiti, kaj želi z nalogo doseči. Pri zgornji nalogi ima na voljo številne možnosti. Naj navedemo le štiri.

1. Nalogo pusti tako, kot je. Namen takšne naloge je navaditi učence natančnega branja navodil, zahtevkov in tega, da se jih je treba dosledno držati. Projekt Tomo nam je tukaj v veliko pomoč, saj ni treba vsakemu učencu posebej razlagati, da njegova rešitev ni dobra zaradi ene same velike črke. Sistem je pač ne sprejme kot pravilne in naloga ni rešena pravilno.
2. Spremeni test tako, da, če učenec napiše "vnesi" namesto "Vnesi", dobi povratno informacijo, da naj pazi na malo/veliko začetnico. To je nekoliko milejša oblika prejšnjega primera, kjer učenca opozorimo na pogosto napako.
3. Spremeni test tako, da je vseeno, če so uporabljene male / velike črke. To je smiselno v primeru, da je učiteljev poudarek pri nalogi drugje in je izpis postranskega pomena.
4. Spremeni test tako, da je popolnoma vseeno kakšen "pozivnik" učenec uporabi v svoji rešitvi.

In seveda - kar je najpomembneje - učiteljeva naloga je reagirati na različne dogodke, ki se lahko med izvajanjem vaj pripetijo. In ravno to je osnovni razlog, da razvijamo Projekt Tomo: da učitelja razbremenimo preprostih opravil in mu damo dodatni čas, da se med vajami lahko pogovori z učencem. Poleg tega menimo, da na ta način omogočamo tudi poglobljeno razpravo z učenci, saj je učitelj razbremenjen vsaj dela odpravljanja rutinskih napak. S tem se je strinjala tudi profesorica, saj je napisala

"No, pod črto..... TOMO je zelo dobrodošlo orodje tako za učitelje kot za učence. Pridobitev na času je očitna. Dijaki lahko napredujejo z različnim tempom. Naučijo se tudi, da je branje navodil pomembno.... Kar se tega tiče, ne bi ničesar spreminjala. Morda pa bi v obrazložitev oranžne pikice dodala tolažilni stavek 'Rešitev sicer deluje, a ni čisto v skladu z navodili'. Tako jih jaz potolažim, ko so nejevoljni, 'kaj ta TOMO tako komplicira').".

6. Vtisi srednješolskih profesorjev

Projekt Tomo uporablja tudi nekaj srednješolskih učiteljev pri poučevanju programiranja. Da bi še izboljšali storitev, smo jih vprašali, kakšno je njihovo mnenje o sistemu. Pripenjamo nekaj odgovorov, ki so se nam zdeli zanimivi. Del njih ste lahko prebrali že v razdelku Prenos gradiv iz drugih predmetov.

"Projekt Tomo, kot neke vrste spletna vadnica je dobrodošel pripomoček učitelju na katerikoli stopnji, pri učenju programiranja. V vsakem primeru pa pri delu v skupini naletimo na različne dijake po zmožnosti dojemanja programerskih prijemov, usvajanju pojmov in sintakse programskih jezikov in samostojnega pisanja programov. Zato je pripomoček, kot je Projekt Tomo, še kako dobrodošel za uporabo pri pouku. V prvi vrsti predvsem zelo poenostavlja delo. Že najbolj banalna stvar - ni treba iskati in projicirati besedil nalog, saj so naloge pregledno napisane z vsemi navodili; sistem omogoča hitro in preprosto izdelavo novega nabora nalog, ali popravljanja obstoječega, z zares domiselnim in preprostim vmesnikom. Predvsem pa, kar je najpomembneje, omogoča enostavno spremljanje samostojnega dela dijakov, spremljanje njihovega napredka, in diferenciacijo. S tem dosežemo zares učinkovito poučevanje in učenje programiranja. Zato sem kot učitelj programiranja s portalom zelo zadovoljen in ga uporabljam vedno, ko imam skupino, s katero programiram." Profesor z ene slovenskih gimnazij

"*Okolje Tomo ni povzročalo nikakršnih težav in so ga vsi dijaki z lahkoto dokaj hitro osvojili. … orodje Tomo mi je zelo pomagalo pri poučevanju programiranja v Pythonu, omogoča mi diferenciacijo pri poučevanju: ni potrebno, da vsi skupaj hkrati rešujemo eno nalogo, ampak lahko dijaki napredujejo z različnimi hitrostmi. Tako se lahko nekateri dijaki naučijo bistveno več, kot če bi šli vsi z istim tempom."* Klemen Bajec, Gimnazija Vič

"Toma sem uporabljala v lanskem šolskem letu pri pouku informatike v 4. letniku (priprava na maturo). Imela sem zelo raznoliko skupino - več kot pol jih je že programiralo, trije pa so bili popolni začetniki. In Tomo mi je bil tu v veliko pomoč. Vanj sem v glavnem dajala naprednejše oz. dodatne naloge, z začetniki pa sem delala osnove. Dijakom je bilo všeč, ker so lahko delali samostojno, s svojo hitrostjo in za rešene naloge dobili povratno informacijo. Nekateri so se pritoževali pri tistih nalogah, kjer je bilo treba npr. narediti za pravilno rešitev natančno tak izpis, kot je bil predviden v rešitvi. Navajala sem jih na to, da je pomembno, da si po oddani rešitvi naloge podrobno pogledajo tudi uradno rešitev in o njej razmislijo. Če je dopuščal čas, smo kakšno pogledali tudi skupaj. Prikaz napredka je bil motivacija za njih, zame pa zelo hiter pregled tega, kaj je kdo naredil oz. česa še ne.

Kopiranje nalog iz pripravljene učilnice za informatiko (projekt NAPOJ) mi ni predstavljalo nobenega problema. Prav tako ne manjši popravki nalog in tudi sestavljanje nalog z bolj preprostim preverjanjem rešitev, za zahtevnejše rešitve bi potrebovala pa še kakšno uro dodatnega izobraževanja in predvsem časa za testiranje in vajo. V glavnem pa se je TOMO izkazal kot odličen pripomoček za učenje programiranja. Začetni vložek časa je za učitelja sicer precejšen, vendar pa ima kasneje lažje delo in veliko boljši pregled nad delom dijakov, lažje je tudi delo z dijaki z različnim predznanjem." Karmen Kotnik, Šolski center Celje, Gimnazija Lava

»Moja želja je bila, da pri pouku informatike v šolskem letu 2016/17 uvedem programiranje, ki po mojem mnenju predstavlja dodano vrednost predmeta… Dijake, ki so se odločili za Python, so prišli z različnim nivojem predznanja/informiranosti. Nekateri so se odločili tako, ker pač v drugi skupini ni bilo prostora. … Tomo se je pokazal kot dobro izhodišče za učenje, ima dobre in slabe lastnosti, ker pa se še razvija lahko postane odlično/odločilno orodje za učenje programiranja.

Profesorja lahko enostavno ustvarja nove sklope in dodaja naloge iz repozitorija. Naloge na TOMO niso zaščitene z avtorskimi pravicami in so prosto dostopne. Naloge na določeno temo lahko enostavno kopiram iz repozitorija v svoj sklop, kjer jih lahko poljubno prilagajam. Za dodajanje novih nalog pa bi bilo treba pridobiti ustrezna znanja, saj programiranje preverjanja ni enostavno opravilo.

Iz organizacijskega vidika morda manjka kakšno dodatno vnosno polje za razporejanje dijakov npr. po oddelkih. Po prijavi vseh dijakov v sistem je seznam kar naenkrat postal dolg in slabo pregleden.

TOMO mi je prihranil veliko dela, saj bi bilo ročno pregledovanje nalog zahteven časovni zalogaj. Povratna informacija s strani večine dijakov je bila pozitivna. Na vajah so radi delali in reševali probleme. Za dijake je bil izziv doseči rešitev, hkrati pa so se mimogrede naučili učno snov. Razlike so seveda obstajale, večini je bil dovolj začetni namig, medtem ko so posamezniki potrebovali dodatno pomoč. Naloge so reševali v šoli, kot tudi doma in predvsem pri delu doma je bilo težko preverjati samostojnost dela. … Problem, ki se je pojavljal pri reševanju nalog je, da je TOMO zelo zahteven glede preverjanja parametrov napisanega programa. Dovolj so že male razlike v zapisu nizov, pa bo javil, da je rešitev napačna. Zaradi tega je vse preveč ukvarjanja s »pravopisnimi« napakami, namesto s programiranjem!

TOMO je zame zelo uporaben pripomoček, dijaki so se v njem lepo znašli. Glede na pozitivne izkušnje ga bom z veseljem uporabljal tudi v tem šolskem letu pri pouku informatike.« Matej Zdovc, profesor na I. gimnaziji Celje.

Fino je, da so naloge razvrščene po tematikah in težavnosti. Učitelji si lahko med seboj naloge izmenjujejo in jih po svoje prirejajo, ne da bi s tem spremenili original. Čeprav je uporaba TOMA dokaj preprosta, je pa nekoliko težji oreh nalogo sestaviti oz. izdelati ustrezno preverjanje rešitev. Jaz bi na tem področju potrebovala še kar nekaj ur usposabljanja.

Dijaki so TOMO zelo lepo sprejeli. V bistvu so se učili "izkustveno" korak za korakom in na ta način sami spoznavali in raziskovali "skrivnosti in prelesti!" Pythona. TOMO je zelo dobrodošlo orodje tako za učitelje kot za učence. pridobitev na času očitna. Dijaki lahko napredujejo z različnim tempom. Naučijo se tudi, da je branje navodil pomembno. Romana Vogrinčič, profesorica na gimnaziji Murska Sobota

7. Zaključek

Projekt Tomo učitelju na zgoraj opisane načine olajša delo, tako da lahko več pozornosti nameni pripravi gradiv in poučevanju. Prav tako pomaga študentom, ki dobijo takojšen odziv glede pravilnosti njihovih rešitev in jim tako omogoča hitrejši napredek.

Pri tem je veliko odvisno od kakovosti povratnih informacij, ki jih učencu vrnejo testni programi. Zato je pisanje testnih primerov zelo zamudno opravilo, saj zahteva veliko premišljevanja o napakah, ki jih učenci pogosto storijo, iskanja robnih primerov in šele nato lahko začnemo pisati testne primere, ki bodo vse te napake prestregli in dali učencu primeren odziv glede na napako, ki jo je naredil. Po naših izkušnjah je to zelo težko storiti naenkrat, ampak zahteva neprestano delo: učitelj pripravi testne primere, nato analizira oddane rešitve in na osnovi analize popravi testne primere in ponavlja postopek.

Na Fakulteti za računalništvo in informatiko Univerze v Ljubljani je bil razvit spletni sistem za pomoč pri poučevanju programiranja CodeQ [11]. Ta uporablja drugačen pristop, in sicer poskuša generirati nasvete programerju samodejno in mu, glede na napake, ki jih dela, posredovati ustrezne povratne informacije.

Za povprečnega učitelja informatike že pisanje preprostih testov predstavlja velik napor. Zato je v teku priprava skupnega projekta, ki bi na osnovi sistemov Projekt Tomo in CodeQ zgradila novo spletno storitev, ki bi kombinirala najboljše rešitve obeh obstoječih sistemov. Med drugim naj bi omogočili grafično sestavljanje testnih primerov, ki bo veliko preprostejše od sedanjega načina. Učiteljem bo omogočil, da bodo lahko hitro sestavili preproste testne primere ne da bi morali pisati testne programe. Seveda pa bo možnosti pisanja testnih primerov v sami kodi še kar ostala, saj bo uporabniški vmesnik primeren le za preprostejše teste.

Prav tako pa naj bi se sistem učil, torej na osnovi prejetih rešitev učitelju predlagal vključitev dodatnih testov in povratnih informacij.

Tovrstna orodja bodo, kot kaže, nujna, saj je tudi v zavest širše javnosti prišlo spoznanje, da je vsaj osnovno znanje programiranja dandanes izjemno koristno. In kot kažejo izkušnje iz tujine, je trenutno glavna težava pri širši vpeljavi učenja programiranja ta, da primanjkuje dobrih učiteljev programiranja. Noben tak sistem seveda (še) ne more nadomestiti učitelja, lahko pa mu pomaga, da se laže sooči z večjimi skupinami.

8. Literatura

[1] M. Pretnar, „Spletna storitev za poučevanje programiranja,“ v *Vzgoja in izobraževanje v informacijski družbi - VIVID 2014* , Kranj: Fakulteta za organizacijske vede, 2014.

[2] M. Pretnar in M. Lokar, „A Low Overhead Automated Service for teaching Programming,“ v *Proceedings of the 15th Koli Calling International Conference on Computing Education Research*, Koli, Finland, 2015.

[3] A. T. Corbett in J. R. Anderson, „Locus of feedback control in computer-based tutoring: impact on learning rate, achievement and attitudes,“ v *In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '01).* , New York, 2001.

[4] M. Pretnar, „Projekt Tomo,“ 25 9 2010. [Elektronski]. Available: https://www.projekt-tomo.si.

[5] „Arnes AAI,“ 2017. [Elektronski]. Available: https://aai.arnes.si/.

[6] M. Pretnar, „GitHub,“ 2017. [Elektronski]. Available: https://github.com/matijapretnar/projekt-tomo.

[7] G. Anželj, A. Brodnik in M. Lokar, „NAPOJ – proti aktivni skupnosti učiteljev računalniških predmetov,“ v *VIVID* , Ljubljana, 2017.

[8] G. Anželj, J. Brank, A. Brodnik, P. Bulić, M. Ciglarič, M. Đukić, L. Fürst, M. Kikelj, A. Krapež, H. Medvešek, N. Mori, M. Pančur in P. Sterle, „Računalništvo in informatika,“ 25 9 2017. [Elektronski]. Available: http://lusy.fri.uni-lj.si/ucbenik/book/index.html.

[9] „Wikipedia,“ 25 9 2017. [Elektronski]. Available: https://en.wikipedia.org/wiki/Markdown.

[10] A. Brodnik, M. Lokar in N. Mori, „Activation of Computer Science Teachers in Slovenia,“ v *World Conference on Computers in Education (WCCE) 2017*, Dublin, Ireland, 2017.

[11] „CodeQ,“ 25 9 2017. [Elektronski]. Available: https://codeq.si.