Programozási feladatok kiértékelése távoktatásban

Automated grading of programming assignments for distance learning

Dr. HORNYÁK Olivér

egyetemi docens.
Miskolci Egyetem, Informatikai Intézet
Miskolc-Egyetemváros, Magyarország, +36 46 565-111, www.uni-miskolc.hu

Abstract

Online programming courses are extremely popular nowadays. Coding exercises could generate a great amount of work for the teachers thus result a big overload in their work. The practical part of learning programming skills required individual, personal coding assignments. In this paper the conception will be presented that allows the automated evaluation of coding assignments, which is the essential part of teaching computer science.

Keywords: distance learning, computer, programming, education, grader

Kivonat

Az online programozási kurzusok manapság nagyon népszerűek. A kódolási feladatok kezelése időigényes lehet a tanárok számára, így nagy túlterheltséget eredményezhetnek a munkájuk során. A programozási ismeretek elsajátításának gyakorlati része egyéni, személyes kódolási feladatokat igényel. Ebben a cikkben bemutatásra kerül az a koncepció, amely lehetővé teszi a kódolási feladatok automatizált kiértékelését, amely a számítástechnika oktatásának lényeges része.

Kulcsszavak: távoktatás, számítógép, programozás, oktatás, kiértékelő

1. BEVEZETÉS

Az utóbbi évek pandémiás helyzete számos változást kényszerített ki az oktatás terén. A személyes oktatást egyik pillanatról a másikra felváltotta az online oktatás. A körülmények számos változást kényszerítettek ki. Az informatika oktatásban talán nem volt annyira drasztikus és fájó a változtatás kényszere. A tárgy természeténél fogva az infrastruktúra többé-kevésbé rendelkezésre állt: az oktatók jellemzően rendelkeztek számítógéppel, azokon jellemzően telepítve voltak az oktatott tárgyhoz kapcsolódó elengedhetetlen szoftverek. Az oktató gépén elkészültek és rendelkezésre álltak a tárgyhoz tartozó prezentációk. Az oktatók rendelkeztek internet eléréssel. Talán a webkamera volt az egyetlen olyan hardver elem, amelyből hirtelen hiány keletkezett és a beszerzések megdrágultak, lelassultak.

Az informatikai tudásátadás technikai része gyakorlatilag zökkenőmentesen tudott megvalósulni. A számonkérésnél azonban nagyon könnyű volt az oktatói munkát túlterhelni. Az informatika oktatás gyakorlati részét egyéni feladatokkal látszott célszerűnek megoldani – és itt a hangsúly az egyéni jellegen van. Ha egy csoport ugyanazt a feladatot kapta, akkor bizony nagy számban születtek olyan megoldások, amelyeken látszott, hogy másolással keletkeztek. Mivel a feladatbeadás sem személyesen történt, még az izgulósabbak is bevállalták a csalás kockázatát. Az egyéni feladatok kiértékelése viszont a beadási határidőhöz közeledve hirtelen túlterhelte az oktatót. Nehéz volt mind volumenében ellátni a feladatot, mind minőségében megtartani az egységes értékelést, a jobbító szándékú kritikai észrevételek megfogalmazását.

Ebben a cikkben bemutatok egy olyan koncepciót, amely lehetővé teszi az informatikai oktatás hangsúlyos részét képező programozási beadandó feladatok automatikus értékelését. A módszert egy népszerű tanulásmenedzsment rendszerbe (Learning Management System) integrálva helyezve implementáltuk, és használjuk.

2. TÁVOKTATÁS

A távoktatás környezetben hatékonyan meg lehet valósítani a tudásátadást. A tapasztalat azt mutatja, hogy vannak jó praktikák, amelyek követése mind a hallgatók mind az oktatók részéről hozzájárulhat a sikerez [Joo et al., 2013]. A feladatok különösen fontosak a tudásátadás szempontjából, hiszen segítik a tudás elmélyítését. A jól felépített feladatsorok az elméletben megszerzett tudást gyakorlati tapasztalattá konvertálják át. Úgy kell ezeket felépíteni, hogy a legkönnyebbtől a legnehezebbig tartsanak.

Az általánosan használt elv szerint a hallgatói motivációt úgy lehet fenntartani ha egy-egy tudásátadási blokk nem tart 15 percnél tovább [Bunce et. al., 2010]. A számítástechnika oktatása ettől eltérő szemléletet kíván. A programozási nyelvek által megkívánt szintaktikai szabályokat szükséges memorizálni, különben a programozási tevékenység nem lesz elég hatásos. Az algoritmizálási képesség egy másik fontos tényező. A hatékony programfejlesztőek ismernie kell azokat a parancsokat, amelyekből a programot fel lehet építeni. Mindezek elérése gyakorlati feladatok megoldásával lehetséges.

Az online számítástechnikai kurzusok úgy épülnek fel, hogy a hallgatóknak programozási feladatokat adnak [Pieterse, 2013], amelyeket a hallgatók megírnak, és teljes működő programot küldenek be. A programozási feladatok sajátossága az, hogy nem egyetlen jó megoldás létezik: ugyanazt az algoritmust számos különböző módon lehet implementálni. A tanulásmenedzsment rendszerbe épített grader (magyarul kiértékelő, osztályozó komponens) felel azért, hogy a beküldött feladatokat automatikus módon kiértékelje.

2.1. A kiértékelő komponens feladatai

A grader hatékonyan egy külső modulként valósítható meg egy tanulásmenedzsment rendszerben, hiszen ez a típusú feladat nem jelentkezik más jellegű tantárgyak oktatásánál. Technikai szempontból a grader egy teljes értékű fordító (compiler), amely képes a hallgató által beküldött kód lefordítására és végrehajtására. A külső program-kiértékelő modul tervezésének néhány szempontja:

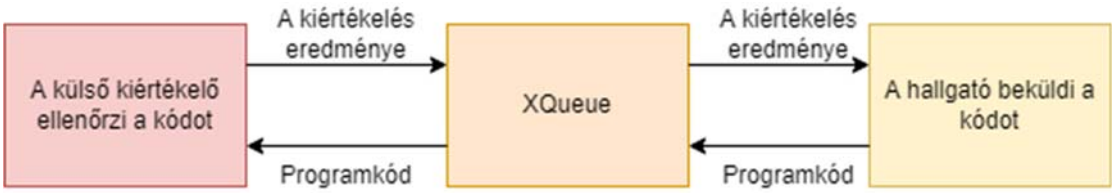
- a kiértékelőnek képesnek kell lennie a kód lefordítására. nem elegendő ha egy grader pusztán összehasonlítja a kódot szövegként egy előre definiált válasszal;
- az osztályozónak külön, elszeparáltan kell futtatnia a tanulók által beküldött kódot. Alice és Bob kódjának végrehajtásának teljesen függetlennek kell lennie;
- a rosszindulatú vagy rosszul megírt kód futtatása nem érintheti az LMS rendszert. Tehát amikor Alice megírja az első végtelen ciklusát, az LMS rendszernek működőképesnek kell maradnia, válaszolnia kell. És amikor Bob utasítja a számítógépet, hogy törölje le a merevlemez tartalmát, a kódot nem szabad végrehajtani az élő rendszeren;
- a hatékonyság fontos tényező: hallgatók százai küldik be kódjukat egyszerre. Az osztályozónak a lehető leggyorsabban sorba kell állítania a kéréseket és a válaszokat;
- virtuális környezetek támogatása. Érdemes lehet az LMS rendszerét úgy megírni, hogy több operációs rendszert is kiszolgálhasson. Előnyös, ha a külső grader virtuális környezetben futhat.

2.2. A külső kiértékelő modul technikai megtervezése

Egy programozást oktató LMS rendszer külső kiértékelő modulja, amely fogadja a hallgatók beküldött feladatait, feldolgozza ezeket, és visszajelzést ad. Szolgáltatásként fut, és az a fejlesztésekor előnyös, ha platformtól függetlenül külön is összeállítható és telepíthető. Az itt bemutatott platform az XQueue-n keresztül kommunikál a külső kiértékelővel. Ez biztosítja a tanulók kódjának továbbítását az graderhez; ezután megkapja az eredményeket az kiértékelőtől, és visszaküldi a tanulóknak. Gondoljunk bele: a kezdő programozó számos hibát elkövethet. A kód nem szükségszerűen működőképes.

A beküldött anyagokat az XQueue-nak nevezett tároló modulban gyűjtük össze, ahol addig maradnak, amíg az kiértékelő aktívan le nem kéri vagy le nem veszi a következő beküldött kódot az osztályozási sorból. A külső osztályozó rendszeres időközönként lekérdezi az XQueue-t egy RESTful interfészen keresztül [XQueue]. Amikor a külső osztályozó lekér egy beküldött anyagot, lefuttatja rajta a teszteket, majd a választ visszaküldi az XQueue-be a RESTful felületen keresztül. Az XQueue ezután elküldi a választ a Learning Management Systemnek. Ezt a folyamatot az aktív kiértékelő modulnak (pull grader) nevezzük.

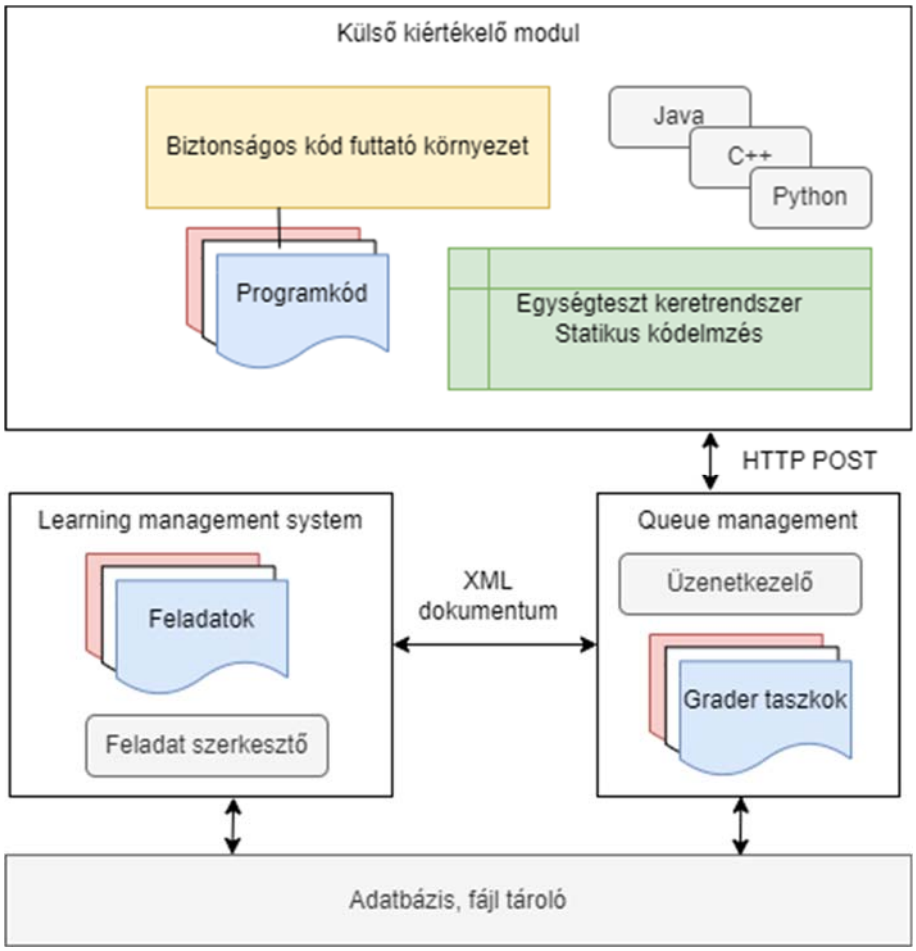
A passzív kiértékelő (push grader) megvárja, amíg az XQueue elküldi az ellenőrizendő feladatot, majd válaszol arra szinkronizálásnak megfelelő időközönként. Az LMS üzeneteket küld, amelyeket egy üzenetsor kezel. Az XQueue ellenőrzi a beállításokat, és meghatározza, hogy melyik feladathoz melyik URL társítható. Az üzenet a megfelelő URL-re lesz kézbesítve. A passzív osztályozó POST kérést kap az XQueue-tól, és szinkronban választ. Az LMS-re vonatkozó választ az XQueue fogja kézbesíteni a megfelelő URL-címen keresztül [Király et. al. 2017].



1. ábra. *A programozási feladat kiértékelése*

A benyújtott feladatot egy fájlként tároljuk, az, XQueue a felhőben vagy lokálisan egy megadott helyre feltölti, így annak egy egyedi URL-je keletkezik, amelyeket a saját adatbázisában is elmenti. A jó megoldásokat a rendszer tárolja, a külső kiértékelő nem kapja meg sem a hallgatók személyes adatait, a kiértékelés anonim.

A rendszer tartalmaz egy üzenet várólistát (RabbitMQ), amely az üzenetet megkapja XQueue-tól, és az előzetes konfiguráció alapján észlel, hogy az adatoknak el kell küldeni a külső kiértékelőnek. A RabbitMQ elolvassa az adatbázis adatait, és a szerializáció után HTTP üzenetet küld a külső kiértékelőnek, és megvárja a választ (passzív kiértékelőnek esetén). A grader elmenti a hallgatónak megküldendő válaszát egy fájlban. A programkód futtatásához ideiglenes állományok is keletkezhetnek, ezeket egy véletlenszerűen elnevezett ideiglenes mappában tároljuk.



2. ábra. *A külső kiértékelő működése*

A 2. ábrán látható, hogy a kiértékelő modul teljesen különálló módon képes futtatni a beküldött forráskódot. Az LMS rendszerrel nincs közvetlen kapcsolatban.

2.3. Egységteszt alapú kiértékelő modul

Az kiértékelés három típusú működés alapján valósulhat meg. Az első a kód szintaktikai hibától mentes működésének vizsgálata. Ezt a compiler elvégzi és az eredményét kiírja a standard outputra. Például a következő kódban elgépelés van:

```
// Check for syntax errors
class HelloWorld {
    public static void main(String[] args) {
        System.out.prrintln("Hello, World!");
    }
}
```

Ezt a fordító jelzi:

```
/tmp/YCfoCNJbUa/HelloWorld.java:5: error: cannot find symbol
        System.out.prrintln("Hello, World!");
                   ^
  symbol:   method prrintln(String)
  location: variable out of type PrintStream
1 error
```

A kiértékelő modul észleli, hogy a kiírt válaszüzenetben szerepel az error szó, és válaszüzenetben visszaadja, hogy a megoldás hibás, valamint továbbítja a hiba szövegét.
A másik megoldás a forráskód szöveges kiértékelése. Például szeretnénk ellenőrizni, hogy a beküldött feladat tartalmaz-e egy szintaktikailag megfelelő **for** ciklust tetszőleges változónevekkel. Ennek ellenőrzésére reguláris kifejezéseket alkalmazhatunk. Például a

```
for
\(\w*\s*\w*\s*\=\s*\w*\s*\;\w*\s*\w*\s*(\<|\>|\>\=|\<\=|\=\=)\s*\w*\s*\;\s*\w*\s
*(\+\+|\-\-\s*|\+\=\s*\w*|\-\=\s*\w*|\=\s*\w*)\s*\)
```

Reguláris kifejezés illeszkedik a következő programsorra:

```
for (int i = 1; i < 5; i++)
```

Azaz kiszűrhetők azok a megoldások, amelyekben ez nem szerepel. Általánosan tetszőleges statikus kódelemzési feladatot el tudunk végezni.

A beküldött program szematikai elemzését egységtesztekkel (unit test) tudjuk ellenőrizni. A feladat kidolgozója előre megírja az egységteszteket, és a benyújtott programokat ezek futtatják. Például, ha a feladat az volt, hogy egy for ciklussal határozzuk meg az első 5 természetes szám szorzatát egy for ciklus segítségével, akkor a lefuttatandó egységteszt a következő programsor lesz:

```
assertEquals(120, actual);
```

Ehhez természetesen a futtató környezetben rendelkezésre kell állnia az egységtesztek futtatásához szükséges könyvtáraknak, mint például a JUnit. A JAVA nyelv lehetővé teszi a futási időben történő, dinamikus osztálykiértékelést is. Ezzel nagyon komplex feladattípusok is egységtesztelhetők, például megvizsgálhatjuk, hogy létrejött-e egy adott nevű osztály, van-e adott nevű interfésze, vannak-e megadott nevű és típusú mezők, sőt akár a módosítók is.

3. ÖSSZEFOGLALÁS

Ebben a cikkben a programozási feladatok értékelésének értékelésének koncepciója került bemutatásra. A módszert a Java programozási nyelvre valósítottuk meg, de kiterjeszthető bármely olyan programozási nyelvre, amely támogatja az egységtesztelést. A tervezés során figyelembe vettük, hogy egyidejűleg nagy számú feladatbeadás is lehetséges legyen. További szempont volt az, hogy a kiértékelés Példákon keresztül szemléltettük, hogy az automatikus kiértékelés során milyen kihívások merülnek fel, és egy virtualizálható környezetben fusson le. Előnye és újszerűsége a módszernek, hogy lehetővé teszi egyéni feladatok kiadását is, akár olyan módon is, hogy egy generátor állítja elő a feladatokat és az készíti el a kiértékelés alapjául szolgáló egységteszteket is. Példákkal bemutatjuk, hgy a kidolgozás során a módszert hogyan implementáltuk. A bemutatott rendszert kipróbálni a www.memooc.hu oldalon elérhető Java programozási kurzusokon lehet.

IRODALMI HIVATKOZÁSOK

[1] Király, S., Nehéz, K., Hornyák, O.: *Some aspects of grading Java code submissions in MOOCs*. Research in Learning Technology, 2017, 25. https://doi.org/10.25304/rlt.v25.1945

[2] Bunce, D. M., Flens, E. A., Neiles, K. Y.: *How long can students pay attention in class*? *A study of student attention decline using clickers*, Journal of Chemical Education, 2010, vol. 87, no. 12, pp. 1438–1443. doi: 10.1021/ed100409p.

[3] Joo, Y. J., Joung, S. Kim, E. K.: *Structural relationships among e-learners'sense ofpresence, usage, flow, satisfaction, and persistence*, Educational Technology & Society, 2013, vol. 16, no. 2, pp. 310–324

[4] Pieterse, V.: *Automated assessment of programming assignments*, Proceedings of the 3rdComputer Science Education Research Conference on Computer Science Education Research, ACM Digital Library 2013, New York, pp. 45–56.

[5] Robins, A., Rountree, J., Rountree, N.: *Learning and teaching programming: A review and discussion*, Computer Science Education, 2003, vol. 13, no. 2, pp. 137–172.

[6] Staubitz, T., Klement, H., Renz, J., Teusner, R., Meinel, C.: Towards practical programming exercises and automated assessment in Massive Open Online Courses, IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE), IEEE, New York, 2015., pp. 23–30.

[7] XQueue, https://github.com/openedx/xqueue (2022.10. 02)