GRÁFELMÉLET

Egy csúcsból induló legrövidebb utak

5. előadás

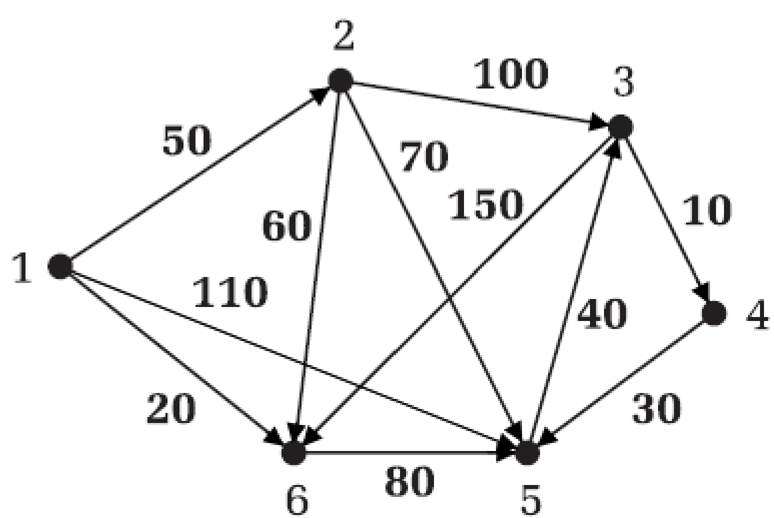
Probléma: A városi tömegközlekedés optimalizálása az utasforgalom statisztikai felmérése után. Az úthálózatban keressük a központi buszállomástól az összes többi buszmegállóhoz vezető legrövidebb utakat.

Probléma: A városi tömegközlekedés optimalizálása az utasforgalom statisztikai felmérése után. Az úthálózatban keressük a központi buszállomástól az összes többi buszmegállóhoz vezető legrövidebb utakat.

Megoldás: A problémát egy összefüggő irányított $\mathrm{G} = (\mathrm{V}, \mathrm{E})$ gráf segítségével modellezhetjük, melynek minden éléhez **súlyok**at ($súly$: $\mathrm{E} \rightarrow \mathrm{R}$) rendelünk. Egy **út súlyá**t úgy definiáljuk, mint az út menti élek súlyainak összegét. Határozzuk meg a G gráf adott csúcspontjából az összes többi csúcspontba vezető legrövidebb utakat!

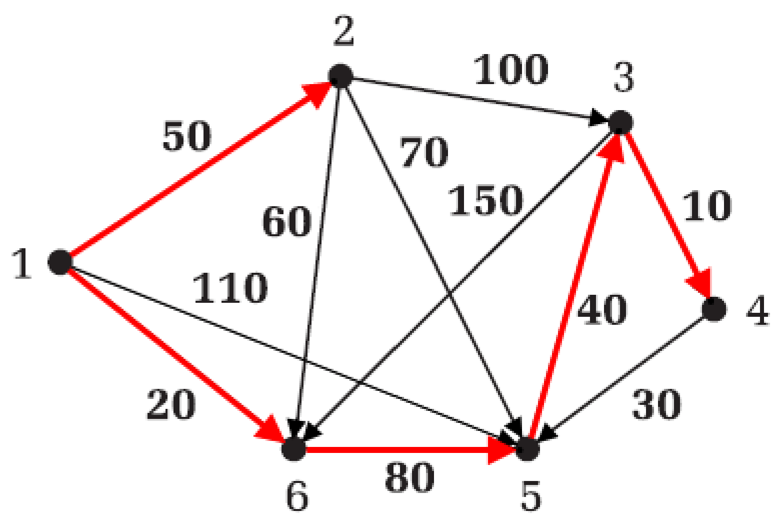
Probléma: A városi tömegközlekedés optimalizálása az utasforgalom statisztikai felmérése után. Az úthálózatban keressük a központi buszállomástól az összes többi buszmegállóhoz vezető legrövidebb utakat.

Megoldás: A problémát egy összefüggő irányított $G = (V, E)$ gráf segítségével modellezhetjük, melynek minden éléhez **súlyok**at ($súly: E \rightarrow R$) rendelünk. Egy **út súlyá**t úgy definiáljuk, mint az út menti élek súlyainak összegét. Határozzuk meg a G gráf adott csúcspontjából az összes többi csúcspontba vezető legrövidebb utakat!



Probléma: A városi tömegközlekedés optimalizálása az utasforgalom statisztikai felmérése után. Az úthálózatban keressük a központi buszállomástól az összes többi buszmegállóhoz vezető legrövidebb utakat.

Megoldás: A problémát egy összefüggő irányított $G = (V, E)$ gráf segítségével modellezhetjük, melynek minden éléhez **súlyok**at ($súly$: $E \rightarrow R$) rendelünk. Egy **út súlyá**t úgy definiáljuk, mint az út menti élek súlyainak összegét. Határozzuk meg a G gráf adott csúcspontjából az összes többi csúcspontba vezető legrövidebb utakat!



A továbbiakban az s és v csúcspontok közötti legrövidebb út súlyát `lru(s,v)` fogja jelölni. Ha s és v között nincs út, akkor definíció szerint `lru(s,v)`$=\infty$. Ha s-től v-hez vezető úton negatív összsúlyú kör található, akkor definíció szerint `lru(s,v)`$=-\infty$.

A továbbiakban az s és v csúcspontok közötti legrövidebb út súlyát `lru(s,v)` fogja jelölni. Ha s és v között nincs út, akkor definíció szerint `lru(s,v)=∞`. Ha s-től v-hez vezető úton negatív összsúlyú kör található, akkor definíció szerint `lru(s,v)=-∞`.

A bemutatásra kerülő algoritmusok a *fokozatos közelítés technikáját* alkalmazzák. Ennél a technikánál az optimalitás alapelve a következő: **egy legrövidebb út részútjai is is legrövidebb utak**.

A továbbiakban az s és v csúcspontok közötti legrövidebb út súlyát `lru(s,v)` fogja jelölni. Ha s és v között nincs út, akkor definíció szerint `lru(s,v)=∞`. Ha s-től v-hez vezető úton negatív összsúlyú kör található, akkor definíció szerint `lru(s,v)=-∞`.

A bemutatásra kerülő algoritmusok a *fokozatos közelítés technikáját* alkalmazzák. Ennél a technikánál az optimalitás alapelve a következő: **egy legrövidebb út részútjai is is legrövidebb utak**.

1. következmény: *a kevesebb élből álló legkisebb súlyú utakból felépíthetők a több élt tartalmazó legkisebb súlyú utak*.

A továbbiakban az s és v csúcspontok közötti legrövidebb út súlyát `lru(s,v)` fogja jelölni. Ha s és v között nincs út, akkor definíció szerint `lru(s,v)=∞`. Ha s-től v-hez vezető úton negatív összsúlyú kör található, akkor definíció szerint `lru(s,v)=-∞`.

A bemutatásra kerülő algoritmusok a *fokozatos közelítés technikáját* alkalmazzák. Ennél a technikánál az optimalitás alapelve a következő: **egy legrövidebb út részútjai is is legrövidebb utak**.

1. következmény: *a kevesebb élből álló legkisebb súlyú utakból felépíthetők a több élt tartalmazó legkisebb súlyú utak*.

2. következmény: *az s pontból induló legrövidebb utak s gyökerű fát alkotnak*. Ez azt jelenti, hogy a már meghatározott legrövidebb utak folyamatosan fát alkotnak. A fa fokozatosan épül fel élről élre, minden él egy újabb csúcsponttal bővíti a fát.

3. következmény: *ha* s*-ből* v*-be tartó legrövidebb úton* u *az utolsó előtti állomás, akkor*

$$\mathtt{lru}(s, v) = \mathtt{lru}(s, u) + súly\,(u, v)$$

3. következmény: *ha* s*-ből* v*-be tartó legrövidebb úton* u *az utolsó előtti állomás, akkor*

$$\mathtt{lru}(s, v) = \mathtt{lru}(s, u) + súly\,(u, v)$$

4. következmény: *ha* s *a kezdőcsúcs, akkor tetszőleges* (u, v) *élre teljesül, hogy*:

$$\mathtt{lru}(s, v) \leq \mathtt{lru}(s, u) + súly\,(u, v)$$

1) Dijkstra algoritmusa

Feltétel: A gráf nem tartalmazhat negatív súlyú élt.

1) Dijkstra algoritmusa

Feltétel: A gráf nem tartalmazhat negatív súlyú élt.

Stratégia: Az optimalitás alapelve kimondja, hogy a legrövidebb utak részútjai is legrövidebb utak. Ha nincs negatív súlyú éle a gráfnak, akkor a hosszabb (nagyobb súlyú) legrövidebb utak felépíthetők a rövidebb (kisebb súlyú) legrövidebb utakból.

1) Dijkstra algoritmusa

Feltétel: A gráf nem tartalmazhat negatív súlyú élt.

Stratégia: Az optimalitás alapelve kimondja, hogy a legrövidebb utak részútjai is legrövidebb utak. Ha nincs negatív súlyú éle a gráfnak, akkor a hosszabb (nagyobb súlyú) legrövidebb utak felépíthetők a rövidebb (kisebb súlyú) legrövidebb utakból.

Dijkstra algoritmusa hosszúságuk (súlyuk) szerint növekvő sorrendben határozza meg az egyes csúcspontokhoz vezető legrövidebb utakat. Az algoritmus nagyon hasonlít Prim algoritmusához. A Q elsőbbségi sor azokat a csúcspontokat tartalmazza, amelyekhez még nem határoztuk meg a legrövidebb utat (kezdetben tehát az összest tartalmazza). A $(V \setminus Q, Q)$ vágás bal oldalán a növekvő legrövidebb utak fája (azokkal a csúcspontokkal, amelyekhez már megvan a legrövidebb út), a jobb oldalán pedig a Q halmazba tartozó csúcspontok találhatók.

1) Dijkstra algoritmusa

Az algoritmus a legrövidebb utak fáját úgy építi fel, mint egy virtuális gyökerű fát. Az s kiindulási csúcspont 0 távolságra, az összes többi csúcspont végtelen távolságra van a virtuális gyökértől. Kezdetben egyetlen csúcspont sincs a fához illesztve, azaz minden csúcspont eleme a Q halmaznak. Első lépésben az s csúcspont automatikusan a fa részévé válik (rátevődik a virtuális gyökérre), és törlődik a Q halmazból.

1) Dijkstra algoritmusa

Az algoritmus a legrövidebb utak fáját úgy építi fel, mint egy virtuális gyökerű fát. Az s kiindulási csúcspont 0 távolságra, az összes többi csúcspont végtelen távolságra van a virtuális gyökértől. Kezdetben egyetlen csúcspont sincs a fához illesztve, azaz minden csúcspont eleme a Q halmaznak. Első lépésben az s csúcspont automatikusan a fa részévé válik (rátevődik a virtuális gyökérre), és törlődik a Q halmazból.

Prim algoritmusától eltérően Dijkstra algoritmusa mindig a fa gyökeréhez (s) legközelebbi csúcspontot csatolja a fához, s nem a fához legközelebbi csúcspontot.

1) Dijkstra algoritmusa

Az algoritmus a legrövidebb utak fáját úgy építi fel, mint egy virtuális gyökerű fát. Az s kiindulási csúcspont 0 távolságra, az összes többi csúcspont végtelen távolságra van a virtuális gyökértől. Kezdetben egyetlen csúcspont sincs a fához illesztve, azaz minden csúcspont eleme a Q halmaznak. Első lépésben az s csúcspont automatikusan a fa részévé válik (rátevődik a virtuális gyökérre), és törlődik a Q halmazból.

Prim algoritmusától eltérően Dijkstra algoritmusa mindig a fa gyökeréhez (s) legközelebbi csúcspontot csatolja a fához, s nem a fához legközelebbi csúcspontot.

A `d[1..n]` tömb elemei a csúcspontoknak a fa gyökerétől kizárólag fa-pontokon keresztül mért legrövidebb távolságát tárolják. Az algoritmus minden lépésben azt a csúcspontot csatolja a fához, amelynek a `d` tömbbeli értéke a legkisebb.

1) Dijkstra algoritmusa

A következő pont tehát, amelyikhez a legrövidebb út meghatározásra kerül, a legrövidebb utak fájának azon ki-szomszédja lesz, amely a fa gyökeréhez legközelebb esik.

1) Dijkstra algoritmusa

A következő pont tehát, amelyikhez a legrövidebb út meghatározásra kerül, a legrövidebb utak fájának azon ki-szomszédja lesz, amely a fa gyökeréhez legközelebb esik.

Tegyük fel, hogy a v csúcspont lesz az, amelyikhez a következő legrövidebb út vezet. Ha az u csúcspont az utolsó előtti állomás ezen az úton, akkor u-hoz már meghatároztuk a legrövidebb utat, vagyis u már része a legrövidebb utak fájának. Tehát a következő legrövidebb út egyik sajátossága, hogy az $(s \to u)$ útszakasz már része a fának, az utolsó (u, v) éle pedig a fa valamelyik ki-éle lesz.

1) Dijkstra algoritmusa

A következő pont tehát, amelyikhez a legrövidebb út meghatározásra kerül, a legrövidebb utak fájának azon ki-szomszédja lesz, amely a fa gyökeréhez legközelebb esik.

Tegyük fel, hogy a v csúcspont lesz az, amelyikhez a következő legrövidebb út vezet. Ha az u csúcspont az utolsó előtti állomás ezen az úton, akkor u-hoz már meghatároztuk a legrövidebb utat, vagyis u már része a legrövidebb utak fájának. Tehát a következő legrövidebb út egyik sajátossága, hogy az $(s \to u)$ útszakasz már része a fának, az utolsó (u, v) éle pedig a fa valamelyik ki-éle lesz.

A v pont fához való csatolása mintegy nyugtázza, hogy az aktuális `d[v]` érték finomítása befejeződött, és hogy az u = `apa[v]` csúcspont az s-ből v-be vezető véglegesített legrövidebb út utolsó előtti állomása. Minden csatolás után frissíteni kell a vágást és ezzel együtt a csatolásra jelölt pontok halmazát, illetve a `d` és `apa` tömböket is.

1) Dijkstra algoritmusa

A vágásból kikerülnek a csatolt v csúcspont fa-pontokból induló be-élei, és bekerülnek a vágás jobb oldalán maradt ki-szomszédaihoz vezető ki-élei. A v csúcsponton keresztül annak z ki-szomszédai `d[v]+súly(v,z)` távolságra kerültek s-től. Ha ez a távolság kisebb, mint az aktuális `d[z]` érték, akkor szükséges a `d[z]` felülírása a `d[v]+súly(v,z)` értékkel, illetve az `apa[z]` érték v csúcspontra való frissítése.

1) Dijkstra algoritmusa

A vágásból kikerülnek a csatolt v csúcspont fa-pontokból induló be-élei, és bekerülnek a vágás jobb oldalán maradt ki-szomszédaihoz vezető ki-élei. A v csúcsponton keresztül annak z ki-szomszédai `d[v]+súly(v,z)` távolságra kerültek s-től. Ha ez a távolság kisebb, mint az aktuális `d[z]` érték, akkor szükséges a `d[z]` felülírása a `d[v]+súly(v,z)` értékkel, illetve az `apa[z]` érték v csúcspontra való frissítése.

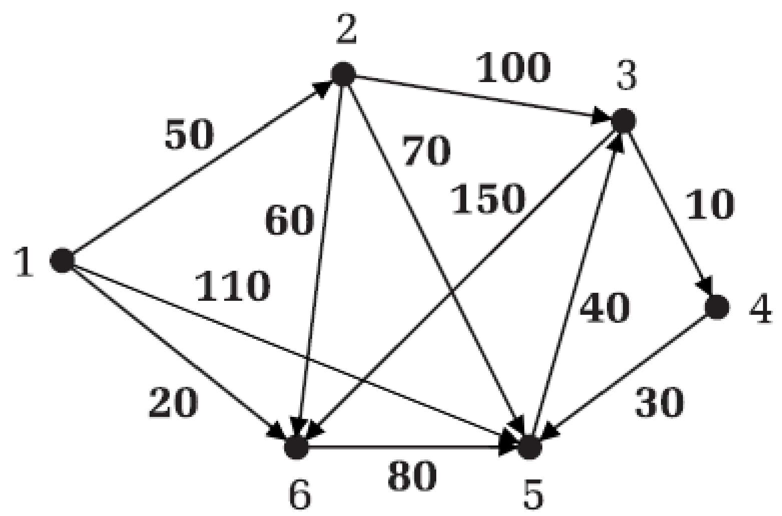
Az algoritmus végén a gráf minden u csúcspontjára a `d[u]` tömbelem az `lrs[s,u]` értéket tárolja, az `apa[u]` tömbelem pedig az u csúcspont apa-csúcspontját a legrövidebb utak fájában.

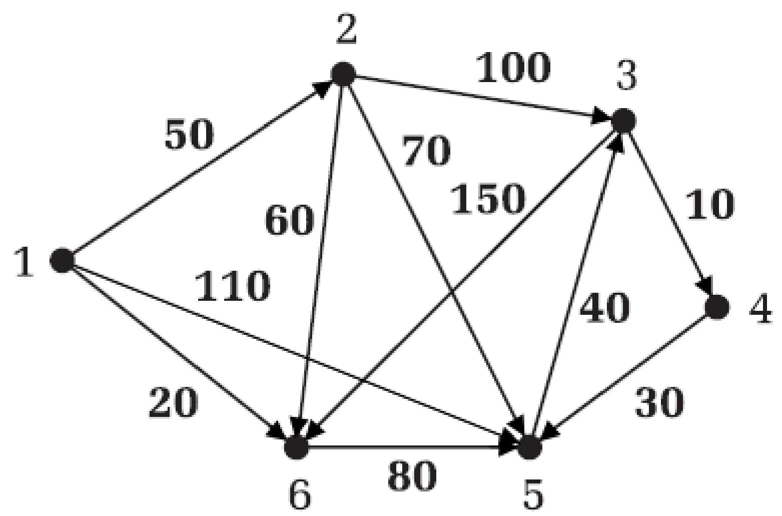
```
függvény DIJKSTRA(G,s)
    Q ← V(G)
    minden v∈Q végezd
        d[v] ← ∞
    vége minden
    d[s] ← 0
    apa[s] ← 0
    amíg Q ≠ ∅ végezd
        u ← KIVESZ_MIN(Q)
        ha u = 0 akkor
            ugorj
        vége ha
        minden v ∈ szomszéd(u) végezd
            ha (v∈Q ÉS (d[u] + súly(u,v) < d[v])) akkor
                apa[v] ← u
                d[v] ← d[u] + súly(u,v)
            vége ha
        vége minden
    vége amíg
    vissza apa
vége DIJKSTRA
```

```
függvény DIJKSTRA(G,s)
   Q ← V(G)
   minden v∈Q végezd
      d[v] ← ∞
   vége minden
   d[s] ← 0
   apa[s] ← 0
   amíg Q ≠ ∅ végezd
      u ← KIVESZ_MIN(Q)
      ha u = 0 akkor
         ugorj
      vége ha
      minden v ∈ szomszéd(u) végezd
         ha (v∈Q ÉS (d[u] + súly(u,v) < d[v])) akkor
            apa[v] ← u
            d[v] ← d[u] + súly(u,v)
         vége ha
      vége minden
   vége amíg
   vissza apa
vége DIJKSTRA
```

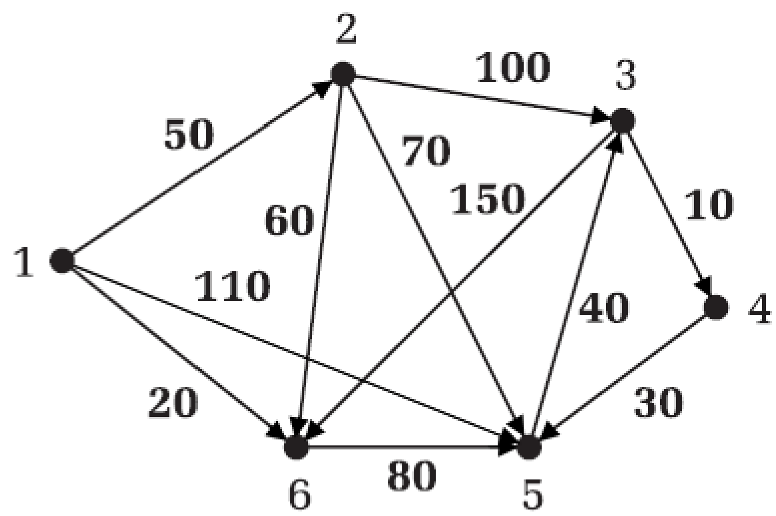
A DIJKSTRA algoritmus bonyolultsága $O(m \log n)$.

A KIVESZ_MIN(Q) függvény azt a csúcspontot adja vissza, amelyiknek a d tömbbeli értéke a legkisebb, s egyben törli azt a pontot a Q sorból. Ha már nincs csúcspont a Q sorban (a fának nincs ki-szomszédja; nincs él a vágásban; minden Q-beli pontnak a d tömbbeli értéke végtelen), akkor a függvény nullát ad vissza. Ez azt jelenti, hogy a Q-ban maradt csúcspontok nem elérhetők s-ből.

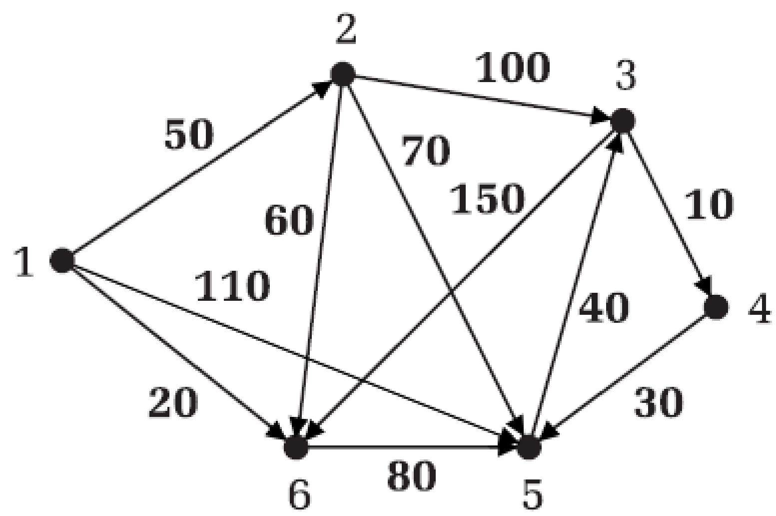

2
100
3
50
70
150
10
60
1
110
40
4
20
30
6
80
5



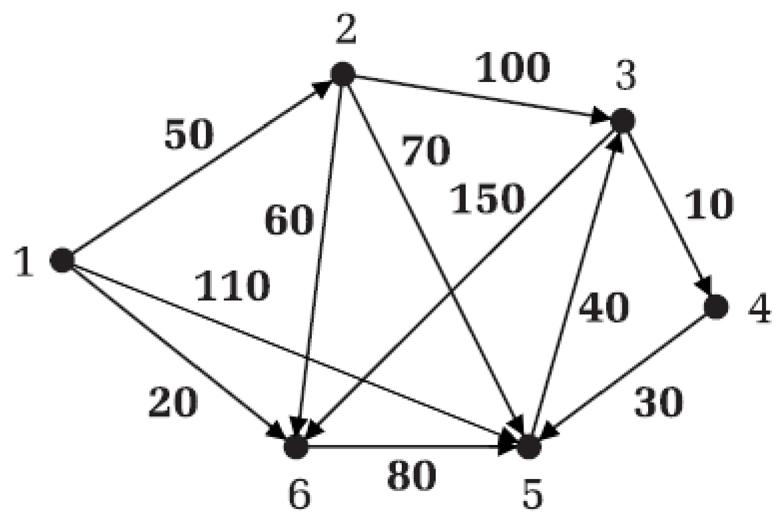
d	0	∞	∞	∞	∞	∞
apa	0	0	0	0	0	0
	1	2	3	4	5	6



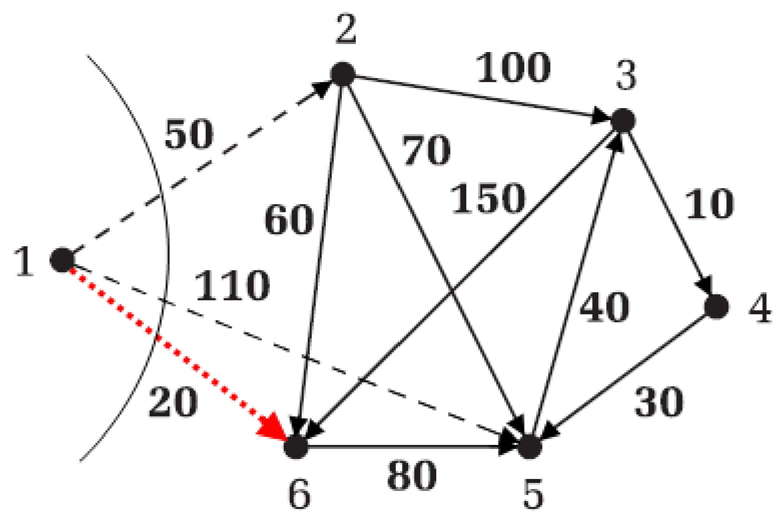
d	0	∞	∞	∞	∞	∞
apa	0	0	0	0	0	0
	1	2	3	4	5	6



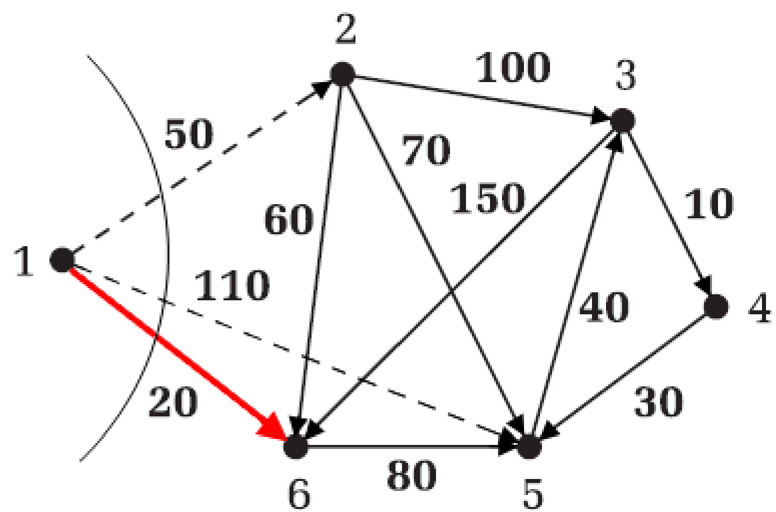
	1	2	3	4	5	6
d	**0**	∞	∞	∞	∞	∞
apa	**0**	0	0	0	0	0



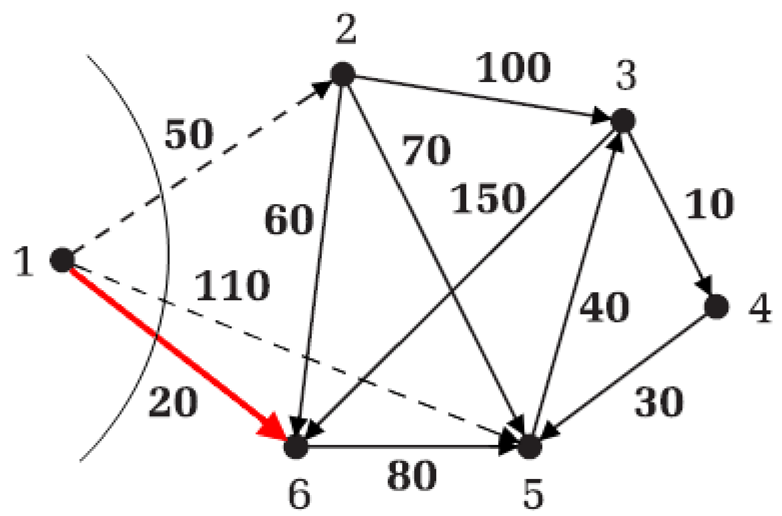
d	**0**	**50**	∞	∞	**110**	**20**
apa	**0**	**1**	0	0	**1**	**1**
	1	2	3	4	5	6



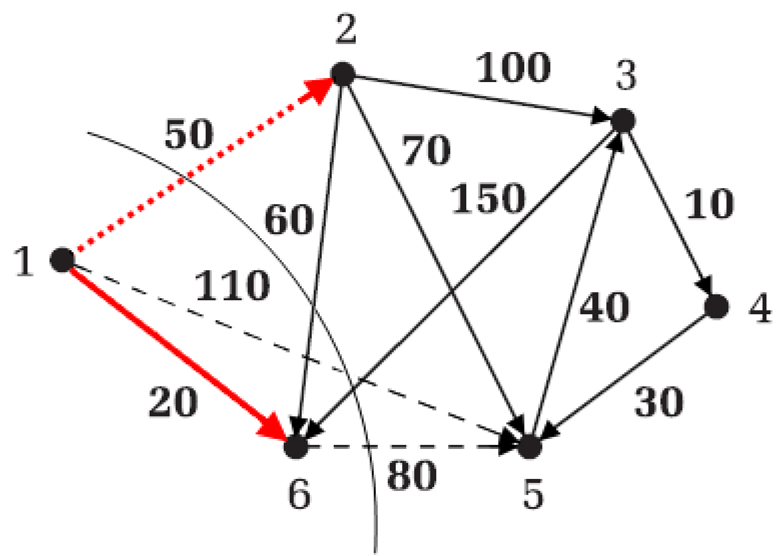
d	0	50	∞	∞	110	20
apa	0	1	0	0	1	1
	1	2	3	4	5	6



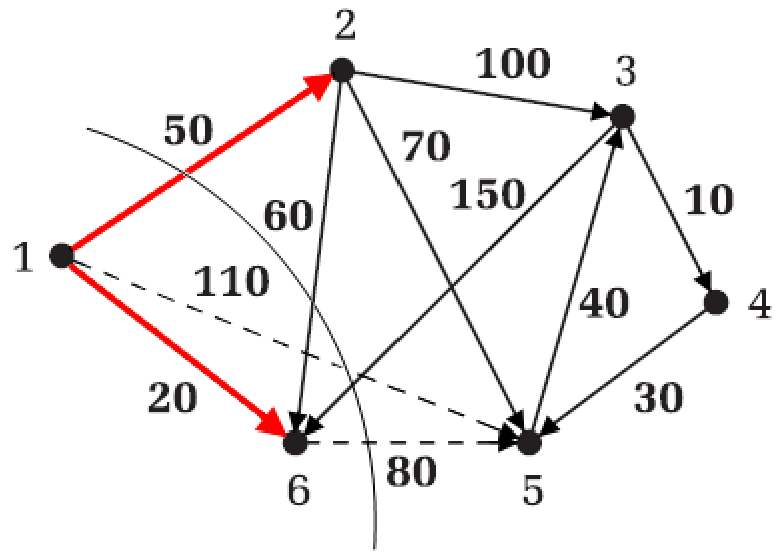
d	0	50	∞	∞	110	20
apa	0	1	0	0	1	1
	1	2	3	4	5	6



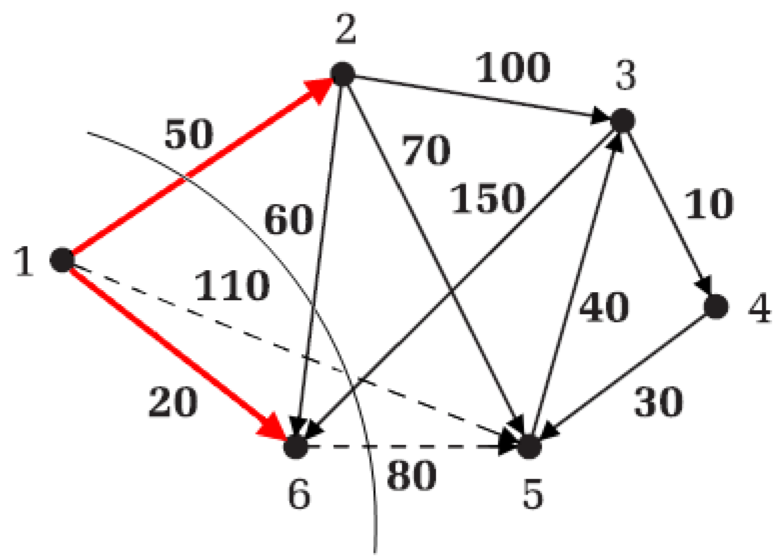
d	0	50	∞	∞	100	20
apa	0	1	0	0	6	1
	1	2	3	4	5	6



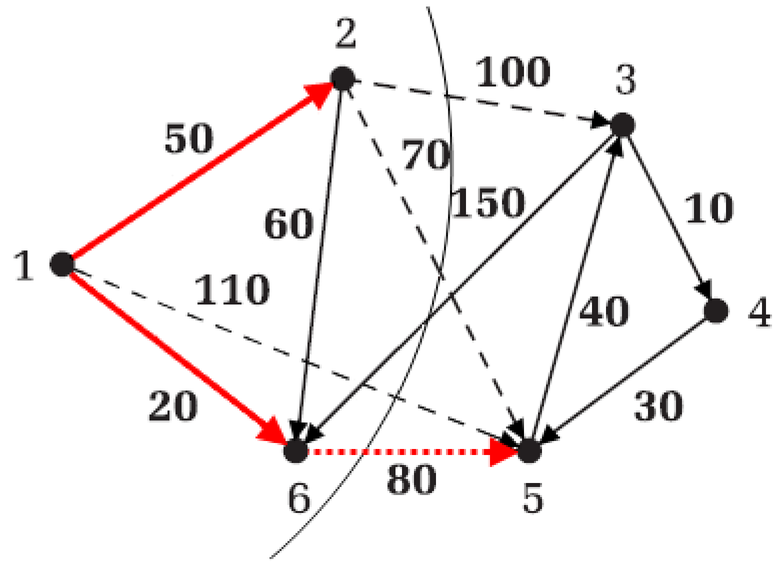
	1	2	3	4	5	6
d	0	50	∞	∞	100	20
apa	0	1	0	0	6	1



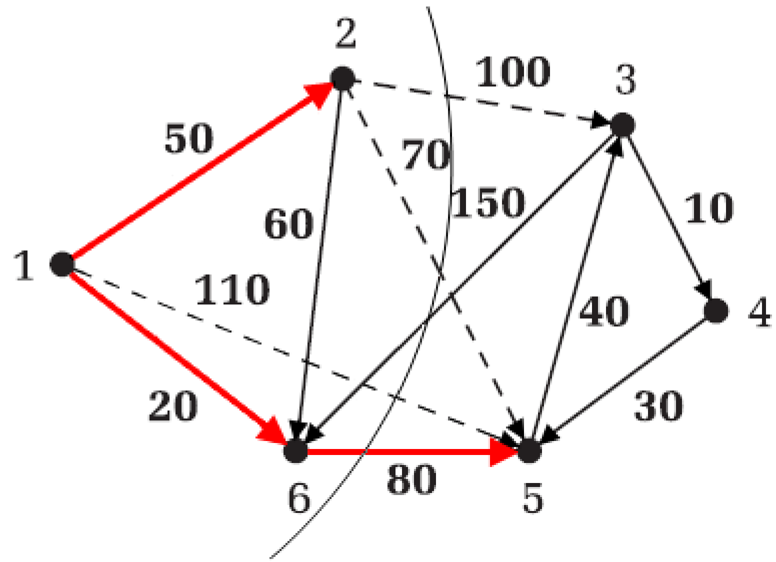
d	0	50	∞	∞	100	20
apa	0	1	0	0	6	1
	1	2	3	4	5	6



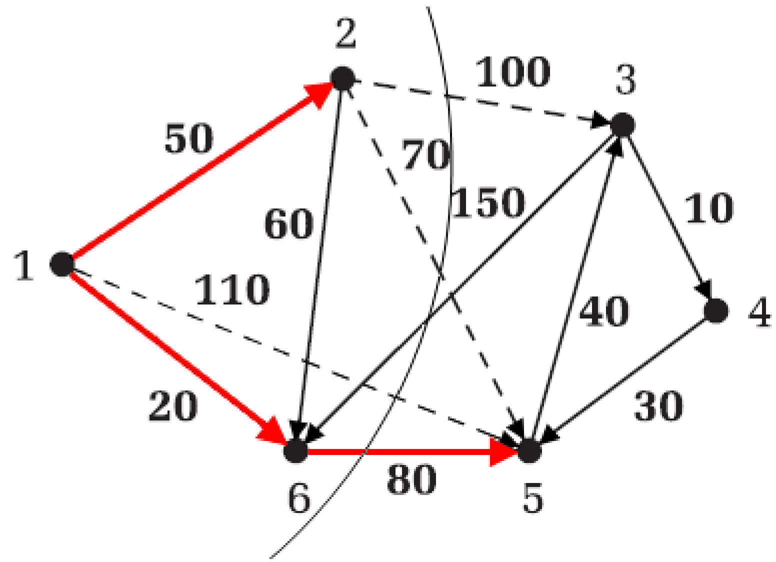
d	0	50	150	∞	100	20
apa	0	1	2	0	6	1
	1	2	3	4	5	6



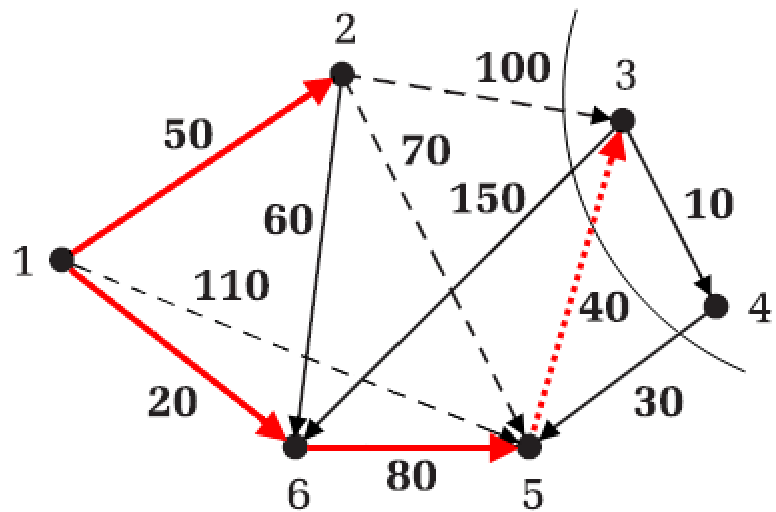
	1	2	3	4	5	6
d	0	50	150	∞	100	20
apa	0	1	2	0	6	1



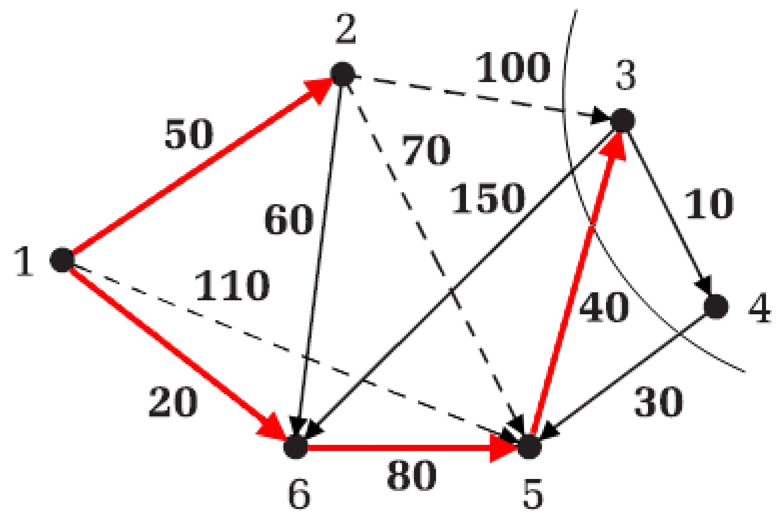
d	0	50	150	∞	100	20
apa	0	1	2	0	6	1
	1	2	3	4	5	6



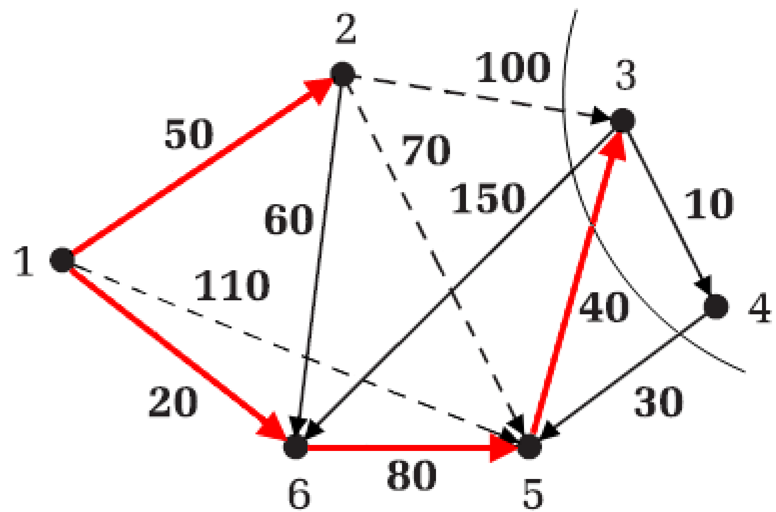
	1	2	3	4	5	6
d	0	50	140	∞	100	20
apa	0	1	5	0	6	1



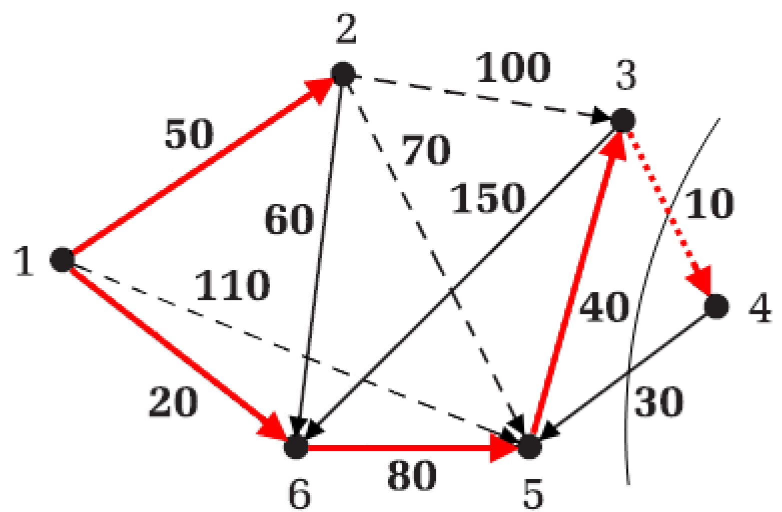
d	0	50	140	∞	100	20
apa	0	1	5	0	6	1
	1	2	3	4	5	6



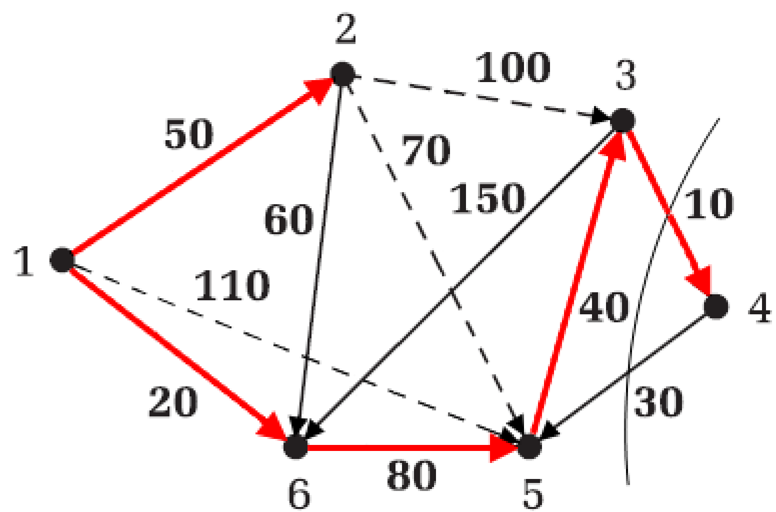
d	0	50	140	∞	100	20
apa	0	1	5	0	6	1
	1	2	3	4	5	6



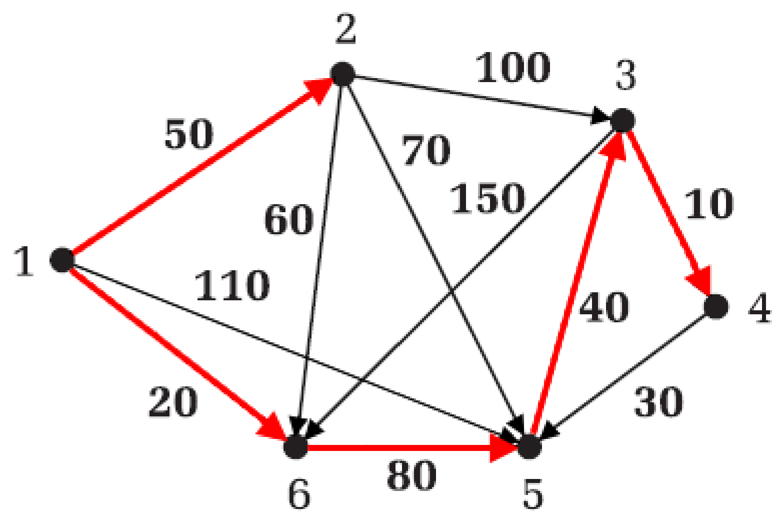
	1	2	3	4	5	6
d	0	50	140	150	100	20
apa	0	1	5	3	6	1



	1	2	3	4	5	6
d	0	50	140	150	100	20
apa	0	1	5	3	6	1



d	0	50	140	150	100	20
apa	0	1	5	3	6	1
	1	2	3	4	5	6



d	0	50	140	150	100	20
apa	0	1	5	3	6	1
	1	2	3	4	5	6

2) Bellman-Ford algoritmus

Feltétel: A gráf nem tartalmazhat a kezdő csúcspontból elérhető negatív összsúlyú kört. Ha van ilyen kör, akkor az algoritmus jelzi, hogy nincs megoldás.

2) Bellman-Ford algoritmus

Feltétel: A gráf nem tartalmazhat a kezdő csúcspontból elérhető negatív összsúlyú kört. Ha van ilyen kör, akkor az algoritmus jelzi, hogy nincs megoldás.

Dijkstra algoritmusa úgy határozza meg a legrövidebb utakat, hogy minden éllel legfeljebb egyszer közelít. Ez azért lehetséges, mert biztosítani tud olyan (az élek kezdőpontjainak `lru` értékein alapuló) élsorrendet, amely alapján történik a megközelítés.

2) Bellman-Ford algoritmus

Feltétel: A gráf nem tartalmazhat a kezdő csúcspontból elérhető negatív összsúlyú kört. Ha van ilyen kör, akkor az algoritmus jelzi, hogy nincs megoldás.

Dijkstra algoritmusa úgy határozza meg a legrövidebb utakat, hogy minden éllel legfeljebb egyszer közelít. Ez azért lehetséges, mert biztosítani tud olyan (az élek kezdőpontjainak `lru` értékein alapuló) élsorrendet, amely alapján történik a megközelítés.

A Bellman-Ford algoritmus esetén a negatív súlyú élek miatt nem ismert olyan él-sorrend, amely garantálná a legrövidebb utak egyetlen menetből történő előállítását, ezért szükségessé válhat az élek többszöri átpásztázása. Az újbóli közelítéseket addig ismételjük, amíg sikerül anélkül végigjárni az éleket, hogy sor került volna közelítésre. Ezt fogja figyelni a volt_közelítés változó.

2) Bellman-Ford algoritmus

Stratégia: Az algoritmus első lépésben csak azokat a legrövidebb utakat határozza meg, amelyeknél a tetszőlegesen választott él-sorrend biztosítja, hogy az út menti élekkel az illető utakon való előfordulásuk sorrendjében történjen a közelítés.

2) Bellman-Ford algoritmus

Stratégia: Az algoritmus első lépésben csak azokat a legrövidebb utakat határozza meg, amelyeknél a tetszőlegesen választott él-sorrend biztosítja, hogy az út menti élekkel az illető utakon való előfordulásuk sorrendjében történjen a közelítés.

Kezdetben csak az s kiindulóponthoz van meg a legrövidebb út hossza. Az első menetben biztosan meghatározásra kerülnek az egy él hosszúságú legrövidebb utak, a másodikban a két élből álló utak, stb. Ha egy adott menetben a $\mathrm{volt\_közelítés}$ változó hamis marad, akkor több menet nem lesz. Mivel egy legrövidebb út legfeljebb $n-1$ élből áll, ezért legfeljebb $n-1$ menetre lesz szükség.

2) Bellman-Ford algoritmus

Stratégia: Az algoritmus első lépésben csak azokat a legrövidebb utakat határozza meg, amelyeknél a tetszőlegesen választott él-sorrend biztosítja, hogy az út menti élekkel az illető utakon való előfordulásuk sorrendjében történjen a közelítés.

Kezdetben csak az s kiindulóponthoz van meg a legrövidebb út hossza. Az első menetben biztosan meghatározásra kerülnek az egy él hosszúságú legrövidebb utak, a másodikban a két élből álló utak, stb. Ha egy adott menetben a $\mathrm{volt\_közelítés}$ változó hamis marad, akkor több menet nem lesz. Mivel egy legrövidebb út legfeljebb $n-1$ élből áll, ezért legfeljebb $n-1$ menetre lesz szükség.

Ha az algoritmus az n-edik menetben is végez közelítést, akkor az negatív összsúlyú kör jelenlétéről tanúskodik.

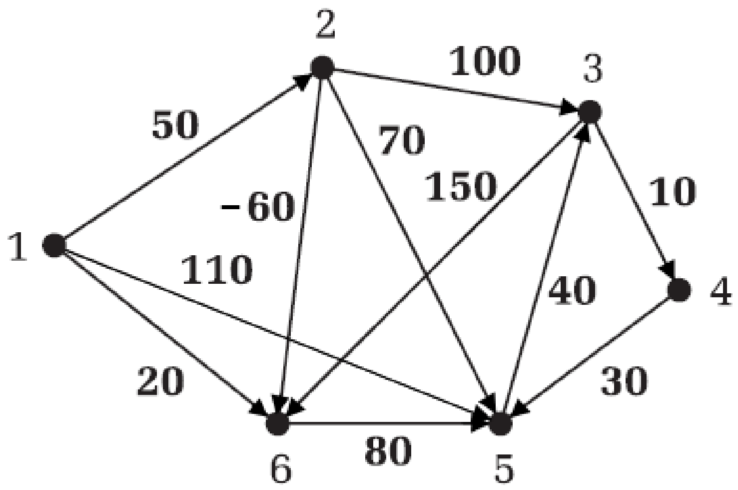
2) Bellman-Ford algoritmus

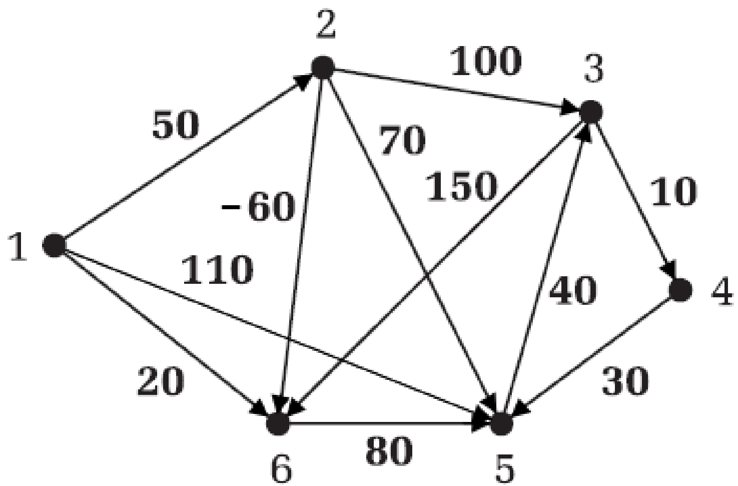
Ha a BELLMAN_FORD függvény igaz értékkel tér vissza, akkor az azt jelenti, hogy az adott gráfban nem tartalmaz s-ből elérhető negatív összsúlyú kört. A futás végén a `d` tömbben az egyes csúcspontokba vezető legrövidebb utak hosszai, az `apa` tömbben pedig a legrövidebb utak utolsó előtti állomásai (csúcspontjai) fognak szerepelni.

```
függvény BELLMAN_FORD(G(V,E),s,d[1..n],apa[1..n],n)
   minden v∈V(G) végezd
      d[v] ← ∞
   vége minden
   d[s] ← 0
   apa[s] ← 0
   i ← 0
   végezd
      volt_közelítés ← HAMIS
      minden (u,v)∈E(G) végezd
         ha d[u] + súly(u,v) < d[v] akkor
            apa[v] ← u
            d[v] ← d[u] + súly(u,v)
            volt_közelítés ← IGAZ
         vége ha
      vége minden
      i ← i + 1
   amíg (volt_közelítés = IGAZ) ÉS (i < n)
   ha (volt_közelítés = IGAZ) ÉS (i = n) akkor
      vissza HAMIS
   különben
      vissza IGAZ
   vége ha
vége BELLMAN_FORD
```

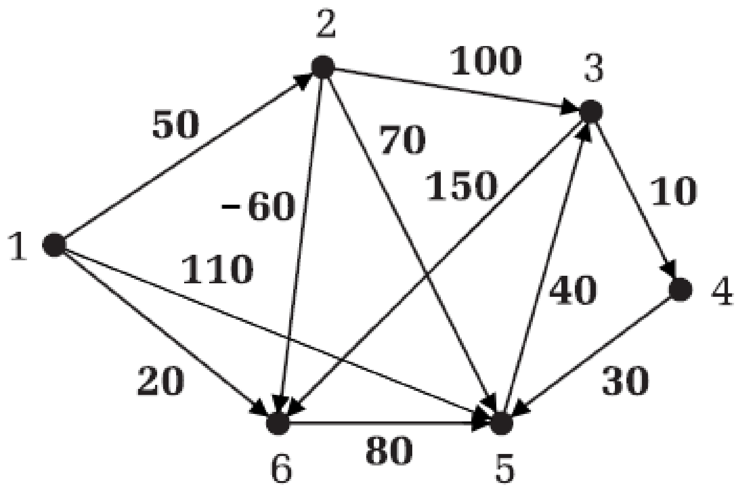
```
függvény BELLMAN_FORD(G(V,E),s,d[1..n],apa[1..n],n)
   minden v∈V(G) végezd
      d[v] ← ∞
   vége minden
   d[s] ← 0
   apa[s] ← 0
   i ← 0
   végezd
      volt_közelítés ← HAMIS
      minden (u,v)∈E(G) végezd
         ha d[u] + súly(u,v) < d[v] akkor
            apa[v] ← u
            d[v] ← d[u] + súly(u,v)
            volt_közelítés ← IGAZ
         vége ha
      vége minden
      i ← i + 1
   amíg (volt_közelítés = IGAZ) ÉS (i < n)
   ha (volt_közelítés = IGAZ) ÉS (i = n) akkor
      vissza HAMIS
   különben
      vissza IGAZ
   vége ha
vége BELLMAN_FORD
```

A BELLMAN_FORD algoritmus bonyolultsága $O(nm)$.


2
100
3
50
70
150
-60
10
1
110
40
4
20
30
80
6
5

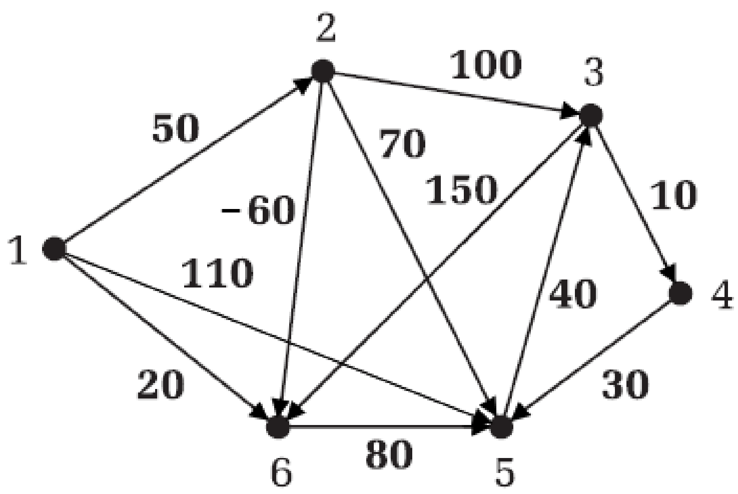


	1	2	3	4	5	6
0. menet	*0*	∞	∞	∞	∞	∞



	1	2	3	4	5	6
0. menet	*0*	∞	∞	∞	∞	∞

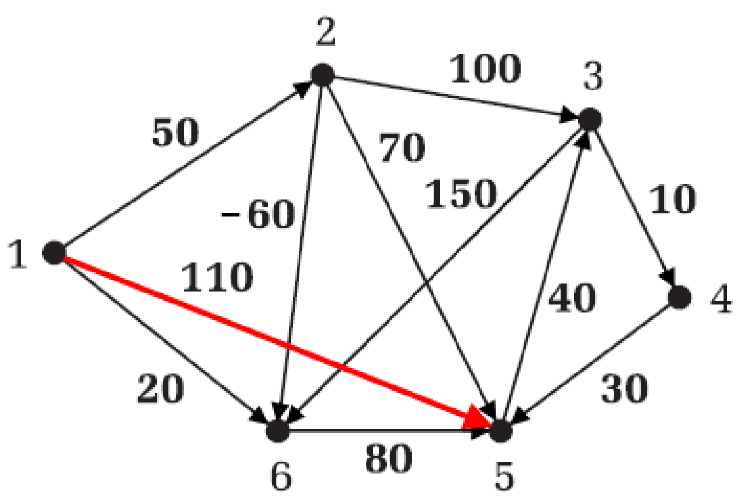
1. menet



		1	2	3	4	5	6
0. menet		*0*	∞	∞	∞	∞	∞
	1:(2,6) [−60]	0	∞	∞	∞	∞	∞

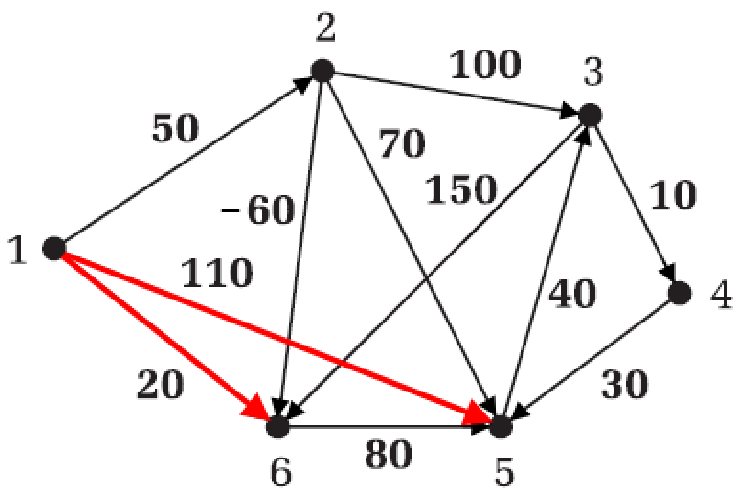
1. menet

Selye János Egyetem
Gazdaságtudományi Kar



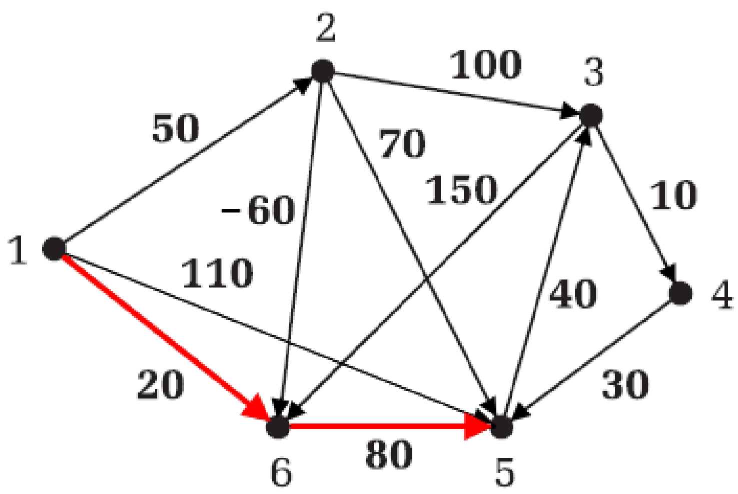
		1	2	3	4	5	6
0. menet		*0*	∞	∞	∞	∞	∞
	1:(2,6) [–60]	0	∞	∞	∞	∞	∞
	2:(1,5) [110]	0	∞	∞	∞	**110**	∞

1. menet



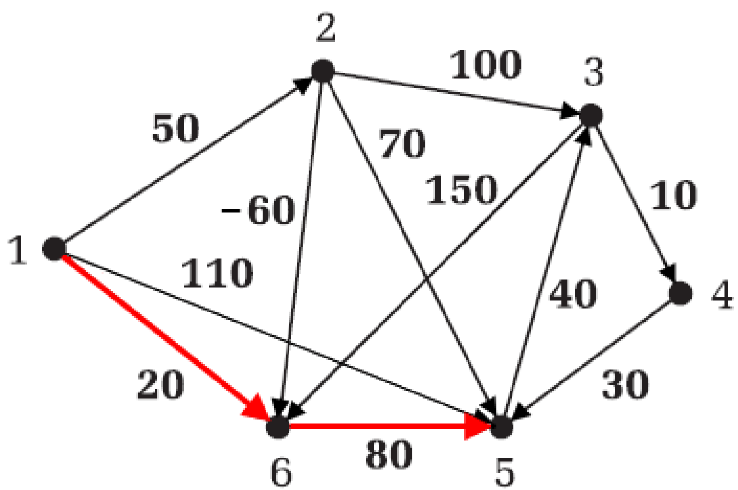
		1	2	3	4	5	6
0. menet		*0*	∞	∞	∞	∞	∞
	1:(2,6) [–60]	0	∞	∞	∞	∞	∞
	2:(1,5) [110]	0	∞	∞	∞	**110**	∞
	3:(1,6) [20]	0	∞	∞	∞	110	**20**

1. menet

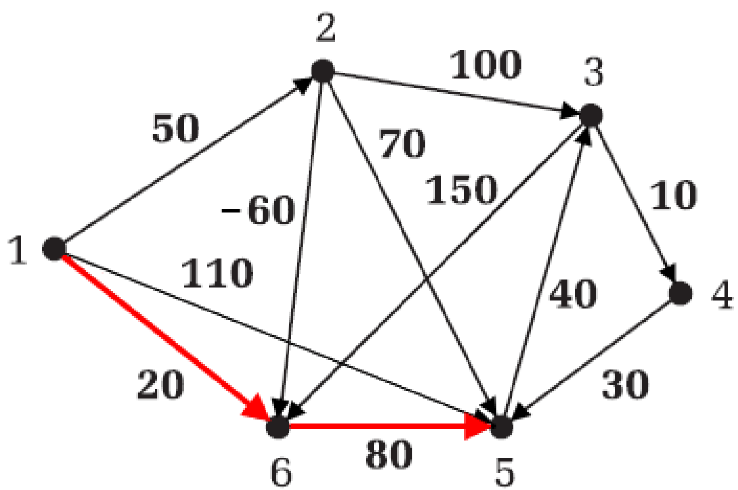


		1	2	3	4	5	6
0. menet		*0*	∞	∞	∞	∞	∞
	1:(2,6) [–60]	0	∞	∞	∞	∞	∞
	2:(1,5) [110]	0	∞	∞	∞	**110**	∞
	3:(1,6) [20]	0	∞	∞	∞	110	**20**
	4:(6,5) [80]	0	∞	∞	∞	**100**	20

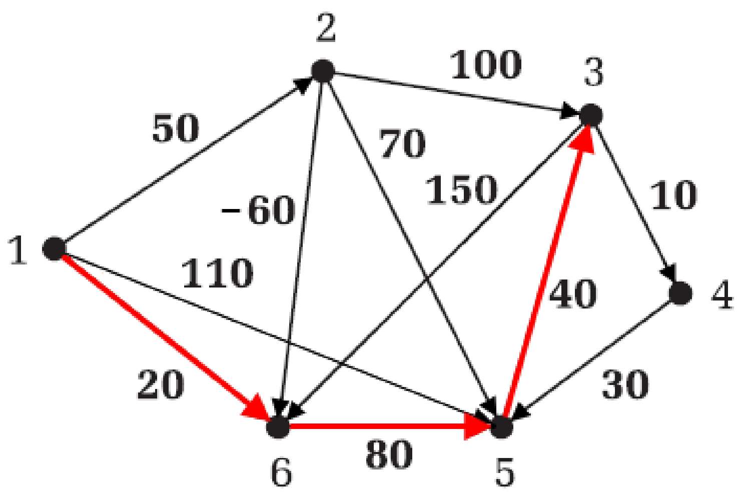
1. menet



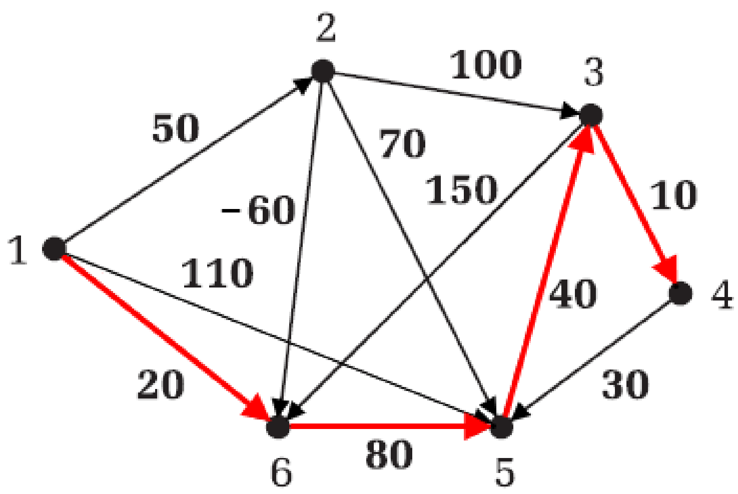
		1	2	3	4	5	6
0. menet		*0*	∞	∞	∞	∞	∞
	1:(2,6) [–60]	0	∞	∞	∞	∞	∞
	2:(1,5) [110]	0	∞	∞	∞	**110**	∞
	3:(1,6) [20]	0	∞	∞	∞	110	**20**
	4:(6,5) [80]	0	∞	∞	∞	**100**	20
1. menet	5:(2,3) [100]	0	∞	∞	∞	100	20



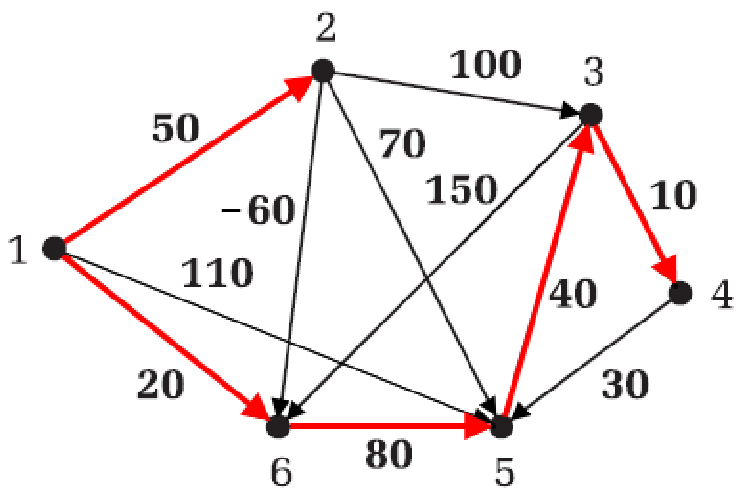
		1	2	3	4	5	6
0. menet		*0*	∞	∞	∞	∞	∞
	1:(2,6) [–60]	0	∞	∞	∞	∞	∞
	2:(1,5) [110]	0	∞	∞	∞	**110**	∞
	3:(1,6) [20]	0	∞	∞	∞	110	**20**
	4:(6,5) [80]	0	∞	∞	∞	**100**	20
	5:(2,3) [100]	0	∞	∞	∞	100	20
1. menet	6:(2,5) [70]	0	∞	∞	∞	100	20



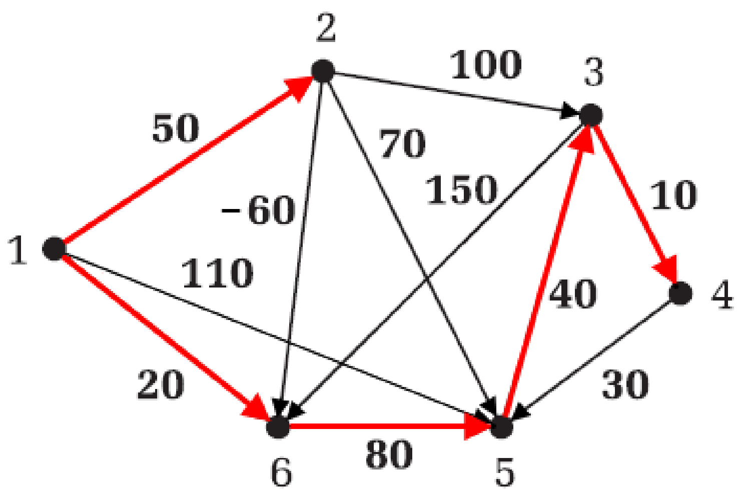
		1	2	3	4	5	6
0. menet		*0*	∞	∞	∞	∞	∞
	1:(2,6) [–60]	0	∞	∞	∞	∞	∞
	2:(1,5) [110]	0	∞	∞	∞	**110**	∞
	3:(1,6) [20]	0	∞	∞	∞	110	**20**
	4:(6,5) [80]	0	∞	∞	∞	**100**	20
1. menet	5:(2,3) [100]	0	∞	∞	∞	100	20
	6:(2,5) [70]	0	∞	∞	∞	100	20
	7:(5,3) [40]	0	∞	**140**	∞	100	20



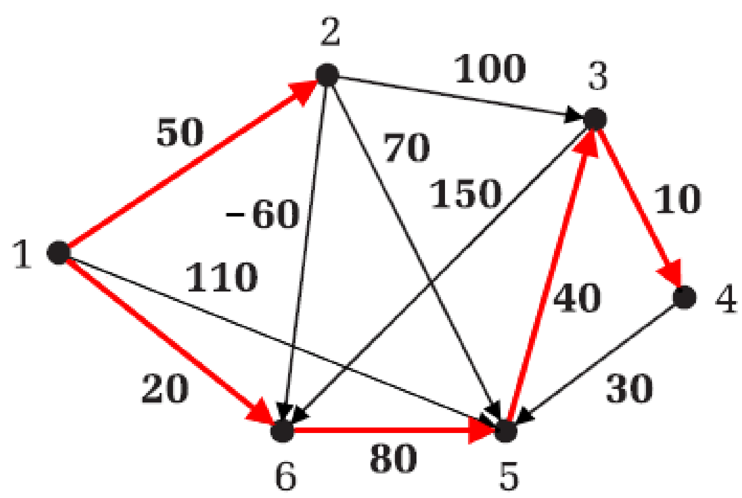
		1	2	3	4	5	6
0. menet		*0*	∞	∞	∞	∞	∞
1. menet	1:(2,6) [–60]	0	∞	∞	∞	∞	∞
	2:(1,5) [110]	0	∞	∞	∞	**110**	∞
	3:(1,6) [20]	0	∞	∞	∞	110	**20**
	4:(6,5) [80]	0	∞	∞	∞	**100**	20
	5:(2,3) [100]	0	∞	∞	∞	100	20
	6:(2,5) [70]	0	∞	∞	∞	100	20
	7:(5,3) [40]	0	∞	**140**	∞	100	20
	8:(3,4) [10]	0	∞	140	**150**	100	20



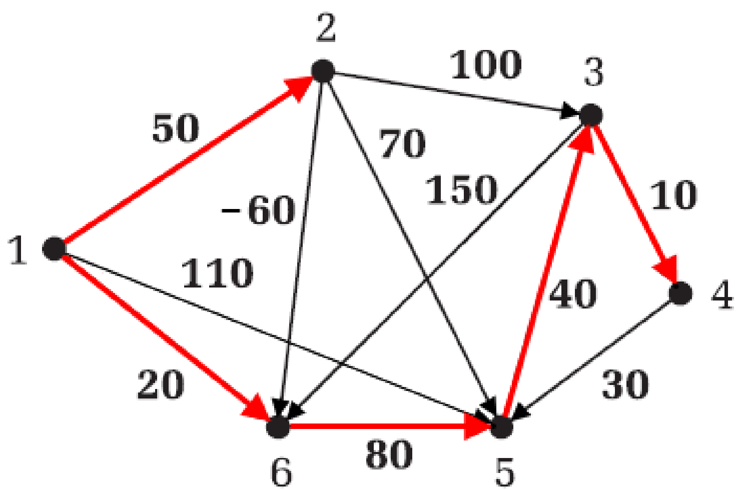
		1	2	3	4	5	6
0. menet		*0*	∞	∞	∞	∞	∞
1. menet	1:(2,6) [–60]	0	∞	∞	∞	∞	∞
	2:(1,5) [110]	0	∞	∞	∞	**110**	∞
	3:(1,6) [20]	0	∞	∞	∞	110	**20**
	4:(6,5) [80]	0	∞	∞	∞	**100**	20
	5:(2,3) [100]	0	∞	∞	∞	100	20
	6:(2,5) [70]	0	∞	∞	∞	100	20
	7:(5,3) [40]	0	∞	**140**	∞	100	20
	8:(3,4) [10]	0	∞	140	**150**	100	20
	9:(1,2) [50]	0	***50***	140	150	100	20



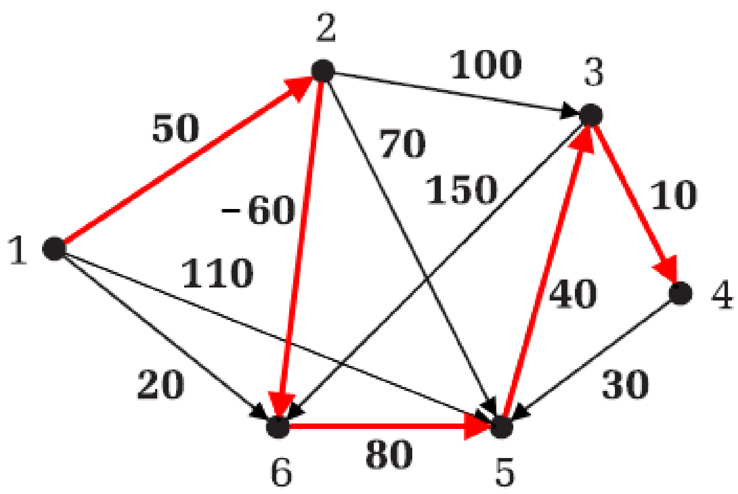
		1	2	3	4	5	6
0. menet		*0*	∞	∞	∞	∞	∞
1. menet	1:(2,6) [–60]	0	∞	∞	∞	∞	∞
	2:(1,5) [110]	0	∞	∞	∞	**110**	∞
	3:(1,6) [20]	0	∞	∞	∞	110	**20**
	4:(6,5) [80]	0	∞	∞	∞	**100**	20
	5:(2,3) [100]	0	∞	∞	∞	100	20
	6:(2,5) [70]	0	∞	∞	∞	100	20
	7:(5,3) [40]	0	∞	**140**	∞	100	20
	8:(3,4) [10]	0	∞	140	**150**	100	20
	9:(1,2) [50]	0	***50***	140	150	100	20
	10:(3,6) [150]	0	50	140	150	100	20



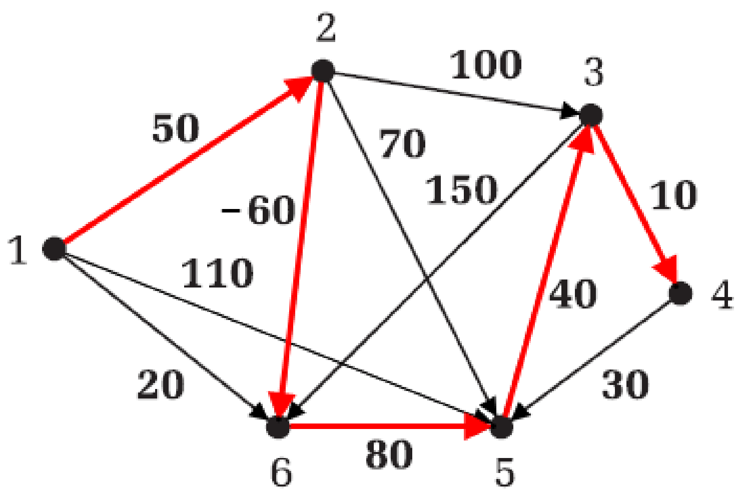
		1	2	3	4	5	6
0. menet		*0*	∞	∞	∞	∞	∞
1. menet	1:(2,6) [–60]	0	∞	∞	∞	∞	∞
	2:(1,5) [110]	0	∞	∞	∞	**110**	∞
	3:(1,6) [20]	0	∞	∞	∞	110	**20**
	4:(6,5) [80]	0	∞	∞	∞	**100**	20
	5:(2,3) [100]	0	∞	∞	∞	100	20
	6:(2,5) [70]	0	∞	∞	∞	100	20
	7:(5,3) [40]	0	∞	**140**	∞	100	20
	8:(3,4) [10]	0	∞	140	**150**	100	20
	9:(1,2) [50]	0	***50***	140	150	100	20
	10:(3,6) [150]	0	50	140	150	100	20
	11:(4,5) [30]	*0*	*50*	*140*	*150*	*100*	*20*



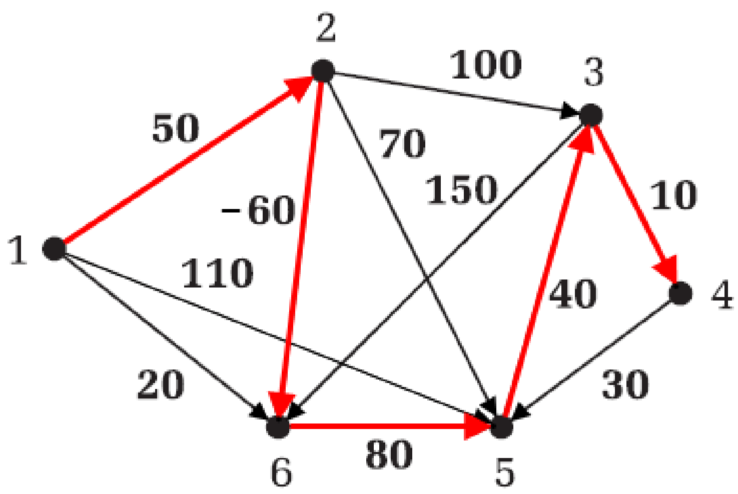
		1	2	3	4	5	6
0. menet		*0*	∞	∞	∞	∞	∞
1. menet	1:(2,6) [–60]	0	∞	∞	∞	∞	∞
	2:(1,5) [110]	0	∞	∞	∞	**110**	∞
	3:(1,6) [20]	0	∞	∞	∞	110	**20**
	4:(6,5) [80]	0	∞	∞	∞	**100**	20
	5:(2,3) [100]	0	∞	∞	∞	100	20
	6:(2,5) [70]	0	∞	∞	∞	100	20
	7:(5,3) [40]	0	∞	**140**	∞	100	20
	8:(3,4) [10]	0	∞	140	**150**	100	20
	9:(1,2) [50]	0	***50***	140	150	100	20
	10:(3,6) [150]	0	50	140	150	100	20
	11:(4,5) [30]	*0*	*50*	*140*	*150*	*100*	*20*
2. menet							



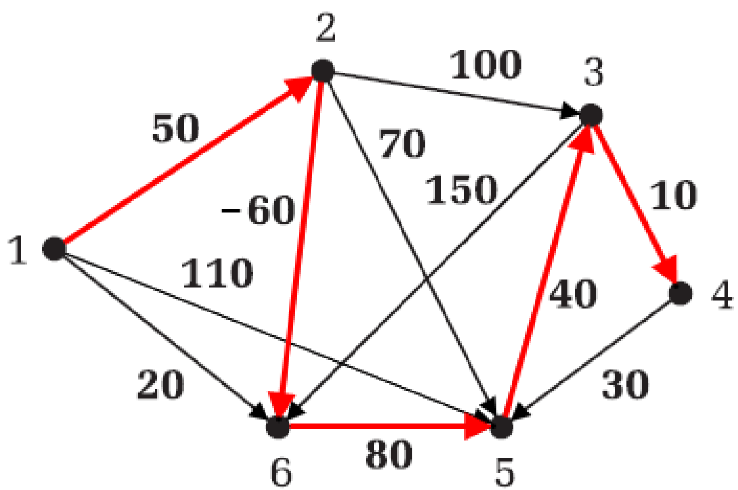
		1	2	3	4	5	6
0. menet		*0*	∞	∞	∞	∞	∞
1. menet	1:(2,6) [–60]	0	∞	∞	∞	∞	∞
	2:(1,5) [110]	0	∞	∞	∞	**110**	∞
	3:(1,6) [20]	0	∞	∞	∞	110	**20**
	4:(6,5) [80]	0	∞	∞	∞	**100**	20
	5:(2,3) [100]	0	∞	∞	∞	100	20
	6:(2,5) [70]	0	∞	∞	∞	100	20
	7:(5,3) [40]	0	∞	**140**	∞	100	20
	8:(3,4) [10]	0	∞	140	**150**	100	20
	9:(1,2) [50]	0	***50***	140	150	100	20
	10:(3,6) [150]	0	50	140	150	100	20
	11:(4,5) [30]	*0*	*50*	*140*	*150*	*100*	*20*
2. menet	**1:(2,6)** [–60]	0	50	140	150	100	**–10**



		1	2	3	4	5	6
0. menet		*0*	∞	∞	∞	∞	∞
1. menet	1:(2,6) [–60]	0	∞	∞	∞	∞	∞
	2:(1,5) [110]	0	∞	∞	∞	**110**	∞
	3:(1,6) [20]	0	∞	∞	∞	110	**20**
	4:(6,5) [80]	0	∞	∞	∞	**100**	20
	5:(2,3) [100]	0	∞	∞	∞	100	20
	6:(2,5) [70]	0	∞	∞	∞	100	20
	7:(5,3) [40]	0	∞	**140**	∞	100	20
	8:(3,4) [10]	0	∞	140	**150**	100	20
	9:(1,2) [50]	0	***50***	140	150	100	20
	10:(3,6) [150]	0	50	140	150	100	20
	11:(4,5) [30]	*0*	*50*	*140*	*150*	*100*	*20*
2. menet	**1:(2,6)** [–60]	0	50	140	150	100	**–10**
	2:(1,5) [110]	0	50	140	150	100	–10

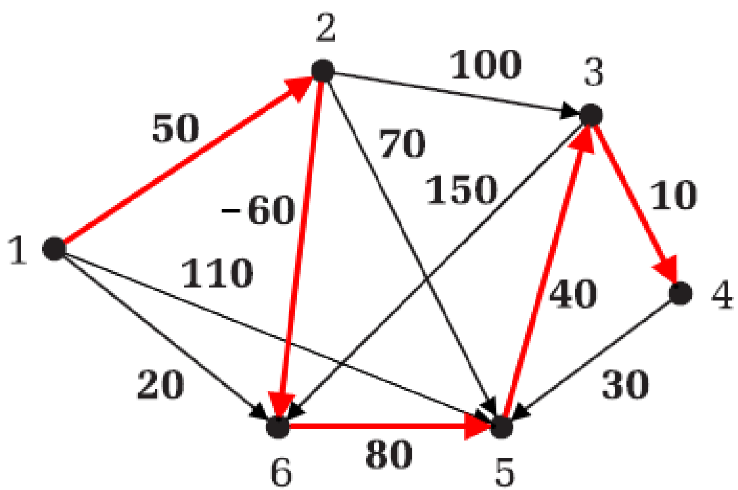


		1	2	3	4	5	6
0. menet		*0*	∞	∞	∞	∞	∞
1. menet	1:(2,6) [–60]	0	∞	∞	∞	∞	∞
	2:(1,5) [110]	0	∞	∞	∞	**110**	∞
	3:(1,6) [20]	0	∞	∞	∞	110	**20**
	4:(6,5) [80]	0	∞	∞	∞	**100**	20
	5:(2,3) [100]	0	∞	∞	∞	100	20
	6:(2,5) [70]	0	∞	∞	∞	100	20
	7:(5,3) [40]	0	∞	**140**	∞	100	20
	8:(3,4) [10]	0	∞	140	**150**	100	20
	9:(1,2) [50]	0	***50***	140	150	100	20
	10:(3,6) [150]	0	50	140	150	100	20
	11:(4,5) [30]	*0*	*50*	*140*	*150*	*100*	*20*
2. menet	**1:(2,6)** [–60]	0	50	140	150	100	**–10**
	2:(1,5) [110]	0	50	140	150	100	–10
	3:(1,6) [20]	0	50	140	150	100	–10

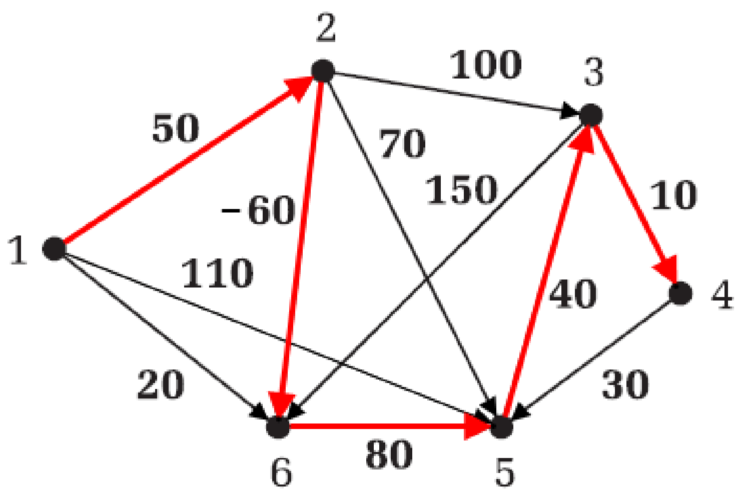


		1	2	3	4	5	6
0. menet		*0*	∞	∞	∞	∞	∞
	1:(2,6) [–60]	0	∞	∞	∞	∞	∞
	2:(1,5) [110]	0	∞	∞	∞	**110**	∞
	3:(1,6) [20]	0	∞	∞	∞	110	**20**
	4:(6,5) [80]	0	∞	∞	∞	**100**	20
	5:(2,3) [100]	0	∞	∞	∞	100	20
1. menet	6:(2,5) [70]	0	∞	∞	∞	100	20
	7:(5,3) [40]	0	∞	**140**	∞	100	20
	8:(3,4) [10]	0	∞	140	**150**	100	20
	9:(1,2) [50]	0	***50***	140	150	100	20
	10:(3,6) [150]	0	50	140	150	100	20
	11:(4,5) [30]	*0*	*50*	*140*	*150*	*100*	*20*
	1:(2,6) [–60]	0	50	140	150	100	**–10**
	2:(1,5) [110]	0	50	140	150	100	–10
	3:(1,6) [20]	0	50	140	150	100	–10
	4:(6,5) [80]	0	50	140	150	**70**	–10

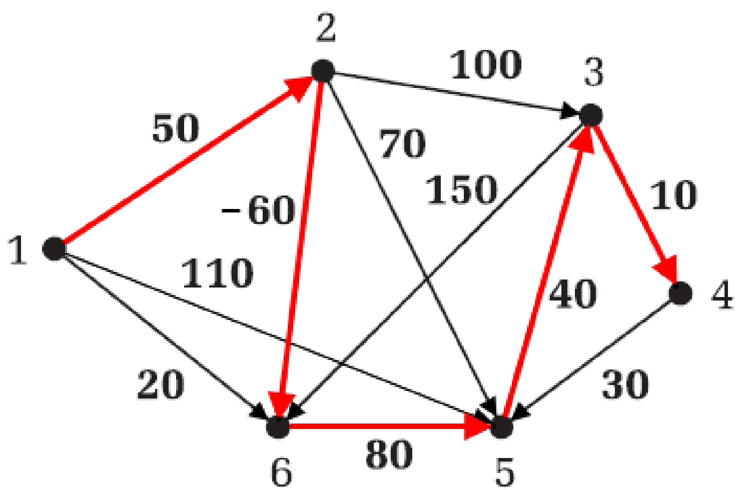
2. menet



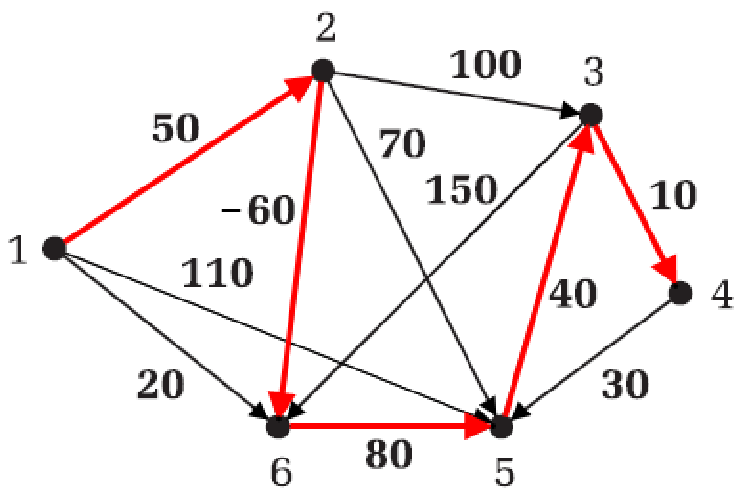
		1	2	3	4	5	6
0. menet		*0*	∞	∞	∞	∞	∞
1. menet	1:(2,6) [–60]	0	∞	∞	∞	∞	∞
	2:(1,5) [110]	0	∞	∞	∞	**110**	∞
	3:(1,6) [20]	0	∞	∞	∞	110	**20**
	4:(6,5) [80]	0	∞	∞	∞	**100**	20
	5:(2,3) [100]	0	∞	∞	∞	100	20
	6:(2,5) [70]	0	∞	∞	∞	100	20
	7:(5,3) [40]	0	∞	**140**	∞	100	20
	8:(3,4) [10]	0	∞	140	**150**	100	20
	9:(1,2) [50]	0	***50***	140	150	100	20
	10:(3,6) [150]	0	50	140	150	100	20
	11:(4,5) [30]	*0*	*50*	*140*	*150*	*100*	*20*
2. menet	**1:(2,6)** [–60]	0	50	140	150	100	**–10**
	2:(1,5) [110]	0	50	140	150	100	–10
	3:(1,6) [20]	0	50	140	150	100	–10
	4:(6,5) [80]	0	50	140	150	**70**	–10
	5:(2,3) [100]	0	50	140	150	70	–10



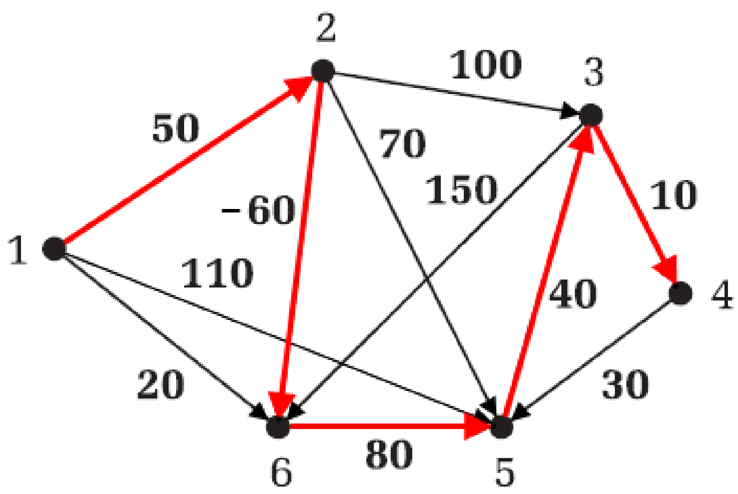
		1	2	3	4	5	6
0. menet		*0*	∞	∞	∞	∞	∞
1. menet	1:(2,6) [–60]	0	∞	∞	∞	∞	∞
	2:(1,5) [110]	0	∞	∞	∞	**110**	∞
	3:(1,6) [20]	0	∞	∞	∞	110	**20**
	4:(6,5) [80]	0	∞	∞	∞	**100**	20
	5:(2,3) [100]	0	∞	∞	∞	100	20
	6:(2,5) [70]	0	∞	∞	∞	100	20
	7:(5,3) [40]	0	∞	**140**	∞	100	20
	8:(3,4) [10]	0	∞	140	**150**	100	20
	9:(1,2) [50]	0	***50***	140	150	100	20
	10:(3,6) [150]	0	50	140	150	100	20
	11:(4,5) [30]	*0*	*50*	*140*	*150*	*100*	*20*
2. menet	**1:(2,6)** [–60]	0	50	140	150	100	**–10**
	2:(1,5) [110]	0	50	140	150	100	–10
	3:(1,6) [20]	0	50	140	150	100	–10
	4:(6,5) [80]	0	50	140	150	**70**	–10
	5:(2,3) [100]	0	50	140	150	70	–10
	6:(2,5) [70]	0	50	140	150	70	–10



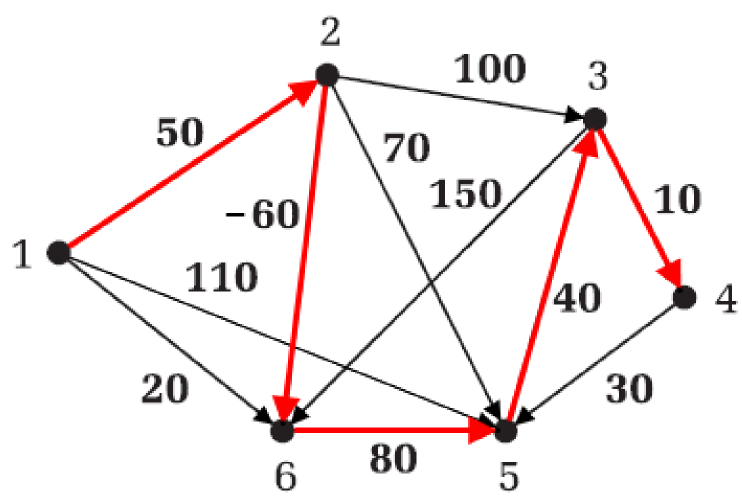
		1	2	3	4	5	6
0. menet		*0*	∞	∞	∞	∞	∞
1. menet	1:(2,6) [–60]	0	∞	∞	∞	∞	∞
	2:(1,5) [110]	0	∞	∞	∞	**110**	∞
	3:(1,6) [20]	0	∞	∞	∞	110	**20**
	4:(6,5) [80]	0	∞	∞	∞	**100**	20
	5:(2,3) [100]	0	∞	∞	∞	100	20
	6:(2,5) [70]	0	∞	∞	∞	100	20
	7:(5,3) [40]	0	∞	**140**	∞	100	20
	8:(3,4) [10]	0	∞	140	**150**	100	20
	9:(1,2) [50]	0	***50***	140	150	100	20
	10:(3,6) [150]	0	50	140	150	100	20
	11:(4,5) [30]	*0*	*50*	*140*	*150*	*100*	*20*
2. menet	**1:(2,6)** [–60]	0	50	140	150	100	**–10**
	2:(1,5) [110]	0	50	140	150	100	–10
	3:(1,6) [20]	0	50	140	150	100	–10
	4:(6,5) [80]	0	50	140	150	**70**	–10
	5:(2,3) [100]	0	50	140	150	70	–10
	6:(2,5) [70]	0	50	140	150	70	–10
	7:(5,3) [40]	0	50	**110**	150	70	–10



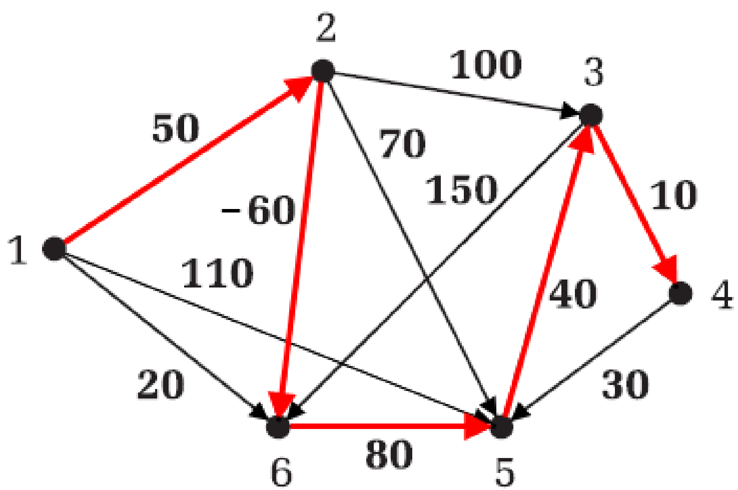
		1	2	3	4	5	6
0. menet		*0*	∞	∞	∞	∞	∞
1. menet	1:(2,6) [–60]	0	∞	∞	∞	∞	∞
	2:(1,5) [110]	0	∞	∞	∞	**110**	∞
	3:(1,6) [20]	0	∞	∞	∞	110	**20**
	4:(6,5) [80]	0	∞	∞	∞	**100**	20
	5:(2,3) [100]	0	∞	∞	∞	100	20
	6:(2,5) [70]	0	∞	∞	∞	100	20
	7:(5,3) [40]	0	∞	**140**	∞	100	20
	8:(3,4) [10]	0	∞	140	**150**	100	20
	9:(1,2) [50]	0	***50***	140	150	100	20
	10:(3,6) [150]	0	50	140	150	100	20
	11:(4,5) [30]	*0*	*50*	*140*	*150*	*100*	*20*
2. menet	**1:(2,6)** [–60]	0	50	140	150	100	**–10**
	2:(1,5) [110]	0	50	140	150	100	–10
	3:(1,6) [20]	0	50	140	150	100	–10
	4:(6,5) [80]	0	50	140	150	**70**	–10
	5:(2,3) [100]	0	50	140	150	70	–10
	6:(2,5) [70]	0	50	140	150	70	–10
	7:(5,3) [40]	0	50	**110**	150	70	–10
	8:(3,4) [10]	0	50	110	**120**	70	–10



		1	2	3	4	5	6
0. menet		*0*	∞	∞	∞	∞	∞
1. menet	1:(2,6) [–60]	0	∞	∞	∞	∞	∞
	2:(1,5) [110]	0	∞	∞	∞	**110**	∞
	3:(1,6) [20]	0	∞	∞	∞	110	**20**
	4:(6,5) [80]	0	∞	∞	∞	**100**	20
	5:(2,3) [100]	0	∞	∞	∞	100	20
	6:(2,5) [70]	0	∞	∞	∞	100	20
	7:(5,3) [40]	0	∞	**140**	∞	100	20
	8:(3,4) [10]	0	∞	140	**150**	100	20
	9:(1,2) [50]	0	***50***	140	150	100	20
	10:(3,6) [150]	0	50	140	150	100	20
	11:(4,5) [30]	*0*	*50*	*140*	*150*	*100*	*20*
2. menet	**1:(2,6)** [–60]	0	50	140	150	100	**–10**
	2:(1,5) [110]	0	50	140	150	100	–10
	3:(1,6) [20]	0	50	140	150	100	–10
	4:(6,5) [80]	0	50	140	150	**70**	–10
	5:(2,3) [100]	0	50	140	150	70	–10
	6:(2,5) [70]	0	50	140	150	70	–10
	7:(5,3) [40]	0	50	**110**	150	70	–10
	8:(3,4) [10]	0	50	110	**120**	70	–10
	9:(1,2) [50]	0	50	110	120	70	–10



		1	2	3	4	5	6
0. menet		*0*	∞	∞	∞	∞	∞
1. menet	1:(2,6) [–60]	0	∞	∞	∞	∞	∞
	2:(1,5) [110]	0	∞	∞	∞	**110**	∞
	3:(1,6) [20]	0	∞	∞	∞	110	**20**
	4:(6,5) [80]	0	∞	∞	∞	**100**	20
	5:(2,3) [100]	0	∞	∞	∞	100	20
	6:(2,5) [70]	0	∞	∞	∞	100	20
	7:(5,3) [40]	0	∞	**140**	∞	100	20
	8:(3,4) [10]	0	∞	140	**150**	100	20
	9:(1,2) [50]	0	***50***	140	150	100	20
	10:(3,6) [150]	0	50	140	150	100	20
	11:(4,5) [30]	*0*	*50*	*140*	*150*	*100*	*20*
2. menet	**1:(2,6)** [–60]	0	50	140	150	100	**–10**
	2:(1,5) [110]	0	50	140	150	100	–10
	3:(1,6) [20]	0	50	140	150	100	–10
	4:(6,5) [80]	0	50	140	150	**70**	–10
	5:(2,3) [100]	0	50	140	150	70	–10
	6:(2,5) [70]	0	50	140	150	70	–10
	7:(5,3) [40]	0	50	**110**	150	70	–10
	8:(3,4) [10]	0	50	110	**120**	70	–10
	9:(1,2) [50]	0	50	110	120	70	–10
	10:(3,6) [150]	0	50	110	120	70	–10



		1	2	3	4	5	6
0. menet		*0*	∞	∞	∞	∞	∞
1. menet	1:(2,6) [–60]	0	∞	∞	∞	∞	∞
	2:(1,5) [110]	0	∞	∞	∞	**110**	∞
	3:(1,6) [20]	0	∞	∞	∞	110	**20**
	4:(6,5) [80]	0	∞	∞	∞	**100**	20
	5:(2,3) [100]	0	∞	∞	∞	100	20
	6:(2,5) [70]	0	∞	∞	∞	100	20
	7:(5,3) [40]	0	∞	**140**	∞	100	20
	8:(3,4) [10]	0	∞	140	**150**	100	20
	9:(1,2) [50]	0	***50***	140	150	100	20
	10:(3,6) [150]	0	50	140	150	100	20
	11:(4,5) [30]	*0*	*50*	*140*	*150*	*100*	*20*
2. menet	**1:(2,6)** [–60]	0	50	140	150	100	**–10**
	2:(1,5) [110]	0	50	140	150	100	–10
	3:(1,6) [20]	0	50	140	150	100	–10
	4:(6,5) [80]	0	50	140	150	**70**	–10
	5:(2,3) [100]	0	50	140	150	70	–10
	6:(2,5) [70]	0	50	140	150	70	–10
	7:(5,3) [40]	0	50	**110**	150	70	–10
	8:(3,4) [10]	0	50	110	**120**	70	–10
	9:(1,2) [50]	0	50	110	120	70	–10
	10:(3,6) [150]	0	50	110	120	70	–10
	11:(4,5) [30]	*0*	*50*	*110*	*120*	*70*	*–10*

Milyen esetben van szüksége a Bellman-Ford algoritmusnak $n-1$ *menetre*?

Milyen esetben van szüksége a Bellman-Ford algoritmusnak $n-1$ *menetre*?

Tegyük fel, hogy létezik egy $n-1$ hosszú legrövidebb út. Legyen ez a $(v_1 = s, v_2, \ldots, v_i, v_{i+1}, \ldots, v_n)$ út.

Milyen esetben van szüksége a Bellman-Ford algoritmusnak $n-1$ *menetre*?

Tegyük fel, hogy létezik egy $n-1$ hosszú legrövidebb út. Legyen ez a $(v_1 = s, v_2, \ldots, v_i, v_{i+1}, \ldots, v_n)$ út.

Természetesen a v_i csúcsponthoz vezető legrövidebb út csak azután határozható meg, ha a v_{i-1} csúcsponthoz vezető út már a rendelkezésre áll.

Milyen esetben van szüksége a Bellman-Ford algoritmusnak $n-1$ *menetre*?

Tegyük fel, hogy létezik egy $n-1$ hosszú legrövidebb út. Legyen ez a $(v_1 = s, v_2, \ldots, v_i, v_{i+1}, \ldots, v_n)$ út.

Természetesen a v_i csúcsponthoz vezető legrövidebb út csak azután határozható meg, ha a v_{i-1} csúcsponthoz vezető út már a rendelkezésre áll.

Nézzük meg mi történik, ha az egyes menetekben a legrövidebb utat alkotó élekkel a (v_{n-1}, v_n), (v_{n-2}, v_{n-1}), $\ldots$, (v_1, v_2) sorrendben (az úton való előfordulásuk fordított sorrendjében) próbálunk közelíteni:

Milyen esetben van szüksége a Bellman-Ford algoritmusnak $n-1$ *menetre*?

Tegyük fel, hogy létezik egy $n-1$ hosszú legrövidebb út. Legyen ez a $(v_1 = s, v_2, \ldots, v_i, v_{i+1}, \ldots, v_n)$ út.

Természetesen a v_i csúcsponthoz vezető legrövidebb út csak azután határozható meg, ha a v_{i-1} csúcsponthoz vezető út már a rendelkezésre áll.

Nézzük meg mi történik, ha az egyes menetekben a legrövidebb utat alkotó élekkel a (v_{n-1}, v_n), (v_{n-2}, v_{n-1}), …, (v_1, v_2) sorrendben (az úton való előfordulásuk fordított sorrendjében) próbálunk közelíteni: mivel a (v_{i-1}, v_i) éllel való végső közelítésre csak a (v_{i-2}, v_{i-1}) éllel való végső közelítés után kerülhet sor, ezért az első menetben, csak a v_2 csúcsponthoz tudjuk meghatározni a legrövidebb utat, majd a második menetben v_3-hoz, s végül az $(n-1)$-edik menetben v_n-hez.