Mi van, ha nem tudok teát főzni?

Szalayné Tahy Zsuzsanna

sztzs@szigbp.hu
ELTE IK

Absztrakt. Az algoritmus fogalmának értelmezésére számos helyen a teafőzés algoritmusát mutatják be. A példa valószínűleg Angliából származik és alapvetően felnőtt oktatásban van helye. Néhányszor megpróbáltam én is ezzel a példával szemléltetni az algoritmizálást, lényegében eredménytelenül. Az ok: a diákok nem szoktak teát főzni, és ha még tudják is, hogy hogyan kell, ez a tudás családonként eltérő. Ezért más példákat szeretnék mutatni. A teafőzés környezetéhez képest feltételezem, hogy a diákok gyerekek (nem érettek, nem családfők vagy eltartók). Jellemzően közoktatásban tanulnak. Azonban azt gondolom, a példák helyett még hatékonyabb, ha a fogalom bevezetése folyamatosan történik, ezért szeretnék bemutatni néhány informatikaórán tanított tartalmat, amely egyben az algoritmus fogalmának, az algoritmus struktúráinak megértését is segíti. Végül szeretnék felvetni egy gondolatot: Az alkalmazói szoftverek oktatása akkor lesz igazán hatékony, ha mindvégig programozást oktatunk, azaz a szoftver menüjének ismerete helyett a programozó által leképezett adatstruktúrát és algoritmust értik meg a diákok.

1. Miért kell tudni teát főzni?

Az algoritmus fogalmának értelmezésére számos helyen ([1]–[7]) a teafőzés algoritmusát mutatják be. Néhányszor megpróbáltam én is ezzel a példával szemléltetni az algoritmizálást, lényegében eredménytelenül. Az ok: a diákok nem szoktak teát főzni, és ha még tudják is, hogy hogyan kell, ez a tudás családonként eltérő. Ahelyett, hogy az algoritmus fogalmát értenék meg, fennakadnak azon, hogy ki hogyan tud, vagy nem tud teát főzni.

Nézzük meg, miért terjedhetett el a teafőzés, mint első algoritmus!

Bár az első programozó nő volt (Ada Byron lady Lovelace), a 20. századra jellemző a szakma férfi dominanciája [8], [9]. Egyes programozói szakokon a férfiak felülreprezentáltsága már problémás.

A szakirodalom nyelve angol és a fejlesztések angol nyelvterületről indultak. A szakma nyelve az angol, mind a programozási nyelvek mind a szoftverek területén az angol nyelv használata magasan első helyen áll.

Korosztályát tekintve a programozás tanulására – úgy tűnik – az a jellemző, hogy felnőtt korban kezdik el. A szakma kialakulásakor ez szinte természetes volt. Néhány tíz-évben mérhető a szakma kora, jelentős azon szakemberek száma, akik más pályáról tértek át (matematikus, fizikus, mérnök). Azok száma pedig még nagyobb, akiknek szakmájukhoz kiegészítő ismeretet jelent a programozás. A programozást oktató könyveket, cikkeket jellemzően felnőtteknek írják.

Azt is vegyük figyelembe, hogy a gyakorlatban jellemző, hogy az algoritmus, algoritmizálás fogalma a programozás tanításához kapcsolódik, az algoritmus így egy kelléke a programozásnak. A vizsgált példák is ezt a tematikák képviselik. Ettől jelentősen eltér a tartalmában is algoritmusokkal foglalkozó képzés, szakirodalom, azonban ezzel jelen cikkünkben nem kívánunk foglalkozni, mert jellemzően nem ezzel találkozik először a diák.

Vegyünk egy tipikus programozást tanuló embert: Felnőtt (legalább 18 éves), férfi és tud angolul. Tudása a világról nehezen bemérhető, de jellemzően önellátó. Tanfolyamon, csoportos képzésben tanulja a programozást és már várja a szünetet – az oktatójával egyetemben. Egy

ilyen típusú tanulónak milyen példa kell? Azt hiszem, nálunk a kávéfőzés lenne az első ötlet, de angol nyelvterületen a teafogyasztás jellemzőbb. A példával nem lenne gond, ha az „átlagos programozást tanuló ember” tényleg ilyen lenne, de ez szerencsére több szempontból is változik és remélhetőleg változni is fog.

2. Mivel helyettesíthető a teafőzés?

Amennyiben nem a fent említett típus a programozás oktatásának célközönsége, az algoritmusra adott példának is célszerű valami mást választani. Olyan példa kell, amit mindenki megért, de legalábbis a résztvevőktől elvárható korábban tanított ismeretre épül.

2.1 Amit mindenki tud

Első közelítésben ilyen példa nem létezik. Mindenkinek egy kicsit más a tudása a világról, ezért bármilyen jónak tűnő példát mondunk, kellő pontosítás nélkül lesznek olyan diákok, akiknek nem lesz egyértelmű a feladat. A „kellő pontosítás” viszont épp az algoritmus megadását jelenti. De azért van néhány olyan, életből vett példa, amelyek nagy valószínűséggel ugyanazt a fogalomrendszert jelentik mindenkinek.

Példa:

Amikor megszületünk, semmit sem tudunk, de az első pillanattól kezdve tanulunk. A születést követő néhány héten belül kialakul a csecsemő napirendje. Írjuk le a csecsemő napirendjét, majd ennek felhasználásával a csecsemő életének 2. és 3. hónapját.

Lehet, hogy eltérő sorrendben, de nagyjából azonos műveleteket fognak megnevezni a diákok: eszik, alszik, sír. Az ismétlést napi 4-6 alkalomra fogják korlátozni, Az időtartam körülbelül 61 napos lesz.

Természetesen nem mondjuk meg egy csecsemőnek, hogy születése pillanatától algoritmusokat tanul. Talán még az alsó tagozatos diáknak sem kell tudnia ezt a fogalmat, de ebben a korban már megalapozhatjuk az algoritmizálás készségét. Rávezethetjük az egymásutániság, ismétlődés és alternatívák felismerésére; elvárható, hogy ismert tevékenységsorozatot elemeire bontson.

Vajon hány programozást oktató gondolt bele abba, hogy egy újszülöttnek is algoritmust tanítunk? Valószínűleg nagyon kevesen. A csecsemő megtanulja a szabályt, alkalmazza, de nem tudná leírni. Kívülről nézve, felnőve, azonban könnyen „ráérzünk” a lényegre. A csecsemő illetve kisgyermek által végzett tevékenységek, „felismert” adatok és objektumok viszonylag egyértelműen meghatározhatók. Egy kisiskolás már valószínűleg el tudja mondani, mit csinál egy csecsemő: „Egész nap eszik, alszik, sír”. Mivel nem saját magáról beszél, hanem egy fejletlenebb gyerekről, könnyen általánosít, el tud tekinteni a részletektől.

Persze lehet, hogy valaki nem tudja, mit csinál egy csecsemő, de legalább nem tudja másképp, mint a többi. A teafőzéshez képest ez a példa azért jobb, mert nem a programozást tanuló diák saját (esetleg nem létező) szokásaira épül a feladat, nem saját tevékenységét kell leírnia, hanem egy általános „szabályt”.

2.2. Amit az oktatott diákok tudnak

A teljesen általános példa helyett sokkal könnyebb olyan példát választani, amit a résztvevők mindegyike tanult. Számtalan algoritmust tanulnak meg a diákok tanulmányaik során, csak épp tipikusan szabálynak nevezik őket. Ezek közül az algoritmusokkal foglalkozó könyvekben is előforduló gyakori

Példa:

„Fogalmazd meg, hogyan kell összeadni két sokjegyű természetes számot!”

A számok összeadása 1–3. osztályban tananyag. Egy ötödik osztályos diáktól (és ennél idősebbektől) elvárható a megoldás gyakorlati kivitelezése. A műveletsorozat végrehajtása azonban még nem jelenti azt, hogy le is tudja írni a diák az egyes lépéseket. Azonban a teafőzéshez képest ebben az esetben konkrétan meghatározhatók az adatok, az egyes lépések (szekvencia, alternatíva, iteráció – vagy ennek megfelelő fogalmak), de akár még az eljárás illetve függvény fogalmának bevezetésére is használhatjuk.

Ha a történelmi gyökerekhez nyúlunk vissza, akkor is valószínűleg az összeadás algoritmusával kezdjük a programozás oktatását: Al Hvarizmi hindu számokról szóló műve, a latinul fennmaradt Algorithmi de numero Indorum. A szerző latinosított nevéből származik az „algoritmus” szó, a műben a tízes-helyiértékes számábrázolást és számolási módszert írja le. [10]

Az összeadás algoritmusa után önállóan megírható a kivonás algoritmusa, majd a szorzásé, osztásé. Ma sokszor halljuk, hogy a diákok nem tudják a négy alapműveletet. Pedig mindenki tanulta, egy kis ideig gyakorolta is, azonban utána évekig nem kell használnia. Valamikor megtanult egy cselekvéssorozatot, talán meg sem értette, csak mechanikusan csinálta… Azután ez az értelmetlen szabályrendszer elfelejtődik. Az algoritmus fogalmának kialakítása egyben a felejtés homályába vesző ismeretek felelevenítésére is alkalmasak. Ebben az esetben az algoritmus lényegében egy korábban tanult végrehajtási szabály leírása.

Az alapműveletek írásbeli elvégzésénél fontos a helyiértékek megfelelő kezelése, ezért ha a csoportban felmerül a precíz megfogalmazás igénye, akkor el lehet indulni a számjegyekből álló tömb fogalmának bevezetésével az összetett adatszerkezetek, indexek tanításának irányába. Emellett be kell vezetni olyan változót is, amelyet általában nem írunk le – például az összeadásnál az átvitelt. Ez a tárolt adat a szó szoros értelmében lehet változó.

De el lehet indulni másik irányba is. Érdemes megmutatni, más nemzetek diákjai hogyan tanulják az írásban elvégzett maradékos osztást. Ahány ország, annyi elrendezés, szabály, leírandó vagy csak megjegyzendő részlet.

Példa:

Írjuk le, hogy az alábbi osztások milyen algoritmusok eredményei.

a) magyar módszer

6	5	8	7	3	:	5	4	=	1	2	1	9
1	1	8										
	1	0	7									
		5	3	3								
			4	7								

b) japán módszer

			1	2	1	9
5	4	6	5	8	7	3
		5	4			
		1	1	8		
		1	0	8		
			1	0	7	
				5	4	
				5	3	3
				4	8	6
					4	7

Megjegyzendő, hogy az osztás algoritmusának vizsgálata erősen érzékelteti a teafőzés algoritmusának egyik problémáját: ahány ország, annyiféleképpen valósítják meg. Hasonlóan a teafőzéshez, egyes megvalósítások csak apróbb módosítást tartalmaznak a magyar közoktatásban tanított változathoz képest – a tárolt adatok elrendezésének módja más –, de azokban az esetekben, amikor a tárolt adat is más, az algoritmus megértése egy magyar diáknak komoly feladat,

így nem alkalmas arra, hogy az algoritmus fogalmát bemutassa. Másrészt az algoritmus fogalmának bevezetésére egy, az eddigi példáktól eltérő módszerére lehet példa.

Az eddigi példák azon alapultak, hogy „mindenki ismeri", írjuk le a végrehajtás módját egységesített kifejezésekkel. A „másképp kinéző osztás" algoritmusának leírása azzal kezdődik, hogy ki kell találni, mit csinálhatott az, aki így írta le a feladat megoldását. A „hogyan működik" kérdésre a válasz egy szabály leírása, amihez meg kell állapodni a közös szóhasználatban azaz az algoritmus leírásának formájában.

Nem tudom, elérhető-e, hogy az itt említett példákat egy alsós tanító vesse fel, elkezdje az algoritmus fogalmával kapcsolatos készségek kialakítását, programozás oktatásának megalapozását. A matematikatanártól sem várhatjuk el, hogy informatikai szemmel nézze az osztás problémáját. Másrészt, ha az algoritmusokról csak informatikatanárok, programozást oktatók beszélnek, akkor a választott példa lehet korábban tanult informatikával kapcsolatos ismeret is.

3. Alkalmazásba integrált programozás

Amikor azt mondjuk valakiről, hogy informatika alkalmazásból jó, vagy azt, hogy az ICT-ben jártas, sokan arra gondolnak, hogy az illető ismeri egy szoftver menüjét és jól tud kattintgatni. Tipikusan ebből származik az az elképzelés, hogy aki jól tud játszani a gépen, az már a munkához szükséges alkalmazások területén is jó lesz. Ma nagyon sok ország nagyon sok kormánya ebben a tévhitben él. Az informatika tanterv lényegében ICT és irodai alkalmazások ismeretét, fedi le. Máshol felismerték, hogy aki nem tud programozni, az nem tudja hatékonyan használni a számítógépet. Ezért informatika tantárgy keretében színtiszta programozást tanítanak, inkább kevesebb, mint több sikerrel.

Azt gondolom, hogy az alkalmazásokat a programozástól nem kell, sőt, nem szabad ennyire szétválasztani. Aki alkalmazást használ, az minden esetben programozó által kódolt algoritmusokat használ. Az alkalmazások oktatása során számtalan esetben van mód arra, hogy a diákok megértsék az algoritmus fogalmát illetve egyes algoritmusokat, adatstruktúrákat, a program „lelkét". Lehet úgy tanítani a szoftverek használatát, hogy eközben algoritmus-készítést és ezzel együtt programozást is tanítunk. Néhány tanított alkalmazás mentén nézzünk néhány példát, mit lehet programozásból tanítani.

3.1 Pixelgrafikus festés (Paint)

A szoftvert már egy óvodás is „tudja" használni, de igazán csak az használja ki a lehetőségeket, aki ismeri például a képpont fogalmát, az RGB jelentését. Néhány algoritmus ismerete elengedhetetlen egy rutinos felhasználó számára, például szín csere (kitöltés), átméretezés, torzítás (hogyan módosul az egyes képpontok színe); forgatás, tükrözés, döntés esetében. Egy kicsit kitekintő, értelmező feladattal segíthetjük, hogy a diák megértse, mi lehet a színes képernyőkép mögött.

Példa:

Egy képet vízszintesen nyomjunk össze felére! Figyeljük meg, az új kép képpontjainak színét és írjuk le, hogy a művelet végrehajtása során hogyan „gondolkozik" a szoftver!

Az osztás elvégzéséhez hasonlóan, kutatómunkával kezdődik a feladat megoldása. Az általam kipróbált festő programban (Windows7 Pait-je) például egy 16×4 pixeles sakktábla ( ) 8×4 pixelesre összenyomása során ( ) az egymás melletti képpont párokból az elsőt elnyelte a szoftver. A 17×4 pixeles szaktáblán ( ) minden képpont-pár másodikja tűnik el ( ). Az algoritmus tartalmaz 2 ciklust, lehet benne elágazás vagy visszafelé számlálós ciklus. (Azt szoktuk tanítani, hogy ez a művelet veszteséges tömörítés, de azt is lehetne tanítani, hogy hogyan vész el az adat.)

Jelen esetben nem az a lényeg, hogy pontosan ismerjük a szakirodalmat, szoftverleírást, hanem az, hogy a szoftverben tapasztalt szabályszerűségeket, algoritmusokat felfedezzük, meg tudjuk fogalmazni. Ez segít ahhoz, hogy később saját feladatainkhoz algoritmusokat tudjunk megfogalmazni, programot tudjunk írni.

Haladó alkalmazók a program használata előtt megtervezik a munkát. Egy kép elkészítésének sebességét jelentősen befolyásolhatja, hogy mit, milyen sorrendben végzünk el – azaz most arról van szó, hogy a rajz elkészítésének módja, az algoritmusunk elég hatékony-e. Ebben az esetben a szoftver ismeretében saját tevékenységünket tervezzük meg, algoritmizáljuk. Egy ilyen feladat segíti a szoftver hatékony használatát, de csak közvetve segíti a programozás oktatását, mivel nem a szoftver, hanem a felhasználó tevékenységét írja le, csak úgy, mint a következő példa.

3.2 Vektorgrafikus rajzolás

Ez az alkalmazás kiválóan alkalmas az osztály, objektum, objektum tulajdonságai, öröklődés fogalmainak tisztázására. Komolyabb feladatok tervezést igényelnek, az ábrák elkészítésének algoritmusa akár bevezetés is lehet a programozáshoz.

Példa:

Készítsd el az Európai Unió Zászlóját!

Egy lehetséges megoldás vázlata: Az alap egy kék téglalap és egy sárga ötágú csillag (objektum létrehozása). A csillagból kell még további 11 másolat (ciklus), kell egy 12-ágú szabályos csillag (objektum-létrehozás, csúcsok és csillagok tömbje). Ennek a csillagnak minden csúcsához el kell csúsztatni az ötágú csillag megfelelőjét (ciklusok egymásba ágyazva), majd a 12-ágú csillagot törölni kell, az ötágú csillagokat egybefoglalni (új objektum létrehozása – csoport, csillagok adatainak módosítása). A csillagokat a zászló közepére kell vinni (két egymás utáni ciklus).

3.3 Szövegszerkesztés

A legegyszerűbb editorokban is van szövegkeresés és -csere lehetőség. A felhasználási módszerek mellett, az oktatás során felvethető a „hogyan végzi el a feladatot a szoftver” kérdés. A válaszban megfogalmazhatjuk a mintaillesztéses keresést, a ciklust.

A karakterek kódolása, kódtáblák a szövegszerkesztés során megtanítandó programozási ismeret.

Fejlett szövegszerkesztők tanításakor az objektum orientált programozást is megalapozhatjuk. Felismerhetők a szöveg objektumai: a karakter a bekezdés, a szakasz, a dokumentum, a táblázat, a grafikai elem, beágyazott tartalmak… Értelmezhető az osztály, objektum, öröklődés, tulajdonság, esemény… fogalma.

A szöveg tördelésével kapcsolatban számos szabályt fogalmazunk meg. A szabályok betanítása mellett érdemes a diákoknak is belegondolni abba, hogy milyen feltételek érvényesülnek a tördelés során, melyik algoritmus ad hibás megjelenést, ha a mondatvégi pont előtt szóköz van.

A szövegszerkesztők (dokumentumkészítésre használt eszközök) jellemzően megengedik ellentmondó értékek beállítását. Érdemes megfigyelni, milyen algoritmus (sorrendiség?) kezeli az ellentmondásokat. Például táblázatban, hasábok használatánál az egyes oszlopok és közök méretének összege egyenlő kellene legyen a teljes táblázatra, oldaltükörre vonatkozó szélességgel. Mit tapasztalunk, ha az értékek egymásnak ellentmondanak?

A fenti példákon túl, algoritmusok felismerésének széles lehetőségét adja egy szövegszerkesztő. Álljon itt részletezés nélkül még néhány önkényesen kiválasztott további példa:

- középre igazítás, sorkizárás algoritmusa;
- a fattyú- és árvasorok kezelése;
- a lábjegyzet elhelyezése és oldaltörés viszonya.
- objektum szövegbe ágyazási és csatolási lehetőségei algoritmus és kód szinten is megfigyelésre méltóak.

Az alkalmazások oktatásának alapszintjén megtanuljuk, hogy a fenti eszközöket hogyan használjuk. A fejlett használathoz, a későbbi programozás oktatáshoz azt is érdemes megtanítani, hogy milyen programozói döntések, milyen algoritmusok eredményezik az adott eszköz működését.

A formázott dokumentum forráskódjának megismerése, a formázás kódolása szintén segíti a programozás tanulását, ez az egyik oka annak, hogy weblapkészítést is tanítunk.

3.4 Táblázatkezelők

A táblázatkezelés akár a programozás előszobájának is tekinthető. Kiválóan alkalmas a szöveg, szám és logikai típus megkülönböztetésére, a szoftverkörnyezet dátumkezelésének megismerésére. Megismerhető az egyes adattípusok tárolási mérete, korlátai, műveletei.

Példa:

	A
1	alma
2	1,2
3	02.jan
4	IGAZ
5	2048.03.15
6	1848.03.15
7	1,23457E+19
8	12345678901112131415

1. ábra: Adatok és adattípusok táblázatkezelőben

A táblázatkezelők számos eljárással, függvénnyel segítik a munkát. Ezek algoritmusa felfedezhető, megfogalmazható:

- Az adatok importálása, illetve a Szövegből oszlopok… lehetősége egy szöveg-feldolgozási algoritmust rejt.
- Egy tömb kijelölése után beírt képlet Ctrl+Enter leütésével a kijelölés minden cellájába beviszi ugyanazt az értéket (foreach ciklus)
- Egy tömb melletti oszlop egy cellájában levő képlet jobb alsó sarkára kattintva a képlet addig másolódik lefelé, amíg a mellette levő oszlopban (vagy az adott oszlopban) adat van. (while)

A táblázatkezelők egyik legfontosabb jellemzője a felhasználó számára elérhető függvények adta lehetőségek. Minden egyes függvény egy algoritmus eredménye. Minden számítás függvény is egyben.

Példák:

Írjuk meg a SZUM(A2:A10) függvény algoritmusát!

Írjuk meg az ÁTLAG(A2:A10) függvény algoritmusát!

Írjuk meg a MIN(A2:B10) függvény algoritmusát!

Kihagyhatatlan a HA() függvény argumentumainak és algoritmusának a tanítása.

Példa:

Adjuk meg a HA(A7>1;"jó;"rossz") függvény algoritmusát;

A kereső függvények (HOL.VAN(); FKERES(); VKERES(); KERES()) használata ma nagyon sok diáknak okoz gondot. Egy-egy ilyen függvényhez – ha nem tudja elképzelni a használója, hogyan „gondolkodik" a függvény (azaz a szoftver, azaz a programozó), nagyon sok értelmetlennek tűnő szabályt kell megjegyezni. Adatok rendezettsége, tartományban keresés visszaadott értéke, az eredmény értelmezése túl sok megjegyzendő szabályt jelent annak, aki csak tanulja, de nem érti a függvényekben leprogramozott algoritmust. E függvények tanításával együtt érdemes megtanítani a lineáris és logaritmikus keresés algoritmusát.

A rendezés általában néhány kattintás, de ha nem látjuk az adatstruktúra fontosságát, a rendezés során megvalósított algoritmust, akkor lehetnek vele problémáink:

- Miért nem ismeri fel a címsort a táblázatkezelő?
- Miért módosul az adatok melletti képletek helye?
- Miért probléma, ha az adatok között különböző sorokra hivatkozó képlet van? Ha abszolút hivatkozással (vagy vegyes- vagy relatívhivatkozással) adjuk meg a képletet, akkor jó lesz?

Összefoglalva: „Miért olyan buta a táblázatkezelőnk, hogy nem érti, mit szeretne eredményül a felhasználó." A probléma megoldása pedig abban rejlik, hogy le tudja-e írni a felhasználó a szoftvernek kiadott feladat – rendezés – egy lehetséges algoritmusát.

4. Programozás oktatása kezdőknek

Mi van, ha nem tudok teát főzni? – Amennyiben a programozás tanulás első példája egy olyan feladat, amit nem tudok megoldani, akkor hogyan fogom megoldani a többi problémát?

Jó lenne, ha az első példa az adott témában olyan lenne, hogy a tanuló meg tudja oldani, így biztosítva a kapcsolatot korábbi ismeretei és a megtanulandó új tudás között. Azok, akik „nem tudnak teát főzni", az első probléma nem az algoritmus fogalmának megértése, hanem a teának az egész problémakörhöz – programozáshoz – való kapcsolata.

A példákat összehasonlítva érdemes elgondolkodni azon, hogy a programozás oktatásához az algoritmus fogalmát emberi tevékenységek, felhasználói műveletek vagy egy kész termék, szoftver segítségével vezessük-e be.

Az előző két fejezetben néhány lehetséges kezdő feladat olvasható. Ezek mindegyikéről elvárható, hogy egy „kezdő programozó" ismerje a feladatot. Mégis, azt kell mondjuk, hogy egyikről sem mondhatjuk biztosan, hogy tudja.

Egy kezdőknek szóló programozási jegyzet elkészítésénél mindenképp meghatároznám, hogy kinek szól a jegyzet, milyen előismeretet kíván a tanulótól. Ez az előismeret biztosan nem lehet nulla. Amennyiben valamilyen iskolai végzettséget el lehet várni (például felsőoktatási intézmény, vagy alapszintű képzést igazoló végzettség), ennek a végzettségnek megfelelő elvárt ismerettel lehet számolni, az első példa származhat a végzettséghez előírt ismeretből. Ilyenek például a matematikai példák. Amennyiben a tanítandó csoport az alapvető algebrai műveleteket ismeri (ezt könnyű ellenőrizni), akkor az algoritmus fogalmát a szó eredetéből is magyarázhatjuk. „Már az ókorban is", majd a középkorban az arab munkák megismerésével Európában is az összeadást így végezzük…. (Nem is értem, miért kell ehhez a tea.)

Egy kicsit más a helyzet akkor, ha a csoport más informatikai ismeretek mellett vagy után tanul programozni. Ebben az esetben a programozás oktatását célszerű lenne az alkalmazói ismeretek mellett folyamatosan elkezdeni. Az egyes informatikai témákon belül „lemenni" a programozás szintjére, megvizsgálni a szoftverben fellelhető adatokat és algoritmusokat.

Az igazán hatékony informatikaoktatást az jelentené, ha egy olyan felhasználói szemléletű képzési környezetben, mint az ECDL is, megjelenne a „miért így működik" szemlélet, azaz nem kaphatna ECDL bizonyítványt az, aki nincs tisztában az adattípusokkal és legalább néhány elemi algoritmussal. Ha összehasonlítjuk ezt a fajta tudást, mondjuk a gépjárművezetéssel, ott sem elég a kormány, fék, pedál és esetleg a kuplung használatának ismerete; elvárt a műszaki és egészségügyi vizsga is. Ma egyre több helyen, egyre nagyobb problémát okoz, hogy vizsgázott „képzett" emberek képtelenek megoldani egyes szoftveres problémákat pusztán azért, mert nem ismerik a szoftver működését. Ez azt mutatja, hogy a programozás alapjait, az algoritmizálást be kell vinni a szoftverhasználat oktatásába.

Mit jelent programozást oktatni kezdőknek? Remélem, a következő generációnak valami egész mást, mint napjainkban. Mert a kezdő programozó évek óta használja a digitális eszközöket, érti, hogyan működik a gépe. Mire eljut odáig, hogy programozást tanul – azaz egy olyan kurzuson vesz részt, amin programot készít –, már tudja mi a különbség a szöveg és a szám között, tudja, hogy egy-egy utasítással a gép egy programkódot fog lefuttatni, jellemzően egy algoritmust fog végrehajtani. Az első órán sem kell visszanyúlni az algoritmus fogalma miatt a mindennapi példákhoz – teafőzéshez –, mert lesz olyan szoftver, ami jól láthatóan hajt végre egy-egy egyszerű algoritmust. Így egy kezdő programozó kurzus első példája mondjuk az lesz:

Példa:

Kijelölted a fájlok listájában az egyik fájlt, majd a Shift lenyomása mellett rákattintasz egy másikra. Mit csinál a fájlkezelő szoftver az alatt a „pillanat alatt", ami ezt a kattintást követi és a két fájl közötti összes fájlt kijelöltté teszi?

5. Összefoglalás

Ma még sok könyvben, tanfolyamon úgy kezdődik a programozás oktatása, hogy megtanítjuk, mi az algoritmus. Pedig a fogalom napjainkban nem lehet új, a példáért sem kell messzire menni, csak másképp kell látni a korábban megismert szoftvereket, alkalmazásokat. Ma már túl vagyunk a digitális kor hajnalán, elvárható, hogy az, aki programozást kezd tanulni, korábban lásson, használjon programokat. De ezt a „másképp látást" már jóval korábban el lehetne kezdeni: az első pillanattól kezdve nem az alkalmazást kell megtanítani, hanem a programozáshoz szükséges alapokat – az algoritmusokat – néhány alkalmazáson keresztül. A tanulás során először kész programok analízise teszi szükségessé az algoritmusok ismeretét, ezt követi a szintézis, amikor szeretnénk megvalósítani valamit és egy eljáráshoz saját algoritmust írunk. Először szedjük darabjaira, elemezzük a kész programokat, és ha már értjük, hogyan működik, akkor kezdjünk el saját programot írni. Felesleges máshonnan hozott példával magyarázni az algoritmus fogalmát. A számítógép működésében is fel lehet ismerni az algoritmusokat, az eredmény alapján meg lehet érteni, hogy hogyan működik. Eddig a pontig nem programozásról beszélünk, ez ismerkedés a szoftverekkel, az algoritmusokkal. Ez közismeret, a digitális társadalom minden tagjának tudnia kell(ene). Egy kezdő programozás kurzuson már az első órán saját algoritmust kellene tervezni a már ismert minták alapján és azzal kellene foglalkozni, hogy hogyan lesz az algoritmusból kód, program.

Irodalom

1. *Methods of Algorithm Description*, (2nd edition) Board of Studies NSW (1995. Mach), p: 14 http://math.uni.lodz.pl/~fulmanp/zajecia/ics/materialy/compalgorithm.pdf (megtekintés: 2013.10.29)
2. Jaana Holvikivi, *Introduction to Computing TIEL0007, CAP05S & DAP05S*, Learning Systems, (2005. ősz), http://users.evtek.fi/~jaanah/IntroC/DBeech/3gl_algorithm1.htm (megtekintés: 2013.10.29)
3. Liam Dunphy, *Algorithms in SSD*, NSW BOS, Terrey Hills,NSW, Australia (2009.07.09) 16. dia http://www.slideshare.net/ldunphy/algorithms-1700665 részeként http://image.slidesharecdn.com/algorithms-090709091828-phpapp02/95/slide-15-728.jpg?1247149155 (megtekintés: 2013.10.29)
4. Tömösközi Péter, *Algoritmizálás alapjai*, EKFT, (2003.03.06) p: 1–2. http://www.ektf.hu/user/tpeter/bevezprogr/FreePascal.doc (megtekintés: 2013.10.29)
5. Pohl László, *A programozás alapjai*, Budapest, (2010.) p: 10-11. http://www.eet.bme.hu/~pohl/h_jegyzet.pdf (megtekintés: 2013.10.29)
6. *Algoritmus*, Programozás Wiki (2013. szeptember 3.) http://wiki.prog.hu/wiki/Algoritmus (megtekintés: 2013.10.29)
7. Ismeretlen bejegyzése (2013.04.22), http://lufi13.blogspot.com/2013/04/a-teafozes-algoritmusa-21-pontban-avagy.html (megtekintés: 2013.10.29)
8. Spiceworks Community, *What is the ratio of women:men in the IT industry?*, Blog (2009.04.03–2011.10.05.) http://community.spiceworks.com/topic/35511-what-is-the-ratio-of-women-men-in-the-it-industry
9. Molnár Marianna, *Ideje félretenni az előítéleteket: lányok férfias szakmákban*, HR Portál 2013.05.02 http://www.hrportal.hu/hr/ideje-felretenni-az-eloiteleteket-lanyok-ferfias-szakmakban-20130502.html
10. Sain Márton, *Matematika-történeti ABC* (Harmadik kiadás), Tankönyvkiadó, Budapest, (1978), p: 17