EÖTVÖS LORÁND TUDOMÁNYEGYETEM

Informatikai Kar

Algoritmusok és Alkalmazásaik Tanszék

Gráfalgoritmusok grafikus szimulációja

Témavezető:

Nagy Sára

mesteroktató

Szerző:

Ladányi Szandra

Programtervező informatikus BSc.

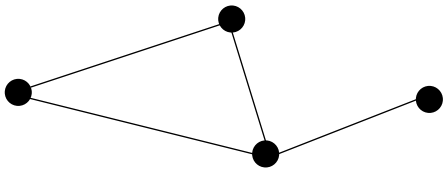
Budapest, 2021

Tartalomjegyzék

1 Bevezetés

1.1 Gráfokról

A gráfokat a legkönnyebben a következőképpen lehet meghatározni: pontok – azaz *csúcsok* – vonalakkal – azaz *élekkel* – összekötve. Így egy gráf ábrázolása elég egyszerűnek hangzik; nem kell mást tenni, mint az előző leírás alapján rajzolni, hogy az 1. ábrán is látható alakzathoz hasonló eredményt kapjunk.



1. ábra: Egy gráf

Kiegészítendő ez a leírás néhány definícióval, hogy a továbbiakban a gráfokat mint összetett adatszerkezeteket lehessen vizsgálni.

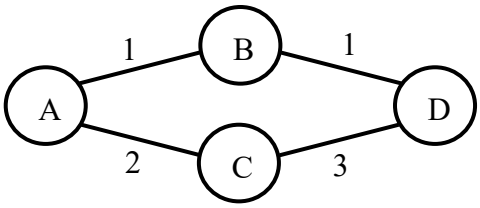
A gráfban *szomszédosnak* nevezünk két csúcsot, ha összeköti őket él, az adott él végén elhelyezkedő csúcsokat az él *végpontjainak* hívjuk.

Útnak nevezzük egy gráfban az élek és csúcsok olyan felsorolását, melyet követve egy csúcsból egy másikba jutunk. Minden csúcs csak egyszer szerepelhet, a csúcsok és az élek egymást váltják, és egy csúcs után csak olyan él jöhet, aminek az az egyik végpontja, majd az él után annak másik végpontja szerepel. Két csúcs között akkor van út, ha létezik ilyen felsorolás úgy, hogy az egyik csúcs az út kezdőpontja, a másik pedig a végpontja. Azt az utat, aminek a kezdő- és végpontja is azonos, *körnek* nevezzük. Egy gráf *összefüggő*, ha bármely két csúcsa között létezik út. A gráfot *fának* nevezzük, ha bármely élét elhagyva már nem lenne összefüggő.

Párhuzamos élek azok az élek, melyeknek páronként megegyeznek a kezdő- és végpontjaik. *Hurokélek* pedig azok az élek, melyeknek mindkét végpontja ugyanaz a csúcs. A dolgozatban tárgyalt program csak *egyszerű gráfokkal* foglalkozik – azaz olyanokkal, amelyekben sem párhuzamos, sem hurokélek nem fordulnak elő.

A csúcsoknak különböző címkéket adhatunk annak érdekében, hogy megkülönböztessük őket. Így a gráf éleit is az határozza meg, hogy melyik két csúcs, azaz melyik két címke között helyezkednek el. A csúcsok elnevezése bármilyen módon történhet, általános példákban számokkal vagy betűkkel szokás jelölni azokat.

Az élek is kaphatnak címkéket, ezeket *élsúlyoknak* nevezzük. Ekkor két pont közötti útnak lehet *költsége*, mely a felhasznált élek súlyainak összege. A 2. ábrán látható, hogy az A és D címkével jelölt csúcsok között két út található. Az egyikben a csúcsok sorrendje A, B, D, míg a másikban A, C, D. Ha például ezek a csúcsok városokat, az élek pedig az őket összekötő utakat reprezentálják, valamint az élek súlyai a fizetendő útdíjat jelentik, akkor már egyáltalán nem mindegy, hogy az első, 2 költségű, vagy a második, 5 költségű utat választjuk.



2. ábra: Élsúlyozott gráf

Egy gráf élei lehetnek *irányítottak*, ekkor egy útban a felsorolás csak az él irányítottságának megfelelő sorrendben történhet. Az irányított élek végén nyíl jelöli ezt a felsorolási irányt.

Ha a kezdeti példát (1. ábra) ezekkel az extra információkkal kiegészítjük, akkor a gráfok nagyon sok feladat modellezésére alkalmassá válnak. A mindennapok részét képezik ezek a modellek. A Google keresőmotorja is egy gráfon dolgozik, melyben a való világ kapcsolatai vannak leírva. A tárgyak, helyszínek, személyek (illetve tulajdonképpen bármi, amit csak el lehet képzelni) a gráf csúcsai, a kapcsolatokat a csúcsok közötti élek jelentik. Így például egy személyhez nem csak a saját tulajdonságai, hanem akár helyszínek, tárgyak is köthetők. Tegyük fel, hogy valaki Leonardo da Vincire keres a Google keresőben. Ebben a „tudás-gráfban” a da Vincit jelentő csúcs össze van kapcsolva a művész alkotásaival, de a vele egy korban élt más híres művészekkel is. Ezen kapcsolatok alapján az oldal ajánlásokat tesz, illetve tágabb értelmezéssel javaslatokat mutat a felhasználónak keresése tárgyáról [1]. Egy feladat gráfokkal történő modellezését követően, a bennük rejlő, a problémát megoldó információhoz hozzá kell valahogyan jutni. Ebben segítenek a gráfalgoritmusok.

1.2 Gráfalgoritmusok

Ahhoz, hogy a gráfokkal modellezett feladatokra megoldást találjunk, valahogy fel kell térképezni a vizsgált gráfot, és értelmezni a kapott adatokat. Többféle algoritmus is létezik gráfok bejárására – az általuk adott eredmények mást és mást mutatnak, más

információt szolgáltatnak az adatokról. Az algoritmus kiválasztása nagy részben függ a kívánt eredménytől. Az alábbiakban a programban is használt négy algoritmus rövid leírása és néhány alkalmazásuk olvasható.

1.2.1 Szélességi gráfkeresés

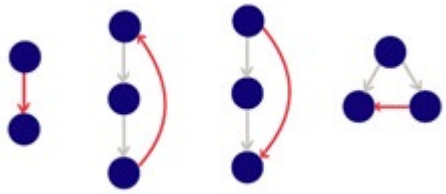
A szélességi gráfkeresést élsúlyozatlan gráfokon értelmezzük, melyek irányítottak vagy irányítatlanok is lehetnek. Az algoritmus egy tetszőleges kezdőcsúcsból az összes többi csúcsba tartó, legkevesebb élt tartalmazó – tehát legrövidebb – utat határozza meg. Az algoritmus a gráfot a kezdőcsúcstól szintenként járja be, azaz egy csúcsnak az összes gyerekéhez vezető utat megvizsgálja, majd ha tud, új csúcsra lép, és megismétli ezt [2].

Használata azokban az esetekben ajánlott, amikor az egyes kapcsolatok között nincs különbség (nincs szükség élsúlyozásra), és a legrövidebb útvonalat kell meghatározni. Épp ezért használják például weblapok indexelésére vagy közösségi hálózatokon való keresésre, ahol az emberek és csoportok kapcsolatait modellezi a gráf [3].

1.2.2 Mélységi gráfkeresés

A mélységi gráfkeresés eredménye a mélységi feszítő erdő, mely egyszerű, irányított, élsúlyozatlan gráfokon számítható ki. A csúcsok feldolgozása során az algoritmus a gyerekeken keresztül a legmélyebb szintig járja be a gráfot, mielőtt az azonos szinten lévő csúcsokat dolgozná fel. A bejárás során a feldolgozott élek a következő típusúak lehetnek:

- fa-él: mélységi fa éle (ezek mentén járjuk be a gráfot)
- vissza-él: a végpontja őse a kezdőpontjának a mélységi feszítő erdőben
- előre-él: a kezdőpontja őse a végpontjának a mélységi feszítő erdőben
- kereszt-él: a mélységi fában két különböző ágon lévő csúcs között található



3. ábra: Élosztályok példái az ábrán látható sorrendben: fa-él, vissza-él, előre-él, kereszt-él

(A szürke élek az utóbbi három esetben a fa-éleket jelentik)

Az élek osztályozása által a mélységi gráfkeresés használható annak megállapítására, hogy vannak-e körök a gráfban. Ha az algoritmus a futása során vissza-élt talál, akkor a

fa-élek és a vissza-él irányított kört[1] alkotnak. A mélységi feszítő erdőből könnyen megkapható a gráf topologikus rendezése[2] [2]. Ezt az algoritmust gyakran használják olyan feladványok megoldására, melynek csak egy helyes eredménye létezik (például útvesztőkben a kijárat megkeresése) [3].

1.2.3 Prim algoritmusa

Prim algoritmusa élsúlyozott, irányítatlan, összefüggő gráfokon értelmezett. Az algoritmus célja minimális feszítőfát[3] találni a gráfban. A minimális feszítőfában minden csúcs a lehető legkisebb élköltséggel csatlakozik a fához. Tetszőleges kezdőcsúcsból indul az algoritmus, ami önmagában – mivel csak egycsúcsú gráf – része az eredeti gráf egyik minimális feszítőfájának. Az eljárás lényege, hogy minden lépésben úgy kell az aktuális fához új élet és csúcsot választani, hogy az így kapott gráf továbbra is az egyik minimális feszítőfa része maradjon [4].

Számítógépes hálózatok esetén az üzenetszórás (broadcasting) során „faépítésre" használják Prim algoritmusát. De alkalmazzák képek szegmentálása és klaszterelemzés során is [3].

1.2.4 Dijkstra algoritmusa

Legrövidebb, vagyis legkisebb élköltségű utak keresésére alkalmas Dijkstra algoritmusa. Élsúlyozott gráfokon alkalmazható, melyek irányítottak és irányítatlanok is lehetnek, azonban fontos, hogy az összes él súlya nemnegatív legyen. A keresés módja nagyon hasonló a szélességi gráfkereséshez; ez az algoritmus is szintenként járja be a gráfot, azonban nem az élek számát, hanem az út összköltségét veszi figyelembe annak eldöntésénél, hogy az adott csúcshoz jobb utat talált-e az eddiginél. Ha lehetne negatív súlyú él a gráfban, akkor lehetne benne negatív kör is. Egy negatív él ismételt hozzáadásával egyre kisebb költségű utat találna az algoritmus, és a keresésnek sosem lenne vége [4].

A leggyakoribb felhasználása Dijkstra algoritmusának a tényleges útkeresés, azaz térképeken két pont között a legrövidebb út megtalálása. Az élköltségek kiszámítása több tényező alapján is történhet ezeknél a feladatoknál, ugyanis nem csak a tényleges távolságot kell figyelembe venni, hanem akár az aktuális forgalmat és útdíjakat is [3].

---

[1] Olyan kör, aminek az élei irányítottak
[2] A csúcsok olyan felsorolása, melyben egy csúcsot az összes őse megelőz
[3] Feszítőfa: a gráf összes csúcsát és az éleinek egy részhalmazát tartalmazó fa, minimális feszítőfa: a gráfban a legkisebb összélköltségű feszítőfafa.

2 Felhasználói dokumentáció

2.1 Az alkalmazás célja, felhasználása

Az alkalmazás két fő feladatot lát el. Egyrészt gráfokat lehet vele szerkeszteni – melyek hagyományos grafikus ábrázolással jelennek meg a felületen –, másrészt a megjelenített gráfon előre meghatározott gráfalgoritmusokat lehet futtatni.

Az alkalmazás négy algoritmus (szélességi bejárás, mélységi bejárás, Dijkstra-algoritmus és Prim-algoritmus) szimulációját teszi lehetővé, melyeket lépésenként, oktató jelleggel mutat be. Emiatt csak kisebb (maximum 15 csúcsból álló) gráfok létrehozása lehetséges, hogy a változások követhetőek legyenek. Az algoritmusok szemléltetése három részre osztható:

1. A megjelenített gráfon különböző színek segítségével jelöli a program, hogy az algoritmus hol jár, mely csúcsokat, éleket vizsgálja épp, és mi a vizsgálat eredménye. A színek jelentése a futtatás során megjelenítésre kerül a felületen.
2. A program kiírja az algoritmus által használt segédstruktúrák aktuális tartalmát. Így a számítások pillanatnyi állapotai jól követhetőek.
3. A program kirajzolja a választott algoritmus struktogramját, jelölve, hogy annak melyik része van éppen hatással a gráfra.

2.2 Használat előtti tudnivalók

2.2.1 Rendszerkövetelmény

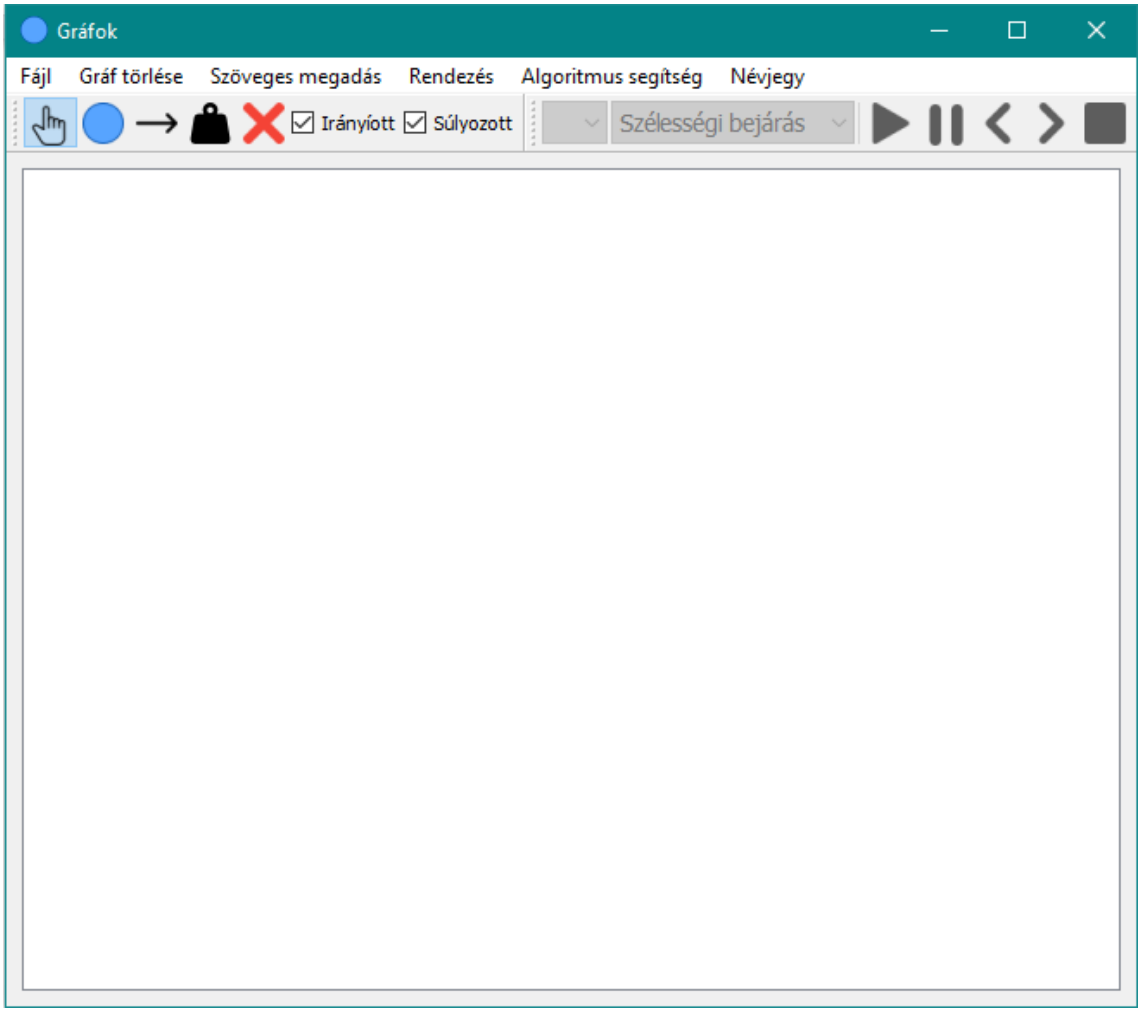
Az alkalmazás Windows operációs rendszerű számítógépeken futtatható. A futtatásnak különleges hardveres előkövetelményei nincsenek.

2.2.2 Telepítés, üzembe helyezés

Az alkalmazás tömörített állományként érhető el, mely tetszőleges célhelyre csomagolható ki a célszámítógépen. Ezt követően a kicsomagolt mappában az *UXQURT_Szakdolgozat.exe* elindításával használható a program. Egyéb telepítést nem igényel, kicsomagolást követően azonnal futtatható.

2.3 Felhasználói felület áttekintése

Az alkalmazás megnyitását követően azonnal a szerkesztőfelületet látja a felhasználó. Az ablak közepén elhelyezkedő fehér felület lesz a gráfok megjelenésének helye, a felső menüsáv és eszköztárak pedig az alkalmazás kezelését teszik lehetővé.



4. ábra: Felhasználói felület

A menüben található a fájlkezelés, általános szerkesztési lehetőségek, illetve segítség a futtatható algoritmusok megértéséhez.

Három menüpontnak *(Gráf törlése, Szöveges megadás, Rendezés)* a gráf szerkesztésében van szerepük. Ezekről részletesebben a Szerkesztés alfejezetben lesz szó.

Az *Algoritmus segítség* menüpont alatt megtekinthető a programban futtatható gráfalgoritmusok rövid leírása és struktogramja. A futtatás során ugyanezek a struktogramok a felületen, a gráf mellett is megjelenítésre kerülnek. A segítségnél minden esetben feltüntetésre kerül, hogy az adott algoritmus milyen gráfon futtatható, így ezeket

a beállításokat kell megtenni, hogy a szimulációt el lehessen indítani. Egyéb esetben figyelmeztető üzenet jelzi, hogy mely paraméter nem megfelelő.

2.4 Fájlkezelés

A *Fájl* menüpont alatt (5. ábra) lehet a szerkesztőfelület tartalmát – tehát az aktuálisan látható gráfot – elmenteni, illetve egy korábban elmentett gráfot a programba betölteni. A mentés saját fájlformátumba *(.graph)* történik, az adatok szöveges állományként kerülnek tárolásra. Amennyiben a mentés vagy a megnyitás művelete sikertelen, felugró ablak figyelmeztet erre. A mentés és a megnyitás a *CTRL+S*, illetve a *CTRL+O* billentyűkombinációkkal is elérhető.

5. ábra: Mentés és megnyitás

Mentés és megnyitás esetén is az operációs rendszerre jellemző fájlkezelő ablak nyílik meg, melyben csak a program által használt *graph* kiterjesztésű fájlok kerülnek megjelenítésre. A program futása során, amennyiben több gráf mentése vagy fájlból való megnyitása történik, akkor a fájlkezelő a korábbi elérési útvonal betöltésével nyílik meg.

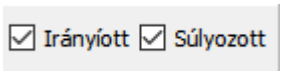
Amennyiben a felhasználó helytelen fájlformátumot próbál a mentés vagy a megnyitás esetén alkalmazni, a művelet végrehajtása sikertelen lesz. Ezt az alkalmazás egy felugró figyelmeztető ablakban jelzi. Hiba léphet fel akkor is, ha egy megfelelő kiterjesztésű fájl tartalma olyan módon módosult, hogy többé nem ábrázol helyesen egy gráfot az alkalmazás számára – például, ha valaki módosítja a szöveges fájl tartalmát, és annak szerkezete ezt követően eltér a várt állapottól. Ekkor szintén meghiúsul a megnyitás művelete, melyre felugró üzenet figyelmeztet.

A programhoz készültek példa állományok, ezek a *Mintagráfok* mappában találhatóak, és az alkalmazás használata során bármikor betölthetőek, szerkeszthetőek. A fájlok elnevezései az általuk leírt gráfok irányítottságára és súlyozottságára utalnak, annak érdekében, hogy egyszerűbb legyen az egyes algoritmusok futtatásához megfelelő példát találni.

Egy gráf betöltése esetén a szerkesztőfelület tartalma lecserélődik, így a korábban szerkesztett gráf – amennyiben korábban nem került mentésre – elveszik.

2.5 Szerkesztés

A programban egyszerű gráfokat lehet szerkeszteni, melyek lehetnek irányítottak vagy irányítatlanok, illetve élsúlyozottak vagy élsúlyozatlanok. A gráf irányítottságát és élsúlyozottságát a bal oldali eszköztáron lehet beállítani egy-egy jelölőnégyzet segítségével (6. ábra). Az irányítottság módosítása a *CTRL+A*, az élsúlyozottság módosítása a CTRL+*Q* billentyűkombinációkkal is megtehető.



6. ábra: Gráf irányítottságának és élsúlyozottságának beállítása

A *Gráf törlése* menüpontra kattintva a korábban hozzáadott csúcsok és élek eltűnnek a felületről, teljesen új (üres) gráf hozható létre.

A létrehozott csúcsok címkézése automatikusan történik az angol ábécé betűivel (A-tól kezdve).

A menüben található *Rendezés* funkció a csúcsokat a szerkesztőfelületen egymástól egyenlő távolságban egy körvonalra illeszti. Ekkor a hozzájuk tartozó élek is felveszik az átrendezésnek megfelelő új pozíciójukat.

2.5.1 Grafikus szerkesztés

Grafikus szerkesztés esetén az 7. ábrán látható eszköztáron található gombokkal kell kijelölni a szerkesztés módját. A megfelelő eszköz kiválasztását követően a szerkesztőfelületen az egér bal gombjának kattintásával van lehetőség egy gráf szerkesztésére.

7. ábra: Szerkesztő eszköztár

A lehetséges szerkesztési módok a következők (a 7. ábrán látható sorrendjük szerint):

1. Átrendezés

   A gráf csúcsai tetszőlegesen átrendezhetőek a szerkesztőfelületen. Az élek követik a csúcsok mozgását, ezért azok mozgatására külön nincs lehetőség.

2. Új csúcsok felvétele

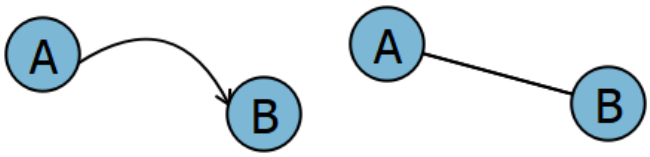
   A kattintás helye a létrejövő csúcs középpontját jelöli ki. Amennyiben már maximális számú csúcs lehelyezésre került, több csúcs felvételére nincs lehetőség – az ablak felső részében pirossal írt üzenet figyelmeztet erre, mely pár másodperc elteltével eltűnik.

3. Élek felvétele

   Élt két csúcs között lehet felvenni, a tervezett végpontjaira kattintva. Ha ez a mód van kiválasztva, akkor egy tetszőleges csúcsra való kattintáskor a csúcs piros körvonallal kijelölésre kerül a 8. ábrán látható módon. Ez a csúcs lesz az él egyik végpontja. Ugyanerre a csúcsra való ismételt kattintással megszüntethető a kijelölés. Ez esetben megint egy tetszőleges csúcs jelölhető ki a létrehozandó él kezdőpontjaként. Amennyiben egy csúcs kijelölésre került, egy attól különböző csúcsra kattintva létrejön az él a két csúcs között.

8. ábra: Megjelölt kezdőpont

Irányítatlan gráf esetén a kiválasztás sorrendje tetszőleges, azonban irányított esetben először a kezdőpontot, majd a végpontot kell megadni. Az irányítottság az egész gráfra vonatkozik, tehát egy gráfnak csak irányított, vagy csak irányítatlan élei lehetnek.



9. ábra: Irányított és irányítatlan él

4. Súlyok módosítása

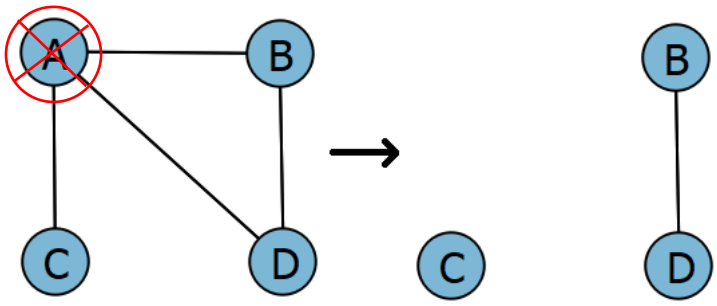
   Ha a gráf élsúlyozott, akkor minden élhez tartozik egy élsúly, mely egy egész szám (akár negatív is lehet). Az él létrehozásakor ez az érték automatikusan 1 lesz. Ezt az értéket lehet módosítani az élre vagy az élen látható élsúlyra való kattintással. A kijelölt súly piros körvonalat kap, a szerkesztőfelület bal felső sarkában pedig

megjelenik egy szövegdoboz. Ebbe a szövegdobozba kell beírni az új értéket, ami az enter billentyű megnyomását követően kerül beállításra.

Ezt a szerkesztési módot csak élsúlyozott gráf esetén lehet kiválasztani, egyéb esetben a gomb le van tiltva (nem lehet rákattintani és szürke a háttere).

5. Csúcsok és élek törlése

A gráfból csúcsokat és éleket is lehet törölni azokra való kattintással. Ha egy törölni kívánt csúcshoz több él is csatlakozik, akkor – ahogy a 10. ábrán látható – azok is automatikusan törlődnek. Élek esetén csupán a kiválasztott él tűnik el.



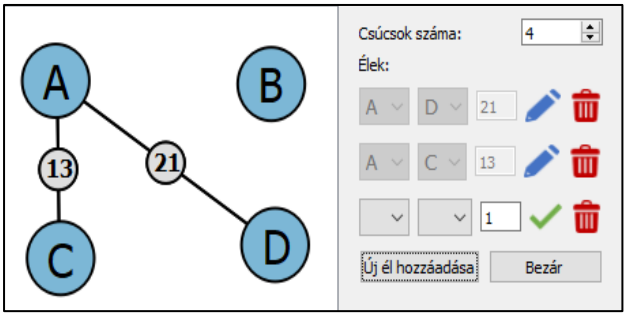
10. ábra: Csúcs törlése

Az összes csúcs és él törlése a *Gráf törlése* gombot kiválasztva is lehetséges.

Egy vagy több csúcs törlése után, új csúcsok felvétele esetén az új csúcs neve mindig az angol ábécé szerinti első olyan betű lesz, ami éppen nincs használatban. Akár azért, mert a gráf mérete eddig nem indokolta az elnevezést, akár azért, mert törölve lett az azonos nevű csúcs. Például a 10. ábrán látható törlést követően a legújabb csúcs *A* címkéjű lesz, majd az azt követő az *E* elnevezést kapja.

2.5.2 „Szöveges” szerkesztés

A gráfok űrlapszerűen is megadhatók. A *Szöveges megadás* menüpont jeleníti meg az ablak jobb oldalán az ehhez szükséges vezérlőket. Ekkor a grafikus felületen használható módok közül csak az átrendezés működik, ez automatikusan bekapcsol.



11. ábra: Gráf űrlapszerű megadása

Megadható a csúcsok száma (0 és 15 közötti szám), illetve új élek vehetők fel az *Új él hozzáadása* gombbal, amennyiben a szerkesztés alatt álló gráf még nem teljes gráf (tehát még vannak behúzható élei).

A szerkesztés hatása a kirajzolt gráfon is látható. A csúcsok számának állítását az enter gomb megnyomásával kell véglegesíteni, akár a nyilakkal, akár számmal való beírás esetén. Ha a csúcsok számát az aktuális értéknél kisebbre állítjuk, akkor a felületről törlődik megfelelő számú csúcs[4] és minden hozzájuk tartozó él. Ha a beállított szám nagyobb, akkor új csúcsok jönnek létre. Ekkor az eddigi és az új csúcsok egy körbe rendeződnek (a korábban felvett élek megmaradnak).

Új él hozzáadása esetén egy kitöltendő üres sor jelenik meg az élek listájában, ahol az első legördülő menüből kiválasztható a kezdőpont, ezt követően pedig a másodikból a végpont. Amennyiben a gráf élsúlyozott, meg kell adni az él súlyát is, mely egy tetszőleges egész szám lehet. A pipára kattintva az új él bekerül a többi közé. Szerkeszteni csak a súlyozott éleket lehet, ebben az esetben is csak az élsúlyokat. Miután létrejött az él, a végpontok módosítására már nincs lehetőség. Ha az adott két csúcs közötti élre nincs szükség a gráfban, akkor a törlés ikonra kattintva lehet eltávolítani azt.

A kezdőcsúcsok listájában csak olyan csúcsok szerepelnek, amelyekből még nem indul él az összes többi csúcsba. A kezdőcsúcs kiválasztása után lehetséges végpontokként pedig már csak azok a csúcsok jelennek meg a második legördülő menüben, melyekhez valóban vezethet még él. Irányított gráfban, ha A és B a gráf két csúcsa, akkor A-ból B-be és B-ből A-ba is hozható létre él, azonban irányítatlan gráf esetén csak az egyik lehetőség vehető fel az élek listájába.

A „szöveges szerkesztő” megnyitásakor a szerkesztő- és megjelenítőfelületen látható gráfnak megfelelő értékek töltenek be a vezérlőkbe.

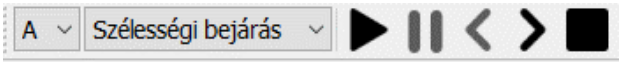
2.6 Gráfalgoritmusok

A megszerkesztett gráfokon négyféle gráfalgoritmus szimulációja tekinthető meg: szélességi bejárás, mélységi bejárás, Prim-algoritmus és Dijkstra-algoritmus. Ezek az algoritmusok más és más típusú gráfon alkalmazhatóak. A követelmények megtalálhatóak az *Algoritmus segítség* menüpont alatt.

[4] A törlés a hozzáadás sorrendjében visszafelé történik.

2.6.1 Kezelés

A szimulálni kívánt algoritmus meghatározása, a kezdőcsúcs kiválasztása (amennyiben szükséges[5]), valamint a szimuláció kezelése a második eszköztáron (12. ábra) található vezérlőkkel lehetséges.



12. ábra: Eszköztár – algoritmusok kezelése

Ha nincs a programban gráf, tehát a szerkesztőfelület üres, akkor nem választható gráfalgoritmus. Ekkor az eszköztár összes eleme le van tiltva. Ha már legalább egycsúcsú a gráf, akkor lehetségessé válik a kiválasztás. A lehetséges kezdőcsúcsok listája csúcsok törlésekor, illetve új csúcsok felvételekor automatikusan frissül.

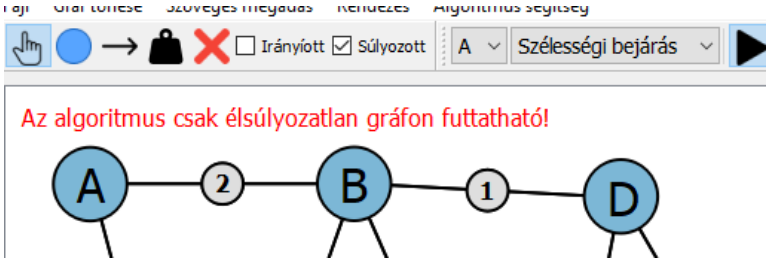
Az előző képen látható gombok működése a hozzájuk társítható hagyományos funkciókkal megegyeznek, mindig csak az a gomb engedélyezett (csak az kattintható), amelyik aktuálisan hatással lehet a működésre. A lejátszó gomb (első gomb) folyamatos, másodpercenkénti léptetéssel elindítja az algoritmus futását, amit a szünet gomb (második gomb) leállíthat úgy, hogy az aktuális állapot megmarad. Ezt követően a lejátszó gomb ismételt megnyomásával ugyanonnan folytatható a működés. A vissza és az előre nyilak (harmadik és negyedik gomb) segítségével egyesével lehet az algoritmus lépéseit megtekinteni. Erre a léptetésre lehetőség van akár a folyamatos lejátszás közben is (ekkor a további lépések szintén „előre haladva”, másodpercenként történnek meg). A stop (ötödik gomb) hatására az algoritmus szimulációja teljesen leáll, az alkalmazás a szerkesztési állapotába tér vissza.

A 12. ábrán látható állapot egy csúcs felvételét követően áll elő. Látható, hogy a szünet és a vissza gomb szürke, hiszen még nem indult el az algoritmus futtatása a gráfon, ekkor még sem szüneteltetni nem lehet azt, sem visszalépni nem lehet benne.

Amennyiben a gráf nem felel meg a kiválasztott algoritmus előfeltételeinek a lejátszó vagy előre gomb lenyomásakor, a szimuláció nem indul el. Ellenben pár másodpercre figyelmeztető szöveg látható, mely leírja, hogy milyen tulajdonságon kell változtatni ahhoz, hogy a választott algoritmus ezen a gráfon futtatható legyen. A 13. ábra

[5] A mélységi bejárásnak nincs szüksége kezdőcsúcsra

szemlélteti, hogy mi történik, ha a súlyozatlan gráfokon értelmezett szélességi bejárást súlyozott gráfon próbáljuk indítani.

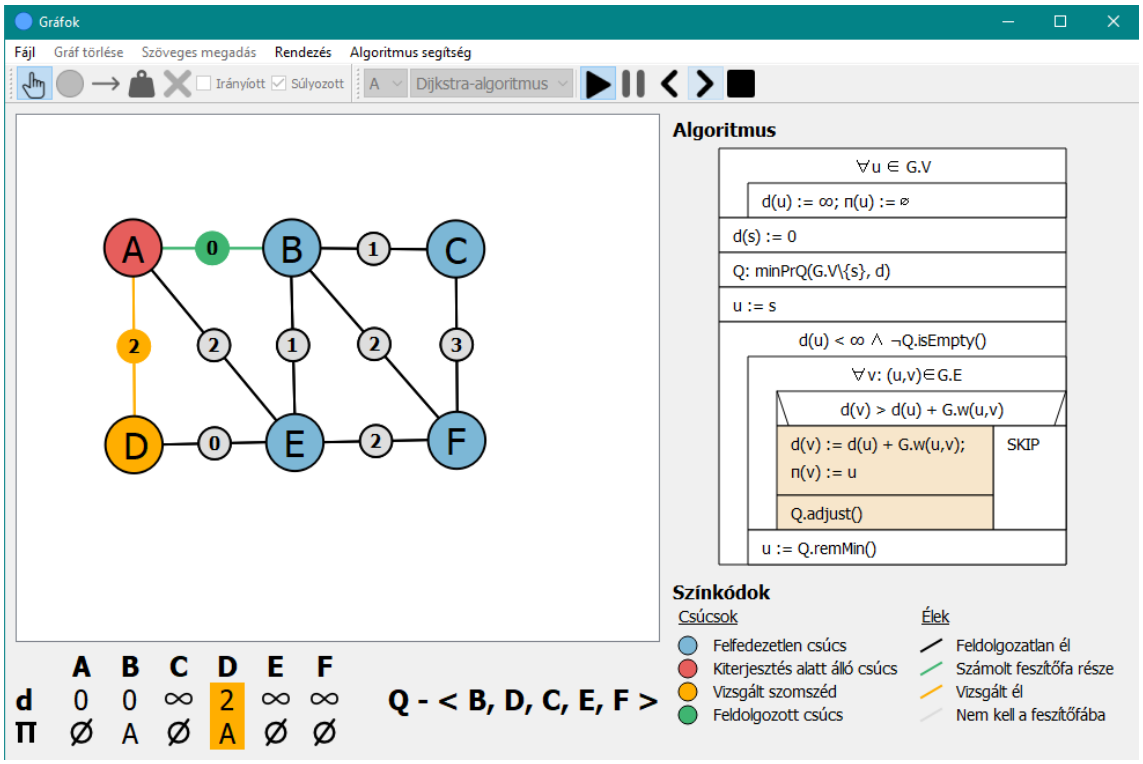


13. ábra: Algoritmus futtatás, ha az előfeltétel nem teljesül

A gráfalgoritmus futása során a gráf szerkesztésére a csúcsok átrendezésén kívül nincs lehetőség.

2.6.2 Felületi megjelenítés

Az egyes gráfalgoritmusok futtatása során a számítás lépéseit háromféleképpen szemlélteti a program a felületen – ezek a 14. ábrán láthatóak.



14. ábra: Dijkstra-algoritmus futtatása

1. Grafikusan, a gráfon

   A szerkesztőfelületen a gráfon különböző színekkel látszódnak az egyes lépések, a számított eredmények. A futás során mindig látható ehhez egy jelmagyarázat az ablak

jobb alsó sarkában. Az ábrán is látható színkódok az összes algoritmusra érvényesek, a szélességi és a mélységi bejárások még kiegészülnek a csúcsok szürke színezésével. A szürke szín az ezen gráfbejárások esetében hagyományosan is szürkének nevezett csúcsokat – azaz azokat, melyeket már megtalált az algoritmus, de még nem dolgozott fel – jelölik.

2. Algoritmus

   Az ablak jobb oldalán látható a választott algoritmus struktogramja, mely a [2] és a [4] alapján készült. A struktogramban eltérő háttérszínnel jelzi a program, hogy az algoritmus aktuális lépése mely utasításblokkok alapján történik. Az összes látható változás a kijelölt részlet futásának eredménye.

3. Számított értékek követése

   A programban szimulálható algoritmusok különböző segédstruktúrák segítségével követik a változásokat a futásuk során, melyek az egyes algoritmusoknál a következőek:

   Szélességi bejárás:

   - d tömb, mely tárolja, hogy az adott csúcsba hány élen keresztül jutunk el
   - π tömb, mely tárolja a csúcsok szülőit (melyik csúcson keresztül érjük el a legrövidebben)
   - Q sor, melybe a következőként feldolgozandó csúcsok kerülnek

   Mélységi bejárás:

   - d tömb, mely a csúcsok elérési idejét (discovery time) tárolja
   - f tömb, mely a csúcsok befejezési idejét (finishing time) tárolja
   - π tömb, mely a csúcsok szülőit tárolja (melyik csúcson keresztül érjük el a legrövidebben)

   Prim-algoritmus:

   - d tömb, mely annak az élnek a súlyát jelenti, amely pillanatnyilag a csúcsot a minimális feszítőfához kapcsolja
   - π tömb, mely tárolja a csúcsok szülőit (melyik csúcson keresztül érjük el a legrövidebben)
   - Q minimum prioritásos sor, melyben a továbbiakban feldolgozandó csúcsok szerepelnek, a rendezés a megfelelő d-beli értékek alapján történik

Dijkstra-algoritmus:

- d tömb, mely tárolja, hogy az adott csúcsba milyen minimális költséggel jutunk el
- π tömb, mely tárolja a csúcsok szülőit (melyik csúcson keresztül érjük el a legrövidebben)
- Q minimum prioritásos sor, melyben a továbbiakban feldolgozandó csúcsok szerepelnek, a rendezés a megfelelő d-beli értékek alapján történik

A Q sornak mindig az első elemét veszi ki az algoritmus, amikor a program a megfelelő utasításhoz ér (a struktogramban Q.rem() vagy Q.remMin()). A táblázatban megjelenített információk esetén a legutóbb módosított értékek háttérszíne a többitől különböző. Így jól látható, ha egy csúcshoz jobb értékeket, jobb utat talált az algoritmus a korábbinál.

Kézi, „papíros” megoldás esetén is ezeket az értékeket számoljuk ki, melyek alapján meghatározható az algoritmus által kapott eredmény.

Ez a három bemutatási mód együttesen teljes képet ad az egyes gráfalgoritmusok működéséről. Látható, hogy a háttérben milyen számítások folynak, illetve azok eredményei. Ezek alapján a megjelenített gráf és az algoritmus belső állapota addig-addig változik, amíg az algoritmustól függően éppen minimális feszítőfát vagy például a legrövidebb utakat kapjuk.

3 Fejlesztői dokumentáció

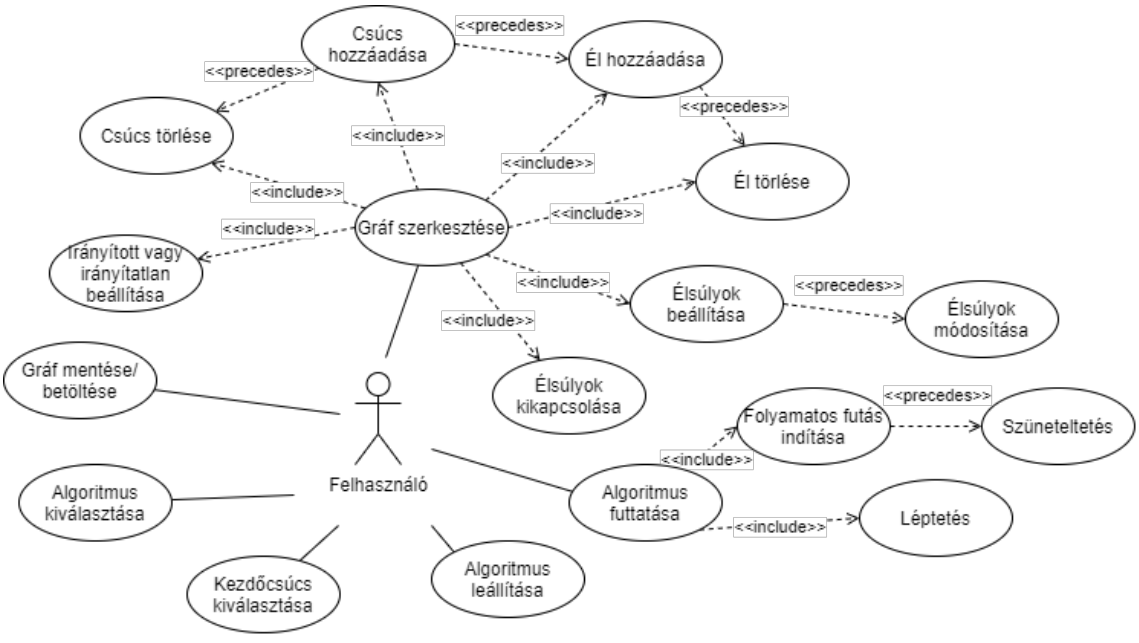
3.1 A program által megoldandó feladat

A program feladata hagyományos, grafikus módon ábrázolt, a felhasználó által megadott gráfon gráfalgoritmusok futásának szemléletes bemutatása. Ehhez a felhasználónak meg kell tudnia adni egy tetszőleges gráfot, majd az általa választott algoritmust futtathatja rajta, lépésenként megtekintheti, hogy milyen hatása van annak a gráfra. Az alkalmazás célja az algoritmusok oktató jellegű bemutatása, így a számítás lépései a gráf bejárásának, és annak eredményének grafikus megjelenítésével együtt láthatóak, ezzel biztosítva a megértést.

Az alkalmazásnak szüksége van egy grafikus felületre, mely lehetőséget biztosít a gráfok szerkesztésére és az algoritmusok futtatására. Annak érdekében, hogy a jól megszerkesztett példák ne vesszenek el, el kell tudni menteni és később betölteni a programba a gráfokat.

3.2 Felhasználói esetek

A 15. ábrán a program felhasználói esetei láthatóak.



15. ábra: Felhasználói eset diagram

3.3 A program szerkezete

A program háromrétegű (modell-nézet-perzisztencia) architektúrában készült C++ nyelven, Qt (5.15) keretrendszer segítségével.

16. ábra: Program rétegei

A különböző rétegek és az azokon belüli osztályok események és függvényhívások segítségével kommunikálnak egymással. A nézet feladata a megjelenítés és a felhasználó tevékenységeinek közvetítése a program logikája, a modell felé. A modell két fő feladatot lát el; egyrészt tartalmazza az alkalmazásban éppen szerkesztett gráf reprezentációját – lehetőséget adva annak szerkesztésére is – másrészt tartalmazza a gráfalgoritmusok implementációját és kezelésükhöz szükséges metódusokat.

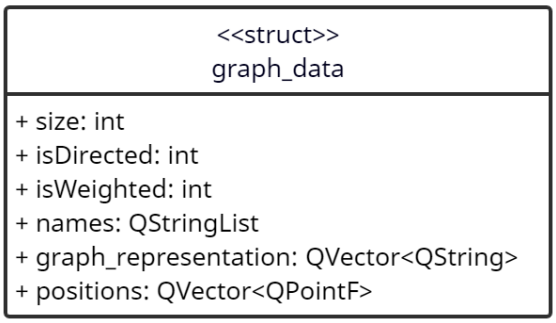
A perzisztencia rétegének egyetlen feladata az alkalmazásban megszerkesztett gráfok mentése szöveges állományba, vagy betöltése fájlból. A mentés és a fájlok megnyitása mindhárom rétegen átívelő feladat. Miután a felhasználó jelzi a mentési szándékát, a nézetnek meg kell jelenítenie az operációs rendszerre jellemző fájlkezelő ablakot (megfelelő fájlformátum beállításokkal), melyben a felhasználó kiválasztja a mentés helyét, és megadja a fájl nevét. Ha ez a művelet sikeres, akkor a nézet meghívja a modellben a mentésért felelős függvényt, és átadja neki a gráf csúcsainak pozícióit, illetve a mentés helyének elérési útvonalát. A modell feladata az gráf adatainak összegyűjtése és átadása a perzisztencia számára annak érdekében, hogy az ezeket az adatokat a megnevezett fájlba beleírhassa. Fájl megnyitásakor is ehhez hasonló folyamat fut végig az alkalmazás rétegein keresztül.

3.4 Perzisztens adattárolás

A perzisztencia egyetlen osztályt tartalmaz: *FileManagement.* Az osztálynak két függvénye van, *saveGraph* és *loadGraph*. Mindkettő megkapja paraméterként a fájl elérési útvonalát, melybe menteni vagy melyből betölteni kell a gráf adatait. Mentés esetén, amennyiben a fájl még nem létezik, előbb létrejön egy új fájl a megfelelő névvel és formátummal, majd megtörténik az adatok mentése. Ha létező fájlt választ a felhasználó, akkor – megerősítést követően – annak tartalmát az új adatok felülírják.

Egy *graph_data* struktúra típusú objektum tárolja a mentés során szükséges adatokat, illetve megnyitáskor a fájl tartalmát ilyen formában tárolhatóvá alakítja. A modell és a nézet mentés esetén a gráf felépítésének adatait és a csúcsok pozícióit egy ilyen

objektumba gyűjti ki, mely alapján a szöveges fájlt a FileManagement osztály elkészíti. Adatok beolvasását követően a modell is ugyanebből a struktúrából nyeri ki a gráfról az információkat, majd ezt követően a nézet a kapott pozíciók alapján megjeleníti a mentéskor látott állapotot.

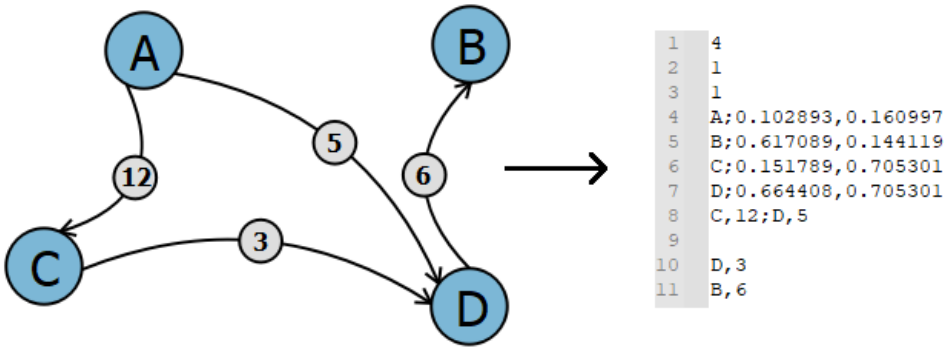


17. ábra: Adatkezelés által használt struktúra

Az adatokat a program szöveges fájlba menti. A fájlban az adatok tárolása a következők alapján, sorokba rendezve történik:

1. *gráf csúcsainak száma* egész számként megadva,
2. *irányított-e* a gráf, 0 és 1 értéket vehet fel, ahol 0 a hamis, míg 1 az igaz értéket reprezentálja,
3. *élsúlyozott-e* a gráf, 0 és 1 értéket vehet fel, ahol 0 a hamis, míg 1 az igaz értéket reprezentálja,
4. *csúcs neve;x koordináta,y koordináta* (soronként az egyes csúcsok koordinátái),
5. *csúcs neve,él súlya* párok pontosvesszővel elválasztva. Minden csúcshoz tartozik egy ilyen sor, melyben a belőle kiinduló éleket és élsúlyokat írják le ezek a párok.

A 18. ábrán látható egy példa erre a leképezésre. Az első sorban a gráf csúcsainak száma (4) olvasható, majd ezt követi, hogy a gráf irányított (1) és élsúlyozott (1). A következő négy sor a csúcsok ábécérendben vett neveit, és koordinátáit sorolja fel, majd megint négy sor írja le az egyes csúcsokból kiinduló éleket a gyermekcsúcsok és élköltségek felsorolásával. A példán látható B csúcs szomszédainak fenntartott kilencedik sor üres, mert egy él sem indul ki belőle.



18. ábra: Megszerkesztett gráf mentési fájlban való reprezentációja

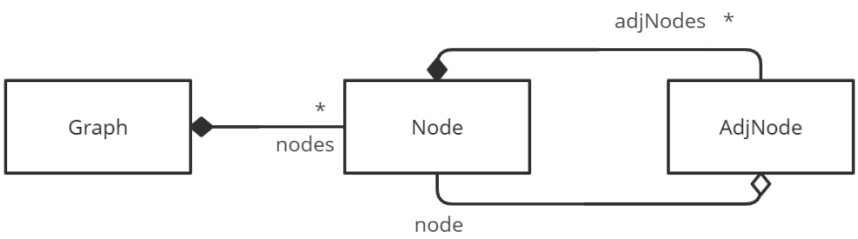
A csúcsok koordinátáit kizárólag a kirajzolás miatt kell tárolni, tehát csak a nézetben van szerepe, míg a többi adatra a gráf modellbeli reprezentációjában is szükség van.

Gráf fájlból való betöltése esetén ellenőrzi a program az adatok helyességét. Amennyiben a szöveges állomány tartalma nem ábrázol a várt formában gráfot, a betöltés megszakad, sikertelen lesz, amit a nézet hibaüzenettel jelez a felhasználó számára.

3.5 Gráfok reprezentációja

A program logikájának egyik jelentős részét képezi a gráfok kezelése. Az ehhez szükséges metódusokat a *Graph* osztály tartalmazza.

A gráfok megvalósítása szomszédsági listás ábrázolással, a 19. ábrán látható osztályok segítségével történik. A szomszédsági listás ábrázolás lényege, hogy a gráf csúcsainak objektumai tárolják a saját gyerekeik listáját, azaz azt, hogy a gráf mely más csúcsaiba vezet él belőlük. Amennyiben élsúlyozott az ábrázolandó gráf, a súlyok tárolása is ebben a listában történik, hiszen a szomszédos csúcsok felsorolásával az adott csúcsból kivezető éleket határozzuk meg, tehát megadható itt azok élköltsége is. Ez a reprezentáció irányított és irányítatlan gráfok ábrázolására is alkalmas. Irányított gráfok esetében a fentebb leírtak egyértelműek, irányítatlan esetben pedig úgy kell leírni a gráfot, mintha az élei mindkét végpontjuk felől irányítottak lennének. Így minden él kétszer kerül felsorolásra, azonban az irányítatlan gráfok bejárása is lehetséges és egyértelmű. A program csak egyszerű gráfokkal foglalkozik, így sem hurokélek sem párhuzamos élek megvalósítására sincs szükség.



19. ábra: Gráfokat leíró osztályok és kapcsolataik

A gráf csúcsait egy-egy *Node* típusú objektum reprezentálja. A Graph osztály tartalmaz egy Node* típusú objektumokat tároló, *nodes* elnevezésű QVectort. Ebben vannak felsorolva a gráf csúcsai. A csúcsoknak van neve (*name*), ami egyetlen karakterből áll, és őket egyértelműen azonosító, egyedi *id*-juk, mely egy egész szám. Az éleket és szomszédos csúcsokat az *AdjNode* osztály írja le, ebből következően a csúcsok AdjNode*-okat tároló QVectorokkal rendelkeznek, mely a szomszédaik listáját tartalmazza. Egy AdjNode objektum egész számként tárolja az általa meghatározott él súlyát, illetve a gyerekcsúcs, azaz az él végpontjának mutatóját. Ezen osztály objektumainak egyetlen feladata az élek reprezentálása, így más adattagokkal nem rendelkezik, és csupán egy *get* és egy *set* metódusa van az élköltség lekérdezésére és beállítására (a konstruktoron kívül).

3.5.1 Node osztály

A Node osztály a gráf csúcsait valósítja meg. A csúcsok egyedi megkülönböztetésére egy egész számú *id* adattag szolgál. Új csúcs létrehozásakor ez automatikusan beállításra kerül az osztályszintű *node_ids* adattag alapján. A *node_ids* alapértéke a program indításakor nulla, és új csúcsok felvétele esetén mindig eggyel nő. Ha a teljes gráf törlésre kerül, akkor a számozás ismét nulláról kezdődhet, hiszen nincs egy csúcs se, nem lehetnek megegyező értékű azonosítók. A visszaállításról a *resetIds* osztályszintű metódus gondoskodik. A csúcsoknak a következő feladatokat kell ellátniuk, így ezeket megvalósító metódusokkal rendelkeznek:

- azonosító lekérdezése (*getId*)
- név lekérdezése (*getName*)
- gyerekcsúcsok számának lekérdezése (*getAdjNum*)
- annak a ténynek a lekérdezése, hogy van-e él az adott csúcsból egy paraméterként átadott másik csúcsba (*hasEdge*)
- élköltségek lekérdezése (*getWeight*)

- adott gyerekcsúcs nevének lekérdezése (*getAdjNodeName*)
- „él megfordítása” – irányítatlan gráfok esetében a szomszédsági listás reprezentáció miatt új él felvételekor az él fordítottját is fel kell venni, törléskor pedig törölni (*setReversedEdge, deleteReversedEdge*)
- él törlése (*deleteEdge)*

A gráf objektuma tartja nyilván a felhasználható neveket, így az gondoskodik a csúcsai elnevezéséről is. Ezért a név beállításának is van függvénye a csúcs osztályában. Egy gyerekcsúcs nevét és a hozzá vezető él költségét a szomszédok listából lehet megkapni. Lehet ezt a listából közvetlenül kiolvasva (index alapján) vagy a csúcs mutatója alapján keresve.

A többi, fentebb említett funkciót megvalósító metódusok csúcsok mutatóit (Node*) várják paraméterül, amik kijelölik az élt, amin a művelet végrehajtandó. Ehhez a *findAdjNode* elnevezésű privát metódust hívják segítségül, mely feladata a Node* mutatóval paraméterként átadott gyerekcsúcs megkeresése a szomszédos csúcsok listájában. Ez a függvény a megtalált csúcs indexével vagy -1 értékkel tér vissza, amennyiben a csúcs nem található.

Az ábrázolás sajátosságából adódik, hogy irányítatlan gráf esetén az élek hozzáadása és törlése egy-egy extra művelettel jár, mert ugyanaz az él mindkét végpont felől el van tárolva. Él felvétele esetén a szomszédok listájába kell új értéket felvenni, míg törléskor onnan egy meglévőt törölni. Vagyis az adott él mindkét végpontján, azok gyerekeit tartalmazó listáin szintén elvégzendőek ezek a műveletek. A gráf tudja magáról, hogy irányított-e vagy sem, így a Graph objektum feladata gondoskodni arról, hogy minden felvétel és törlés megtörténjen.

3.5.2 Graph osztály

Egy gráf mérete a csúcsainak számával adható meg, ez a Graph osztályban a *nodes* lista elemszámát jelenti. Általában egy gráf tetszőleges méretű lehet, azonban jelen esetben egy osztályszintű változó (*maxNodeNum*) maximalizálja ezt az értéket. Új csúcsok hozzáadása esetén mindig ellenőrizni kell, hogy még nem érte-e el a csúcsok száma ezt a maximumot. Új csúcs felvétele kétféleképpen történhet; egy paraméter nélküli függvény segítségével (*addNode*), mely hozzáad egy új elemet a csúcsok vektorához a következő azonosítóval és az első éppen nem használt névvel, illetve annak egy paraméterrel túlterhelt változata, mely hívásakor a csúcs nevét, egy karaktert kell megadni. Gráf fájlból

való betöltése esetén alkalmazandó a névvel ellátott függvény, annak érdekében, hogy a mentéskor látott állapotot a nevek is tükrözzék.

A csúcsok az angol ábécé szerint kapnak nevet. Az első, az adott pillanatban nem használt betűt kapják meg A-tól kezdődően. Ebből következik, hogy az új csúcs neve nem feltétlenül az előzőleg hozzáadott nevét követő betű – például csúcs törlése esetén, következőnek a törölt csúcs neve kerül kiosztásra –, ezért a használatban lévő neveket egy *names* elnevezésű logikai értékeket tároló vektorban tartja nyilván az objektum. A nevek karakterek, így a vektor indexei egyértelműen megfeleltethetőek a karakterkódoknak, ahol az A karakter foglalja el a nulladik helyet. Csúcsok felvételekor és törlésekor figyelni kell ezen értékek frissítésére. A *getNames* QStringList visszatérési értékű függvény a használatban lévő neveket gyűjti össze.

A gráf belső szerkezetét, adatait a Graph osztályon keresztül lehet elérni, így a gráfra, a csúcsokra és az élekre vonatkozó lekérdező és módosító műveletek is megtalálhatóak benne. A gráf irányítottsága és élsúlyozottsága egy-egy logikai értékként kerül eltárolásra, ezeket lekérdezni és módosítani is lehet az adott Graph objektumon. Új csúcs létrehozásakor a gráf beállítja az új Node objektumon az egyedi azonosítót, és éppen nem használt nevet ad neki. A gyerekcsúcsok listája kezdetben értelemszerűen üres, a gráf objektumnak azzal teendője nincs.

Az élekre, kapcsolatokra vonatkozó adatokat a csúcsok listájából kell kinyernie a Graph objektumnak. A Node osztály élekre vonatkozó függvényeinek két csúcsra vett megfelelője szerepel a Graph osztályban is. A gráf megkeresi a kezdőpontot a csúcs listájában, majd annak – a kívánt funkciónak eleget tevő – metódusát alkalmazza a végponttal meghatározott élre.

A gráf a csúcsait a *nodes* vektorban szereplő indexük, az egyedi azonosítójuk vagy a nevük alapján kezeli. Így a gráfot leíró, módosító függvények többféle paraméterezéssel szerepelnek. A paraméterek által megjelölt csúcsok objektumait privát függvények keresik meg, és szintén privát, már Node mutatókat használó függvények hajtják végre a tényleges módosításokat. Így többféle módja van a gráf megadásának, azonban a háttérfolyamatok mindig ugyanazt teszik, és a kód sem lesz redundáns.

A *setEdge* és *deleteEdge* függvények is paraméterként várják a kezdő- és végpontját az adott élnek, így az a szerkezeti leírásban egyértelműen beazonosíthatóvá válik. A kezdőpont kikeresendő a csúcsok listájából, majd az a saját szomszédainak listáján keresi

meg a törlendő élt, vagy ad hozzá újat. Új él felvétele esetén lehetőség van egy harmadik paraméter megadására is, ami az új él súlyát határozza meg. Ennek alapértékként 1 van meghatározva. Ez azt jelenti, hogy súlyozatlan gráfok esetében is élsúlyokkal jönnek létre az új élek. Annak ellenére, hogy ez az adat mindig eltárolásra kerül, élsúlyozatlan gráfok esetében egyszerűen figyelmen kívül hagyható.

Csúcs törlése (*deleteNode*) azonosító alapján történik. A Node osztály a destruktorában törli a belőle induló éleket – az *adjNodes* lista elemeit –, tehát a kiinduló élekkel nem kell ekkor külön foglalkozni. A többi csúcs gyerekeinek listájából azokat az elemeket kell eltávolítani, amelyek a törölt csúcsot tartalmazzák.

Lehet még adott számú csúcsot egyszerre (a hozzájuk tartozó élekkel együtt) (*deleteNodes*), illetve akár az egész gráfot is törölni (*deleteAll*). Utóbbi esetben az összes csúcs objektum törlésre kerül, és a segédértékek (*node_ids*, *names*) visszaállnak az alapértékeikre.

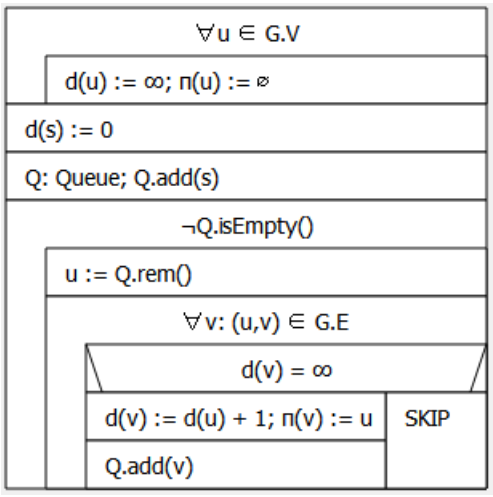
A mentés és betöltés feladatát a *saveGraph,* illetve a *loadGraph* metódusok látják el. A fájlból való olvasás, és fájlba írás a perzisztencia réteg feladata, ezeket a FileManagement osztály egy objektuma látja el. A Graph osztály menthető formába csomagolja az adatokat mentéskor, és a beolvasott adatokat kicsomagolja betöltéskor, vagyis a kapott adatokat megfelelteti a gráf reprezentációjában szereplő adattagoknak, felépíti belőlük a gráfot.

3.6 Gráfalgoritmusok elemzése

Az egyes algoritmusok implementálásának alapjául a [2]-ben és [4]-ben található struktogramok, jegyzetek szolgáltak. Az alábbiakban a program által bemutatott gráfalgoritmusok elemzése olvasható. Az elemzés célja az algoritmusok megértése annak érdekében, hogy azt bemutatható lépések sorozatára lehessen bontani, majd megtalálni azokat a közös pontokat bennük, melyek segítségével a folyamat a lehető legáltalánosabban leírható. Így a bemutatás folyamata a hasonlóságok nyomán egységesíthető. Az alább látható struktogramok megegyeznek a program felületén megjelenítettekkel.

3.6.1 Szélességi bejárás

A szélességi bejárás a gráf egy tetszőleges *s* csúcsától indítható algoritmus. Célja meghatározni, hogy a többi csúcs minimum hány élből álló úton keresztül érhető el a kezdőcsúcsból. A *d* tömbben az utak hosszát, míg a π tömbben azokat a szülőcsúcsokat tartja nyilván, amiken keresztül az adott csúcsba a legkevesebb élt felhasználó út vezet.



20. ábra: Szélességi bejárás struktogramja

Ha egy csúcs elérhetetlen *s*-ből, akkor a *d*-beli értéke ∞, azaz egy maximalizált érték marad, ezzel jelképezve, hogy egy út sem vezet hozzá. A feldolgozandó csúcsok a *Q* sorban találhatóak, mely kezdetben csak *s*-t tartalmazza. A számítás lényegi része a második ciklus. Az algoritmus megállási feltétele – tehát a ciklusfeltétel – a *Q* sor kiürülése. Minden ciklusmag elején a sorból kiveszünk egy csúcsot, ez lesz *u* értéke, azaz az aktuálisan feldolgozandó csúcs, melyet a gráf csúcsainak listájában található indexével jelölünk.

A szélességi bejárás szintenként járja be a gráfot, tehát a feldolgozandó csúcs minden gyerekét megvizsgálja mielőtt újabb elemet terjesztene ki. A gyerekek bejárásáért a belső ciklus felelős. Az egyes csúcsok gyerekei – a gráf ábrázolása miatt – a csúcs objektumában található vektorban helyezkednek el, így a belső ciklus tulajdonképpen ennek a vektornak az elemeit járja be.

A főciklust „új csúcs kiválasztása” és egy „gyerekcsúcs vizsgálata” – azaz a belső ciklus egy iterációja – típusú lépésekre lehet szétbontani. Ez utóbbi lépés ismétlése 0…(*n*-1) alkalommal történhet meg (ahol *n* a gráf mérete), hiszen a belső ciklus a gyerekcsúcsok mennyiségének megfelelő alkalommal ismétlődik. A bemutatandó lépések szempontjából az alábbi esetek állhatnak fent:

1. Új feldolgozandó csúcsot kell választani a főciklus első iterációja alatt, vagy azért, mert az előző csúcs gyerekeit mind végignézte az algoritmus.
2. A vizsgált csúcs következő gyerekcsúcsához vezető út ellenőrizendő (ha nincsenek gyerekei a csúcsnak, ismét az első pont teljesül). Ekkor megvizsgálandó ehhez a gyerekcsúcshoz vezető út *d*-ben nyilvántartott hossza.

Ha az ∞, akkor még nem érte el korábban az algoritmus ezt a csúcsot, tehát megtaláltuk az egyik legrövidebb utat hozzá. A *d* és a π értékeit ez alapján be kell állítani, illetve fel kell venni a *Q* sorba a kiterjesztendő csúcsok közé. Ha a csúcs *d* értéke korábban már megváltozott, akkor ahhoz már ismerjük a legrövidebb utat, nincs teendője az algoritmusnak (hiszen az összes többi út legalább olyan hosszú, mint a korábban talált.).

Az aktuálisan kiterjesztett csúcsot jelentő *u* értéke és annak egy gyerek csúcsának indexe (*adj_ind_in_u*) alapján eldönthető, hogy milyen lépés hajtandó végre. Kezdetben *u* értéke -1, ezért a ciklusmag legelső végrehajtásakor egyértelmű, hogy ennek kell új értéket kapnia, egyébként pedig akkor kell ezt a lépést végrehajtani, ha a gyerekcsúcsok indexe elérte az *u* gyerekcsúcsainak számát. Ez utóbbi esetben azért kell új *u*-t választani, mert az aktuális csúcs gyerekeit mind megvizsgálta az algoritmus, ekkor az *adj_ind_in_u*, tehát a következőnek vizsgálandó gyerekcsúcs indexe 0-ra állítható. Az összes további esetben egy gyerekcsúcsot kell megvizsgálni. Az ellenőrzést követően az indexet növelni kell annak érdekében, hogy a következő lépés a következő szomszéddal foglalkozzon (vagy új kiterjesztendő csúcsot válasszon).

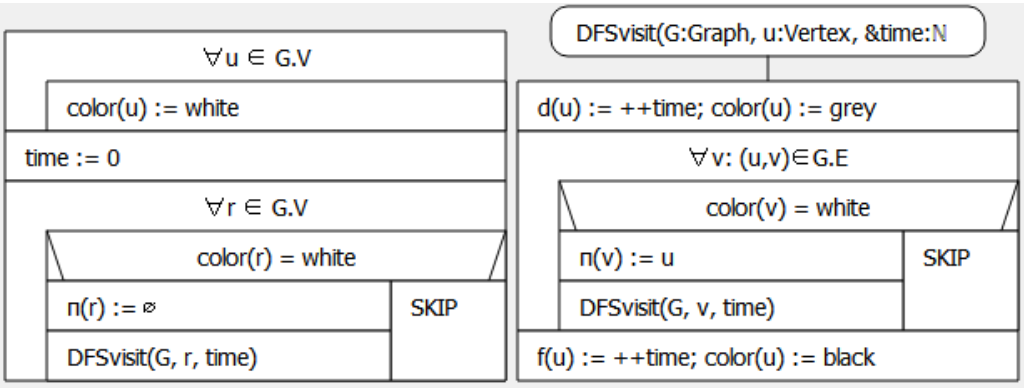
Amennyiben kiürült a *Q* sor, és a szomszédos csúcsok vizsgálata véget ért, az algoritmus futása is befejeződik. Minden lépés elején meg kell vizsgálni, hogy vége van-e az algoritmusnak. Egyik fontos feltétel a *Q* sor üressége, azonban e mellett meg kell vizsgálni, hogy a belső ciklusban tart-e a feldolgozás, mert ez esetben előfordulhat, hogy a sor üres, de az algoritmusnak még nincs vége. A további lépésekben lehetséges, hogy valamelyik gyerekcsúcsot fel kell venni *Q*-ba. Ennek eldöntése az *adj_ind_in_u* index segítségével történik.

3.6.2 Mélységi bejárás

Ez az algoritmus a gráfokat úgy járja be, hogy a csúcsok gyerekein keresztül a lehető legmélyebb szintig megy el, amikor pedig nem tud tovább a gráfban lefelé haladni – azért, mert levélcsúcsot talált, vagy azért, mert már csak ismert csúcsokkal találkozik –, akkor visszalép egy szintet. Addig lép vissza, ameddig nem talál olyan felfedezetlen gyerekcsúcsot, amin keresztül megint haladhat lefelé a gráfban.

A mélységi bejárás a gráf csúcsait a *color* címke alapján osztályozza. Ez határozza meg, hogy az adott csúcs feldolgozandó-e. A fehér (*white*) csúcsokkal még nem találkozott az algoritmus a bejárás során, a szürke (*grey*) csúcsokat már elérte, de még nem dolgozta

fel, a fekete (*black*) csúcsokkal pedig nincsen további dolga, elérte és fel is dolgozta őket. Az algoritmusban használt idő (*time*) nulla kezdőértékről indul, és akkor nő, ha az algoritmus egy olyan csúcsot talál, amivel teendője van. Ha először találkozik egy csúccsal, akkor a szürke szín mellett a felfedezési idejét (*discovery time*) is beállítja, ha pedig már visszatér hozzá, mert az összes leszármazott csúcsot kiterjesztette, a befejezési idejét (*finishing time*) állítja be az idő aktuális értékére, illetve a színét feketére. A 21. ábrán látható struktogramban a *d* a felfedezési időket, míg *f* a befejezési időket tárolja.



21. ábra: Mélységi bejárás struktogramja

A mélységi bejárás fő ciklusa a gráf összes csúcsára lefut egyszer, melyekre a DFSvisit eljárást hívja meg, amennyiben a csúcs még felfedezetlen, tehát fehér volt. A gráfban lefelé haladó viselkedést a DFSvisit rekurzív hívása valósítja meg. Ha elér a program egy fehér csúcsot, akkor arra meghívja a DFSvisit metódust. Ekkor megtörténik az elérési idő és a szín beállítása, majd az összes, az algoritmus szempontjából még ismeretlen (fehér) gyerekre meghívódik a DFSvisit. A szülőcsúcs beállításával a mélységi fa épül. Ha nincs egy csúcsnak több fehér gyereke, akkor beállítható a befejezési idő és a program visszatér az előző DFSvisit eljárásba.

A rekurzív függvényhívások miatt ezt az algoritmust nehezebb lépésekre osztani. Azt, hogy a főprogramban vagy a rekurzió valamelyik szintjén tart-e a feldolgozás, egy logikai érték segítségével könnyen el lehet dönteni. Amikor meg kell hívni a DFSVisit metódust, ez az érték igazra állítandó, amikor pedig az utolsó rekurzív hívásból is visszatért a program, hamis értéket kap. A DFSvisit eljárás során három eset lehetséges:

1. Éppen megtaláltuk a csúcsot, és szürkére kell állítani, következőnek a gyerekeit vizsgáljuk. (DFSvisit első utasításblokkja hajtandó végre.)
2. Egy gyerekcsúcsot vizsgálunk, mely következtében vagy nem kell csinálni semmit (SKIP ág) vagy rekurzív függvényhívás történik, tehát következőnek

megint az egyes pont teljesül. Ha nem kellett csinálni semmit, akkor a következő lépésben vagy megint egy gyerekcsúcsot vizsgál a program, vagy feketére állíthatja a csúcsot (ez a gyerekek számától függ).

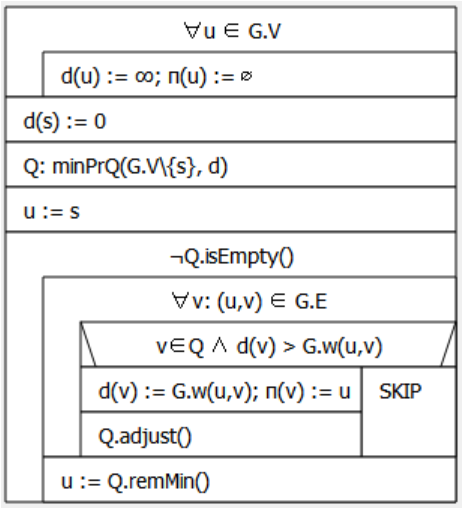
3. Ha nem tud mélyebbre menni az algoritmus az adott csúcstól a gráf szerkezetében, akkor beállítja a befejezési időt és feketére állítja a csúcsot.

Ha az algoritmusnak már visszafelé kell lépnie a gráfban, akkor a következő csúcsot a szülők tömbjéből nyerheti ki, ugyanis ugyanazon az úton kell visszafelé is haladni, amin keresztül elértük az adott csúcsot (vagyis a faéleken keresztül). A pontos követhetőség érdekében nem elég az aktuális csúcs (*u*) és a vizsgált gyerek sorszámát tárolni. A lépések követéséhez az összes csúcs esetében ismerni kell következőként vizsgálandó leszármazott csúcsot. Ezt az *adj_ind_in_us* tömb tárolja, ahol a tömb indexe a gráfon belüli csúcsot, míg az egyes értékei a következőként vizsgálandó gyerekcsúcs indexét jelentik. Ha az adott csúcshoz visszatér a végrehajtás, a megfelelő szintről, megfelelő leszármazottól kell folytatni a bejárást.

Ha a rekurzív függvényhívásokból a főprogramba visszatér a program, annak ciklusa a gráf következő csúcsára indít mélységi bejárást (DFSvisit), amennyiben az még fehér. Ebben az esetben az *r* változó adja meg, hogy mi a ciklus által következőként vizsgálandó csúcs, és akkor van vége az algoritmusnak, ha ennek az értéke eléri a gráf méretét.

3.6.3 Prim algoritmusa

A Prim algoritmus célja a gráf egy minimális feszítőfájának megkeresése. Az algoritmus nagyon hasonlít a fentebb bemutatott szélességi bejáráshoz, így elég csak a különbségekre kitérni. Az algoritmus most is egy tetszőlegesen választott csúcsból indítható; ez egycsúcsú fa, tehát a gráf egy minimális feszítőfájának biztosan a részfája. Ez után újabb és újabb csúcsokat kell hozzávenni ehhez a fához egy jó éllel úgy, hogy az továbbra is a gráf egy minimális feszítőfájának részfája maradjon. A Prim algoritmus feladata ezeket a „jó éleket” megtalálni.



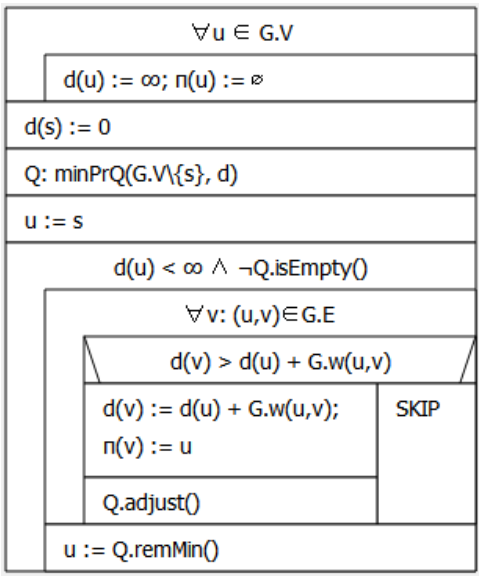
22. ábra: Prim algoritmus struktogramja

A korábban látott *d* tömb értékei most élsúlyokat jelentenek. Annak az élnek a súlyát, amely az adott csúcsot az eddig felépített fához csatolja. A feldolgozandó csúcsokat most egy, a *d* tömb értékei szerint rendezett minimum prioritásos sor tartja nyilván, mely kezdetben a gráf kezdőcsúcstól különböző csúcsait tartalmazza. Azért nem elég most egy egyszerű sor, mert az éleket a súlyaik szerint sorba állítjuk, míg a szélességi bejárás esetében nem teszünk különbséget közöttük.

Ennél a gráfalgoritmusnál is a *Q* sor ürességének vizsgálata a második ciklus feltétele. Ezen ciklus magjában az utolsó teendő az új elem választása. Elég a ciklusfeltételt vizsgálni az algoritmus futásának befejezéséhez is, hiszen ez az ellenőrzés közvetlen a sor egy elemének eltávolítása után történik meg. Az *u* új értékének kiválasztásán kívül itt is egy belső ciklus található még, mely az aktuális csúcs gyerekeit járja be. A Prim algoritmus akkor állít a *d* és π tömbökben új értéket, ha a vizsgált gyerekcsúcs még a *Q* sor eleme és az eddiginél kisebb élköltségű éllel csatlakoztatható a fához (az is lehet, hogy korábban nem csatlakozott). Ennek hatására a *Q* sor is változik, hiszen az a *d* tömb szerint rendezett. A sorból kivett elem már minimális élköltséggel szerepel a fában.

3.6.4 Dijkstra algoritmusa

A Dijkstra algoritmus egy adott kezdőcsúcstól számítja ki az élsúlyokkal vett távolságokat a gráf egyes csúcsaira. Ez az algoritmus mind a szélességi bejárásra, mind a Prim algoritmusra hasonlít. A legrövidebb utak hosszát a korábban is látott *d* tömb tárolja. A feldolgozandó csúcsokat a *d* szerint rendezett minimum prioritásos *Q* sor tartalmazza.



23. ábra: Dijkstra algoritmus struktogramja

Jelen esetben is egyik feltétele az algoritmus befejezésének, hogy a Q sor kiürüljön, azonban itt az is véget vethet a futásnak, ha a sorból utoljára kivett csúcshoz semmilyen út sem vezet, tehát a d-beli értéke ∞. Ez utóbbi esetben a sor d szerinti rendezése miatt tudható, hogy már egyik csúcshoz sem létezik út s-ből, így felesleges lenne a további vizsgálat.

Ennél az algoritmusnál is először a gyerekcsúcsok bejárása, majd új feldolgozandó csúcs kiválasztása történik a második ciklusban. Egy gyerekcsúcs értékei akkor változnak, ha kedvezőbb költségű utat találtunk hozzá az aktuális csúcson keresztül a korábban ismertnél. Egyéb esetben itt sem tesz semmit az algoritmus.

A program szerkezete, így a tárolandó adattagok a három algoritmus esetében azonosak lehetnek, mert bár a d tömb értékei nem ugyanazt jelentik, az adatok típusa, tárolása azonos. A ciklusok felbontásához azt kell tudni, hogy melyik csúcsot dolgozza fel az algoritmus, és melyik gyerekcsúcsánál jár. Így a lépések egyenként, csupán megfelelő feltételek vizsgálatával végrehajthatóvá válnak.

3.7 Gráfalgoritmusok a programban

A gráfalgoritmusokat az *Algorithm* osztály implementálja. Feladata a fentebb vizsgált algoritmusok lépésenkénti megvalósítása egy, a konstruktorában mutatóként megkapott gráfon. Ezen algoritmusok tevékenységeit általánosan a következő lépésekre lehet lebontani:

1. Kiszámítandó értékek inicializálása

   Az algoritmusok különböző segédstruktúrákban tárolják az aktuális értékeiket, melyek utolsó állapota határozza meg az algoritmus eredményét. Ilyen, mind a négy algoritmusra jellemző érték például az egyes csúcsok szülőcsúcsa, amely alapján meghatározható az adott algoritmus által számolt eredményfa. Az algoritmus kezdetén még egyik csúcsnak sincs szülője, ezt be kell állítani.

   Ezen beállításokról az Algorithm osztály *init* metódusa gondoskodik.

2. Kezdőértékek beállítása

   A gráf bejárását megelőzően be kell állítani azokat az értékeket, amiből a számítások kiindulnak. Ennek leginkább azoknál az algoritmusoknál van jelentősége, amelyek a gráf tetszőleges – a felhasználó által megadott – csúcsából indíthatóak. Így például a Prim algoritmus esetén tudható, hogy a kezdő csúcs 0 élköltséggel kerül a gráf minimális feszítőfájába, így annak értéke már beállítható. A mélységi gráfkeresés a csúcsokat sorban mind bejárja, míg a programban szereplő többi gráfalgoritmus egy-egy sort vagy minimum prioritásos sort használ a még feldolgozandó csúcsok nyilvántartására. Ennél a lépésnél kell ezt a sort is a megfelelő értékekkel feltölteni.

   Ezeket a feladatokat az *initNode* függvény implementálja.

3. Gráf feldolgozása

   Az igazán nagy különbség az egyes algoritmusok között ennél a lépésnél található. A részletes leírásuk a Gráfalgoritmusok elemzése fejezetben olvasható.

A feldolgozást a *stepAlgorithm* metódus kezeli. Ha az első két lépés még nem történt meg, ezekkel kezdi. A legelső lépés elején meg kell vizsgálni, hogy a gráf szerkezete kompatibilis-e a választott algoritmussal. Ehhez a *Graph* osztály következő lekérdező metódusaira van szükség:

- élsúlyozott-e a gráf (*getWeighted*)
- irányított-e a gráf (*getDirected*)
- a gráf összes élsúlya nemnegatív-e (*checkAllEdgesNonnegative*)
  - erre az ellenőrzésre csak Dijkstra algoritmusa esetén van szükség
- összefüggő-e a gráf (*checkConnectivity*)
  - erre az ellenőrzésre csak Prim algoritmusa esetén van szükség

Ha valamelyik ellenőrzés eredménye nem felel meg a választott gráfalgoritmusnak, akkor az inicializáció nem történik meg, és megfelelő eseménnyel jelzi a program a problémát. Ha az ellenőrzések sikeresek voltak, akkor a fentebb kifejtett első két lépés egymás után megtörténik. A *stepAlgorithm* következő végrehajtása – amennyiben nem szakítja félbe a gráfalgoritmust a *reset* függvény hívása – a harmadik lépést fogja megtenni. Egészen az algoritmus befejezéséig ez a lépés fog ismétlődni.

A program célja a fentebb említett gráfalgoritmusok bemutatása lépésenként úgy, hogy a haladás felhasználói oldalról legyen előidézhető. Egy-egy ilyen lépés az algoritmus legkisebb olyan egységét[6] tartalmazza, mely a gráfon láthatóan megjeleníthető és logikailag szorosan összefüggő. Az algoritmusokat úgy kell megalkotni, hogy ezek a lépések egyesével végrehajthatóak legyenek. Ez többek között azzal jár, hogy az algoritmusok fentebb harmadikként definiált szakaszát – a fő ciklust – nem lehet egy az egyben implementálni és futtatni, mert akkor felhasználói szempontból egy lépés lenne a teljes gráf feldolgozása. Ez akkor lenne megfelelő, ha az elvárás csupán az eredmény megjelenítése, és nem a folyamat bemutatása lenne.

A fentebb említett okok miatt, az Algorithm osztály globális változókkal tárolja, hogy hol tart egy gráfalgoritmus végrehajtása. Ezen változók értékeitől függ, hogy az algoritmus mely részét kell az aktuális lépésben végrehajtani. Az egyes lépésekben az algoritmus megfelelő részei hajtódnak végre, azonban az eredeti ciklusok helyett azokat helyettesítő feltételvizsgálatok alapján kerül meghatározásra a végrehajtandó kódrészlet. Globális adattagok tárolják az algoritmusok futása során kiszámítandó értékeket, és a sor tartalmát is, amely a feldolgozandó csúcsokat tartalmazza. Ez a sor (*queue*) egy egész számokat tartalmazó vektor segítségével valósítható meg. Lehetne sor adattípust is használni, de a Prim és Dijkstra algoritmusok minimum prioritásos sort használnak, és vektorként megvalósítva az egyszerű sorból és a minimum prioritásos sorból való kivétel is megoldható, tehát ugyanaz az adatszerkezet alkalmazható. A többi tömbszerű tárolóban (távolságok (*distances*), szülők (*parents*) stb.) az indexek az Algorithm objektumnak átadott gráf csúcsaival, azok vektorban való elhelyezkedésük alapján feleltethetőek meg, a sor ezeket az indexeket tárolja. Mivel az indexek egyediek és az algoritmusok futása során nem változnak, így ez egyértelműen meghatározza a továbbiakban feldolgozandó elemeket. Egyszerű sor esetén új elemet a vektor végére kell fűzni, kivételkor pedig az

[6] Ez általában egy vagy több értékadást jelent.

első elemet lekérdezni, majd törölni kell. A minimum prioritásos sorok esetén szintén a vektor végére fűzendő az új elem, a kivételhez pedig a *remMin* elnevezésű algoritmust kell implementálni, melynek feladata a megadott vektor *distances* szerinti legkisebb elemének visszaadása és annak a vektorból való eltávolítása. Ebben az egész számokat tároló vektorban kell megkeresni a legkisebb értéket, mely indexe jelöli a következőként feldolgozandó csúcsot. Így ezt a csúcsot – azaz indexet – adja meg értékül és távolítja el a sorból a *remMin* függvény.

A használt gráfalgoritmusok egy *Algorithms* elnevezésű felsorolási típus (enum) elemeiként határozhatók meg. Így az Algorithm osztály egy ilyen típusú változóban tárolja azt az algoritmust, amely lépéseit a gráfon végre kell hajtani. Szintén felsorolási típusok értékeiként kerülnek tárolásra a csúcsok (*NodeType*) és az élek (*EdgeType*) aktuális állapotai. Az algoritmusokban a csúcsok a következő állapotokat veszik fel:

- még nem történt vele semmi (*BaseNode*),
- az algoritmus éppen ennek a tulajdonságait vizsgálja (*ExaminedNode*),
- kiterjesztés alatt álló csúcs, tehát éppen az ebből induló élek bejárása folyik (*ExamineAdj*)
- feldolgozott, tehát olyan csúcs, amivel további feladata az algoritmusnak nincs (*Processed*)

A mélységi és a szélességi bejárás miatt szükség van még egy állapot bevezetésére (*ReachedButNotProcessed*), mely az ezeknél az algoritmusoknál hagyományosan szürkének nevezett, tehát olyan csúcsokat jelenti, amit a bejárás már megtalált, de még nem dolgozott fel.

Az élek állapotait is vezetni kell, ezek mind a négy algoritmusnál a következők lehetnek:

- még nem történt vele semmi (*BaseEdge*)
- az algoritmus által számított eredményfának része (*NeededEdge*)
- az algoritmus által számított eredményfának nem része *(NotNeededEdge*)
- aktuálisan vizsgált él (*ExaminedEdge*)

A mélységi bejárás az éleit négy osztályba sorolja. A fa-élek azok az élek, melyek a számított mélységi fa élei, tehát *NeededEdge* típussal írhatók le. A további három éltípusra még három elemmel bővíteni kell a fentebbi listát; vissza-él (*BackEdge*), előre-él (*ForwardEdge*) és kereszt-él (*CrossEdge*) hozzáadása szükséges.

Ezek a csúcs- és élosztályozások az algoritmusok futtatásai szempontjából feleslegesek, azonban a bemutatás miatt szükségesek. A változásokat események kiváltása is követi, melyek ezeket az állapotokat használják fel. A nézetnek elég ezekre az eseményekre feliratkozni, és az eseményeknek megfelelő módosításokat véghez vinni a felületen.

Az Algorithm osztály tartalmaz egy időzítőt, mely a *startAlgorithm* függvényhívás hatására indul el. Ez másodpercenként jelez, aminek hatására a *stepAlgorithm* meghívódik, és az algoritmus lépései végrehajtódnak. Publikus metódus maga a *stepAlgorithm*, így nem csak időzítve, hanem teljesen különálló lépésekként is végrehajtható az algoritmus futásának folyamata. Ha időzített módon indult a végrehajtás, akkor a *pauseAlgorithm* hívásával szüneteltetni lehet azt. Ekkor az utoljára kapott állapot megmarad, ismételt indítás után ugyanúgy másodpercenként történnek a lépések, vagy a *stepAlgorithm* hívásával egyesével is léptethető az algoritmus. A választott gráfalgoritmus teljes futtatásának félbeszakítását a *reset* függvény írja le. Ekkor az időzítő leáll, és minden, a számítások alatt használt adattag visszakapja az alapértékét.

3.7.1 Visszafelé lépés

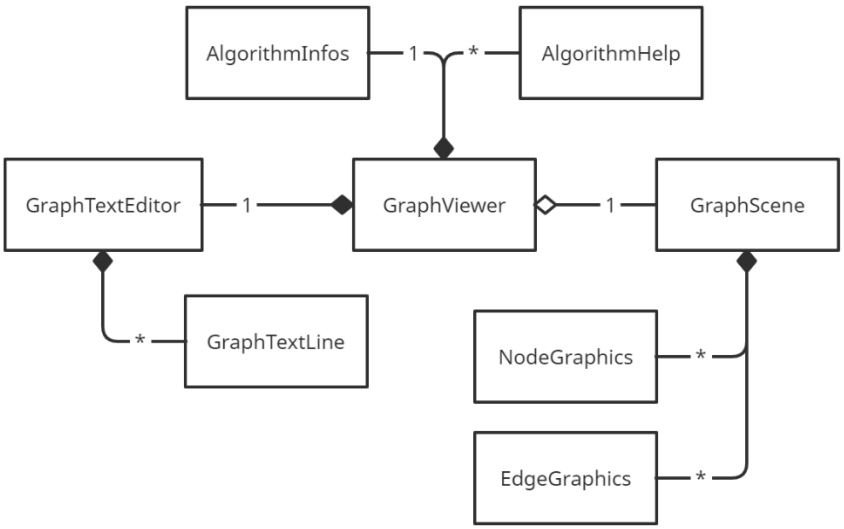
Nem lenne teljes a bemutatás, ha a léptetés csak előrefelé lenne lehetséges, így a gráfalgoritmusokban visszafelé is lehet lépni. Ehhez a gráfalgoritmus belső állapota minden – a külső szemlélő számára is továbbított – változása tárolásra kerül. Ebben az *AlgorithmState* struktúra játszik szerepet, melyben a számítások során használt globális belső változók szerepelnek. Ennek következtében bármely algoritmus, bármely állapotát le lehet írni úgy, hogy az ne csak visszaállítható legyen, de az algoritmus a futását onnan problémamentesen folytathassa.

Az Algorithm osztály *addState* elnevezésű, paraméter nélküli metódusa menti el az aktuális állapotot, tehát ez a függvény minden lépés végén meghívandó. Az állapotok a *steps* nevű, verem típusú adattagban kerülnek tárolásra. Kezdetben ez a verem üres, majd minden lépés végén a tetejére kerül az aktuális állapot. Visszalépéskor így egyesével visszatölthetőek az algoritmus korábbi állapotai. Ezek a veremből törölhetőek is, hiszen, ha megint előrefelé történik a futtatás, a gráfalgoritmusok lépéseit az utolsó állapot szerint kell végrehajtani, az új állapotok pedig megint a verem tetejére kerülnek.

3.8 Megjelenés

A nézet a 24. ábrán látható osztályokat valósítja meg. A megjelenítés középpontjában a *GraphViewer* osztály áll, mely az alkalmazás fő ablakaként funkcionál, feladata a nézet

elemeinek kezelése. Ez az osztály tartja a kapcsolatot a modellel, egy-egy objektumot példányosítva a Graph és az Algorithm osztályokból, hogy azok metódusait meghívja, eseményeire feliratkozzon.



24. ábra: Nézet által megvalósított osztályok

A főablak (GraphViewer) a QMainWindow osztály leszármazottja, így menü és eszközsávok hozzáadása egyszerű, elhelyezkedésük is adott a felületen. A menüben a mentés és megnyitás, gráf törlése, szöveges megadás, rendezés, algoritmus segítség és névjegy menüpontok találhatók, melyek kattintására történő feladatukat QAction típusú objektumok definiálják. Ezek paramétereként beállítható gyorsgomb is.

Két eszköztárra van szükség, egyiken a gráf szerkesztéséért felelős gombok találhatóak (*editToolbar*), míg a másikon az algoritmus választásáért és futtatásáért felelős vezérlők helyezkednek el (*algorithmToolbar*).

3.8.1 Gráf megjelenítése

A gráf grafikus szerkesztése és megjelenítése *QGraphicsItem* típusú elemek megjelenítésére alapszik. A megjelenítésért a *QGraphicsView* típusú *view* objektum, az eseménykezelésért és elemek tárolásáért pedig az annak konstruktorában átadott *QGraphicsScene* objektum gondoskodik. A programban a QGraphicsScene-ből származtatott *GraphScene* osztály egy objektuma tölti be ezt a szerepet. A QGraphicsItem típusú objektumok a *view* vezérlőn mozgathatóak az *ItemIsMovable* állapotjelző (flag) beállítása után, ezért a mozgatás megvalósítására külön nincs szükség. A gráf megjelenítéséhez két egyedi, a QGraphicsItem-ből származtatott osztályra, grafikus

objektumra van szükség. Ezek a gráf egy csúcsát és egy élét megjelenítő objektumok. A megjelenés személyre szabása az ezekhez az objektumokhoz tartozó ecset (*brush)* és toll (*pen*) beállításával, valamint a *paint* metódus felüldefiniálásával történik. Az ecset az alakzat kitöltését, míg a toll a körvonalát határozza meg. Mivel az algoritmusok szimulációja közben az alakzatok színei mutatják, hogy hol tart a feldolgozás, mind a csúcsok, mind az élek esetében a *brush* vagy a *pen* objektum módosítása feleltethető meg azok állapotváltozásainak. Így az állapot változásakor csak meg kell változtatni, hogy az objektum melyik előre definiált ecsetet vagy tollat használja önmaga kirajzolásához. A *paint* metódus törzsében egy *QPainter* rajzolóobjektum segítségével határozható meg, hogy pontosan hova és mit rajzoljon a program az adott alakzat megjelenítésekor.

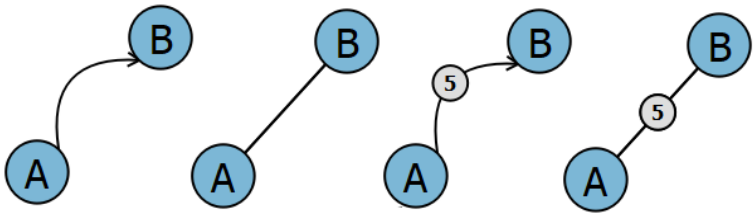
A gráf csúcsa (*NodeGraphics*) kör alakja miatt a *QGraphicsEllipseItem* osztályból származik le, hiszen ez létrehozásakor a konstruktorban megadott befoglaló téglalap alapján egy ellipszist jelenít meg. A csúcsok méretét az osztályszintű, konstans *size* adattag írja le, így létrehozáskor ennek segítségével határozható meg a befoglaló négyzet mérete, hogy az összes csúcs azonos méretű legyen. Egy ilyen objektum a modellben szereplő gráf egyetlen csúcsát, azaz egy Node-ot hivatott megjeleníteni. Mivel a reprezentációban egy csúcs azonosítója és neve nem változik, ezek az adatok a grafikus objektumnak is átadhatóak. A név a megjelenítés miatt szükséges, hogy a csúcs közepére kiírható legyen, míg az azonosító a későbbiekben a grafikusan megjelenített csúcs és annak modellbeli reprezentációja közötti megfeleltetés miatt nélkülözhetetlen. A csúcs osztályszintű QColor változókban színeket határoz meg. A kitöltés színe egy csúcs állapotváltozása esetén ezek között vált. Az *ItemSendsGeometryChange* állapotjelző beállítását követően felüldefiniálható az *itemChange* metódus. Az *ItemIsMovable* beállítása után alapesetben egy grafikus objektum tetszőlegesen mozgatható – így akár az ablak látható területéről is eltüntethető. Ezért az előbb említett metódus az eredeti funkcióján túl azt az esetet is kezeli, amikor a mozgatott elem eléri az őt tartalmazó *scene* objektum határát, és nem engedi azon túl mozgatni azt.

Az élek (*EdgeGraphics*) az előre definiált objektumok közül a *QGraphicsLineItem* típusúakra hasonlítanak leginkább, így a származtatásuk ebből az osztályból történik. Azonban esetükben (a csúcsokhoz képest) több változtatásra van szükség. Az élek megjelenítése függ az azokat tartalmazó gráf beállításaitól, így más a megjelenítés irányított, irányítatlan, súlyozott és súlyozatlan esetekben is. Ezeket a tulajdonságokat az élek logikai típusú adattagokban tárolják magukról, a kirajzolás ezeket ellenőrzi. A

csúcsokhoz hasonlóan az alakzat pontos meghatározása és kirajzolása a felüldefiniált *paint* metódusban történik. A függvény elején meg kell határozni azt a szakaszt, ami az élt jelenti, tehát két gráfbeli csúcsot köt össze. Ehhez a *calculate* függvény implementálására van szükség. Az éleknek a csúcsok körvonalától kell indulnia, azonban az objektumok koordinátáit a befoglaló négyzetük bal felső sarka határozza meg, így az élt reprezentáló egyenes végpontjait a csúcsok koordinátáihoz képest el kell tolni. Ebben a két összekötendő csúcs koordinátái által meghatározott egyenes egységvektora és a QMatrix osztály lineáris transzformációi segítenek. Ha a gráf élei irányítatlanok, akkor ez a vonal meghatározza az él egyenesét, azonban irányított gráf esetén ez nem elég.

Amennyiben két csúcs között mindkét irányban van irányított él, akkor ezek eltakarnák egymást, csak az utolsóként megjelenített lenne látható, arra lehetne csak rákattintani (törléshez vagy súly beállításához). Ezért az éleknek irányított esetben görbének kell lenniük, hogy elkerüljék egymást. A görbe vonal megalkotásához még egy extra pontra van szükség, mely a korábban kiszámított egyenes középpontjától, annak normál vektorának irányában eltolva helyezkedik el. Az irányított él helyét négyzetes Bezier görbe adja meg. Kiszámítására a *quadTo* függvény használható, ami az extra pontot kontrollpontként és az egyenes eredeti végpontjait várja paraméterként [5]. Az él végén található nyilat két szakasz alkotja, melyek hossza fix, az irányukat pedig a görbe szöge határozza meg. Ez annak az egyenesnek a szöge, amely a kontrollpontot a végponttal köti össze.

Ha a gráf súlyozott, akkor az élsúlyok is megjelenítendőek az él közepén, egy körben ábrázolva azt. A kiírandó súlyt az él a *weight* nevű változójában tárolja.

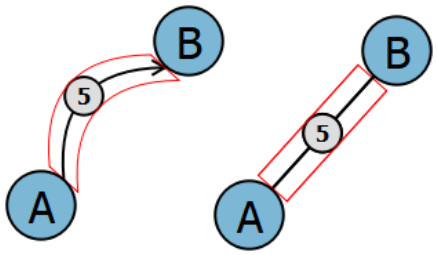


25. ábra: Egy él lehetséges megjelenési formái

Az élek azonosítása a végpontjaik alapján történik, így azok grafikus megfelelőit tárolják az él objektumok.

A QGraphicsItem osztály és leszármazottai számára a *boundingRect* függvény a befoglaló téglalapot adja meg, azaz azt, hogy mekkora területen látszik az alakzat. Így,

ha ez a téglalap kisebb a kirajzolt alakzatnál, bizonyos részei nem lesznek láthatóak. A *shape* függvény egy összetett alakzat formájában (*QPainterPath* segítségével) adja meg azt a területet, melyen az alakzat ténylegesen megtalálható, a 26. ábrán ez piros körvonallal látható. Ennek a területnek a jól megközelített leírása a kattintás szempontjából fontos, ugyanis a súly állításához és az él törléséhez is az élre (azaz erre a területre) kell kattintani. Nem célszerű csak az él vonalát kijelölni, ugyanis akkor a felhasználónak nagy pontossággal kellene az él fölé vinnie az egeret, ami felhasználói szempontból kényelmetlen lehet.



26. ábra: Kattintáskor élként figyelt terület (shape)

Az ősosztály (QGraphicsLineItem) az egyenes számára meghatározza a *boundingRect* és a *shape* visszatérési értékét, azonban az irányított él a nyíl vonalai és az él görbesége miatt, a súlyozott él pedig a kirajzolt súly miatt lóg ki az örökölt területekből – ezeket a függvényeket felül kell definiálni. A *boundingRect* beállításakor elég az, hogy a teljes él a téglalap belsejében legyen, nem kell pontos számítás, azonban nem lehet túl nagy sem, mert az a megjelenítéskor több számítási igénnyel is bír. A *shape* esetében már pontosabbnak kell lenni, így az él egy szűk környezetét kell definiálni, mint „kattintható területet” az objektumok számára. Az élek mozgatására nincs szükség, azok a csúcsokkal együtt mozognak.

A csúcsokhoz hasonlóan az élek színei is aktuális állapotaiknak megfelelően változnak, így ezeket tárolni és állítani kell. Az élek esetében nem a kitöltés, hanem a körvonal színe változik. A mélységi bejárás osztályozza az éleket, ezen élosztályok az élek állapotai között is szerepelnek. Mivel a mélységi bejárás súlyozatlan gráfokkal dolgozik, a súly helyén írja ki a rajzolófüggvény az él számított típusát, amennyiben az nem fa-él. Fa-él esetén az eredményfa része az adott él, amit csak a színe jelöl.

A csúcsok és az élek kapcsolatait a mozgások követése és a törlések érdekében kétféleképpen tárolja a program. Egyrészt egy NodeGraphics objektum rendelkezik az

általa megjelenített csúcshoz kapcsolódó élek grafikus megfelelőinek listájával, másrészt egy EdgeGraphics objektum ismeri a kezdő- és végpontját jelentő grafikus csúcsobjektumokat. Egy csúcs mozgatása esetén az összes hozzá kapcsolódó él pozíciója frissül azok *calculate* függvényének hívásával. Csúcs törlésekor a hozzá tartozó éleket a másik végpontjuk listájából is törölni kell az él törlése előtt, él törlése esetén pedig mindkét csúcs listájából el kell távolítani annak mutatóját.

A fentebb leírt objektumokat a GraphScene osztály tartalmazza és kezeli. Ez a QGraphicsScene leszármazottja, a fő megjelenítés és grafikus gráf-objektumok között hídként, közvetítőként szolgál. A *GraphMode* felsorolási típus a szerkesztési állapotokat határozza meg, ez a következők egyike lehet: mozgatás (*Move*), csúcs hozzáadása (*AddNode*), él hozzáadása (*AddEdge*), súly beállítása (*SetWeight*), illetve törlés (*Delete*). Az egérkattintás hatása a *mode* adattagban tárolt szerkesztési állapottól függ.

A mozgatás művelete esetén ugyanaz történik egérkattintásra, mint felüldefiniálatlan esetben a QGraphicsScene osztályban. Mivel a *scene* QGraphicsItem típusú objektumokat tárol (melyek számára már implementálva van a mozgatás), és a csúcsok és élek is ennek leszármazottjaiként lettek létrehozva, egyszerűen mozgathatóak az objektumok, csupán a megfelelő állapotjelző (*ItemIsMovable*) beállításának kérdése. A korábban leírtak szerint csak a csúcsok mozgathatóak, ezek pozícióváltozásait az élek követik.

Új csúcs hozzáadása esetén a kattintás pozíciója a létrehozandó csúcs középpontja, így a csúcs befoglaló négyzetének bal felső sarkát ezen pozíciótól a csúcsot jelentő kör sugarával vízszintesen és függőlegesen is eltolva kell megadni.

A többi mód esetén meg kell határozni, hogy valamilyen grafikus elemre kattintott-e a felhasználó. Él hozzáadása két csúcs egymást követő kiválasztásával lehetséges. A *currFrom* NodeGraphics* objektum tárolja a létrehozandó él kezdőpontját, tehát a korábban választott csúcsot. Mód váltása vagy a már kijelölt kezdőpont ismételt kiválasztása esetén a mutató értéke *null*-ra változik. Élsúly módosítására kijelölni lehet egy élt, majd az új súly beállítása a főablak feladata. A kijelölt él az EdgeGraphics* típusú *currEdge* adattagban található, mely a csúcshoz hasonlóan, más él választása vagy módváltás esetén módosul, vagy üres lesz. Törléskor a csúcs vagy él megfelelő függvényeit kell meghívni.

A gráf irányítottságának vagy élsúlyozottságának módosulásakor az éleknek a *scene* adja tovább a beállítás paramétereit. Az élek létrehozásakor azok *setDirected* és *setWeighted* logikai paramétert váró eseménykezelőit a *scene graphDirectedChanged* és *graphWeightedChanged* eseményekhez kapcsolja. Így ezen módosítások esetén a grafikus objektumok is azonnal frissülnek, a megjelenésük rögtön a gráf beállításait tükrözi.

A menüben található *Rendezés* menüpontra kattintva, illetve a szöveges megadáskor a csúcsok számának növelésének hatására a megjelenített csúcsok egy kör vonalára rendeződnek. Ezt a *scene updateNodePositions* függvénye oldja meg, mely paraméterül a rendezendő csúcsok grafikai elemeinek listáját várja.

Annak érdekében, hogy az ablak változásait követve a gráf elemei mindig arányos távolságban legyenek egymástól, az *originalPositions* függvény meghatározza a csúcsok aktuális koordinátáit, majd ezeket az *updatePositions* megkapja, és a számára szintén átadott vízszintes és függőleges arányszámok segítségével transzformálja. Az új koordináták alapján frissül a megjelenés.

3.8.2 Szöveges szerkesztő

A *GraphTextEditor* osztály tartalmazza a szöveges szerkesztéshez tartozó elemeket. Egy ilyen objektumot példányosít a főablak, mely változásait események segítségével figyeli és kezeli, szükség esetén a *scene* felé közvetíti azokat, hogy valós időben frissüljön a gráf megjelenítése. A *graphTextEditor* beállításai az *initViews* függvényben történnek meg. Kezdetben a szerkesztőpanel a felületen nem látszik, a szöveges szerkesztés menüpont hatására jelenik meg, majd a rajta található bezárás gomb megnyomásával ismét eltűnik.

A *graphTextEditor* objektum a gráfot megadó vezérlőket tartalmaz. A csúcsok számát egy *QSpinBox* tartalmazza, mely csak egész, 0 és a gráf maximális mérete közötti számokat fogad el. Ezt az objektum létrehozásakor kell beállítani.

Az élek egy listában láthatóak. Új elemeket egyesével lehet felvenni. Egy élt leíró elem a *GraphTextLine* egy objektuma. Ez az osztály két legördülő menüt tartalmaz a végpontok neveivel, illetve egy szöveges mezőt az élköltség számára. Egy ilyen objektum rendelkezik még három gombbal: szerkesztés, elfogadás és törlés. Az objektum létrehozásakor az elfogadás és a törlés gombok láthatóak. Ha mindhárom vezérlő kapott értéket, akkor az elfogadás gomb megnyomásának hatására az él szerkesztési vezérlőit ki kell kapcsolni, hogy csak olvashatóak legyenek, illetve eseménnyel jelez az élt leíró

objektum, hogy az él felvehető a gráfba. Súlyozott él esetén a súly szerkeszthető. Ekkor meg kell jeleníteni a szerkesztő gombot, és annak kattintására szerkeszthetővé tenni a súlyt. A törlés gombra kattintva eseménnyel jelez a GraphTextLine objektum az őt tartalmazó *graphTextEdit* számára, majd az törli a gráfból és élek megjelenített listájából is.

Azon csúcsok listáját, amelyek között élek képezhetőek folyamatosan frissíteni kell. Új él létrehozásakor a kezdőcsúcs legördülő menüjében azon csúcsok nevei láthatóak, amelyekből még nem megy él az összes többi csúcsba. Ezt a listát az *updateNames* metódus frissíti, melyre új sor hozzáadásakor vagy abban az esetben van szükség, ha az új él még szerkesztés alatt van (nem lett a gráfban elmentve), de egy korábban létrehozott élt töröl a felhasználó. Ekkor lehet, hogy „felszabadul” egy csúcs, ahova megint lehet új élet húzni. A kezdőcsúcs kiválasztását vagy módosítását követően a *fromNodeChanged* eseménykezelő meghatározza a kiválasztott csúcs neve alapján a potenciális végpontokat. Az összes csúcs nevei közül kiveszi a választott csúcs nevét – hiszen hurokélek nem lehetségesek – valamint azon csúcsok neveit, amelyekbe már tart él a megjelölt kezdőcsúcsból. Ezen ellenőrzésekkel és beállításokkal a felhasználó csak helyes élek felvételére képes.

3.8.3 Algoritmusok megjelenítése

Az algoritmusok megjelenítése három részre bontható. Az első a kirajzolt gráfon, annak állapotváltozásait különböző színekkel jelezve történik – ezt a Gráf megjelenítése fejezet bővebben tárgyalja.

A másik kettő az algoritmus struktogramjának és a számított értékeinek megjelenítése. Az algoritmus által a végeredmény szempontjából fontos értékeket az *algoValues* nevű *QGridLayout*-ban táblázatként rajzolja ki a program. Az algoritmusokban használt sorok tartalmának megjelenítésére egy címke (QLabel) szolgál. A kezdeti beállításokat, kirajzolásokat az *initAlgoView* metódus hajtja végre. Az értékek kitöltése az algoritmus objektumának eseményei hatására történik, így aktualizálva folyamatosan a megjelenített tartalmakat. Az *algoInitReady* függvény hatására a táblázat fejlécét kitölti a program a csúcsok neveivel, a sorok címkéi a választott algoritmusban számítandó értékek neveit veszik fel, valamint a kezdőértékek is kiírásra kerülnek. Az algoritmus futása közben a belső állapota változásai eseményekkel jelezi, melyekre a nézet a következő eseménykezelőkkel reagál:

- *parentChanged*: paraméterként a módosult csúcs indexét és az új szülőcsúcs nevét kapja meg. A függvény megkeresi a módosítandó címkét, és átírja a régi szülő nevét az új nevére.
- *distChanged*: paraméterként a módosult csúcs indexét és az új, hozzá vezető út hosszát vagy azon él súlyát kapja meg, ami azt az eredményfához kapcsolja. A megfelelő címkét megkeresi a függvény, majd módosítja az értékét.
- *discoveryFinishChanged*: a módosult csúcs indexét, felfedezési és befejezési idejét kapja meg paraméterként. Ezek közül az értékek közül egyszerre csak az egyik frissül, de egy címkén vannak megjelenítve, ezért a módosításhoz mindkét értékre szükség van.
- *queueChanged*: az algoritmusokban található sor tartalma külön címkén jelenik meg. Paraméterként megkapja a sor tartalmát szöveges formában, így csupán frissíteni kell azt.

Mivel a táblázatban való módosítások a megfelelő csúcsok oszlopait megkülönböztető színnel jelölik a 27. ábrán látható módon, ezért minden állapotváltozás előtt a korábbi színezéseket vissza kell vonni, hogy csak az új módosítások látszódjanak. Ezt a *clearColorsInAlgTable* oldja meg. A táblázat soraiban szereplő összes címke háttérszínét eltávolítja.

	A	B	C	D	E	F
d	0	1	∞	2	∞	∞
π	Ø	A	Ø	B	Ø	Ø

27. ábra: Szélességi keresés során D csúcs értékeinek változása

A struktogram megjelenítéséért az *AlgorithmInfos* osztály felelős. Ezt a vezérlőt az ablak *algorithmInfos* néven példányosítja. A *setAlgorithm* függvény állítja be, hogy melyik algoritmus struktogramját kell kirajzolni. A vezérlő pontos megjelenése a *paintEvent* metódus felüldefiniálásával van megadva. Az egyes struktogramok kirajzolását külön függvények valósítják meg (*paintSzelessegi, paintDijkstra, paintPrim, paintMelysegi*). A struktogramok megalkotásához főként téglalapok és szövegek kirajzolására van szükség. Külön színnel jelezni kell azt is a struktogramon, hogy az algoritmus végrehajtása hol tart. Ehhez bevezetésre kerül az *InAlgorithm* felsorolási típus. Egy ilyen értéket a *rowsToColor* adattag tárol. A kirajzoláskor ennek értékétől függően a struktogram bizonyos részei eltérő háttérszínt kapnak. Azt, hogy melyik részeket kell kiszínezni, azaz

milyen értéket kap az *InAlgorithm* változó, a nézetben található algoritmus objektum eseményei határozzák meg, melyek az algoritmus belső állapotának változásakor jelzik azt is, hogy hol tart a program végrehajtása.

A felhasználhatóság javítása érdekében a struktogram alatt a *paintColorHelp* függvény a megjelenített gráfhoz – a csúcsok és élek színeinek jelentésére – jelmagyarázatot készít.

3.8.4 Segítség

Az *AlgorithmHelp* osztály feladata olyan grafikus elemek megvalósítása, melyek segítenek a programban látható gráfalgoritmusok megértésében. Új ablakban rövid leírást adnak azok működéséről, használatáról.

Ezek az objektumok egy-egy gráfalgoritmus megadásával jönnek létre, hogy azok adatait jelenítsék meg. Tartalmazzák az algoritmus nevét, rövid leírását, és a programban a szimuláció során látható struktogram rajzát is.

Az osztály a négy gráfalgoritmus számára egy-egy statikus logikai értéket tart fent. Ezek azt jelzik, hogy az adott pillanatban létezik-e az adott algoritmussal inicializált példánya az osztálynak. A főablakból hozhatóak létre ezek a „segítség-objektumok”, melyek új ablakban nyílnak meg, így a főablak állapotától függetlenül láthatóak. A logikai értékek ellenőrzésével megkapható, hogy az adott algoritmushoz tartozó segéd-ablak éppen létezik-e, és csak akkor nyíljon új, ha nem. Annak érdekében, hogy a sok új objektumlétrehozás ne okozzon memóriaszivárgást, az *AlgorithmHelp* konstruktorában a *Qt::WA_DeleteOnClose* attribútum beállítása történik. Ennek hatására, ha az adott algoritmust bemutató ablakot bezárja a felhasználó, annak objektuma a memóriából is törlődik.

A *closeEvent* eseménykezelő gondoskodik arról, hogy egy-egy ilyen segéd-ablakot ismét meg lehessen nyitni. Egyetlen feladata, hogy a bezárás hatására a megfelelő algoritmushoz tartozó osztályszintű logikai értéket igazzá állítsa. Ezzel biztosítja azt, hogy a legközelebbi megnyitási kísérlet sikeres lehessen.

4 Tesztelés

A program tesztelése két nagy részre bontható: üzleti logika, illetve felhasználói felület tesztje. Az üzleti logika, azaz a modell egységtesztek (unit test), míg a nézet manuális tesztek segítségével kerül ellenőrzésre.

4.1 Modell tesztelési terve

A modell ellenőrzéséhez a Graph és Algorithm osztályok publikus metódusainak tesztelésére van szükség. Az egyszerű *getter* és *setter* metódusok[7] vizsgálata elhagyható, ezekről feltételezzük a helyes működésüket, így a tesztesetekben alkalmazhatóak. A Qt Creator lehetőséget biztosít tesztprojekt létrehozására. Ehhez a projekt alprojektekbe való rendezésére van szükség, hogy az alkalmazás fájljai elérhetőek legyenek a tesztelés során. Minden metódus esetében ellenőrizendő a helyes viselkedés és a „rossz" paraméterezés is. A rossz paraméterek leggyakoribb esetei: nem létező azonosító, nullpointer, tömbszerű adatszerkezetek (tömb, vektor, lista stb.) esetén azokból kimutató indexek.

4.1.1 Gráf tesztjei

Minden teszteset alatt egy *graph* nevű Graph objektumon történik a gráf függvényeinek hívása. Ez az objektum minden teszteset előtt létrejön, majd a teszteset végén törlődik, így biztosítva az egyes esetek „tiszta lapról" indított ellenőrzését.

Irányítottság és élsúlyozottság beállítása *(testChangeDirected, testChangeWeighted)*

A gráf irányítottságát egy logikai érték átadásával lehet módosítani, melyet a *getDirected* függvény segítségével lehet lekérdezni. A teszt ellenőrzi, hogy az aktuálissal ellentétes és azonos értéket megadva is helyesen történik-e a beállítás.

Az élsúlyozottság ellenőrzése ugyanígy történik. Ezen esetekben nincsenek vizsgálandó szélső értékek.

Új csúcs hozzáadása *(testAddNode)*

Új csúcsot felvenni paraméter nélkül, illetve egy karakter megadásával lehet.

Mindkét esetben legfeljebb a Graph osztály *maxNodeNum* statikus adattagja által meghatározott számú csúcs vehető fel. Ha a csúcsok száma eléri ezt a maximumot, az ismételt függvényhívás ellenére sem fog tovább nőni a gráf csúcsainak száma.

[7] Egy adattag, vagy tároló adattag egy elemének lekérdezése és módosítása

Név szerinti csúcsfelvétel esetén, ha a megadott karakter *A*-nál kisebb[8], vagy *A*-tól számítva a maximális csúcsszámnál nagyobb (15 csúcs esetében a legutolsó használható karakter az *O*), a gráf felvesz magának egy új csúcsot, azonban a helytelen névválasztás miatt az első helyes és még használható nevet választja. A csúcsfelvétel sikertelen, ha az előbbiek szerint helyes, azonban már használatban lévő nevet kap paraméterként a metódus.

Mindkét függvény esetében – ha a paraméterek helyesek és még nem érte el a csúcsok száma a maximumot – a gráf mérete eggyel nő.

Több csúcs hozzáadása *(testAddNodes)*

Paraméterként az *addNodes* függvény egy egész számot vár, ennyi új csúcsot ad hozzá a gráfhoz. Jelen esetben a negatív számok rossz paraméternek bizonyulnak, a gráf csúcsainak száma nem változik ilyen megadás esetén. A gráf méretének a paraméterként megadott értékkel kell nőnie addig, amíg az nem haladja meg a csúcsok maximumát (*maxNodeNum*) – ennél több csúcsot nem tartalmazhat a gráf. Ha meghaladná ezt a felső határt, a csúcsok száma már nem változik ezen függvény hatására.

Azonosító lekérdezése (*testGetId*)

A *getId* függvény a *nodes* vektor egy indexét várja paraméterül. Ha az index helytelen (negatív vagy eléri a tároló méretét), akkor -1-gyel tér vissza. Helyes paraméter esetén a csúcs egyedi, -1-től különböző azonosítóját kapjuk.

Név lekérdezése (*testGetName*)

Paraméterként a csúcs azonosítóját kell megadni. Nem létező azonosító esetén üres karaktert kapunk.

Összes, a csúcsoknak kiosztott név lekérdezése (*testGetNames*)

Ha nincsenek a gráfnak csúcsai, üres listát ad, egyébként pedig a csúcsok neveit gyűjti össze – abban az esetben is, ha megadott névvel lettek azok létrehozva.

Az élekre és gyerekcsúcsokra vonatkozó lekérdezések (*testAdjNodes*)

A *getWeight* függvény két csúcs azonosítója alapján megadja az első csúcsból a másodikba vezető él költségét. A helyes eredmények mellett, azokat az eseteket kell

[8] Karakterek rendezése szerint

megvizsgálni, amikor nem létező azonosítók alapján történik a lekérdezés, vagy az általuk jelölt csúcsok között nem vezet él.

A *getAdjNum* indexet vár, mely a *nodes* egy elemét jelenti. Ha az index értéke nem felel meg a vektornak, akkor a visszakapott érték -1. Az ellenőrizendő helyes eset: a csúcs leszármazottjainak számát adja meg a függvény. Kezdetben ez nulla, majd a belőle kiinduló élek számával egyenlő.

A *getAdjName, getAdjWeight, getAdjIndexInNodes* metódusok tesztelése nagyon hasonló. Mindegyik egy-egy indexet vár: az első a *nodes* vektor elemét határozza meg, míg a második az így megadott csúcs *adjNodes* vektorának elemét jelenti. Tesztelni kell valós indexek esetén a várt eredményt, illetve bármely index helytelensége esetén kapott értékeket is.

Csúcs törlése *(testDeleteNode)*

Ellenőrizni kell a helytelenül megadott azonosító esetét, valamint a sikeres törlést. Mivel a törlést követően a *nodes* is frissül, így egymás után többször az első elem azonosítója alapján való törlés is a várt eredményt hozza.

Több csúcs törlése (*testDeleteNodes*)

Ennek a függvénynek a törlendő csúcsok számát kell átadni. Ellenőrizni kell a működését negatív számra – ekkor semmi sem történik – illetve a gráf méreténél magasabb értékre is. Ez utóbbi esetben az összes csúcsot kitörli a program.

Új él hozzáadása, élsúly módosítása (*testSetEdge*)

Az élek hozzáadása, élsúlyuk módosítása az azokat meghatározó csúcsok azonosítói vagy nevei alapján történik. Ezek a függvények logikai visszatérési értékkel rendelkeznek; igazat adnak, ha új él került hozzáadásra, hamisat, ha nem sikerült hozzáadni vagy pedig az élsúly módosult. A korábbiakban látott szélsőséges eseteken kívül, a gráf ábrázolásából kifolyólag ellenőrizendőek a következők:

- hurokél beállítása nem lehetséges (a tesztelés során kiderült, hogy hurokéleket is fel lehetett venni, így ezt javítani kellett – az alkalmazásban felhasználás közben ez a probléma nem jöhetett elő, mert a nézet is ellenőrizte ezt),
- párhuzamos él beállítása nem lehetséges,
- irányított gráf esetén két csúcs között mindkét irányban felvehető él,

- irányítatlan gráf esetén két csúcs között csak az egyik irányban vehető fel él (a másik irányban felvett él párhuzamos él lenne).

Él törlése (*testDeleteEdge*)

A törlés sikerességét az élsúly lekérdezésével lehet ellenőrizni (ugyanis nemlétező él esetén a *getWeight* INT32_MAX értéket ad). A teszt része a *deleteEdge* függvény meghívása üres gráfra, illetve két olyan csúcsra, amik között nem vezet él. Az utóbbi ellenőrzés nem ment át a teszteken először, lekezeletlen eset volt, amit korrigálni kellett.

Külön vizsgálandó irányított és irányítatlan él törlése is. Irányítatlan esetben az él fordítottja is törlésre kerül.

Gráf törlése (*testDeleteAll*)

Minden esetben ugyanazt az eredményt (üres gráf) adja. Akkor is, ha eredetileg se voltak csúcsok felvéve, akkor is, ha csak csúcsok voltak, és akkor is, ha élek is voltak beállítva.

Rendezés (*testSort*)

A *sortAllChildren* és *sortNodes* paraméter nélküli metódusok, így elég a helyes működésüket ellenőrizni, miszerint a csúcsokat és a gyerekek listáit a nevek szerint rendezi.

Minden élköltség nemnegatív (*testAllEdgesNonnegative*)

Ha nincsenek csúcsok, akkor ez igaz. Egyébként pedig akkor ad hamis értéket, ha legalább egy él súlya negatív.

Összefüggőség (*testCheckConnectivity*)

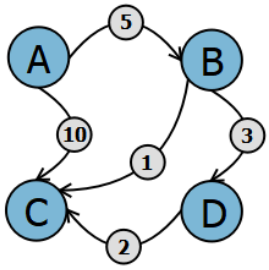
Üres és egycsúcsú gráf összefüggő. Mivel nincs paramétere a függvénynek, így elég az összefüggőség definíciója alapján tesztelni.

4.1.2 Gráfalgoritmusok tesztjei

Minden teszteset alatt egy *algo* nevű Algorithm objektumon történik a gráfalgoritmusokhoz köthető függvények hívása. Ez az objektum minden teszteset előtt létrejön (a korábbi tesztekben látott *graph*-ot megkapva), majd a teszteset végén törlődik.

A gráfalgoritmusok kevés publikus metódussal rendelkeznek, és eredményeiket események kiváltásával közvetítik, így az egységtesztek csak a helyes beállításokat ellenőrzik. A számítások helyessége a felhasználói felület tesztje során állapítható meg.

Mivel a tesztelés során sok különböző gráf létrehozása szükséges, a tesztosztály kiegészítésre került egy privát *buildGraph* metódussal, mely két logikai paramétert vár. Az első a készítendő gráf irányítottságát, a második pedig élsúlyozottságát jelenti. A paramétereknek megfelelő beállításokkal létrejön egy négycsúcsú gráf a *graph* objektumban.



28. ábra: buildGraph függvény által létrehozott gráf

Gráfalgoritmus indítása előtti beállítások (*testSelections*)

A két tesztelendő metódus a *selectAlgorithm*, illetve a *selectStartNode*. Ezek határozzák meg a gráfalgoritmust, illetve azt a csúcsot, amelyből a számítás indul. Amennyiben helyesek ezek a beállítások, az *algo* objektum *getInitState* lekérdezése igaz értéket ad.

Léptetés (*testStepAlgorithm*)

Ha az algoritmus még inicializálatlan állapotban van, akkor a *stepAlgorithm* hívása ellenőrzi a gráf beállításait a gráfalgoritmus feltételei szerint. Ha alkalmazható a gráfon a kiválasztott algoritmus, akkor az inicializálás megtörténik.

Mind a négy gráfalgoritmus esetében ellenőrizni kell, hogy helytelenül megadott gráf esetén a *getInitState* értéke hamis maradjon, tehát ne induljon el a gráf feldolgozása. Például a szélességi gráfkeresés nem történhet meg súlyozott gráfon.

Az egységtesztek futtatása során, a fentebb jelzett javításokat követően minden teszteset végeredménye sikeres volt.

4.2 Felhasználói felület tesztelése

A manuális tesztelés során az alábbi tevékenységek egyike sem okozta a program hirtelen leállását, és mindig teljesültek a várt eredmények.

- A program megnyitása esetén elindul az alkalmazás üres szerkesztőfelülettel.

- A megnyitást követően a jobb oldali eszköztárra való kattintásnak nincs hatása (a vezérlők le vannak tiltva).
- A bal oldali eszköztáron az egyes gombokra kattintás kijelöli azokat.
- A két jelölőnégyzetre kattintással eltűnik vagy megjelenik bennük a pipa. Ha a „Súlyozott” felirat melletti jelölőnégyzetre kattint a felhasználó, kikapcsolásnál elszürkül a súly-eszköz gombja, bekapcsolásnál pedig ismét kattinthatóvá válik.
- A menüben a „Névjegy” feliratra kattintva felugró ablakban információk jelennek meg a programról.
- Az „Algoritmus segítség” menüpontra kattintva legördülő menüben jelenik meg a programban bemutatott négy algoritmus neve.
    - Bármely listaelemre rákattintva új ablak nyílik meg az adott gráfalgoritmus leírásával. A főablakhoz a megnyitott „segédablak” mellett is vissza lehet térni.
    - Olyan algoritmus pontjára kattintás a legördülő menüben, amely ablaka nyitva van, nem csinál semmit.
    - Olyan algoritmus pontjára kattintás a legördülő menüben, amely ablaka nincs nyitva, megnyitja azt.
    - Egy segédablak bezárását követően új ablak nyitható ugyanazon algoritmus kiválasztásával.
- A program bezárása (és ezzel leállítása) bármely állapotában lehetséges.

4.3 Szerkesztés

4.3.1 Grafikus szerkesztés

A szerkesztőfelületen való kattintás – az első eszköztáron kiválasztott gomb funkciójától függő – hatásainak tesztjei.

- A mozgatás, új él hozzáadása, súly módosítása vagy törlés funkciók kiválasztása után, amennyiben a szerkesztőfelület üres, az arra való kattintás hatására semmi sem történik.
- Új csúcs hozzáadása esetén a kattintás helyén létrejön egy új csúcs. Az új csúcs neve mindig az első még nem használt betű az angol ábécéből.
- Ha a csúcsok száma eléri a maximumot (15), akkor több csúcs létrehozása nem lehetséges. Minden próbálkozáskor piros figyelmeztető szöveg jelenik meg.
- Új él hozzáadása csak akkor lehetséges, ha legalább két csúcs van.

    - Egy csúcsra kattintáskor, annak a körvonala piros kiemelést kap. Ismételten erre a csúcsra kattintva eltűnik a kijelölés. Olyan csúcsra kattintáskor, amelybe már vezet innen él, szintén eltűnik a kiemelés. Az előzőtől különböző csúcsra való kattintás következtében új él keletkezik.
- Irányítottság módosítása esetén a látható élek a beállításnak megfelelően módosulnak, az új élek helyesen jönnek létre.
    - Irányítatlan gráf esetén az irányítottság beállítása azt eredményezi, hogy a meglévő élek helyén mindkét irányban irányított él jelenik meg.
- Súlyozottság módosítása esetén eltűnnek vagy megjelennek az élsúlyok. Az új élek a beállításnak megfelelően jönnek létre.
- Ha a gráf élsúlyozottra van beállítva, akkor módosíthatóak az élköltségek. A súly állításakor, az élre való kattintás esetén annak súlya piros körvonalat kap, és az ablak bal felső sarkában megjelenik egy szövegmező.
    - A szövegmezőbe csak számok beírása lehetséges (más meg sem jelenik).
    - Egy egész szám beírását követően az enter gomb nyomására eltűnik a szövegmező, a kijelölt súly körvonala ismét fekete, az értéke pedig az újonnan megadott lesz.
    - Az enter billentyű leütése hatástalan, amennyiben a szövegmező nincs fókuszban vagy üres.
    - A kijelölés változtatására bármikor van lehetőség egy másik élsúlyra való kattintással. Ekkor a szövegmező a helyén marad, az új értéket a legutóbb kiválasztott él kapja.
- Akár csúcs van kijelölve él beállításához, akár élsúly van kijelölve annak átállításához, más szerkesztési mód választásának hatására eltűnik a kijelölés, a szerkesztés zavartalanul folyik tovább.
- Mozgatni csak a csúcsokat lehet, a hozzájuk tartozó élek követik a mozgásukat.
- Él törlésekor eltűnik a kattintott él.
- Csúcs törlésekor eltűnik a csúcs és a hozzá csatlakozó élek.
- A menüben a „Gráf törlése” gombra kattintva a szerkesztőfelület – tartalmától függetlenül – kiürül.
- A menüben a „Rendezés” gombra kattintva a felületen látható csúcsok egy kör vonalára rendeződnek

- Legalább egy csúcs felvételét követően aktívvá válnak a második eszköztár vezérlői. Az összes csúcs törlésekor pedig mindegyik inaktív lesz.

4.3.2 Szöveges szerkesztés

- A menüben a szöveges megadás gombra kattintva az ablak jobb oldalán megjelennek a szöveges szerkesztés vezérlői. A grafikus szerkesztés esetén használható funkciók (az átrendezéseket[9] kivéve), és a jobb oldali eszköztár teljes tartalma inaktív.
- A „Bezár” gomb kattintására a szöveges szerkesztő vezérlői eltűnnek és az eszköztár elemei ismét aktívvá válnak.
- Amennyiben a szerkesztőfelület tartalma nem üres, megnyitáskor a vezérlők ennek megfelelő értékeket kapnak. Az élek sorai súlyozott gráf esetén az élköltségeket is tartalmazzák, egyébként nem.
- A csúcsok száma 0 és 15 között módosítható. Sem a nyilakkal változtatva, sem begépelve nem adható meg ettől különböző érték. A felületen megjelenített csúcsok száma az enter megnyomására vagy abban az esetben változik, amennyiben a vezérlő elveszíti a fókuszt.
- A gráf méretét kisebbre állítva, az élek listájából eltűnnek azok a sorok, amelyek valamely csúccsal együtt törölt élt reprezentálják.
- Ha nem teljes gráf látható a felületen, akkor az „Új él hozzáadása” gomb megnyomásának hatására új él felvételére alkalmas sor jelenik meg.
- Ha a gráf teljes gráf, akkor az „Új él hozzáadása” gomb megnyomásának hatására figyelmeztető üzenet jelenik meg: „Nincs több lehetséges él!”.
- Él hozzáadásakor a pipára kattintás csak akkor hatásos, ha egyik legördülő menüben sem üres a kiválasztott elem.
- Él hozzáadásakor az első legördülő menüben csak azok a csúcsok jelennek meg, amelyekből még nem indul az összes többi csúcsba él. Kezdőcsúcs kiválasztása után a végpontok legördülő menüjében csak azon csúcsok nevei jelennek meg, amelyekbe a választott kezdőpontból vezethet még él. (Irányított és irányítatlan esetben is.)

[9] Az egérrel a csúcsok egyesével vagy a „Rendezés” gombra kattintás hatása az egész gráf átrendezhető.

- Ha az él hozzáadása sikeres, megjelenik a szerkesztőfelületen, és a végpontokat meghatározó legördülő menük inaktívak lesznek, a kiválasztott tartalmuk csak olvasható.
- Ha élsúlyozott a gráf, akkor lehet a súlyát módosítani. Mentés hatására a súly módosul.
- Az élsúly csak egész számként adható meg, más érték nem jelenik meg a szövegmezőben. Ameddig annak tartalma üres, az él mentése nem lehetséges.
- Ha a gráf súlyozatlan a szerkesztést jelölő ceruza nem látszik. (Élt mentés után már nem lehet szerkeszteni.)
- A listából a kukára való kattintással lehet élt törölni, mely a szerkesztőfelületről is azonnal eltűnik.

4.4 Mentés és betöltés

- A szerkesztőfelület tartalma az alkalmazás bármely állapotában (akár gráfalgoritmus szimulációja során is) elmenthető.
- A mentést kiválasztva az operációs rendszerre jellemző fájlnézegető nyílik meg, melyben egy nevet és útvonalat megadva, a „Mentés” gomb megnyomását követően létrejön az így meghatározott fájl. (Vagy felülíródik a már létező állomány tartalma.)
- Gráf megnyitása esetén, ha a kiválasztott fájl tartalma nem írj le a várt módon egy gráfot, akkor a szerkesztőfelület tartalma nem változik, egyébként törlődik, és a betöltött gráf jelenik meg rajta.
- Egy gráf mentését tartalmazó szöveges állományban az extra sorok nem számítanak. Ha kevesebb a sorok száma a vártnál, az hibát okoz. A szerkezeti módosítások hibát okoznak.

4.5 Gráfalgoritmsuok

- Egy gráfalgoritmust annak futási feltételeit nem kielégítő gráfon indítva a gráffal nem történik semmi, azonban figyelmeztető szöveg jelenik meg a helytelen beállításról.
- Gráfalgoritmus futása közben a mozgatáson kívül a szerkesztésért felelős funkciók egyike sem elérhető.

- Egy gráfalgoritmust helyes gráfon, a start gombbal indítva elindul az algoritmus futása másodpercenként módosítva az aktuális állapotot. (Ha eléri a végét, akkor egyszerűen leáll, az utolsó állapotot megjelenítve).
- Az egyesével léptetés hatására a következő feldolgozó lépés látszik a gráfon, a táblázatban és a struktogramon is.
- Az automatikus léptetés során is lehet az előre és vissza gombokkal az algoritmus lépései között váltani.
- A leállítás gombra kattintva a szimuláció állapotától függetlenül visszakerül a program szerkesztő módba.
- A gráfalgoritmusok végrehajtásának helyessége a Mintagráfokon való futtatásukkal ellenőrizhető a forrásokban található pontos lépésekkel való összehasonlítással:
    - Szélességi gráfkeresés: iranyitott_sulyozatlan.graph, iranyitatlan_sulyozatlan.graph [2]
    - Mélységi gráfkeresés: iranyitott_sulyozatlan_3.graph [2], iranyitott_sulyozatlan_4.graph
    - Prim algoritmusa: iranyitatlan_sulyozott_3.graph [4], iranyitatlan_sulyozott_2.graph
    - Dijkstra algoritmusa: iranyitatlan_sulyozott.graph [4]

5 Összefoglalás

A gráfok különböző feladatok modellezésére alkalmazhatóak. A szélességi bejárás, mélységi bejárás, Prim algoritmusa és Dijkstra algoritmusa olyan algoritmusok melyek gráfok bejárására és feldolgozására képesek, hogy információkat nyerjenek ki belőlük. A működésük, alkalmazásuk és használatuk célja más és más, így különböző gráfokon (azaz modelleken), különböző elvárásokkal lehet alkalmazni őket. A kapott eredmények feldolgozásával megoldást lehet találni a kitűzött feladatra.

A dolgozatban tárgyalt program célja az említett négy gráfalgoritmus bemutatása úgy, hogy a felhasználó azok működését lépésről lépésre megismerhesse. Az alkalmazás Qt keretrendszer segítségével készült, grafikus felülettel rendelkezik, melyen gráfok szerkesztésére, azok mentésére (majd később megnyitására), illetve a fentebb említett gráfalgoritmusok szimulációjának megtekintésére, kezelésére van lehetősége a felhasználónak. A gráf megszerkesztésére kétféle lehetőség van: egyrészt a szerkesztőfelületen „összekattintgatva”, másrészt űrlapszerű elemek kitöltésével. Az egyes gráfalgoritmusok szimulációját, a gráf feldolgozását annak grafikus megjelenítésén, illetve az algoritmus struktogramja és a kiszámított értékek alapján lehet követni. Ezek a felületen mind megjelenítésre kerülnek.

A gráfalgoritmusok implementációjához azok eredeti szerkezetének átalakítására volt szükség annak érdekében, hogy a ciklusok és – a mélységi bejárás esetében – rekurzív függvényhívások ne egyben fussanak le, hanem bemutatható lépésekre lehessen bontani azokat.

A végeredmény egy gráfszerkesztő alkalmazás lett, melyben a fentebb említett négy algoritmus lépései egy tetszőleges (a programban megalkotható) gráfon „oda-vissza” megtekinthetőek, a számítások lépései folyamatosan követhetőek.

6 További fejlesztési lehetőségek

„Mentés” és „mentés másként” lehetőségek megoldása. Jelenleg csak a mentésre van lehetőség, így mindig új fájlt kell választani. Az átalakítás után, ha a felhasználó betölt egy gráfot fájlból, majd módosítás után a mentésre kattint, akkor nem kell új fájlt választania, azt módosítja, amit megnyitott. Akkor kell fájlt választani, ha a „mentés másként”-et választja, vagy úgy ment, hogy nem nyitott meg korábban egy gráfot.

Jelenleg folyamatosan futtatva egy másodpercenként lehet a gráfalgoritmusok szimulációját megtekinteni. Fejlesztési lehetőség a folyamatos futás lépési idejének beállítása.

A még hatásosabb bemutatás érdekében a struktogramon az egyes utasításblokkokban általános elnevezések helyett meg lehetne jeleníteni azon csúcs nevét, amit az adott sor módosít, lekérdez.

Jelentős átalakítással járna a program módosítása úgy, hogy a gráfalgoritmusok a felhasználó által is „beprogramozhatók”, majd elmenthetők legyenek. Ehhez a programhoz már adatbázist lenne érdemes kapcsolni. Ha van adatbázis, akkor pedig a gráfok mentése és betöltése is átalakítandó. Az adatbázis egyrészt tárolná a mentett gráfokat azok nevével és adataival (több tábla segítségével reprezentálható az egész gráf a modellbeli megjelenéséhez hasonlóan), illetve az általa szimulálni képes algoritmusokat, melyet a felhasználó a programban „alkothat meg” – például struktogramok vagy pseudo-kód segítségével. A létrehozáskor a felhasználónak kellene megadnia az algoritmus leírását, és a szimuláció előtt ellenőrizendő feltételeket. Ehhez a programot fel kell készíteni a gráfok egyéb tulajdonságainak ellenőrzésére, mert jelenleg csak azokat tartalmazza, amik a használt algoritmusok számára szükségesek.

Ez az átalakítás sok tervezést és a szerkezet jelentős átdolgozását jelentené, hiszen a program jelenlegi helyzetében több pontján is épít arra a tudásra, hogy melyik gráfalgoritmusok szerepelnek benne, illetve azok pontos működése is a programkód része.

7 Irodalomjegyzék

[1] Google, „Introducing the Knowledge Graph,” [Online]. Available: https://www.youtube.com/watch?v=mmQl6VGvX-c&t=1s. [Hozzáférés dátuma: 22 04 2021].

[2] dr. Ásványi Tibor, „Algoritmusok és adatszerkezetek II. előadásjegyzet: Elemi gráfalgoritmusok,” 5 10 2020. [Online]. Available: http://aszt.inf.elte.hu/~asvanyi/ad/ad2jegyzet/ad2jegyzetGrafok1.pdf. [Hozzáférés dátuma: 19 02 2021].

[3] „towards data science,” [Online]. Available: https://towardsdatascience.com/10-graph-algorithms-visually-explained-e57faa1336f3. [Hozzáférés dátuma: 22 04 2021].

[4] dr. Ásványi Tibor, „Algoritmusok és adatszerkezetek II. előadásjegyzet: Élsúlyozott gráfok és algoritmusaik,” 17 11 2020. [Online]. Available: http://aszt.inf.elte.hu/~asvanyi/ad/ad2jegyzet/ad2jegyzetGrafok2.pdf. [Hozzáférés dátuma: 12 02 2021].

[5] W. Spekkink, „Wouter Spekkink,” 11 09 2018. [Online]. Available: https://www.wouterspekkink.org/software/q-sopra/technical/qt/2018/09/11/drawing-parallel-edges-in-qt.html. [Hozzáférés dátuma: 05 02 2021].