Eötvös Loránd Tudományegyetem

Informatikai Kar

Informatikatudományi Intézet

Algoritmusok és alkalmazásaik Tanszék

DAG legrövidebb utak szemléltetése grafikus webalkalmazással

Szerző:

Kis Botond

Programtervező informatikus BSc.

Témavezető:

Ásványi Tibor Dr.

Egyetemi docens, Informatikus PHD

Budapest, 2024

Tartalomjegyzék

1. Bevezetés

A szakdolgozatom célja a Directed Acyclic Graphs (DAG) legrövidebb utak optimalizálásának szemléltetése egy grafikus webalkalmazáson keresztül. A DAGok irányított, körmentes gráfok. A szakdolgozat során egy JavaScript nyelven írt programot fejlesztek, amely lehetővé teszi a felhasználók számára, hogy megadják a DAG gráfjaikat és azokhoz kapcsolódó adatokat.

A fő algoritmus, amelyet implementálok és vizsgálok, a DAGshP algoritmus lesz, mely első lépésként meghatározza az s-ből elérhető csúcsokat, majd ezeket sorba rendezve optimalizálja az útvonalakat. A megoldás tartalmazná a gráf részleges topologikus rendezését is. A program lehetővé teszi az adatok megadását, és szemlélteti azokat a felhasználó számára a grafikus felületen keresztül. Emellett a program magyar és angol nyelven is elérhető lesz, hogy magyar és külföldi hallgatók is tudják használni algoritmusok és adatszerkezetek tanulmányai alatt.

2. Elméleti háttér

2.1. Alapfogalmak

Gráf: A gráfok csúcsok, csúcsok és rajtuk értelmezett összeköttetések, élek halmaza.

Irányított gráf: Az irányított gráf olyan gráf, amely irányított élekkel összekapcsolt csúcsok halmazából áll, amelyeket gyakran íveknek neveznek.

2.2. DAG Alapfogalmak

A Directed Acyclic Graph (DAG) egy olyan irányított gráf, amelyben nincsenek körök, vagyis nincs olyan útvonal, amelyben a kiindulási csúcsba vissza lehetne térni. Az irányítottság és ciklusmentesség miatt a DAG különösen hasznos bizonyos problémák modellezésében, például ütemezési problémákban, függőségi rendszerekben, változások és verziók nyomon követésében, valamint a legfontosabb csúcsok meghatározásában.

2.3. Topologikus rendezés

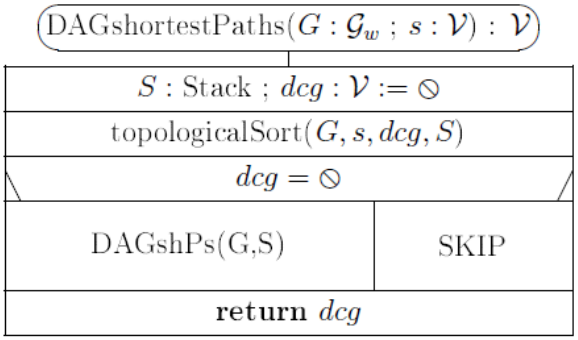
A topologikus rendezés egy olyan sorrend a csúcsok között, amelyben minden él iránya a csúcsok sorrendjét követi. Ezt a sorrendet könnyű egy DAG-ban meghatározni egy mélységi bejárás (DFS) segítségével, ahol a csúcsokat azok befejezési ideje szerint rendezzük csökkenő sorrendben. A topologikus rendezés számos feladatban hasznos, például függőségek kezelésében és feladatütemezésben.

2.3.1. Mélységi Bejárás (DFS) Alapú Topologikus Rendezés

Az algoritmus a csúcsok bejárása közben minden csúcs befejezési idejét rögzíti, és a visszafelé vezető élek hiánya miatt a DAG-ban a rendezés biztosítja, hogy minden forráscsúcs a célcsúcs előtt szerepel.

2.4. DAG Legrövidebb utak egy forrásból (DAGshP) [1]

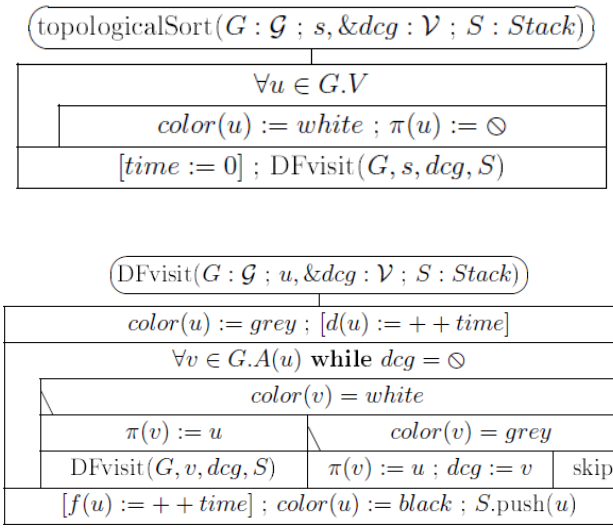
,,Előfeltétel: A G : G_w gráf irányított, és a gráfban nincs s-ből elérhető irányított kör. Ebben az esetben nincs a gráfban s-ből elérhető negatív kör sem, így a legrövidebb utak egy forrásból feladat megoldható. Ez az algoritmus ellenőrzi az előfeltételét. Ha teljesül, a DAGshortestPaths() függvény null értékkel tér vissza. Különben megtalál egy irányított kört, aminek egyik csúcsával tér vissza. Ebből indulva a π címkék mentén a kör fordított irányban bejárható.

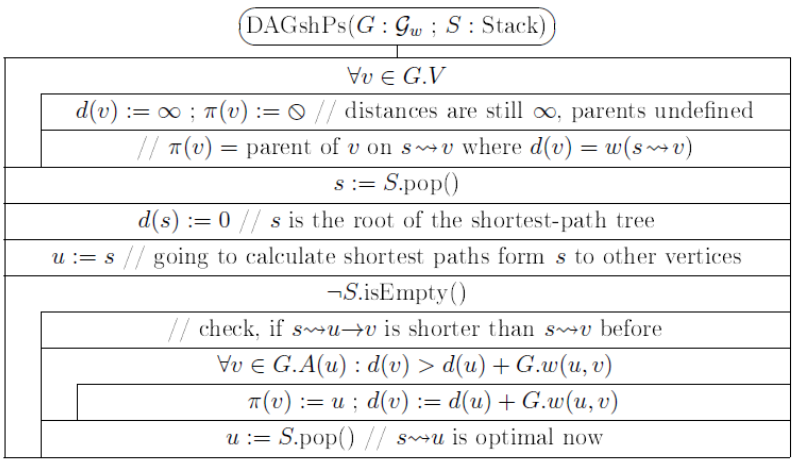


1. ábra: DAGshP Algoritmus stuktogramja

A topologikus rendezés csak az s-ből elérhető részgráfot próbálja topologikusan rendezni.

A time nevű változóra - amit az egyszerűség kedvéért globálisnak képzelünk - és a hozzá kapcsolódó utasításokra csak a topologikus rendezés szemléltetésének megkönnyítése végett van szükségünk. Ezek az implementációkból elhagyhatók, ezért a vonatkozó utasításokat szögletes zárójelbe tettük. “





2. ábra: DAGshP Algoritmus stuktogramja

2.4.1. A táblázat felépítése

A táblázat célja, hogy egy DAG-ra alkalmazott legrövidebb utak egy forrásból algoritmus eredményeit jelenítse meg. Az alábbiakban bemutatom, hogyan fog kinézni:

1. **Fejléc (Oszlopok)**: A táblázat első sora a vízszintes fejléc, ahol minden egyes csúcs neve szerepel, amik a gráfhoz adás sorrendjét követik. Ezek az oszlopok a gráf összes csúcsát reprezentálják, azaz minden olyan csúcsot, amelyhez elérési útvonalat keresünk a kezdő csúcsból.

2. **Első Sor (Kezdőértékek)**: Az első sor a kezdőállapotot mutatja. Az első (kezdő) csúcs távolsága itt 0, míg minden más csúcsnak ∞ (végtelen) távolsága van, mivel még nincs útvonal meghatározva hozzájuk.

3. **Sorok (Topologikus Sorban Rendezetten)**: Az összes további sor egy-egy csúcs kiterjesztését reprezentálja a topologikus rendezés sorrendjében (kivéve a topologikus rendezés utolsó csúcsát, arra a sorra már nincs szükségünk, mivel addigra már az összes elérhető csúcsba meghatároztuk az optimális utat), és minden sor elején a csúcs neve szerepel, valamint annak távolságértéke. A csúcs minden oszlopában megjelenik az az érték, amely az aktuális csúcsból az adott célcsúcsba vezető él költségével növelt értéket mutatja.

    - **Minimális Érték**: Minden oszlopban csak a legkisebb elérhető távolságot jeleníti meg, vagy üres cellát, ha nincs kedvezőbb érték az aktuális csúcstól indulva.

4. **Eredmény Sor (Legutolsó Sor)**: Az utolsó sor minden egyes csúcsnak a kezdő csúcstól vett távolságát (d) és a hozzá vezető legrövidebb úton a közvetlen megelőzőjét (π) jeleníti meg. Ha egy csúcs nem érhető el, akkor az d-értéke végtelen (∞) marad.

Ez a táblázat vizuálisan ábrázolja a kezdő csúcsból az összes többi csúcsba vezető legkedvezőbb útvonalakat és az adott csúcsoknak a közvetlen előzmény csúcsát (3. ábra). A táblázat segíti a felhasználót az optimális útvonal követésében és a csúcsok közötti legrövidebb távolságok azonosításában.

DAG legrövidebb út algoritmus

	a	b	c	d	e	f
	0	∞	∞	∞	∞	∞
a : 0		1 a				
b : 1			1 b	3 b		
c : 1					2 c	
e : 2						
Eredmény:	0 ⊗	1 a	1 b	3 b	2 c	∞

3. ábra: DAGshP algoritmus táblázata példa adatokkal

3. Felhasználói dokumentáció

A felhasználói dokumentációban az alkalmazás futtatásához szükséges követelményekről, felhasználási lehetőségeiről és módjairól esik szó, valamint sor kerül a felhasználói felület részletes bemutatására.

3.1. Bevezetés

A program célja, hogy segítse a felhasználót a DAG legrövidebb utak algoritmusának megértésében egy könnyen kezelhető, grafikus felülettel.

3.1.1. Rendszerkövetelmény

A Programnak nincsenek különleges erőforrás igényei csak egy modern böngésző futtatásához szükséges hardveres/szoftveres erőforrásokat igényli.

- Képernyőfelbontás: Minimum 1280x720 pixel.
- Böngészőkövetelmények:
  - Modern Böngésző: Chrome 80 (vagy újabb), Firefox 75 (vagy újabb), Microsoft Edge.
  - JavaScript engedélyezve: A böngészőben szükséges, mivel React alapú alkalmazás.

3.1.2. Program futtatása

A program futtatásához a felhasználónak a következő weboldalt kell megnyitnia:

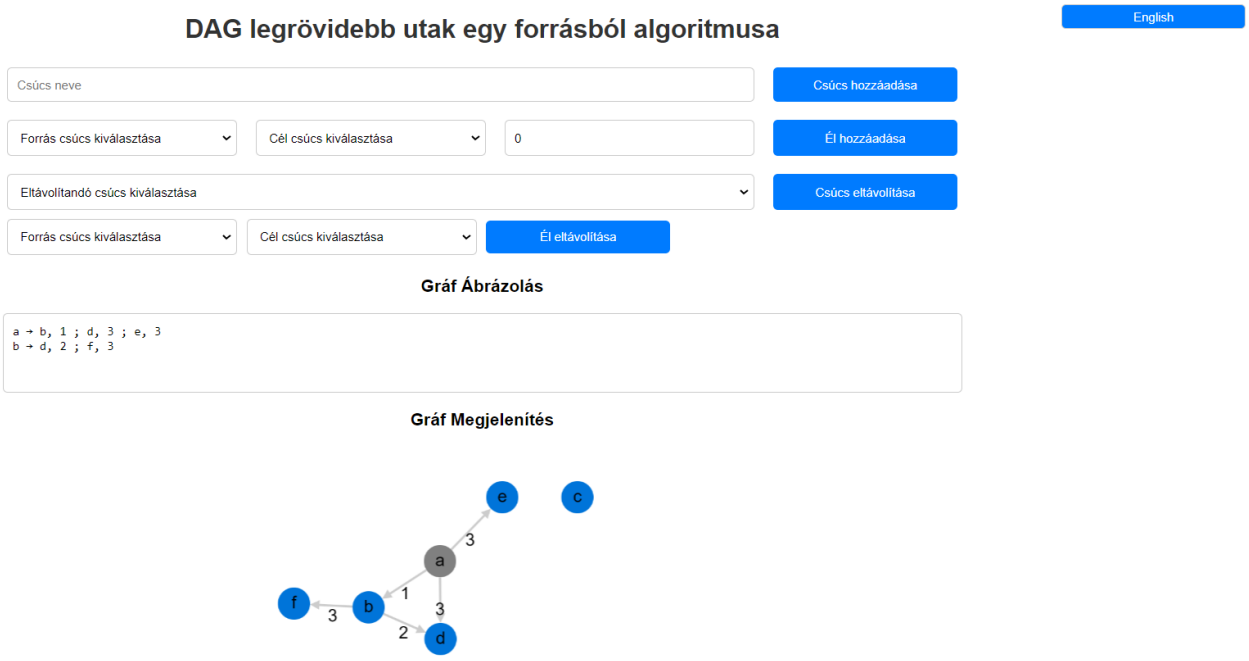
http://botondkis.web.elte.hu/

3.2. Felhasználási útmutató

A felhasználási útmutatóban bemutatom a programablak felépítését.

3.2.1. Program felülete

A program egyablakos rendszerű. Az ablak egy felső és alsó részre van osztva, a felső részben a vezérlőelemek láthatók (4. ábra), alatta pedig a DAG legrövidebb utak algoritmusának adatai és az eredmény gráf látható (5. ábra).



4. ábra: A képen a vezérlőelemek és az aktuális gráf látható.

Topológiai rendezés

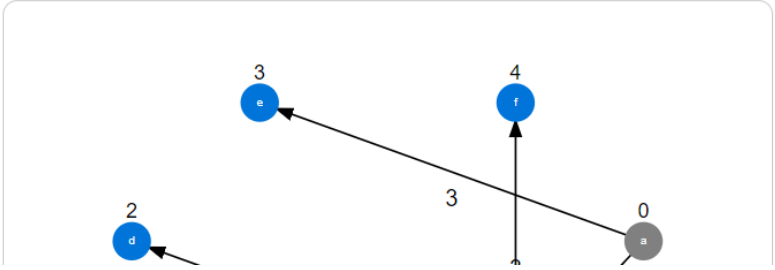
```
a, e, b, f, d
```

Csúcs kezdő- és befejezési idők

- a: 1/10
- b: 2/7
- d: 3/4
- f: 5/6
- e: 8/9

DAG legrövidebb út algoritmus

	a	b	c	d	e	f
	0	∞	∞	∞	∞	∞
a : 0		1 a		3 a	3 a	
e : 3						5 e
b : 1				2 b		4 b
f : 4						
Eredmény:	0 ⊗	1 a	∞	2 b	3 a	4 b



5. ábra: A képen a DAG legrövidebb utak algoritmusának adatai és az eredmény gráf látható

3.2.2. Vezérlőelemek használata

3.2.2.1. Csúcs hozzáadása

A csúcs hozzáadása gomb megnyomásával hozzáadhatunk egy csúcsot a gráfunkhoz. Ha a gráf még üres akkor ez a hozzáadott csúcs lesz a kezdő csúcs. A kezdő csúcs szürke színű lesz, míg a többi csúcs kék színű lesz. A csúcsok nevének egyedinek kell lennie. A csúcs beszúrásának algoritmusának egyik feltétele, hogy ellenőrizze, hogy a név egyedi.

6. ábra: Csúcs hozzáadása

3.2.2.2. Csúcs eltávolítása

A csúcs eltávolítása gomb megnyomásával eltávolíthatunk egy csúcsot a meglévő gráfunkból, ha a kezdő csúcsot távolítjuk el, akkor az időrendi sorrendben hozzáadott következő csúcs lesz az új kezdő csúcs.

7. ábra: Csúcs eltávolítása

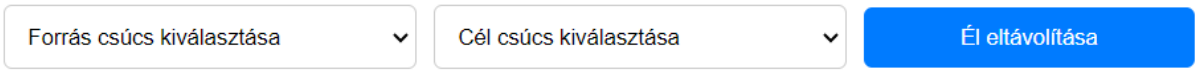
3.2.2.3. Él hozzáadása

Az él hozzáadása gomb megnyomásával hozzáadhatunk egy élt két meglévő csúcshoz. Az élek hozzáadásánál 3 rész van. A forrás csúcs kiválasztása, a célcsúcs kiválasztása és az élsúly meghatározása. A forrás csúcsokat és a célcsúcsokat a már meglévő csúcsok közül választhatjuk ki. Oda kell figyelni élek hozzáadásánál arra is, hogy nehogy ciklus keletkezzen a gráf forrás csúcsból elérhető részében. Nem engedélyezzük, hogy egy csúcs önmagába mutató éllel rendelkezzen.

8. ábra: Él hozzáadása

3.2.2.4. Él eltávolítása

Az él eltávolítása gomb megnyomásával eltávolíthatunk éleket a gráfból. Kiválasztjuk az éleket, hasonló módon, mint az élek hozzáadásánál, élsúly megadása itt nem szükséges. Ha az él nem létezik akkor kapunk egy figyelmeztetést.



9. ábra: Él eltávolítása

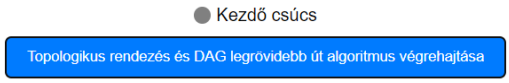
3.2.2.5. Nyelv kiválasztása

Lehetőségünk van a nyelv kiválasztására. A program magyarul indul, de lehetőségünk van angol nyelvre váltani.

10. ábra: Nyelv választása

3.2.2.6. Topologikus rendezés és DAG legrövidebb út algoritmus végrehajtása

Topologikus rendezés és DAG legrövidebb út algoritmus végrehajtása gomb megnyomásával elindítjuk a program lényeges részét. Láthatjuk a gomb megnyomása után a topologikus rendezés eredményét, a csúcs kezdő- és befejezési időket, a DAG legrövidebb út algoritmusát és az eredmény gráfot. Két ellenőrző feltétel is tartozik ehhez az algoritmushoz. A gomb megnyomásánál nem lehet a gráf üres és nem lehet ciklus a gráfban.

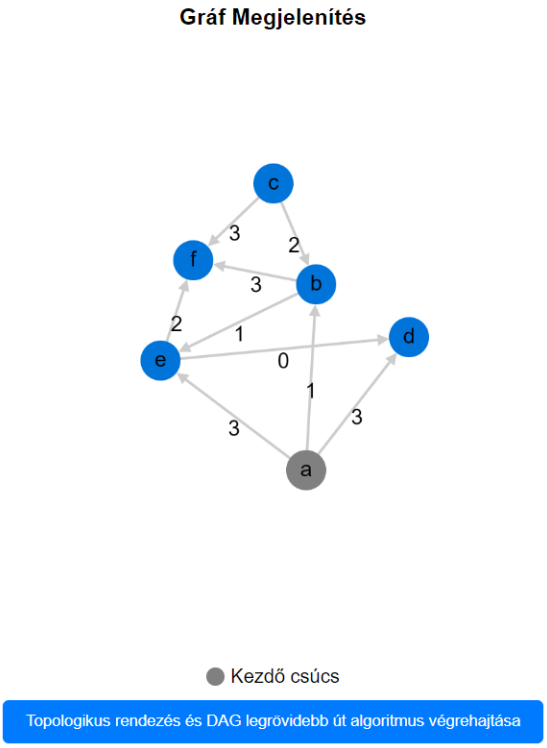


11. ábra: 3.2.2.6. Topologikus rendezés és DAG legrövidebb út algoritmus végrehajtása

3.2.3. A gráfok elemei és jelentésük

A programban két gráf van. A bemeneti gráf, ahol követhetjük a gráfunk csúcsait és éleit. A második pedig az eredmény gráf (a legrövidebb utak fája).

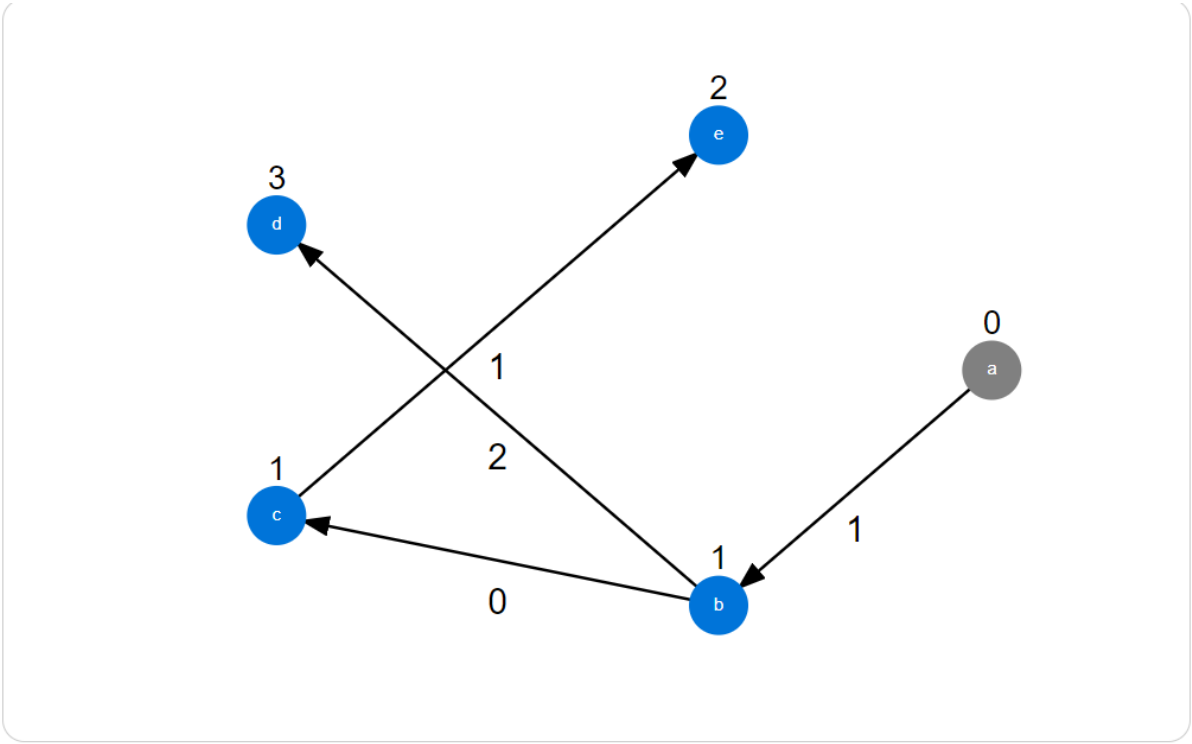
3.2.3.1. Bemeneti gráf



12. *ábra: Bemeneti gráf*

Ebben a gráfban megfigyelhetők a gráf szerkezetét alkotó csúcsok, élek és élsúlyok. A gráf pozíciója interaktívan módosítható, a csúcsok egyenkénti helyzetének megváltoztatása bal kattintással érhető el, valamint a teljes gráf pozíciója is áthelyezhető. Emellett lehetőség van a gráf nézetének nagyítására és kicsinyítésére, lehetővé téve a részletesebb vagy átfogóbb áttekintést. A ‘Topologikus rendezés és DAG legrövidebb út algoritmus végrehajtása’ gomb megnyomása után a csúcsok felett megjelennek a kezdő- és befejezési idők.

3.2.3.2. Eredmény gráf



13. *ábra: Eredmény gráf*

Az eredmény gráfban láthatjuk a DAG legrövidebb utak algoritmusának az eredményét. Az adatok leolvashatóak a felette lévő táblázatról. A csúcsok felett itt nem a kezdő- és befejezési idők látszódnak, hanem az érték amennyire a kezdő csúcstól vannak.

4. Fejlesztői dokumentáció

A fejlesztői dokumentációban fogom részletezni a program szerkezetét, felépítését a választott technológiákat, a tesztelést és magát a feladatot.

4.1. Feladat

Az alkalmazásom célja, hogy a felhasználó könnyebben meg tudja érteni a DAG legrövidebb utak egy forrásból algoritmus működését. Ebben segítséget nyújt, hogy az általa megszerkesztett gráfokon tudja futtatni az algoritmusokat.

4.2. A választott technológia

A választott programozási nyelvem a JavaScript, azon belül egy népszerű könyvtára, a React (react.js), a fejlesztési környezet pedig a Visual Studio Code. Több érv szól a JavaScript mellett, mivel az egyetemi tanulmányaim alatt alaposan megismertem és elsajátítottam. A JavaScript kiváló a webfejlesztésre, különösen dinamikus felhasználói felületek létrehozására és modern webes alkalmazások fejlesztésére. Ezen technológiai választások segítségével az alkalmazásom hatékonyan és könnyen használható módon valósítja meg a Directed Acyclic Graph-ok (DAG-ok) elemzését és a legrövidebb út optimalizálását egy grafikus webes felületen.

4.3. A tervezés során felmerült igények, problémák és megoldásaik

A szakdolgozat célja egy interaktív grafikus alkalmazás létrehozása, amely hatékonyan segít megérteni a legrövidebb út kiszámítását egy irányított aciklikus gráfban (DAG). A fejlesztés során a következő problémák merültek fel, melyekre megoldásokat kellett találni:

1. **Nézet és adatlogika elkülönítése**: A kód átláthatóságához és a felhasználói élmény javítása érdekében célszerű a felhasználói felületet és az adatlogikát elkülöníteni, ehhez pedig a React.js keretrendszer használata ideális választás volt.

2. **Változó nyelvi beállítások kezelése**: Az alkalmazás többnyelvű támogatást kapott, hogy a felhasználók magyar és angol nyelven is hozzáférjenek az eszközhöz. A nyelvváltó gomb megvalósításával és a fordítások kezelésére szolgáló translations objektummal a nyelvi beállítások dinamikusan frissülnek a felhasználói élmény megszakítása nélkül.

3. **Gráf és algoritmusok vizualizálása**: A gráf vizualizációja érdekében a Cytoscape.js könyvtár lett integrálva. Ez lehetővé teszi az interaktív csúcsok és élek megjelenítését, valamint a topologikus sorrend és a legrövidebb útvonal vizuális ábrázolását is. Kihívást jelentett a megfelelő megjelenítési forma kiválasztása, hogy minél jobban látható legyen minden objektum és hogy minden elem megjelenjen (irányított élek, élsúlyok, kezdő- és befejezési idők, stb.) A megjelenítésen kívül kihívást okoztak még a műveletek folyamatos frissítése, amit az adat- és állapotkezelés megoldásával sikerült optimalizálni.

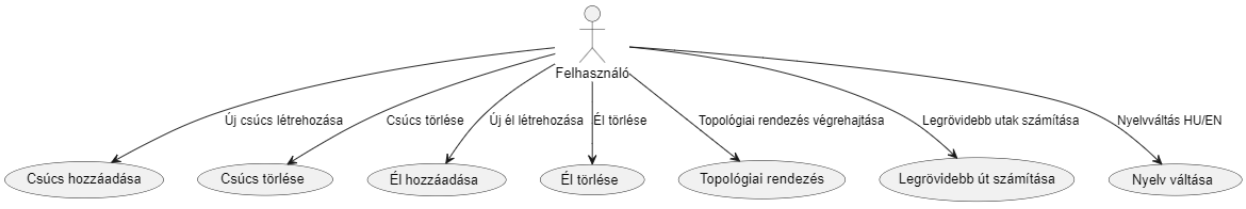
4.3.1. Tervezési minta

A Model-View-Controller (MVC) tervezési mintát alkalmaztam, hogy az alkalmazás logikai rétegei jól elkülönüljenek, ezáltal elősegítve a fenntarthatóságot és a kód újrahasznosíthatóságát:

- **Modell**: Ez a réteg tartalmazza a gráf csúcsait, éleit, valamint az azok közötti kapcsolatokat, és képes tárolni a topologikus sorrendet és a csúcsok kezdési/befejezési idejét.
- **Nézet**: A GraphBox komponens felelős a felhasználói felület megjelenítéséért, beleértve a gráfok rajzolását, az interaktív nyelvváltást, a csúcsok és élek létrehozását és eltávolítását.
- **Vezérlő**: Az eseménykezelő funkciók (pl. addNode, removeNode, addEdge) irányítják a felhasználói interakciókat és meghívják a szükséges műveleteket, továbbítva a kapott eredményeket a nézet rétegnek.

Ez a minta nagyban megkönnyíti a kód karbantartását és a további bővítéseket, hiszen az egyes rétegek függetlenek egymástól és könnyen átlátható kódot kapunk.

4.3.1.1. Use Case diagram (Használati eset diagram)



14. ábra: Use case diagram

4.3.1.2. User Stories (Felhasználói történet)

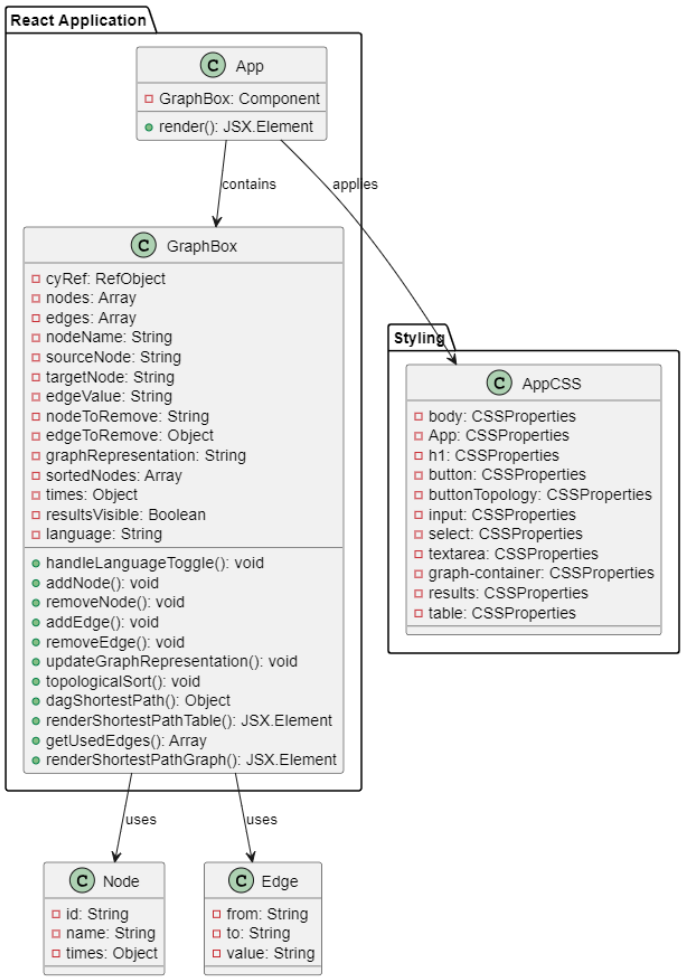
1. táblázat: User storyk

Funkció	Rövid leírás	Given-When-Then
Csúcs hozzáadása	Új csúcsot adhatunk a gráfhoz a megadott névvel.	**Given:** A felhasználó beírja a csúcs nevét. **When:** A felhasználó a hozzáadás gombra kattint. **Then:** Az új csúcs megjelenik a gráfban.
Csúcs eltávolítása	Törli a megadott csúcsot a gráfból.	**Given:** A felhasználó kiválasztja a törölni kívánt csúcsot. **When:** A felhasználó a törlés gombra kattint. **Then:** A csúcs és annak összes kapcsolata törlődik.
Él hozzáadása	Új élt adhatunk két csúcs közé a megadott súllyal.	**Given:** A felhasználó kiválaszt két csúcsot és beírja a súlyt. **When:** A felhasználó az él hozzáadás gombra kattint. **Then:** Az él megjelenik a gráfban a megfelelő csúcsok között.
Él eltávolítása	Törli a megadott élt a gráfból.	**Given:** A felhasználó kiválasztja az eltávolítandó élt. **When:** A felhasználó a törlés gombra kattint. **Then:** Az él törlődik a gráfból.

Topologikus rendezés	A gráf topologikus rendezését jeleníti meg, és időbélyegeket rendel minden csúcshoz.	***Given:*** *A gráf csúcsokkal és élekkel rendelkezik.* ***When:*** *A felhasználó a* Topologikus rendezés és DAG legrövidebb út algoritmus végrehajtása *gombra kattint.* ***Then:*** *A topologikus sorrend megjelenik a csúcsok kezdő és befejező időpontjaival.*
Legrövidebb út számítása	Kiszámítja a legrövidebb út távolságát minden csúcs számára egy adott kezdőcsúcstól.	**Given:** A topologikus rendezés eredménye elérhető. **When:** A felhasználó a Topologikus rendezés és DAG legrövidebb út algoritmus végrehajtása gombra kattint. **Then:** A csúcsok közötti legrövidebb út hossza megjelenik.
Nyelv váltása	A felhasználó átválthat angol és magyar nyelv között.	**Given:** Az alkalmazás betöltve van. **When:** A felhasználó a nyelvváltó gombra kattint. **Then:** Az alkalmazás nyelve megváltozik.

4.4. A program felépítése

A programot 3 fő részre lehet szétosztani. Vezérlőkre, nézetre és az algoritmusokat végrehajtó függvényekre.



15. ábra: UML diagram

4.4.1. Vezérlők:

1. addNode függvény

Az addNode függvény felelős a megadott csúcsokat hozzáadni a gráfhoz. Létrehoz egy új node (csúcs) Objektumot, és hozzáadja egy ‘nodes’ tömbhöz, ha még nincs ilyen nevű csúcs a gráfban.

2. removeNode függvény

A removeNode függvény egyszerűen kitörli a kiválasztott csúcsot a ‘nodes’ tömbből.

3. addEdge függvény

Az addEdge függvény hozzáad egy új kapcsolatot, élt a gráfhoz, ha megfelel a következő feltételeknek: A megadott forrás csúcs és célcsúcs nem egyezik és a megadott él még nem létezik (megváltozott élsúly nem számit). Ha megfelelt ezeknek a feltételeknek, akkor hozzáadja az 'edges' tömbhöz az új élt.

Mind a három vezérlő függvény hatással van a 'nodes' és 'edges' tömbökre, emiatt az összes csúcs és él változásnál meghívunk egy useEffect nevű react specifikus hook-ot, amelyet arra használunk, hogy mellékhatásokat végezzünk, például frissítéseket, adatbetöltéseket, vagy az állapot megváltozásakor történő funkcióhívásokat. Ebben az esetben a useEffect a updateGraphRepresentation függvényt hívja meg minden 'nodes' és 'edges' változásra.

```
useEffect(() => {
  updateGraphRepresentation();
}, [nodes, edges]);
```

16. ábra: useEffect

Ezen a useEffect-en kívül még egyszer használunk useEffect-et, amikor a bemeneti gráfot rajzoljuk ki.

4. removeEdge függvény

A removeEdge függvény egyszerűen kitörli a kiválasztott élt az 'edges' tömbből, ha létezik a megadott él, ha nem létezik akkor egy figyelmeztetéssel tér vissza.

5. handleLanguageToggle függvény

A handleLanguageToggle függvény a nyelv változtatását figyeli. Az adatokat a 'translations' nevű objektumból kapja.

4.4.2. Nézet

1. return

A felhasználói felület összes eleme a return blokkban található, amely a kódom legvégén helyezkedik el. Az alábbi részek tartalmazzák az alkalmazás funkcióit és vizuális elemeit, amelyek a felhasználóval való interakciót szolgálják.

return elemei:

- **Nyelvváltás gomb**

  A handleLanguageToggle függvény által kezelt gomb a felhasználói felület jobb felső sarkában helyezkedik el, és lehetőséget ad a felhasználónak az alkalmazás nyelvének megváltoztatására (angol és magyar között). A nyelv szerint megjelenített feliratokat a translations objektum biztosítja.

- **Feliratok**

  A feliratokat, mint például a főcím, a felület közepén helyeztük el.

- **Csúcsok és élek hozzáadása**

  A csúcsok hozzáadása egy input mező és egy gomb segítségével történik, ahol az addNode függvény a felhasználó által megadott csúcsnevet hozzáadja a gráfhoz. Az élek hozzáadása két select mezővel valósul meg, amelyek a forrás és célcsúcsokat választhatóvá teszik, valamint egy input mezővel az él értékének megadására, amelyet az addEdge függvény kezel.

- **Csúcsok és élek eltávolítása**

  A csúcsok eltávolítása egy select mező és egy gomb kombinációjával történik, amely a kiválasztott csúcsot eltávolítja a gráfból az removeNode függvény segítségével. Az élek eltávolítása hasonló módon két select mezővel történik, ahol a forrás és célcsúcsok kiválasztásával az removeEdge függvény végzi el az eltávolítást.

- **Gráf reprezentáció**

  A gráf aktuális reprezentációja egy textarea mezőben jelenik meg, amely csak olvasható. Ez automatikusan frissül a nodes és edges tömbök minden módosításakor.

- **Gráf vizualizáció**

  A gráf vizuális megjelenítéséhez egy div elem szolgál, amely egy külső grafikus könyvtárral (Cytoscape.js) integrált. Ez lehetővé teszi a gráf struktúrájának vizuális megjelenítését.

- **Jelmagyarázat**

  A jelmagyarázat a csúcsok jellemzőit mutatja. Segít a felhasználónak a vizualizáció értelmezésében.

- **Topologikus rendezés**

  A Topologikus rendezés és DAG legrövidebb út algoritmus végrehajtása gomb a gráf

topologikus sorrendjét és a DAGshP algoritmus eredményét jeleníti meg. A gomb megnyomásával a topologicalSort függvény kerül meghívásra, amely a megfelelő rendezést biztosítja. A rendezett csúcsokat egy olvasható textarea mezőben jelenítjük meg.

- **Csúcsok kezdési és befejezési ideje**
  A csúcsok kezdési és befejezési ideje egy lista formájában jelenik meg, ahol minden csúcs neve mellett látható a kezdési és befejezési időpont.
- **DAG legrövidebb út**
  A DAG legrövidebb útjainak megjelenítésére egy külön blokkot használunk. A renderShortestPathTable függvény generál egy táblázatot a legrövidebb utak vizualizálásához, míg a renderShortestPathGraph egy új grafikus ábrát készít a legrövidebb út szemléltetésére.

2. Bemeneti gráf

Ez a gráf szerkeszthető és módosítható. Minden csúcs és él hozzáadása vagy törlése automatikusan frissíti az ábrázolást. Az gráf célja, hogy a legoptimálisabb és legátláthatóbb megjelenítést biztosítsa. A vizualizáció nemcsak a csúcsokat, hanem az irányított éleket és azok súlyait is megjeleníti. Ezen felül látható a kezdő csúcs is, amelyre az algoritmusokat alkalmazzuk.

a Topologikus rendezés és DAG legrövidebb út algoritmus lefutása lehetővé teszik, hogy megjelenjenek a csúcsok kezdési és befejezési idejei.

A gráf kirajzolásához a Cytoscape könyvtárat használjuk, amely tökéletes megoldás arra, hogy az összetett gráfstruktúrát letisztultan, jól követhető módon jelenítsük meg.

3. renderShortestPathGraph függvény

A renderShortestPathGraph függvény felelős az eredmény gráf elkészítéséért. Megkapjuk a dagShortestPath függvényből a távolságokat és megkapjuk a használt élek listáját egy segéd függvényből a getUsedEdges-ből. Ezek az adatok segitségével hozza létre a gráfot. Ez a gráf sok dologban hasonlít a bemeneti gráfra, például: A kezdő csúcs ugyanúgy szürke színnel van jelölve, míg a többi csúcs kék színű az éleken látszódnak az élsúlyok. A különbség viszont, hogy

csak a legrövidebb utak egy forrásból algoritmus által használt éleket jelenítjük meg és a csúcsok tetején nem a kezdő- és befejezési idők jelennek meg, hanem a távolságuk a kezdő csúcstól.

4. css

A css fájlunkban a programunk stílusát, kinézetét szerkesztettem, mint például a főcímet, a gombokat, inputokat.

4.4.3. Algoritmusokat végrehajtó függvények

1. topologicalSort függvény

A topologicalSort függvény a gráf topologikus sorrendjének meghatározására szolgál. Ez az algoritmus a csúcsok olyan sorrendjét határozza meg, amelyben minden él a forráscsúcsból a célcsúcs felé mutat, biztosítva, hogy a gráf minden csúcsát azelőtt látogatjuk meg, hogy bármelyik abból kiinduló él célcsúcsát feldolgoznánk.

A függvényhez különböző adatstruktúrákat használunk: visited (a már látogatott csúcsokat tartja számon), a startFinishTimes (a csúcs bejárásának kezdési és befejezési időpontjait tárolja), a nodeGraph (a gráf éleinek ábrázolására szolgál egy szomszédsági lista formájában), valamint az inStack (a rekurziós verem állapotát követi a körök detektálásához). Amennyiben a gráf üres, a függvény figyelmezteti a felhasználót, és leállítja a további végrehajtást.

A csúcsok és élek alapján először létrejön a gráf szomszédsági listája, amely a gráf szerkezetét ábrázolja. Ezután a függvény mélységi bejárást (DFS) használ az algoritmus fő logikájának megvalósításához. A DFS segédfüggvény minden csúcsra meghívásra kerül, ahol először a csúcsot látogatottként jelöli, hozzáadja azt a rekurziós verembe, majd feljegyzi a kezdési időpontját. Ezután rekurzívan feldolgozza a csúcs szomszédait. Ha egy szomszédot még nem látogatott meg, akkor azt rekurzívan hívja, míg ha egy szomszéd már a veremben van, akkor az algoritmus körként azonosítja az adott él által alkotott kapcsolatot. A feldolgozás végén a csúcs kikerül a rekurziós veremből, a befejezési időpontja rögzítésre kerül, és a csúcs hozzáadódik a topologikus sorrend listájához.+

Miután az algoritmus bejárta az összes csúcsot, ellenőrzi, hogy talált-e kört a gráfban. Ha kört talál, figyelmezteti a felhasználót, mivel a topologikus rendezés nem végezhető el ciklikus

gráfokon. Sikeres végrehajtás esetén a topologikus sorrend eredményét fordított sorrendben tárolja, biztosítva a kívánt sorrend helyességét, és a kezdési és befejezési időket is rögzíti.

2. dagShortestPath függvény

A dagShortestPath függvény egy topologikus rendezésen alapuló algoritmus, amely a Directed Acyclic Graph (DAG) csúcsai között meghatározza a kezdő csúcsból kiinduló legrövidebb útvonalakat. Ez az algoritmus a DAG topologikus sorrendjét kihasználva biztosítja, hogy minden csúcsot csak egyszer dolgozzon fel, és az élhosszak értékeinek felhasználásával frissíti a távolságokat és elődöket.

A függvény elején ellenőrzi, hogy rendelkezésre áll-e a topologikus sorrend. Ha nem áll rendelkezésre, akkor a függvény visszaad egy üres távolság- és elődhalmazt, jelezve, hogy a számítás nem végezhető el. Ha a sorrend rendelkezésre áll, akkor inicializálja a distances és a predecessors objektumokat. Minden csúcsra alapértelmezettként végtelen (Infinity) távolságot állít be, jelezve, hogy a csúcs elérhetetlen, illetve az elődöket egy speciális ⊗ szimbólummal inicializálja. A kezdő csúcs távolságát nullára állítja, mivel a kezdőpont saját magához való távolsága nulla.

Az algoritmus ezután a csúcsokat a topologikus sorrendjükben dolgozza fel. Minden csúcshoz végig megy az éleken, hogy megtalálja az adott csúcshoz tartozó kilépő éleket és azok célcsúcsait. Ha talál egy élt, amelynek forráscsúcsa az aktuálisan feldolgozott csúcs, és a forráscsúcs távolsága nem végtelen, akkor kiszámítja az új lehetséges távolságot a célcsúcs felé. Ha ez az új távolság kisebb, mint a korábban rögzített távolság, akkor frissíti a célcsúcs távolságát és elődjét.

A feldolgozás végére a függvény minden csúcsra kiszámítja a kezdő csúcsból elérhető legrövidebb távolságot, valamint az optimális útvonalon található előző csúcsot. Ezek az adatok egy objektumként térnek vissza, ahol a distances objektum a csúcsok minimális távolságait tartalmazza, míg a predecessors objektum a megelőző csúcsokat azonosítja az optimális útvonalak mentén.

3. renderShortestPathTable függvény

A renderShortestPathTable függvény generálja le a DAG-ban számított legrövidebb utak vizuális megjelenítésére szolgáló táblát. A függvény működése:

A függvény a dagShortestPath eredményeit használja fel, amely tartalmazza az egyes csúcsok minimális távolságát a kezdő csúcstól és az elód csúcsokat.

Először létrehoz egy segéd adatstruktúrát, az úgynevezett edgeMap-et, amely minden csúcshoz hozzárendeli annak kilépő éleit és azok értékeit. Létrehozunk még egy columnMinimums nevű objektumot, amely a táblázat oszlopainak minimális értékeit tartja nyilván.

A táblázat fejlécében minden csúcs nevét megjeleníti vízszintes sorban. Az első sorban megjeleníti a kezdő csúcs távolságát nullaként, míg a többi csúcs esetében végtelen (∞) értéket használ. A további sorok a topologikus sorrend szerint jelenítik meg az egyes csúcsokat, valamint az azokból kiinduló élek értékeit és célcsúcsait. Minden cellában a megjelenített érték az aktuális csúcstól az adott célcsúcshoz vezető legrövidebb távolságot jelenti. A kinézetéről részletes leírás a 2.4.1-es pontban van.

A táblázat utolsó sorában a végeredmény jelenik meg: minden csúcsra a minimális távolság és az elód csúcs. Ez az eredményes útvonalak tömör összefoglalása, amely segíti a felhasználót a DAG belső struktúrájának és a kezdő csúcsból való elérhetőségnek a megértésében.

4. getUsedEdges függvény

A getUsedEdges segédfüggvény célja, hogy azonosítsa a DAG legrövidebb útvonalainak meghatározásában használt éleket. Ehhez a dagShortestPath által visszaadott distances és predecessors adatokat használjuk fel.

A függvény végig megy minden csúcson, és azok elődcímkéi (predecessors) alapján megkeresi azokat az éleket, amelyek az optimális útvonal részei. Ha egy csúcsnak van elődje (azaz nem az alapértelmezett ⊗ értéket tartalmazza), akkor az adott él információit az edges listában keresi meg. Az él azonosítása a forrás- és célcsúcsok ID-jai alapján történik.

A megtalált éleket egy usedEdges nevű tömbben gyűjti össze, majd ezt a tömböt adja vissza, amely az optimális útvonalak éleit tartalmazza. Ez a függvény segít az eredmény gráf megjelenítéséhez szükséges adatokra.

4.5. Tesztelés

Tesztelési terv

Teszt neve	**Teszteset folyamata**	**Eredmény sikeressége**
Csúcs létrehozása egyedi névvel	1. A felhasználó beír egy új csúcs nevet az "Csúcs neve" mezőbe. 2. Rákattint az "Csúcs hozzáadása" gombra. 3. Ellenőrzi, hogy betöltött adatok mellett sincs névütközés új csúcs létrehozásakor. 4. Ellenőrzi, hogy a csúcs megjelenik-e a gráfban. 5. A csúcs mindkét gráf reprezentációjában szerepel.	IGEN
Csúcs létrehozása nem egyedi névvel	1. A felhasználó beír egy új csúcs nevet az "Csúcs neve" mezőbe. 2. Rákattint az "Csúcs hozzáadása" gombra. 3. Névütközés van a gráfban. 4. Előjön egy figyelmeztetés (’Ez a csúcs már létezik’) és nem történik létrehozás.	IGEN
Él létrehozása	1. A felhasználó kiválaszt egy "Forrás" csúcsot a legördülő menüből. 2. Kiválaszt egy "Cél" csúcsot, amelyikkel össze akarja kötni az előzőt. 3. Megad egy élsúlyt, majd rákattint az "Él hozzáadása" gombra. 4. Ha a "Forrás" és "Cél" csúcs megegyezik, nem történik él létrehozás. 5. Ha az él már létezik, nem történik létrehozás. 6. Ellenőrzi, hogy az él létrejött-e és megjelent-e a gráfban és megjelent a gráfábrázolás ablakban.	IGEN

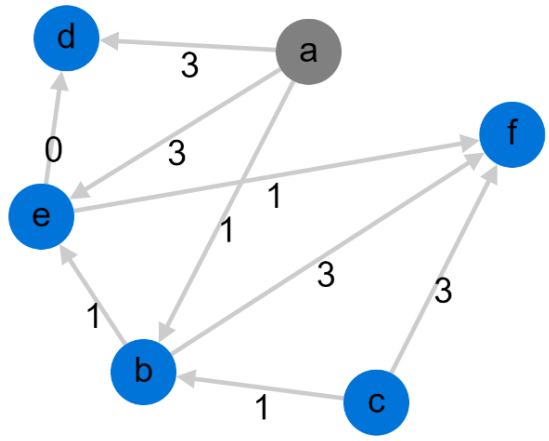
Él létrehozása megegyező forrás és célcsúccsal	1. A felhasználó kiválaszt egy "Forrás" csúcsot a legördülő menüből. 2. Kiválaszt egy "Cél" csúcsot, amelyikkel össze akarja kötni az előzőt. 3. Megad egy élsúlyt, majd rákattint az "Él hozzáadása" gombra. 4. Ha a "Forrás" és "Cél" csúcs megegyezik, előjön egy figyelmeztetés ('A forrás és célcsúcs nem lehet ugyanaz') és nem történik él létrehozás.	IGEN
Duplikált él létrehozása	1. A felhasználó kiválaszt egy "Forrás" csúcsot a legördülő menüből. 2. Kiválaszt egy "Cél" csúcsot, amelyikkel össze akarja kötni az előzőt. 3. Megad egy élsúlyt, majd rákattint az "Él hozzáadása" gombra. 4. Ha az él már létezik, előjön egy figyelmeztetés (’Ez az él már létezik’) és nem történik él létrehozás.	IGEN
Csúcs törlése	1. A felhasználó kiválasztja a törlendő csúcsot a "Eltávolítandó csúcs kiválasztása" legördülő menüből. 2. Rákattint a "Csúcs eltávolitása" gombra. 3. Ha a csúcs nem létezik 4. A csúcs az élkapcsolataival együtt törlődik. 5. A csúcs mindkét gráf reprezentációjából törlődik.	IGEN
Él törlése	1. A felhasználó kiválaszt egy "Forrás" és "Cél" csúcsot az él törléséhez. 2. Rákattint a "Él törlése" gombra. 3. Ellenőrzi, hogy létezik-e a megadott él. 4. Ellenőrzi, hogy az él eltűnt-e a gráf vizualizációból.	IGEN

Él törlése nem létező éllel	1. A felhasználó kiválaszt egy "Forrás" és "Cél" csúcsot az él törléséhez. 2. Rákattint a "Él törlése" gombra. 3. Ellenőrzi, hogy létezik-e a megadott él, ha nem előjön egy figyelmeztetés (’Nem létezik ilyen él’) és nem történik változás.	IGEN
Algoritmus végrehajtása gomb megnyomása üres gráfnál	1. A felhasználó rákattint a "Topologikus rendezés és DAG legrövidebb út algoritmus végrehajtása" gombra. 2. Ellenőrzi, hogy nem üres a gráf, ha üres, előjön egy figyelmeztetés (’A gráf üres. Kérjük, adjon hozzá csúcsokat és éleket a topologikus rendezés előtt’) és nem fut le az algoritmus.	IGEN
Topologikus rendezés	1. A felhasználó rákattint a "Topologikus rendezés és DAG legrövidebb út algoritmus végrehajtása" gombra. 2. Ellenőrzi, hogy nem üres a gráf. 3. Ellenőrzi, hogy a csúcsok sorrendje megjelenik a topologikus sorrendnek megfelelően. 4. Ellenőrzi, hogy a kezdési és befejezési idő megfelelően jelenik meg minden csúcsra.	IGEN
Legrövidebb utak vizualizálása DAG-ban	1. A felhasználó rákattint a "Topologikus rendezés és DAG legrövidebb út algoritmus végrehajtása" gombra. 2. Ellenőrzi, hogy az új gráf vizualizáció megjelenik a helyes legkisebb súlyú utakkal. 3. Az eredménytáblázatban ellenőrzi az él értékeket.	IGEN
Algoritmus tesztelése, ha ciklus	1. A felhasználó elkészíti a kívánt gráfot, van benne kör	IGEN

van a gráfnak az elérhető részében	2. A felhasználó rákattint a "Topologikus rendezés és DAG legrövidebb út algoritmus végrehajtása" gombra. 3. Ellenőrizzük, hogy van-e ciklus a gráfnak az elérhető részében, ha igen akkor előjön egy figyelmeztetés ('A gráfban van kör. Kérjük, távolítsa el a ciklusokat, majd próbálja újra'), az algoritmus nem fut tovább.	
Algoritmus tesztelése, ha ciklus van a gráfnak a nem elérhető részében	1. A felhasználó elkészíti a kívánt gráfot, van benne kör 2. A felhasználó rákattint a "Topologikus rendezés és DAG legrövidebb út algoritmus végrehajtása" gombra. 3. Ellenőrizzük, hogy van-e ciklus a gráfnak az elérhető részében. (Jelen esetben nincs) 4. Az algoritmus tovább folytatódik.	IGEN
Algoritmus tesztelése az A gráfra (17. ábra)	1. A felhasználó elkészíti a kívánt gráfot 2. A felhasználó rákattint a "Topologikus rendezés és DAG legrövidebb út algoritmus végrehajtása" gombra. 3. Ellenőrizzük az eredményt.	IGEN
Algoritmus tesztelése a B gráfra (19. ábra)	1. A felhasználó elkészíti a kívánt gráfot 2. A felhasználó rákattint a "Topologikus rendezés és DAG legrövidebb út algoritmus végrehajtása" gombra. 3. Ellenőrizzük az eredményt.	IGEN

Algoritmus tesztelése a C gráfra (21. ábra)	1. A felhasználó elkészíti a kívánt gráfot 2. A felhasználó rákattint a "Topologikus rendezés és DAG legrövidebb út algoritmus végrehajtása" gombra. 3. Ellenőrizzük az eredményt.	IGEN

A gráf



a → b, 1 ; d, 3 ; e, 3
b → e, 1 ; f, 3
c → b, 1 ; f, 3
e → d, 0 ; f, 1

18. ábra: 'A' bemeneti gráf

-Topologikus rendezés: a, b, e, f, d

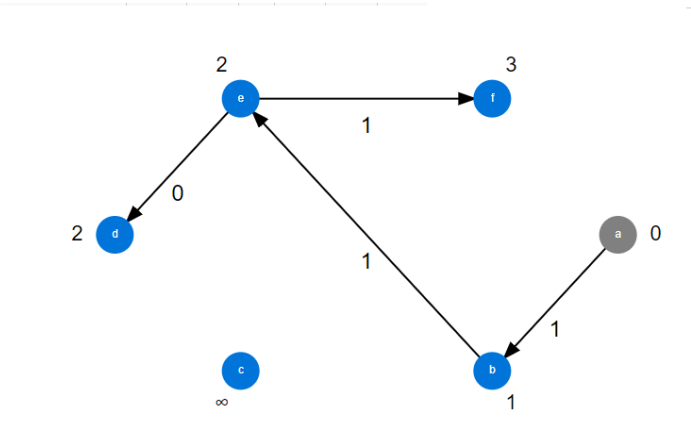
-Csúcs kezdő- és befejezési idők:

a: 1/10 d: 4/5
b: 2/9 f: 6/7
e: 3/8

-DAG legrövidebb út algoritmus

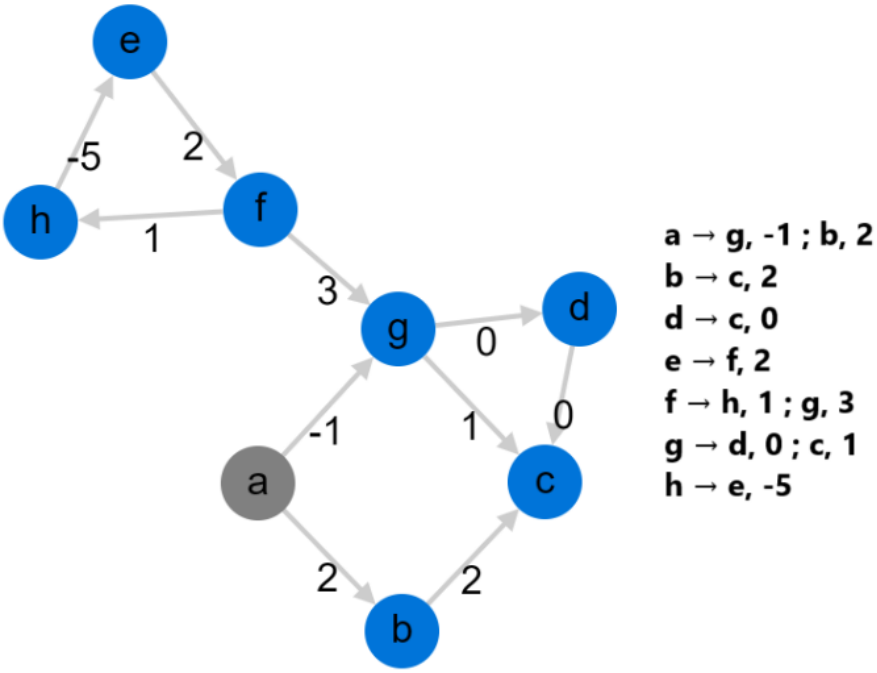
DAG legrövidebb út algoritmus

	a	b	c	d	e	f
	0	∞	∞	∞	∞	∞
a : 0		1 a		3 a	3 a	
b : 1					2 b	4 b
e : 2				2 e		3 e
f : 3						
Eredmény:	0 ⊗	1 a	∞	2 e	2 b	3 e



19. ábra: DAGShP algoritmus eredménye

B gráf



***20**. ábra: 'B' bemeneti gráf*

-Topologikus rendezés: a, g, d, b, c

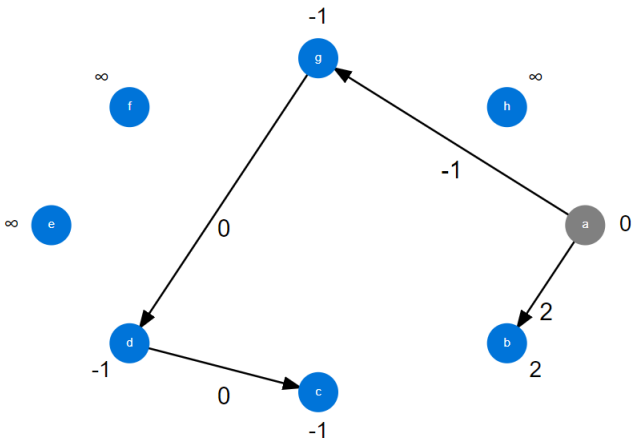
-Csúcs kezdő- és befejezési idők:

a: 1/10 g: 6/9
b: 2/5 d: 7/8
c: 3/4

-DAG legrövidebb út algoritmus

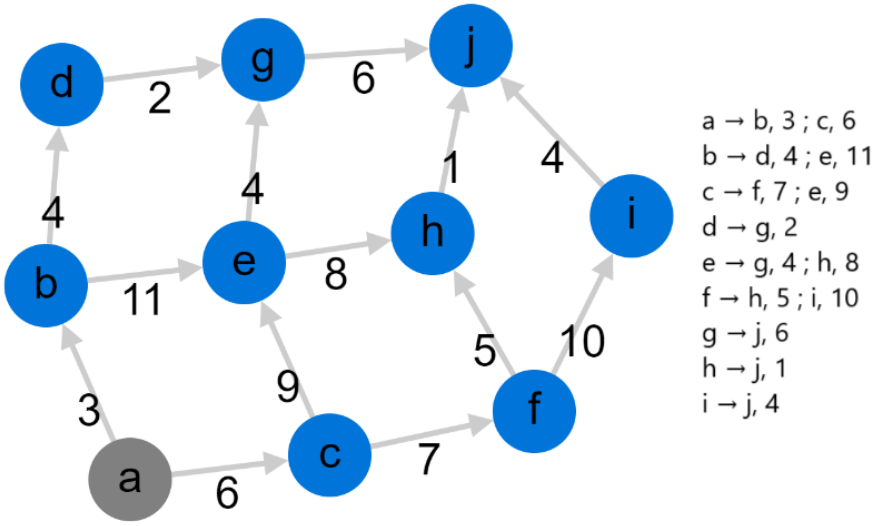
DAG legrövidebb út algoritmus

	a	b	c	d	e	f	g	h
	0	∞	∞	∞	∞	∞	∞	∞
a : 0		2 a					-1 a	
g : -1			0 g	-1 g				
d : -1			-1 d					
b : 2								
Eredmény:	0 ⊗	2 a	-1 d	-1 g	∞	∞	-1 a	∞



21. ábra: DAGShP algoritmus eredménye

C gráf



***22**. ábra: 'C' bemeneti gráf*

-Topologikus rendezés: a, c, f, i, b, e, h, d, g, j

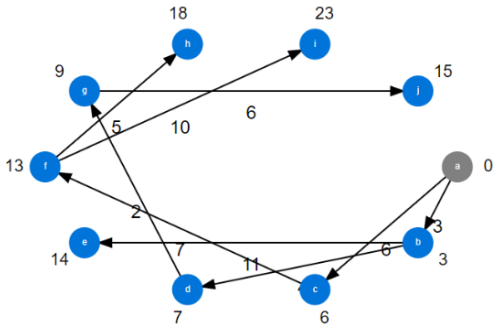
-Csúcs kezdő- és befejezési idők:

a: 1/20 e: 9/12
b: 2/13 h: 10/11
d: 3/8 c: 14/19
g: 4/7 f: 15/18
j: 5/6 i: 16/17

-DAG legrövidebb út algoritmus

DAG legrövidebb út algoritmus

	a	b	c	d	e	f	g	h	i	j
	0	∞	∞	∞	∞	∞	∞	∞	∞	∞
a : 0		3 a	6 a							
c : 6					15 c	13 c				
f : 13								18 f	23 f	
i : 23										27 i
b : 3				7 b	14 b					
e : 14							18 e			
h : 18										19 h
d : 7							9 d			
g : 9										15 g
Eredmény:	0 ⊗	3 a	6 a	7 b	14 b	13 c	9 d	18 f	23 f	15 g



23. ábra: DAGShP algoritmus eredménye

5. Összefoglalás és további fejlesztési lehetőségek

5.1. Összefoglalás

A program segíthet olyan hallgatóknak, akik nem értették még meg teljesen az algoritmus működését, vagy olyanoknak, akik csak tesztelni szeretnék az eredményüket más gráfadatokkal is. Oktatóknak is hasznos eszköz lehet a szoftver, jól tudja helyettesíteni a kézzel írást, ha inkább gépen szeretne dolgozni.

Az alkalmazás elkészítése során fontosnak tartottam, hogy kezelése bárki számára nagyon könnyen elsajátítható legyen, könnyen meg lehessen érteni benne mindent. Angol és magyar nyelvben is elérhető, segítve ezzel a külföldi hallgatókat is.

Összességében a program megoldást nyújt a problémára, amely a DAG Legrövidebb utak egy forrásból algoritmus megértésében segít.

5.2. Továbbfejlesztési lehetőségek

- Lehessen a gráfban kattintással létrehozni, törölni csúcsokat és éleket
- Felhasználói felület fejlesztése, kinézet szépítésé
- Világos és sötét téma hozzáadása a felülethez
- Algoritmus animálása, automatikus lejátszása
- Csúcs és él megjelenésének szerkeszthetősége
- Gráf képbe való lementése
- Lépések visszavonása szerkesztésnél
- Ne csak a DAGShP Algoritmus lehessen megcsinálni, hanem más gráf algoritmusokat is

6. Irodalomjegyzék

[1] Dr. Ásványi Tibor: Algoritmusok és adatszerkezetek II. előadásjegyzet: Élsúlyozott gráfok és algoritmusaik

[2] A diagramok PlantUML kódban készültek: https://plantuml.com/