23. SZÉLESSÉGI BEJÁRÁS

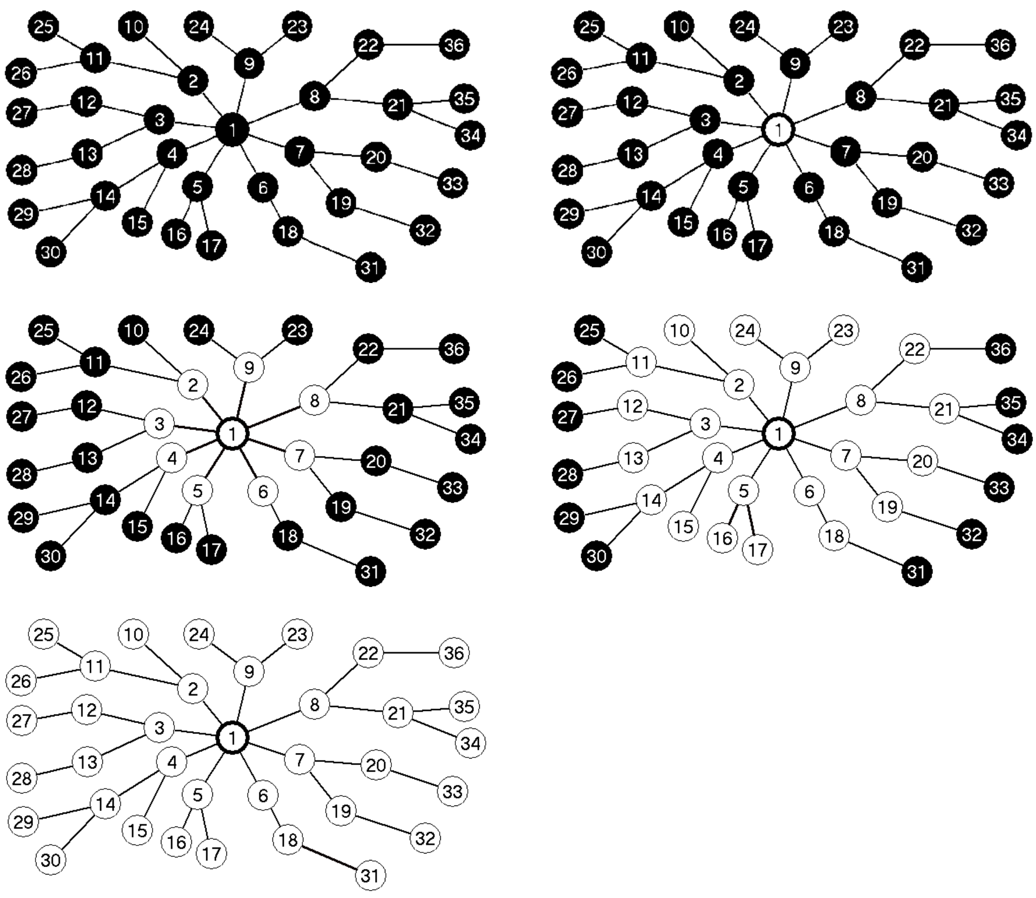
A bejárási algoritmusok feladata általában a gráf csúcsainak végiglátogatása valamilyen stratégia szerint. A bejárás gyakran azért hajtjuk végre, mert adott tulajdonságú csúcsot keresünk a gráfban. Két bejárási algoritmust ismerünk meg, a szélességi és a mélységi bejárást. Nézzük először a *szélességi bejárást*, néhány fejezettel később pedig a mélységi stratégiát ismertetjük.

23.1. A szélességi bejárás stratégiája

A bejárási stratégiákról megkapóan szemléletes leírást találhatunk a *Rónyai Lajos*, *Ivanyos Gábor* és *Szabó Réka* szerzőhármas *Algoritmusok* című könyvében. Szabadon és tömören idézzük „az öreg városka girbe-gurba utcáin bolyongó kopott emlékezetű lámpagyújtogató esetét”, illetve, most csak az egyik módszert arra, hogy végül az összes köztéri lámpa világítson.

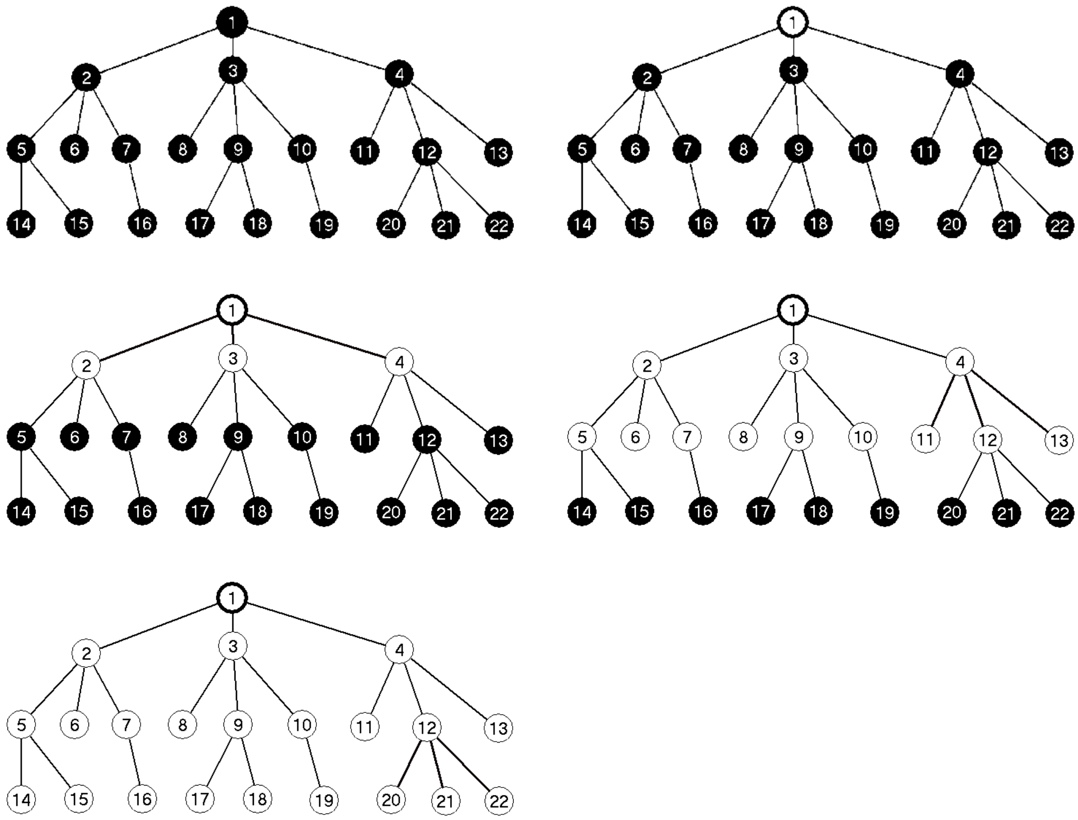
Az egyik eljárás szerint a lámpagyújtogatót egy este nagyszámú barátja elkíséri a városka főterétre, ahol együtt meggyújtják az első lámpát. Utána annyi felé oszlanak, ahány utca onnan kivezet. A különvált kis csapatok meggyújtják az összes szomszédos lámpát, majd tovább osztódnak. A városka lámpáit ilyen módon széltében terjeszkedve érik el.

Ha fentről néznénk a várost, ahogy kigyulladnak a lámpák, azt látnánk, hogy a középpontból a város széle felé egyre nagyobb sugarú körben terjed a világosság. Ez a szemléletes alapelve a *szélességi bejárásnak*. A szélességi stratégiát a 23.1. ábrán látható néhány pillanatfelvétel illusztrálja.



23.1. ábra. Szélességi bejárás egy gráfon

A 7. fejezetben ismertettük a *bináris fák szintfolytonos bejárásának* algoritmusát, amelyet egy *sor* adatszerkezet segítségével sikerült megvalósítani. A 23.2. ábra a fa adatstruktúra a bejárásának néhány fázisát szemlélteti. Látható, hogy a szintfolytonos bejárás a szélességi bejárás *speciális* esete fákra alkalmazva, a fa gyökerét véve kezdőcsúcsnak.



23.2. ábra. Szintfolytonos bejárás egy fán

23.2. A szélességi bejárás algoritmusa

Feladat: Adott egy $G = (V, E)$ irányított vagy irányítás nélküli, véges gráf. Írjuk ki a csúcsokat egy $s \in V$ kezdőcsúcstól való távolságuk *növekvő* sorrendjében. Minden csúcsra jegyezzük fel a *kezdőcsúcstól való távolságát*, és a hozzá vezető (egyik) legrövidebb úton a *megelőző csúcsot*. Az azonos távolságú pontok egymás közötti sorrendjére nincs megkötés, az legyen tetszőleges.

Elsőként vezessük be a csúcsok *s-től való távolságát*, mint a bejárás alapját képező értéket. Legyen $G = (V, E)$ gráf és $s, u \in V$ csúcsok, és $s \rightsquigarrow u = \langle v_0, v_1, \dots, v_k \rangle$ út, ahol $s = v_0$ és $u = v_k$. Az út hossza legyen az út mentén érintetett élek száma, azaz $|s \rightsquigarrow u| = |\langle v_0, v_1, \dots, v_k \rangle| - 1 = k$. Az u csúcs s-től való távolsága legyen az $s \rightsquigarrow u$ utak közül a legrövidebbnek az élszáma, azaz $d(s, u) = \min\{|s \rightsquigarrow u|\}$. Ha nincs $s \rightsquigarrow u$ út a gráfban, akkor legyen $d(s, u) = \infty$.

Az algoritmus *elvét* az előzőekben már láttuk, most foglaljuk össze röviden, lépésenként.

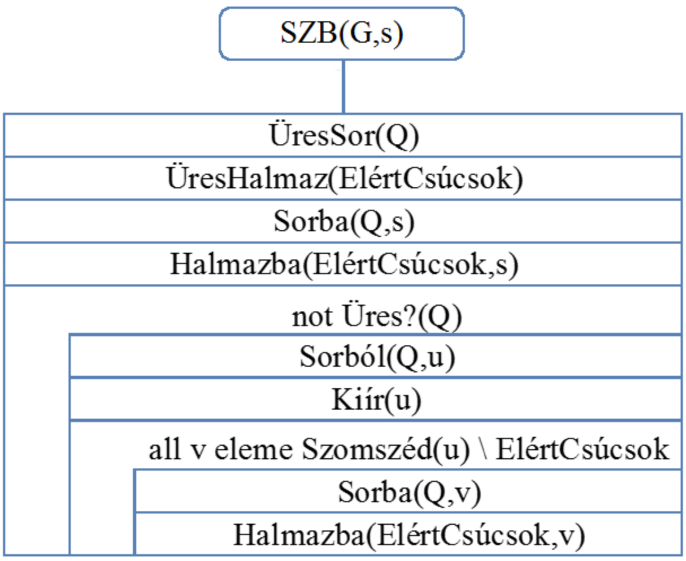
1. Először elérjük a kezdőcsúcsot.
2. Ezután elérjük a kezdőcsúcstól 1 távolságra lévő csúcsokat (a kezdőcsúcs szomszédjait)
3. Majd elérjük a kezdőcsúcstól 2 távolságra lévő csúcsokat (a kezdőcsúcs szomszédjainak a szomszédjait), és így tovább.
4. Ha egy csúcsot egyszer már elértünk, akkor a későbbi odajutásoktól el kell tekinteni.

Hogyan tudjuk biztosítani a fenti elérési sorrendet? Az elérési sorrendnél azt kell figyelembe venni, hogy amíg az összes, a kezdőcsúcstól k (≥ 0) távolságra lévő csúcsot ki nem írtuk, addig nem szabad k-nál nagyobb távolságú csúcsokat kiírni, és amikor egy k távolságú csúcsot kiírunk, addigra már az összes k-nál kisebb távolságú csúcsot ki kellett írnunk. Egy $k+1$ távolságú csúcs biztosan egy k távolságú csúcs szomszédja (az egyik legrövidebb úton a megelőző csúcs biztosan k távolságra van a kezdőcsúcstól), tehát a $k+1$ távolságú csúcsokat a k távolságú csúcsok szomszédai között kell keresni (nem biztos, hogy az összes szomszéd $k+1$ távolságú, lehet, hogy egy rövidebb úton már elértük).

Használjunk *sor* adattípust és biztosítsuk azt az invariáns tulajdonságot, hogy a sorba csak k vagy $k+1$ távolságú csúcsok lehetnek az elérésük sorrendjében, amely egyben az s-től való távolságuk szerint növekedő sorrendnek is megfelel. Ameddig ki nem ürül a sor, vegyünk ki az első elemet, írjuk ki és terjesszük ki, azaz a még "meg nem látogatott" szomszédjait érjük el és rakjuk be a sorba.

Az állíthatjuk, hogy említett ciklust végrehajtva, teljesül a fenti invariáns és a csúcsokat távolságuk sorrendjében érjük el és írjuk ki. Az állítás könnyen igazolható teljes indukcióval (k távolságú csúcsok a sorban megelőzik a $k+1$ távolságú csúcsokat, és a sorban lévő csúcsok távolságának legfeljebb 1 az eltérése).

- $k=0$: Induláskor berakjuk a sorba a kezdőcsúcsot, ami 0 távolságra van. Ki is vesszük rögtön ezt a csúcsot a sorból és kiírjuk, majd a szomszédjait, az 1 távolságra lévő csúcsokat rakjuk be a sorba.
- $k \to k+1$: Indukciós feltevés szerint a sorba csak k és $k+1$ távolságú csúcsok vannak távolságuk szerint növekvően, és a sor invariánsa az, hogy a korábban bekerült csúcsot korábban veszi ki. Az összes k távolságú csúcsot kiterjesztjük, mielőtt egy $k+1$ távolságú csúcsot kivennénk, és amíg ki nem vettük az összes k távolságút, addig csak olyan csúcsokat helyezünk el a sorban, amelyek $k+1$ egységre vannak a kezdőcsúcstól. Csak az első $k+1$ távolságú csúcs kivételénél kerülhet a sorba egy $k+2$ távolságú, de addigra már az összes $k+1$ távolságú csúcs bekerül a sorba, mert az összes k távolságra lévő csúcsot kiterjesztettük. Tehát a $k+1$ távolságú csúcsok megelőzik a $k+2$ távolságúakat, és a távolság különbség is mindig legfeljebb 1 marad.



23.3. ábra. A szélességi bejárás algoritmusa (ADS szint)

Ha egy csúcsot *egyszer* már elértünk, akkor *bejártnak* tekintjük, és nem akarjuk később újra elérni. Használjunk egy olyan halmazt, amelybe az elért csúcsokat rakjuk; kezdetben csak a kezdőcsúcsot tartalmazza. Amikor egy csúcsot először elérünk, helyezzük a halmazba, és minden csúcs kiterjesztésénél csak azokat a szomszédokat tekintsük (helyezzük a sorba), amelyeket még nem értünk el, azaz nincsenek benne az elért halmazban. Így minden csúcsot csak egyszer érünk el és helyezünk a sorba). Ezzel együtt természetesen minden csúcsot csak egyszer írunk ki.

Mivel a csúcsok száma véges és minden csúcsot legfeljebb egyszer érünk el és terjesztünk ki. Tehát a bejárás biztosan terminál. A teljes algoritmus a 23.3. ábrán látható.

Az algoritmusban használt $\text{Szomszéd}\,(u)$ absztrakt függvény az $u \in V$ csúcs szomszédjainak halmazát adja meg. A többi jelölés magától értetődik.

23.3. Az algoritmus szemléltetése

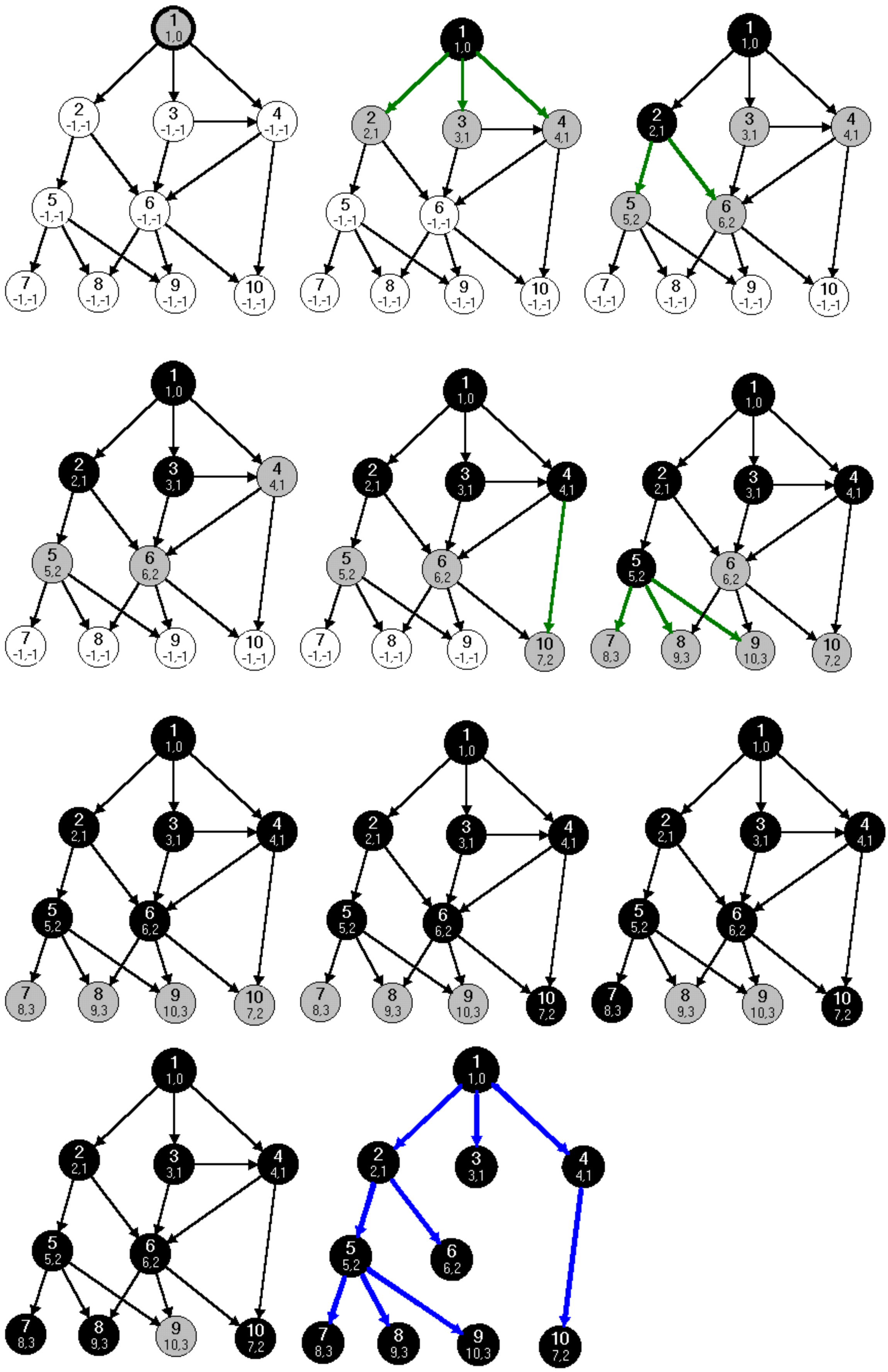
Az absztrakt szinten bevezetett halmazt, amelybe az elért csúcsokat helyezzük, most a csúcsok *színezésével* valósítjuk meg. Elegendő lenne, hogy a csúcsoknak csak két állapotát különböztessük meg, azonban a működés még szemléletesebbé tétele céljából *három színt* alkalmazunk.

1. Amikor egy csúcsot még nem értünk el legyen *fehér* színű. Induláskor a kezdőcsúcs kivételével minden csúcs ilyen ($u \notin Q$ és $u \notin \text{ElértCsúcsok}$).
2. Amikor egy csúcsot elérünk és beillesztjük a sorba, színezzük *szürkére*. A kezdőcsúcs induláskor ilyen ($u \in Q$ és $u \in \text{ElértCsúcsok}$).
3. Amikor egy csúcsot kivettünk a sorból és kiterjesztettük (elértük a szomszédjait), a színe legyen *fekete* ($u \notin Q$ és $u \in \text{ElértCsúcsok}$).

Az általános leírás absztrakt szintjén egy csúcs szomszédjainak az *elérési sorrendjéről* (a $\text{Szomszéd}\,(u) \setminus \text{ElértCsúcsok}$ feldolgozási sorrendjéről) nem tettünk fel semmit. A szomszéd csúcsok elérése nem egyértelmű, tehát egy nem determinisztikus algoritmust kaptunk. (A gyakorlatban néha megköveteljük a szomszéd csúcsok elérésének egyértelmű sorrendjét, azért, hogy az algoritmus működése kiszámítható és ellenőrizhető legyen. Általában, a szomszédokat a csúcsok címkéje szerint növekedően rendezve veszi sorra az algoritmus.)

A 23.4. ábrán tanulmányozható a szélességi keresés algoritmusa lépésenként. A csúcsokra, a címkén kívül, felírunk két pozitív egész számot. Az első szám megadja, hogy az illető csúcsot *hányadikként* írnánk ki, a második szám pedig a *kezdőcsúcstól való távolságot* tartalmazza. Kezdetben legyenek -1 extremális értékűek. A kezdőcsúcs legyen az 1-es címkéjű csúcs. Kezdetben minden csúcs fehér kivéve a 1-es csúcsot, amelynek szürke a színe. Kezdetben, a sorban is csak az 1-es csúcs van. Az első lépésben kivesszük a sorból az 1-es csúcsot, majd kiterjesztjük, azaz elérjük az 1-es csúcs még fehér szomszédjait (2, 3, 4), amelyeket szürkére színezünk, és bedobunk a sorba. Az 1-es csúcsot kiterjesztettük, ezután feketére színezzük. Figyeljük meg, hogy a sor a szürke csúcsokból áll, az elérési szám (első szám) szerint rendezve. Mindig azt a szürke csúcsot terjesztjük ki, amelyiknek az elérési száma a legkisebb, mivel ez a csúcs került be legkorábban a sorba.

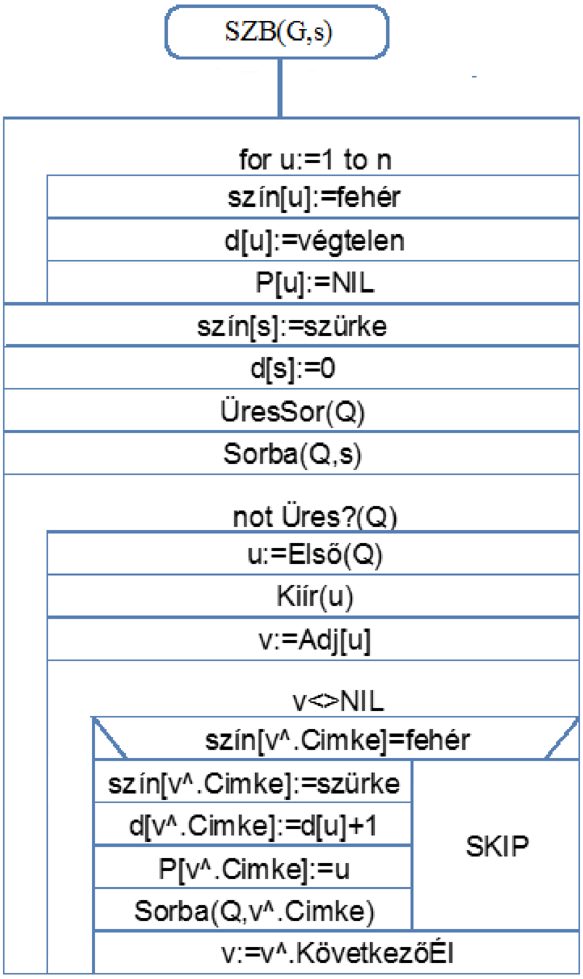
Az utolsó állapotban berajzoltuk a kezdőcsúcsból az illető csúcsba vezető élsorozatot, amely az algoritmus által felderített *legrövidebb utat* alkotja. A csúcsok és a berajzolt élek alkotta részgráf egy kezdőcsúcs gyökerű fát alkot, amelyben minden csúcs a legrövidebb úton érhető el. A fát a gráf *szélességi (feszítő) fájának* nevezzük. Tehát $F = (V', E')$ gráf a $G = (V, E)$ gráf szélességi fája, ha V' elemei az s-ből elérhető csúcsok, $E' \subseteq E$ és $\forall v \in V'$ csúcsra pontosan egy egyszerű út vezet s-ből v-be, és ez az út egyike az s-ből v-be vezető legrövidebb G-beli utaknak.



23.4. ábra. A szélességi bejárás lépésenkénti végrehajtása

23.4. Az algoritmus megvalósítása reprezentáció szinten

Tekintsük az algoritmus megvalósítását egy *éllistával* (másik elnevezéssel: *szomszédsági listával*) reprezentált gráfon (23.5. ábra). A csúcsok színét a csúcsokkal indexelt $szín\ [1 \dots n]$) tömbben tároljuk. További feladatunk a csúcs *s-től való távolságának*, és a hozzá vezető úton a *megelőző csúcsnak* az eltárolása. Ezt a $d[1 \dots n]$ és a $P[1 \dots n]$ tömbben tesszük meg. Az értékeket akkor ismerjük, amikor elérjük a csúcsot, vagyis amikor szürkére színezzük, így ekkor írjuk azokat a tömbbe. Kezdetben legyen minden csúcs végtelen távolságra a kezdőcsúcstól, és ha nincs a gráfban $s \rightsquigarrow u$ út, akkor u távolsága a startcsúcstól végtelen is marad.



23.5. ábra. A szélességi bejárás algoritmusa az éllistás reprezentációra

Az algoritmust a 23.4. ábrán bemutatott gráfon lefuttatva, a következő eredményeket kapjuk: $d[1..10] = [0, 1, 1, 1, 2, 2, 3, 3, 3, 2]$ és $P[1..10] = [NIL, 1, 1, 1, 2, 2, 5, 5, 5, 4]$. Látható, hogy a $P[1..10]$ tömb tartalmából könnyen előállítható a szélességi fa, illetve bármely csúcsra kiírható a kezdőcsúcsból hozzá vezető legrövidebb út.

23.5. Műveletigény

A műveletigényt *összefüggő* gráfra határozzuk meg. Feltesszük tehát, hogy a bejárás megadott kezdőcsúcsából a minden gráf *minden csúcsa elérhető*. Az algoritmus az inicializáló lépés során minden csúcsnak beállítja a színét, valamint a d és P tömbbeli értékét. Ez a csúcsok számával arányos műveletigényt ad, azaz $\Theta(n)$.

Éllistás ábrázolás esetén minden csúcsot (amelybe a feltevés szerint vezet él) *pontosan egyszer* teszünk be a sorba és onnan ki is vesszük, hiszen végül kiürül a sor. Ez $\Theta(n)$ műveletigényt jelent.

Amikor a sorból kivesszük a csúcsot, végignézzük az éllistáján lévő csúcsokat. Mivel minden csúcs pontosan egyszer kerül be a sorba és onnan ki is vesszük, ezért minden éllistán *egyszer* megyünk végig, tehát összességében a gráf *minden éllistán* áthaladunk egyszer. Az éllisták együttes hossza e, így ennek műveletigénye $\Theta(e)$. (Megjegyezzük, hogy ha egy él olyan szomszédhoz vezet, amelyet már korábban elért a bejárás, akkor ez az él csak egy kérdés feltételére készteti az algoritmust, hiszen a bejárt szomszédot már nem kell feldolgozni. Ez minimális költség, de az elméleti tisztaság érdekében nem lehet figyelmen kívül hagyni.)

Összesítve azt kapjuk, hogy egy összefüggő gráf *szélességi bejárásának műveletigénye éllistás reprezentáció* esetén:

$$T(n) = \Theta(n) + \Theta(e) = \Theta(n + e)$$

Csúcsmátrixos ábrázolás esetén egy csúcs szomszédjainak a vizsgálata a gráfot reprezentáló mátrix egy n hosszú sorának a végigjárását igényli. Ezt az összes csúcsra tekintetbe véve kapjuk a *szélességi bejárás műveletigényét a mátrixos ábrázolás* mellett:

$$T(n) = \Theta(n) + \Theta(n \cdot n) = \Theta(n^2)$$

A továbbiakban külön nem hangsúlyozzuk, hogy e-vel arányos műveletigény csúcsmátrixos ábrázolás esetén mindig n^2-el arányos műveletet jelent.