Multidiszciplináris tudományok, 10. kötet. (2020) 2 sz. pp. 469-480 **https://doi.org/10.35925/j.multi.2020.2.52**

KÓDMINŐSÉG MÉRÉSE A PROGRAMOZÁS TANULÁSÁT TÁMOGATÓ RENDSZEREKBEN

Fekete Antal Kristóf
BSc-hallgató, Miskolci Egyetem, Informatikai Intézet, Általános Informatikai Intézeti Tanszék
3515 Miskolc, Miskolc-Egyetemváros, e-mail: antal.fekete98@gmail.com

Absztrakt

A cikkben bemutatásra kerül néhány programozástanulást támogató online rendszer, amelyek a kezdő programozók számára nyújtanak segítséget a programozás otthoni, önálló gyakorlásához. Az alkalmazások tesztelése során összegyűjtésre került ezek funkcionalitása és a felsőfokú oktatásba való integrálás tekintetében hozzáadásra került két lényeges követelmény: a feladatmegoldáshoz használt idő mérése és a kód minőségi jellemzése. Az ezen elvárások figyelembevételével létrehozott webes alkalmazás használata a cikk végén kerül bemutatásra egy példán keresztül.

Kulcsszavak: *programozásgyakorlás, automatikus kódértékelés, kódminőségmérés*

Abstract

This paper presents some existing automatic code assessment tools that can help beginners to develop their programming skills in home settings. After analyzing their functionalities, two important requirements were identified to be added in order that these tools can be efficiently applied to support teaching programming at tertiary level. The paper ends with the introduction of the web-based application developed for code assessment, which involves the measuring of coding time and code quality as well.

Keywords: *learning to program, automatic code assessment, measuring code quality*

1. Bevezetés

A 2017-es OECD jelentés szerint Magyarországon a felsőoktatásban tanulók kevesebb mint a fele szerez diplomát a képzési időszak végén [1]. Ez az alacsony végzési arány az informatikus alapszakokon is jellemző, amelynek egyik tényezője az első féléves Programozás kurzus alacsony sikerességi rátája. Ennek magyarországi okait kutatva az egyik biztosan az, hogy a programozás elemeinek tanítása nem szerepel a közoktatásban kötelezően tanítandó területek között, ezért a felsőoktatásba belépő hallgatók közül sokan a tanulmányaik során találkoznak először programozási feladatokkal [2].

A programozás tanítása során kifejezett hangsúlyt kell fektetni a gyakorlati képzésre. Ennek a megvalósítása egyre nagyobb kihívások elé állítja a tanárokat, mert az oktatás hatékonysága nagyban függ a tanulók korábban elsajátított ismereteitől, egyéni képességeitől. Ideális körülmények között, azaz kis létszámú, homogén csoportok kialakítása mellett érhető el a leglátványosabb fejlődés a programozási készségek terén. Ehhez azonban vagy az oktatási rendszer kapacitásának bővítésére, vagy az egyéni fejlesztést támogató alkalmazások gyakorlati oktatásba való bevonására van szükség.

A programozástanulást támogató rendszerek napjainkban széles körben hozzáférhetők és a kezdő programozók számára nélkülözhetetlen segédeszközök [3]. Ebben a cikkben először bemutatok néhány otthoni tanulást támogató online alkalmazást, majd összegyűjtöm a velük szemben támasztott

követelményeket, melyek közül a kódminőség mérésére térek ki részletesen a 4. pontban. Végül ismertetem azt az alkalmazást, amit a Miskolci Egyetem Informatikai Intézete számára készítettem a C programozás gyakoroltatása céljából.

2. Programozás gyakoroltató rendszerek áttekintése

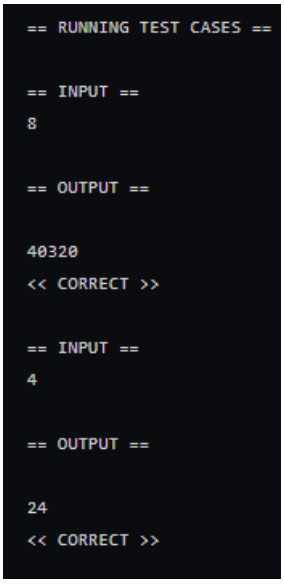
A következőkben négy programozás oktatását és gyakorlását segítő ingyenesen elérhető online szolgáltatást vizsgálok meg: CoderByte [4], CodinGame [5], Codeboard [6], C Puzzles [7]. Ennek során a C programozási nyelvre összpontosítok annak figyelembevételével, hogy mely mértékben és milyen lehetőségekkel támogatják a kezdő programozók fejlődését.

2.1. CoderByte

A https://www.coderbyte.com/ oldalon egy rövid regisztrációt követően néhány ingyenesen megtekinthető (angol nyelven narrált) videó áll a felhasználó rendelkezésére JavaScript, Python, valamint Ruby programozási nyelvekkel kapcsolatban. Ezek a videók a felsorolt programozási nyelvek alapműködését és használatát mutatják be teljesen kezdők számára.

A weblapon „Challenges” címszó alatt található néhány feladat, amelyeket számos programozási nyelven – ideértve természetesen a C programozási nyelvet is – meg lehet oldani a rendszerbe integrált fejlesztői környezet segítségével. A megírt programkódok beküldhetők, amelyekre az időráfordítást, továbbá a szemantikai ellenőrzés eredményét figyelembe véve pontot ad a rendszer. Amennyiben a felhasználó teljesen elakadna egy feladat megoldásával, lehetőség van a beküldött megoldásokat megtekinteni, kipróbálni.

A rendszert egy faktoriális számítását elvégző feladat megoldásával próbáltam ki, ennek során megállapítottam, hogy a szolgáltatás a begépelt programkódon a szintaktikai ellenőrzésen túl szemantikai vizsgálatot is végez tesztesetek futtatásával.



1. ábra. *Egész szám faktoriálisát számító program szemantikai tesztelése*

2.2. CodinGame

A https://www.codingame.com/start weblapon ugyancsak regisztrációt követően hasonlóan több programozási nyelv választható ki. A weboldal kissé játékossá teszi a programozást, mivel először egy olyan programkódot kell írni *printf()*, illetve *scanf()* függvények segítségével, amely az ellenséges hajókat időben megsemmisíti azáltal, hogy mindig a legközelebb lévő ellenségre lő. Később számos további példán keresztül gyakorolhatók akár C, akár más támogatott programnyelven a vezérlési szerkezetek, továbbá az alapalgoritmusok.

Amennyiben a felhasználó elakadna egy feladat megoldása során, a „hints” lehetőségre kattintva több tippet és segítséget is olvashat, ami segíti a programkód megírásában. Végső soron a kódolandó algoritmus angol nyelvű szöveges leírása, a pszeudokód, majd a tényleges megoldásként szolgáló programkód is megtekinthető. A CodinGame felület is végez szintaktikai, valamint szemantikai ellenőrzést.

Here's what your code should do at every turn of the game:

```
max ← 0
imax ← 0
for i starting from 0 to 7 do
    read mountain_h
    if mountain_h is greater than max then
        max ← mountain_h
        imax ← i
    end if

end for
print imax
```

2. ábra. *Maximum kiválasztó algoritmus pszeudo kódja*

2.3. Codeboard

A Codeboard ugyancsak egy webes felület (https://codeboard.io/), amely saját fejlesztői környezettel rendelkezik. Ezen keresztül létrehozhatók saját, tetszőleges projektek C programozási nyelven is, amelyeket egy link segítségével meg is lehet osztani másokkal. Emellett több mint kilencvenezer nyilvános – mások által készített és megosztott – projekt között lehet keresgélni és böngészni. A szolgáltatás elsősorban a megírt programkód fordítását és futtatását teszi lehetővé, a szemantikai ellenőrzés tetszőleges programkódok esetében nem támogatott.

Ugyanakkor lehetősége nyílik például egy oktatónak arra, hogy különböző feladatokat fogalmazzon meg a tanulók számára, ezek beküldéséhez időintervallum is rendelhető. A kiírt feladat beküldése során lehetőség van a feladványt kiadó felhasználó munkáját megkönnyítendő automatikus kiértékelést alkalmazni.

Automatizált pontozásra az egyik támogatott módszer egy úgynevezett eredménysztring használata. Ennek során a Codeboard beküldéskor lefordítja és futtatja az elküldött programkódot, majd ellenőrzi, hogy a program kimenete megegyezik-e egy speciális karakterlánccal. A kiértékelésre a másik lehetőség az egységtesztek alkalmazása, azonban a jelen cikkben hangsúlyozott C programozási nyelven ennek lehetősége nem biztosított [8].

További hátrányként értékeltem, hogy a Codeboard a fentebb bemutatott CoderByte és CodinGame alkalmazásokkal ellentétben nem biztosítja a legminimálisabb segítséget sem arra az esetre, ha a felhasználó egy feladat megoldása során megakadna.

2.4. C Puzzles

A C Puzzles (https://chortle.ccsu.edu/CPuzzles/) használatához nincs szükség még regisztrációra sem. Az oldal rövid, egyre nehezedő feladatokat („rejtvényeket") tartalmaz számos témakört érintve, mint például a ciklusok, random számok, egy- és kétdimenziós tömbök, láthatósági szabályok, pointerek, struktúrák, sztringek. A feladatok döntő része hozzásegíti a tanulókat a C programozási nyelv alapvető működésének megértéséhez. A különböző feladványok többségénél megtekinthető egy kódváz, amelyet a feladat sikeres teljesítéséhez csupán ki kell egészíteni.

Az oldal nem biztosít fejlesztői környezetet a feladatok megoldásához, ebből következően semmilyen automatizált ellenőrzésre nincs lehetőség. Ugyanakkor lehetőség van a feladatokhoz megtekinteni a javasolt megoldás forráskódját.

3. Programozást gyakoroltató rendszerekkel szembeni elvárások

Egy programozási alapismereteket oktató egyetemi kurzushoz kapcsolódóan egy gyakorlásra használható online rendszerrel szemben az egyszerűség és a hatékonyság mellett számos konkrétabb elvárást is támaszthatunk, mint például szintaktikai és szemantikai ellenőrzés, időmérés, valamint a programkód minőségével kapcsolatos visszajelzés.

3.1. Szintaktikai ellenőrzés

A szintaktikai ellenőrzés alapvető, szinte megkerülhetetlen funkció. Egy programkód sikeres fordításának feltétele az, hogy szintaktikailag helyes legyen. Ebből következően egy adott programozási nyelv elsajátítása a szintaktikai elemek, struktúrák és az alapalgoritmusok megértésével kell, hogy kezdődjön, mivel az adott programnyelv elméleti felépítésének és helyesírási formáinak ismerete elengedhetetlenül szükséges a széleskörű alkalmazhatósághoz. Egy kezdő programozónak mindezek nehézséget okozhatnak, mivel a szintaktika és a gondolkodásmód elsajátítása mellett az alapalgoritmusok funkcionális alkalmazására már kevesebb figyelmet tud fordítani [9].

3.2. Szemantikai ellenőrzés

A valós tartalmi, jelentés- és értelmezésbeli szabályok adják meg a szemantikai szabályokat [10]. A szemantikai ellenőrzés ugyancsak kulcsfontosságú, mivel ennek segítségével bizonyosodhatunk meg arról, hogy a program valóban azt csinálja-e, amit a feladatleírásban megfogalmaztak. Ezt tesztesetekkel lehetséges ellenőrizni egy már előállított és sikeresen fordított programkód alkalmazásával.

3.3. Időmérés

A programozás gyakorlása során lényeges visszajelzés, hogy egy adott feladatot, problémát mennyi idő alatt oldott meg a tanuló. Ez segít a vizsgára és az évközi számonkérésekre való felkészülésben, tekintve, hogy a számonkérések alkalmával mindenképpen egy adott időkereten belül kell a feladatot megoldani. Gyakorlás útján pedig fejleszthető az, hogy a tanuló minél rövidebb idő alatt képes legyen a különböző programozási feladatokat megoldani.

3.4. Kódminőséggel kapcsolatos visszajelzés

Az informatikus képzésben fontos szempont a tanuló képessége a magas színvonalú forráskód írására. Elvárjuk a kódolási szabvány betartását, az alapalgoritmusok helyes használatát, a kód hatékonyságát és redundanciamentességét. A statikus kódelemzés hasznos, mivel a programkódban olyan hibák feltárására alkalmas, melyeket más tesztelési eljárással nehéz megtalálni. Például az alkalmazás lefagyását eredményező null pointer hivatkozás statikus kódelemzés útján kiszűrhető. A statikus kódelemzés történhet kizárólag a forráskód alapján, továbbá a forráskód és a modell együttese alapján is. Előbbi esetében értelemszerűen kizárólag a forráskód kerül felhasználásra, míg utóbbinál a forráskódhoz tartozik egy modell is, amely elő- és utófeltételekkel, továbbá invariánsokkal leírja, hogyan kellene a programnak működnie [11]. A tanulók ezen képességének fejlesztése érdekében fontos motiválni őket, hogy a programozás-számonkéréseken is magas színvonalú programkódot tudjanak előállítani [12].

4. Kódminőség mérése a programozástanulást támogató rendszerekben

A kódminőség rendkívül fontos tényező a programozás során, hiszen a gyenge minőségű programkód biztonsági és egyéb kockázatokhoz vezethet. Egy programkód akkor tekinthető megfelelőnek, ha azt a problémát oldja meg, amelyet valóban kell, egységes stílust követ, könnyen megérthető, jól dokumentált és tesztelhető is. Nincs egyetlen és egységes módszer a kódminőség méréséhez, azonban létezik néhány kulcsfontosságú alapelv, melyek követése magasabb minőségű programkódot eredményez [13].

4.1. Megbízhatóság

Egy szoftver minősége egyenes arányosságban áll a megbízhatósággal. A termék rendelkezésre állása és a hibák száma befolyásolja legjelentősebben a megbízhatóságot. Ahhoz, hogy minél kevesebb legyen a hiba, a program futása során az előforduló problémák számát szükséges megbecsülni, majd ezen értéket csökkenteni. A problémák száma statikus kódelemzéssel állapítható meg, míg a rendelkezésre állás a meghibásodások közötti átlagidővel (MTBF – Mean Time Between Failures) mérhető [13]. Az MTBF a vizsgálat időtartama és az ezen időintervallum alatti meghibásodások számának hányadosaként számítható több modulból álló, komplex szoftverek tesztelése során [14]. Értelemszerűen minél nagyobb az MTBF értéke, annál megbízhatóbb és magasabb a rendelkezésre állása az adott szoftverterméknek.

4.2. Karbantarthatóság

A karbantarthatóság az a mérőszám, amely megmutatja, hogy milyen hatékonyan (egyszerűen és gyorsan) lehet egy szoftverterméket módosítani. A módosítások magukban foglalják a szoftver javítását, fejlesztését, a környezeti változásokhoz, követelményekhez és funkcionális előírásokhoz történő adaptálását. Karbantartásnak minősülnek továbbá a speciális támogató személyzet, az üzleti vagy üzemeltetési személyzet, továbbá a végfelhasználók által végrehajtott változtatások is. Emellett ide tartozik még a frissítések, javítások telepítése is [15]. A fenntarthatóság vonatkozik a szoftver méretére, a struktúrára, a kódbázis komplexitására és a következetességre is. A fenntartható forráskód biztosítása számos további tényezőn alapul, például tesztelhetőség, olvashatóság és érthetőség [13].

A forráskód bonyolultságát számos módszerrel lehet mérni. Ezek közül a Halstead-féle komplexitási metrikát mutatom be, mely a forráskód szövegelemzésén és információelméleti meggondolásokon alapul.

A Halstead-metrika négy alapmennyiségből indul ki: n_1 – különböző operátorok és írásjelek száma, n_2 – különböző operandusok száma, N_1 – az operátorok és írásjelek összesített száma, N_2 – az operandusok összesített előfordulási száma. Ezek segítségével lehet kiszámítani az alábbi, komplexitást jellemző mérőszámokat [16, 17]:

1. A program szókincse: a programban található különböző azonosítók számát jelenti, mely az alábbi képlettel kapható meg:

$$n = n_1 + n_2 \tag{1}$$

2. A program hossza: a programban található azonosítók összes előfordulásainak együttes számát jelenti. Képlete:

$$N = N_1 + N_2 \tag{2}$$

3. A program volumene: a szókincs információtartalma, vagyis a program (mint szöveg) kódolásához szükséges bitek számát jelenti. Képlete:

$$V = N \cdot n \tag{3}$$

4. A program nehézségi szintje: a program megírásához vagy megértéséhez kapcsolódó mutató:

$$D = \frac{\frac{n_1}{2} \cdot N_2}{n_2} \tag{4}$$

5. A program implementálásához szükséges erőfeszítés mértéke arányos a program volumenével és nehézségi szintjével:

$$E = V \cdot D \tag{5}$$

6. A program implementálásához szükséges idő: tapasztalati kísérletek alapján Halstead megállapította, hogy ha az erőfeszítést elosztjuk 18-cal, akkor megkapjuk a program implementálásához szükséges időt másodpercben:

$$T = \frac{E}{18} \tag{6}$$

7. Lehetséges hibák száma:

$$B = \frac{E^{2/3}}{3000} \tag{7}$$

4.3. Tesztelhetőség

A tesztelhetőség mérhető azzal, hogy hány teszteset szükséges, melyet a szoftver összetettsége és mérete határoz meg. A szükséges tesztesetek számának meghatározásához használható a ciklomatikus komplexitás, amely a forráskód alapján kiszámítja a lineárisan független útvonalak számát.

A ciklomatikus komplexitás fogalmát Thomas J. McCabe vezette be 1976-os publikációjában [18], melyben bemutatja, hogy a gráfelmélet hogyan alkalmazható programozási szempontból. A ciklomatikus komplexitás a lehetséges kimeneteket köti össze a megfelelő bemenetekkel. Értéke a

$$CYC = e - n + 2 \cdot p \tag{8}$$

formulával kapható meg, ahol

e a gráf éleinek száma (vezérlés átadása),
n a gráf csúcsainak száma (szekvenciális utasítások csoportja),
p az összefüggő komponensek száma (kilépési pontok száma).

4.4. Hordozhatóság

A hordozhatóság azt deklarálja, hogy egy szoftver mennyire platformfüggetlen, azaz hogyan támogatja a különböző környezetekben való futtatást ugyanaz a kód és mekkora mennyiségű energiabefektetéssel alakíthatók át a nem platformfüggetlen egységek szükség esetén úgy, hogy más környezetben is működőképes legyen. Akkor hordozható egy egysége a szoftvernek, ha az új platformra való adaptálás alacsonyabb költségigényű, mint az adott egység újrafejlesztése [19]. Környezet vagy platform alatt értendő egyebek mellett az operációs rendszer, a processzor és más fizikai környezet [20]. A hordozhatóságot a hordozási és újjáépítési költségek függvényével lehet értelmezni, amelyet egy adott célkörnyezethez viszonyítunk [19].

4.5. Újrafelhasználhatóság

Az újrafelhasználhatóság a szoftverek azon jellemzője, hogy a meglévő szoftver, komponens vagy programkód milyen mértékben használható fel újra más alkalmazások előállítása során [20]. Ezzel jelentős idő – és ebből következően költség – takarítható meg a fejlesztés során, mivel a már meglévő, ismételten hasznosítható komponenseket nem szükséges újból kifejleszteni.

Az újrafelhasználhatóságra építkeznek szoftverfejlesztési módszertanok is, például a komponens-alapú fejlesztésparadigma szerint megkísérelük újrafelhasználható komponensekből felépíteni az elkészítendő szoftverterméket. Az újrafelhasználás-orientált modell a meglévő és elérhető újrahasznosítható komponensekre, illetve azok egységes szerkezetbe történő integrációjára támaszkodik [21].

5. Az ME-IIT programozás gyakoroltató webes alkalmazása

A Miskolci Egyetemre felvett, majd gazdaságinformatikus, mérnökinformatikus vagy programtervező informatikus alapképzésre beiratkozott hallgatóknak az első szemeszterben egyebek mellett a számítógépes programozás alapjait szükséges elsajátítani, amihez a C programozási nyelvet használják fel. Ezen keresztül nyernek betekintést a programozás világába, ennek segítségével ismerkednek meg a programírás lépéseivel és az alapalgoritmusokkal, azaz a C programnyelv szintaktikai elemeit és működését tanulják meg először.

A lemorzsolódás csökkentése érdekében, a kezdő programozóknak készült a C programozási feladatokat gyakoroltató webes alkalmazás, amely az önálló felkészüléshez nyújt segítséget (https://github.com/fekete20/szakdolgozat/tree/master). A gyakorolni kívánt témakör kiválasztása után szintenként nehezedő programozási feladatokat kap a tanuló, amelyek megoldását a beviteli mezőben kell megadnia. A kód szintaktikai ellenőrzését a *GNU-GCC fordító* végzi, míg a szemantikai ellenőrzéshez az adott feladatra megírt tesztesetek fognak lefutni a szerveroldalon. Ez utóbbihoz a *CUnit* keretrendszer által biztosított eszközöket használtam fel. A kódminőséggel kapcsolatos visszajelzést három szoftver kimenetéből állítom elő. A statikus kódelemzést a *cppcheck* program, a Halstead-féle komplexitást mérő számokat a *Frama-C* keretrendszer, míg a ciklomatikus komplexitást a *pmccabe* program szolgáltatja. Ezek a számadatok önmagukban a kezdő programozók számára nem informatí-

vak, ezért a tanuló forráskódjára kiszámított értékeket összehasonlítom az adott feladathoz előgyártott mintamegoldás mutatóival, és a tanuló számára az eltérés mértéke is megjelenítésre kerül.

Ezen túlmenően reguláris kifejezések alkalmazásával megvalósítottam, hogy a változók, a struktúra és a szimbolikus konstansok deklarálása esetén a program ellenőrizze a névkonvenciók betartását, valamint ne engedje globális változó deklarálását: amennyiben a tanuló programkódja ezen elvárásokat nem teljesíti, úgy a felületen megjelennek azon programsorok számai, ahol a konvenciók betartása nem volt sikeres. Mindezeken felül a 3.3. pontban említett időmérést is megvalósítottam. A megoldás beküldése után a tanulónak megjelenik a feladat kérése és beküldése között eltelt időtartam másodpercben.

A rendszer működésének szemléltetéséhez tekintsük azt a feladatot, amely 1 és 100 ezer között megkeresi és kiírja az ikerprímeket. A mintamegoldás forráskódja az alábbi:

```c
#include <stdio.h>
#include <stdbool.h>

bool prim(int);

int main()
{
    int i;
    for(i = 1; i <= 100000; i++) {
        if(prim(i) && prim(i+2)) {
            printf("(%d, %d) ", i, i+2);
        }
    }
    return 0;
}

bool prim(int num) {
    int div = 2;
    bool found = false;
    while(div <= num/2 && !found) {
        if(num%div == 0) {
            found = true;
        }
        div++;
    }
    return !found;
}
```

3. ábra. *Ikerprímek keresése – mintamegoldás forráskódja*

A Halstead-komplexitás mérőszámai alapján a megoldás szókincse 158, hossza 519 karakter, volumene 3790,66 bit, nehézségi szintje 44,51, az igényelt erőfeszítés mértéke 168718,93, az időigénye 6373,27 másodperc. A ciklomatikus komplexitás értéke a main és a prim függvényeket tekintve egyaránt 4.

Az alábbi alternatív megoldás során több hibát is elkövettem: deklaráltam egy globális változót (j), egy lokális változót (I) pedig nagy kezdőbetűvel adtam meg. A kód komplexebbé is vált, mert a vizsgálandó számpárok előállítását két egymásba ágyazott ciklussal valósítottam meg.

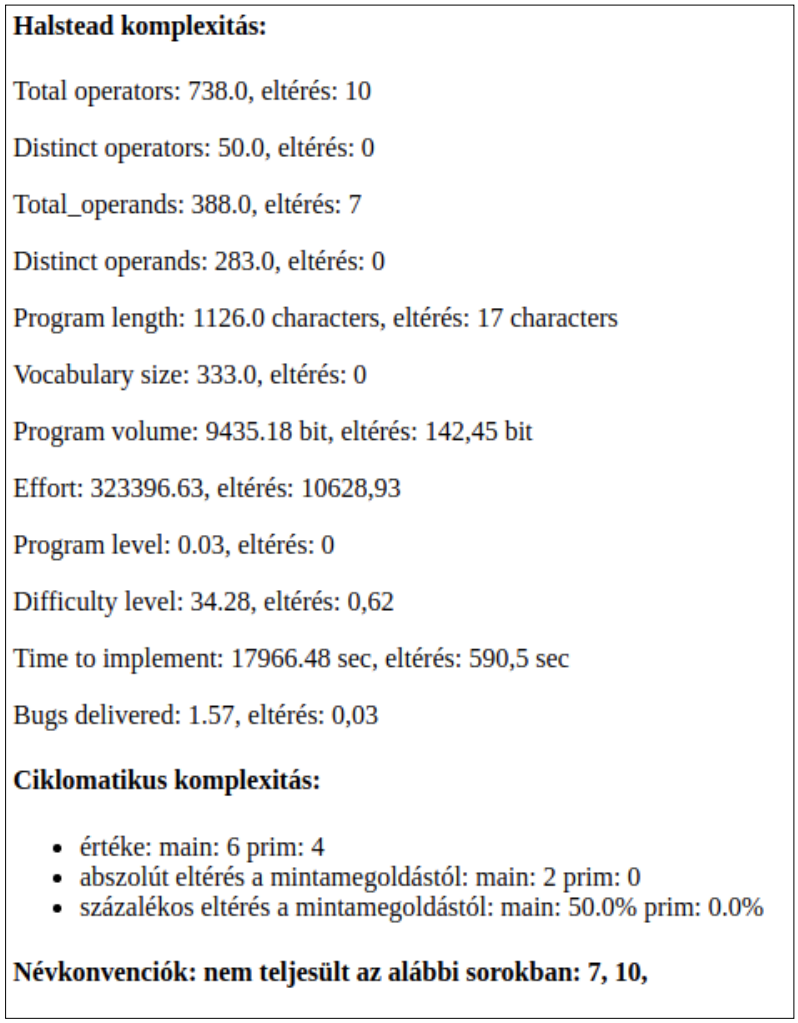
```c
#include <stdio.h>
#include <stdbool.h>
#include <math.h>

bool prim(int);

int j;

int main()
{
    int I;
    for(I=1; I<=100000; I++) {
        for(j=I+1; j<=100000; j++) {
               if (j-I == 2 && prim(I) && prim(j))
                 printf("(%d, %d) ", I, j);
        }
    }
    return 0;
}

bool prim(int num) {
    int div = 2;
    bool found = false;
    while(div <= sqrt(num) && !found) {
        if(num % div == 0) {
            found = true;
        }
        div++;
    }
    return !found;
}
```

4. ábra. *Tanuló megoldásának forráskódja*

A tanuló programkódjának futási eredményét az 5. ábrán látható képernyőkép mutatja. Az első hét mérőszám pozitív értéke, köztük az operátorok száma, a program hossza és volumene azt fejezik ki, hogy a megoldás tömörebben, kevesebb változó használatával és kevesebb programsorban leírható. A program nehézségi szintje az egymásba ágyazott ciklusok miatt növekedett. Következésképpen a feladat implementálásához több erőfeszítésre és időre volt szükség.

A ciklomatikus komplexitás értékének növekedése szintén az egymásba ágyazott ciklusokkal magyarázható, és azt mutatja, hogy a feladatot egyszerűbben is meg lehetett volna oldani.

Az értékelés végén azok a kódsorok vannak felsorolva, ahol a programozó nem tartotta be a konvenciókat: itt a globális változó és a nagybetűs lokális változó deklarációkra hívja fel a figyelmet.



5. ábra. *A kódminőséget jellemző mérőszámok (képernyőkép)*

6. Összefoglalás

A programozói készség csak gyakorlás útján fejleszthető. Ez a kezdeti stádiumban sok idő- és energiaráfordítást igényel a tanulóktól, aminek csak töredékét lehet megvalósítani tantermi keretek között. Ezért hasznosak az otthoni körülmények között is elérhető, önálló tanulást segítő programozást gyakoroltató, automatikus kód kiértékelő rendszerek. Áttekintve az ingyenes szoftverek kínálatát, arra a következtetésre jutottam, hogy ezek az alkalmazások alapszintű visszacsatolást adnak a felhasználó forráskódjával kapcsolatban, azaz szintaktikai és esetleg szemantikai ellenőrzésnek vetik alá. Ha azonban a cél a professzionális szintű programozói készség kialakítása, a forráskód további elemzésére is szükség van. Meg kell vizsgálni, hogy mennyire komplex a kód, milyen a karbantarthatósága, platform független-e és jól tesztelhető-e. Ezen vizsgálatok elvégzéséhez már léteznek szoftverek, amelyek különböző mérőszámokat határoznak meg és ezzel jellemzik a kód minőségét a felsorolt szempontok alapján. Bár ezeknek a metrikáknak az a célja, hogy számszerűen jellemezzék a felhasználó által írt

programkód minőségét, önmagukban nehezen értelmezhetők. Más megoldásváltozatok számértékeivel összehasonlítva viszont jelentősen megnő az információtartalmuk.

Az általam készített C programozás tanulást támogató webes alkalmazás funkciói között ezért a szintaktikai és szemantikai tesztelés mellett helyet kapott a kódminőség jellemzése a Halstead komplexitás metrikák, valamint a ciklomatikus komplexitás érték segítségével. A tanuló ezeknek a mérőszámoknak nemcsak az abszolút értékét látja, hanem azok előjeles eltérését is az előgyártott mintamegoldásra kiszámított értékekhez képest. Ezek mellett fontos tájékoztatni a tanulót a feladatmegoldásra fordított időről, mert a programozási készség fejlődését az időráfordítás csökkenése is mutatja.

A programozástanulás következő szintjén olyan szempontokat is figyelembe kell venni, mint például a kód olvashatósága és újra felhasználhatósága. Egy programkódra akkor mondjuk, hogy jól olvasható, ha a kód írójától különböző, más programozók is gyorsan és egyértelműen értelmezni tudják. Ez akkor valósulhat meg, ha minden programozó azonos alapelvek és kódolási szabványok követésével dolgozik. Ezen szabályok betartásának az ellenőrzése, valamint az újra felhasználható kódrészek beazonosítása a szövegbányászat témakörébe tartozó komplex feladatok. Jelen kutatás keretében reguláris kifejezések alkalmazásával annyit valósítottam meg, hogy változók, struktúrák és szimbolikus konstansok deklarálása esetén a program ellenőrzi a névkonvenciók betartását. A programkód további, szövegalapú elemzése az alkalmazás továbbfejlesztési céljai között szerepel.

7. Köszönetnyilvánítás

A cikkben ismertetett kutatómunka az EFOP-3.6.1-16-2016-00011 jelű *Fiatalodó és Megújuló Egyetem – Innovatív Tudásváros – a Miskolci Egyetem intelligens szakosodást szolgáló intézményi fejlesztése* projekt részeként – a Széchenyi 2020 keretében – az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósul meg.

Irodalom

[1] *Supporting entrepreneurship and innovation in higher education in Hungary*. Ch.1. Overview of the Hungarian higher education system, OECD/European Union, ISBN 978-92-64-27334-4, **https://doi.org/10.1787/9789264273344-en**

[2] Pasztor, A., Pap-Szigeti, R., Török, E. (2014). *Innovatív informatikai eszközök és módszerek a programozás oktatásban*. Gradus, 1, 22–27.

[3] Cardoso, M., de Castro, A. V., Rocha, Á., Silva, E., Mendonça, J. (2020). Use of automatic code assessment tools in the programming teaching process. In *First International Computer Programming Education Conference (ICPEC 2020),* Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

[4] *CoderByte homepage* – https://www.coderbyte.com/ (Utolsó hozzáférés: 2020. július 15.).

[5] *CodinGame homepage*– https://www.codingame.com/start (Utolsó hozzáférés: 2020. július 15.).

[6] *Codeboard homepage* – https://codeboard.io/ (Utolsó hozzáférés: 2020. július 16.).

[7] *C Puzzles homepage* – https://chortle.ccsu.edu/CPuzzles/ (Utolsó hozzáférés: 2020. július 17.).

[8] *Codeboard dokumentáció* – https://codeboard.io/docs/submission#create (Utolsó hozzáférés: 2020. július 16.).

[9] Molnár, Gy., Nyirő, P. (2016). A gyakorlati programozás tanításának játékfejlesztésen alapuló, élménypedagógiai alapú módszerének bemutatása. Karlovitz, J. T. (szerk.). *Pedagógiai és szakmódszertani tanulmányok*. 89–98.

[10] Juhász, I. (2013). *Programozás*. https://people.inf.elte.hu/kinnaai/programoz%E1s/prog1/programozas120040519.pdf (Utolsó hozzáférés: 2020. október 2.).

[11] Ficsor, L., Kovács, L., Kusper, G., Krizsán, Z.: *Szoftvertesztelés*, https://regi.tankonyvtar.hu/hu/tartalom/tamop425/0046_szoftverteszteles/adatok.html (Utolsó hozzáférés: 2020. október 2.).

[12] Kasahara, R., Sakamoto, K., Washizaki, H., Fukazawa, Y. (2019). *Applying gamification to motivate students to write high quality code in programming assignments*. In Innovation and Technology in Computer Science Education (ITiCSE '19), July 15ś17, 2019, Aberdeen, Scotland, UK. ACM, New York, NY, USA, 7 pages, **https://doi.org/10.1145/3304221.3319792**

[13] Bellairs, R. (2019). *What is code quality, and how to improve code quality?* https://www.perforce.com/blog/sca/what-code-quality-and-how-improve-code-quality (Utolsó hozzáférés: 2020. augusztus 12.).

[14] Gopal, M. K. (2016). Design quality metrics on the package maintainability and reliability of open source software. *International Journal of Intelligent Engineering and Systems 2016*, 9(4):195–204, **https://doi.org/10.22266/ijies2016.1231.21**

[15] ISO/IEC 25010:2011 4.2.7 – https://www.iso.org/obp/ui#iso:std:iso-iec:25010:ed-1:v1:en (Utolsó hozzáférés: 2020. augusztus 13.).

[16] Kun, I:. *Informatikai rendszerek megbízhatóságának matematikai modellezése*. http://www.hadmernok.hu/2009_4_kun.pdf (Utolsó hozzáférés: 2020. október 2.)

[17] Measurement of Halstead Metrics with Testwell CMT++ and CMTJava (Complexity Measures Tool), 2017 – https://www.verifysoft.com/en_halstead_metrics.html (Utolsó hozzáférés: 2020. augusztus 13.).

[18] McCabe, T. J. (1976). A complexity measure. *IEEE Transactions on Software Engineering 1976*, 2(4):308. **https://doi.org/10.1109/TSE.1976.233837**

[19] Mooney, J. D. (1997). *Bringing portability to the software process*. Dept. of Statistics and Comp. Sci., West Virginia Univ., Morgantown WV.

[20] Ulbert, Zs. (2014). *Szoftverfejlesztési folyamatok és szoftver minőségbiztosítás*. https://regi.tankonyvtar.hu/hu/tartalom/tamop412A/2011-0042_szoftverfejlesztesi_folyamatok_magyar (Utolsó hozzáférés: 2020. október 2.).

[21] Ficsor, L., Krizsán, Z., Mileff, P.: *Szoftverfejlesztés*. https://regi.tankonyvtar.hu/hu/tartalom/tamop425/0046_szoftverfejlesztes/index.html (Utolsó hozzáférés: 2020. október 2.).