Tartalomjegyzék

[Tartalomjegyzék 1](#_Toc465113257)

[Cél 2](#_Toc465113258)

[A program 3](#_Toc465113259)

[Követelmények 3](#_Toc465113260)

[Felhasznált eszközök, keretrendszerek 3](#_Toc465113261)

[A program szerkezete 4](#_Toc465113262)

[JavaScript fájlok 5](#_Toc465113263)

[CSS fájl(ok) 8](#_Toc465113264)

[Fejlesztési lehetőségek 9](#_Toc465113265)

[Tanulási terv 10](#_Toc465113266)

[Egy adatszerkezet / algoritmus bemutatásának szerkezete 10](#_Toc465113267)

[Új anyag készítése 11](#_Toc465113268)

[Példa – n faktoriális 12](#_Toc465113269)

[Felhasznált irodalom 16](#_Toc465113270)

# 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.

Emellett úgy fejlesztettem le az egyes funkciókat, hogy a hozzájuk tartozó függvényeket, eljárásokat könnyen újra lehessen használni, ezáltal könnyen tehetünk a weboldalra újabb algoritmusokat, amiket szeretnénk bemutatni, megtanítani a programom 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.

# 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 próbálni. Az oldal létrehozásához mindössze HTML, CSS, illetve az oldalon történő eseményekhez JavaScript programozási nyelveket használtam.

A weblap készítése során több külsős, ingyenesen felhasználható keretrendszert is felhasználtam. Ezekről részletesen nem írók, de pár gondolatot mindenképpen érdemes róluk mondani.

**jQuery** (2.1.4) JavaScript keretrendszerrel a DOM-ban (Document Object Model) lévő elemeket könnyebben, gyorsabban el lehet érni, illetve azok tulajdonságait átállítani. Rengeteg hasznos függvény van benne, ami a felhasználói élményt növeli, de mindezt kevés és érthető kódolással.  
(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 bonyolultabbá válik, azonban ezután a különböző nyelvekre való fordítás jóval egyszerűbb, könnyebb lesz. Így később a fordításhoz nem is kell programozói ismeret elég egy fájlban lévő szöveges elemeket lefordítani.  
(Honlap: <http://i18next.com/>)

**Bootstrap** rengeteg CSS tulajdonságot, class-okat tartalmaz, ami az egységes, mai szemmel szépnek mondható kinézetet adja. Illetve glyphiconokat tartalmaz, amik egy-egy oldalon lévő elemeknek még több jelentést tud adni, jobban szimbolizálja a gomb a 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. Plusz a később bemutatott egyedi algoritmus fejlesztésével akár a diákok saját kódjukat is feltölthetik, így egy egyre növekvő kódgyűjtemény lesz és még a diákot is büszkeséggel töltheti el, hogy mások is megnézhetik az ő kreálmányát.

Használtam az UglifyJS 2 nevű JavaScript minifier-t, amivel könnyen csökkenteni tudtam a JS fájlok méretét, ezzel csökkentve 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 könnyedén létre tudjuk hozni a tömörített 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.

*cpp* könyvtár tartalmaz C++ minta programokat, minden egyes algoritmushoz. Melyek megfelelő fordítóval egyből lefordíthatóak, majd futtatás során kipróbálhatóak.

*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. Itt akár adhatunk hozzá értelemszerűen új nyelvet, vagy esetlegesen nyelvi javításokat, bővítéseket tehetünk.

*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 az is, hogy még könnyebben lehessen újrafelhasználni az általam már létrehozott dolgokat.

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

### JavaScript fájlok

Elsősorban itt vannak a már részletesebben is leírt JavaScript könyvtárak, így itt már csak említésszerűen írnám ezeknek a fájljait: i18next-1.10.3.min.js, jquery-2.1.4.min.js, underscore.js.

A következőekben az általam írt lényegesebb függvényekről említek meg részleteket, hogy későbbiekben akár könnyebben lehessen őket újrafelhasználni, refaktorálni.

*eljarasok.js*

Nagyon sok függvény van implementálva ebben a fájlban, amiket könnyen lehet használni a későbbiekben is, 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.  
*tombEllenorzes(t)* – Ellenőrzi, hogy a bemenet tényleg 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.  
*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.   
*barValtoztatasa()* – Attól függően, hogy melyik állapotot mutatjuk, aszerint jeleníti meg a bar-t.  
*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.

*esemenyek.js*Itt állítom be, hogy az alapnyelv mindenképpen magyar legyen, későbbiekben ha több nyelven is elérhető lesz az oldal, akkor itt módosítani lehet rajta:

//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. Illetve a *play()* függvénnyel elindul, míg a *stop()* függvénnyel megáll az algoritmus bemutatása az adott lépésnél.  
Ez a JavaScript fájl még tartalmazó 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*

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

*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.

### 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-sal, illetve formázási beállítással.

## Fejlesztési lehetőségek

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

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

Az algoritmusokra lehetne még több tesztesetet létrehozni, hogy minél részletesebben be lehessen mutatni a kívételes eseteket.

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 továbbfejlesztési 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 a kódban.

# Tanulási terv

Rengeteg óraterv, tematikus terv létezik, így ezeket itt már nem szeretném részletesebben leírni. Csak vázlatosan említeném meg, hogy az egyes tanórákon miről lenne szó. A tanórákon fontos, hogy a tanuló megismerkedjen, kipróbálja, megértse, majd elsajátítsa az egyes algoritmusok feladatát, működését.

|  |  |
| --- | --- |
|  | 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 az alkalmazás segítségével, akkor szerintem a következőképpen lehetne ezt megtenni.

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

Majd a következőekben 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. Próbálunk meg minél több esetet megnézni, minél szélsőségesebb esetekkel is akár. Remélhetőleg ennél a résznél már megértik az algoritmust működését, sajátosságát, így később saját maguk is tudják implementálni majd.

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

## Új anyag készítése

Ami talán érdekessé tehet az ezzel való tanulást, hogy a végén 1-2 tanórában (vagy akár a diákok otthon egyedül is) létrehozhatnak saját algoritmusokat, amiket feltehetnek az oldalra is, így növelhetik az oldal átadott tudást, saját maguk is fejlődnek, illetve mások is megnézhetik a diák által létrehozott részletesen bemutatott tananyagot.

A következőekben 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. Az algoritmusokat megírjuk (pszeudokód, struktogram, C++), és a programsorokhoz melyeknél kiírjuk majd a változók akkori állapotát, ahhoz teszünk nyilakat (egyéni class-okkal), hogy majd a későbbiekben hivatkozva rá meg tudjuk mutatni, hogy az algoritmus lefutása alatt hol tartunk éppen.
2. A *locales* mappában beírjuk az általunk használandó új nyelvi elemeket a JSON fájlba/fájlokba.
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 módosítai, akkor a szovegValtoztatas() függvénybe egyúj feltétel felvételével megtehetjük azt. Ezen keresztül pedig az oldalon lévő algoritmusok nyilainak láthatóságát is tudjuk állítani.
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 fájloknak. 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, nem ezeket a verziókat hívjuk meg.
8. Az oldal kipróbálása, tesztelése után már csak az marad, hogy feltöltsük a módosításainkat a git repoba.

## Példa – n faktoriális

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

1. Létrehozom egy már elkészült példa alapján (mondjuk a *megszamolas.html*) 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
2. Jelen esetben nem volt szükség CSS módosításra, illetve új class-ok felvételére.
3. 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");

}

}

1. A *faktorialis.js* fájlban megcsináltam faktoriális algoritmusának leírását (az 1. pontban említett HTML fájlban meg is hívom ezt a fájlt).

**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 feleleősek. Ezáltal az algoritmusokban is látszanak, hogy az algoritmus éppen hol tart.

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 algoritmust, melyet fordítás után ki is lehet próbálni.

#include <iostream>

**using namespace** std;

//Beolvasás eljárás

void beolvasas(int &n){

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

cin >> n;

}

//Faktoriális 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. Lefuttatom az uglify.sh-t így előállnak a „\*.min.js” fájlok.
2. Vagy elküldjük egy olyan személynek a változtatásainkat, aki fel tudja tölteni a git repoba azokat, vagy ezt mi magunk is megtehetjük a következőféleképpen:
   1. git commit –m „*algoritmus neve*”
   2. git pull (így letöltjük a jelenlegi repot)
   3. git push (feltöltjük a módosításunkat, commit-unkat)
   4. szükség esetén, ha ütközés van, akkor merge-léssel feloldjuk a többszörösen módosított sorokat a kódban

# Felhasznált irodalom

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, forrásai:

* jQuery: <https://jquery.com/>
* Bootstrap: <http://getbootstrap.com/>
* i18n: <http://i18next.com/>