Celostna prenova priprave nalog in ocenjevanja pri uvodnem programerskem predmetu

Luka Fürst

Fakulteta za računalništvo in informatiko, Večna pot 113, 1000 Ljubljana
E-pošta: luka.fuerst@fri.uni-lj.si

Holistic Reform of Task Preparation and Grading in an Introductory Programming Course

In the introductory programming course taught at the University of Ljubljana, Faculty of Computer and Information Science, we introduced a new task preparation and grading system. The main novelty of the system is the automatic evaluation of the students' programs using a set of test cases. The other, no less important part of the reform was the restructuring of the entire body of programming assignments. All assignments are now consistently formatted and equipped with a set of test cases divided into difficulty groups. In the paper, we present the rationale for the introduction of the new system, the system itself, and our experiences four years since its adoption.

1 Uvod

Računalništvo in informatika prodirata v vse pore našega življenja. Poučevanje ni nikakršna izjema. S »pametnimi« tablami je opremljena domala vsaka osnovna šola, da o tehnološki usposobljenosti, ki jo izkazujejo današnji učenci, niti ne govorimo. Računalniki pa niso v pomoč le pri podajanju snovi, ampak tudi pri ocenjevanju znanja. Obstaja čedalje več sistemov za ocenjevanje nalog, celo takih, pri katerih ne bi pričakovali, da jih bo mogoče kdaj avtomatizirati [1]. Prav nič nas ne preseneča, da so bile programerske naloge predmet enega od prvih avtomatskih ocenjevalcev [2], danes pa je na voljo kar lepo število sistemov [3, 4, 5].

Pri predmetu Programiranje 1 v prvem letniku univerzitetnega študija na Fakulteti za računalništvo in informatiko Univerze v Ljubljani (v nadaljevanju: FRI) imamo pestro zgodovino različnih ocenjevalnih režimov. V pričujočem prispevku predstavljamo prehod od načina ocenjevanja, ki smo ga v študijskem letu 2009/10 uvedli kot odziv na smernico, da naj bi študent polovico svoje ocene pri predmetu prejel s sprotnim delom tekom semestra, do avtomatiziranega sistema, ki smo ga po naraščajočem nezadovoljstvu z obstoječim stanjem vpeljali pet let kasneje. Poleg sistema ocenjevanja smo prenovili tudi sistem priprave programerskih nalog. Pri tem smo kot glavno merilo upoštevali konsistentnost: vse naloge naj bodo strukturirane na enak način in naj se po enakem postopku tudi ocenjujejo.

V razdelku 2 bomo predstavili predmet Programiranje 1 na FRI. Razdelek 3 je namenjen opisu ocenjevalnega sistema pred prenovo, razdelek 4 pa opisu trenutnega ocenjevalnega sistema. V razdelku 5 bomo razpravljali o izkušnjah s prenovo, z razdelkom 6 pa bomo članek zaključili.

2 Predmet Programiranje 1 na FRI

Pri predmetu Programiranje 1 se študentje na primeru programskega jezika java seznanijo s temeljnimi koncepti proceduralnega in objektno usmerjenega programiranja. Snov predmeta zajema krmilne strukture, funkcije (metode), tabele, razrede in objekte, dedovanje in osnove računalniške grafike. Predmet se izvaja v obliki predavanj in vaj. Predavanja so namenjena vsem študentom hkrati, na vajah pa so slušatelji razdeljeni v skupine po 18.

Študent[1] lahko na podlagi svojega sprotnega dela pridobi največ 50 točk, na izpitu pa največ 60. Da predmet opravi, mora skupaj zbrati najmanj 50 točk, od tega najmanj 25 na izpitu. Z uspešnim sodelovanjem na vsakoletnem tekmovanju v programiranju namiznih iger ali natečaju za pripravo domačih nalog [6] se mu k točkam za sprotno delo prišteje še nekaj dodatnih točk.

Gradiva za predmet Programiranje 1, ki jih pripravljamo in vzdržujemo izvajalci, vključujejo okrog 150 programerskih nalog. Poleg nalog, ki se rešujejo na predavanjih in vajah, ima študent na voljo 10 obveznih domačih in 76 neobveznih dodatnih nalog.

3 Ocenjevanje pred prenovo

Način ocenjevanja programerskih nalog se je v zgodovini predmeta Programiranje 1 večkrat spremenil. V tem razdelku bomo opisali sistem, ki smo ga uporabljali v letih od 2009/10 do vključno 2013/14.

V navedenem obdobju so študentje lahko prejeli do 30 točk iz domačih nalog, 20 točk iz teoretičnih kolokvijev in 50 točk iz izpita, ki so ga opravljali na papirju. Vsaka domača naloga je bila vredna po 10 točk, vendar pa smo pravilni rešitvi namenili le 4 točke. Preostalih 6 točk je študent lahko prejel na vajah v tednu po izteku roka za oddajo, in sicer tako, da je sprogramiral tri ločene nadgradnje domače naloge, urejene po naraščajoči težavnosti. Vsaka nadgradnja je prinesla po dve točki.

[1] Moška oblika se v prispevku uporablja nevtralno za oba spola.

Plagiatorstvo je tako izgubilo precejšen del svoje privlačnosti; če je študent rešitev prepisal, ne da bi jo razumel, je imel pri programiranju nadgradenj težave.

Začetno navdušenje nad sistemom pa je kmalu začelo plahneti. Prvič, težavnost implementacije nadgradnje je pogosto odvisna od implementacije osnovne naloge. Drugič, ker so študentje nadgradnje programirali na vajah in ker letno izvajamo 16 ali 17 skupin vaj, smo morali vsakokrat pripraviti več različnih domačih nalog in še več kompletov nadgradenj. Ponavadi smo pripravili štiri ali pet domačih nalog (za vsak dan izvajanja vaj po eno) in po dve, tri ali celo štiri komplete nadgradenj za vsako nalogo. Pri takšni količini je bilo težko zagotoviti že enakovrednost domačih nalog, kaj šele nadgradenj.

Slabosti so se pokazale tudi pri samem ocenjevanju. Od 90 minut, kolikor traja termin vaj, so študentje za programiranje nadgradenj imeli na voljo 60 minut, preostalih 30 minut pa je bilo namenjenih ocenjevanju. Asistent je tako posameznemu študentu namenil v povprečju manj kot dve minuti, kar pomeni, da je vsako nadgradnjo le enkrat pognal in morda na hitro preletel programsko kodo. Ker nismo imeli konsistentnih kriterijev (kaj šele avtomatiziranih testov), je na ocenjevanje dostikrat vplival halo-učinek. Tistemu, ki si je pridobil status »dobrega« študenta, smo (morda nehote) pogledali skozi prste in se zadovoljili z nedokončano rešitvijo nadgradnje, saj smo predpostavljali, da bi jo rešil do konca, če bi imel na voljo kakšno minuto več.

Poleg naštetega nas je čedalje bolj motilo dejstvo, da so trije termini vaj letno namenjeni izključno ocenjevanju. Bolje bi bilo, da bi v tistem času reševali naloge in tako pripravljali študente na izpit in poklicno življenje. Naposled smo se odločili, da navedene slabosti odpravimo, in tako smo v študijskem letu 2014/15 uvedli ocenjevalni sistem, ki ga še vedno uporabljamo.

4 Ocenjevanje po prenovi

4.1 Pregled sprememb

Glavna sprememba, ki jo je prenesla celostna prenova ocenjevalnega sistema, je uvedba *avtomatskega ocenjevanja*. Rešitve se po novem ne ocenjujejo več z ročnim poganjanjem in pregledovanjem programov, ampak z orodjem, ki program avtomatsko oceni na podlagi vnaprej pripravljenih testnih primerov [7]. Na ta način smo izločili subjektivni faktor ocenjevanja, poleg tega pa smo omogočili bistveno temeljitejše preverjanje rešitev, saj asistent na vajah praviloma ni imel časa, da bi preveril vse mogoče robne primere.

Vse naloge na vajah, domače naloge, izpitne naloge in neobvezne dodatne naloge so sedaj konsistentno oblikovane in opremljene z naborom testnih primerov. Izpit se po novem rešuje na računalnik, ocenjuje pa se na popolnoma enak način kot domače naloge.

Da bi povečali konsistentnost težavnosti nalog, smo se nadgradnjam odpovedali, zato pa smo število domačih nalog povečali s 3 na 10. Ta sprememba je skladna z že omenjeno smernico, ki vzpodbuja sprotno delo študentov.

4.2 Tipi nalog

Vsaka naloga pripada enemu od sledečih tipov:

Vhod-izhod. Pri nalogah tega tipa študentov program prebere svoje vhodne podatke s standardnega vhoda, rezultate pa izpiše na standardni izhod. Vsak testni primer je sestavljen iz datoteke s podatki, ki tvorijo vhod, in datoteke s podatki, ki tvorijo pripadajoči pričakovani izhod. Ocenjevalno orodje posreduje vsebino vhodne datoteke na standardni vhod študentovega programa, standardni izhod programa pa preusmeri v posebno datoteko. Ta datoteka se nato primerja z datoteko s pričakovanim izhodom. Testni primer se šteje kot pravilno obravnavan natanko tedaj, ko sta datoteki enaki.

Razred-izhod. Pri nalogah tega tipa študentje ne napišejo samostojnega programa, ampak enega ali več (neizvršljivih) razredov.[2] Vsak testni primer je sestavljen iz datoteke z izvršljivim testnim razredom in datoteke s pripadajočim pričakovanim izhodom. Testni razred ustvari enega ali več objektov razredov, ki tvorijo rešitev naloge, nato pa nad njimi kliče posamezne metode in rezultate klicev izpisuje na standardni izhod. Ocenjevalno orodje požene vsak testni razred posebej in standardni izhod preusmeri v posebno datoteko, njena vsebina pa se primerja z vsebino datoteke s pričakovanim izhodom. Testni primer se šteje kot pravilno obravnavan natanko tedaj, ko se vsebini ujemata.

Grafika. Pri nalogah tega tipa študentje dopolnijo vnaprej pripravljeno ogrodje tako, da se bo ob zagonu na grafično podlago (to je bodisi okno bodisi slikovna datoteka) narisal zahtevani vzorec. Študentje napišejo metodo, ki sprejme širino in višino podlage ter objekt javanskega razreda `Graphics2D`, s katerim je na podlago mogoče risati. Vsak testni primer je sestavljen iz izvršljivega razreda in pripadajoče referenčne slikovne datoteke. Ocenjevalno orodje požene izvršljivi razred, ta pa ustvari slikovno datoteko in pokliče študentovo metodo, da vanjo nariše vzorec. Takó izdelana datoteka se nato primerja z referenčno slikovno datoteko.

Za razliko od nalog tipov vhod-izhod in razred-izhod se posamezni testni primeri ne ocenjujejo zgolj z enico (popolno ujemanje) ali ničlo. Namesto tega se referenčna slika R in slika S, ki jo izdela študentova metoda, primerjata po posameznih pikah (slikovnih elementih), in to tako, da imajo vse *barve* enako težo; vsako konceptualno komponento slike je pri teh nalogah namreč treba narisati v svoji barvi.

Oceno ujemanja slik R in S izračunamo po formuli $\min\{s_{RS}, s_{SR}\}$, pri čemer

[2] Razred je mogoče pognati, če vsebuje metodo `main`.

$$s_{RS} = \frac{1}{C_R} \sum_{p=1}^{w_R} \sum_{q=1}^{h_R} \frac{t_{RS}(p,q)}{N_R(R(p,q))},$$

$$s_{SR} = \frac{1}{C_R} \sum_{p=1}^{w_S} \sum_{q=1}^{h_S} \frac{t_{SR}(p,q)}{N_R(S(p,q))},$$

$$t_{IJ}(p,q) = \begin{cases} 1 & \text{če } \exists u, v \in \{-1,0,1\}: \\ & \qquad J(p+u,\, q+v) = I(p,\, q) \\ 0 & \text{sicer} \end{cases}$$

Oznaka $I(p,\, q)$ predstavlja barvo pike na koordinatah $(x,\, y) = (p,\, q)$, oznaka $N_I(b)$ število pik barve b na sliki I, oznaka C_I pa število različnih barv na sliki I. Vse barve na referenčni sliki imajo enako skupno težo: če je, denimo, referenčna slika sestavljena iz r rdečih in z zelenih pik, ima vsaka rdeča pika težo $1/(2r)$, vsaka zelena pa težo $1/(2z)$. Količina $t_{IJ}(p,q)$ je enaka 1 natanko tedaj, ko se na položaju $(p,\, q)$ na sliki J ali v neposredni okolici tega položaja nahaja pika, ki ima enako barvo kot pika na položaju $(p,\, q)$ na sliki I. Odstopanje za eno piko v vsako smer dopuščamo zaradi morebitnih zaokrožitvenih napak.

Pri vseh treh tipih nalog ima študentov program na voljo le omejeno količino časa, denimo 1 sekundo za vsak testni primer. Če program do izteka časovne omejitve ne proizvede pravilnega izhoda, se izvajanje prekine, testni primer pa se šteje kot napačno obravnavan.

4.3 Zgradba nalog

Vse naloge (naloge na vajah, domače, dodatne in izpitne) imajo konsistentno zgradbo. Pri nalogah tipa vhod-izhod pričnemo z opisom problema, sledi natančna specifikacija vhoda in izhoda, zaključimo pa s konkretnim primerom vhoda in izhoda. Podatkovnih struktur ne predpisujemo, le pri (zelo redkih) nalogah kaj prepovemo. Na primer, kadar se v nalogi števila obravnavajo po števkah (npr. izračunaj vsoto števk podanega števila), prepovemo rabo tabel, nizov in podobnih struktur, ki programerju omogočajo, da se izogne aritmetičnim operacijam.

Pri nalogah tipa razred-izhod podamo specifikacije javno dostopnih konstruktorjev in metod, ki jih morajo študentje implementirati v svojih razredih. Za vsak tak element podamo tipe parametrov in morebitne omejitve ter pričakovani rezultat oziroma učinek. Privatnih elementov razreda (npr. atributov) ne predpisujemo.

Pri grafičnih nalogah študentje dopolnijo metodo, ki sprejme širino in višino grafične podlage ter objekt za risanje na podlago. V besedilu naloge opišemo posamezne grafične elemente ter njihove barve in relativne položaje in velikosti (glede na širino in višino podlage).

4.4 Struktura testnih primerov

Vsaki nalogi pripada po 10 javnih in 50 skritih testnih primerov. S pomočjo javnih primerov lahko študentje preizkušajo svoje rešitve, skriti pa so namenjeni ocenjevanju, zato jih objavimo šele po izteku roka za oddajo domače naloge. Testni primeri so razdeljeni na skupine različnih težavnostnih stopenj. Razmerja med skupinami v množici javnih primerov so enaka kot v množici skritih. Na primer, če je množica javnih primerov razdeljena na skupino treh najlažjih primerov, skupino petih srednje težkih primerov in skupino dveh najtežjih primerov, bodo v množici skritih primerov pripadajoče skupine obsegale 15, 25 in 10 primerov.

Pri nekaterih nalogah je težavnost določena predvsem z obsegom dela (za več pravilno obravnavanih testnih primerov in s tem za boljšo oceno je treba več narediti), pri nekaterih pa višja številka testnega primera pomeni trši oreh. Težavnost lahko marsikdaj uravnavamo z omejitvami vhodnih podatkov: za lažje testne primere določimo ostrejše, za težje pa milejše omejitve. Na ta način za nižjo oceno zadošča rešiti poseben, lažji problem, za višjo pa splošnejši (in tako težji) problem.

Oglejmo si razdelitev testnih primerov pri sedmi domači nalogi v sezoni 2014/15. Pri tej nalogi so študentje napisali program, ki s standardnega vhoda prebere število vrstic (m), število stolpcev (n) in dvojiško matriko velikosti $m \times n$, na standardni izhod pa izpiše število 4-povezanih skupkov enic v dvojiški matriki. Testni primeri so bili razdeljeni v šest skupin:

1. Vsak skupek je sestavljen iz ene same enice.
2. Skupki imajo obliko vodoravnih in navpičnih daljic.
3. Skupki imajo obliko polnih pravokotnikov.
4. Vsaka vrstica skupka je sestavljena iz enega samega strnjenega zaporedja enic.
5. Skupki so lahko poljubne oblike.
6. Skupki so lahko poljubne oblike, poleg tega pa velja $m, n \in [1, 1000]$ (za vse ostale skupine velja $m, n \in [1, 100]$).

Kot vidimo, so lastnosti lažjih testnih primerov vsebovane v lastnostih težjih; lažji primeri so tako zgolj posebni primeri težjih. Skupek, ki je sestavljen iz ene same enice, je hkrati tudi daljica in pravokotnik. Njegova edina vrstica je sestavljena iz enega samega zaporedja enic, seveda pa sodi tudi v kategorijo poljubnih oblik.

Če želimo rešiti peto skupino, moramo implementirati vsaj rekurzivni algoritem poplavljanja (angl. flood fill), pri šesti pa niti to ni dovolj, saj nam zaradi velikosti vhodne matrike pri izvajanju zmanjka sklada. Zato popolna rešitev vključuje iterativni algoritem poplavljanja. Kdor se zadovolji z nižjim številom točk, pa lahko izbira med različnimi *ad hoc* pristopi. Za rešitev prve skupine testnih primerov, denimo, zadošča, da preštejemo enice v vhodni matriki.

5 Izkušnje

Vpliv celostne prenove ocenjevanja težko kvantificiramo, še težje pa sklepamo, kaj lahko pripišemo prenovi, kaj pa drugim faktorjem. Rezultati izpitov so v zadnjih letih

boljši kot v preteklosti, vendar pa današnje izpite, ki potekajo na računalniku in se avtomatsko ocenjujejo, težko primerjamo z nekdanjimi papirnimi. Druga kvantitativna mera, na katero bi se morda lahko oprli, so študentske ocene. Tudi te se v splošnem izboljšujejo (avtor prispevka je za leto 2016 prejel celo študentsko nagrado »naj asistent na univerzitetnem študiju«), žal pa ne vemo, v kolikšni meri lahko večje zadovoljstvo z učnim osebjem pripišemo prenovi, v kolikšni pa aktivnejšemu načinu izvajanja vaj, upoštevanju študentskih mnenj itd.

Če se omejimo na kvalitativna merila, lahko trdimo, da nas je prenova odrešila številnih problemov, s katerimi smo si belili glavo v preteklosti. Sedaj ni več pripomb glede različne težavnosti domačih nalog, saj je vsaka domača naloga za vse študente enaka. Avtomatsko ocenjevanje je po eni strani neusmiljeno, saj ne pozna »skoraj« pravilnih rešitev, po drugi strani pa študente navaja na natančno sledenje specifikacijam. (Javni testni primeri to krutost sicer precej ublažijo.) Glavna prednost avtomatskega ocenjevanja pa je njegova objektivnost. Če so ogledi v času papirnih izpitov lahko trajali tudi dve uri, saj se je marsikdo skušal za oceno še pogajati, je ta vidik ocenjevanja po uvedbi avtomatizacije povsem izginil. Skrite testne primere objavimo po izteku roka za oddajo rešitev izpitnih nalog, tako da si študentje lahko oceno nemudoma izračunajo sami.

Ker se domače naloge po novem ne preverjajo več na vajah, ampak na službenih računalnikih asistentov, imamo vseh 14 tednov v semestru na voljo za reševanje nalog in pripravo na izpit. Zaradi konsistentne zgradbe nalog so študentje do izpita vsaj po tehnični plati (delo z vhodom in izhodom, poganjanje orodja za testiranje ...) že zelo dobro usposobljeni, zato na izpitu nimamo tovrstnih težav. Redne domače naloge študente silijo v sprotno delo, kar se komu morda zdi omejujoče, vendar pa po drugi strani v zadnjih letih precej redkeje opažamo študente, ki bi se še dan pred izpitom mučili z zanko za izpis prvih desetih naravnih števil. Z ocenjevanjem domačih in izpitnih nalog ni več skoraj nobenega dela; treba je zgolj pognati program, ki vse postori sam. Sicer je res, da moramo več truda vložiti v pripravo *posameznih* nalog in testnih primerov, a glede na to, da moramo sedaj na leto pripraviti bistveno manj nalog kot pri prejšnjem režimu ocenjevanja, smo časovno še vedno na boljšem.

Za največjo pomanjkljivost novega sistema se je izkazalo plagiatorstvo. Čeprav je število študentov, ki so nalogo vsaj enkrat prepisali ali ponudili v prepisovanje, od začetka do danes nekoliko upadlo (od 72 v sezoni 2014/15 do 49 v sezoni 2017/18) in čeprav primere prepisovanja s pomočjo sistema Moss[3] dosledno odkrivamo in (morda preblago) kaznujemo, problem še zdaleč ni izkoreninjen. Poleg tega gotovo obstajajo tudi študentje, ki jih ne zazna noben sistem za odkrivanje plagiatorstva, saj jim domačo nalogo reši nekdo izven letnika. Take študente lahko odkrijemo kvečjemu na vajah, če naletimo na očitno neskladje med njihovim samostojnim delom v laboratoriju in ocenami domačih nalog. Problem bi lahko do neke mere rešili z dodatnim preverjanjem razumevanja rešitev, vendar pa bi za to potrebovali čas in pripravljenost na »spopadanje« s problematičnimi študenti.

[3] `http://theory.stanford.edu/~aiken/moss/`

6 Zaključek

V članku smo predstavili prenovo ocenjevalnega sistema pri uvodnem programerskem predmetu na ljubljanski Fakulteti za računalništvo in informatiko. Poleg avtomatizacije ocenjevanja smo popolnoma prenovili sistem izdelave programerskih nalog. Naloge so sedaj konsistentno strukturirane in opremljene s testnimi primeri, razdeljenimi v težavnostne skupine. Ocenjujemo, da je prenova prinesla veliko pozitivnih učinkov. Edini zares negativni je po našem mnenju plagiatorstvo, seveda pa je to izziv za vse sisteme, v katerih se domače naloge ocenjujejo.

Če odštejemo tiste redke naloge, v katerih prepovemo uporabo tabel in nizov, imajo študentje pri reševanju domačih in tudi izpitnih nalog popolno svobodo; edini pogoj je ta, da program za podani vhod ali testni razred znotraj predpisanih časovnih okvirov proizvede pričakovani izhod. Pri četrti nalogi v sezoni 2014/15 pa smo manjši del točk namenili tudi »eleganci« rešitev. Naloga je zahtevala implementacijo razredov `Tocka` in `Premica`, ki sta bila zasnovana tako, da je bilo večino metod mogoče sprogramirati s klici že napisanih metod. Na primer, v metodi za izračun pravokotne projekcije točke na premico smo si lahko pomagali z metodo za izračun parametrov pravokotnice skozi točko in metodo za izračun presečišča premic. Število točk za »eleganco« se je izračunalo na podlagi števila pričakovanih klicev, ki so jih vsebovale posamezne metode. Tovrstnih zamisli kasneje nismo več preizkušali, kljub temu pa predstavljajo zanimivo temo za prihodnje raziskovalno delo.

Literatura

[1] K. Zupanc, Z. Bosnić, "Automated essay evaluation with semantic analysis," *Knowledge-Based Systems*, vol. 120, str. 118–132, 2017.

[2] J. Hollingsworth, "Automatic graders for programming classes," *Communications of the ACM*, vol. 3, no. 10, str. 528–529, 1960.

[3] K. Ala-Mutka, "A survey of automated assessment approaches for programming assignments," *Computer Science Education*, vol. 15, no. 2, str. 83–102, 2005.

[4] R. Romli, S. Sulaiman, K. Zamli, "Automatic programming assessment and test data generation - a review on its approaches," in *Information Technology (ITSim) 2010, Kuala Lumpur, Malaysia*, 2010, str. 1186–1192.

[5] V. Pieterse, "Automated assessment of programming assignments," in *Computer Science Education Research Conference (CSERC) 2013*, 2013, str. 45–56.

[6] L. Fürst, V. Mahnič, "Introductory programming course: motivating students with prior knowledge," *World Transactions on Engineering and Technology Education*, vol. 11, no. 4, str. 400–405, 2013.

[7] M. Poženel, L. Fürst, V. Mahnič, "Introduction of the automated assessment of homework assignments in a university-level programming course," in *MIPRO 2015, Opatija, Croatia, 2015*, 2015, str. 761–766.