EÖTVÖS LORÁND TUDOMÁNYEGYETEM

INFORMATIKAI KAR

ALGORITMUSOK ÉS ALKALMAZÁSAIK TANSZÉK

GRÁFALGORITMUSOK ÉS HATÉKONY ADATSZERKEZETEK SZEMLÉLTETÉSE

Témavezető:

Veszprémi Anna
mestertanár

Készítette:

Bognár Gergő
programtervező informatikus BSc

Budapest, 2012.

TARTALOMJEGYZÉK

I. BEVEZETÉS

Dolgozatom témája egyrészt néhány gráfalgoritmus bemutatása volt: páros gráfok vizsgálata, maximális párosítás, és minimális költségű feszítőfa illetve feszítő erdő keresése, másrészt annak szemléltetése, hogy ezen algoritmusok hatékony implementációjában használt adatszerkezetek hogyan működnek. A két témakör köré csoportosuló algoritmusokban közös, hogy egy optimális megoldást adnak egy adott problémára, és jól szemléltethetőek, mivel a megoldó algoritmusok alapötlete a gráf éleinek vagy csúcsainak csoportosítása, színezése.

Munkám eredménye egy interaktív, grafikus felületű, Java virtuális gépen futó, akár a böngészőből is elindítható program. Az elkészített felület lehetőséget nyújt gráfok létrehozására, megjelenítésére és szerkesztésére, valamint a kiválasztott algoritmusok szemléltetésére, azok hatékony implementálása mellett. Az algoritmusok áttekintése és lépésenkénti nyomon követése mellett bemutatja az implementációjukban kulcsfontosságú, speciális adatszerkezeteket is.

Páros gráfok és maximális párosítás témakörben irányítatlan, egyszerű gráfok párosságának vizsgálatát szemléltetem szélességi bejárás segítségével, valamint bemutatom a *magyar módszer* működését páros gráf maximális párosításának megkeresésére. Minimális feszítőfa és feszítő erdő témakörben irányítatlan, egyszerű, élsúlyozott gráfokon szemléltetem a piros-kék eljáráson alapuló *Prim* és *Kruskal* algoritmusok működését. Az algoritmusok megvalósítása során használt hatékony adatszerkezetek közül a *sort*, a *kupacot* és az *unió-holvan* adatszerkezetet mutatom be.

A bemutatott algoritmusok többsége az Algoritmusok és adatszerkezetek II. tárgy során tárgyalásra került. Munkám során motivált, hogy az elkészült szemléltető program tanulási segédeszközként is hasznosítható legyen, elősegítse az algoritmusok megértését, azokkal történő megismerkedést. Munkámmal ugyanakkor egy kitekintést is szerettem volna nyújtani a tárgy keretein túlra, bemutatva olyan, általam érdekesnek vélt algoritmust és adatszerkezetet is, melyet a tárgy csak érintett.

II. FELHASZNÁLÓI DOKUMENTÁCIÓ

Szakdolgozati munkám egy interaktív, grafikus felületű program, melynek két fő feladata van: gráf szerkesztés és algoritmus szemléltetés.

A gráf szerkesztő egy felületet biztosít gráfok létrehozására, megjelenítésére, szerkesztésére és tárolására. A program irányítatlan, egyszerű gráfokkal dolgozik. A gráf csúcsait egyedi sorszámokkal lehet címkézni, melyek egész számok. A sorszámok szerepet kapnak az algoritmusok futásakor – ha az algoritmusnak több azonos tulajdonságú csúcs közül kell választania egyet, akkor mindig a legkisebb sorszámút választja. A gráf élei igény szerint lehetnek súlyozottak, ekkor egész számokat lehet az élekhez súlyként hozzárendelni. A gráf szerkesztő felület működésével, az általa nyújtott lehetőségekkel a *Felhasználói felület* részben foglalkozom részletesebben.

Az algoritmus szemléltetés során a négy kiválasztott algoritmus és a megvalósításukhoz használt adatszerkezetek kerülnek megjelenítésre. Az algoritmusok az egyszerű áttekintés mellett lépésekre bontva is nyomon követhetők. Az algoritmusok részletes leírását – amely magába foglalja az algoritmus lépésekre bontásának és az algoritmussal együtt bemutatott adatszerkezetnek a megadását – a *Bemutatott algoritmusok és adatszerkezetek* rész tartalmazza, az algoritmusokat szemléltető felület használatát a *Felhasználói felület* részben tárgyalom.

2.1. RENDSZERKÖVETELMÉNYEK

A program Java nyelven íródott, futtatásához Java virtuális gép, legalább Java SE 6 szükséges. A bájtkódokat tartalmazó Java archívum (*graf.jar*) általában a *java -jar graf.jar* paranccsal, Windows rendszereken a fájl megnyitásával indítható. Parancssori indításkor további opcionális paraméterként megadható a megnyitandó, gráfot tároló fájl neve, vagy a „.” paraméter. Ekkor a program a kezdőlap átugrásával betölti a megadott fájlt, illetve létrehoz egy új, üres gráfot.

A programnak létezik böngészőből indítható, beágyazott Java applet változata is. Ennek futtatásához a Java virtuális gép mellett appleteket támogató böngésző szükséges. Windows 7, Windows XP és Ubuntu Linux rendszereken végzett teszteléseim alapján a program appletként történő futtatásához Java SE 7, Firefox 11, Internet Explorer 9 vagy Google Chrome 17 optimális. Az applet digitális aláírással rendelkezik, indításakor ennek jóváhagyását kéri.

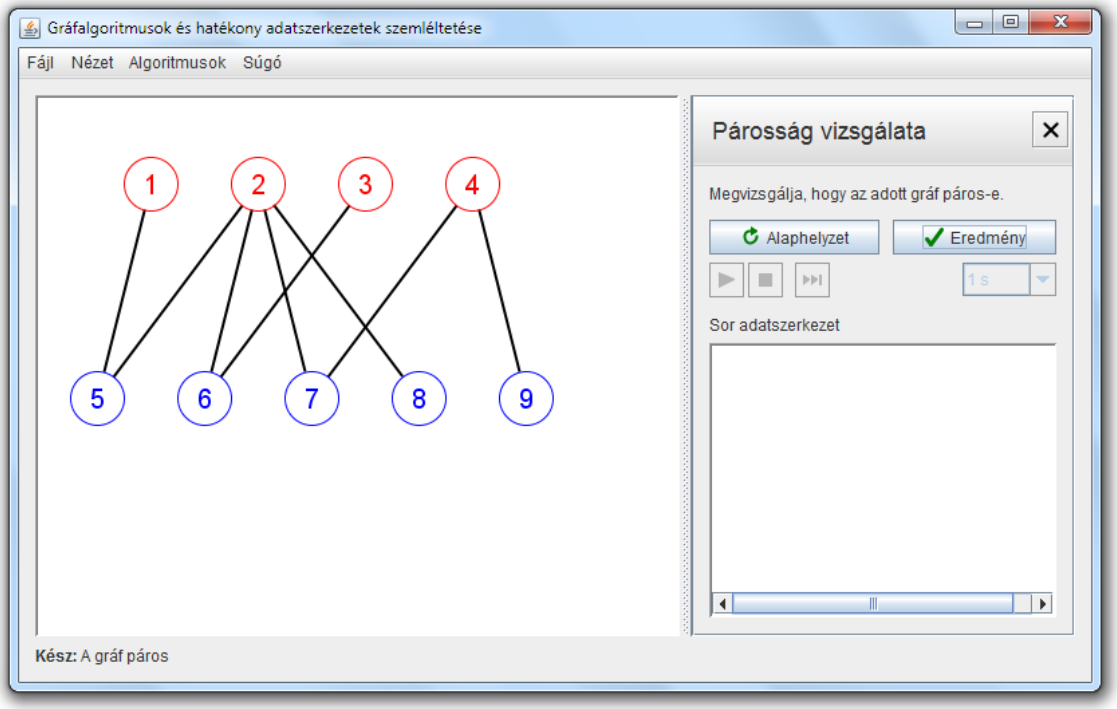
2.2. BEMUTATOTT ALGORITMUSOK ÉS ADATSZERKEZETEK

Ebben a részben a programomban szemléltetett algoritmusok leírása szerepel. Ismertetem az algoritmusok működését, műveletigényét és az algoritmus megvalósításához szükséges adatszerkezetet. A továbbiakban az algoritmusok leírásánál n-nel jelölöm a gráf csúcsainak számát, e-vel éleinek számát, V-vel a gráf csúcshalmazát, E-vel az élhalmazát – amelyet $V \times V$ speciális részhalmazának tekintek. A gráf csúcsai az implementációban rendelkeznek sorszámmal és színnel ($u \in V$ esetén *u.sorszám* és *u.szín*), valamint rendelkezhetnek egyéb attribútumokkal is (például *u.d*, a csúcshoz rendelt távolságérték és *u.p*, az őt megelőző csúcs a bejárás során). A gráf élei pedig súllyal és színnel rendelkeznek ($e \in E$ esetén *e.súly* és *e.szín*). A műveletigény meghatározásakor feltételezem, hogy a gráfokat éllistával ábrázoljuk, mivel az ismertetett algoritmusok implementációja ritka gráfokon ilyen ábrázolás mellett hatékony.

Az algoritmusok megvalósításánál és leírásánál több forrásra támaszkodtam. A gráf párosságának vizsgálatánál az Algoritmusok és adatszerkezetek II. tárgyban tanultakra hagyatkoztam, az algoritmus a tárgy jegyzetében[1] szereplő szélességi bejáráson alapul. A maximális párosítás megkereséséhez használt magyar módszer implementálását a Rónyai, Ivanyos, Szabó: Algoritmusok[2] könyvben leírtak alapján készítettem el, a gráf színezését a Lovász, Gács: Algoritmusok[3] könyvben leírt algoritmusváltozat alapján végeztem. A minimális költségű feszítőfát illetve feszítő erdőt kereső algoritmusok – Prim és Kruskal – ismertetésekor és az Unió-Holvan adatszerkezet implementálásakor az előbbi jegyzetből[1], a Rónyai, Ivanyos, Szabó: Algoritmusok[2] könyvből és a Cormen, Leiserson, Rivest, Stein: Új algoritmusok[4] könyvből merítettem.

2.2.1. PÁROSSÁG VIZSGÁLATA

Megvizsgálja, hogy egy adott irányítatlan, egyszerű gráf páros-e. Ha páros, akkor megadja a csúcsok egy lehetséges páros felosztását.

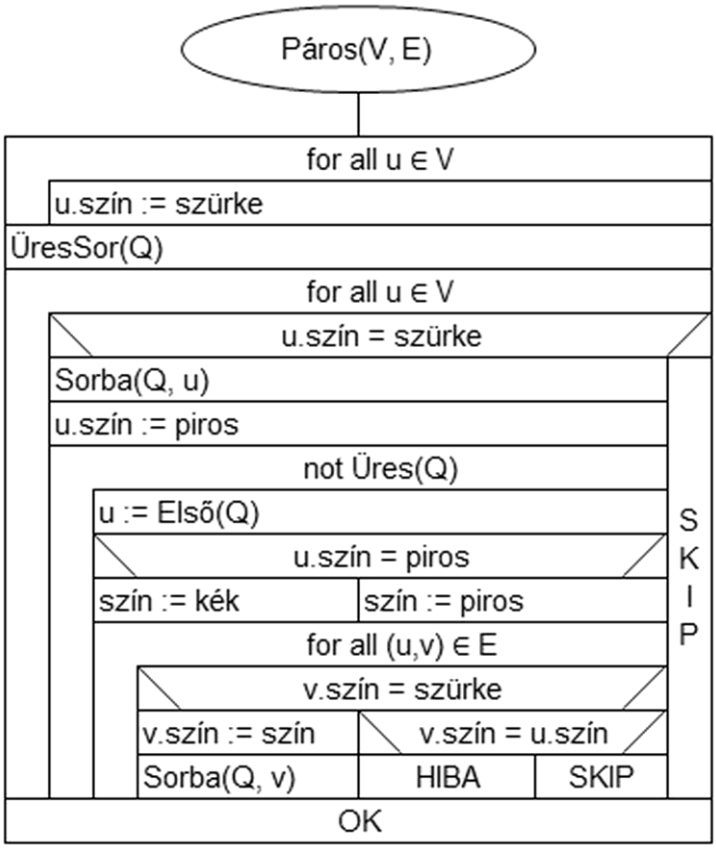


1. ábra

Az 1. ábrán az algoritmus eredményeképpen a csúcsok felosztása látható.

ALGORITMUS

Az algoritmus a szélességi bejárás nem összefüggő gráfokra vonatkozó változatán alapul, pirosra és kékre színezi a gráf csúcsait a páros felosztás mentén. Választ egy csúcsot a gráf egy még fel nem derített komponenséből, pirosra színezi, majd belőle, mint kezdőcsúcsból indulva bejárja az adott komponenst. A kiválasztott csúcs még felderítetlen szomszédjait kékre színezi, azok szomszédjait pirosra és így tovább. Ha a bejárás során egy olyan csúcshoz jut, aminek van azonos színű szomszédja, akkor a gráf nem lehet páros, az algoritmus lefutása sikertelen. Egyébként az algoritmus bejárja az összes komponenst, a gráf minden csúcsát pirosra vagy kékre színezi, és ez a színezés a csúcsok egy lehetséges páros felosztását adja.



2. ábra

A 2. ábrán látható algoritmus működése a következő lépésekre bontható:

- Inicializáláskor az algoritmus a gráf csúcsait szürkére színezi.
- A későbbi lépésekben ha van szürke csúcs – vagyis a gráfban van még fel nem derített komponens –, akkor kiválasztja a legkisebb sorszámú ilyen csúcsot, pirosra színezi, és elindít belőle egy szélességi bejárást – a gráf megfelelő komponensének felderítésére.
- A szélességi bejárás során minden lépésben a gráf aktuálisan elért csúcsának szomszédjait vizsgálja. Ha a szomszéd szürke – tehát a bejárás még nem érte el –, akkor kékre illetve pirosra színezi, aszerint, hogy az aktuális csúcs piros illetve kék volt. Ha a szomszéd színe megegyezik az aktuális csúcs színével, akkor az algoritmus sikertelen válasszal befejezi futását.
- Ha az algoritmus minden csúcsot beszínezett pirosra vagy kékre, akkor sikeres választ ad.

Az algoritmus alapját képező szélességi bejárás leírása és elemzése megtalálható az Algoritmusok és adatszerkezetek II. tárgy jegyzetében[1]. A megjelenítés során a még

meg nem látogatott csúcsok *szürke* színnel jelennek meg, így a programban is szürke a színük, ez felel meg a jegyzetben közölt algoritmusban a *fehér* színnek.

Adatszerkezet

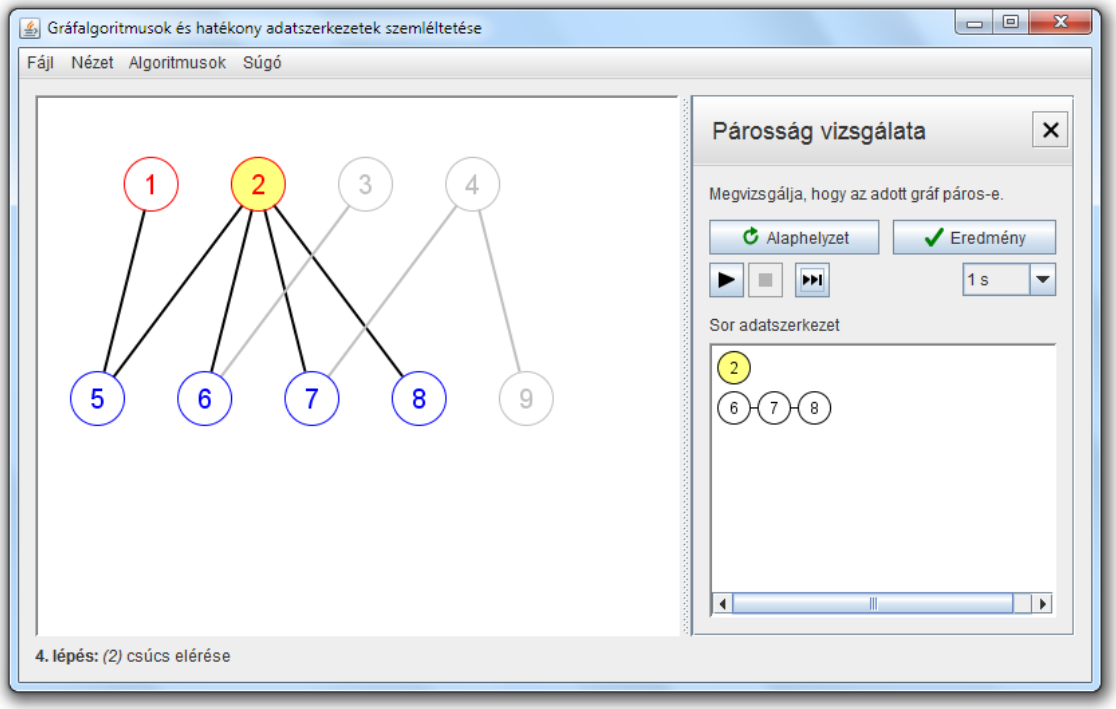
Az algoritmus központi adatszerkezete a szélességi bejárás során használt sor adatszerkezet. A sor a következő, konstans műveletigényű műveletekkel rendelkezik:

- *ÜresSor:* létrehoz egy új, üres sort.
- *Üres:* megvizsgálja, hogy az adott sor üres-e.
- *Sorba:* berak egy elemet a sor végére.
- *Első:* kiveszi a sor első elemét.

Műveletigény

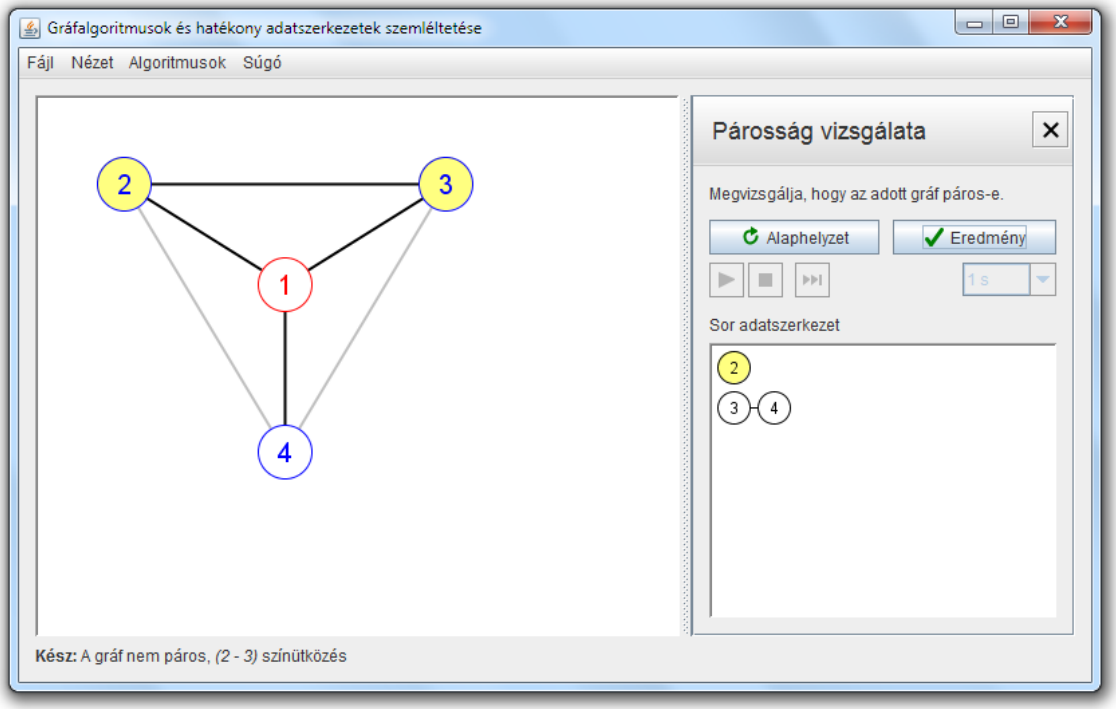
Az algoritmus műveletigénye megegyezik a szélességi bejárás műveletigényével: $O(n+e)$.

Szemléltetés



3. ábra

Az algoritmus megjelenítése során a hatékonyabb szemléltetés érdekében az aktuálisan elért csúcsot sárgával jelölöm, és jelzem, hogy a szélességi bejárás épp mely éleken halad, a 3. ábrán látható módon. Sikertelen befejezés esetén a sikertelenséget okozó, megegyező színű szomszédot is megjelölöm sárgával, a 4. ábrának megfelelően.



4. ábra

A sor szemléltetésekor ábrázolom a sorban lévő elemeket balról jobbra haladva, és a sor felett sárgával megjelölve megjelenítem a sorból legutoljára kivett elemet.

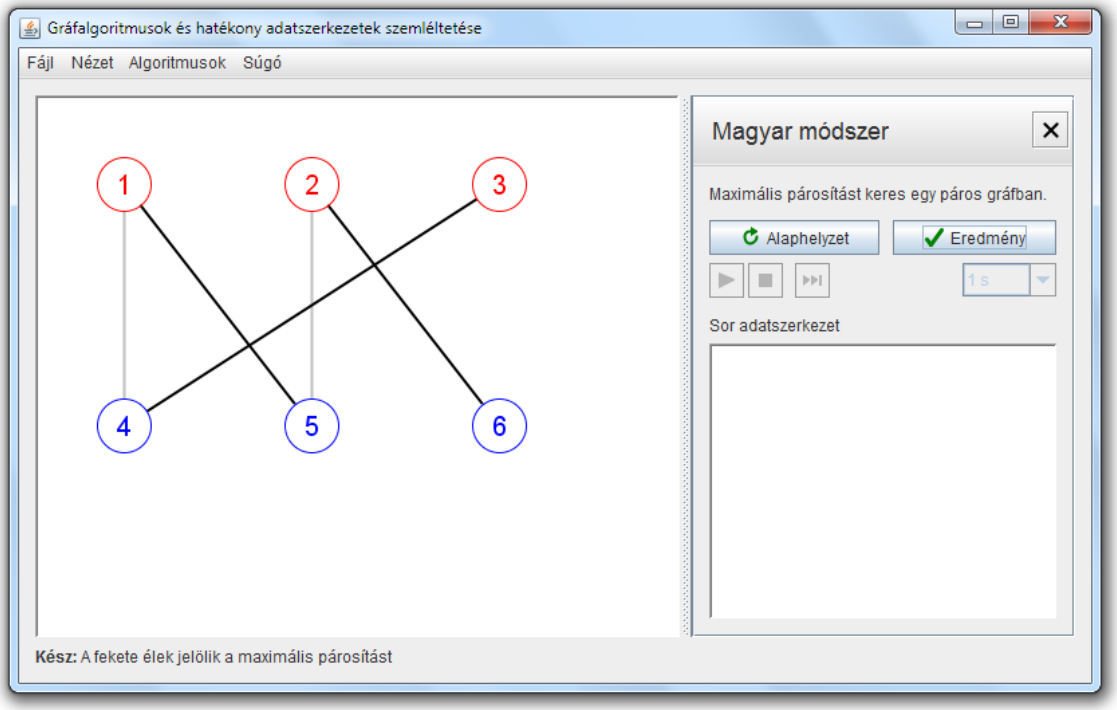
Lépések leírása

A szemléltetés során megjelenítem az aktuális lépés leírását az állapotsorban:

- „Csúcsok szürkére színezése” – inicializáláskor.
- „*(u)* csúcs elérése” – a szélességi bejárás lépései során, az *u* csúcs elérésekor.
- „*(u)* pirosra színezése, szélességi bejárás indítása” – új komponens felderítésekor, az *u* csúcsból történő szélességi bejárás indításakor.
- „A gráf páros” – az algoritmus sikeres lefutása után.
- „A gráf nem páros, *(u - v)* színütközés” – sikertelen lefutás esetén, a két ütköző csúcs megjelölésével.

2.2.2. MAGYAR MÓDSZER

Maximális párosítást keres egy irányítatlan, egyszerű páros gráfban.



5. ábra

A magyar módszer lefutásának eredményeként létrejött maximális párosítás az 5. ábrán látható.

ALGORITMUS

Az algoritmus a kiinduló gráf csúcsait pirosra és kékre színezi az előző pontban ismertetett, a gráf párosságát vizsgáló algoritmus segítségével. Ezután a gráf egy lehetséges maximális párosításba tartozó párosított éleit feketére színezi. Az algoritmus lefutása minden esetben sikeres, ha a gráf páros.

A magyar módszer alapötlete az, hogy tekinti a gráfban az alternáló utakat – olyan utakat, melyeknek élei felváltva tartalmaznak párosított (fekete) és nem párosított éleket. Ha egy alternáló út mindkét végpontja párosítatlan – nem indul belőlük párosított (fekete) él –, akkor javító út, segítségével a párosított élek száma eggyel növelhető. Ehhez az úton szereplő párosított élek helyett az út nem párosított éleit kell bevenni a pá-

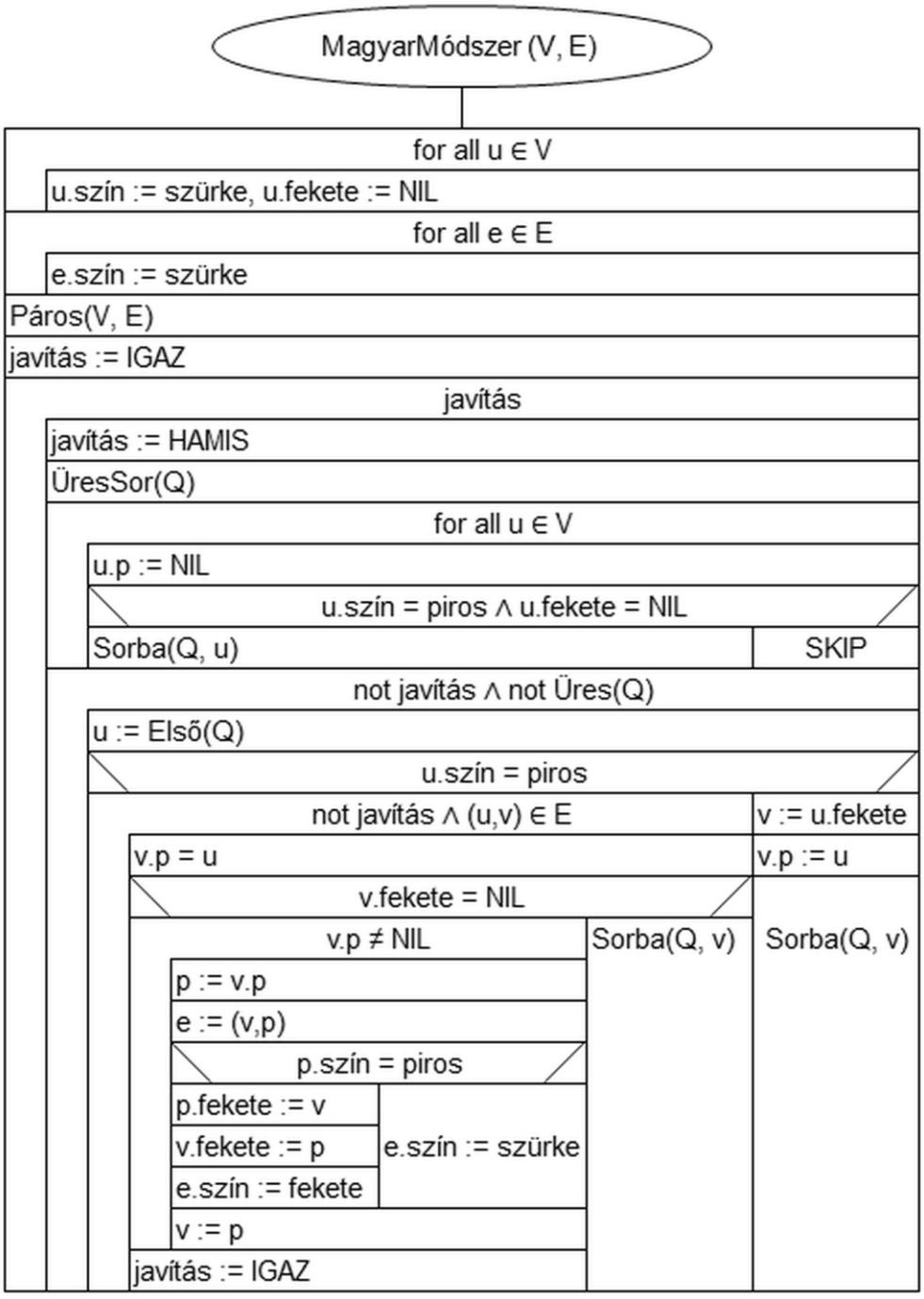
rosításba és feketére színezni. A javító út keresés elvégezhető akkor is, ha a párosítás üres – még a gráf egyik éle sem párosított –, és mindaddig működik, amíg a párosított élek nem adnak egy maximális párosítást. A fentieknek megfelelően a magyar módszer algoritmusa a következőképpen dolgozik: kiindul az üres párosításból, és amíg tudja, javító utak segítségével növeli az aktuális párosítást.

A bemutatott változatban a javító út megtalálásához az algoritmus alternáló erdőt készít. Az erdő elkészítéséhez egy, a szélességi bejáráson alapuló módszert használ. Kezdetben a gráf piros párosítatlan pontjait teszi a sorba, és így indítja el a bejárást. A módszer a továbbiakban abban különbözik a szélességi bejárástól, hogy piros csúcsnak csak nem párosított éleken keresztül elérhető kék szomszédjait, kék csúcsnak csak a párosított élen keresztül elérhető piros szomszédját vizsgálja. Ha egy vizsgált kék szomszéd párosítatlan, akkor ez a szomszéd egy javító út végpontja. Ez esetben befejezi a bejárást és az úton visszafelé haladva növeli a párosítást. Ha a bejárás azért fejezi be a futását, mert nem tud több csúcsot elérni, akkor a gráf nem tartalmaz javító utat.

A 6. ábrán látható algoritmus működése a következő lépésekre bontható:

- Inicializáláskor az algoritmus a gráf csúcsait és éleit szürkére színezi.
- Első lépésként megvizsgálja, hogy a gráf páros-e.
- A bejárások indításakor feltölti a sort a gráf párosítatlan piros csúcsaival.
- A bejárás során, ha az aktuálisan elért csúcs piros, akkor azon, még el nem ért kék szomszédjait vizsgálja, melyekbe nem párosított élen keresztül jutott el. Ha a vizsgált szomszéd párosított, akkor berakja a sorba, viszont ha nem párosított, akkor ez a kék szomszéd egy javító út végpontja. Ekkor befejezi a bejárást, a javító lépés következik. Ha a bejárás során aktuálisan elért csúcs kék, akkor a belőle induló párosított él piros végpontját teszi a sorba.
- Javító út megtalálásakor visszafelé végighalad azon, szürke éleit feketére, fekete éleit szürkére színezi. A párosítás növelése után új bejárást indít.
- Ha a bejárás nem talál javító utat, akkor az algoritmus befejezi futását, a fekete élek jelölik a maximális párosítást.

A magyar módszer részletes leírása és elemzése megtalálható a Rónyai, Ivanyos, Szabó: Algoritmusok[2] könyvben, egy másik változat – a színezéses szemlélettel – a Lovász, Gács: Algoritmusok[3] könyvben.



6. ábra

Megjegyzések: Mivel a bejárás a piros csúcsokból indul, így a kék csúcsokat pontosan akkor nem érte még el, ha azokra az *u.p* érték üres, vagyis még nincs őket megelőző csúcs. A kék csúcsok csak akkor kerülnek be a sorba, ha párosítottak, így ha az aktuális csúcs kék, akkor biztosan indul belőle párosított él. A bejárás gyorsítható, ha minden csúcshoz eltároljuk a belőle induló – létezése esetén egyedüli – párosított élt: *u.fekete*.

Adatszerkezet

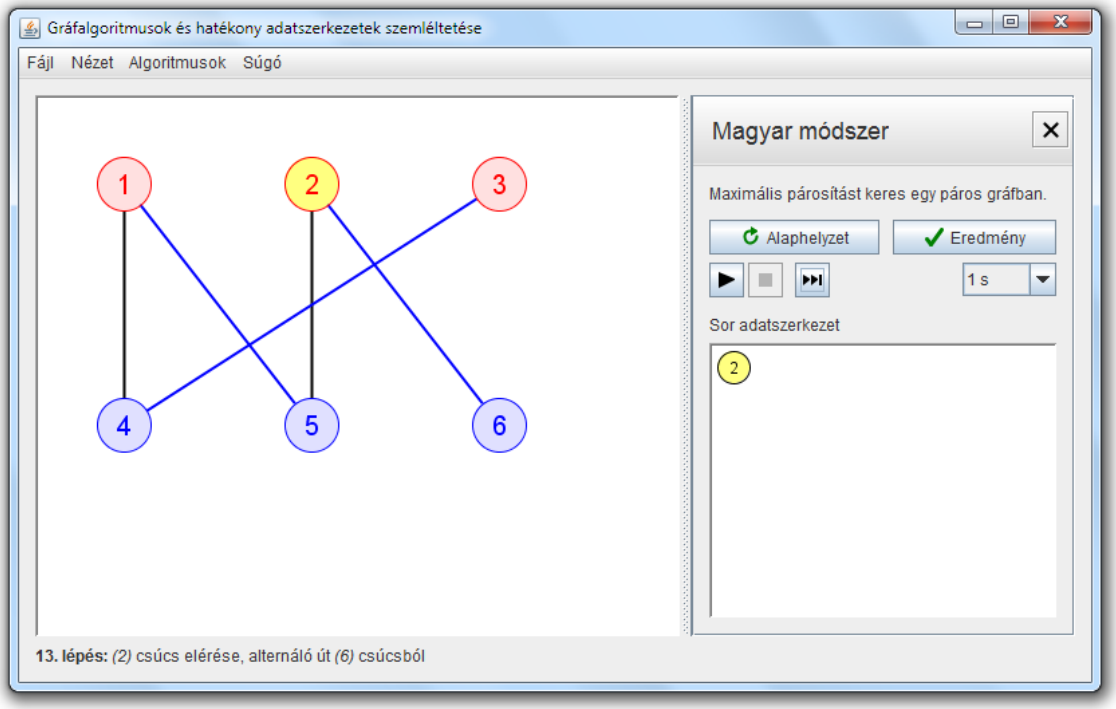
Az algoritmus központi adatszerkezete a bejárás során használt sor adatszerkezet. Műveletei és szemléltetése megegyezik a *Párosság vizsgálata* algoritmusnál leírtakkal.

Műveletigény

A párosság vizsgálatának műveletigénye $O(n+e)$. Az alternáló erdő készítésének műveletigénye megegyezik a szélességi bejárás műveletigényével, $O(n+e)$. A maximális párosítás legfeljebb $\left\lfloor \frac{n}{2} \right\rfloor$ élt tartalmazhat, emiatt az algoritmus is legfeljebb $\left\lfloor \frac{n}{2} \right\rfloor + 1$ alkalommal keres javító utat.

Ezek alapján az algoritmus teljes műveletigénye $O(n^2+ne)$.

Szemléltetés



7. ábra

Az algoritmus szemléltetésekor a párosság vizsgálatának lépéseit nem jelenítem meg, a teljes vizsgálat az algoritmus egy lépését teszi ki. A hatékonyabb szemléltetés érdekében az aktuálisan elért csúcsot sárgával, a bejárás során már elért csúcsokat kitöltéssel, az erdő éleit pedig kékkel jelölöm, a 7. ábrán látható módon.

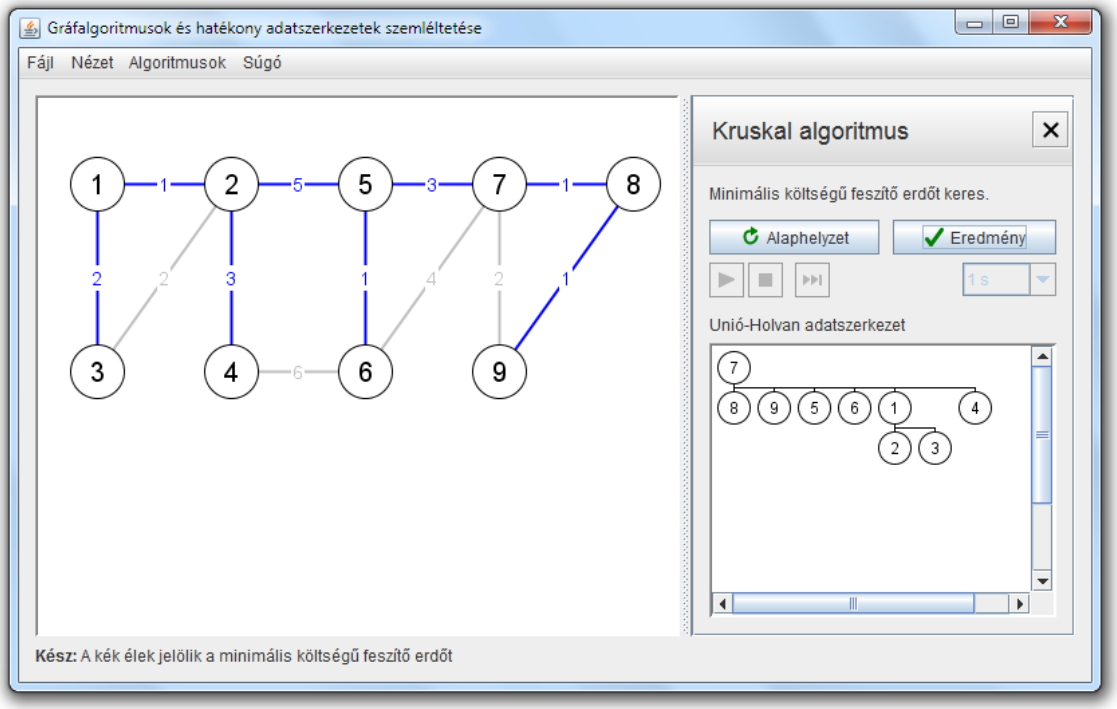
LÉPÉSEK LEÍRÁSA

A lépések leírásai az állapotsorban a szemléltetés során a következők:

- „Csúcsok és élek szürkére színezése” – inicializáláskor.
- „A gráf páros” – az első, a gráf párosságát vizsgáló lépés sikeressége esetén.
- „Sor feltöltése, alternáló erdő készítés indítása” – a bejárások indításakor.
- „*(u)* csúcs elérése” – a bejárások során, az *u* csúcs elérésekor.
- „*(u)* csúcs elérése, javító út *(v)* csúcsból” – a bejárások során, az *u* csúcs elérésekor, ha javító utat talált.
- „Javító út *(u - v)*, párosítás növelése” – javító úton történő végighaladáskor.
- „A fekete élek jelölik a maximális párosítást” – az algoritmus lefutása után.
- „A gráf nem páros” – sikertelen lefutás esetén.

2.2.3. KRUSKAL ALGORITMUS

Minimális költségű feszítőfát, nem összefüggő gráf esetén feszítő erdőt keres egy irányítatlan, egyszerű, élsúlyozott gráfban.



8. ábra

A 8. ábrán a Kruskal algoritmus által megtalált minimális feszítőfa látható.

PIROS-KÉK ELJÁRÁS

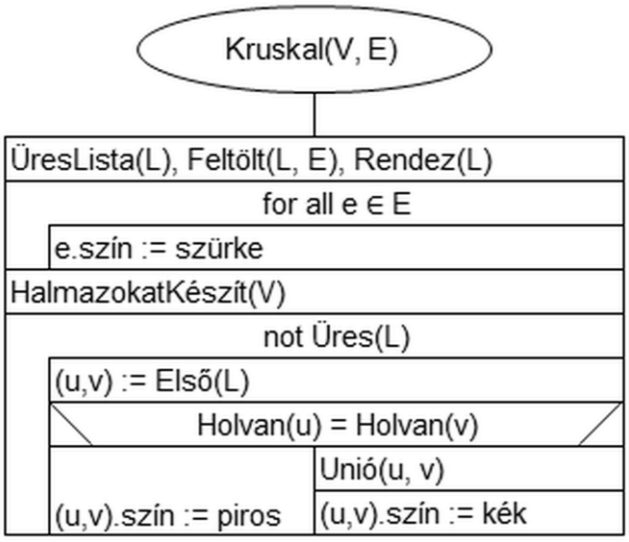
Az általános, nem determinisztikus, minimális költségű feszítőfát illetve feszítő erdőt kereső piros-kék eljárás egy kezdetben csak szürke élt tartalmazó gráfból indul ki, a kék szabály mentén kékre színezi a feszítő erdőbe bekerülő éleket és pirosra színezi az abba már biztosan be nem kerülő éleket. A két szabály a következő:

- *Kék szabály:* tekint egy nem üres X csúcshalmazt, amiből nem vezet ki kék él. A legkisebb súlyú X-ből kivezető szürke élt kékre színezi.
- *Piros szabály:* választ a gráfban egy olyan egyszerű kört, amiben nincs piros él. A kör legnagyobb súlyú szürke élét pirosra színezi.

ALGORITMUS

A Kruskal algoritmus a piros-kék eljárás egy mohó színezési stratégián alapuló speciális esete. Az algoritmus *n* db kék fából indul ki, a gráf minden csúcsát különálló kék fának, a gráf éleit szürkének tekinti. Minden lépésben veszi a gráf legkisebb súlyú, még szürke élét. Ha a kiválasztott él két végpontja különböző fában van, akkor kékre, különben pirosra színezi. Az algoritmus lefutása minden esetben sikeres, a kék színű élek jelölik a gráf egy lehetséges minimális feszítő erdőjének éleit.

A bemutatott változatban a legkisebb súlyú, még szürke él megkeresése az élek élsúly alapján történő nagyság szerinti rendezésével, a rendezett listán történő végighaladással valósul meg. A gráf csúcsainak kék fákra történő bontása diszjunkt halmaz műveletek, az Unió-Holvan adatszerkezet segítségével történik.



9. ábra

A 9. ábrán látható algoritmus lépései a következők:

- Inicializáláskor szürkére színezi és az élsúly alapján nagyság szerint sorba rendezi az éleket, és elkészíti a halmazokat.
- Minden lépésben veszi a nagyság szerint következő élt. Ha végpontjai különböző halmazokba esnek, akkor kékre színezi és egyesíti a halmazokat, különben pirosra színezi.

A piros-kék eljárás és a Kruskal algoritmus leírása és elemzése megtalálható az Algoritmusok és adatszerkezetek II. tárgy jegyzetében[1], az élek előre rendezésének ötlete

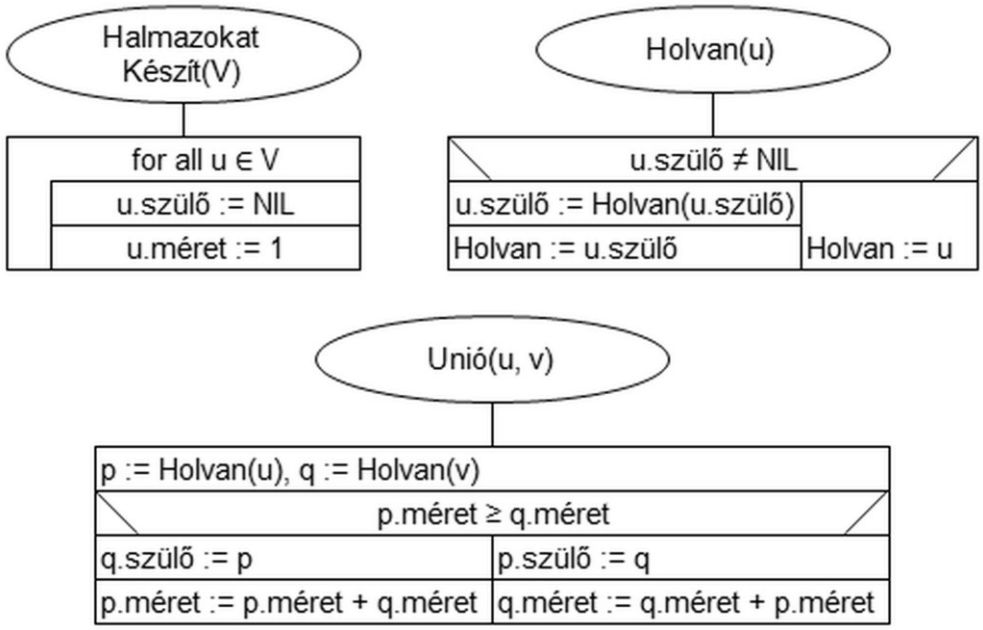
a Cormen, Leiserson, Rivest, Stein: Új algoritmusok[4] könyvben. Az Unió-Holvan adatszerkezet részletes leírása és elemzése a Rónyai, Ivanyos, Szabó: Algoritmusok[2] könyvben található.

ADATSZERKEZET

Az algoritmus központi adatszerkezete a diszjunkt halmazokat megvalósító Unió-Holvan adatszerkezet. Műveletei a következők:

- *HalmazokatKészít:* elkészíti az *n* db, egyelemű halmazt.
- *Holvan:* meghatározza, hogy egy elem melyik halmazban van.
- *Unió:* egyesíti a két elemhez tartozó halmazt.

A halmazok implementációja gyökeres, felfelé irányított fákkal történik. Egy fa csúcsaiban az adott halmaz elemeit tárolja, egy szülőmutatóval együtt. A gyökérelemek kitüntetett szerepet játszanak, ők címkézik az adott halmazt és tárolják a halmaz elemeinek számát. A műveletek megvalósítása a 10. ábrán látható. A Holvan művelet felfelé halad a fában, a gyökér csúcsot eredményül adva. Végrehajtásakor útösszenyomást hajt végre, a vizsgált elemtől a gyökérhez vezető út minden pontját közvetlenül a gyökér alá csatlakoztatja át. Az Unió művelet két halmazt egyesít, a nagyobb halmazhoz tartozó fa gyökeréhez gyerekként hozzákapcsolja a másikhoz tartozó fa gyökerét.



10. ábra

Az algoritmus egy absztrakt lista adatszerkezetet is használ az élek rendezésére és sorbavételére. A lista a következő műveletekkel rendelkezik:

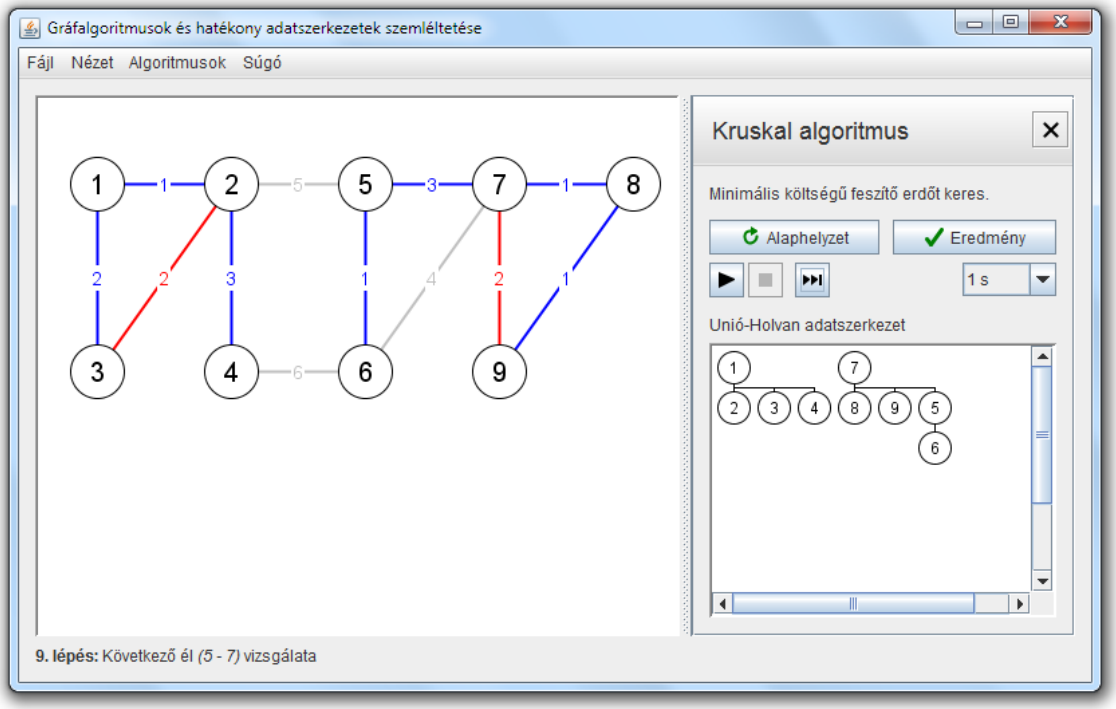
- *ÜresLista:* létrehoz egy új, üres listát.
- *Feltölt:* a listát feltölti az élekkel.
- *Rendez:* rendezi a listát az élsúlyok alapján.
- *Üres:* megvizsgálja, hogy az adott lista üres-e.
- *Első:* kiveszi a lista első elemét.

Műveletigény

A halmazok elkészítésének műveletigénye $\Theta(n)$, az élek szürkére színezése $\Theta(e)$, az élsúlyok szerinti rendezés $O(e \log e)$ alatt elvégezhető. Az élek sorbavétele az Unió-Holvan adatszerkezet ismertetett implementációjával $O(e\ \alpha(e))$ műveletigényű – ahol α az inverz Ackermann-függvény –, a Rónyai, Ivanyos, Szabó: Algoritmusok[2] könyvben leírtak alapján.

Az algoritmus teljes műveletigénye tehát $O(e \log e)$.

Szemléltetés



11. ábra

A 11. ábrán látható az algoritmus szemléltetése futás közben. Az algoritmus lefutása után csak a feszítő erdőt alkotó kék éleket jelenítem meg, a menetközben pirosra színezett élek végül újból szürke színt kapnak. Az élek rendezésére használt lista adatszerkezet nem kerül megjelenítésre.

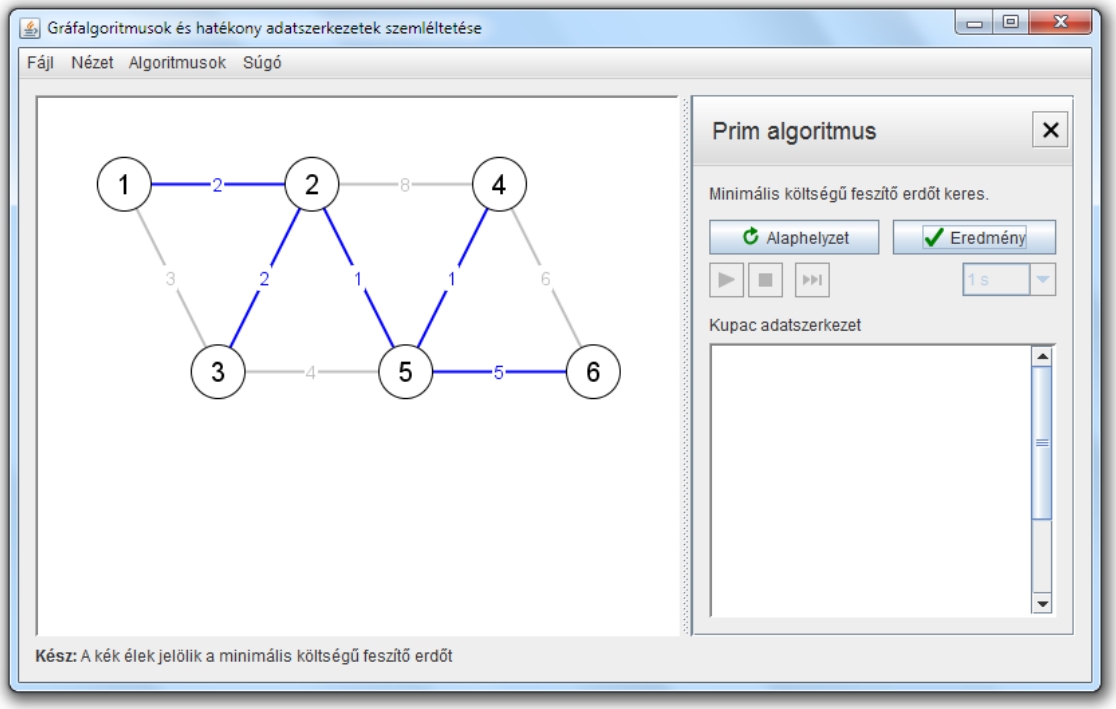
Lépések leírása

A lépések leírásai az állapotsorban a szemléltetés során a következők:

- „Élek szürkére színezése és nagyság szerinti sorrendbe rendezése, halmazok készítése” – inicializáláskor.
- „Következő él *(u - v)* vizsgálata” – az *(u,v)* él vizsgálatakor.
- „A kék élek jelölik a minimális költségű feszítő erdőt” – az algoritmus lefutása után.

2.2.4. PRIM ALGORITMUS

Minimális költségű feszítőfát, nem összefüggő gráf esetén feszítő erdőt keres egy irányítatlan, egyszerű, élsúlyozott gráfban.



12. ábra

A 12. ábrán a Prim algoritmus által megtalált minimális feszítőfa látható.

ALGORITMUS

A Prim algoritmus is a Kruskal algoritmus bevezetőjében leírt piros-kék eljárás egy mohó stratégián alapuló speciális esete. Kezdetben a gráf éleit szürkére színezve minden lépésben a kék szabályt alkalmazza egy üres erdőből kiindulva, amíg el nem készül a keresett feszítő erdő. A megfelelő, kékre színezhető él megtalálásához az algoritmus számon tartja az aktuális kék erdő csúcsait. Az egyes lépésekben az erdőhöz nem tartozó csúcsok közül kiválasztja azt, amelyiknek az erdőtől vett távolsága a legkisebb. A kiválasztott csúcsot hozzáveszi az erdőhöz, a hozzá vezető élt kékre színezi. Az algoritmus bemutatott változata nem összefüggő gráfokra is működik. Ha egy adott lépésben az aktuális kék erdő egyik fájából sem vezet ki él, akkor az algoritmus választ egy csú-

csot az erdőhöz még nem tartozó csúcsok közül és új, egyelemű faként hozzáveszi az erdőhöz. Az algoritmus lefutása minden esetben sikeres, a kék színű élek jelölik a gráf egy lehetséges minimális feszítő erdőjének éleit.

A bemutatott implementáció az erdő csúcsait fekete, az erdőhöz nem tartozó csúcsokat szürke színnel jelöli. A szürke csúcsoknál számon tartja az erdőtől vett távolságot és a csúcs erdőbeli megelőző csúcsát, amely értékeket minden lépésben aktualizál. A legkisebb távolságú szürke csúcs kiválasztása prioritásos sor, a kupac adatszerkezet segítségével történik, amelyben az elemek összehasonlításának alapja a csúcsok erdőtől vett távolsága.

13. ábra

A 13. ábrán látható algoritmus lépései a következők:

- Inicializáláskor szürkére színezi a gráf csúcsait és éleit. A csúcsok távolságát végtelenre, megelőzőjüket üresre állítja, majd egy prioritásos sorba tölti őket.
- Minden lépésben veszi a prioritásos sor legkisebb elemét, és a hozzá tartozó csúcsot feketére színezi. Ha az aktuális csúcs rendelkezik erdőbeli megelőző csúccsal, akkor a tőle a csúcshoz vezető élt kékre színezi. Ezután minden eset-

ben frissíti az aktuális csúcs szürke szomszédjainak az erdőtől vett távolságát és megelőzőjét az aktuális csúcsból kivezető élek alapján.

Megjegyzés: az algoritmus leírásában ismertetettek alapján előfordulhat, hogy egy adott lépésben a prioritásos sor minden elemének távolságértéke végtelen, megelőzője üres. Ekkor az algoritmus a lépés során a legkisebb sorszámú, még a sorban lévő – szürke – csúcsot választja ki.

A Prim algoritmus eredeti változatának leírása és elemzése megtalálható az Algoritmusok és adatszerkezetek II. tárgy jegyzetében[1].

ADATSZERKEZET

Az algoritmus központi adatszerkezete a prioritásos sort megvalósító kupac adatszerkezet. A kupac a csúcsok távolsága (*u.d*) alapján rendezi az elemeket, azonos távolságértékű csúcsok esetén a kisebb sorszámút veszi előre. Műveletei a következők:

- *ÜresSor:* létrehoz egy új, üres kupacot.
- *Üres:* megvizsgálja, hogy az adott kupac üres-e.
- *Feltölt:* feltölti a kupacot a gráf csúcsai alapján.
- *KiveszMin:* kiveszi a kupac legkisebb elemét.
- *Helyreállít:* helyreállítja a kupacot az adott elem megváltozása után.

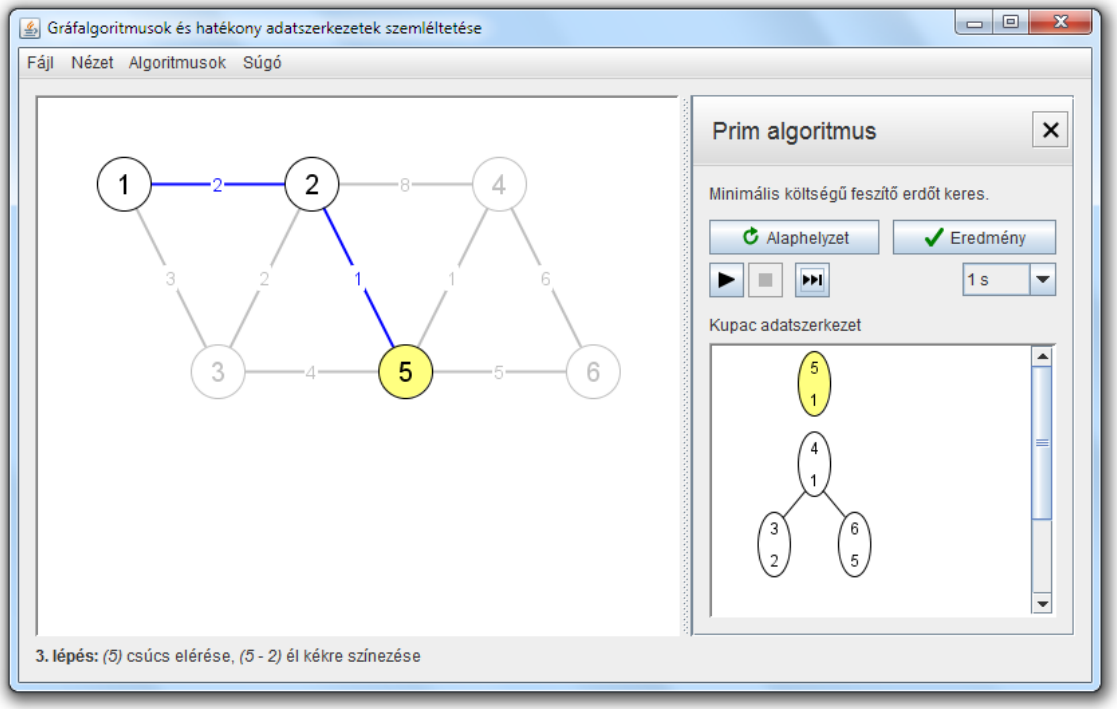
Az ÜresSor és Üres műveletek konstans, a Feltölt művelet lineáris, a KiveszMin és Helyreállít műveletek logaritmikus műveletigényűek.

MŰVELETIGÉNY

Az élek szürkére színezésének műveletigénye $\Theta(e)$, a csúcsok szürkére színezése, beállítása és a kupac feltöltése $\Theta(n)$ műveletigényű. Az erdőt bővítő lépések minden csúcsot kivesznek a kupacból és feketére színeznek. Minden élen végighaladnak, igény szerint aktualizálják az él végpontjának adatait, és helyreállítják a kupacot. A kupac műveleteinek műveletigénye alapján a kupac teljes kiürítése és a csúcsok színezése $O(n \log n)$, az összes aktualizálás és helyreállítás $O(e \log n)$ műveletigényű.

Tehát az algoritmus teljes műveletigénye $O((n+e) \log n)$.

Szemléltetés



14. ábra

Az algoritmus megjelenítése során a hatékonyabb szemléltetés érdekében az aktuálisan vizsgált csúcsot sárgával jelölöm, a 14. ábrán látható módon.

A kupac szemléltetésekor ábrázolom a kupacban lévő elemeket a fastruktúrának megfelelően, és a fa felett sárgával megjelölve megjelenítem a kupacból legutoljára kivett elemet. Az elemeken feltüntetem a hozzátartozó csúcs sorszámát és távolságértékét.

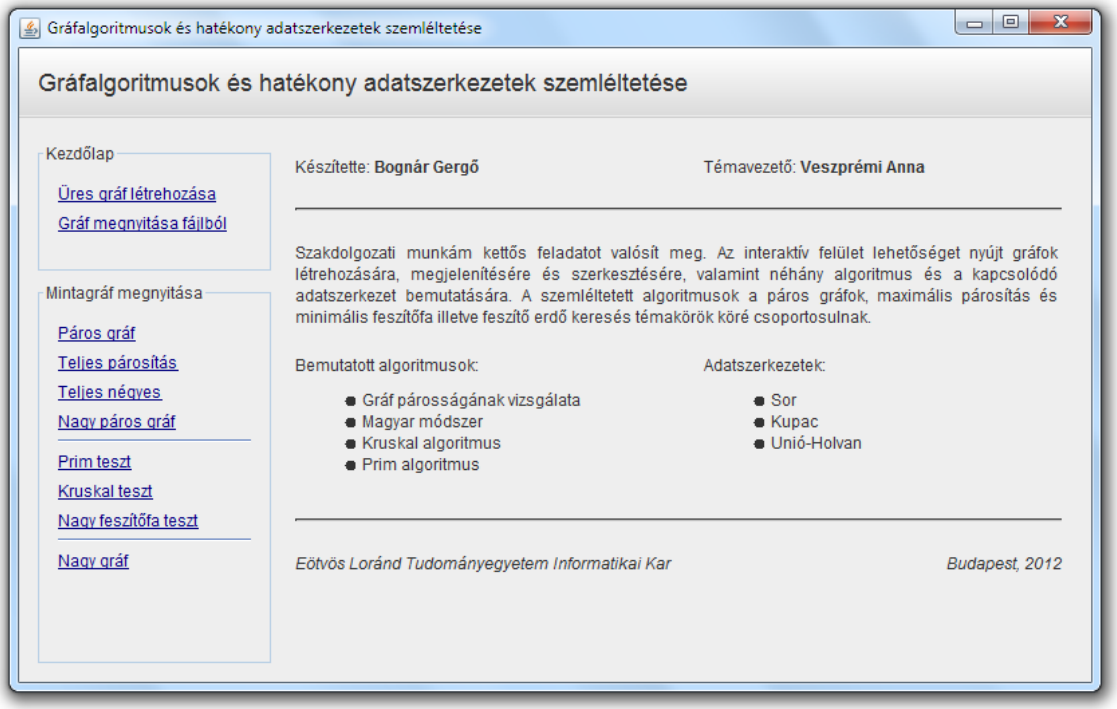
Lépések leírása

A lépések leírásai az állapotsorban a szemléltetés során a következők:

- „Csúcsok és élek szürkére színezése, kupac feltöltése” – inicializáláskor.
- „*(u)* csúcs elérése” – az *u* csúcs elérésekor, ha az nem rendelkezik erdőbeli megelőző csúccsal.
- „*(u)* csúcs elérése, *(u - v)* él kékre színezése” - az *u* csúcs elérésekor, ha az rendelkezik erdőbeli megelőző csúccsal (*v*).
- „A kék élek jelölik a minimális költségű feszítő erdőt” – az algoritmus lefutása után.

2.3. FELHASZNÁLÓI FELÜLET

Ebben a részben a program által nyújtott interaktív, grafikus felhasználói felületet ismertetem. A felület indításkor egy kezdőlapot jelenít meg, majd a kezdő lépések után két üzemmódban képes működni: gráf szerkesztés és algoritmus szemléltetés módban, ezek működésére részletesen kitérek.



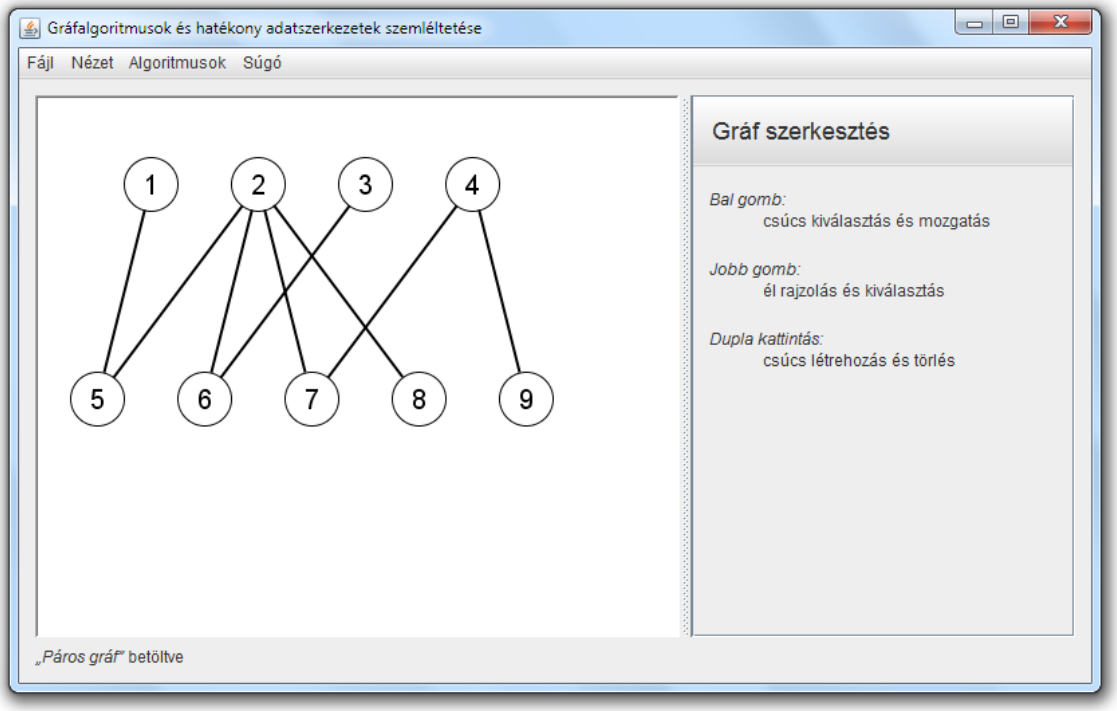
15. ábra

A 15. ábrán a program kezdőlapja látható. A kezdőlap középen egy összefoglalást tartalmaz a program nyújtotta lehetőségekről, bal oldalt a kezdő műveletek listáját. A kezdő műveletek között szerepel üres gráf létrehozása, gráf megnyitása fájlból és mintagráf betöltése.

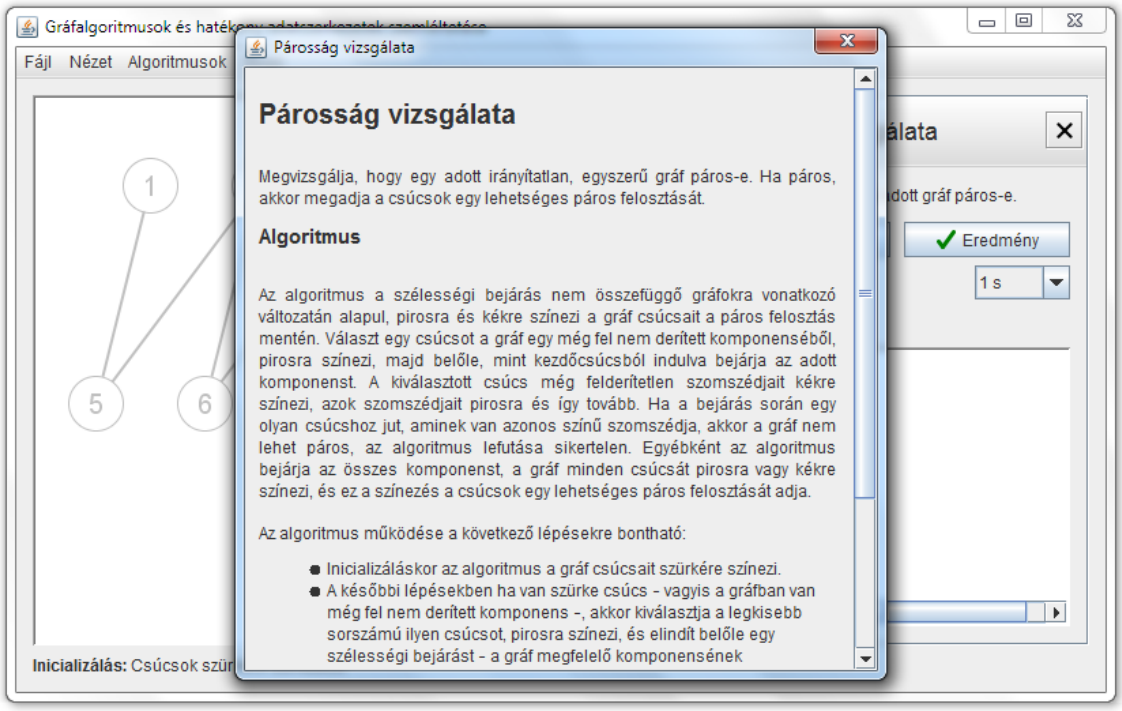
A 16. ábrán a program főablaka látható, egy mintagráf megnyitása után. Az ablak 4 részből áll: egy menüsorból, az ablak alján egy állapotsorból, közepén bal oldalt egy gráf megjelenítőből, jobboldalt pedig egy szerkesztőablakból.

A menüsor tartalmazza a programból elérhető alapvető funkciókat. A gráf megjelenítőn keresztül lehet a gráfot szerkeszteni, az algoritmusok lépéseit megtekinteni. A szer-

kesztőablak többféle feladatot ellát: ezen keresztül lehet a csúcsok és élek adatait szerkeszteni, valamint az algoritmusokat futtatni. Az állapotsorban megjelenő üzeneteket a menüpontok és az üzemmódok leírásakor ismertetem.



16. ábra



17. ábra

A futó program opcionálisan megjeleníthet egy súgóablakot, amely egy algoritmus részletes leírását tartalmazhatja, a 17. ábrán látható módon.

2.3.1. Kezdőlap műveletei

A kezdőlapon szereplő kezdő műveletek kiválasztásakor a program létrehoz vagy betölt egy gráfot, és gráf szerkesztés módba vált. A műveletek a menüsor *Fájl* menüjének egyes menüpontjaihoz kapcsolódnak:

- Üres gráf létrehozása – Új menüpont.
- Gráf megnyitása fájlból – Megnyitás menüpont.
- Mintagráfok – Mintagráf megnyitása menü pontjai.

2.3.2. Menüsor

A menüsor 4 menüt tartalmaz, mindegyik több menüponttal rendelkezik:

- *Fájl:* gráf létrehozással, tárolással, fájlműveletekkel kapcsolatos menüpontok.
- *Nézet:* a gráf megjelenítésével kapcsolatos menüpontok.
- *Algoritmusok:* az egyes algoritmusok szemléltetését indító menüpontok.
- *Súgó:* információk a programról, leírások az algoritmusokról.

Fájl menü

A fájl menü a következő menüpontokat tartalmazza:

- *Új:* létrehoz egy új, üres gráfot, és gráf szerkesztés módba vált.
- *Megnyitás:* megjelenít egy fájl megnyitás párbeszédablakot, a kiválasztott fájlban tárolt gráfot betölti, megjeleníti és gráf szerkesztés módba vált.
- *Mintagráf megnyitása:* ez alatt a menüpont alatt néhány előre elkészített, a tesztelés során használt mintagráfot lehet megnyitni.
- *Mentés:* megjelenít egy fájl mentés párbeszédablakot, majd a gráfot a kiválasztott néven menti.

- *Gráf mentése képként:* megjelenít egy kép mentés párbeszédablakot, majd a gráf megjelenítőn aktuálisan látható gráfot képként menti a kiválasztott néven. Mentéskor PNG, GIF és JPEG formátumban lehet menteni a képet.
- *Adatszerkezet mentése képként:* az előző menüponthoz hasonlóan, algoritmus szemléltetés során lehetőség van az aktuálisan látható adatszerkezet képként történő mentésére.

A fájl műveletek végrehajtása során az állapotsorban üzenetek jelennek meg azok sikerességét vagy sikertelenségét jelezve. Az üzenetekben *NÉV* helyén a fájl neve, *HIBA* helyén a hibaüzenet szerepel.

- *„NÉV”* betöltve
- *Hiba: „NÉV”* betöltése sikertelen: *HIBA*
- *„NÉV”* elmentve
- *Hiba: „NÉV”* mentése sikertelen: *HIBA*

NÉZET MENÜ

A nézet menü menüpontjai a következők:

- *Élsúlyok megjelenítése:* a menüpont segítségével meg lehet adni, hogy a gráf élsúlyai megjelenjenek-e. A menüpont csak gráf szerkesztés módban érhető el, algoritmus szemléltetés módban az élsúlyok megjelenítését az határozza meg, hogy az algoritmus foglalkozik-e az élsúlyokkal. A menüpont beállítása új gráf létrehozásakor és gráf fájlból való betöltésekor nem változik, de mintagráf betöltésekor módosul a mintagráfnak megfelelően.
- *Rácspontok megjelenítése:* rácspontokat jelenít meg a gráf megjelenítő hátterében. A rácspontok távolsága a gráf csúcsainak kirajzolásakor használt körök sugarával egyenlő.
- *Rácshoz igazítás:* a gráf csúcsainak mozgatásakor az előző menüpontban ismertetett rácspontokhoz igazítja a csúcsok koordinátáit. Bekapcsolása esetén automatikusan megjelennek a rácspontok.

ALGORITMUSOK MENÜ

Az algoritmusok menü a következő menüpontokat tartalmazza:

- *Alaphelyzet:* kilép algoritmus szemléltetés módból, visszatér gráf szerkesztés módba.
- *Párosság vizsgálata, Magyar módszer, Kruskal algoritmus, Prim algoritmus:* a négy algoritmus menüpontjával az algoritmusok szemléltetése indítható.

Ha az algoritmus szemléltetésének elindítása valamilyen belső hibába ütközik, akkor egy üzenet jelenik meg az állapotsorban. Az üzenetben *HIBA* a hiba okát jelöli.

- *Hiba:* Az algoritmus betöltése sikertelen: *HIBA*

SÚGÓ

A súgó menü menüpontjai a következők:

- *Névjegy:* egy párbeszédablakot jelenít meg a szerző, az egyetem és kar, valamint a készítés évének megjelenítésével.
- *Párosság vizsgálata, Magyar módszer, Kruskal algoritmus, Prim algoritmus:* egy súgóablakot jelenít meg az egyes algoritmusok részletes leírásával.

2.3.3. GRÁF SZERKESZTÉS

Gráf szerkesztés módban csúcsok és élek hozzáadására és törlésére, azok adatainak módosítására van lehetőség. Az üzemmód a program alapmódjának tekinthető, a kezdőlap elhagyása után a program ebben a módban indít, a későbbiekben új gráf létrehozása, gráf betöltése és az algoritmus szemléltetés befejezése után vált vissza erre.

A gráf szerkesztés módhoz három megjelenés és szerkesztőablak tartozik, aszerint, hogy van-e csúcs illetve él kijelölve. Az üzemmódba váltáskor nincs se csúcs, se él kijelölve. A 16. ábrán látható módon ekkor a szerkesztőablakban egy rövid összefoglaló látható a lehetséges műveletekről. Csúcs és él kiválasztása esetén a szerkesztőablakban egy, az adott elem adatait szerkesztő, az elem törlését lehetővé tevő felület jelenik meg.

Gráf szerkesztés módban a *Nézet* menün keresztül megadható, hogy megjelenjenek-e az élsúlyok. A beállítás nem változtatja meg a már meglévő élsúlyokat, csupán azoknak a megjelenítő felületen történő feltüntetését szabályozza.

Egér műveletek

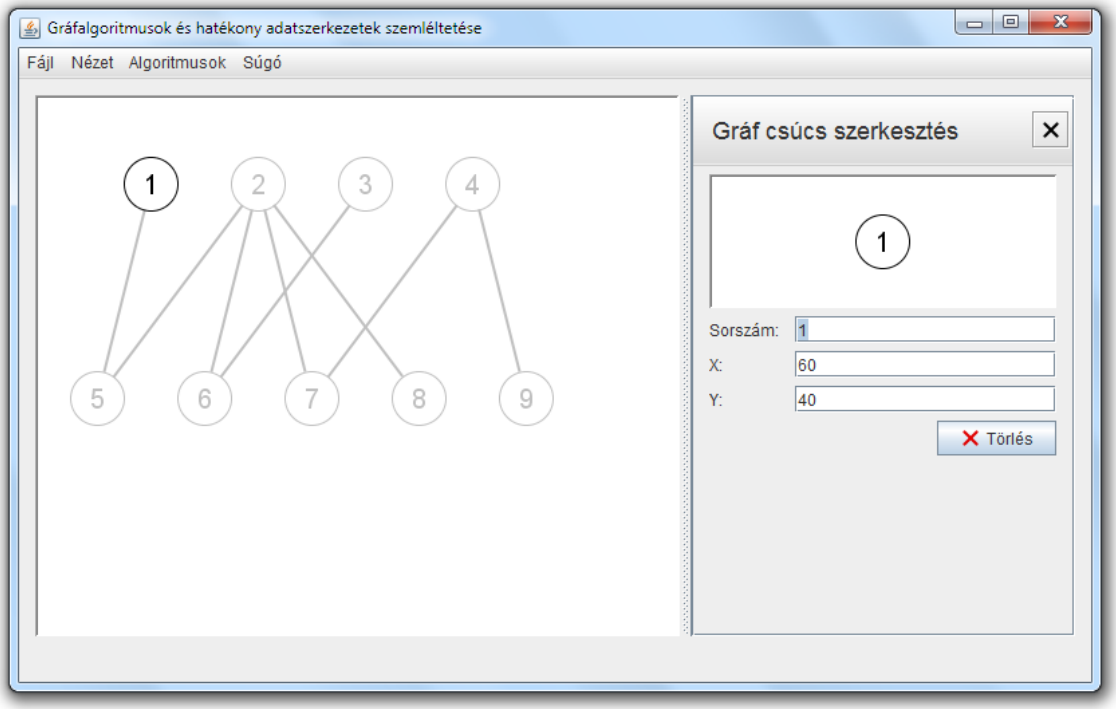
A csúcsok és élek hozzáadására és szerkesztésre kijelölésére a gráf megjelenítőn keresztül, az egér segítségével van lehetőség. Az egérrel végezhető műveletek a következőek:

- *Csúcs létrehozása:* bal egérgombbal történő dupla kattintással a háttéren. Az új csúcs létrehozáskor a következő, a legnagyobb sorszámú csúcs sorszámánál eggyel nagyobb sorszámot kapja meg, üres gráf esetén 1-et. A csúcs létrehozás után rögtön kijelölésre is kerül.
- *Csúcs törlés:* bal egérgombbal történő dupla kattintással az adott csúcson.
- *Csúcs kijelölés:* bal egérgombbal történő kattintás az adott csúcson.
- *Csúcs mozgatás:* a bal egérgomb nyomva tartása mellett a csúcs mozgatható az egér segítségével.
- *Él létrehozás, kijelölés:* jobb egérgombbal van lehetőség élt létrehozni vagy egy meglévő élt kijelölni. Ez a jobb egérgomb nyomva tartásával, rajzolással valósítható meg: a jobb gombot az él egyik végpontja fölött lenyomva, és nyomva tartás közben a másik csúcs fölé húzva. Rajzolás közben egy fekete vonal jelenik meg az él kezdőcsúcsa és az egér között. A sikeres rajzolás végén a két kiválasztott csúcs közötti él kerül kijelölésre, illetve előzőleg létrehozásra, ha még nincs köztük él. Utóbbi esetben az új él 1 súllyal jön létre.
- *Kijelölések megszüntetése:* a háttéren bal vagy jobb egérgombbal történő kattintás hatására az aktuális csúcs illetve él kijelölés megszűnik.

Csúcs mozgatás

A csúcsok mozgatás közben balra és felfelé csak a megjelenítő felület széléig mozgathatók. Jobbra és lefelé nincs ilyen megkötés, a gráf lehet nagyobb, mint a megjelenítő felület, a felületen kívül eső csúcsok görgetéssel láthatóvá tehetők.

A csúcsok mozgatásakor lehetőség van rácspontok megjelenítésére illetve a csúcsok rácspontokhoz történő igazítására, a *Nézet* menüben szereplő beállítások segítségével. Utóbbi beállításakor az adott csúcs mozgatás közben mindig az egérhez legközelebb eső rácsponthoz ugrik.



18. ábra

A 18. ábrán látható a program ablaka egy csúcs kijelölése után. A megjelenítő felület a többi csúcsot és az éleket elhalványítva, szürkére színezve jeleníti meg, a szerkesztőablakban a kijelölt csúcs adatai módosíthatók:

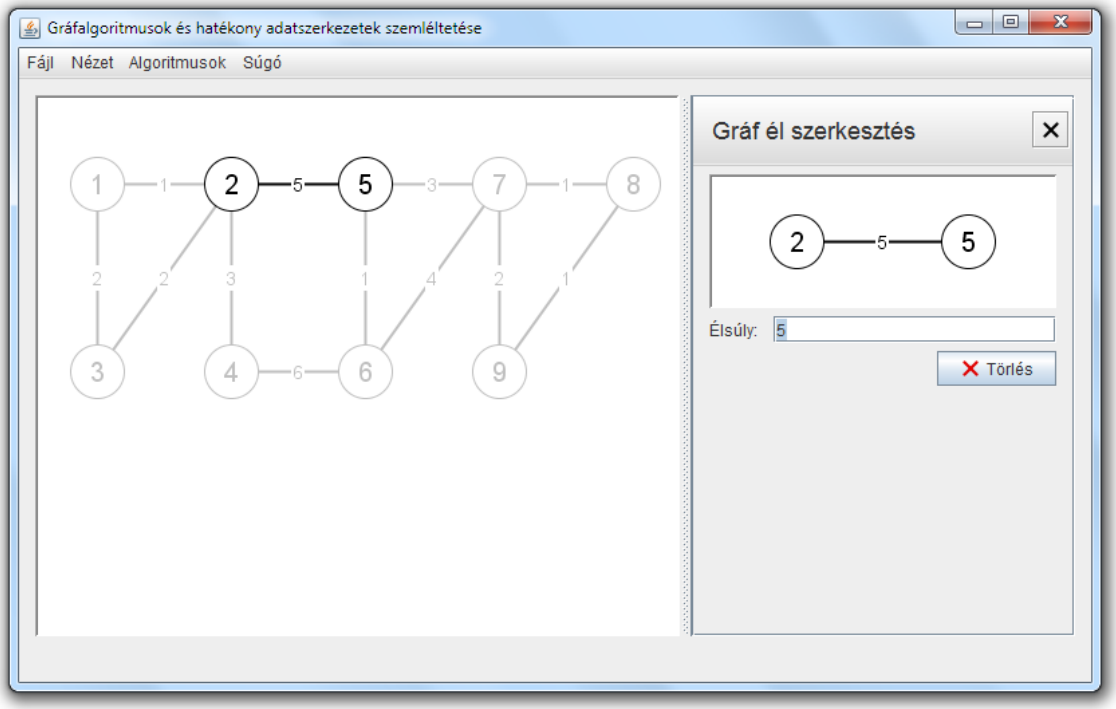
- *Sorszám:* a csúcs egyedi sorszáma, egész szám.
- *X, Y:* a csúcs koordinátái a megjelenítő felületen, egész számok. Módosításukkal azonnal megváltozik a csúcs helyzete a megjelenítő felületen. Negatív érték megadáskor 0-t kapnak értékül.

A szerkesztőkben a megadott értékek jóváhagyása az *ENTER* billentyűvel történik. Hibás érték megadása esetén üzenetek jelennek meg az állapotsorban, a hiba okát jelezve. Az üzenetekben *ÉRTÉK* a hibás értéket jelöli.

- *Hiba:* Érvénytelen sorszám: *„ÉRTÉK”*
- *Hiba:* Már szerepel ilyen sorszámú csúcs
- *Hiba:* Érvénytelen koordináta érték: *„ÉRTÉK”*

A szerkesztőablakban szereplő *Törlés* gomb segítségével el lehet távolítani a csúcsot és az összes, hozzá kapcsolódó élt a gráfból. A szerkesztőablak jobb felső sarkában lévő *X* gombbal a csúcs kijelölése megszüntethető.

ÉL SZERKESZTÉS



19. ábra

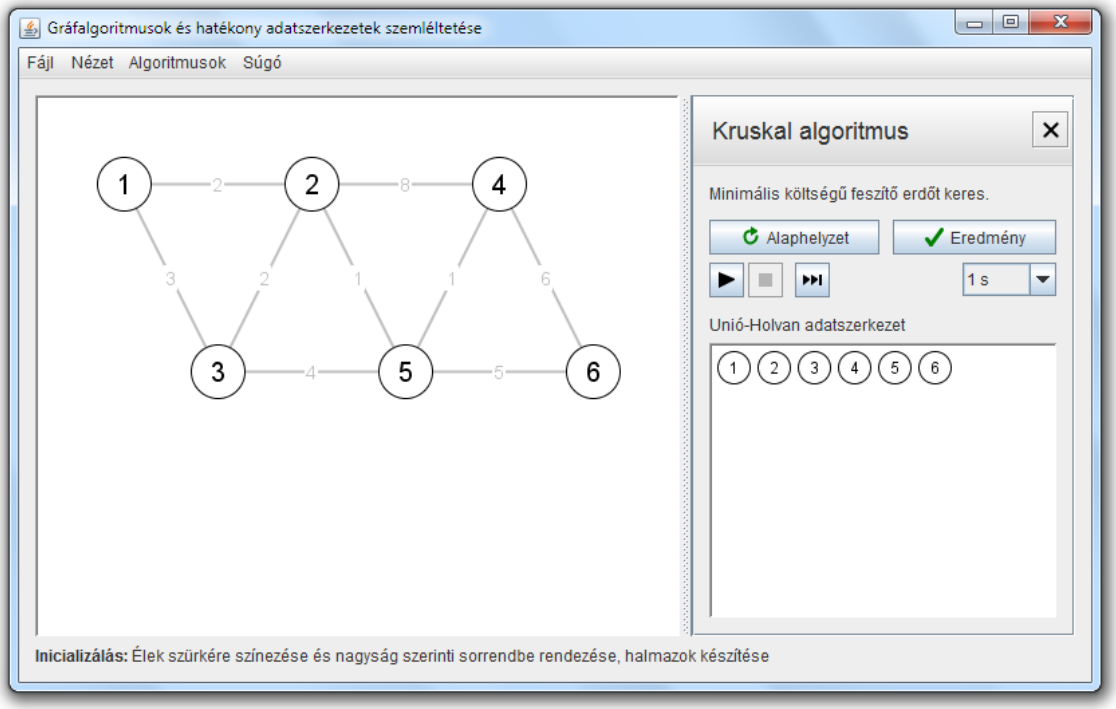
A 19. ábrán látható a program ablaka él kijelölés után. A megjelenítő felület a többi élt és az él végpontjai kivételével a csúcsokat elhalványítva, szürkére színezve jeleníti meg, a szerkesztőablakban a kijelölt él súlya módosítható, amely egész szám.

A súly szerkesztőben a megadott érték jóváhagyása az *ENTER* billentyűvel történik. Hibás súly megadása esetén egy üzenet jelenik meg az állapotsorban. Az üzenetben *ÉRTÉK* a hibás értéket jelöli.

- *Hiba:* Érvénytelen súly: *„ÉRTÉK”*

A szerkesztőablakban szereplő *Törlés* gomb segítségével el lehet távolítani az élt a gráfból. Az él eltávolítása nem törli annak két végpontját. A szerkesztőablak jobb felső sarkában lévő *X* gombbal az él kijelölése megszüntethető.

2.3.4. ALGORITMUS SZEMLÉLTETÉS



20. ábra

Az algoritmus szemléltetés mód az *Algoritmusok* menü egyik algoritmusának kiválasztásával indítható. Ebben a módban nincs lehetőség a gráf szerkesztésére, egyedül a csúcsok mozgatása lehetséges, az előző részben leírtak alapján. Algoritmus szemléltetés módban a szerkesztőablak a 20. ábrán látható módon egy algoritmus vezérlő felületet tartalmaz, az állapotsorban az algoritmus aktuális lépésének leírása látható.

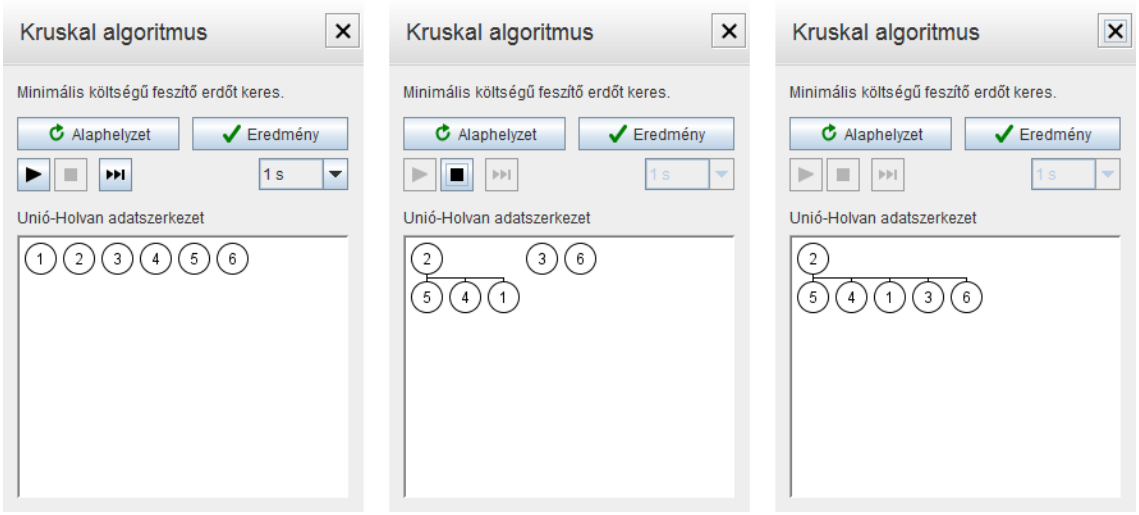
Algoritmus szemléltetés módban a megjelenítő felület az alapján jeleníti meg az élsúlyokat, hogy az algoritmus használja-e azokat. A szemléltetés során lehetőség van a gráf mentésére, valamint a gráf illetve az adatszerkezet képének mentésére. Ezen műveletek végrehajtása nem vált üzemmódot, de megváltoztatja az állapotsor értékét, az adott művelet sikerességével kapcsolatos üzenetet jelenítve meg.

VEZÉRLŐK

A szerkesztőablakban látható vezérlő felület tartalmazza a kiválasztott algoritmus nevét, rövid leírását, az algoritmusban történő navigáláshoz szükséges vezérlőket, az algoritmus központi adatszerkezetének nevét és megjelenítését.

A navigáló gombok segítségével lehetőség van az algoritmus kezdő- és végállapotának megtekintésére, az algoritmus lépésenkénti nyomon követésére, valamint időzített lejátszására. A navigáló vezérlők a következőek:

- *Alaphelyzet:* megszakítja az aktuális szemléltetést, az algoritmust annak kezdeti állapotába állítja, végrehajtva az algoritmus inicializáló lépését. Ez a lépés jellemzően a gráf csúcsainak vagy éleinek szürkére színezését, és az adatszerkezet feltöltését foglalja magába.
- *Eredmény:* megszakítja az aktuális szemléltetést, a háttérben lefuttatja az algoritmust és annak eredményét jeleníti meg.
- *Lejátszás:* az algoritmus lépéseinek időzített lejátszását indítja el, az algoritmus aktuális lépésétől kezdődően. A lejátszás során az egyes lépések animáltan jelennek meg, a lépések közötti időtartam a *Késleltetés* vezérlőn keresztül beállított érték.
- *Megállítás:* megállítja az időzített lejátszást. A szemléltetés a megállítás pontjától kezdődően léptetéssel vagy a lejátszás újbóli indításával folytatható.
- *Előre:* lépteti az algoritmust egy lépéssel, animáltan megjelenítve a lépést.
- *Késleltetés:* egy legördülő lista, melyen keresztül a lejátszás időzítését lehet beállítani, a lejátszás egyes lépései közötti várakozási idő formájában. A program indításakor értéke 1 másodperc.



21. ábra

Az egyes vezérlők elérhetősége függ az algoritmus aktuális állapotától, a 21. ábrán látható módon. Az alaphelyzet és eredmény gombok mindig elérhetőek. A többi vezérlő közül időzített lejátszás közben csak a megállítás gomb (középső kép), az algoritmus

indításakor és a befejezése előtti lépésekben a lejátszás, előre gombok és a késleltetés lista érhetők el (bal oldali kép). Az algoritmus befejeztével a vezérlők elérhetetlenné válnak (jobb oldali kép).

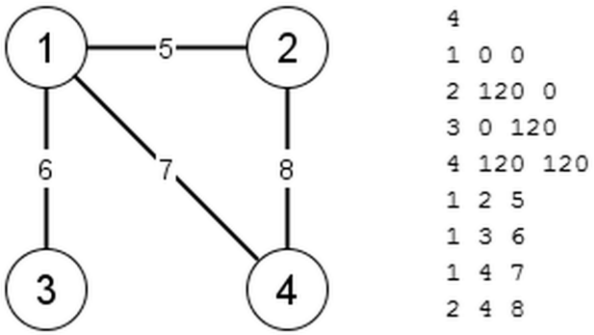
A szerkesztőablak jobb felső sarkában lévő *X* gombbal az algoritmus szemléltetés befejezhető, a program kilép algoritmus szemléltetés módból, visszatér gráf szerkesztés módba.

LÉPÉSEK LEÍRÁSA

Az algoritmus inicializálása után, az egyes lépések során az adott lépés leírása, az algoritmus végén annak eredménye látható az állapotsorban. Az algoritmusok lépésekre bontását és a lépésekhez tartozó leírásokat a *Bemutatott algoritmusok és adatszerkezetek* rész tartalmazza. Az egyes algoritmusok lépéseinek leírásánál *N* tartalmazza, hogy az aktuális lépés az algoritmus hányadik lépése, *LEÍRÁS* a lépéshez tartozó leírás:

- *Inicializálás:* LEÍRÁS
- *N. lépés:* LEÍRÁS
- *Kész:* LEÍRÁS

2.3.5. GRÁF FÁJLFORMÁTUM



22. ábra

A gráf fájlban történő tárolása egyszerű szöveges fájl formájában történik, amely a csúcsok és élek felsorolását tartalmazza. A fájl első sorában a csúcsok számának (*n*), egy nemnegatív egész számnak kell állnia. A következő *n* sorban a csúcsok adatai következnek: sorszám és két koordináta, mind egész számok, fehérszóközökkel elválaszt-

va. A fájl további sorai az éleket tartalmazzák: végpontok sorszáma és súly, mind egész számok, fehérszóközökkel elválasztva.

A 22. ábrán egy gráf, és a belőle mentett fájl mintája látható. Mentéskor a csúcsok és élek sorrendje sorszámuk szerint növekvő, de ez a rendezettség nem szükséges követelmény fájl betöltésekor. Egy él többszöri megadása sem okoz hibát betöltéskor, a később megadott súly felülírja a korábbit. A csúcsok koordinátái mentéskor nemnegatív egész számok, beolvasáskor a negatív koordináta értékek 0-ra cserélődnek le.

Megnyitáskor, ha a fájl beolvasása hibába ütközik a fájl helytelen formátuma miatt, akkor a sikertelenség oka kerül jelzésre. A jelzésekben *N* a sort, *ÉRTÉK* a helytelen értéket jelöli. A 3-5. jelzések a csúcsok, a 6-7. jelzések az élek hibáira utalnak.

- *A fájl üres*
- *N. sor, érvénytelen érték: „ÉRTÉK”*
- *N. sor, hiányzó csúcs definíció*
- *N. sor, formailag hibás csúcs definíció*
- *N. sor, már szerepel ilyen sorszámú csúcs*
- *N. sor, formailag hibás él definíció*
- *N. sor, hibás csúcs megadás*

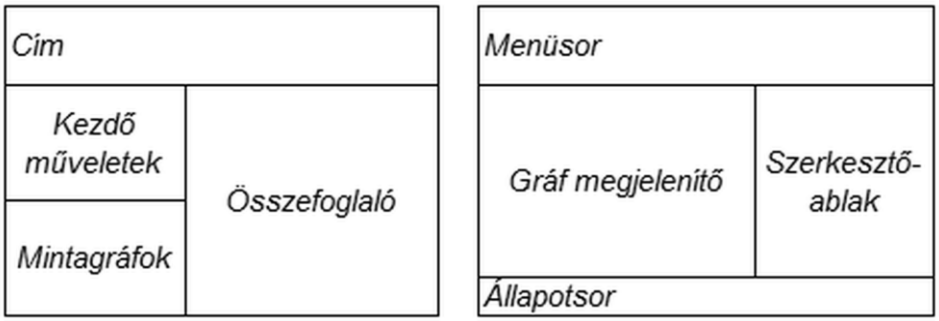
III. FEJLESZTŐI DOKUMENTÁCIÓ

3.1. RENDSZERTERV

Ebben a részben a program tervét, a tervezési szempontokat és a program egyes komponenseit ismertetem. A tervezéskor figyelembe vettem, hogy a programot objektumorientált és eseményvezérelt környezetben, Java nyelven, a Java grafikus könyvtárainak felhasználásával tervezem készíteni. A tervezett osztályokat több csomagra osztottam, működésük és egymáshoz kapcsolódásuk szerint. A felület kialakításánál fontos szempont volt, hogy átlátható, könnyen kezelhető legyen, egyszerű legyen vele a gráfok szerkesztése, és jól szemléltesse a bemutatott algoritmusokat.

3.1.1. FELÜLET

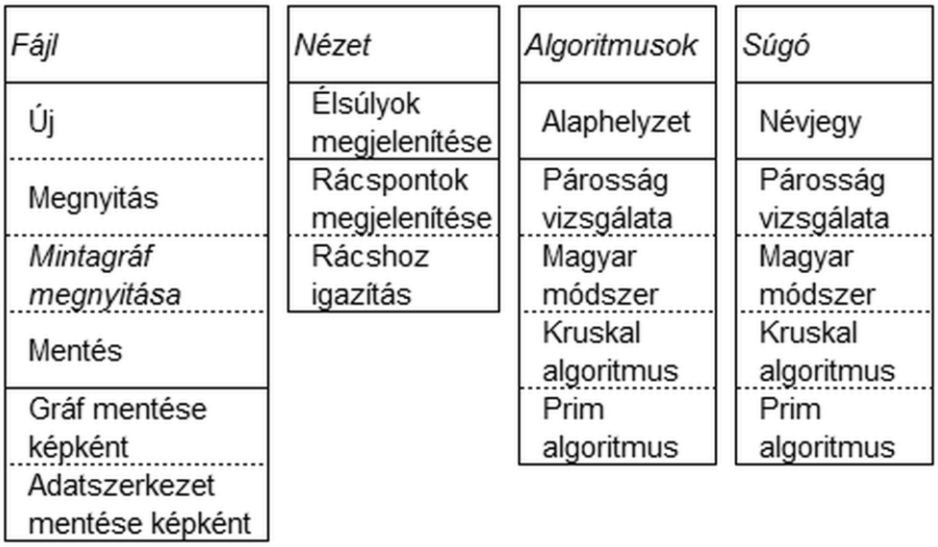
FŐABLAK



23. ábra

A 23. ábrán a program főablakának két terve látható: a kezdőlap és a kezdőlap utáni ablakterve. A kezdőlap középen egy összefoglalást tartalmaz a program nyújtotta lehetőségekről, bal oldalt a kezdő műveletek listáját. A kezdőlap után a főablak 4 részből áll: egy menüsorból, az ablak alján egy állapotsorból, közepén bal oldalt egy gráf megjelenítőből, jobboldalt pedig egy szerkesztőablakból.

MENÜ



24. ábra

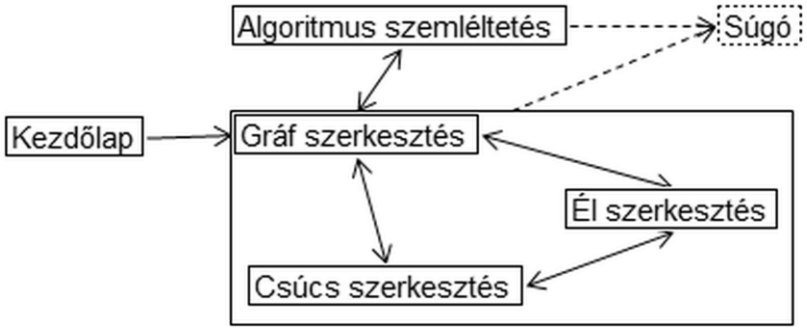
A menü terve az 24. ábrán látható.

SZERKESZTŐABLAKOK

25. ábra

A szerkesztőablak többféle szerkesztő és vezérlő felületet tartalmazhat a 25. ábrán látható módon: gráf szerkesztés összefoglaló, csúcs szerkesztő, él szerkesztő és algoritmus vezérlő. A csúcs és él szerkesztő megjeleníti a kijelölt csúcsot illetve élt, lehetővé teszi az elem adatainak szerkesztését és az elem törlését. Az algoritmus vezérlő tartalmazza a kiválasztott algoritmus nevét, rövid leírását, az algoritmusban történő navigáláshoz szükséges vezérlőket, az algoritmus központi adatszerkezetének nevét és megjelenítését. A szerkesztőablak a program üzemmódjai – gráf szerkesztés vagy algoritmus szemléltetés – alapján jeleníti meg a megfelelő szerkesztőt vagy vezérlőt.

KAPCSOLATOK



26. ábra

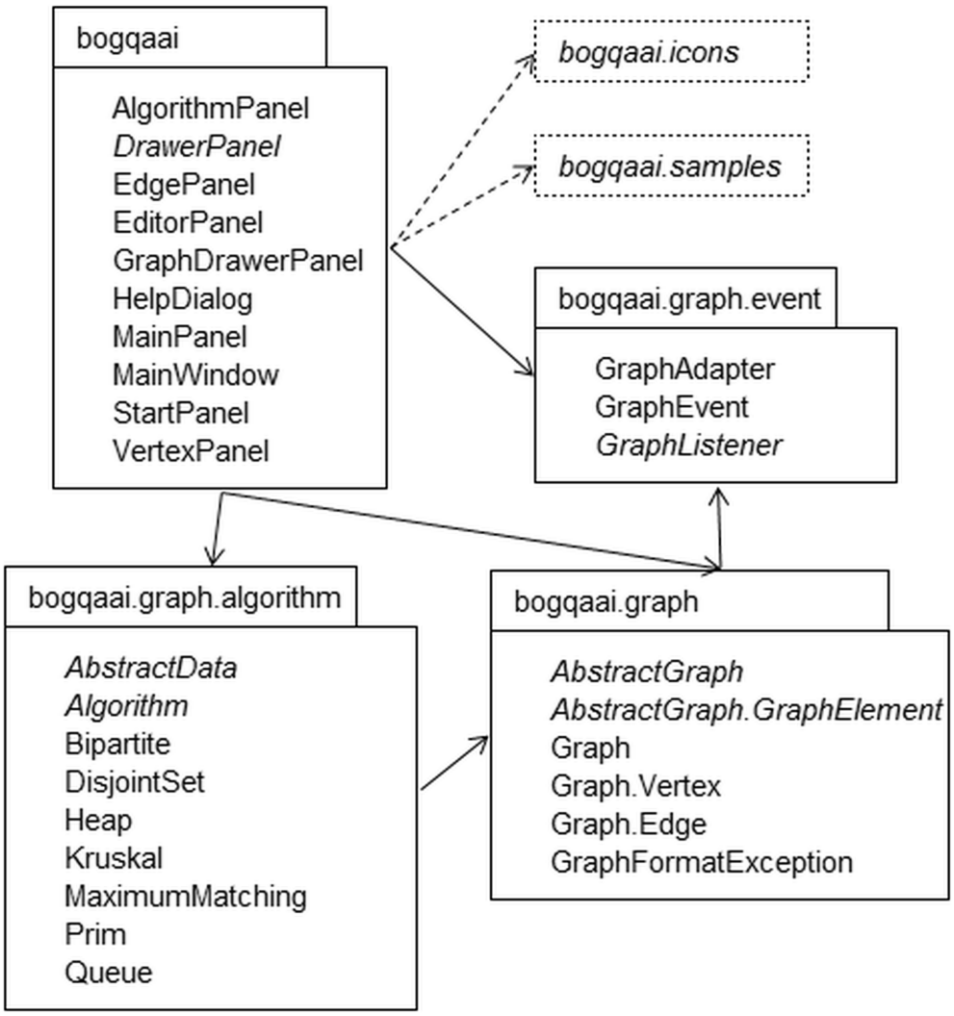
A 26. ábrán a program a tervezett felületei közötti kapcsolatok és átmenetek láthatók, a felületet megjelenítő üzemmóddal együtt.

3.1.2. ESEMÉNYKEZELÉS

A tervezett program grafikus felületű, ezért alapvetően eseményvezérelt felépítésű. A grafikus felület egyes elemei a felhasználói interakció során – egér műveletek, menüpont, gomb, más vezérlő kiválasztása, szerkesztő módosítása – különböző eseményeket generálnak. Az esemény kezelése lehet többszintű: a gráf módosítása – akár felhasználói esemény, akár belső hatásra – egy gráf megváltozási eseményt generál, amiről a gráf értesíti az őt tartalmazó felületeket.

A programtervet úgy alakítottam ki, hogy a program lényegében egy végrehajtási szállal dolgozzon: az eseményeket kezelő szállal. Az algoritmusok lépéseinek időzített lejátszására és animálására a Java könyvtár biztosította timert alkalmaztam. Ez a megoldás időzítés és animálás esetén megadott időközönként megadott eseményt generál, amelynek végrehajtása az eseményeket kezelő, központi szálon történik.

3.1.3. CSOMAGOK



27. ábra

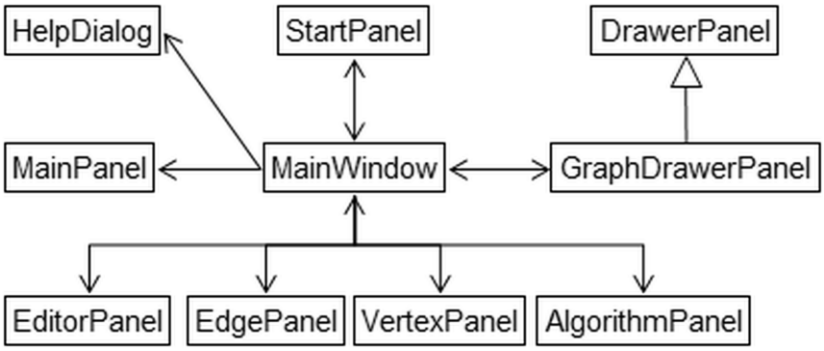
A 27. ábrán láthatók a csomagok és a köztük lévő kapcsolatok. A következő csomagokat alakítottam ki:

- *bogqaai:* a grafikus felület létrehozásáért felelős osztályok.
- *bogqaai.graph:* a gráf ábrázolást megvalósító osztályok.
- *bogqaai.graph.event:* a gráf és a grafikus felület közötti eseményvezérelt kommunikáció osztályai.
- *bogqaai.graph.algorithm:* a szemléltetett algoritmusok és adatszerkezetek osztályai.
- *bogqaai.icons:* erőforrás csomag, a grafikus felületen használt képek tárolására.

- *bogqaai.samples:* erőforrás csomag, a mintagráfok tárolására, szöveges fájlok formájában.

3.1.4. OSZTÁLYOK

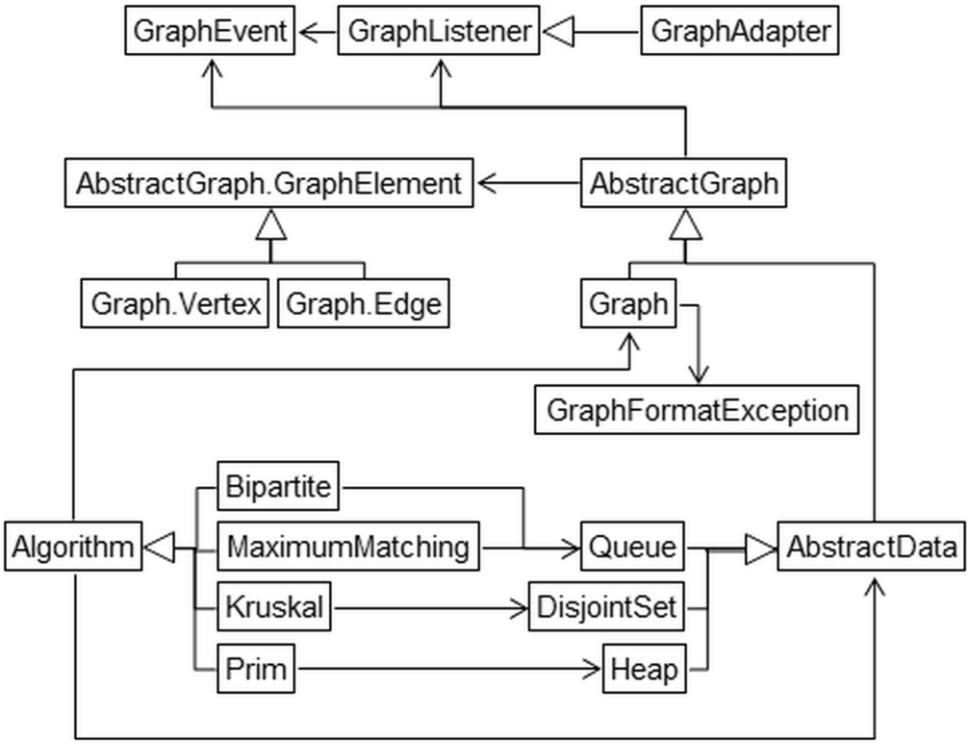
GRAFIKUS FELÜLET



28. ábra

A 28. ábrán a grafikus felület legfontosabb osztályai, és a köztük lévő kapcsolatok láthatók. A csomag rendelkezik egy kitüntetett osztállyal (*MainWindow*), amely a program főablakát – akár önálló ablakként, akár böngészőbe ágyazottan – valósítja meg, és koordinálja a felület egyes elemeit. Az osztályszerkezet kialakításakor a felület egyes komponenseit külön osztályokba soroltam:

- *MainWindow:* a főablakot megvalósító osztály.
- *StartPanel:* az indításkor megjelenő kezdőlap osztálya.
- *MainPanel:* a főablak váza a kezdőlap után. A váz tartalmazza a gráf megjelenítőt, a szerkesztőablakot és az állapotsort.
- *EditorPanel, VertexPanel, EdgePanel:* gráf szerkesztés módban a szerkesztőablakot kitöltő felületek – összefoglaló, csúcs és él szerkesztő.
- *AlgorithmPanel:* algoritmus szemléltetés módban a szerkesztőablakban megjelenő algoritmus vezérlő felület.
- *DrawerPanel:* a gráfot, annak részeit és az adatszerkezeteket kirajzoló felületek közös, absztrakt ősosztálya.
- *GraphDrawerPanel:* a gráf megjelenítőben lévő, gráfot kirajzoló felület.
- *HelpDialog:* a súgóablakot megvalósító osztály.



29. ábra

A 29. ábrán a gráf, algoritmusok és adatszerkezetek legfontosabb osztályai láthatók, a köztük lévő kapcsolatokkal együtt. A program elsődleges célja a gráfok grafikus szerkesztése és az algoritmusok szemléltetése. Ennek megfelelően az osztályszerkezet megtervezésekor felismertem, hogy a gráf és az adatszerkezetek a grafikus megjelenítés szempontjából hasonlóan viselkednek: csúcsokat és éleket képesek kirajzolni egy grafikus felületre. A gráf elemeinek – csúcsoknak és éleknek is – több közös vonásuk van a grafikus megjelenítés és animáció szempontjából. Ezeknek megfelelően létrehoztam egy absztrakt gráf ősosztályt (*AbstractGraph*) és egy absztrakt gráf elem ősosztályt (*AbstractGraph.GraphElement*). A tényleges gráfot megvalósító osztály (*Graph*) és az adatszerkezetek ősosztálya (*AbstractData*) az absztrakt gráf ősosztály, a gráf csúcsai és élei (*Graph.Vertex*, *Graph.Edge*) az absztrakt gráf elem ősosztály leszármazottjai. Az egyes adatszerkezetek az adatszerkezet ősosztály, az algoritmusok az algoritmus ősosztály (*Algorithm*) leszármazottai.

A grafikus felület és a gráf illetve adatszerkezet közötti kommunikáció eseményvezérelt megvalósítását szolgálja a *bogqaai.graph.event* csomag. A kommunikáció alapvetően egyirányú: a grafikus felület tudja közvetlenül elérni az általa létrehozott, a háttér-

ben működő objektumokat. A tervezés folyamán észrevettem, hogy a gráf megváltozása esetén a felület több, egymástól független elemét is értesíteni kell a gráf változásáról. Ennek megoldására egy eseményvezérelt rendszert hoztam létre: a felület elemei kérhetnek értesítést a gráftól, annak megváltozása esetén.

Az ábrán látható főbb osztályok:

- *AbstractGraph:* az absztrakt gráf ősosztály.
- *AbstractGraph.GraphElement:* az absztrakt gráf elem ősosztály.
- *Graph:* a tényleges gráf osztály.
- *Graph.Vertex, Graph.Edge:* a tényleges csúcs és él osztályok.
- *GraphFormatException:* hibás gráf fájlformátum esetén generált kivétel.
- *AbstractData:* az absztrakt adatszerkezet ősosztály.
- *Queue, Heap, DisjointSet:* az egyes adatszerkezetek osztályai.
- *Algorithm:* az absztrakt algoritmus ősosztály.
- *Bipartite, MaximumMatching, Kruskal, Prim:* az egyes algoritmusok osztályai.
- *GraphEvent:* a gráf megváltozásakor küldött esemény osztálya.
- *GraphListener, GraphAdapter:* a gráf megváltozásakor küldött eseményt fogadó interfész illetve ősosztály.

3.2. IMPLEMENTÁCIÓ

Ebben a részben a tényleges implementációval kapcsolatos tudnivalók, az elkészített osztályok leírásai szerepelnek. A tervezési lépésekhez hasonlóan az implementálás során is figyelembe vettem, hogy elsődlegesen szemléltető jellegű programról van szó. Ennek megfelelően hatékonysági szempontoknál mindig a megjelenítés hatékonyságát helyeztem előtérbe.

Az egyes osztályok leírásakor megadom az adattagok és metódusok listáját és szignatúráját. Az osztály publikus metódusait és az ősosztályok leszármazottaiban felülírásra kerülő belső metódusait részletesen is ismertetem. A belső használatú adattagok és változók magyarázatai a kód megjegyzéseiben megtalálhatók.

3.2.1. FEJLESZTŐI KÖRNYEZET

A programot Java nyelven írtam, NetBeans 7.1 fejlesztőkörnyezetben. A forrásfájlok fordítását JDK 1.7 segítségével végeztem, de a kód és a lefordított bájtkód is kompatibilis a JDK előző, 1.6-os változatával.

FORDÍTÁSI TUDNIVALÓK

Az implementálás során a csak a Java könyvtárat használtam, más külső forrásokat nem. A *StartPanel*, *EditorPanel*, *VertexPanel*, *EdgePanel* és *AlgorithmPanel* osztályok által megvalósított grafikus felületek a NetBeans felületszerkesztőjével készültek, a felületek adatait az osztályokhoz tartozó *form* fájlok tartalmazzák. A fordítás NetBeans-en keresztül történt, a lefordított bájtkódokból egy Java archívumot képezett (*graf.jar*), ami a tényleges programnak tekinthető. Az archívumot digitális aláírással láttam el, ez biztosítja, hogy appletként történő futtatás esetén is lehetőség legyen a gráf beolvasására, mentésére.

Az osztályok leírásakor hivatkozott osztályok a Java könyvtár részei, dokumentációjuk megtalálható a Java API Specification[5] alatt.

30. ábra

A projekt részét képező fájlok struktúrája az 30. ábrán látható. Az *src* könyvtár tartalmazza a tényleges forráskódokat, a többi mappa és fájl a NetBeans projekt beállításait tartalmazza.

3.2.2. GRAFIKUS FELÜLET CSOMAGJA

MAINWINDOW

MainWindow
- String title - String aboutText - String openSuccessMessage - String openFailMessage - String saveSuccessMessage - String saveFailMessage - String algorithmFailMessage - String sampleFormat - Graph graph - Graph.Vertex vertex - Graph.Edge edge - Algorithm algorithm - boolean start - java.awt.Component rootComponent - javax.swing.RootPaneContainer root - StartPanel startPanel - MainPanel mainPanel - GraphDrawerPanel graphPanel - EditorPanel editorPanel - VertexPanel vertexPanel - EdgePanel edgePanel - AlgorithmPanel algorithmPanel - javax.swing.JFileChooser fileChooser - javax.swing.JFileChooser pictureChooser - HelpDialog helpDialog - javax.swing.JMenuBar menuBar - javax.swing.JMenu sampleMenu - javax.swing.JMenuItem dataPictureMenuItem - javax.swing.JCheckBoxMenuItem weightedMenuItem - javax.swing.JCheckBoxMenuItem gridMenuItem - javax.swing.JCheckBoxMenuItem alignMenuItem - javax.swing.JMenu algMenu - javax.swing.JMenu helpMenu

31. ábra

Az osztály feladata a programhoz tartozó főablak – önálló ablakként, vagy applet formájában –, a felület egyes elemeinek és a gráfot tartalmazó objektumnak a létrehozása, az ezek közötti kapcsolatok koordinálása. A felület többi osztálya rajta keresztül képes üzenetet megjeleníteni az állapotsorban, csúcsot és élt kijelölni, az aktuális kijelölést és a menüpontok beállításait lekérdezni. (31. ábra)

public <T extends java.awt.Component & javax.swing.RootPaneContainer>
MainWindow(T root)

A koordináló osztály konstruktora. Létrehozza a gráfot tartalmazó objektumot és a felület egyes elemeit, paramétere a főablakot megvalósító objektum. A felület elemeihez tartozó szövegeket és az üzenetek szövegét a *bogqaai/Bundle.properties* erőforrásfájlból tölti be, a párbeszédablakok feliratait az *UIManager.properties*, a mintagráfok

listáját a *bogqaai/Samples.properties*, az algoritmusok listáját pedig a *bogqaai/Algorithms.properties* fájlból.

public static void main(String[] args)

A program belépési pontja. Létrehozza a főablakot (*javax.swing.JFrame*) és egy példányt a koordináló osztályból. A főablak tartalma a parancssori paraméterektől függ. Ha nincs paraméter, akkor a kezdőablakot jeleníti meg, különben átugorja azt. Ha a paraméter „.”, akkor egy üres gráfot hoz létre, egyébként betölti a megadott, gráfot tartalmazó fájlt.

public void setStatus(String status)

Az állapotsor tartalmának beállítása.

public void setStatus(String status, Object ... arguments)

Az állapotsor tartalmának beállítása, egy megadott üzenet formátum és annak paraméterei segítségével.

public boolean isWeighted()

Az *Élsúlyok megjelenítése* menüpont beállításának lekérdezése.

public boolean showGrid()

A *Rácspontok megjelenítése* menüpont beállításának lekérdezése.

public boolean alignToGrid()

A *Rácshoz igazítás* menüpont beállításának lekérdezése.

public Graph.Vertex getVertex()

A szerkesztésre kijelölt csúcs lekérdezése.

public Graph.Edge getEdge()

A szerkesztésre kijelölt él lekérdezése.

public Algorithm getAlgorithm()

A szemléltetésre kiválasztott algoritmus lekérdezése.

public void selectNone()

Csúcs, él és algoritmus kijelölések megszüntetése.

public void selectVertex(Graph.Vertex u)

Csúcs kijelölése szerkesztésre.

public void selectEdge(Graph.Edge e)

Él kijelölése szerkesztésre.

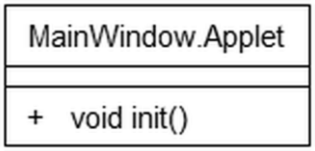
public void newGraph()

Üres gráf létrehozása.

public void openGraph()

Fájl megnyitás párbeszédablak megjelenítése és gráf betöltése fájlból.

MAINWINDOW.APPLET



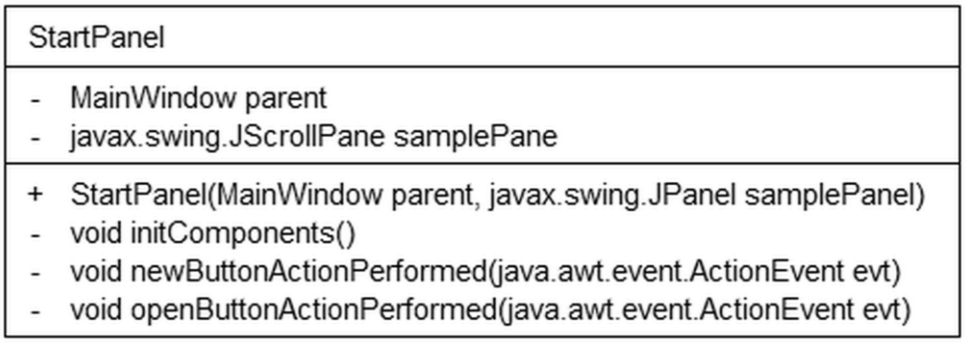
32. ábra

A böngészőbe ágyazható applet osztálya. Az osztály a *javax.swing.JApplet* osztályból származik. (32. ábra)

public void init()

Az applet inicializálásakor lefutó kód. Létrehoz egy példányt a koordináló osztályból, és megjeleníti a kezdőlapot.

STARTPANEL



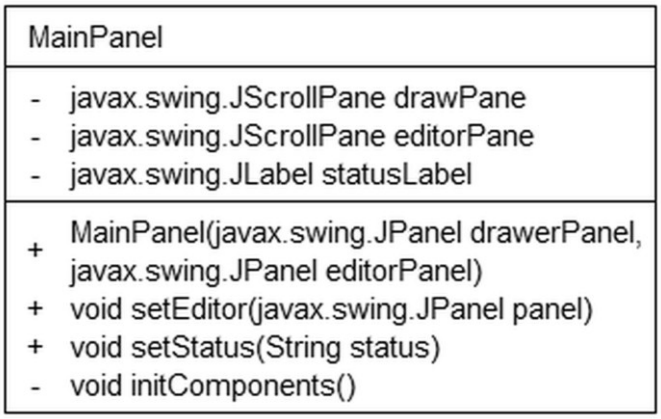
33. ábra

A kezdőlap felülete. Grafikus, *javax.swing.JPanel* leszármazott osztály, a NetBeans szerkesztőjével készült. (33. ábra)

public StartPanel(MainWindow parent, javax.swing.JPanel samplePanel)

A kezdőlap konstruktora, paramétere a koordináló objektum és a mintagráfok listáját tartalmazó *JPanel*.

MAINPANEL



34. ábra

A főablak kezdőlap utáni vázát leíró felület. Grafikus, *javax.swing.JPanel* leszármazott osztály, a NetBeans szerkesztőjével készült. A váz tartalmazza a gráf megjelenítőt, a szerkesztőablakot és az állapotsort. (34. ábra)

public MainPanel(javax.swing.JPanel drawerPanel, javax.swing.JPanel editorPanel)

Konstruktor, paraméterei a gráf megjelenítő és a szerkesztőablak kezdeti tartalma.

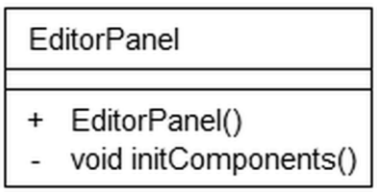
public void setEditor(javax.swing.JPanel panel)

A szerkesztőablak tartalmának beállítása.

public void setStatus(String status)

Az állapotsor tartalmának beállítása.

EDITORPANEL



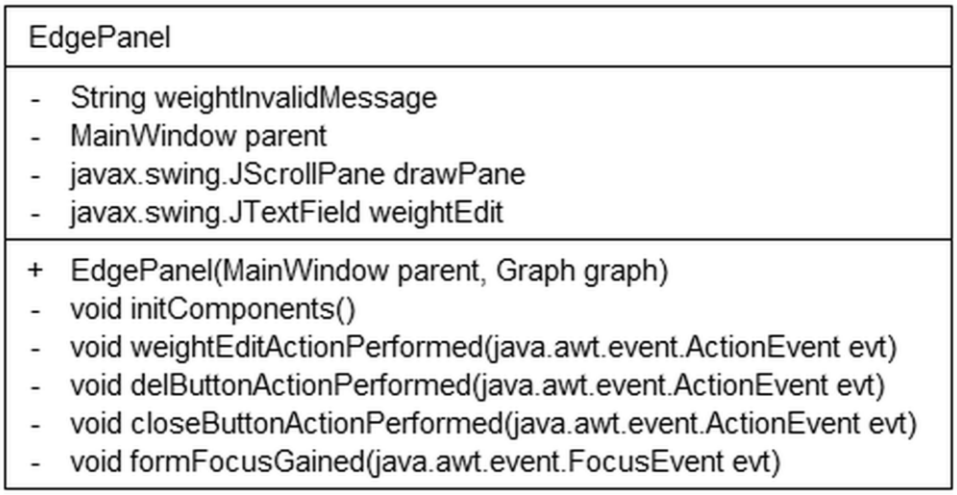
35. ábra

Gráf szerkesztés összefoglaló felület. Grafikus, *javax.swing.JPanel* leszármazott osztály, a NetBeans szerkesztőjével készült. (35. ábra)

public EditorPanel()

Az összefoglaló felület konstruktora.

EDGEPANEL



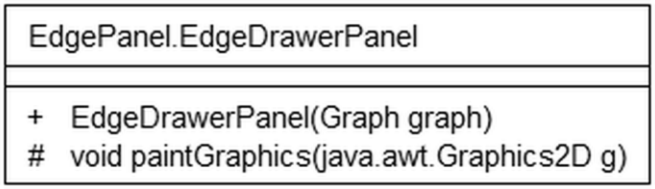
36. ábra

A szerkesztőablakban megjelenő él szerkesztő felület. Grafikus, *javax.swing.JPanel* leszármazott osztály, a NetBeans szerkesztőjével készült. (36. ábra)

public EdgePanel(MainWindow parent, Graph graph)

Az él szerkesztő felület konstruktora, paramétere a koordináló objektum és a gráf.

EDGEPANEL.EDGEDRAWERPANEL



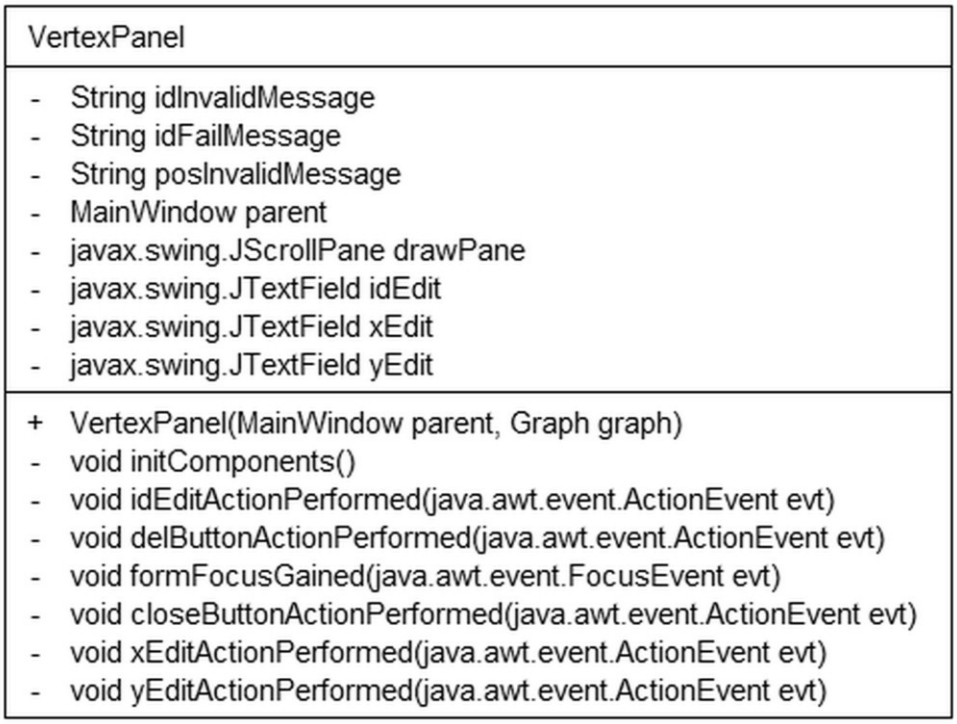
37. ábra

Az élt és két végpontját kirajzoló belső osztály. Az absztrakt gráfot kirajzoló *DrawerPanel* osztályból származik. A rajzolási felület közepén a szerkesztésre kijelölt élt és annak végpontjait jeleníti meg. (37. ábra)

public EdgeDrawerPanel(Graph graph)

A rajzoló osztály konstruktora, paramétere a gráf.

VERTEXPANEL



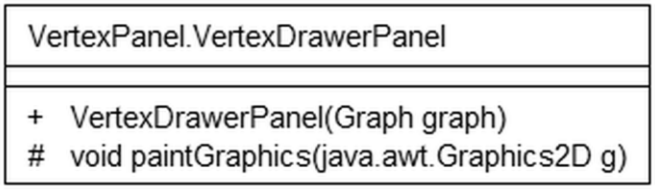
38. ábra

A szerkesztőablakban megjelenő csúcs szerkesztő felület. Grafikus, *javax.swing.JPanel* leszármazott osztály, a NetBeans szerkesztőjével készült. (38. ábra)

public VertexPanel(MainWindow parent, Graph graph)

A csúcs szerkesztő felület konstruktora, paramétere a koordináló objektum és a gráf.

VERTEXPANEL.VERTEXDRAWERPANEL



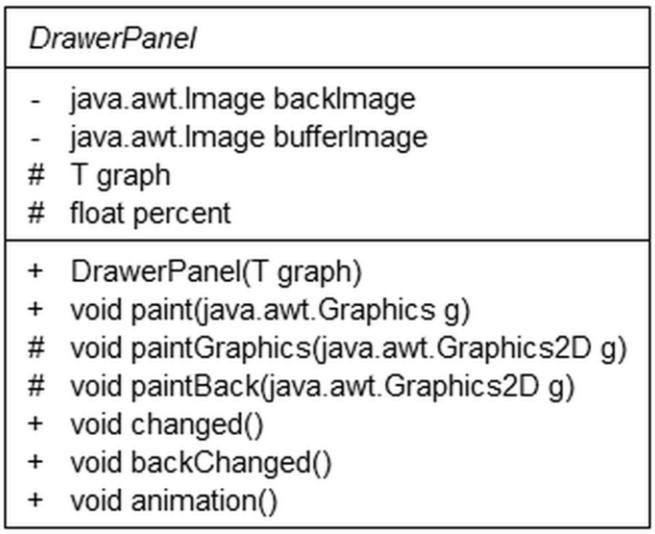
39. ábra

A csúcsot kirajzoló belső osztály. Az absztrakt gráfot kirajzoló *DrawerPanel* osztályból származik. A rajzolási felület közepén a szerkesztésre kijelölt csúcsot jeleníti meg. (39. ábra)

public VertexDrawerPanel(Graph graph)

A rajzoló osztály konstruktora, paramétere a gráf.

DRAWERPANEL



40. ábra

Az absztrakt gráfot kirajzoló felületek ősosztálya. Az osztály a *javax.swing.JPanel* osztályból származik. Egy alapértelmezett működési mechanizmust biztosít: az absztrakt gráf megváltozásakor és animálásakor automatikusan újrarajzolja magát. A tényleges újrarajzolás módját a leszármazott osztályok adják meg. (40. ábra)

public DrawerPanel(T graph)

A rajzoló felület létrehozása, paramétere az absztrakt gráf.

public final void paint(java.awt.Graphics g)

A háttér és az absztrakt gráf képének kirajzolása, előtte azok szükség szerinti frissítése. A metódus a befoglaló *JPanel* rajzolási művelete.

protected void paintGraphics(java.awt.Graphics2D g)

Az absztrakt gráf kirajzolása a leszármazott osztályokban.

protected void paintBack(java.awt.Graphics2D g)

A háttér kirajzolása a leszármazott osztályokban.

public void changed()

Az absztrakt gráf megváltozásának eseménye.

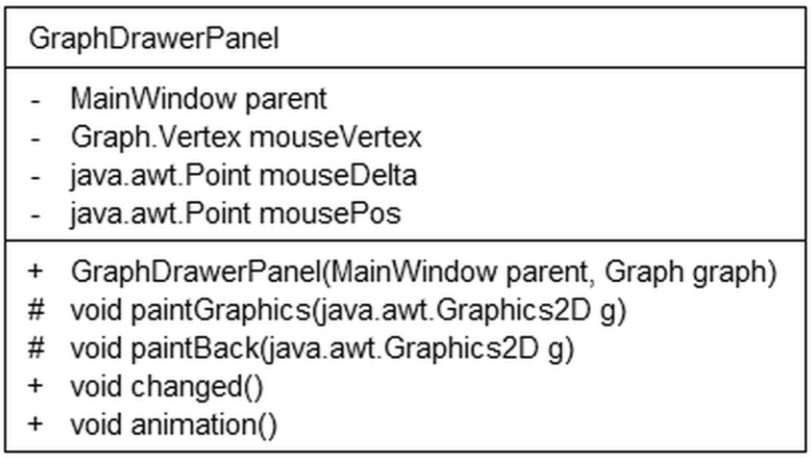
public void backChanged()

A háttér megváltoztatása.

public void animation()

Az absztrakt gráf animálásának eseménye.

GRAPHDRAWERPANEL



41. ábra

A gráf megjelenítőben lévő, a gráfot kirajzoló osztály. Az absztrakt gráfot kirajzoló *DrawerPanel* osztályból származik. A rajzolási felületen a gráf csúcsait és éleit jeleníti meg, azok színezésével együtt. Az egérrel végezhető szerkesztési műveletek ezen a felületen keresztül kerülnek végrehajtásra. (41. ábra)

public GraphDrawerPanel(MainWindow parent, Graph graph)

Gráf rajzoló osztály konstruktora, paramétere a koordináló objektum és a gráf.

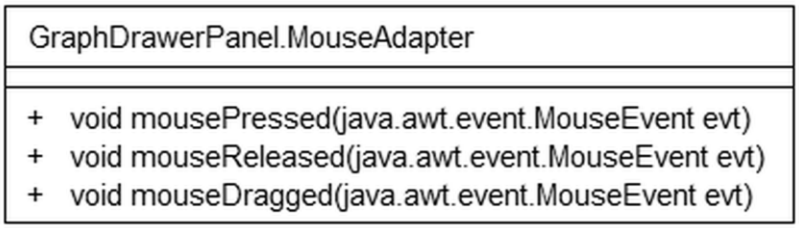
public void changed()

A gráf megváltozásának eseménye.

public void animation()

A gráf animálásának eseménye.

GRAPHDRAWERPANEL.MOUSEADAPTER



42. ábra

Egéreseményekkezelésének belső osztálya. (42. ábra)

public void mousePressed(java.awt.event.MouseEvent evt)

Egérgomb lenyomás eseménye.

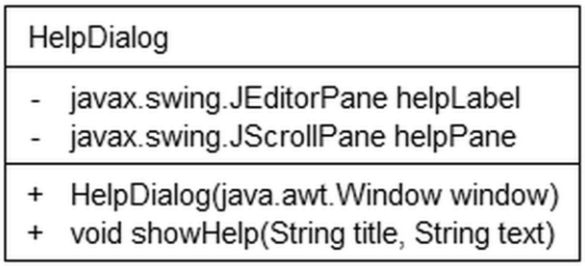
public void mouseReleased(java.awt.event.MouseEvent evt)

Egérgomb felengedés eseménye.

public void mouseDragged(java.awt.event.MouseEvent evt)

Egér húzás eseménye.

HELPDIALOG



43. ábra

Súgóablak az algoritmusok részletes leírásának megjelenítésére. Az ablak a *javax.swing.JDialog* osztályból származik. (43. ábra)

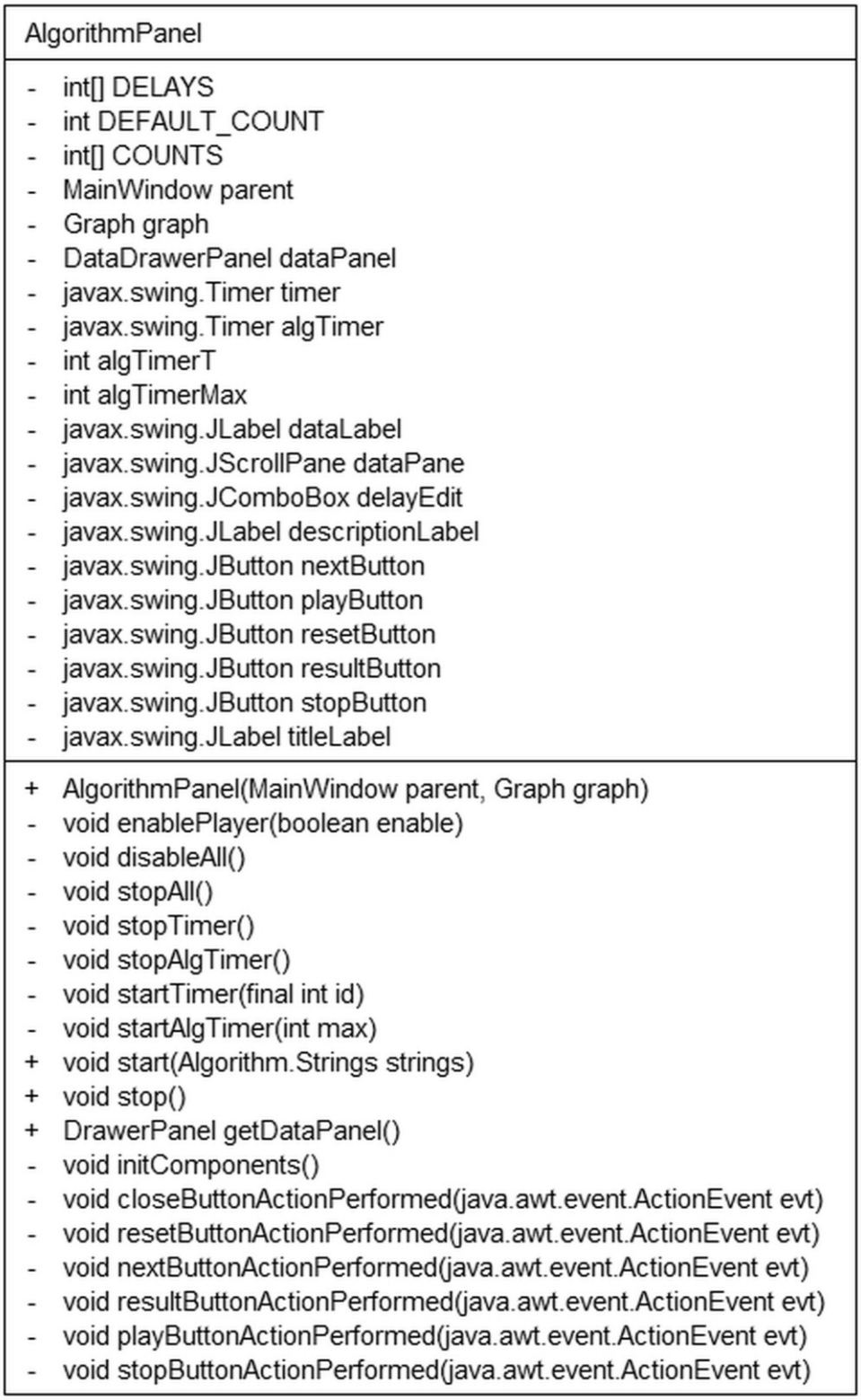
public HelpDialog(java.awt.Window window)

A súgóablak konstruktora, paramétere az őt létrehozó *Window*.

public void showHelp(String title, String text)

Súgó megjelenítése, paraméterei az ablak címe és a megjelenítendő leírás.

ALGORITHMPANEL



44. ábra

A szerkesztőablakban megjelenő algoritmus vezérlő felület. Grafikus, *javax.swing.JPanel* leszármazott osztály, a NetBeans szerkesztőjével készült. Az algo-

ritmusok léptetését és animálását *javax.swing.Timer* segítségével valósítja meg, ehhez több belső állandót, adattagot és metódust használ. (44. ábra)

public AlgorithmPanel(MainWindow parent, Graph graph)

Az algoritmus vezérlő felület konstruktora, paramétere a koordináló objektum és a gráf.

public void start(Algorithm.Strings strings)

Algoritmus szemléltetés indítása, paramétere az algoritmus adatait leíró rekord.

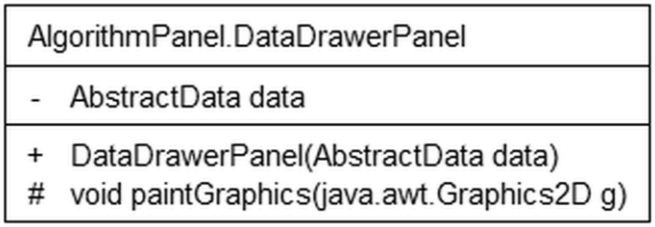
public void stop()

Algoritmus szemléltetés leállítása.

public DrawerPanel getDataPanel()

Az adatszerkezetet kirajzoló felület lekérdezése.

ALGORITHMPANEL.DATADRAWERPANEL



45. ábra

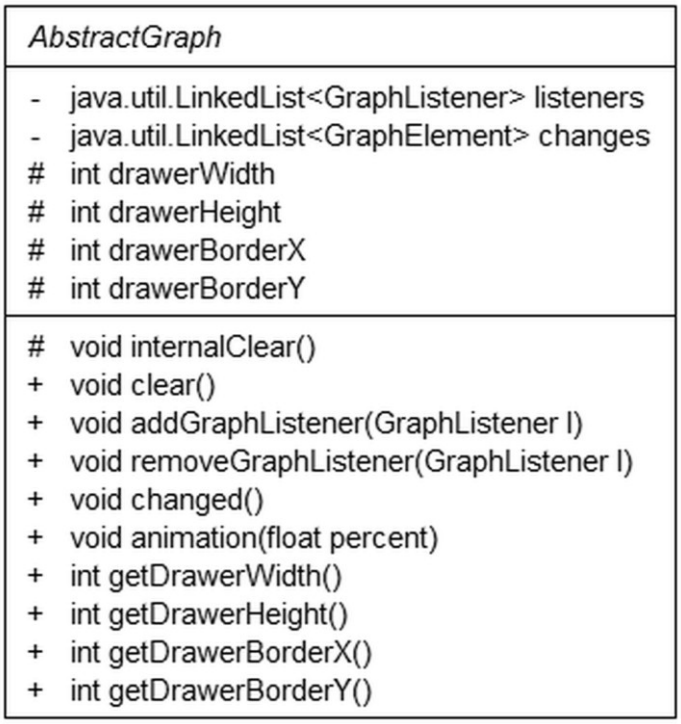
Az adatszerkezetet kirajzoló belső osztály. Az absztrakt gráfot kirajzoló *DrawerPanel* osztályból származik. (45. ábra)

public DataDrawerPanel(AbstractData data)

A rajzoló osztály konstruktora, paramétere az adatszerkezet.

3.2.3. GRÁF ÁBRÁZOLÁS CSOMAGJA

ABSTRACTGRAPH



46. ábra

Az absztrakt gráf ősosztály, a grafikusan kirajzolható gráf és adatszerkezetek őse. Tartalmazza az absztrakt gráf elem ősosztályt, egy csúcsot és élt rajzoló osztályt és a rajzoláshoz szükséges konstansokat. Az osztály biztosítja a gráf megváltozás értesítések eseményvezérelt mechanizmusát: a grafikus komponensek értesítést kérhetnek a gráf megváltozásáról illetve kiválthatják az eseményt a gráf elemeinek módosítása után. Az esemény lehet egyszerű vagy animált. (46. ábra)

protected void internalClear()

A gráf beállításainak törlése a leszármazott osztályokban.

public final void clear()

A gráf beállításainak törlése, gráf megváltozás esemény kiváltása.

public final void addGraphListener(GraphListener l)

Feliratkozás a gráf megváltozásának eseményére.

public final void removeGraphListener(GraphListener l)

Leiratkozás a gráf megváltozásának eseményéről.

public final void changed()

Gráf megváltozás esemény kiváltása a gráf elemeinek módosítása után.

public final void animation(float percent)

Gráf megváltozás esemény animált kiváltása a gráf elemeinek módosítása után, paramétere az animáció állása.

public final int getDrawerWidth()

A gráf valódi szélessége rajzoláskor. Értéke a leszármazott osztályokban a belső *drawerWidth*, *drawerBorderX* változók segítségével adható meg.

public final int getDrawerHeight()

A gráf valódi magassága rajzoláskor. Értéke a leszármazott osztályokban a belső *drawerHeight*, *drawerBorderY* változók segítségével adható meg.

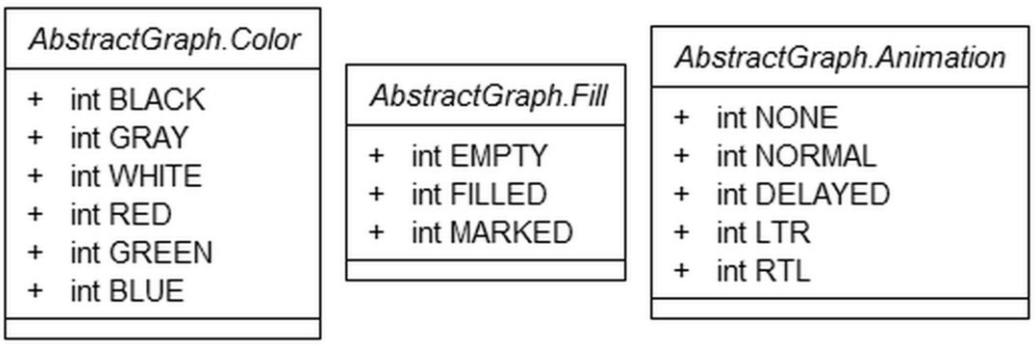
public final int getDrawerBorderX()

Vízszintes szegély vastagsága rajzoláskor. Értéke a leszármazott osztályokban a belső *drawerBorderX* változó segítségével adható meg.

public final int getDrawerBorderY()

Függőleges szegély vastagsága rajzoláskor. Értéke a leszármazott osztályokban a belső *drawerBorderY* változó segítségével adható meg.

ABSTRACTGRAPH.COLOR, ABSTRACTGRAPH.FILL, ABSTRACTGRAPH.ANIMATION



47. ábra

A csúcsok és élek rajzolásához használható színek, kitöltés típusok – nincs kitöltés, a csúcs színével megegyező a kitöltés vagy a csúcs sárgával megjelölt –, és animáció tí-

pusok – nincs animáció, színátmenetes, késleltetett, élek esetén adott irányban színátmenetes animáció. (47. ábra)

ABSTRACTGRAPH.DRAWER

AbstractGraph.Drawer	
+	int BORDER
-	java.awt.Color YELLOW
-	java.awt.Color WHITE
-	java.awt.Color[] COLORS
-	java.awt.Color[] FILLS
-	int borderX
-	int borderY
-	int radiusX
-	int radiusY
-	boolean multiLineVertex
-	java.awt.Stroke vertexStroke
-	java.awt.Stroke edgeStroke
-	java.awt.Font vertexFont
-	java.awt.Font edgeFont
-	java.awt.FontMetrics vertexFm
-	java.awt.FontMetrics edgeFm
-	int vertexFmTop
-	int edgeFmHeight
-	int edgeFmAscent
-	java.awt.Graphics2D g
+	Drawer(java.awt.Graphics2D g, boolean border, int radiusX, int radiusY, boolean multiLineVertex, int vertexFontSize, int edgeFontSize, int vertexWidth, int edgeWidth)
#	java.awt.Color fillColor(int color, int fill)
#	java.awt.Color mixColor(java.awt.Color oldColor, java.awt.Color color, float percent)
+	void drawVertex(int x, int y, java.awt.Color color, java.awt.Color fill, String text)
+	void drawEdge(int x1, int y1, int x2, int y2, java.awt.Color color)
+	void drawEdgeLabel(int x1, int y1, int x2, int y2, java.awt.Color color, String text)
+	void drawGrid(int width, int height)

48. ábra

A csúcsokat és éleket kirajzoló absztrakt osztály. A rajzolás alapvető mechanizmusát biztosítja: csúcsot, élt és rácspontokat képes rajzolni a megfelelő paraméterekkel, és tartalmazza a rajzoláshoz szükséges adatokat. (48. ábra)

public Drawer(java.awt.Graphics2D g, boolean border, int radiusX, int radiusY, boolean multiLineVertex, int vertexFontSize, int edgeFontSize, int vertexWidth, int edgeWidth)

A rajzoló osztály konstruktora, paraméterei a rajzolási cél és a rajzolás beállításai.

public final void drawVertex(int x, int y, java.awt.Color color, java.awt.Color fill, String text)

Csúcs rajzolása megadott színnel, kitöltéssel és felirattal.

public final void drawEdge(int x1, int y1, int x2, int y2, java.awt.Color color)

Él rajzolása megadott színnel.

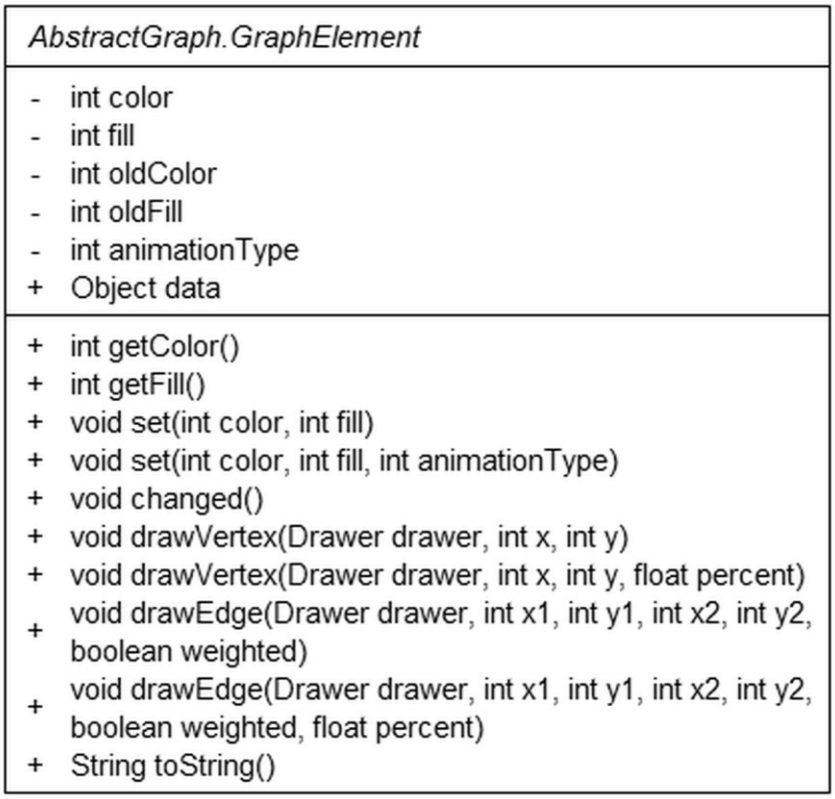
public final void drawEdgeLabel(int x1, int y1, int x2, int y2, java.awt.Color color, String text)

Él feliratának rajzolása megadott színnel.

public final void drawGrid(int width, int height)

Rácspontok rajzolása, megadott szélességben és magasságban.

ABSTRACTGRAPH.GRAPHELEMENT



49. ábra

Az absztrakt gráf elem ősosztály, a tényleges csúcs és él osztályok őse. Tartalmazza a gráf elemek közös adattagjait, a megváltoztatásukhoz, kirajzolásukhoz és animálásukhoz szükséges metódusokat. Minden gráf elem rendelkezik színnel és kitöltéssel, tárolja az aktuális animáció típusát és egy *data* adattagot további információk hozzárendelésére. Ezen adatok lekérdezése és módosítása mellett az elemnek lekérdezhető a felirata, és kirajzolható csúcsként vagy élként. (49. ábra)

public final int getColor()

Szín lekérdezése.

public final int getFill()

Kitöltés típusának lekérdezése.

public final void set(int color, int fill)

Szín és kitöltés megváltoztatása.

public final void set(int color, int fill, int animationType)

Szín és kitöltés megváltoztatása animáltan.

public final void changed()

Az elem adatainak megváltozásakor, a kirajzolás illetve animáció utáni jelzés.

public void drawVertex(Drawer drawer, int x, int y)

Az elem kirajzolása csúcsként, egy rajzoló objektum segítségével.

public void drawVertex(Drawer drawer, int x, int y, float percent)

Az elem animált kirajzolása csúcsként, egy rajzoló objektum segítségével.

public void drawEdge(Drawer drawer, int x1, int y1, int x2, int y2, boolean weighted)

Az elem kirajzolása élként, egy rajzoló objektum segítségével.

public void drawEdge(Drawer drawer, int x1, int y1, int x2, int y2, boolean weighted, float percent)

Az elem animált kirajzolása élként, egy rajzoló objektum segítségével.

public String toString()

Felirat szövege a leszármazott osztályokban.

GRAPH

Graph
- String emptyMessage - String intInvalidMessage - String vertexMissingMessage - String vertexInvalidMessage - String vertexFailMessage - String edgeInvalidMessage - String edgeFailMessage - java.util.ArrayList<Vertex> vertices - java.util.ArrayList<java.util.ArrayList<Edge>> edges - int size - boolean locked
+ Graph() + void open(java.io.Reader in) throws java.io.IOException + void save(java.io.Writer out) throws java.io.IOException # void internalClear() - int internalParseInt(String s, int line) - <T> java.util.ArrayList<T> internalCreateList(int newSize, T o) - <T> void internalMove(java.util.ArrayList<T> list, int oldIndex, int newIndex) - int internalGetVertexIndex(int id) - Vertex internalGetVertex(int id) - Vertex internalAddVertex(int id, int index, int x, int y, boolean aligned) - Edge internalAddEdge(Vertex u, Vertex v, int w) - void internalDimensionChanged() + int getSize() + java.util.List<Vertex> getVertices() + Edge getEdge(Vertex u, Vertex v) + Vertex findVertex(int x, int y) + void setLocked(boolean lock) + boolean isLocked() + Vertex addVertex(int x, int y, boolean aligned) + Edge addEdge(Vertex u, Vertex v) + boolean removeVertex(Vertex u) + boolean removeEdge(Edge e) + boolean setVertexId(Vertex u, int id) + void reset(boolean gray) + void resetVertices(boolean gray) + void resetEdges(boolean gray) + Drawer drawGrid(java.awt.Graphics2D g, int width, int height) + Drawer drawGraph(java.awt.Graphics2D g, boolean weighted, float percent) + Drawer drawVertex(java.awt.Graphics2D g, Vertex u, int x, int y) + Drawer drawEdge(java.awt.Graphics2D g, Edge e, int x1, int y1, int x2, int y2, boolean weighted)

50. ábra

A gráfot leíró osztály, az absztrakt gráf ősosztályból származik. Tartalmazza a csúcsokat és éleket leíró osztályokat és egy saját rajzoló osztályt. A csúcsokat és éleket egy a sorszámuk alapján rendezett tömbben illetve mátrixban tárolja. A reprezentáció kiválasztásakor a megjelenítés és a szemléltetés hatékonysága volt a meghatározó szempont, a gráf változtatási műveletek hatékonyságának rovására. A tömbös ábrázolással a csúcsok bejárásának és elérésének hatékonnyá tétele, a rendezettséggel az algoritmusok determinisztikussá tétele volt a cél. Az osztály az ősosztály funkcionalitása mellett metódusokat biztosít a gráf mentésére és beolvasására, változtatási és kirajzoló műveleteire. (50. ábra)

public Graph()

A gráf osztály konstruktora. Betölti a fájl megnyitással kapcsolatos hibaüzeneteket a *bogqaai/graph/Graph.properties* fájlból.

public void open(java.io.Reader in) throws java.io.IOException

Gráf megnyitása fájlból. Törli a gráf tartalmát és beállításait és betölti a gráfot tartalmazó fájlt. Sikertelen megnyitás esetén *java.io.IOException*, hibás fájl formátum esetén *GraphFormatException* kivételt generál a hiba okát megjelölve.

public void save(java.io.Writer out) throws java.io.IOException

Gráf mentése fájlba. Sikertelen mentés esetén *java.io.IOException* kivételt generál a hiba okát megjelölve.

public int getSize()

A gráf méretének, a csúcsok számának lekérdezése.

public java.util.List<Vertex> getVertices()

A csúcsok rendezett tömbjének lekérdezése.

public Edge getEdge(Vertex u, Vertex v)

Két csúcs között húzódó él lekérdezése. Ha nincs ilyen él, akkor visszatérési értéke *null*.

public Vertex findVertex(int x, int y)

Csúcs keresése koordináták alapján. Visszatérési értéke az a legnagyobb sorszámú csúcs, aminek a megjelenített körébe beleesnek a megadott koordináták, vagy *null*, ha nincs ilyen.

public void setLocked(boolean lock)

A gráf szerkesztésének letiltása vagy engedélyezése.

public boolean isLocked()

A szerkeszthetőség letiltásának lekérdezése.

public Vertex addVertex(int x, int y, boolean aligned)

Csúcs hozzáadása a gráfhoz, megadott koordinátákon, opcionális rácshoz igazítással. A létrehozott csúcs sorszáma a legnagyobb sorszámú csúcs sorszámánál eggyel nagyobb, vagy 1, ha a gráf üres. Visszatérési értéke a létrehozott csúcs, vagy *null*, ha a gráf nem szerkeszthető.

public Edge addEdge(Vertex u, Vertex v)

Él létrehozása vagy meglévő él kijelölése két csúcs között. Visszatérési értéke az él, vagy *null*, ha a gráf nem szerkeszthető.

public boolean removeVertex(Vertex u)

Csúcs törlése, ha a gráf szerkeszthető.

public boolean removeEdge(Edge e)

Él törlése, ha a gráf szerkeszthető.

public boolean setVertexId(Vertex u, int id)

Csúcs sorszámának módosítása. Visszatérési értéke a művelet sikeressége.

public void reset(boolean gray)

Összes csúcs és él szürkére vagy feketére színezése.

public void resetVertices(boolean gray)

Összes csúcs szürkére vagy feketére színezése.

public void resetEdges(boolean gray)

Összes él szürkére vagy feketére színezése.

public Drawer drawGrid(java.awt.Graphics2D g, int width, int height)

Rácspontok rajzolása, megadott szélességben és magasságban. Visszatérési értéke a rajzoló objektum.

public Drawer drawGraph(java.awt.Graphics2D g, boolean weighted, float percent)

Gráf kirajzolása, paraméterei az élsúlyok megjelenítése és az animáció állása. Visszatérési értéke a rajzoló objektum.

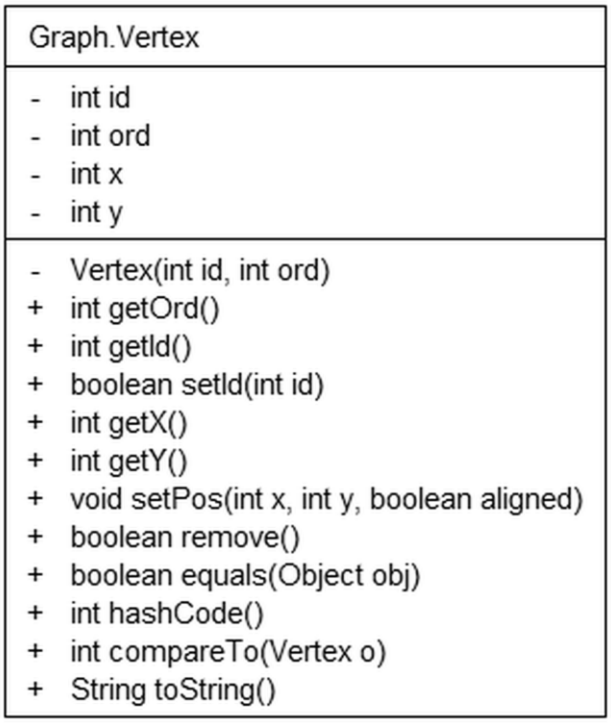
public Drawer drawVertex(java.awt.Graphics2D g, Vertex u, int x, int y)

Kijelölt csúcs kirajzolása egy megadott helyre.

public Drawer drawEdge(java.awt.Graphics2D g, Edge e, int x1, int y1, int x2, int y2, boolean weighted)

Kijelölt él kirajzolása egy megadott helyre.

GRAPH.VERTEX



51. ábra

A gráf csúcsait leíró osztály, az absztrakt gráf elem ősosztályból származik. Az ősosztály működése mellett minden csúcs rendelkezik egy, a felhasználó által megadott, a gráfon belül egyedi sorszámmal, tárolja a koordinátáit és a csúcsok tömbjében lévő helyét. A csúcs törölhető, sorszáma változtatható és összehasonlítható önmagával a rendezett tároláshoz. Kirajzoláskor felirata a sorszáma. (51. ábra)

public int getOrd()

A csúcsok tömbjében lévő hely lekérdezése.

public int getId()

Sorszám lekérdezése.

public boolean setId(int id)

Sorszám megváltoztatása, paramétere az új sorszám. Visszatérési értéke jelzi a művelet sikerességét.

public int getX()

X koordináta lekérdezése.

public int getY()

Y koordináta lekérdezése.

public void setPos(int x, int y, boolean aligned)

Koordináták beállítása, opcionális rácshoz igazítással. A művelet egy gráf megváltozási eseményt generál.

public boolean remove()

Csúcs törlése.

public boolean equals(Object obj)

Csúcsok azonosságának vizsgálata.

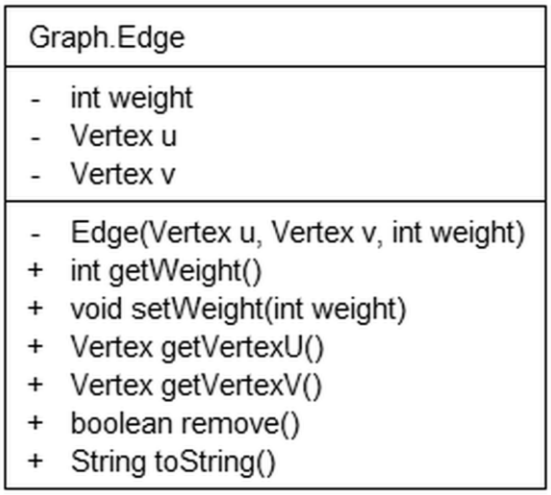
public int compareTo(Vertex o)

Összehasonlító operátor a rendezett tároláshoz: a sorszámuk szerint hasonlítja össze a csúcsokat.

public String toString()

A sorszám feliratként.

GRAPH.EDGE



52. ábra

A gráf csúcsait leíró osztály, az absztrakt gráf elem ősosztályból származik. Az ősosztály működése mellett minden él rendelkezik egy súllyal és ismeri a két végpontját. Az él törölhető és a súlya változtatható. Kirajzoláskor felirata a súlya. (52. ábra)

public int getWeight()

Súly lekérdezése.

public void setWeight(int weight)

Súly megváltoztatása. A művelet egy gráf megváltozási eseményt generál.

public Vertex getVertexU()

Kisebbik sorszámú végpont lekérdezése.

public Vertex getVertexV()

Nagyobbik sorszámú végpont lekérdezése.

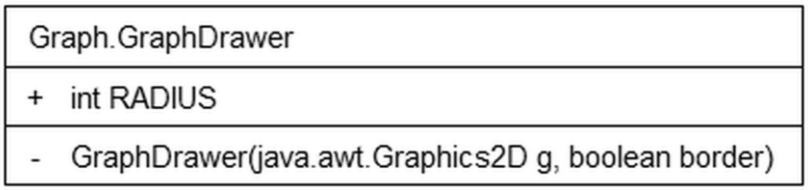
public boolean remove()

Él törlése.

public String toString()

A súly feliratként.

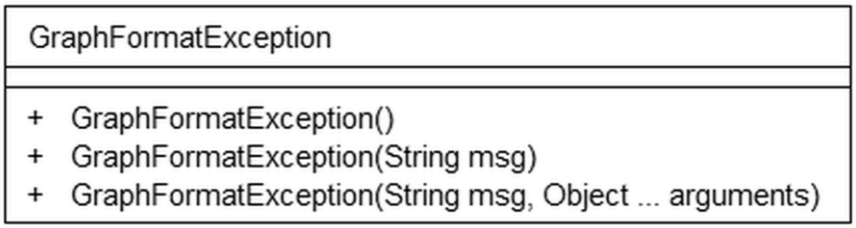
GRAPH.GRAPHDRAWER



53. ábra

A csúcsokat és éleket kirajzoló osztály, a gráf elemeit kirajzoló absztrakt osztály leszármazottja. Definiálja a rajzolás paramétereit. (53. ábra)

GRAPHFORMATEXCEPTION



54. ábra

Gráf fájlból történő beolvasásakor hibás formátum esetén kiváltott kivétel. (54. ábra)

public GraphFormatException()

A kivétel osztály konstruktora.

public GraphFormatException(String msg)

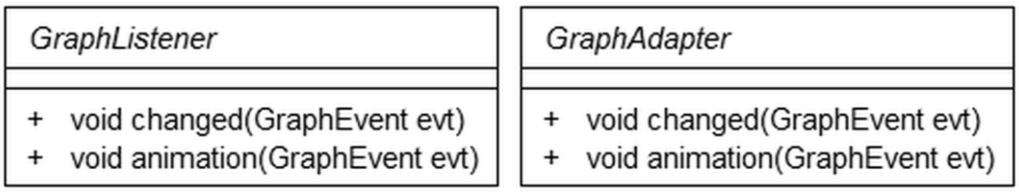
A kivétel osztály konstruktora, paramétere a hibaüzenet.

public GraphFormatException(String msg, Object ... arguments)

A kivétel osztály konstruktora, paraméterei a hibaüzenet formátuma és annak paraméterei.

3.2.4. GRÁF ESEMÉNYEK CSOMAGJA

GRAPHLISTENER, GRAPHADAPTER



55. ábra

Gráf megváltozás eseményre várakozók interfésze és absztrakt ősosztálya. Tartalmazza a megváltozás és az animált megváltozás események fogadó metódusait. (55. ábra)

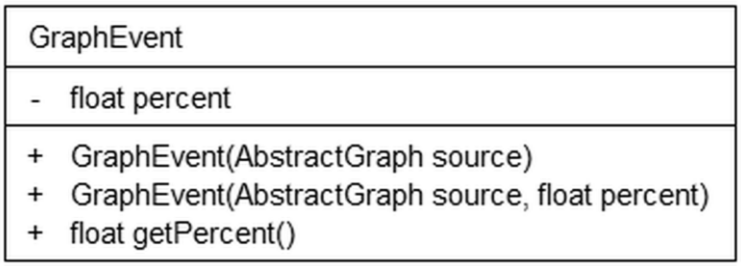
public void changed(GraphEvent evt)

Gráf megváltozása esemény fogadó metódusa.

public void animation(GraphEvent evt)

Gráf animált megváltozása esemény fogadó metódusa.

GRAPHEVENT



56. ábra

A gráf megváltozása eseményt leíró osztály. Tartalmazza az eseményt kiváltó absztrakt gráfot és az animáció állását. Az animáció állása 0 és 1 közötti lebegőpontos szám lehet, ha az esemény nem animált, akkor értéke 1. (56. ábra)

public GraphEvent(AbstractGraph source)

Az esemény osztály konstruktora, gráf megváltozása esetén. Paramétere az absztrakt gráf, az animáció állását 1-re állítja.

public GraphEvent(AbstractGraph source, float percent)

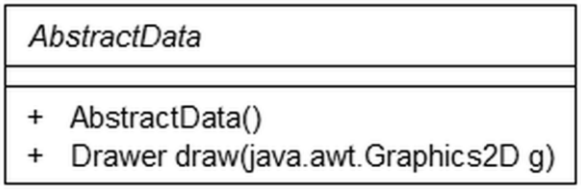
Az esemény osztály konstruktora, gráf megváltozása esetén. Paramétere az absztrakt gráf és az animáció állása.

public float getPercent()

Animáció állásának lekérdezése.

3.2.5. ALGORITMUSOK ÉS ADATSZERKEZETEK CSOMAGJA

ABSTRACTDATA



57. ábra

Az absztrakt adatszerkezet osztály, az adatszerkezet osztályok őse, az absztrakt gráf osztályból származik. Az ősosztály funkcionalitása mellett egy rajzolás műveletet tartalmaz, amit a leszármazott osztályok a saját kirajzolásukra használhatnak fel. (57. ábra)

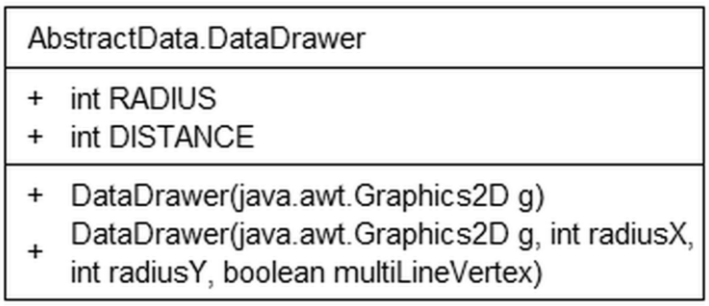
public AbstractData()

Az absztrakt adatszerkezet osztály konstruktora.

public Drawer draw(java.awt.Graphics2D g)

Adatszerkezet kirajzolása, a leszármazott osztályokban felülírható metódus. Visszatérési értéke a rajzoló objektum.

ABSTRACTDATA.DATADRAWER



58. ábra

Az adatszerkezetet kirajzoló belső osztály, a gráf elemeit kirajzoló absztrakt osztály leszármazottja. Definiálja a rajzolás paramétereit. (58. ábra)

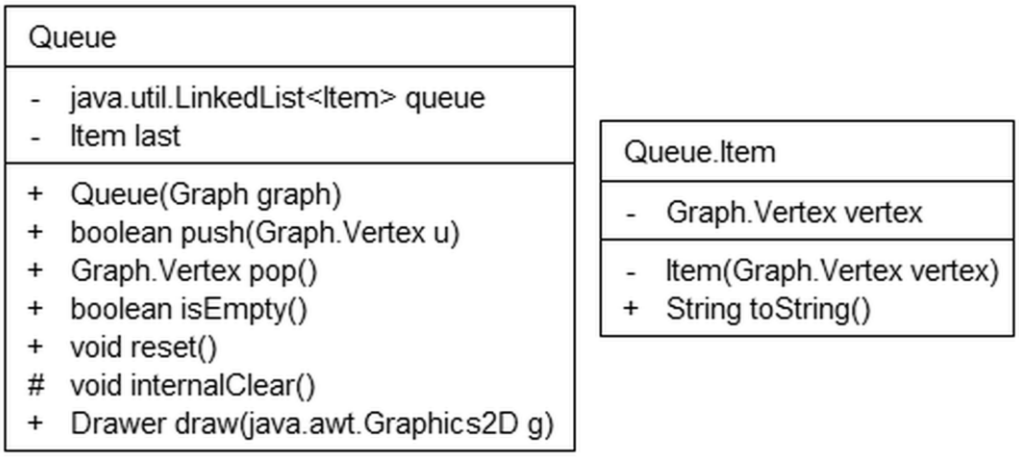
public DataDrawer(java.awt.Graphics2D g)

A rajzoló osztály konstruktora, paramétere a rajzolási cél. Beállítja a rajzolás paramétereit.

public DataDrawer(java.awt.Graphics2D g, int radiusX, int radiusY, boolean multiLineVertex)

A rajzoló osztály konstruktora, paramétere a rajzolási cél és a rajzolás beállításai.

QUEUE



59. ábra

A sor adatszerkezetet megvalósító osztály a gráf csúcsainak tárolására, az absztrakt adatszerkezet osztály leszármazottja. A sor rendelkezik a szokásos műveletekkel – elem berakása, kivétele, üresség lekérdezése –, kirajzolásnál megjeleníti a sorból legutoljára kivett elemet és még a sorban lévő elemeket. A sort egy láncolt lista segítségével implementálja, a sor műveleteit a lista műveletein keresztül valósítja meg. (59. ábra)

public Queue(Graph graph)

A sor adatszerkezetet konstruktora, paramétere a vizsgált gráf.

public boolean push(Graph.Vertex u)

Csúcs berakása a sor végére.

public Graph.Vertex pop()

Legelső csúcs kivétele a sorból.

public boolean isEmpty()

A sor ürességének vizsgálata. Visszatérési értéke igaz, ha a sor üres.

public void reset()

Sor alaphelyzetbe állítása, a sor elemeinek és a sorból utoljára kivett elemnek a törlése.

public Drawer draw(java.awt.Graphics2D g)

Sor és a sorból legutoljára kivett elem kirajzolása. Visszatérési értéke a rajzoló objektum.

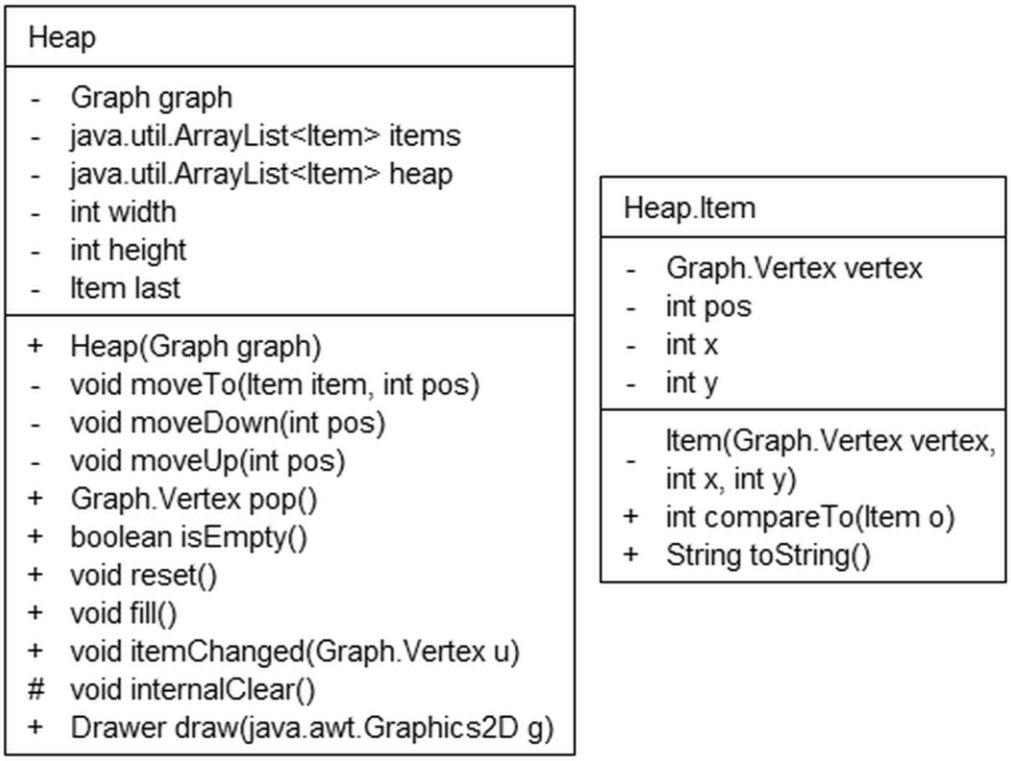
QUEUE.ITEM

A sor elemeinek belső osztálya, az absztrakt gráf elem osztály leszármazottja. Az elemekhez a gráf csúcsait lehet hozzárendelni. (59. ábra)

public String toString()

Az elem felirata a hozzárendelt csúcs felirata.

HEAP



60. ábra

A kupac adatszerkezetet megvalósító osztály a gráf csúcsainak tárolására, az absztrakt adatszerkezet osztály leszármazottja. A kupacban a csúcsok rendezése elsődlegesen a csúcs *data* adattagjához hozzárendelt érték alapján, másodlagosan a csúcs sorszáma alapján történik. A kupac rendelkezik a szokásos műveletekkel – elem berakása, módosítása, legkisebb elem kivétele –, kirajzolásnál megjeleníti a kupacból legutoljára kivett elemet és a kupacban lévő elemeket a fastruktúrának megfelelően. A kupac implementálása egy tömb segítségével történik, a tömb a fastruktúra szerint szintfolytonosan tárolja a kupac elemeit. (60. ábra)

public Heap(Graph graph)

A kupac adatszerkezetet konstruktora, paramétere a vizsgált gráf.

public Graph.Vertex pop()

Legkisebb elem kivétele a kupacból, kupac helyreállítása. A legkisebb elem a kupacot reprezentáló tömb első eleme. Kivételkor helyére a tömb utolsó eleme kerül, a kupachoz tartozó fa utolsó levele. A kupac helyreállítása az elemek cseréivel, az első helyre került elem fastruktúrabeli lefelé mozgatásával történik.

public boolean isEmpty()

A kupac ürességének vizsgálata. Visszatérési értéke igaz, ha a kupac üres.

public void reset()

Kupac alaphelyzetbe állítása, a tömbök és az utoljára kivett elemnek a törlése.

public void fill()

Kupac feltöltése a vizsgált gráf csúcsaival.

public void itemChanged(Graph.Vertex u)

Elem megváltoztatása a kupacban, a kupac helyreállítása. A művelet feltételezi, hogy a megváltoztatott elem értéke nem nőtt. A kupac helyreállítása az elemek cseréivel, a megváltozott elem fastruktúrabeli felfelé mozgatásával történik.

public Drawer draw(java.awt.Graphics2D g)

Kupac fastruktúra és a kupacból legutoljára kivett elem kirajzolása. Visszatérési értéke a rajzoló objektum.

HEAP.ITEM

A kupac elemeinek belső osztálya, az absztrakt gráf elem osztály leszármazottja. Az elemekhez a gráf csúcsait lehet hozzárendelni. (60. ábra)

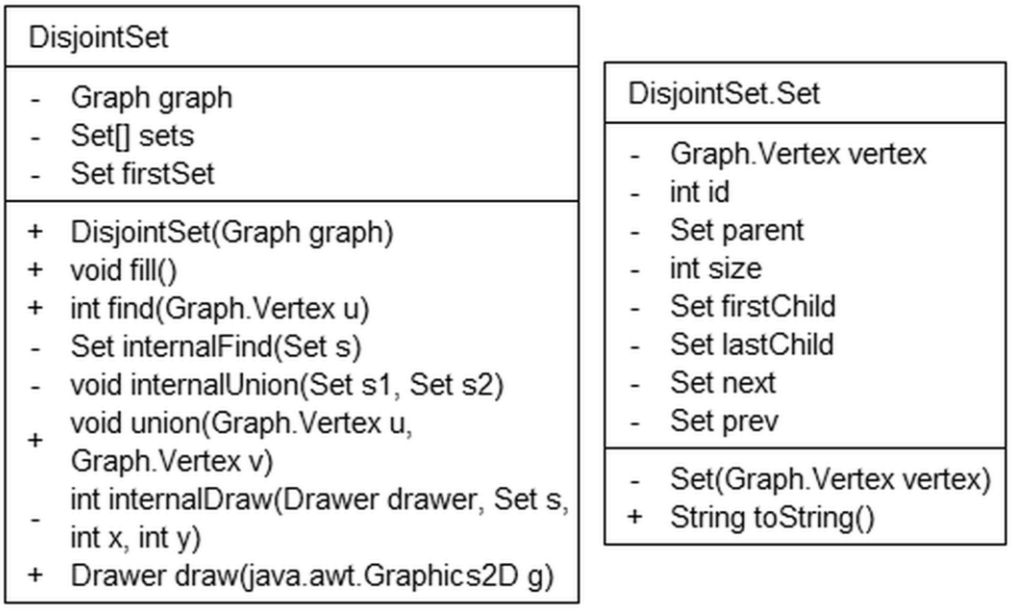
public int compareTo(Item o)

Összehasonlító operátor a rendezett tároláshoz: a hozzárendelt csúcs *data* adattagja és sorszáma alapján.

public String toString()

Az elem felirata a hozzárendelt csúcs felirata és a csúcs *data* adattagjának felirata, többsoros feliratként.

DISJOINTSET



61. ábra

Az Unió-Holvan adatszerkezetet megvalósító osztály, az absztrakt adatszerkezet osztály leszármazottja. Az adatszerkezet az unió és a holvan műveletekkel rendelkezik, kirajzolásnál a reprezentáló fastruktúrát jeleníti meg. Az adatszerkezet implementálása a hatékonyabb megjelenítés érdekében egy láncolt fa adatszerkezet segítségével történik, amelynek elemei a csúcsokhoz tartozó részhalmazok. A fa elemei a szülőmutató mellett tárolják az első és utolsó gyereküket, és a közvetlen szomszédjaikat is. (61. ábra)

public DisjointSet(Graph graph)

Az Unió-Holvan adatszerkezetet konstruktora, paramétere a vizsgált gráf.

public void fill()

Halmazok létrehozása, minden csúcs külön halmazba sorolása.

public int find(Graph.Vertex u)

Holvan művelet, meghatározza, hogy az adott csúcs hányadik halmazban van. A művelet a csúcsot tartalmazó fa gyökérelemének sorszámát adja meg, a keresés közben útösszenyomást hajtva végre.

public void union(Graph.Vertex u, Graph.Vertex v)

Unió művelet, két csúcshoz tartozó halmazt egyesít. A nagyobb halmazhoz tartozó fa gyökeréhez gyerekként hozzákapcsolja a másikhoz tartozó fa gyökerét.

public Drawer draw(java.awt.Graphics2D g)

Halmazok fastruktúrájának a kirajzolása. Visszatérési értéke a rajzoló objektum.

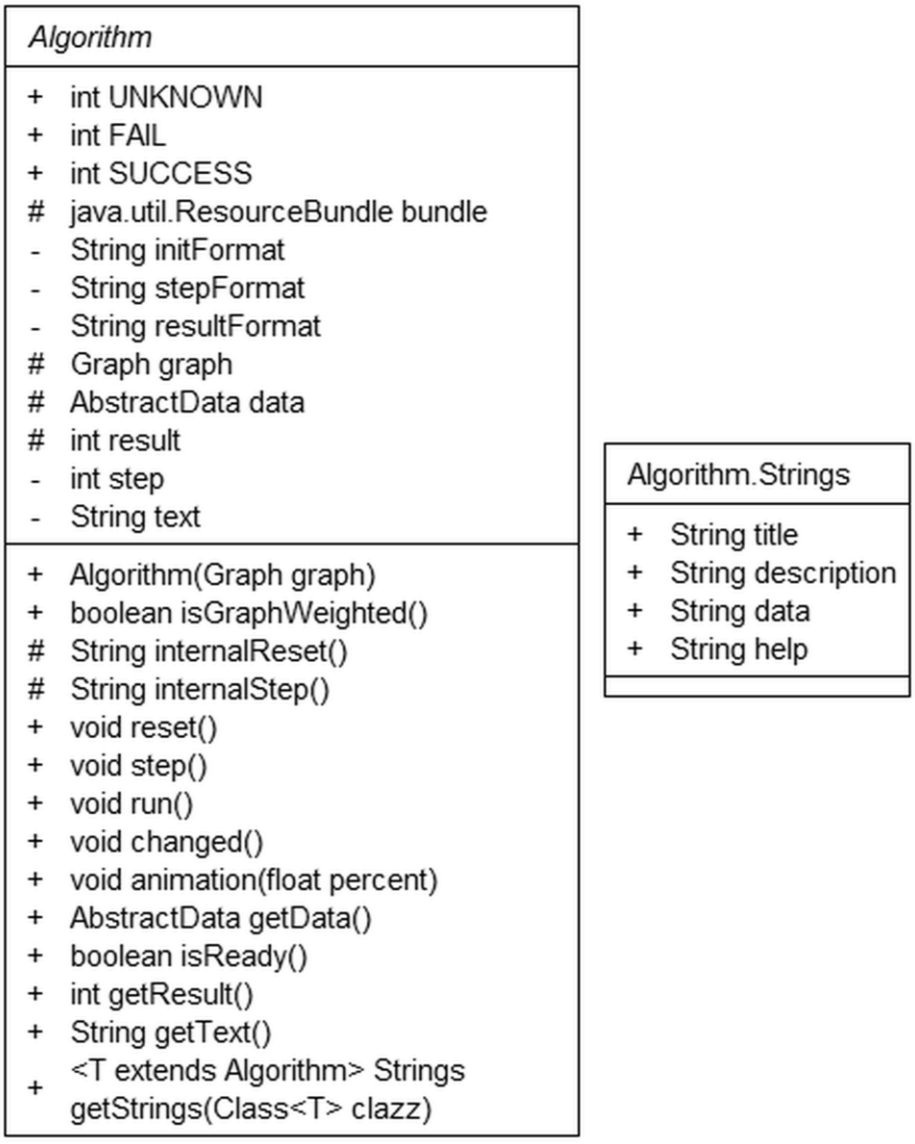
DISJOINTSET.SET

A csúcsokhoz tartozó részhalmazok, a fa elemeinek belső osztálya, az absztrakt gráf elem osztály leszármazottja. Tárolja a halmaz méretét, sorszámát és mutatóit. (61. ábra)

public String toString()

A halmaz felirata a hozzárendelt csúcs felirata.

ALGORITHM



62. ábra

Az absztrakt algoritmus ősosztály. Az osztály egy alapértelmezett működési mechanizmust biztosít az algoritmusoknak. Az algoritmus adatai lekérdezhetők, az algoritmus léptethető, a háttérben lefuttatható, lekérdezhető az aktuális lépés és az eredmény leírása, hozzákapcsolható adatszerkezet és kiválthat eseményeket a gráf megváltozásáról és animálásáról. A tényleges működést a leszármazott osztályok belső metódusok felülírásával és adattagok módosításával tudják meghatározni. (62. ábra)

public Algorithm(Graph graph)

Az algoritmus osztály konstruktora, paramétere a vizsgálta gráf. Létrehozásakor zárolja a gráfot, letiltja annak szerkeszthetőségét. Az algoritmusok leírásait a *bogqaai/graph/algorithm/Algorithm.properties* fájlból tölti be.

public abstract boolean isGraphWeighted()

Élsúlyok használata, a leszármazott osztályokban definiált metódus.

protected abstract String internalReset()

Az algoritmus alaphelyzetbe állítása a leszármazott osztályokban. Visszatérési értéke az inicializálás leírása.

protected abstract String internalStep()

Az algoritmus léptetése a leszármazott osztályokban. Visszatérési értéke a lépés szövege vagy az algoritmus eredménye.

public final void reset()

Az algoritmus alaphelyzetbe állítása.

public final void step()

Az algoritmus léptetése, ha még nem fejezte be futását, lépés leírásának formázott megjelenítése.

public final void run()

Az algoritmus futtatása a háttérben.

public final void changed()

Gráf megváltozás esemény kiváltása.

public final void animation(float percent)

Gráf megváltozás esemény animált kiváltása.

public final AbstractData getData()

Az adatszerkezet lekérdezése, értékét a leszármazottak a belső *data* adattagon keresztül állítják be. Visszatérési értéke *null*, ha az algoritmus nem használja.

public final boolean isReady()

Megvizsgálja, hogy az algoritmus befejezte-e futását.

public final int getResult()

Lekérdezi az algoritmus futásának eredményét. Ha az algoritmus még nem ért véget, akkor értéke *UNKNOWN*.

public final String getText()

Inicializáláshoz, lépéshez, eredményhez tartozó szöveg lekérdezése.

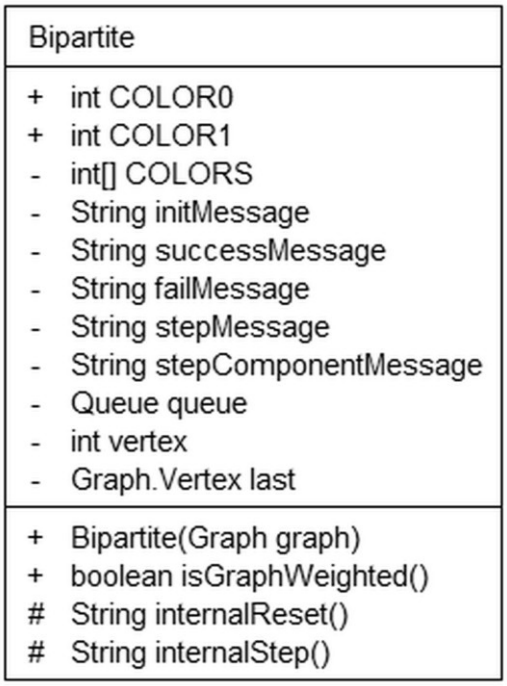
public static <T extends Algorithm> Strings getStrings(Class<T> clazz)

Leszármazott osztály adatait tartalmazó rekord elkészítése.

ALGORITHM.STRINGS

Az algoritmusok adatait leíró rekord. Tartalmazza az algoritmus és a kapcsolódó adatszerkezet nevét, az algoritmus rövid és részletes leírását. (62. ábra)

BIPARTITE



63. ábra

A párosság vizsgálatát megvalósító algoritmus osztálya, őse az absztrakt algoritmus osztály. (63. ábra)

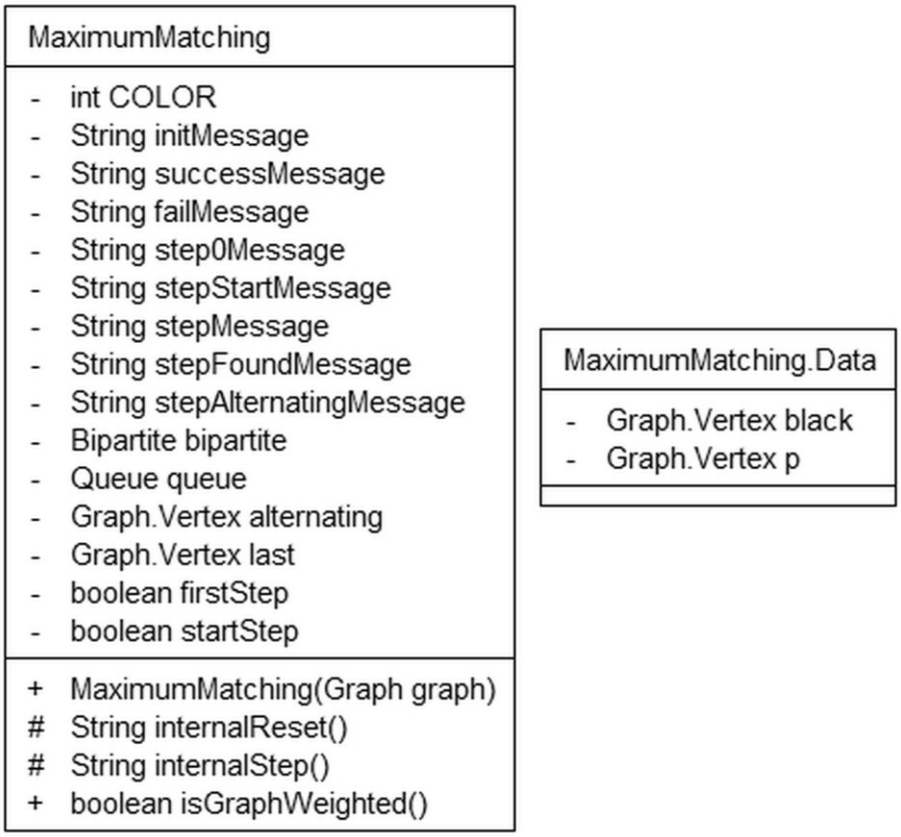
public Bipartite(Graph graph)

Párosság vizsgálata algoritmus konstruktora, paramétere a vizsgált gráf.

public boolean isGraphWeighted()

Az algoritmus nem használ élsúlyokat, visszatérési értéke hamis.

MAXIMUMMATCHING



64. ábra

A magyar módszert megvalósító algoritmus osztálya, őse az absztrakt algoritmus osztály. (64. ábra)

public MaximumMatching(Graph graph)

A magyar módszer algoritmusának konstruktora, paramétere a vizsgált gráf.

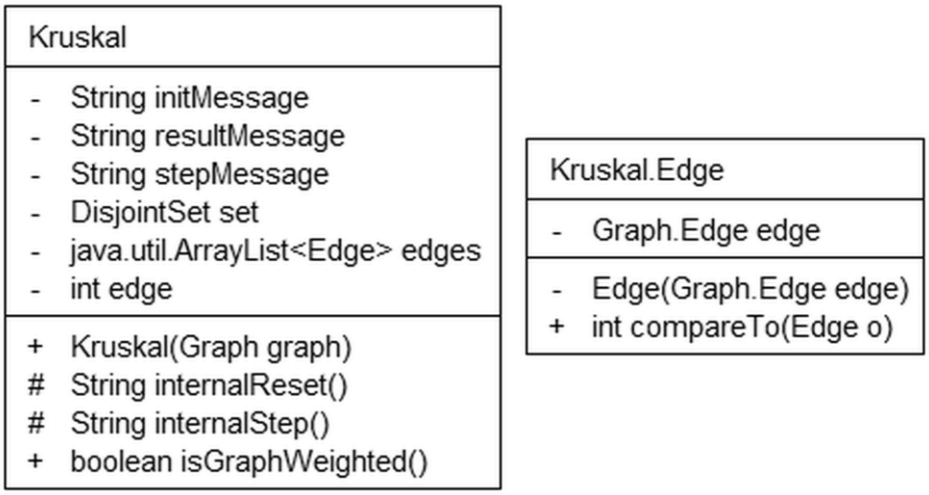
public boolean isGraphWeighted()

Az algoritmus nem használ élsúlyokat, visszatérési értéke hamis.

MAXIMUMMATCHING.DATA

A csúcsokhoz hozzárendelt adatok az algoritmus futása során: megelőző csúcs és a belőle induló párosított él. (64. ábra)

KRUSKAL



65. ábra

A Kruskal algoritmust megvalósító osztály, őse az absztrakt algoritmus osztály. (65. ábra)

public Kruskal(Graph graph)

A Kruskal algoritmus konstruktora, paramétere a vizsgált gráf. Nagyság szerint növekvő sorrendbe rendezi az éleket.

public boolean isGraphWeighted()

Az algoritmus használja az élsúlyokat, visszatérési értéke igaz.

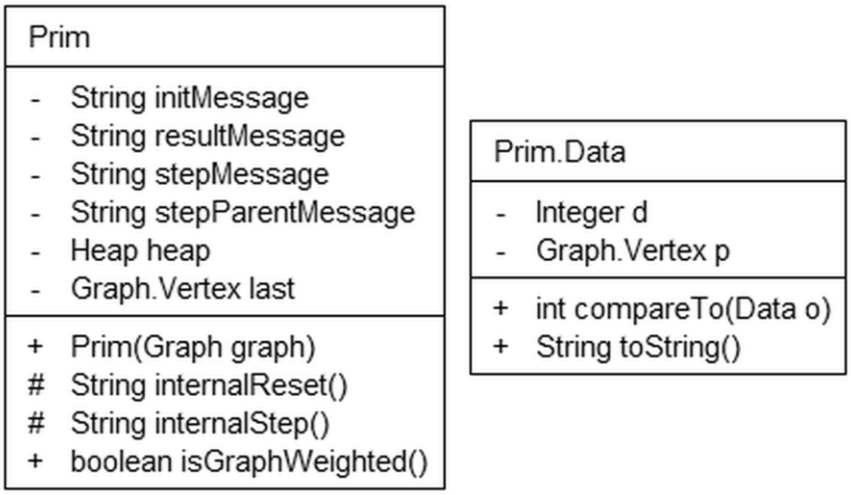
KRUSKAL.EDGE

Az élek belső osztálya, a gráf éleit lehet hozzárendelni. (65. ábra)

public int compareTo(Edge o)

Összehasonlító operátor az élek rendezéséhez a súlyuk szerint.

PRIM



66. ábra

A Prim algoritmust megvalósító osztály, őse az absztrakt algoritmus osztály. (66. ábra)

public Prim(Graph graph)

A Prim algoritmus konstruktora, paramétere a vizsgált gráf.

public boolean isGraphWeighted()

Az algoritmus használja az élsúlyokat, visszatérési értéke igaz.

PRIM.DATA

A csúcsokhoz hozzárendelt adatok az algoritmus futása során: megelőző csúcs és az erdőtől vett távolság. (66. ábra)

public int compareTo(Data o)

Összehasonlító operátor a távolságok rendezésére. A *d* adattag *null* értéke a végtelent reprezentálja.

public String toString()

Az erdőtől vett távolság feliratként. A feliraton szerepelhet a végtelen.

3.2.6. ERŐFORRÁS CSOMAGOK, SPECIÁLIS FÁJLOK

FELIRATOK, ÜZENETEK

A feliratok és üzenetek tárolására több erőforrásfájl szolgál, funkcionalitás szerint csoportosítva. A grafikus felület feliratai és alapvető üzenetei a *bogqaai/Bundle.properties* fájlban, a párbeszédablakok feliratai az *UIManager.properties* fájlban találhatók. Az egyes algoritmusok leírása a *bogqaai/graph/algorithm/Algorithm.properties* fájlban, a gráf formátummal kapcsolatos üzenetek a *bogqaai/graph/Graph.properties* fájlban helyezkednek el.

ALGORITMUSOK

Az algoritmusok listájának tárolására a *bogqaai/Algorithms.properties* fájl szolgál. A hagyományos Java erőforrásfájlban az algoritmusokat megvalósító osztályok nevei szerepelnek értékként. A bejegyzésekhez rendelt kulcsok kétjegyű egész számok, a program ezen kulcsok alapján rendezi az algoritmusokat a menüben. A fájlban szereplő „-” érték egy elválasztójelet hoz létre az algoritmusok menüjében, a megadott helyen.

Az algoritmusok leírásai a *bogqaai/graph/algorithm/Algorithm.properties* fájlban találhatók. A fájl *ALGORITMUS.NÉV=ÉRTÉK* formátumú bejegyzéseket tartalmaz, ahol *ALGORITMUS* az algoritmust megvalósító osztály neve, *NÉV* a leírás megnevezése, *ÉRTÉK* pedig egy szöveges leírás. A bejegyzések között a következők minden algoritmus esetén szerepelnek:

- *title:* az algoritmus neve.
- *description:* az algoritmus leírása.
- *data:* az adatszerkezet neve.
- *help:* az algoritmushoz tartozó súgó szövege.
- Az algoritmusok lépéseinek leírásai és eredményeihez tartozó üzenetek.

MINTAGRÁFOK

A mintagráfok listájának tárolására a *bogqaai/Samples.properties* fájl szolgál, a tényleges mintagráfok szöveges fájlként a *bogqaai.samples* csomagban vannak tárolva. Az algoritmusokéhoz hasonlóan hagyományos Java erőforrásfájlról van szó, hasonló kulcs-

kezeléssel. A nem „-” értékű bejegyzések formátuma *NÉV:FÁJL:S*, ahol *NÉV* a gráf neve, *FÁJL* a fájlnév a csomagban, *S* pedig *1* vagy *0* értéket vehet fel, aszerint, hogy megnyitáskor a gráf súlyai megjelenjenek-e.

KÉPEK

A grafikus felülethez tartozó képek – az algoritmus vezérlő felület ikonjai – a *bogqaai.icons* csomagban vannak tárolva.

3.2.7. FEJLESZTÉSI LEHETŐSÉGEK

Az osztályszerkezet kialakításakor és az implementáció során is szem előtt tartottam a továbbfejlesztés lehetőségét. A programhoz könnyen lehet új mintagráfot vagy egy új algoritmust hozzáadni az eredeti forrásfájlok újrafordítása nélkül, csupán a Java archívum módosításával.

MINTAGRÁF HOZZÁADÁSA

Mintagráf hozzáadásakor első lépésként a gráfot tartalmazó fájlt kell a *bogqaai.samples* erőforrás csomagban elhelyezni. Ezután a *bogqaai/Samples.properties* fájlban létre kell hozni egy bejegyzést a mintagráfhoz, tetszőleges kulccsal, az *Erőforrás csomagok, speciális fájlok* részben leírtak alapján.

ALGORITMUS HOZZÁADÁSA

Bővítéskor az új algoritmusnak a *bogqaai.graph.algorithm.Algorithm* osztály leszármazottjának kell lennie, megvalósítva annak tisztán virtuális metódusait. Követelmény továbbá, hogy az osztály a *bogqaai.graph.algorithm* csomagban helyezkedjen el, és legyen egy publikus konstruktora, ami pontosan egy gráfot vár paraméterül. A *bogqaai/Algorithms.properties* fájlban létre kell hozni egy bejegyzést az algoritmushoz, a gráf leírásait a *bogqaai/graph/algorithm/Algorithm.properties* fájlban kell elhelyezni, az *Erőforrás csomagok, speciális fájlok* részben leírtak alapján. Ha az algoritmus olyan adatszerkezetet használ, ami a *bogqaai.graph.algorithm.AbstractData* osztály leszármazottja, akkor az algoritmus szemléltetés során lehetőség nyílik az adatszerkezet szemléltetésére is.

3.3. TESZTELÉS

A program tesztelése során teszteltem a grafikus felületet és az elkészített algoritmusokat és adatszerkezeteket is.

A program alapvetően szemléltető jellegű, nem célja a nagy adattömeg melletti működés. Nagy gráf mellett a szemléltetés átláthatatlanná válik és a sok grafikus elem kirajzolása a felület lassabb működését eredményezi. A program ablakában alapesetben egy 30 csúcsból álló gráf fér el kényelmesen, a tesztesetekben ennél kisebb gráfok szerepelnek.

3.3.1. MODULTESZTELÉS

FEKETE DOBOZ TESZTELÉS

A gráf szerkesztő felület tesztelése során a következőket vizsgáltam:

- Csúcsok és élek létrehozása, törlése, mozgatása, kijelölése, és szerkesztése.
- Csúcs és él szerkesztésekor hibás értékek megadása.
- Gráf fájlműveletek vizsgálata: új gráf létrehozása, fájl mentése, megnyitása.
- Gráf megnyitásakor hibás fájl megadása, a lehetséges hibákkal.
- Gráf megjelenítés tesztelése: élsúlyok megjelenítés és elrejtése, rács megjelenítése és rácshoz igazítás, gráf képének mentése.

Az algoritmusok és adatszerkezetek tesztelése során a következő, általános teszteseteket vizsgáltam:

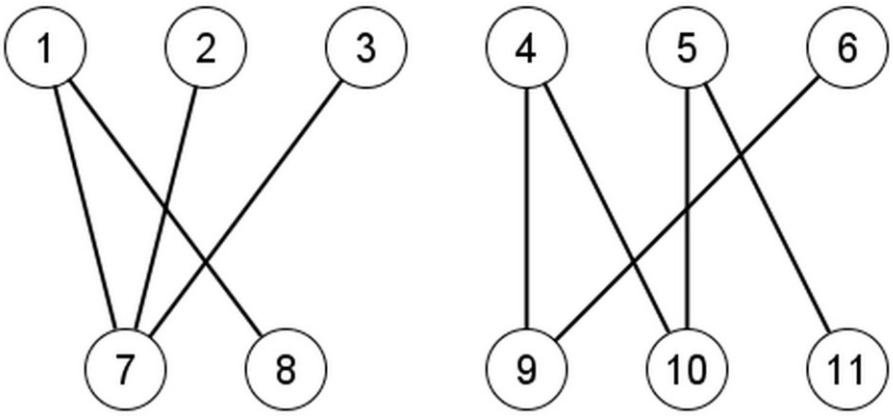
- Algoritmusok működése üres és néhány csúcsot tartalmazó gráfok esetén.
- Tesztelés páros és nem páros gráfokra, páros gráfok esetén teljes és részleges párosítást tartalmazó gráfokra.
- Összefüggő és nem összefüggő gráfok vizsgálata.
- Fák és kört tartalmazó gráfok vizsgálata.
- Élsúlyok különböző eloszlása.

Az algoritmus szemléltető tesztelése során annak funkcióit próbáltam ki, egymás utáni többszöri végrehajtással.

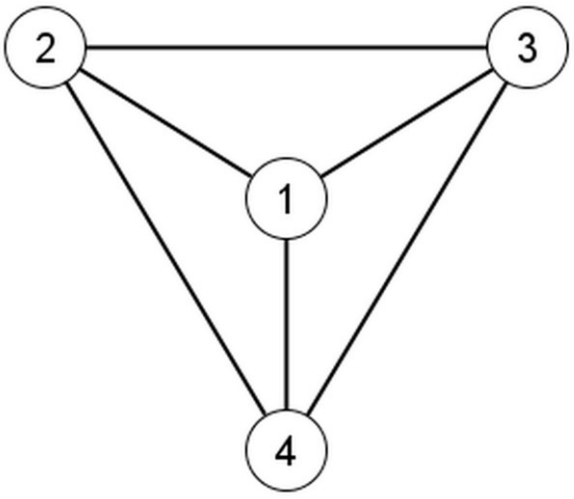
Fehér doboz tesztelés

A grafikus felület fehér doboz tesztelését a fekete doboz tesztekhez hasonlóan végeztem, de figyelembe vettem a tényleges implementációt és a gráf reprezentációját.

Az algoritmusok tesztelése során a fekete doboz tesztek mellett olyan teszteseteket vizsgáltam, melyek lefedik az algoritmus lefutásának minden lehetséges ágát. Ezek az ágak az algoritmusok leírásaiban ismertetett lépéseknek felelnek meg. Ennek megfelelően az algoritmusok fő tesztelését a következő gráfokkal végeztem:

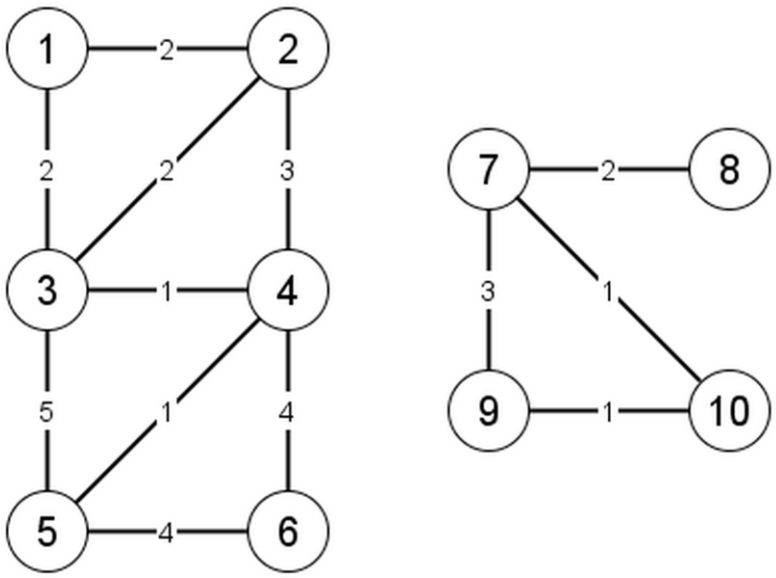


67. ábra



68. ábra

- A párosság vizsgálatát és a maximális párosítás keresését a 67. ábrán látható páros gráffal és a 68. ábrán látható nem páros gráffal teszteltem.



69. ábra

- A feszítőfa illetve feszítő erdő keresését a 69. ábrán látható gráffal teszteltem.

A fenti gráfok betölthetők a kezdőlapról és a *Mintagráf megnyitása* menün keresztül. A felsoroltakon kívül a listában szerepel egy nagy gráf és szerepelnek kisebb, könnyebben átlátható gráfok is, amivel az algoritmusok részleges tesztelését végeztem: két egyszerűbb páros gráf és az Algoritmusok és adatszerkezetek II. tárgy jegyzetében[1] található, a feszítőfát kereső algoritmusok bemutatásához használt gráfok.

3.3.2. RENDSZERTESZTELÉS

A rendszertesztelés során a programot – az önálló ablakkal rendelkező változatot és a böngészőből indítható változatot egyaránt – több platform, operációs rendszer, Java környezet és böngésző mellett teszteltem:

- Windows 7 – 32 és 64 bites változat –, Windows XP és Ubuntu Linux
- Java SE 6 és Java SE 7
- Mozilla Firefox, Internet Explorer és Google Chrome

3.3.3. TAPASZTALATOK

A teszteléseket már a fejlesztés korai fázisaitól kezdve végeztem az esetleges implementációs hibák kiküszöbölésére. A tesztelés több esetben visszahatással volt az implementációra is. A gráf csúcsainak és éleinek reprezentációjára a kezdeti tesztelések eredménye alapján választottam rendezett tömböt illetve mátrixot. Az algoritmusokat úgy akartam megvalósítani, hogy ha több lehetséges csúcs közül kell választania, akkor mindig a legkisebb sorszámút válassza, ehhez rendezett adatszerkezetre volt szükség. A tesztelés alapján úgy döntöttem, hogy a gráf a kirajzolás szempontjából legyen optimális, ezért választottam a rendezett tömböt és mátrixot. Ezáltal a gráf megváltoztatási műveletek költségesebbek lettek, de a sebességet meghatározó grafikus megjelenítés teljesítménye javult.

A rendszertesztelés során azt tapasztaltam, hogy a Java fájl párbeszédablak minden platformon angolul jelenik meg, feliratai nem szerepelnek a Java virtuális gép erőforrásai között magyarul. Ennek a hiányosságnak a pótlására hoztam létre az *UIManager.properties* erőforrásfájlt, ami a kérdéses feliratokat tartalmazza, magyarul.

A rendszertesztelés alapján az önálló ablakként futó változat bármely vizsgált platform és környezet mellett stabilan működik, a böngészőből indítható applet futtatásához Java SE 7, Firefox 11, Internet Explorer 9 vagy Google Chrome 17 optimális.

A böngészőből indítható változat tesztelése során kiderült az is, hogy az appletnak csak korlátozott jogosultságai vannak, például nem tud fájlt olvasni, írni. A probléma kiküszöbölésére az archívumot digitális aláírással láttam el. Ha a felhasználó az applet indításakor ezt jóváhagyja, akkor a program böngészőben is teljes funkcionalitással működik.

IV. Összefoglalás

Munkám során egy interaktív, grafikus alapú szemléltető programot készítettem, aminek célja a kiválasztott gráfalgoritmusok és kapcsolódó adatszerkezetek működésének bemutatása volt. Remélem, hogy segítségével könnyebbé válik az algoritmusok megértése, és betekintést lehet nyerni azok hátterében működő speciális adatszerkezetekbe. Mivel a tárgyaltak szoros kapcsolatban állnak az Algoritmusok és adatszerkezetek II. tárgy anyagával, így reményeim szerint az elkészített anyag tanulási segédeszközként is megállja a helyét.

A kiválasztott algoritmusoknak több szempontból speciális változatait dolgoztam fel. Az algoritmusokat kiterjesztettem nem összefüggő gráfokra, esetenként a szakirodalomban fellelhető több változatból is merítettem, illetve apróbb megszorításokat tettem, ügyelve az algoritmus hatékonyságának megőrzésére.

Az elkészített programnak két fő része van, egy gráf megjelenítő és szerkesztő felület és az algoritmus szemléltetés. Az algoritmus szemléltetés magába foglalja az algoritmusok részletes ismertetését, lépésenkénti áttekintését, animált megjelenítését, és egy betekintést a hozzájuk kapcsolódó adatszerkezet működésébe.

A tervezés és a fejlesztés során egy objektumorientált, eseményvezérelt elven működő rendszert készítettem, amiben logikailag szétválnak a program egyes rétegei: a grafikus felületet felépítő elemek, a gráf ábrázolás, az algoritmusok és adatszerkezetek megvalósítása. Az egyes rétegek egymással különféle kapcsolatban állnak, de közös vonásuk, hogy mind szorosan összefonódnak a grafikus megjelenítéssel, hiszen ez a program fő célja. A komponensek fejlesztését ennek figyelembe vételével végeztem, a grafikus vonások jóformán a program egészét végigkísérik. Hatékonysági szempontoknál mindig a grafikus megjelenítés hatékonyságát, a felület átláthatóságát és könnyű kezelhetőségét helyeztem előtérbe.

A program elkészítésekor szem előtt tartottam a továbbfejlesztés lehetőségét, a kialakított osztályszerkezet és struktúra lehetővé teszi, hogy könnyen bővíthető legyen új algoritmusokkal, adatszerkezetekkel.

IRODALOMJEGYZÉK

[1] Fekete István: Gráfalgoritmusok (egyetemi jegyzet)
http://people.inf.elte.hu/fekete/docs_2/grafalg/grafalg.pdf (2012.05.07.)

[2] Rónyai Lajos, Ivanyos Gábor, Szabó Réka: Algoritmusok, Typotex, 1999, [379], ISBN-963-9132-16-0

[3] Lovász László, Gács Péter: Algoritmusok, Műszaki Könyvkiadó, 1978, [179], ISBN-963-10-2067-3

[4] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein: Új algoritmusok, Scolar Kiadó, 2003, [992], ISBN-963-9193-90-9

[5] Java Platform, Standard Edition 7, API Specification
http://docs.oracle.com/javase/7/docs/api/ (2012.05.07.)