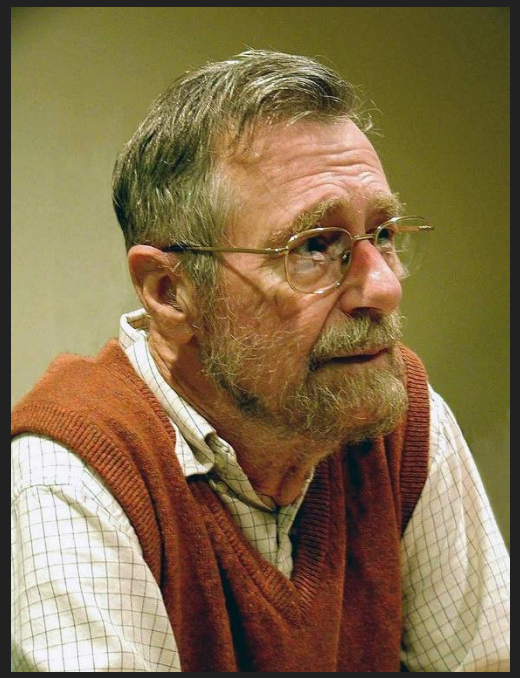
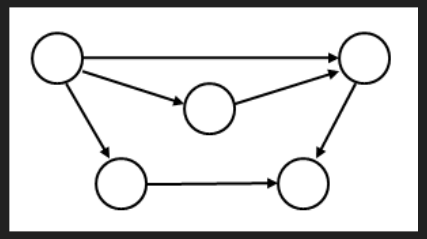
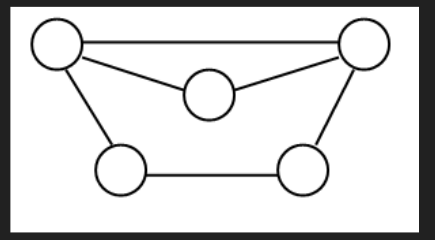
Dijkstra algoritmus működése MAR környezetre

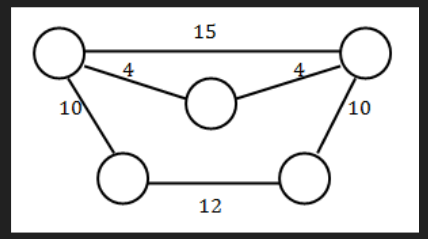


Edsger Wybe Dijskrta (1930-2002)

- Az algoritmust Edsger Wybe Dijskrta, holland matematikus és informatikus alkotta meg 1956-ban.

Általános ismertető

- **Mohó algoritmus, ágens**
- **Lényege**: gráfban (irányított  vagy irányítatlan )
- gráfkeresés
- **Egy pontból induló**



a legrövidebb út megtalálása adott csúcsból

- **Feltétel**: élek súlya **nemnegatív**
- Minden pontra megadja a legrövidebb utat a kezdőcsúcsból nézve, majd (algoritmus megírásától függően) tárolja ezeket az utakat.

A legrövidebb út „gráf”-elmélete

Definíció (legrövidebb út):
Legrövidebb út alatt a gráfelméletben egy minimális hosszúságú utat értünk egy gráf két különböző ***u*** és ***v*** csúcsa között. Amennyiben a gráfunk éleihez **nem tartoznak súlyok**, akkor ez egyet jelent egy olyan úttal ***u*** és ***v*** csúcs között, amelyben a **legkevesebb él szerepel**.
Ha **vannak súlyok** a gráf élein, akkor pedig olyan útról beszélünk, amelynek élein szereplő **súlyok összege minimális.** Vagyis, ha adott egy $G = (V;E)$ gráf a $k(f); f \in E$ élsúlyokkal, akkor:

$$d(u,v) = min \sum_{f \in P} k(f)$$

Ahol *k(f)* a költségfüggvény, értékkészlete a valós számok halmaza, és az egészet az élekre € szokás értelmezni (*megj.: léteznek csomópont súlyozású gráfok is, itt az élek ($k(f_{é})$) + csomópontok ($k(f_{cs})$) súlyozása az adott útvonalon,…* Adja a költségfüggvény értékét az *d(u,v)* útvonalra:

$$d(u,v)=min\Sigma(k(f_{é})+k(f_{cs}))$$

*Ha: u=v → d(u,v)=0; ha u≠v → d(u,v)= az „u” „v” szakasz **súlya**, ha nincs út „u” és „v” között d(u,v)=∞*

Legrövidebb utak:

1. **Legrövidebb út egy kiinduló pont és az összes többi pont között:** meg szeretnénk találni az összes $v \in V$ csúcshoz egy adott $s \in V$ kezdőcsúcsból odavezető legrövidebb utat. (*Dijsktra, Bellmann-Ford*)

2. **Legrövidebb út két különböző csúcs között:** keressük egy adott *u* csúcsból egy adott *v* csúcsba vezető egyik legrövidebb utat.

3. **Legrövidebb út egy végpont és az összes többi pont között** (ez az 1. megfordítása).

4. **Legrövidebb út az összes csúcspár között:** keressük az összes u és v csúcspárra egy u -ból a v csúcsba vezető legrövidebb utat. Természetesen akadhat olyan legrövidebb út probléma, amelyben előfordulnak negatív élek. *(Floyd, Warshall)*

Dijkstra algoritmus és jellegzetességei

- A **Dijkstra-algoritmus** , amivel irányított vagy irányítás nélküli gráfokban lehet megkeresni a legrövidebb utakat egy adott csúcspontból kiindulva. Az algoritmust Edsger Wybe Dijkstra holland informatikus fejlesztette ki.
- Az algoritmus inputja egy **súlyozott** *G* gráf és **„*s*”** a *G* gráf egy csúcsa. Az **„*s*”** csúcs az út kiindulási pontja. Jelöljük ***V***-vel a *G* gráf csúcsainak a halmazát, és legyen (u,v) a *G* gráf *u*-t *v*-vel összekötő éle, ahol $(u, v) \in V$ a gráf csúcsai. Jelöljük E-vel a *G* gráf éleinek a halmazát. Az élekhez rendelt súlyokat a ***w***: $E \rightarrow [0,\infty]$ súlyfüggvény adja meg, tehát w(u,v) az (u,v) él súlya.
- **Az élekhez rendelt költségeket tekinthetjük a két csúcs közötti távolság általánosításának**. A **START (s)** és **CÉL (t)** csúcsok közötti út költsége az **úton lévő élek költségének az összege**. Adott s és *t* ***V***-beli csúcsokra az algoritmus megkeresi a legkisebb költségű **„*s*”**-ből ***t***-be vezető utat (azaz a legrövidebb utat). Az algoritmus használható arra is, hogy az **adott pontból kiindulva** a gráf **összes többi pontjába** vezető legrövidebb utakat megkeressük (legrövidebb utak fája).

Az algoritmus működése

Feladat: Adott egy *G=(V,E)* élsúlyozott, irányított vagy irányítás nélküli, **negatív élsúlyokat nem tartalmazó,** véges gráf. Továbbá adott egy $s \in V$ forrás (kezdőcsúcs). Határozzuk meg, $\forall v \in V$ csúcsra, *s*-ből *v*-be vezető **legrövidebb utat és annak hosszát**!

Az algoritmus elve:

Minden lépésben tartsuk nyilván az **összes csúcsra**, a forrástól az illető csúcsba vezető, **eddig talált l**egrövidebb utat (a már megismert módon a ***d*[1..*n*]** tömbben a távolságot, és ***P*[1..*n*]** tömbben a megelőző csúcsot).

1.) Kezdetben a távolság legyen a kezdőcsúcsra „**0**" a többi csúcsra „∞"

2.) Minden lépésben a nem **KÉSZ** csúcsok közül tekintsük az egyik **legkisebb távolságú** (d_{min}) csúcsot:

a.) Azt mondhatjuk, hogy ez a $v \in V$ csúcs már **KÉSZ**, azaz **ismert** a hozzá vezető legrövidebb út.

b.) A ***v***-t **terjesszük ki**, azaz ***v*** csúcs szomszédjaira számítsuk ki a (már ismert) ***v***-be vezető, és onnan egy kimenő éllel meghosszabbított út hosszát. Amennyiben ez jobb (**kisebb**), mint az illető szomszédba eddig talált legrövidebb út, akkor innentől kezdve ezt az utat tekintsük, az adott szomszédba vezető, eddig talált legrövidebb útnak. Ezt az eljárást szokás **közelítésnek (iteráció)** is nevezni.

Az algoritmus egy lépése

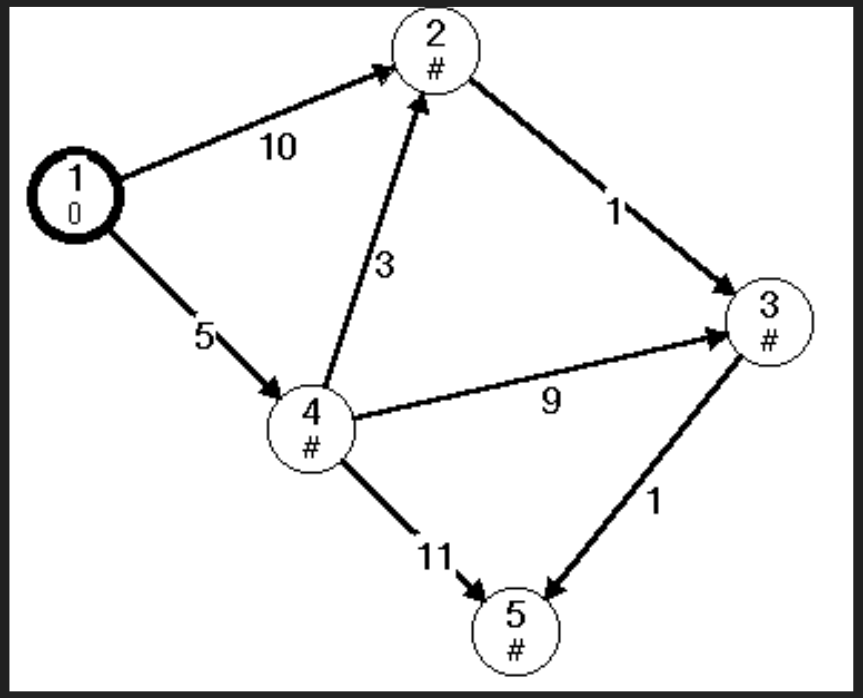
Kezdeti állapot: Kezdetben minden csúcs **N** állapotban van, $d(s) = 0$, a többi csúcsnak pedig még nincs $d(v)$ értéke (tekinthető úgy, hogy $d(v) = \infty$).
A csomópontok állapotai: ***L***-lezárt (KÉSZ), végleges értékkel, ***N***- nem lezárt (NEM KÉSZ), ez lehet azért mert nem volt még meglátogatva, vagy már meg volt látogatva, de értéke még nem véglegesített.

Az algoritmus egy lépése:
Az ***N*** állapotúak közül kiválasztjuk azt az ***u*** csúcsot, amire ***d(u)*** minimális (ha több ilyen is van, akkor tetszőlegesen választunk ezek közül egyet). Minden *u,v* élre megvizsgáljuk a $d(v)$ értékeket. Ha még nincs $d(v)$ érték vagy $d(u) + l(u,v) < d(v)$, akkor $d(v)$ új értéke $d(u) + l(u,v)$ lesz, továbbá *v* szülője *u* lesz. Ezután az *u,v* élt töröljük a gráfból. Ha már nincs *uv'* él, akkor *u* állapotát *L*-re változtatjuk és a következő lépésre ugrunk.
Az algoritmust addig folytatjuk, amíg ***t*** csúcs ***L*** állapotú nem lesz. Nem összefüggő gráf esetén az is előfordulhat, hogy az ***N*** állapotú csúcsok egyikének sincs (∞ -től különböző) d(v) értéke, ilyenkor is megállunk. $d(t)$ leálláskori értéke adja meg a legrövidebb *s,t* út hosszát.

megjegyzés. Ha a Dijkstra algoritmust addig futtatjuk, amíg az összes elérhető csúcs ***L*** állapotú lesz, akkor valójában nem csak *t-re*, hanem az összes *s*-ből elérhető ***v*** csúcsra meghatározza a legrövidebb *s,v* út hosszát.

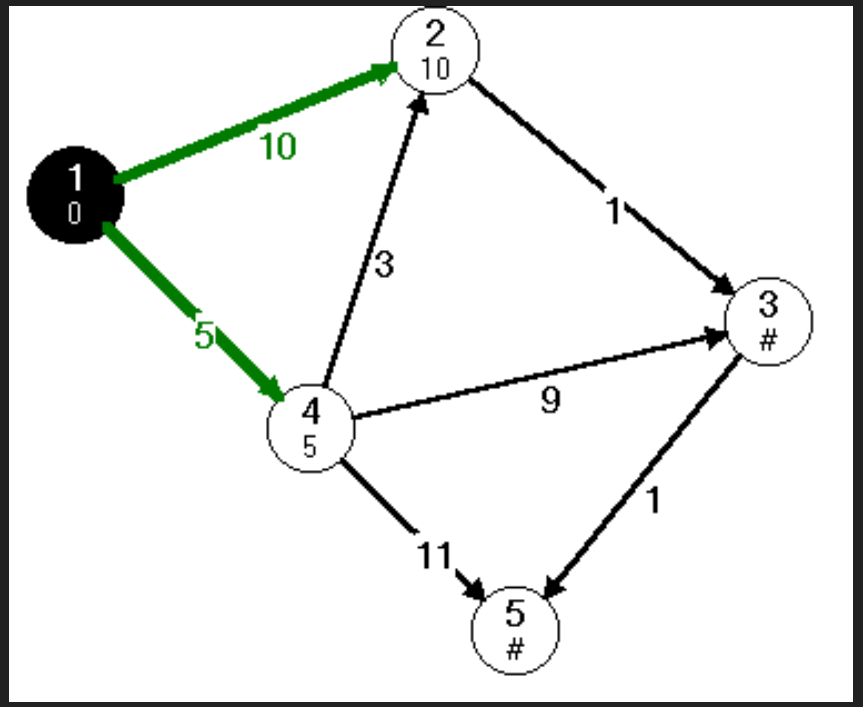
Az algoritmus lépésenkénti bemutatása - inicializálás

A gráfunk öt csomópontból áll, melyeket rendre: **1,2,3,4,5** – számmal nevezünk el, majd az identifikáció alá odaírjuk a *d(u,v)* távolságokat, amik kezdetben (inicializálás) a kiinduló csomópontnál (**1**), *d(1,1)=0*, míg a többi csomópontnál „∞”, (ami az ábrán: #). A cél csomópont az (**5**).



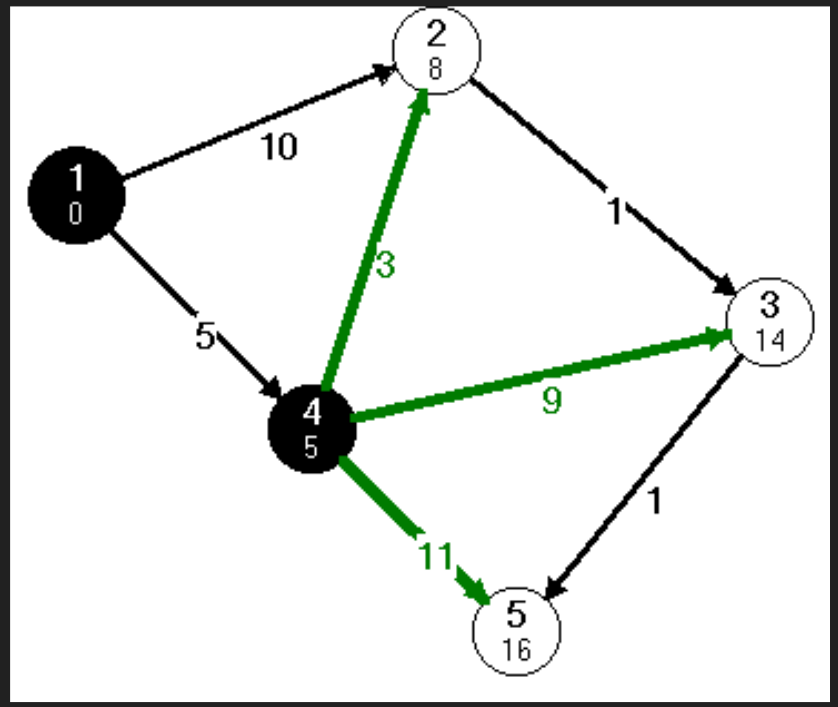
Az inicializáló lépés után a kezdőcsúcs 0, a többi csúcs végtelen súllyal szerepel az elsőbbségi sorban.

Az algoritmus lépésenkénti bemutatása – 1. lépés



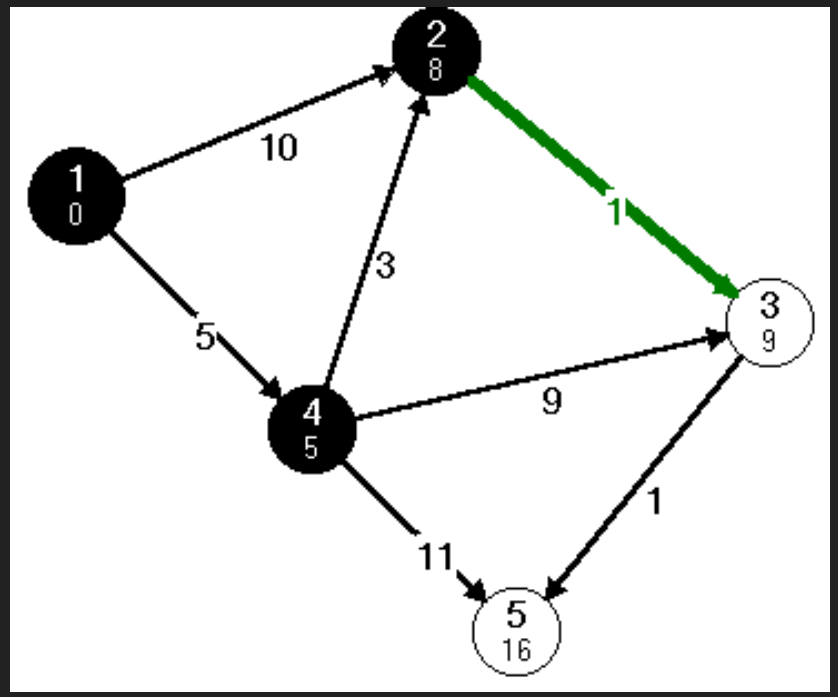
Az első lépésben kivesszük a prioritásos sorból az 1-es csúcsot (mivel az ő prioritása a legkisebb). Az 1-es csúcshoz már ki van számítva a legrövidebb út, tehát ez a csúcs már elkészült, színezzük feketére. Kiterjesztjük az 1-est, azaz a szomszédjaira kiszámítjuk az 1-esből kimenő éllel meghosszabbított utat. Ha ez javító él, azaz az 1-esen átmenő út rövidebb, mint az adott szomszédba eddig talált legrövidebb út, akkor a szomszédban ezt feljegyezzük (*d* és *P* tömbbe). Az ábrán kiemeltük a javító éleket.

Az algoritmus lépésenkénti bemutatása – 2. lépés



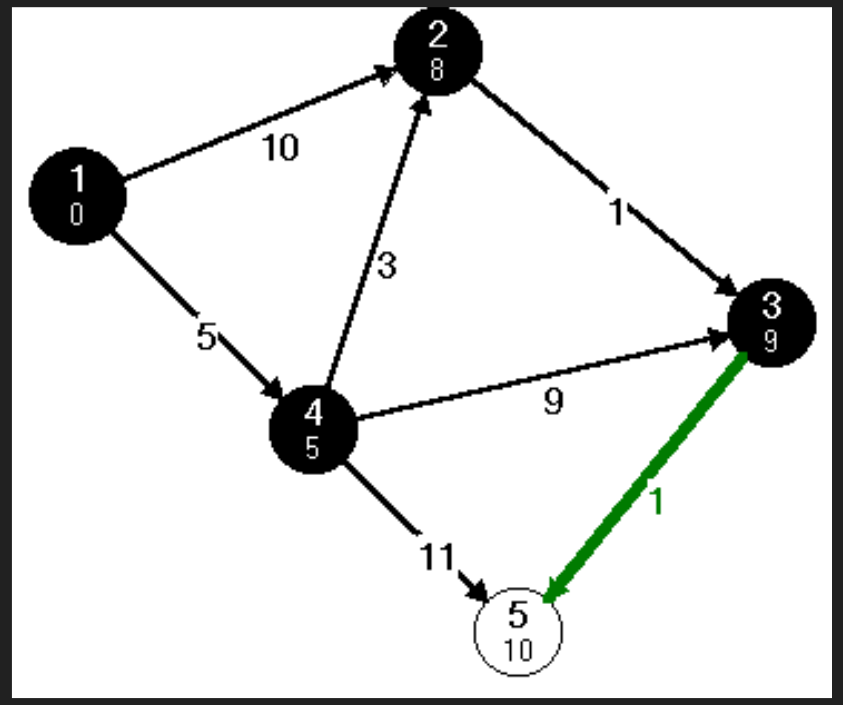
Megfigyelhető, hogy a 2-es csúcsba már korábban is találtunk 10 hosszú utat *1,2* , de a második lépésben, a 4-es csúcs kiterjesztésekor, találunk, a 4-es csúcson átmenő rövidebb 8 hosszú utat.

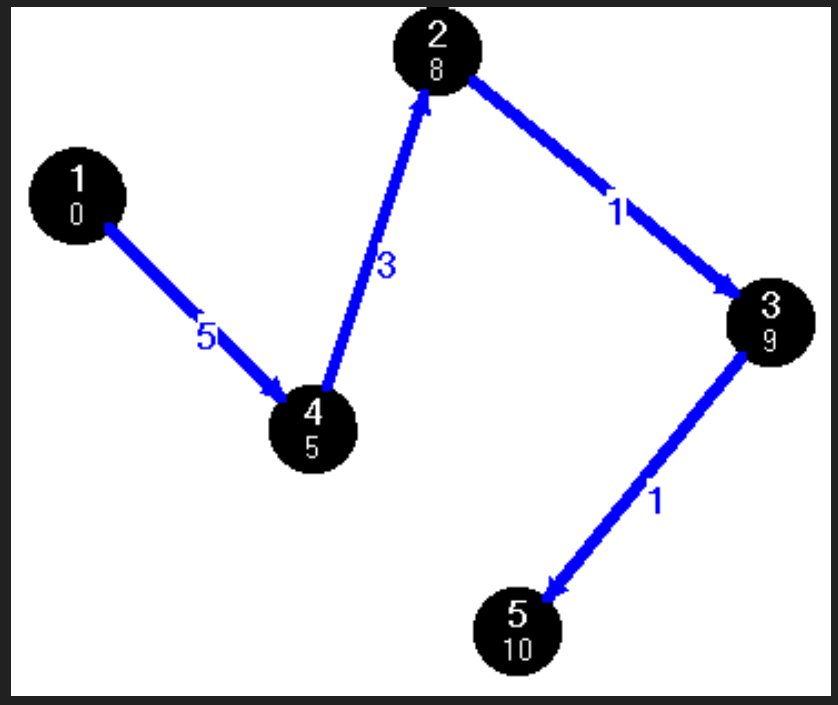
Az algoritmus lépésenkénti bemutatása – 3. lépés



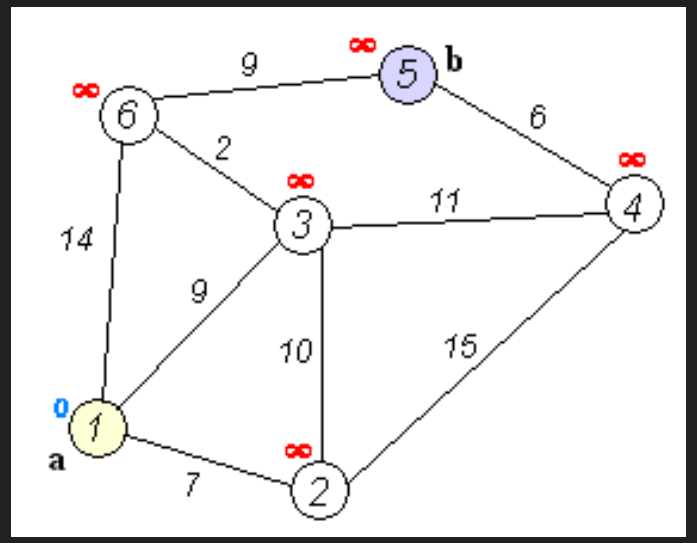
A 2-es csúcs kiterjesztésekor a 3-as csúcsba találtunk egy rövidebb utat.

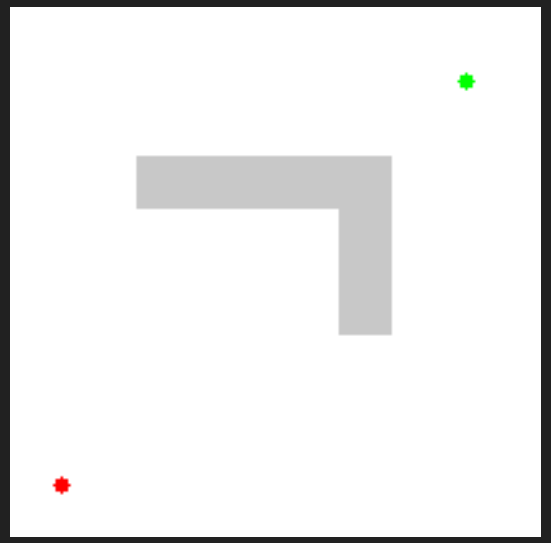
Az algoritmus lépésenkénti bemutatása – 4. lépés és a végeredmény





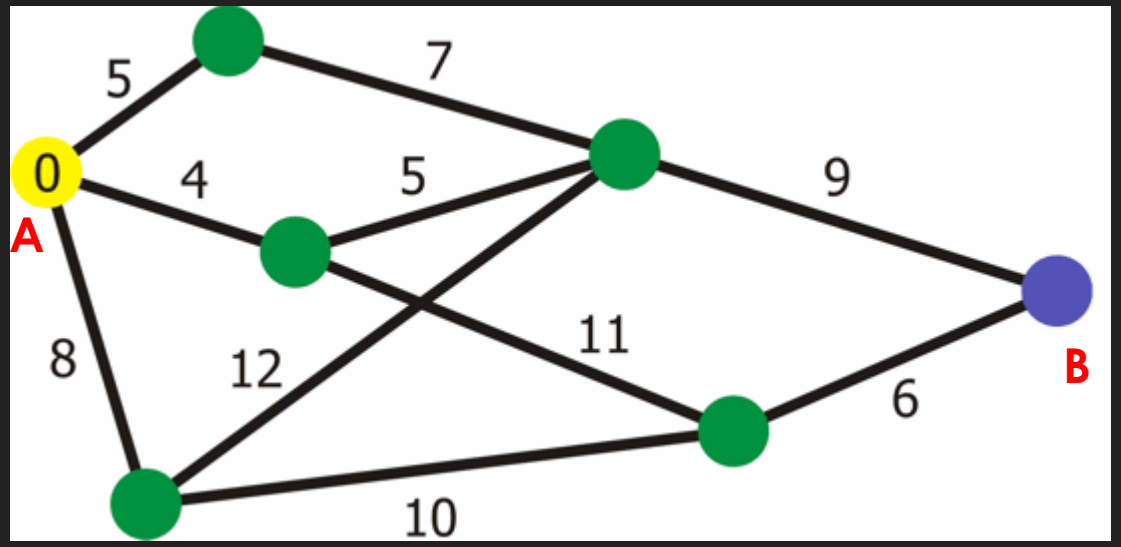
Dijkstra rövid példa





Minél vörösebbel van jelölve az adott pont, annál kisebb a költsége és egyre növekszik a zöld szín felé haladva. S startpontból haladunk a C célpont felé, ahol a sötétkék pontok akadályt (tereptárgy, fal) jelentenek, az ufózöld pedig a végső legkisebb költségűnek feltételezett útvonalat

A szimuláció működése



Az “***A***“ és “**B**“ közötti legrövidebb út megkeresése Dijkstra-algoritmussal. Az algoritmus mindig a legkisebb távolságú még meg nem látogatott csúcsot választja, majd megnézi, hogy ezen csúcson keresztül mekkora út megtételével tudna eljutni egyes szomszédjaihoz.

A csúcsot meglátogatottnak (**sárga az ábrán**) jelöli, ha végzett a csúcsból elérhető összes többi csomóponttal és ez a legkisebb költségű csomópont.

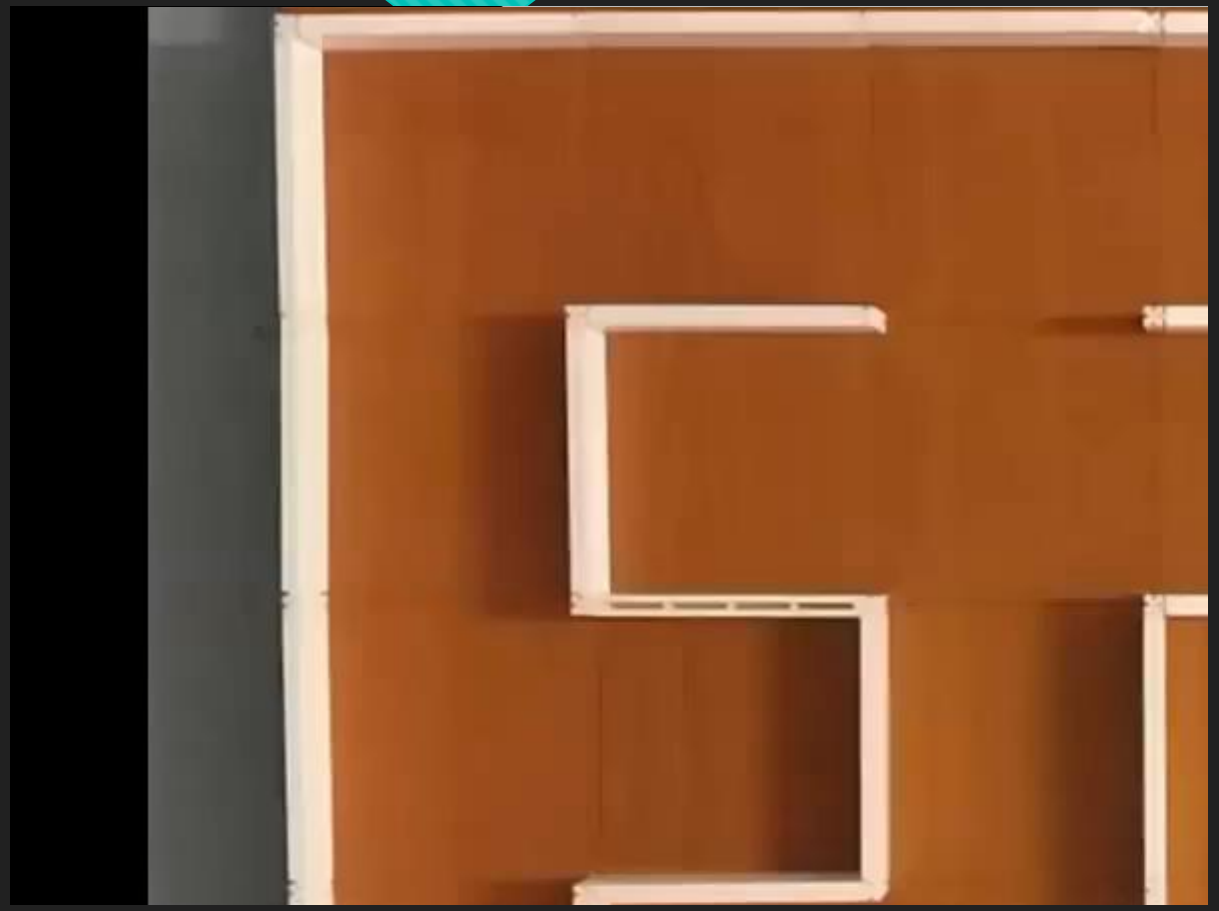
Az előző szimuláció működési pszeudokódja

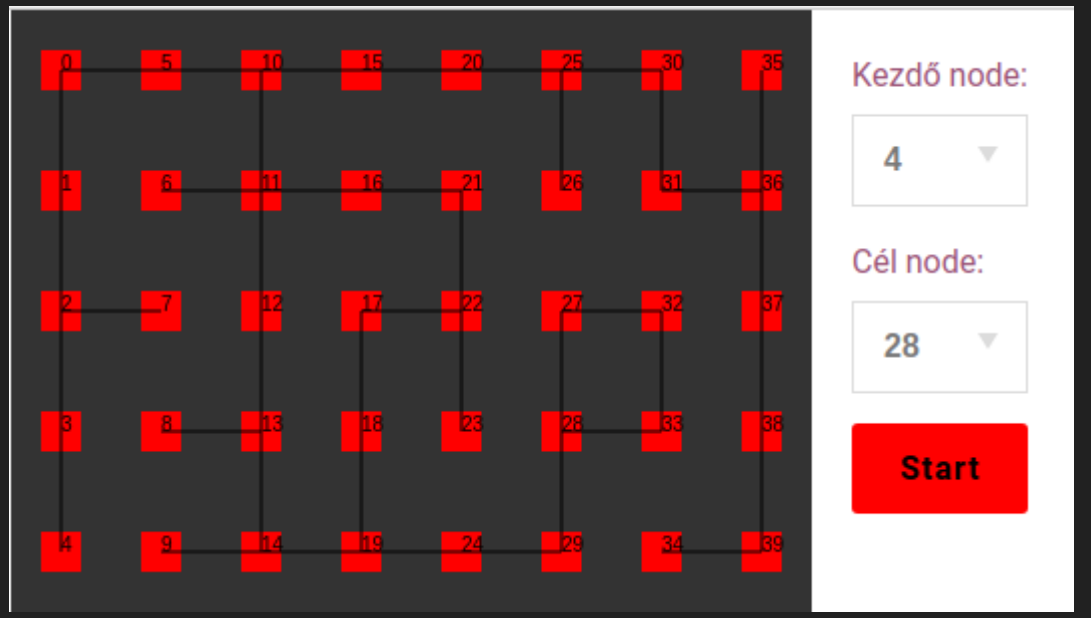
```
1  function Dijkstra(Graph, s):
2      for each vertex v in Graph:                 // inicializáció
3          dist[v] := infinity                     // kezdetben minden pont távolsága ismeretlen
4          previous[v] := undefined
5      dist[s] := 0                                // a source csúcsból a source csúcsba 0 út megtételével jutunk
6      Q := copy(Graph)                            // meg nem látogatott csúcsok halmaza
7      while Q is not empty:
8          u := extract_min(Q)                     // kivesszük a számunkra legjobb csúcsot a prioritási sorból
9          for each neighbor v of u:
10             alt = dist[u] + length(u, v)
11             if alt < dist[v]                    // ha ebből a csúcsból kedvezőbben juthatunk el v csúcsba,
12                 dist[v] := alt                  // akkor frissítünk
13                 previous[v] := u
```

Valós környezeti szimuláció

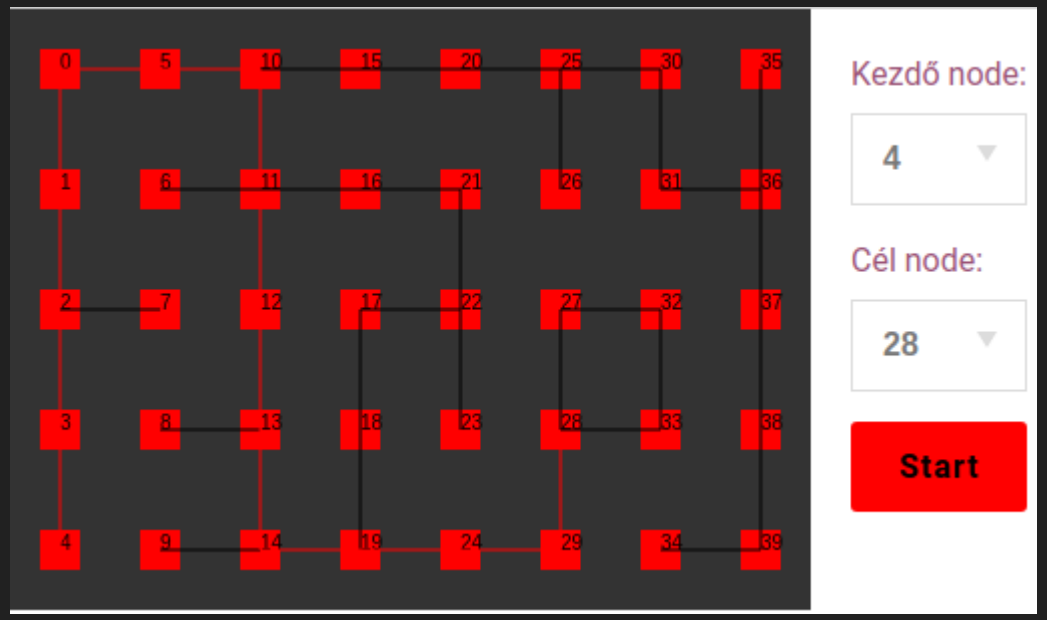
A következőkben egy ismeretlen környezetben mozgó ágens **feltérképezi** a környezetét, majd elkészíti a környezet **gráftérképét**, majd ezen a gráftérképen megkeresi s S (start) és C (cél) pozíciók közti **legrövidebb** útvonalat.

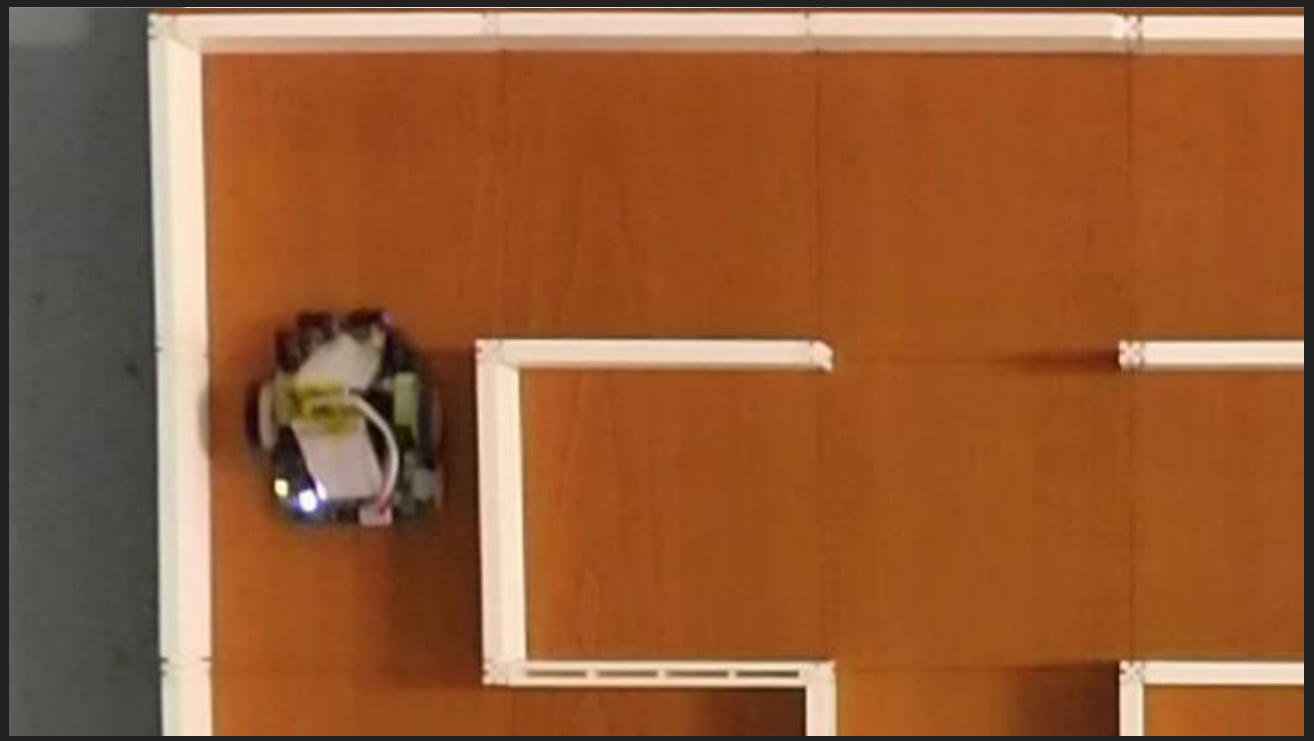
Egy rövid példa:
A munkaterület feltérképezése





A térkép alapján az S(4) → C(28) útvonal



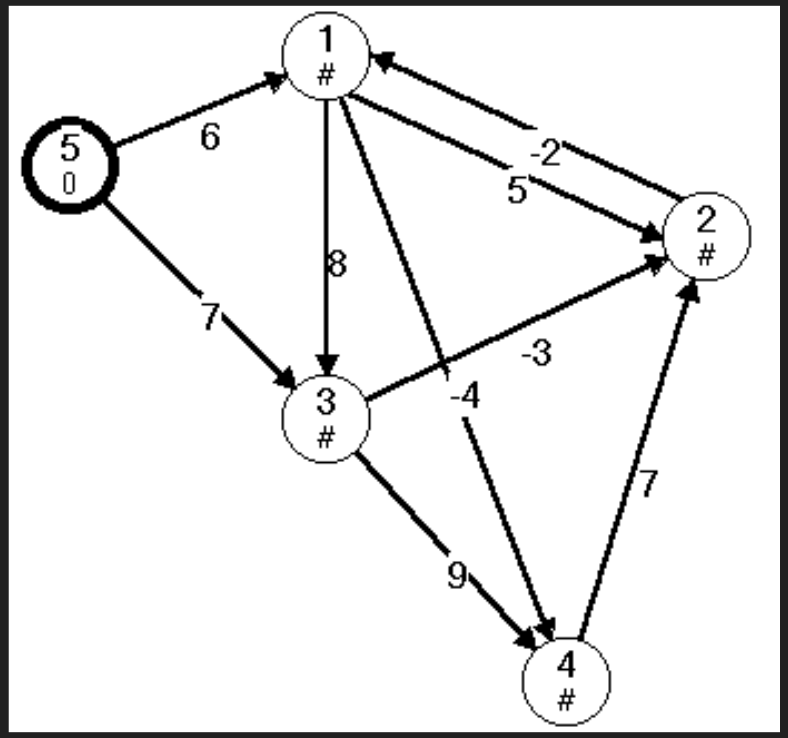


További kereső algoritmusok
Bellman-Ford algoritmus

Adott egy *G=(V,E)* élsúlyozott, irányított vagy irányítás nélküli, **negatív összköltségű irányított kört nem tartalmazó** véges gráf, továbbá egy $s \in V$ forrás (kezdőcsúcs). Határozzuk meg, $\forall v \in V$ csúcsra, *s*-ből *v*-be vezető **legrövidebb utat és annak hosszát**.!
(ezek az algoritmusok (a negatív élekre való tekintettel) csak speciális esetekben használatos valós, fizikai pályakeresésre a mobilrobotok világában.)

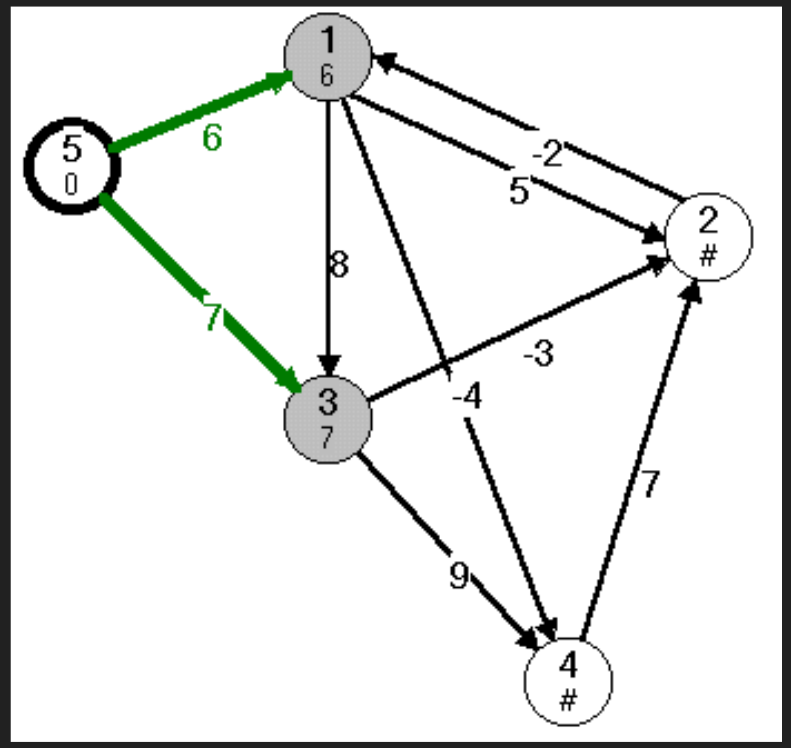
Az algoritmus elve:
Minden csúcsra, **ha létezik legrövidebb út**, akkor létezik **egyszerű legrövidebb út** is. Mivel a körök összköltsége nem negatív, így a kört elhagyva az út költsége nem nőhet. Egy ***n*** pontú gráfban, a **legnagyobb élszámú** egyszerű út élszáma, **legfeljebb *n-1*** lehet.
A Bellman-Ford algoritmus a Dijkstra algoritmusnál megismert **közelítés műveletét** végzi, azaz egy csúcson át a szomszédba vezető él mentén vizsgálja, hogy az illető él része-e a legrövidebb útnak, javító él-e. Egy menetben az összes élre megvizsgálja, hogy javító él-e vagy sem. Összesen ***n-1*** menetet végez.

Az algoritmus lépésenkénti bemutatása – inicializálás után



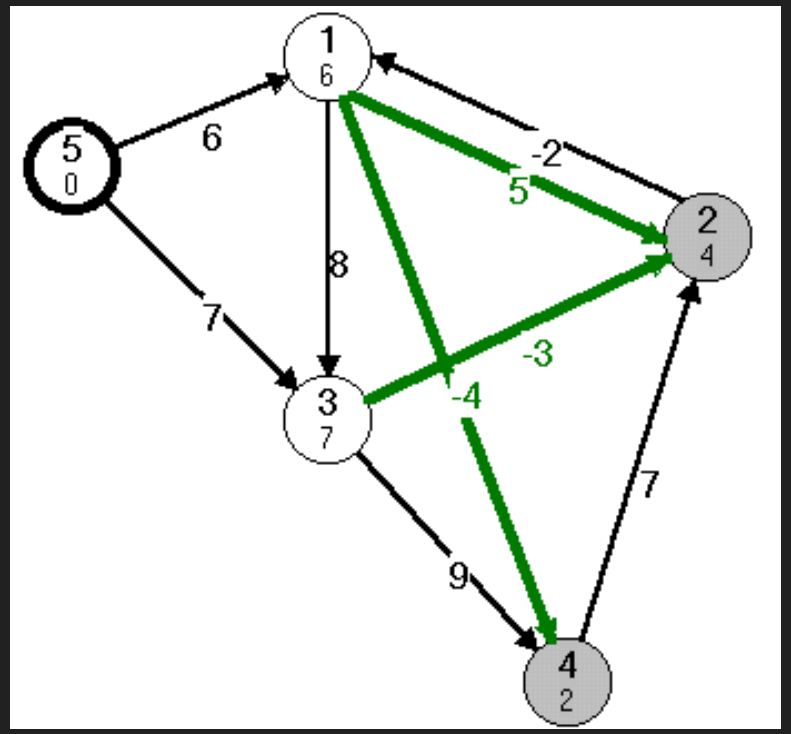
Az inicializáló lépés során beállítjuk a ***d*[1..*n*]** és ***P*[1..*n*]** tömb értékeit. A végtelen értéket most is '#' jellel jelöljük.

Az algoritmus lépésenkénti bemutatása – 1. lépés után



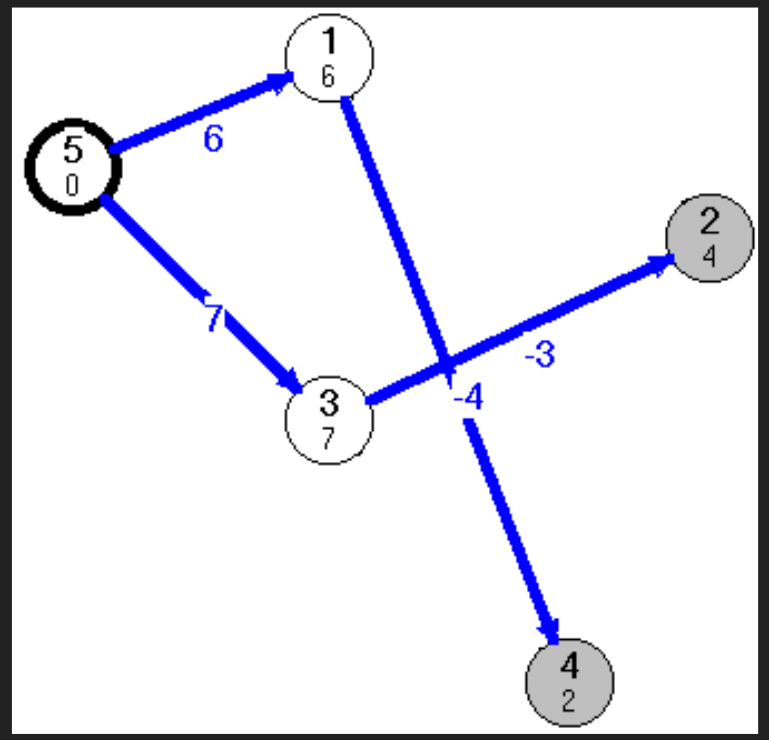
Az első 7 él *((1,2),(1,3),…,(4,2))* közelítésénél nem történik változás, mivel végtelen értékek növelésénél szintén végtelent kapunk, ami nem javít.
Csak két javító élt találunk.
Most állíthatjuk, hogy minden csúcshoz megtaláltuk az **s**-ből hozzá vezető, minimális költségű, **1** élszámú utat.

Az algoritmus lépésenkénti bemutatása – 2. lépés után



Az ábrán látható, mely csúcsokhoz találtunk javító élt.

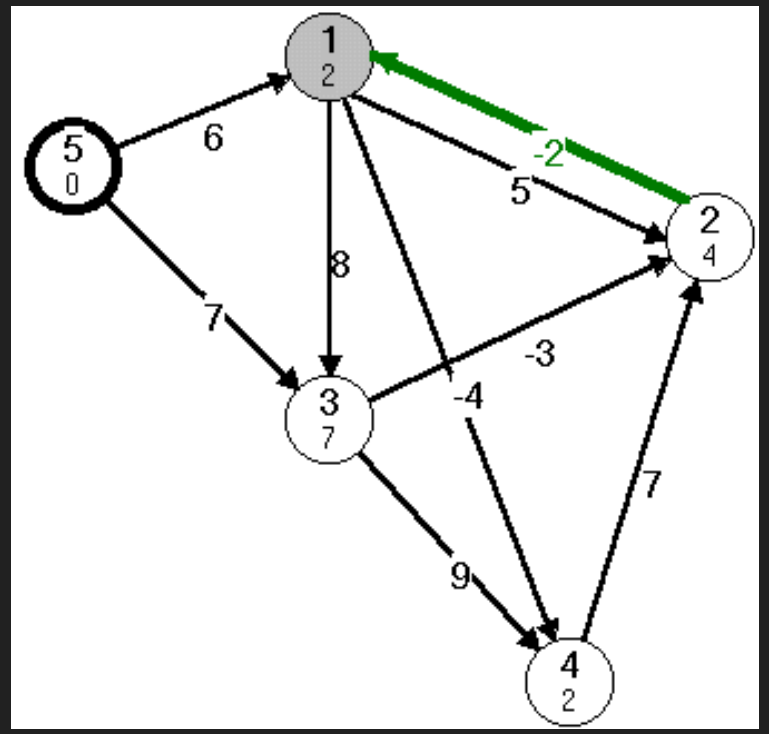
Az algoritmus lépésenkénti bemutatása – a 2. lépés fája



Minden csúcshoz meghatároztuk a legkisebb költségű, **1** vagy **2** élszámú utat.
Az ábrán látható, az egyes csúcsokba vezető, **1** vagy **2** élszámú legrövidebb utakból kialakult fa.
Ez a fa változhat, mivel lehet, hogy egy csúcsba el lehet jutni nagyobb élszámú olcsóbb úton is.

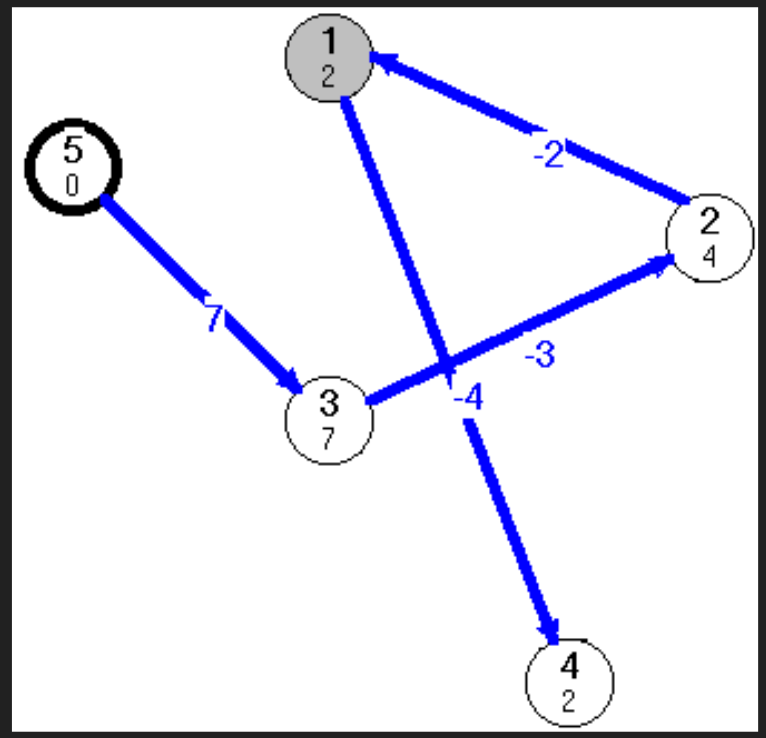
(vizsgáljuk tovább)

Az algoritmus lépésenkénti bemutatása – 3. lépés után



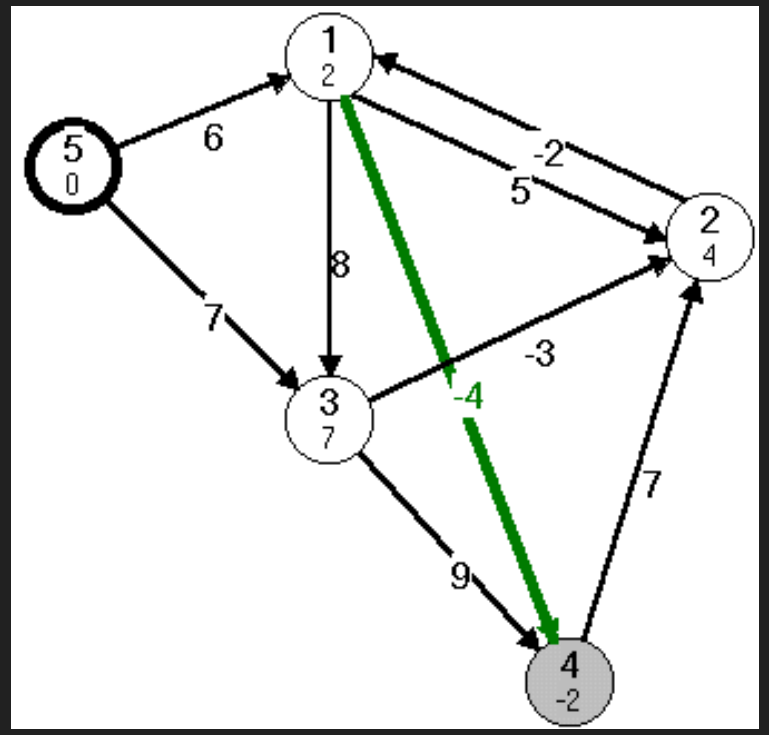
Az **1**-be olcsóbb **3** élszámú utat találtunk.

Az algoritmus lépésenkénti bemutatása – a 3. lépés fája



A fa változott mivel az **1**-be már nem **1** élszámú, hanem **3** élszámú, de rövidebb úton juthatunk el a kezdőcsúcsból.
A **4** megelőzője a korábban talált **1**-es, csak most nem **2** élhosszú úttal, hanem **4** élhosszúval **5,3,2,1,4** .
Mivel az (**1,4**) élt korábban dolgoztuk fel, mint (**2,1**) élt, így a **4**-es csúcsnál bejegyzett költség nem konzisztens a fával.

Az algoritmus lépésenkénti bemutatása – 4. lépés után



A fa már nem változik, csak 4-es csúcsnál bejegyzett költség veszi fel a helyes értéket.

A legrövidebb utak *minden csúcspárra*

Ez a módszer elterjedt a MÁR útvonalkeresésénél, mert egy algoritmusban letudjuk a különböző csúcspontokból induló ágensek optimális útvonalainak megtervezését. Természetesen az ütközési csomópontokat (tér-idő kontinuumban) itt is „szemaforokkal" kell ellátni.
A DIJKSTRA algoritmusnál mindegyik ágens lefuttatja a maga Dijkstra algoritmusát, majd további algoritmusnak kell megkeresni az ütközési csomópontokat (tér-idő kontinuumban) és „szemaforokat" elhelyezni az adott pontokban.

Ezekben az esetekben –minden csúcspár-, általában ***táblázatos*** (vagy **mátrix**) formában keressük a legrövidebb útvonalakat.

Legrövidebb utak minden csúcspárra, értelmezések - 1

Célunk egy gráf valamennyi rendezett csúcspárjára a két csúcs közti legrövidebb út megkeresése.
Csakúgy, mint korábban $G=(V,E)$ egy súlyozott irányított gráfot jelöl, amelynek élhalmazán egy $w:E\rightarrow R$ valós értékű súlyfüggvény van megadva. A gráf minden $u,v \in V$ csúcspárjára keressük az u-ból v-be vezető legrövidebb (legkisebb súlyú) utat, ahol egy út súlya az úthoz tartozó élek súlyának az összege. Az eredményt általában táblázatos formában keressük; a táblázat u-hoz tartozó **sorában** és v-hez tartozó **oszlopában** álló elem az u-ból v-be vezető legrövidebb út hossza.
Az egy csúcsból kiinduló legrövidebb utakat megadó algoritmusok általában a gráf éllistás megadását igénylik. A bemenő adat tehát egy $\boldsymbol{W}_{n\,x\,n}$ -es mátrix lesz. A mátrix egy n csúcsú irányított $G=(V,E)$ gráf élsúlyait tartalmazza: $\boldsymbol{W}=(w_{ij})$, ahol:

$$w_{i,j}=\begin{cases} 0, & ha\ i=j \\ az\ irányított\ (i,j)él\ hossza, & ha\ i\neq j\ és\ (i,j)\in E \\ \infty, & ha\ i\neq j\ és\ (i,j)\ nem\ eleme\ E \end{cases}$$

Legrövidebb utak minden csúcspárra, értelmezések - 2

Az algoritmus **kimenete** az összes párra adott legrövidebb úthosszakat tartalmazó táblázat egy $\boldsymbol{D}_{n \times n}$ -es mátrix lesz. A mátrix d_{ij} eleme az i csúcsból a j csúcsba vezető legrövidebb út súlyát tartalmazza. A végeredményként kapott d_{ij} értékek tehát megegyeznek a δ_{ij}-vel, az i csúcsból a j csúcsba vezető legrövidebb út súlyával.

Csak akkor mondhatjuk, hogy a kitűzött feladatot megoldottuk, ha a legrövidebb utak súlya mellett magukat az utakat is meg tudjuk adni. E célból egy $\Pi = (\pi_{ij})$ megelőzési mátrixot is meg kell adnunk, amelyben π_{ij}=NIL, ha $i=j$ vagy ha nem vezet i és j között út; ellenkező esetben π_{ij} a j-t megelőző csúcs az egyik i-ből j-be vezető legrövidebb úton.

Mielőtt az algoritmust ismertetnénk, állapodjunk meg néhány, szomszédsági mátrixokkal kapcsolatos, jelölésben. Először is a $G=(V,E)$ gráfnak általában n csúcsa lesz, azaz n=|V| . Másodszor a mátrixokat nagybetűvel, egyes elemeiket pedig alulindexelt kisbetűvel fogjuk jelölni, tehát például $\boldsymbol{W}$ és $\boldsymbol{D}$ elemei w_{ij}, d_{ij} lesznek. Bizonyos esetekben iterációk jelölésére a mátrixokat zárójelezett felsőindexszel látjuk el, úgy mint $\boldsymbol{D}^{(m)}=(d_{ij}^{(m)})$. Végül egy adott $\boldsymbol{A}_{n \times n}$ -es mátrix esetén sorok-száma [A] tartalmazza n értékét.

A legrövidebb utak *minden csúcspárra* Floyd-Warshall algoritmus

A következőkben dinamikus programozási feladatként értelmezzük a **legrövidebb utak** keresését. Egy $G=(V,E)$ irányított gráfon a keletkező ún. Floyd-Warshall-algoritmus futási ideje $\theta(n^3)$.
A bemenő gráfban megengedünk negatív élsúlyokat, negatív összsúlyú köröket azonban nem.

Dinamikus programozásunk a legrövidebb utak „belső" csúcsait tekinti, ahol egy egyszerű $p=\langle v1,v2,\ldots,vl\rangle$ út belső csúcsa p minden $v1$-től és vl-től különböző csúcsa, azaz a $\{v2,v3,\ldots,v_{l-1}\}$ halmaz minden eleme.

A szerkezet jellemzése a következő észrevételen alapul. Legyen a G gráf csúcshalmaza $V=\{1,2,..,n\}$, és tekintsük valamely k-ra az $\{1,2,..,k\}$ részhalmazt. Legyen p a legrövidebb i-ből j-be vezető olyan út, melynek belső csúcsait az $\{1,2,..,k\}$ részhalmazból választhatjuk. (A p út egyszerű, hiszen G nem tartalmaz negatív összsúlyú köröket.)
A Floyd-Warshall-algoritmus a p út és az olyan legrövidebb utak kapcsolatát alkalmazza, melynek belső csúcsait az $\{1,2,..,k-1\}$ részhalmazból választjuk. E kapcsolat két esetre osztható attól függően, hogy k belső csúcsa-e p-nek vagy sem:

1. Ha k a p útnak nem belső csúcsa, akkor a p út minden belső csúcsa az $\{1,2,..,k-1\}$ halmaz eleme. Így a legrövidebb i-ből j-be vezető és belső csúcsként csak az $\{1,2,..,k-1\}$ halmaz elemeit használó út szintén legrövidebb út lesz, ha a belső csúcsok az $\{1,2,..,k\}$ halmazból kerülhetnek ki.

2. Ha k belső csúcs a p úton, akkor felbontjuk a p utat két útra (p_1,p_2), oly módon: $i \rightarrow k \rightarrow j$. A p_1 egy olyan legrövidebb út i és k között, melynek belső csúcsai az $\{1,2,..,k\}$ halmaz elemei. Sőt k nem is lehet p_1 út belső csúcsa, így p_1 egyben olyan legrövidebb út is, melynek belső csúcsai $\{1,2,..,k-1\}$-beliek. Hasonlóképpen p_2 egy legrövidebb, belső csúcsként csak $\{1,2,..,k-1\}$ -et használó k-ból j-be vezető út.

A csúcspárok közti legrövidebb utak rekurzív megadása

Legyen d_{ij}^k a legrövidebb olyan *i*-ből *j*-be vezető út hossza, melynek minden belső csúcsa az *{1,2,..,k}* halmaz eleme. A *k=0* érték esetén egy olyan *i*-ből *j*-be vezető útnak, melyen a belső csúcsok sorszáma legfeljebb *0* lehet, egyáltalán nem lehet belső csúcsa. Egy ilyen útnak tehát legfeljebb egyetlen éle lehet és így $d_{ij}^0 = w_{ij}$.

A további értékeket a következő rekurzió szolgáltatja: $$d_{ij}^k = \begin{cases} w_{ij}, & ha\ k = 0 \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right), & ha\ k \geq 1 \end{cases}$$

A végeredményt a $D^n = (d_{ij}^n)$ mátrix tartalmazza, hiszen az egyes legrövideb utak belső csúcsai a *{1,2,..,n}* halmaz elemei, és így $d_{ij}^n = \delta(i,j)$ minden $i,j \in V$ esetén.

<u>Az úthossz kiszámítása alulról felfelé haladva:</u>

A fenti rekurzió segítségével a d_{ij}^k értékeket alulról felfelé, *k* értéke szerinti növekvő sorrendben számíthatjuk. Az algoritmus bemenő adatai az (1) egyenlőség által definiált $\boldsymbol{W}_{n\,x\,n}$ -es mátrix lesz, eredményül pedig a legrövidebb úthosszak $\boldsymbol{D}^{(n)}$ mátrixát adja.

(1) $$w_{i,j} = \begin{cases} 0, & ha\ i = j \\ az\ irányított\ (i,j)él\ hossza, & ha\ i \neq j\ és\ (i,j) \in E \\ \infty, & ha\ i \neq j\ és\ (i,j)\ nem\ eleme\ E \end{cases}$$

A $D^{(n)}$ mátrix elemeinek kiszámolása

	Floyd-Warshall(W)	
1	$n \leftarrow$ sorok_száma$[W]$	$\Theta(n^3)$
2	$D^{(0)} \leftarrow W$	
3	**FOR** $k \leftarrow 1$ **TO** n **DO**	
5	**FOR** $i \leftarrow 1$ **TO** n **DO**	
6	**FOR** $j \leftarrow 1$ **TO** n **DO**	
7	$d_{ij}^{(k)} \leftarrow \min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\}$	
8	**RETURN** $D^{(n)}$	

A Floyd-Warshall-algoritmus futási idejét a 3-6. sorok háromszorosan egymásba ágyazott FOR ciklusa határozza meg. A 6. sor minden egyes alkalommal $O(1)$ időben végrehajtható, így a teljes futási idő $\theta(n^3)$. A megadott programkód tömör és csak egyszerű adatszerkezeteket igényel. A θ - jelölésű állandó tehát kicsi, és a Floyd-Warshall-algoritmus még közepesen nagy méretű gráfok esetén is hatékony.

A $\Pi^{(n)}$ mátrix elemeinek kiszámolása

	Minden-párhoz-utat-nyomtat$(\prod, i, j)$
1	**IF** $i = j$
2	**THEN** PRINT i
3	**ELSE IF** $\pi_{ij} = NIL$
4	**THEN** PRINT i „-ből” j„-be nem vezet út”
5	**ELSE** Minden-párhoz-utat-nyomtat$(\prod, i, \pi_{ij})$
6	PRINT j

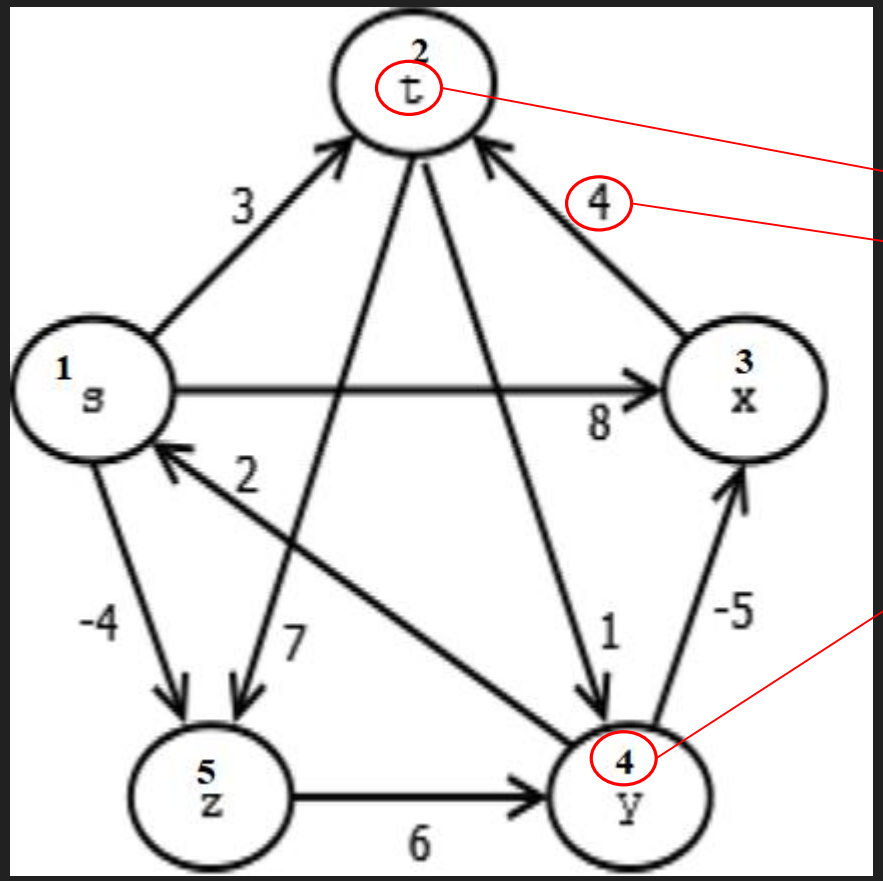
A Π megelőzési mátrixot számíthatjuk a Floyd-Warshall-algoritmus menetében a $D^{(k)}$ mátrixokkal egyidejűleg is. Pontosabban mátrixok egy $\{\Pi^{(1)}, \Pi^{(2)}, .., \Pi^{(n)}\}$ sorozatát számíthatunk, melyre $\Pi=\Pi^{(n)}$. E sorozatban π_{ij}^{k}-t úgy definiáljuk, mint egy olyan legrövidebb i-ből j-be vezető úton a j-t megelőző csúcsot, melynek belső csúcsai a $\{1,2,..,k\}$ halmaz elemei.
A π_{ij}^{k} értékeit definiáló rekurzió:

$$\pi_{ij}^{(0)} = \begin{cases} NIL, ha\ i = j\ vagy\ w_{ij} = \infty, \\ i, ha\ i \neq j\ és\ w_{ij} < \infty \end{cases}$$

$k \geq 1\ mellett\ a\ rekurzió\ egyenlete$:

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)}, ha\ d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)}, \\ \pi_{kj}^{(k-1)}, ha\ d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)}, \end{cases}$$

Floyd-Warshall, futtatása egy példán keresztül - magyarázat



$D^{(0)}=$

		1	2	3	4	5
		s	t	x	y	z
1	s	0	3	8	∞	-4
2	t	∞	0	∞	1	7
3	x	∞	4	0	∞	∞
4	y	2	∞	-5	0	∞
5	z	∞	∞	∞	6	0

u (a felső két sor)

v (az első két oszlop)

1-ből a 4-be nincs út

$\Pi^{(0)}=$

		1	2	3	4	5
		s	t	x	y	z
1	s	NIL	1	1	NIL	1
2	t	NIL	NIL	NIL	2	2
3	x	NIL	3	NIL	NIL	NIL
4	y	4	NIL	4	NIL	NIL
5	z	NIL	NIL	NIL	5	NIL

4-ből az 1-be van út, értéke a csomópont száma

Floyd-Warshall, futtatása egy példán keresztül – iterációk a rekurziós képletek alapján

$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(0)} = \begin{pmatrix} NIL & 1 & 1 & NIL & 1 \\ NIL & NIL & NIL & 2 & 2 \\ NIL & 3 & NIL & NIL & NIL \\ 4 & NIL & 4 & NIL & NIL \\ NIL & NIL & NIL & 5 & NIL \end{pmatrix}$$

$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(1)} = \begin{pmatrix} NIL & 1 & 1 & NIL & 1 \\ NIL & NIL & NIL & 2 & 2 \\ NIL & 3 & NIL & NIL & NIL \\ 4 & 1 & 4 & NIL & 1 \\ NIL & NIL & NIL & 5 & NIL \end{pmatrix}$$

$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(2)} = \begin{pmatrix} NIL & 1 & 1 & 2 & 1 \\ NIL & NIL & NIL & 2 & 2 \\ NIL & 3 & NIL & 2 & 2 \\ 4 & 1 & 4 & NIL & 1 \\ NIL & NIL & NIL & 5 & NIL \end{pmatrix}$$

$$D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(3)} = \begin{pmatrix} NIL & 1 & 1 & 2 & 1 \\ NIL & NIL & NIL & 2 & 2 \\ NIL & 3 & NIL & 2 & 2 \\ 4 & 3 & 4 & NIL & 1 \\ NIL & NIL & NIL & 5 & NIL \end{pmatrix}$$

$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{(4)} = \begin{pmatrix} NIL & 1 & 4 & 2 & 1 \\ 4 & NIL & 4 & 2 & 1 \\ 4 & 3 & NIL & 2 & 1 \\ 4 & 3 & 4 & NIL & 1 \\ 4 & 3 & 4 & 5 & NIL \end{pmatrix}$$

$$D^{(5)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{(5)} = \begin{pmatrix} NIL & 3 & 4 & 5 & 1 \\ 4 & NIL & 4 & 2 & 1 \\ 4 & 3 & NIL & 2 & 1 \\ 4 & 3 & 4 & NIL & 1 \\ 4 & 3 & 4 & 5 & NIL \end{pmatrix}$$