Tanári szakdolgozat

Tanulmány

Algoritmusok lépésenként

*Készítette:* Bende Imre

tanári mesterszakos hallgató,

informatika–matematika szakterület

*Témavezető:* Dr. Zsakó László (tanszékvezető egyetemi docens)

ELTE Informatika Kar

Eötvös Loránd Tudományegyetem

Budapest, 2016.

Tartalomjegyzék

[Tartalomjegyzék 2](#_Toc466795846)

[Cél 4](#_Toc466795847)

[A program 5](#_Toc466795848)

[Követelmények 5](#_Toc466795849)

[Felhasznált eszközök, keretrendszerek 5](#_Toc466795850)

[A program szerkezete 7](#_Toc466795851)

[JavaScript fájlok 7](#_Toc466795852)

[CSS fájl(ok) 14](#_Toc466795853)

[Fejlesztési lehetőségek 15](#_Toc466795854)

[Tanulási terv 16](#_Toc466795855)

[Egy adatszerkezet / algoritmus bemutatásának szerkezete 17](#_Toc466795856)

[Algoritmus megismerése az oldal segítségével 18](#_Toc466795857)

[Néhány érdekesebb példa algoritmusokra 20](#_Toc466795858)

[Megszámolás 20](#_Toc466795859)

[Maximumkiválasztás 20](#_Toc466795860)

[Logaritmikus keresés 21](#_Toc466795861)

[Buborékos rendezés, Minimumkiválasztásos rendezés 21](#_Toc466795862)

[Összefésüléses rendezés 22](#_Toc466795863)

[Adatszerkezet megismerése az oldal segítségével 23](#_Toc466795864)

[Példák adatszerkezetekre 24](#_Toc466795865)

[Verem 24](#_Toc466795866)

[Sor 24](#_Toc466795867)

[Új anyag készítése 25](#_Toc466795868)

[Példa – n faktoriális 27](#_Toc466795869)

[Összegzés 31](#_Toc466795870)

[Felhasznált irodalom, jegyzék 32](#_Toc466795871)

# Cél

Az volt a célom ennek az alkalmazásnak az elkészítésével, hogy az egyes programozási tételeket, rendezéseket, adatszerkezeteket vizuális megjelenítéssel, illetve az algoritmusok lépésenkénti megtekintésével könnyebben meg lehessen érteni, tanulni, át lehessen látni azokat.

Emellett úgy fejlesztettem le az egyes funkciókat, oldalakat, hogy a hozzájuk tartozó függvényeket, eljárásokat könnyen újra lehessen használni, ezáltal könnyen létrehozhatunk a weboldalra újabb algoritmusokat, amiket a későbbiekben be tudunk mutatni, meg tudjuk tanítani a program felhasználásával.

Az ELTE Informatika Karán szereplő Programozási alapismeretek című kurzushoz is segédanyagként felhasználható lehet, hiszen részben fedi a tárgyhoz szükséges alap tudásanyagot. A kurzusról annyit kell tudni, hogy a programozás alapjainak megtanítása a célja egy általános programozási nyelven keresztül (pszeudokód / struktogram), majd az algoritmusok kipróbálása, implementálása C++-ban történik. Így a kurzus végén a hallgatók elsajátítják a programozáshoz szükséges algoritmikus gondolkodást, illetve egy kis tárgyi tudást is szereznek a mai programozási nyelvek struktúrájából. Ezáltal a további tanulmányaikhoz megfelelő és elégséges tudást szereznek.

# A program

## Követelmények

Nincs túl nagy követelménye a programnak, elég hozzá egy böngésző (Internet Explorer, Mozilla FireFox, Google Chrome) és akadály nélkül használható. Teljesen böngésző, platform, operációs rendszer függetlenül csináltam meg.

A jQuery követelményeit, támogatottságát alapul véve (<https://jquery.com/browser-support/)> a következő böngészőkön biztosan működik: Internet Explorer 9+, Google Chrome 50.x-51.x, Mozilla FireFox 46.x-47.x .

## Felhasznált eszközök, keretrendszerek

Úgy csináltam meg az alkalmazást, hogy ne kelljen hozzá külön szervert létesíteni, így akár lokálisan, ha egy diák/tanár letölti a saját gépére akkor is ki tudja majd próbálni. Az oldal létrehozásához mindössze HTML (HyperText Markup Language), CSS (Cascading Style Sheets), illetve az oldalon történő eseményekhez JavaScript programozási nyelvet használtam.

A weblap készítése során több külsös, ingyenesen felhasználható keretrendszert is felhasználtam. Ezekről részletesen nem írok, de pár gondolatot mindenképpen érdemes róluk mondanom, hogy miért is tettem bele és mire is használtam ezeket.

**jQuery** (2.1.4) JavaScript keretrendszerrel a DOM-ban (Document Object Model) lévő elemeket könnyebben, gyorsabban el lehet érni, illetve azok tulajdonságait egyszerűen át lehet állítani. Így rövidebben, egyszerűbben olvasható kóddal tudtam létrehozni a háttérben futó eseményekhez tartozó kódot. Rengeteg hasznos függvény van benne, ami a felhasználói élményt növeli, de mindezt kevés és érthető kódolással tudjuk megtenni. (Honlap: <https://jquery.com/>)

Az **i18next** JavaScript keretrendszer, ami a nyelvi elemek kezelését segíti, ezáltal könnyen lehet az oldalt nyelvi fájlokkal lefordítani. Az alapoldal megszerkesztése kissé bonyolultabbá válik, azonban ezután a különböző nyelvekre való fordítás jóval egyszerűbb, könnyebb lesz. Később a fordításhoz nem is kell programozói ismeret elég egy JSON (JavaScript Object Notation) fájlban lévő szöveges elemeket lefordítani és már az oldal elérhető válik azok számára is, akik azt a nyelvet ismerik, beszélik. (Honlap: <http://i18next.com/>)

**Bootstrap** rengeteg CSS tulajdonságot, class-okat tartalmaz, ami az egységes, mai szemmel szépnek mondható kinézetet adja. Illetve glyphiconokat tartalmaz még, amik egy-egy oldalon lévő elemeknek még több jelentést tud adni, esetünkben például jobban szimbolizálja egy-egy gomb jelentését. (Honlap: <http://getbootstrap.com/>)

Emellett létrehoztam a programnak egy **git** repo-t (<https://github.com/imrebende/algotan>) is a github-on. Ezt azért is tartom jó ötletnek, mert így ténylegesen teljesen nyílt forráskódú az alkalmazás, bárki elérheti. De nem csak megnézhető, hanem akár bug-okat is lehet bejelenteni rajta, így mindig naprakész lesz, ezek javítása után. Plusz a később bemutatott egyedi algoritmus fejlesztésével akár a diákok saját kódjukat is feltölthetik, így egy egyre növekvő kódgyűjtemény lesz az eredménye és még a diákot is büszkeséggel töltheti el, hogy mások megnézhetik az ő kreálmányaikat, az ő általuk létrehozott oldalt.

Használtam az **UglifyJS 2** nevű JavaScript minifier-t, amivel csökkenteni tudtam a JS (Javascript) fájlok méretét, ezzel rövidítve az oldalt betöltődési idején. Illetve így az alkalmazás kisebb helyen is elfér. Ha feltelepítettük az UglifyJS 2-t, akkor az *uglify.sh* nevű shell script lefuttatásával létre tudjuk hozni a tömörített, „minify”-olt fájljainkat. (Honlap: <https://github.com/mishoo/UglifyJS2>)

## A program szerkezete

A gyökérben helyezkednek el a HTML fájlok, minden egyes algoritmusnak, adatszerkezetnek külön fájljai, oldalai vannak. Itt van még a „*.htaccess”* fájl is, amivel cache-elési tulajdonságokat, illetve gzip tömörítést állítottam be a szerver kisebb leterheltsége érdekében. Végül, de nem utolsó sorban pedig van egy „README.md” fájl a gyökérmappában, ami a git repohoz tartozó leírást tartalmazza.

*cpp* könyvtár tartalmaz C++ minta programokat, bizonyos algoritmushoz. Melyek megfelelő fordítóval egyből lefordíthatók, majd futtatás során kipróbálhatók.

*css* mappában szerepelnek a CSS fájlok (ebből mindössze kettő van a Bootstrap saját CSS-e, illetve az én általam létrehozott *style.css*).

*locales*-ban vannak a lokalizációs fájlok, nyelvi fájlok (JSON kiterjesztésben). Itt akár adhatunk hozzá értelemszerűen új nyelvet, vagy esetlegesen nyelvi javításokat, bővítéseket tehetünk.

*img* mappába lehet képeket tenni. Jelenleg eddig erre nem volt szükség, egyetlen képet tettem bele, amely az ELTE címerét tartalmazza.

*doc* könyvtárban egyetlen fájl szerepel, méghozzá az oldalhoz tartozó dokumentáció.

*fonts* mappában a glyphiconokat tartalmazó fájlok szerepelnek, így ezeknek a fájlok felhasználásával tudjuk megjeleníteni például az oldalon szereplő start, stop, forward, backward gombokat kis ikonokkal.

*js* mappában vannak a JavaScript fájlok. Mivel itt rengeteg fájl, nagyon sok függvénnyel szerepel, ezért erről mindenképpen szeretnék részletesebben írni, hogy ezáltal könnyebben érthető legyen, amit csináltam, illetve azért is, hogy még könnyebben lehessen újra felhasználni az általam már létrehozott kódot.

### JavaScript fájlok

A következőkben az általam írt lényegesebb függvényekről említek meg részleteket, hogy későbbiekben akár könnyebben lehessen őket újra felhasználni, refaktorálni.

eljarasok.js

Nagyon sok függvény van implementálva ebben a fájlban, amiket könnyen lehet használni a későbbiekben, és érdemes is. Párnál, ahol érdemesnek láttam nem csak egy rövid leírást írtam róla, hanem a kódot is bemásoltam.

tombRendezettE(t)

Ellenőrzi a tömb rendezettségét.

//Ellenőrzi, hogy egy tömb rendezett e

**function** tombRendezettE**(**t**){**

**var** l **=** **true,** i **=** 0**;**

**while(**l **&&** i **<** t**.**length **-** 1**){**

l **=** **(**t**[**i**]** **<=** t**[**i **+** 1**]);**

i**++;**

**}**

**return** l**;**

**}**

tombEllenorzes(t)

Ellenőrzi, hogy a bemenet ténylegesen tömb-e.

//Ellenörzi a tömb tartalmát, hogy csak számok, vagy szövegek lehessenek, illetve, hogy jól legyenek a vesszők

**function** tombEllenorzes**(**t**){**

**var** i **=** 0**;**

**while(**i **<** t**.**length**){**

**if(!((**t**[**i**]** **>=** 0 **&&** t**[**i**]** **<=** 9**)** **||** **(**t**[**i**]** **>=** 'a' **&&** t**[**i**]** **<=** 'z'**)** **||** t**[**i**]** **===** ',' **||** t**[**i**]** **===** '-' **||** t**[**i**]** **===** '.'**)){**

**return** **false;**

**}**

i**++;**

**}**

**if(**t**[**t**.**length **-** 1**]** **===** ',' **||** t**[**0**]** **===** ','**){**

**return** **false;**

**}**

**return** **true;**

**}**

tombKiirasa(t, a, b, szoveg)

Kiír egy tömböt a *tomb* id-jú elembe. Ha a, b egy-egy index, akkor azokat kékkel jelöli. Ha a egy tömb, akkor az a tömbben lévő indexeket jelöli kékkel. Ha b egy tömb, akkor a b tömbben lévő indexeket zölddel jelöli.

//Tömb kiírása a "tomb" id-vel rendelkező tartományba

**function** tombKiirasa**(**t**,** a**,** b**,** szoveg**){**

**var** tombClass **=** ""**;**

**if(**szoveg**){**

tombClass **=** szoveg**;**

**}**

**var** tombSzoveg **=** ""**;**

tombSzoveg **+=** '<div class="tomb hidden ' **+** tombClass **+** '">[ '**;**

**for(var** i **=** 0**;** i **<** t**.**length**;** i**++){**

**var** className **=** ""**;**

**if(**i **===** **parseFloat(**a**)** **||** i **===** **parseFloat(**b**)** **||** **(**$**.**isArray**(**a**)** **&&** a**.**indexOf**(**i**)** **>=** 0**)){**

className **=** "blueTombElem"**;**

**}**

**if(**$**.**isArray**(**b**)** **&&** b**.**indexOf**(**i**)** **>=** 0**){**

className **=** "greenTombElem"**;**

**}**

tombSzoveg **+=** '<span class="elem ' **+** className **+** ' ">' **+** t**[**i**]** **+** '</span>'**;**

**if(**i **!==** t**.**length **-** 1**){**

tombSzoveg **+=** ", "**;**

**}**

**}**

tombSzoveg **+=** ' ]</div>'**;**

$**(**"#tombok"**).**append**(**tombSzoveg**);**

**}**

valtozokKiirasa()

Kiírja a *valtozok* id-jú elembe az argumentekben lévő változókat. Egy-egy argumentnek lehet „nev”-e, „ertek”-e és „class”-a amivel a egy CSS class-t tudunk rátenni. Külön kezeli a változókat és a tömböket. Tömb esetén minden elemét megjeleníti szögletes zárójelek között, vesszőkkel elválasztva. Mindkét esetben a változó név után egy kettőspont szerepel, majd pedig a változó értéke.

//Változók kiírása a "valtozok" id-vel rendelkező tartományba

**function** valtozokKiirasa**()** **{**

**var** s **=** '<div class="valtozo hidden ' **+** arguments**[**arguments**.**length **-** 1**]** **+** '">'**;**

**for** **(var** i **=** 0**;** i **<** arguments**.**length **-** 1**;** i**++)** **{**

**if(**$**.**isArray**(**arguments**[**i**].**ertek**)){**

s **+=** '<div class="nev">' **+** arguments**[**i**].**nev **+** '</div>: <div class="ertek ' **+** arguments**[**i**].**nev **+** " " **+** arguments**[**i**].class** **+** '">[ ' **+** arguments**[**i**].**ertek **+** " ]</div><br/>"**;**

**}** **else** **{**

s **+=** '<div class="nev">' **+** arguments**[**i**].**nev **+** '</div>: <div class="ertek ' **+** arguments**[**i**].**nev **+** " " **+** arguments**[**i**].class** **+** '">' **+** arguments**[**i**].**ertek **+** "</div><br/>"**;**

**}**

**}**

s **+=** '</div>'**;**

$**(**"#valtozok"**).**append**(**s**);**

**}**

elsoLepesMegjelenitese()

Megjeleníti az algoritmus első lépésének megfelelő változók, tömbök állásait.

**function** elsoLepesMegjelenitese**(){**

$**(**"#tombok div"**).**first**().**removeClass**(**"hidden"**).**addClass**(**"active"**);**

$**(**"#valtozok div"**).**first**().**removeClass**(**"hidden"**).**addClass**(**"active"**);**

**}**

barValtoztatasa()

Attól függően, hogy melyik állapotot mutatjuk, aszerint jeleníti meg a bar-t, ami azt jelzi, hogy hol tartunk éppen a folyamatban, algoritmusban.

**function** barValtoztatasa**(){**

**if(**$**(**".valtozo"**).**length **!==** 0**){**

**var** allas **=** **(**$**(**".valtozo"**).**index**(**$**(**".valtozo.active"**))** **+** 1**)** **/** $**(**".valtozo"**).**length**;**

$**(**"#algo-progressbar"**).**animate**({**

width**:** **(**allas **\*** 100**)** **+** "%"

**},** 0**,** **function()** **{**

**});**

**}** **else** **if(**$**(**".tomb"**).**length **!==** 0**){**

**var** allas **=** **(**$**(**".tomb"**).**index**(**$**(**".tomb.active"**))** **+** 1**)** **/** $**(**".tomb"**).**length**;**

$**(**"#algo-progressbar"**).**animate**({**

width**:** **(**allas **\*** 100**)** **+** "%"

**},** 0**,** **function()** **{**

**});**

**}**

**}**

lejatszas(k)

Magától elindítja az algoritmus lejátszását 3 másodpercenkénti léptetéssel.

//Lejátsza a folyamatot 3mp/lépés sebességgel

**var** timeouts **=** **[];**

**function** lejatszas**(**k**)** **{**

tombValtoztatas**();**

**for(var** i **=** 0**;** i **<** k**;** i**++)** **{**

timeouts**.**push**(setTimeout(**kovetkezoAllas**,** **(**i **+** 1**)** **\*** 2200**));**

**}**

timeouts**.**push**(setTimeout(**pause**,** k **\*** 2200 **+** 1**));**

**}**

elozoAllas() / kovetkezoAllas()

Lépteti a folyamatot egyik, illetve másik irányba (előre, hátra).

//A változók következő állásának betöltése

**function** kovetkezoAllas**()** **{**

//Következő változó állás betöltése, ha van

**if(**$**(**"#valtozok .active"**).**next**().**length **>** 0**){**

$**(**"#valtozok .active"**).**next**().**addClass**(**"active"**);**

$**(**"#valtozok .active"**).**first**().**addClass**(**"hidden"**).**removeClass**(**"active"**);**

$**(**"#valtozok .active"**).**removeClass**(**"hidden"**);**

allasValtoztatasFunction**();**

**}**

//Következő tömb állás betöltése, ha van

**if(**$**(**"#tombok .active"**).**next**().**length **>** 0**){**

$**(**"#tombok .active"**).**next**().**addClass**(**"active"**);**

$**(**"#tombok .active"**).**first**().**addClass**(**"hidden"**).**removeClass**(**"active"**);**

$**(**"#tombok .active"**).**removeClass**(**"hidden"**);**

allasValtoztatasFunction**();**

**}**

**}**

//A változók előző állásának betöltése

**function** elozoAllas**()** **{**

**if(**$**(**"#valtozok .active"**).**prev**().**length **>** 0**){**

$**(**"#valtozok .active"**).**prev**().**addClass**(**"active"**);**

$**(**"#valtozok .active"**).**last**().**addClass**(**"hidden"**).**removeClass**(**"active"**);**

$**(**"#valtozok .active"**).**removeClass**(**"hidden"**);**

allasValtoztatasFunction**();**

**}**

**if(**$**(**"#tombok .active"**).**prev**().**length **>** 0**){**

$**(**"#tombok .active"**).**prev**().**addClass**(**"active"**);**

$**(**"#tombok .active"**).**last**().**addClass**(**"hidden"**).**removeClass**(**"active"**);**

$**(**"#tombok .active"**).**removeClass**(**"hidden"**);**

allasValtoztatasFunction**();**

**}**

**}**

esemenyek.js

Itt állítom be, hogy az alapnyelv mindenképpen magyar legyen. A későbbiekben, ha több nyelven is elérhető lesz az oldal, akkor itt módosítani lehet rajta, azzal, hogy a „hu” helyére a language változót tesszük, így a böngésző nyelve lesz azalapértelmezett nyelv, tehát ehhez a nyelvhez tartozó JSON fájlt fogja betölteni, ezen a nyelven lesz olvasható az oldal. Próbaként az angol nyelvű JSON fájlt is nagyrészt megcsináltam, így ha ki szeretnénk próbálni, vagy érdekel minket, hogy hogyan nézne ki az oldal angol nyelven, akkor a „hu” helyére „en”-et írva megtehetjük ezt (így a *locales/en/translation.json* nyelvi fájlt fogja betölteni az oldalra).

//Nyelvi elemek beállítása

language\_complete **=** **navigator.**language**.**split**(**"-"**);**

language **=** **(**language\_complete**[**0**]);**

i18n**.**init**({** lng**:** "hu" **},** **function()** **{**

$**(**"html"**).**i18n**();**

**});**

elozo(), kovetkezo(), play(), stop()

Az algoritmus lejátszását állítja. Az előző, illetve a következő lépésre ugrik a függvény nevétől értelemszerűen. Illetve a *play()* függvénnyel elindul, míg a *stop()* függvénnyel megáll az algoritmus bemutatása az adott lépésnél.

Ez a JavaScript fájl még tartalmazza azokat a függvényeket, amelyek a különböző algoritmusok elindításáért felelősek (többnyire algoritmus neve + „Start” név formájában).

programozasitetelek.js, osszetettprogtetelek.js, rendezesek.js, sor.js, verem.js, faktorialis.js

Ezek a JavaScript fájlok tartalmazzák az algoritmusokat leíró függvényeket. Felsorolva, hogy melyeket tartalmazzák jelenleg:

* Programozási tételek: Összegzés, Megszámolás, Maximum-kiválasztás, Keresés, Logaritmikus keresés, Másolás, Kiválogatás, Szétválogatás, Unió, Metszet
* Rendezések: Buborékos rendezés, Minimum-kiválasztásos rendezés, Gyorsrendezés, Összefésüléses rendezés
* Adatszerkezetek és műveleteik: Sor, Verem
* Egyéb algoritmusok: Faktoriális

Nézzünk meg részletesebben is egy algoritmust:

//Maximum-kivásztás tétel

**function** maximumKivalasztas**(**t**)** **{**

tombKiirasa**(**t**);**

**var** maxh **=** 0**;**

valtozokKiirasa**({**nev**:** „MAX”**,** ertek**:** maxh **+** 1**},** „elsoMax”**);**

**for** **(**i **=** 1**;** i **<** t**.**length**;** i**++)** **{**

valtozokKiirasa**({**nev**:** „MAX”**,** ertek**:** maxh **+** 1**},** **{**nev**:** „i”**,** ertek**:** i **+** 1**},** „ciklusLepes”**);**

**if(**t**[**i**]** **>** t**[**maxh**])** **{**

maxh **=** i**;**

valtozokKiirasa**({**nev**:** „MAX”**,** ertek**:** maxh **+** 1**},** **{**nev**:** „i”**,** ertek**:** i **+** 1**},** „maxValtozas”**);**

**}**

**}**

megjelenitesInit**();**

**}**

Alapjaiban véve vesszük a maximum-kiválasztás tétel algoritmusát, majd minden egyes utasításnál, amikor egy változó értékét módosítjuk meghívjuk a „*valtozokKiirasa*” nevű függvényt. Ez egy div-be kiírja a változók neveit, értékeit, illetve az utolsó attribútumban lévő kifejezést megadja egy class-ként (ezzel fogjuk majd az algoritmusokban lévő nyilakat irányítani). A függvény végén pedig a „*megjelenitesInit*” nevű függvényt hívjuk meg, ami az algoritmus lépéseinek bemutatásához szükséges első teendőket teszi meg.

jquery-2.1.4.min.js

jQuery keretrendszer függvényeit tartalmazza, aminek segítségével tudjuk használni a jQuery ( $() ) függvényt. Így érve el könnyebben, gyorsabban a DOM elemeit.

i18next-1.10.3.min.js

Ez végzi el azt, hogy a beállított nyelv alapján megjelenjenek az oldalon lévő címkék, szövegek a locales mappában lévő JSON fájlból.

bootstrap.min.js

Bootstrap JS fájlja. Jelenleg az oldal nem használ fel ebből semmit. De a későbbi fejlesztések során lehet rá szükség.

### CSS fájl(ok)

style.css

Ez a fájl tartalmazza az összes általam létrehozott design-beli változtatást, szépítést. Többnyire a struktogram, pszeudokódhoz tartozó stílus elemeket tartalmaz. De a sor, verem megjelenítését is ezzel oldottam meg, módosítottam olyanná, ahogy az oldalon most láthatjuk.

bootstrap.min.css

A Bootstrap CSS fájlja a már említett rengeteg class-szal, illetve formázási beállítással.

## Fejlesztési lehetőségek

A honlapot mivel úgy csináltam meg, hogy van egy (jelen esetünkben magyar) nyelvi fájl, ami a szövegeket tartalmazza, ezért könnyen lehet újabb nyelvi fájlokat létrehozni, amivel az egész oldalt le tudjuk fordítani más nyelvekre is.

Az általánosan megírt függvények, eljárások miatt könnyen hozhatunk létre, bővíthetjük a gyűjteményt újabb algoritmusokkal, adatszerkezetekkel.

Az algoritmusokra lehetne még több teszteseteket létrehozni, hogy minél részletesebben be lehessen mutatni a kivételeseseteket.

Akár több programozási nyelven, több minta kódot is fel lehetne tenni az oldalra, hogy az iskolában tanított programozási nyelven is ki tudják próbálni a diákok az algoritmusokat. Ezáltal a tanár is könnyebben beépíthetné az informatika órába az oldal használatát.

Jóval nagyobb ívű tovább fejlesztési lehetőség, de a már megírt függvényekkel akár meg lehetne írni egy olyan összetettebb alkalmazást, programot, weboldalt is, amely egy pszeudókód alapján is be tudna mutatni egy algoritmust, úgy, hogy minden egyes utasítást, lépést követni lehessen vele, illetve az algoritmusba is jelezné, hogy hol tartunk éppen. Ehhez persze nagyon sok feladat van, többek között kéne egy pszeudokód fordító, amely minden egyes utasítást értelmez, illetve az oldalon megjeleníti az algoritmus jelenlegi helyzetét (változók nevei, értékei).

# Tanulási terv

Rengeteg óraterv, tematikus terv létezik a programozási tételek, algoritmikus gondolkodás tanításáról, így ezeket itt már nem szeretném részletesebben leírni. Csak vázlatosan említeném meg, hogy az egyes tanórákon miről lenne, lehetne szó. A tanórákon fontos, hogy a tanulók megismerkedjenek, kipróbálják, megértsék, majd elsajátítsák az egyes algoritmusok feladatát, működését és az köztes lépések, utasítások lényegét, funkcióját. Elsőként három órában szerepelnének a programozási tételek, majd egyben a halmazműveletek (melyek igazából ugyancsak programozási tételek kicsit komplexebb formában), utána lennének a rendezések (itt csoporttól, osztálytól függően kiválaszthatjuk, hogy a rendezések közül csak az egyszerűbbeket mutatjuk be, vagy esetleg a nehezebbek közül is kiválasztunk párat), adatszerkezetek (sor, verem) és a végén pedig akár egy tanórát is rá lehet szánni, hogy egy új anyagot készítsenek a diákok az oldalra, aminek mikéntjéről részletesebben is lesz szó a későbbiekben.

|  |  |
| --- | --- |
|  | Témák |
| 1. óra | Programozási tételek: Összegzés, Megszámolás, Maximum-kiválasztás |
| 1. óra | Programozási tételek: Keresés, Logaritmikus keresés |
| 1. óra | Programozási tételek: Másolás, Kiválogatás, Szétválogatás |
| 1. óra | Halmazműveletek: Unió, Metszet |
| 1. óra | Rendezések: Buborékos rendezés, Minimum-kiválasztásos rendezés |
| 1. óra | Opcionális: Rendezések: Gyorsrendezés, Összefésüléses rendezés |
| 1. óra | Adatszerkezetek: Sor, Verem |
| + 2 óra | Új anyag készítése (részletesebben később) |

## Egy adatszerkezet / algoritmus bemutatásának szerkezete

Egy tanórán, mikor szeretnénk bemutatni, megtanítani a gyerekeknek egy algoritmust, akkor szerintem a következőképpen lehetne ezt megtenni.

Először bevezetésként feldobunk egy valós, létező problémát, amiről tudjuk, hogy ezzel a legkönnyebb megoldani. Legjobb esetben akár egy diák ki is találhatja előzetes ismeretek nélkül, hogy hogyan is kéne megoldani.

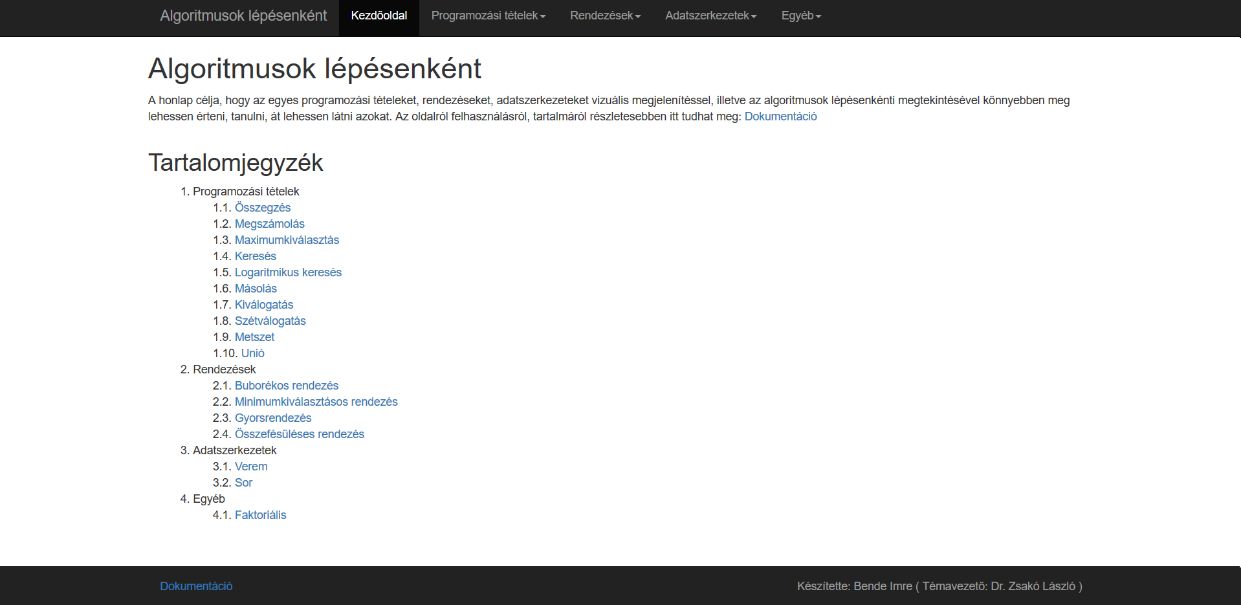
Ezután bemutatjuk, hogy mi is ez az algoritmus, majd iskolától, tanártól, diákoktól függően megmutatjuk a pszeudokódot / struktogramot / C++ kódot (bizonyos esetben, kerettől, környezettől függően akár mindegyiket is bemutathatjuk).

Ezután lépésenként is megnézzük egy-egy bemenetre a működését. Próbáljunk meg minél több esetet megnézni, minél szélsőségesebb esetekkel is akár! Remélhetőleg ennél a résznél már megértik az algoritmus működését, sajátosságát, így később saját maguk is tudják implementálni majd.

Az utolsó lépés, hogy különböző feladatokkal maguk is használják a tanult algoritmust, begyakorolják azt, így biztos tudást szereznek a témában.

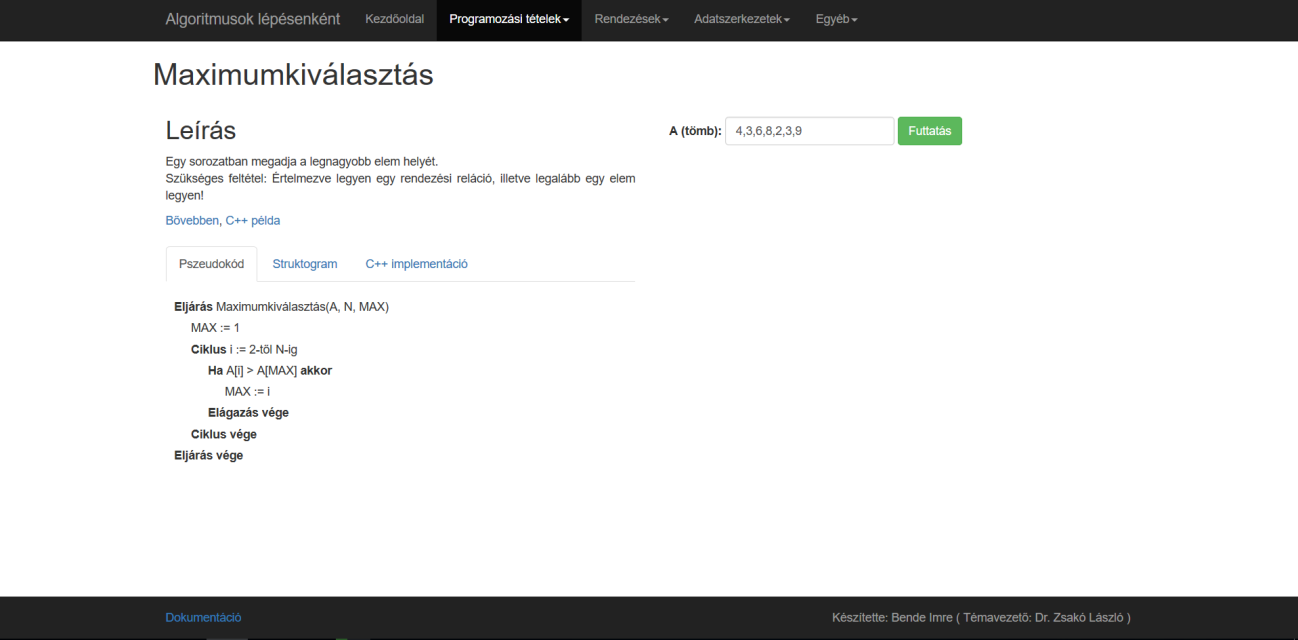
## Algoritmus megismerése az oldal segítségével

A következőkben szeretnék egy rövid leírást írni, hogy hogyan is tudnánk a honlapot felhasználva bemutatni egy algoritmus működését. Első lépésként a felső menüsorról, vagy a főoldalról kiválasztjuk a tanulni kívánt algoritmust.



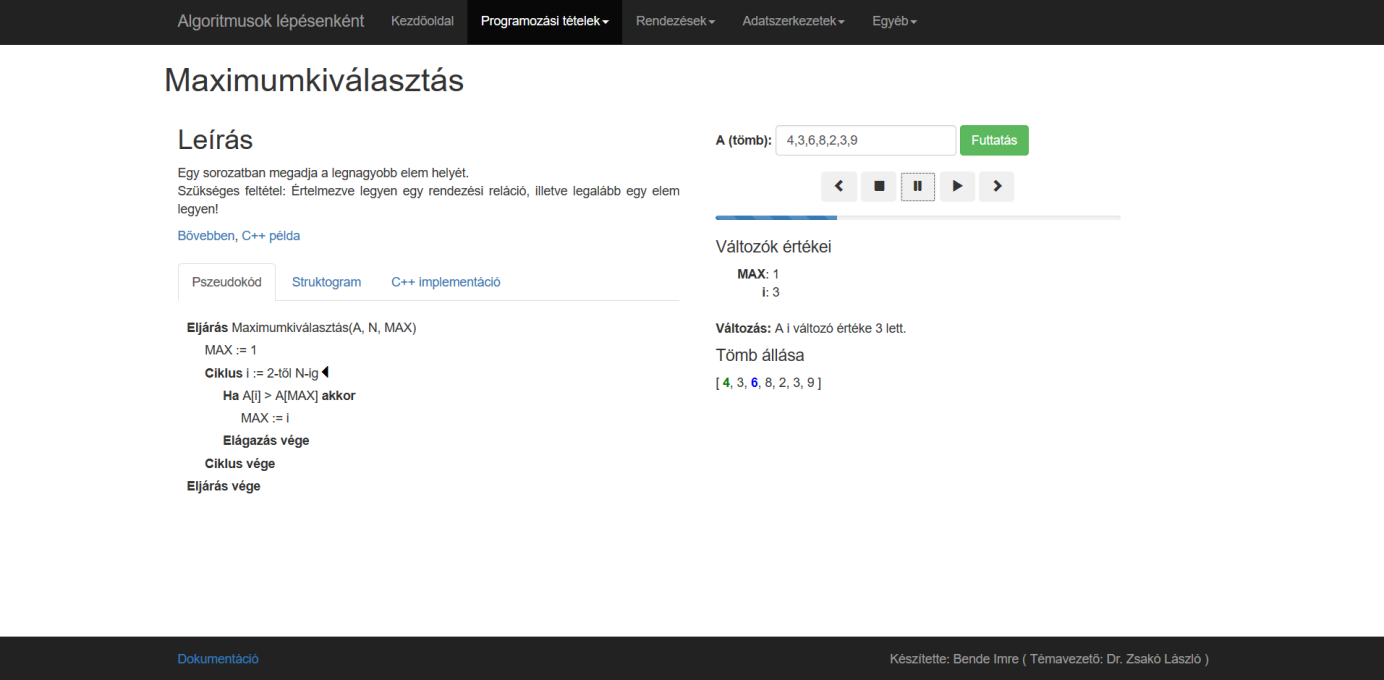
. ábra A program főoldala

Ha kiválasztottuk, akkor megjelenik a kiválasztott anyag oldala. Én most a maximum kiválasztás programozási tételen keresztül mutatom be az oldal működését.



. ábra A maximumkiválasztás programozási tétel oldala

Az oldal két részre oszlik. Bal oldalon szerepel: az algoritmus neve, leírása, C++ minta kód elérése, majd lejjebb az algoritmus maga 3 különböző nyelven is: pszeudokód, struktogram és a C++ kód. Jobb oldalon pedig a bemeneti mezők, és az algoritmus elindításáért felelős gomb van.



. ábra Maximumkiválasztás algoritmus egy lépése

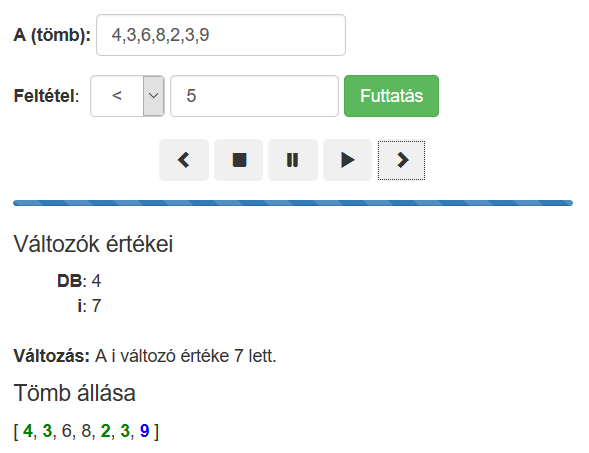
A futtatás gomb megnyomására elindul az algoritmus, mely három másodpercenként egy újabb lépést mutat meg. Hibás bemenet esetén közli ezt velünk az oldal, például, ha nem tömböt adunk meg vesszővel elválasztva, amikor tömb bemenetnek kéne lennie, akkor az algoritmus lépéseinek bemutatása helyett egy „Rossz bemenet” szöveg fog megjelenni. Öt gomb jelenik meg az input mező(k) alatt, melyek balról jobbra a következő szerepet töltik be: vissza egy lépéssel, stop (megállítja az algoritmus futását, illetve az elejére tekeri), szünet (megállítja a folyamatot), lejátszás (elindítja a folyamatot, ugyanúgy három másodperces időközzel halad), előre egy lépéssel. Alatta a változók, tömbök állásait, értékeit mutatja, bizonyos esetekben akár színezéssel is jelzi, ha valami fontosabb szerepben van, vagy valaminek hangsúlya van az adott lépésben (a következő fejezetben részletesebben, példákkal is bemutatom, hogy mely témáknál milyen szerepet játszik a színezés). Opcionálisan akár egy szöveg is megjelenhet, ami jelzi, hogy mi is történt az adott lépésben. Valamint egy-egy lépésnél a bal oldalon szereplő sorokat is jelzi, hogy éppen mely utasításokat vettük figyelembe, melyek azok, amik változtatták a változóink, tömbjeink értékeit.

## Néhány érdekesebb példa algoritmusokra

Ebben a fejezetben szeretnék bemutatni pár olyan algoritmust, részletet, aminél van valami kis „plusz”. Gondolok itt arra, hogy vannak nehezebben megérthető részek egy-egy lépésben és az alkalmazás ezeknél kiemel lényegesebb dolgokat, ezáltal téve könnyebben érhetővé az algoritmus megértését.

### Megszámolás

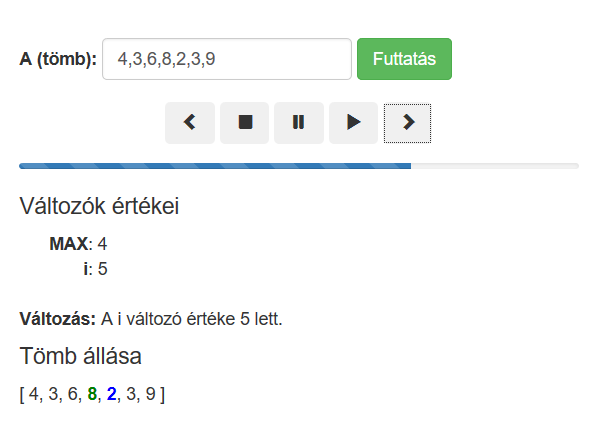
Megszámolásnál folyamatosan láthatjuk a db változó értékének változását, azonban akármely lépésnél is tartunk láthatjuk, hogy az előzetesen vizsgált elemek közül, melyekre igaz az előre meghatározott feltétel (zöld színnel jelzi a program ezeket az elemeket).



. ábra Megszámolás egy lépése

### Maximumkiválasztás

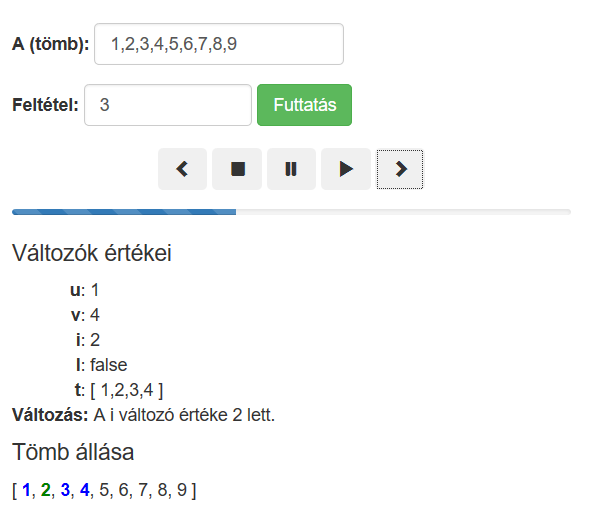
Maximumkiválasztásnál zölddel van jelölve, hogy az adott lépésben mely az eddigi legnagyobb elem, amit találtunk. Ha lineárisan haladva talál egy újabb maximális elemet, akkor az előző zölddel színezett elemről leveszi a színezést és ráteszi erre.



. ábra Maximumkiválasztás egy lépése

### Logaritmikus keresés

A logaritmikus keresésnél ugyebár mindig felezzük a lehetőségek számát, így az algoritmus log2n idő alatt megtalálja egy rendezett sorban a feltételnek megfelelő elemet. Az oldal nagyon jól ábrázolja (a képen is látszik), hogy kékkel jelöli, hogy már csak mely területen lehet a kívánt érték, zölddel pedig, hogy melyik elemet vizsgáljuk olyan szempontból, hogy a keresendő érték kisebb, nagyobb vagy egyenlő-e. Csak akkor kezd el futni az algoritmus, ha az input mezőbe lévő tömb elemei rendezetten helyezkednek el, ellenkező esetben jelzi a program, hogy hibás a bemenet.



. ábra Logaritmikus keresés egy lépése

### Buborékos rendezés, Minimumkiválasztásos rendezés

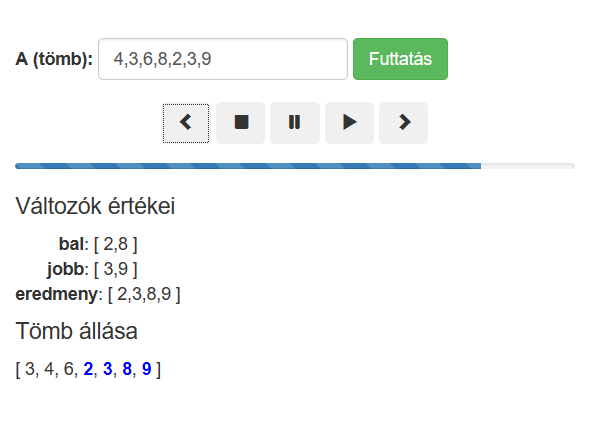
A buborékos rendezésnél és a minimumkiválasztásos rendezésnél is, ha egy elem már a helyére került (minimumkiválasztásnál az adott ciklusban lévő legkisebb elem, míg a buborékos rendezésnél a felbuborékoztatott érték), akkor azt a program zölddel jelzi. Így minden lépésnél látjuk, hogy mely elemek azok, amik már biztosan a helyükre kerültek. Illetve mindkét rendezésnél, a kicserélendő elemeket kékkel jelzi a cserénél.



. ábra Buborékos rendezés (balra) és a minimumkiválasztásos rendezés (jobbra) egy lépése

### Összefésüléses rendezés

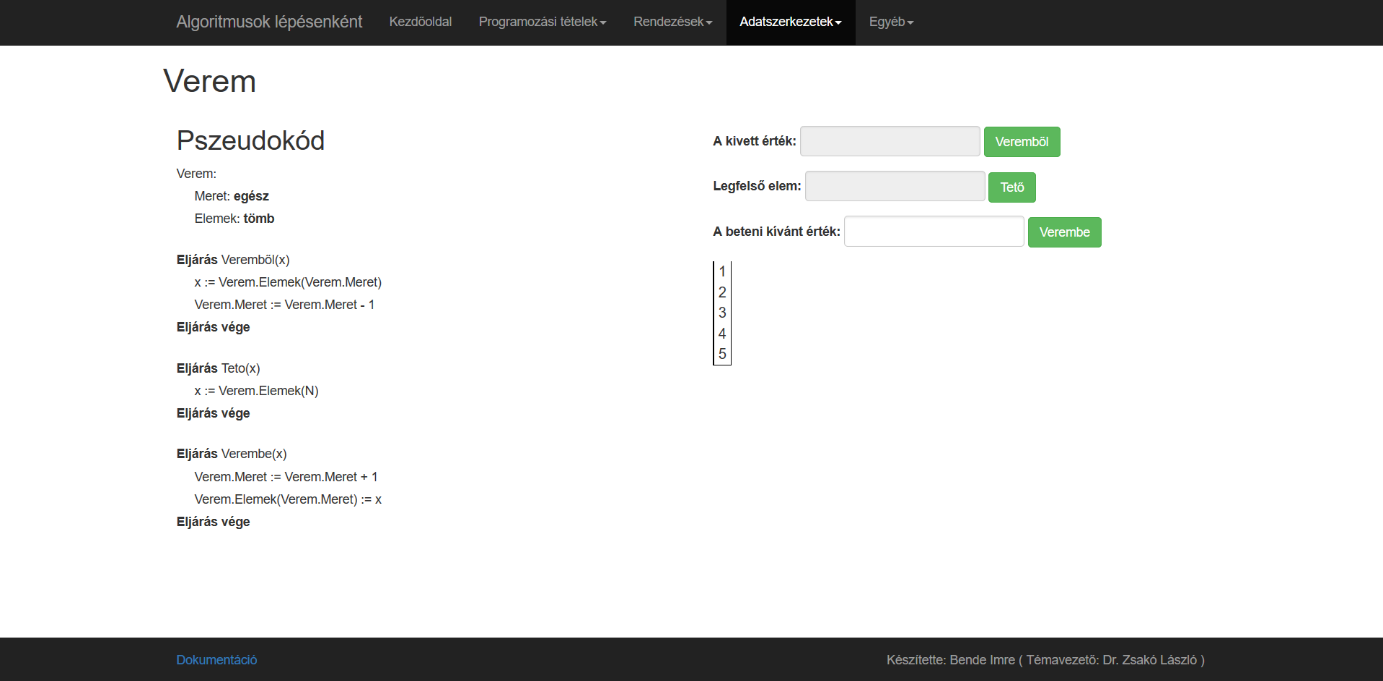
Összefésüléses rendezésnél mindig mutatja a program, hogy mely két (jobb, bal) tömböt fésüljük össze, illetve az összefésült eredmény tömb elemeit a végső, rendezett tömbben is jelzi kék színnel.



. ábra Összefésüléses rendezés egy lépése

## Adatszerkezet megismerése az oldal segítségével

Az adatszerkezetek oldalai nagyon hasonlóak az algoritmusok oldalaira. Bal oldalon szerepelnek az adatszerkezet adattagjai, illetve műveletei. Jobb oldalon pedig ki is tudjuk próbálni azokat. Többnyire egy-egy gombnyomással tudjuk meghívni a műveleteit, majd visszatérési értéktől függően, megjelenítjük az adatokat. Adatszerkezettől függően pedig vizuálisan is megjelenítjük, hogy hogyan is néz ki éppen az adatszerkezetünk az elemeivel.



. ábra A verem adatszerkezet oldala

## Példák adatszerkezetekre

### Verem

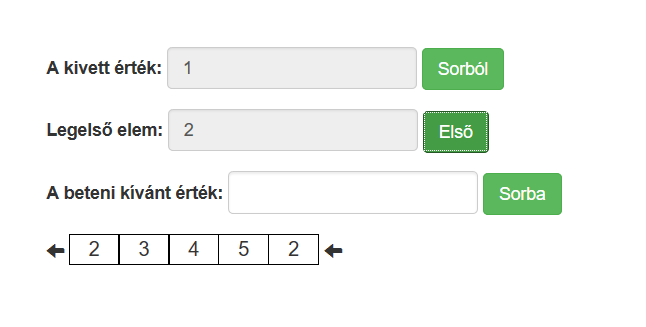
A veremnél három műveletet tudunk kipróbálni: Veremből, Verembe, Tető. A Tető és a Veremből műveleteknél a kivett elemet egy kikapcsolt állapotban lévő input mezőbe teszi, a Veremből műveletnél pedig még ki is veszi azt az elemet a veremből. A Verembe azt csinálja, hogy az input mezőbeírt értéket beleteszi a verem tetejére, majd az input mezőből ki is törli azt.



10. ábra Verem adatszerkezet műveletei, vizuális megjelenítése

### Sor

A sor oldala a veremhez nagyon hasonló elven működik. Három művelete szerepel az oldalon: Sorból, Sorba, Első. A Sorból és az első megjeleníti az adatszerkezet első elemét egy-egy input mezőbe, illetve a Sorból ki is törli az elemet a sorból. A Sorba művelet pedig az input mezőbe lévő elemet beteszi a sorba, majd kitörli a beviteli mezőbe írt értéket.



11. ábra Sor adatszerkezet kipróbálható műveletei, vizuális megjelenítése

## Új anyag készítése

Ami talán érdekessé teheti az ezen az alkalmazással való tanulást, hogy a témakör végén 1-2 tanórában (vagy akár a diákok otthon egyedül is megtehetik ezt) létrehozhatnak saját algoritmusokat, adatszerkezeteket amiket feltehetnek az oldalra is, így növelhetik az oldallal átadott tudást, saját maguk is fejlődnek, illetve mások is megnézhetik a diákok által létrehozott részletesen bemutatott tananyagot.

A következő részben lépésenként szeretném megmutatni, hogy általánosságban, hogy hogyan is kell egy új anyagot létrehozni.

1. Először egy minta HTML oldal alapján előállítjuk a vázat. Ebben kicseréljük az algoritmus specifikus nyelvi elemeket a sajátjainkra (vagy akár, ha nem akarunk vele a későbbiekben foglalkozni, beírhatjuk csak a magyar szavakat, kifejezéseket is). Illetve a „Futtatás” gombra egy saját függvényt teszünk rá (algoritmus neve + „Start”). Az algoritmusokat megírjuk (pszeudokód, struktogram, C++), és a programsorokhoz melyeknél kiírjuk majd a változók akkori állapotát, ahhoz teszünk glyphicon nyilakat (egyéni class-okkal), hogy majd a későbbiekben hivatkozva rá meg tudjuk mutatni, hogy az algoritmus lefutása során hol tartunk éppen.
2. A *locales* mappában a JSON fájlba/fájlokba beírjuk az általunk használandó új nyelvi elemeket.
3. Ha módosítottunk a HTML-ben egyes elemek kinézetén esetleg új CSS class-t akarunk létrehozni, akkor azt a *css/style.css* fájlban megtehetjük.
4. Az *esemenyek.js* fájlban felvesszük gombra felvitt eseményt. Meghívjuk benne a „fuggvenyElottiInit();” nevű függvényt, majd bemenet ellenőrzés után elindítjuk az algoritmus függvényét.
5. A saját algoritmusunknak létrehozhatunk egy új JavaScript fájlt, vagy akár már egy meglévőbe is írhatjuk (attól függően, hogy esetleg kötődik e már egy létező témához). Ebben implementáljuk az új algoritmust, majd az a legjobb, ha minél gyakrabban, minden egyes változásnál kiírjuk a változókat, tömböt/tömböket a „valtozokKiirasa()”, „tombKiirasa()” függvényekkel. A végén pedig meghívjuk a „megjelenitesInit();” nevű függvényt. Ha szeretnénk a magyarázat szövegét is módosítani, akkor a szovegValtoztatas() függvénybe egy új feltétel felvételével megtehetjük azt. Ezen függvényen keresztül történik még, hogy az oldalon lévő nyilak láthatóságát állítgatjuk. Tetszés szerint a nyilaknak adhatunk új neveket, de akár a régi neveket is használhatjuk, csak annyira kell figyelni, hogy ha újat hozunk létre, akkor itt mindenképpen csináljunk egy új elágazást.
6. Az algoritmust egy C++ kódban is implementálhatjuk, amelyet később az oldalon is el lehet érni (vagy akár más programozási nyelveken megírt forrásprogramokat is feltölthetünk).
7. Ezek után elkészültek a JavaScript fájljaink a gyökérmappában indítsuk el az uglify.sh nevű scriptet, amivel előállítjuk a minify-olt verzióit a fájljainknak. Ha nem tudtunk feltenni az UglifyJS 2-t, akkor online is minify-olhatunk (<https://javascript-minifier.com/>), illetve kikerülhetjük úgy is, hogy a HTML fájlunkban, a sima, nem tömörített verziókat hívjuk meg.
8. Az oldal kipróbálása, tesztelése után már csak az maradt, hogy feltöltjük a módosításainkat a git repoba. Melynek a lépései a következők:
   1. git commit –m „*algoritmus\_neve*” (lokálisan egy névvel ellátva eltároljuk a változtatásainkat, fontos, hogy a commit neve beszédes legyen, hogy a későbbiekben is tudjuk név alapján, hogy mik történtek, változtak benne)
   2. git pull (így letöltjük a jelenlegi repot, ha esetleg a készítés során más valaki már feltöltött módosítás, akkor ezzel így ezt letöltjük)
   3. git push (feltöltjük a módosításainkat, commit-unkat)
   4. szükség esetén, ha ütközés (conflict) van, akkor merge-léssel feloldjuk a többszörösen módosított sorokat a kódban

## Példa – n faktoriális

Példaként a folyamatot bemutatnám egy algoritmusra is, mely kiszámítja az n faktoriálist.

1. Létrehozom egy már elkészült példa alapján (mondjuk a *megszamolas.html* alapján) a HTML fájlunkat. (lsd. *faktorialis.html*). Az *index.html* fájlban az egyebek közé pedig felvettem linkként ezt az új oldalt:

<li><a href=**"faktorialis.html"** data-i18n=**"egyeb.faktorialis.nev"**></a></li>

1. *locales/hu/translation.json* fájlban beleteszem az új kifejezéseket, leírásokat.

"egyeb"**:** **{**

"faktorialis"**:** **{**

"leiras"**:** "Kiszámítja az N faktoriálist (N!)."**,**

"nev"**:** "Faktoriális"

**}**

**}**

1. Csak annyit módosítottam a style.css fájlunkban, hogy felvettem az input mezők közé a „#bemenetSimple”-t, hogy ugyanúgy nézzen ki mint a többi ilyen mező.

**#bemenet,** **#bemenet2,** **.**inputMezo**,** **#logKerBemenet,** **#bemenetSimple** **{**

**width: 200px;**

**display: inline;**

**}**

1. Létrehoztam a *faktorialis.js* nevű fájlt majd abban felvettem a *faktorialisStart()* nevű függvényt, amely a „Futtatás” gomb lenyomására fogja elindítani az algoritmust.

**function** faktorialisStart**(){**

fuggvenyElottiInit**();**

**if(**$**(**"#bemenetSimple"**).**val**())** **{**

faktorialis**(**$**(**"#bemenetSimple"**).**val**());**

$**(**"#eredmenyek"**).**removeClass**(**"hidden"**);**

$**(**"#hibasbemenet"**).**addClass**(**"hidden"**);**

**}**

**}**

Plusz még az *esemenyek.js* fájlba felvettem egy sort, ami alapból betesz egy értéket az oldalon lévő input mezőbe.

$**(**"#bemenetSimple"**).**val**(**"12"**);**

A *faktorialis.js* fájlban megcsináltam faktoriális algoritmusának leírását.

**function** faktorialis**(**n**){**

**var** fakt **=** 1**;**

valtozokKiirasa**({**nev**:** "fakt"**,** ertek**:** fakt**},** "elsoLepes"**);**

**for(var** i **=** 2**;** i **<=** n**;** i**++){**

valtozokKiirasa**({**nev**:** "i"**,** ertek**:** i**},** **{**nev**:** "fakt"**,** ertek**:** fakt**},** "ciklusLepes"**);**

fakt **\*=** i**;**

valtozokKiirasa**({**nev**:** "i"**,** ertek**:** i**},** **{**nev**:** "fakt"**,**ertek**:** fakt**},** "faktorialisLepes"**);**

**}**

megjelenitesInit**();**

**}**

Látszik, hogy igyekszem minél többször kiírni a változók állását: fakt változó inicializálásánál, ciklus lépésenként, illetve a fakt változó értékének változásánál is. Mindegyik változókiíráshoz tartozik egy kulcsszó is, ami a HTML-ben lévő kis nyilak megjelenítéséért felelősek. Ezáltal az algoritmusokban is látszanak, hogy az algoritmus éppen hol tart.

Az 1. pontban említett HTML fájlban meg is hívom ezt a fájlt a következőképpen: <script src=**"js/egyeb/faktorialis.js"**></script>.

A szovegValtoztatas() nevű függvényben felvettem a következő felvételt:

**else** **if(**$**(**"#valtozok .active"**).**hasClass**(**"faktorialisLepes"**)){**

$**(**".faktorialisLepesNyil"**).**removeClass**(**"hidden"**);**

magyarazatValtoztatas**(**"fakt"**,** $**(**"#valtozok .active .ertek.fakt"**).text());**

**}**

Ezáltal, ha faktorialisLepes van, akkor a magyarázatnál kiírja a fakt nevű változó értékét, plusz a faktorialisLepes class-szal rendelkező nyilakat fogja megjeleníteni az algoritmusokban.

1. Létrehoztam a faktorialis.cpp fájlt, amiben C++ nyelven van implementálva az algoritmus, melyet fordítás után ki is lehet próbálni.

#include <iostream>

**using** **namespace** std**;**

//Beolvasásá eljárása

void beolvasas**(**int **&**n**){**

cout **<<** "Adja meg az n-et!"**;**

cin **>>** n**;**

**}**

//Faktoriás kiszámítása

int faktorialis**(**int n**){**

int db **=** 1**;**

**for(**int i **=** 2**;** i **<=** n**;** i**++){**

fakt **\*=** i**;**

**}**

**return** fakt**;**

**}**

int main**(){**

int n**;**

beolvasas**(**n**);**

cout **<<** faktorialis**(**n**)** **<<** endl**;**

**return** 0**;**

**}**

1. Beleírom az uglify.sh nevű fájlba, hogy a *faktorialis.js*-t is átalakítsa, majd az oldalba is módosítom a JS fájl elérését. Lefuttatom az így elkészült scriptemet, ezáltal előállnak a „\*.min.js” fájlok. A következő sort írtam az uglify.sh nevű fájlba:

uglifyjs -o js**/**faktorialis.min.js **--**compress **--**mange **--** js**/**faktorialis.js

1. Vagy elküldjük egy olyan személynek a változtatásainkat, aki fel tudja tölteni a git repoba azokat, vagy ezt mi magunk is megtehetjük a következő utasítások lefutattatásával a git konzolban:
   1. git commit –m „*faktorialis*”
   2. git pull
   3. git push
   4. conflict feloldása szükség esetén

# Összegzés

Létrehoztam egy weboldalt, melynek segítségével akár 10 éves kor felett bárki egyszerűbben elsajátíthatja az algoritmikus gondolkodás sajátosságait, illetve megtanulhatja az alapvető algoritmusokat, majd azok működését, tulajdonságait is.

A felhasználók kellő ismerettel bővíthetik az algoritmusok tárházát, ezáltal maga is a honlap egy szerkesztőjévé válhat. Így az oldal lehetőségei végtelenek, és kellő társasággal egy nagy, algoritmusokban gazdag környezetet lehet létrehozni, ami által a látogatók folyamatosan láthatnak, tanulhatnak valami újat.

Nem csak a mögöttes tudást lehet bővíteni, hanem egyre több nyelvre lefordítva, akár egyre több helyre is eljuthat az alkalmazás, így akár több országban is tanulhatnak innen, illetve a későbbiekben akár ugyancsak bővíthetik a tartalmat.

Szerintem ez egy jó irányzat, amelyet manapság egyre több helyen használnak, így a szoftverek, alkalmazások mindig naprakészek tudnak lenni és sokkal használhatóbbak is lesznek mindenki számára. Plusz a forráskódot ismerve rengeteg is lehet tanulni akár egy következő, fejlettebb generáció elkészítésére.

# Felhasznált irodalom, jegyzék

Algoritmusok leírásai:

* <http://wiki.prog.hu/wiki/%C3%96sszef%C3%A9s%C3%BCl%C5%91_rendez%C3%A9s_(algoritmus)>
* <https://hu.wikipedia.org/wiki/Gyorsrendez%C3%A9s>
* <http://wiki.prog.hu/wiki/Bubor%C3%A9krendez%C3%A9s_(algoritmus)>
* <http://wiki.prog.hu/wiki/Kiv%C3%A1laszt%C3%A1sos_rendez%C3%A9s_(algoritmus)>

Programozási tételek részletesebb leírásai, használatai:

* <http://progalap.elte.hu/downloads/seged/eTananyag/>

Javascript könyvtárakról, keretrendszerekről információk, forrásaik:

* jQuery: <https://jquery.com/>
* Bootstrap: <http://getbootstrap.com/>
* i18n: <http://i18next.com/>
* UglifyJS 2: <https://github.com/mishoo/UglifyJS2>

Honlap github-os elérése:

* <https://github.com/imrebende/algotan>

Honlap kitelepített verziója:

* <http://ermi46.web.elte.hu/dev/algotan/>