GRÁFELMÉLET

Minimális súlyú feszítőfák

4. előadás

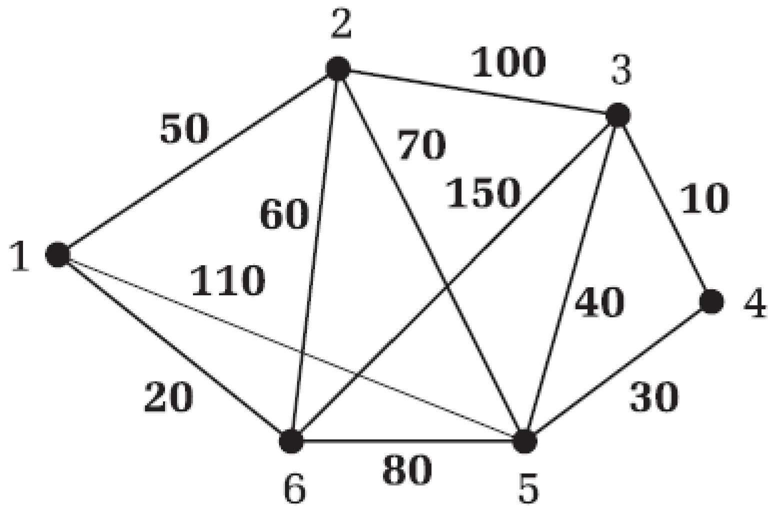
Probléma: Egy szolgáltatatási központnak (*vízvezeték*, *gázvezeték*, *telefon*) úgy kell bekapcsolnia a fogyasztókat a rendszerbe, hogy az ellátás ne történjen körforgalomban. Azt a megoldást kell megtalálni, amelyikhez a legkevesebb vezetékre van szükség.

Probléma: Egy szolgáltatatási központnak (*vízvezeték*, *gázvezeték*, *telefon*) úgy kell bekapcsolnia a fogyasztókat a rendszerbe, hogy az ellátás ne történjen körforgalomban. Azt a megoldást kell megtalálni, amelyikhez a legkevesebb vezetékre van szükség.

Megoldás: A problémát egy összefüggő irányítatlan $G = (V, E)$ gráf segítségével modellezhetjük, melynek minden éléhez **súlyok**at ($súly: E \rightarrow R$) rendelünk. Határozzuk meg a G gráf minimális súlyú feszítőfáját!

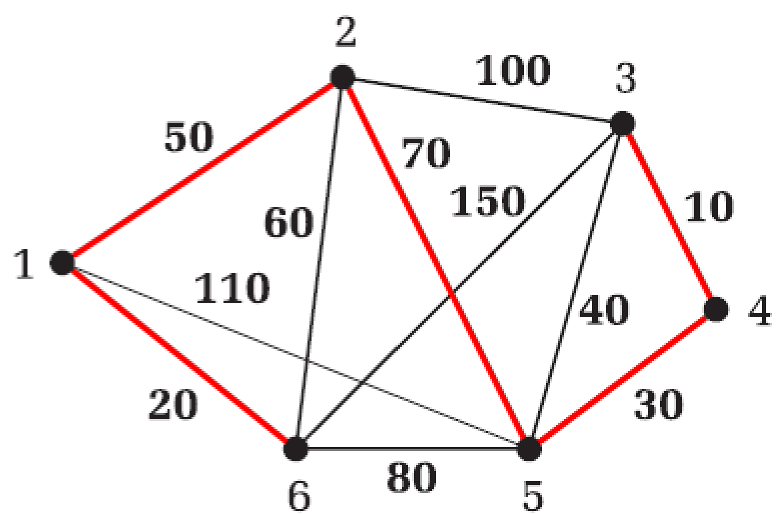
Probléma: Egy szolgáltatatási központnak (*vízvezeték*, *gázvezeték*, *telefon*) úgy kell bekapcsolnia a fogyasztókat a rendszerbe, hogy az ellátás ne történjen körforgalomban. Azt a megoldást kell megtalálni, amelyikhez a legkevesebb vezetékre van szükség.

Megoldás: A problémát egy összefüggő irányítatlan $G = (V, E)$ gráf segítségével modellezhetjük, melynek minden éléhez **súlyok**at ($súly$: $E \rightarrow R$) rendelünk. Határozzuk meg a G gráf minimális súlyú feszítőfáját!



Probléma: Egy szolgáltatatási központnak (*vízvezeték*, *gázvezeték*, *telefon*) úgy kell bekapcsolnia a fogyasztókat a rendszerbe, hogy az ellátás ne történjen körforgalomban. Azt a megoldást kell megtalálni, amelyikhez a legkevesebb vezetékre van szükség.

Megoldás: A problémát egy összefüggő irányítatlan $G = (V, E)$ gráf segítségével modellezhetjük, melynek minden éléhez **súlyok**at ($súly\colon E \to R$) rendelünk. Határozzuk meg a G gráf minimális súlyú feszítőfáját!



A probléma megoldására két algoritmust ismertetünk, amelyek J. B. KRUSKAL és R. C. PRIM amerikai matematikusok nevéhez fűződnek.

A probléma megoldására két algoritmust ismertetünk, amelyek J. B. KRUSKAL és R. C. PRIM amerikai matematikusok nevéhez fűződnek.

Mindkét algoritmus az ún. **mohó stratégián** alapul. Általában egy algoritmus az egyes lépések során több lehetőség közül választ. A mohó stratégia mindig azt a lépést részesíti előnyben, amelyik az adott pillanatban a legjobbnak látszik.

Ez a stratégia általában nem garantálja az optimális megoldás megtalálását. A minimális súlyú feszítőfa probléma esetében azonban igazolható, hogy bizonyos mohó stratégiák a kívánt megoldást állítják elő.

1) Kruskal algoritmusa

Stratégia: Gondolatban eltávolítjuk a gráf összes élét, majd súlyuk szerint növekvő sorrendben visszatesszük őket, átugorva azokat az éleket, amelyek kört hoznának létre. A keresett minimális súlyú feszítőfát az A halmaz fogja tartalmazni mint éllistát.

1) Kruskal algoritmusa

Stratégia: Gondolatban eltávolítjuk a gráf összes élét, majd súlyuk szerint növekvő sorrendben visszatesszük őket, átugorva azokat az éleket, amelyek kört hoznának létre. A keresett minimális súlyú feszítőfát az A halmaz fogja tartalmazni mint éllistát.

Mivel gondolatban minden élet eltávolítottunk, kezdetben a gráf csúcspontjai úgy foghatók fel mint különálló fák. Úgy is mondhatnánk, hogy van egy n darab egycsúcspontú fából álló erdőnk. Ezen erdő egyes fáihoz tartozó csúcsokat diszjunkt halmazokban tároljuk. Kezdetben természetesen minden csúcs külön halmazba kerül (HALMAZT_KÉSZÍT eljárás).

1) Kruskal algoritmusa

Stratégia: Gondolatban eltávolítjuk a gráf összes élét, majd súlyuk szerint növekvő sorrendben visszatesszük őket, átugorva azokat az éleket, amelyek kört hoznának létre. A keresett minimális súlyú feszítőfát az A halmaz fogja tartalmazni mint éllistát.

Mivel gondolatban minden élet eltávolítottunk, kezdetben a gráf csúcspontjai úgy foghatók fel mint különálló fák. Úgy is mondhatnánk, hogy van egy n darab egycsúcspontú fából álló erdőnk. Ezen erdő egyes fáihoz tartozó csúcsokat diszjunkt halmazokban tároljuk. Kezdetben természetesen minden csúcs külön halmazba kerül (HALMAZT_KÉSZÍT eljárás).

A soron következő él akkor nem alkot kört a már visszahelyezett élekkel, ha végpontjai két különböző fához tartoznak, azaz más-más halmazban vannak (ez a HOLVAN függvény segítségével ellenőrizzük.)

1) Kruskal algoritmusa

Az él visszahelyezésével viszont egyesül az illető két fa. Ezt úgy valósítjuk meg, hogy egyesítjük a két végpont halmazait (ÚNIÓ eljárás).

1) Kruskal algoritmusa

Az él visszahelyezésével viszont egyesül az illető két fa. Ezt úgy valósítjuk meg, hogy egyesítjük a két végpont halmazait (ÚNIÓ eljárás).

Az algoritmus futása végén az erdő fái egyetlen fává egyesülnek, melyet a visszahelyezett élek fognak alkotni. Ez lesz a keresett minimális súlyú feszítőfa.

```
függvény KRUSKAL(G)
    A = ∅
    minden u ∈ V(G) végezd
        HALMAZT_KÉSZÍT(u)
                  // minden pontot külön halmazokba helyezünk el
    vége minden
    RENDEZÉS_SÚLY_SZERINT(E(G))
    minden (u,v) ∈ E(G) végezd
        ha HOLVAN(u) ≠ HOLVAN(v) akkor
                                // u és v más-más halmazban vannak
            A = A ∪ {\{}(u,v){\}}
            UNIÓ(u,v)               // egyesítjük u és v halmazait
        vége ha
    vége minden
    vissza A
vége KRUSKAL
```

```
függvény KRUSKAL(G)
   A = ∅
   minden u ∈ V(G) végezd
      HALMAZT_KÉSZÍT(u)
             // minden pontot külön halmazokba helyezünk el
   vége minden
   RENDEZÉS_SÚLY_SZERINT(E(G))
   minden (u,v) ∈ E(G) végezd
      ha HOLVAN(u) ≠ HOLVAN(v) akkor
                         // u és v más-más halmazban vannak
         A = A ∪ {\{}(u,v){\}}
         UNIÓ(u,v)             // egyesítjük u és v halmazait
      vége ha
   vége minden
   vissza A
vége KRUSKAL
```

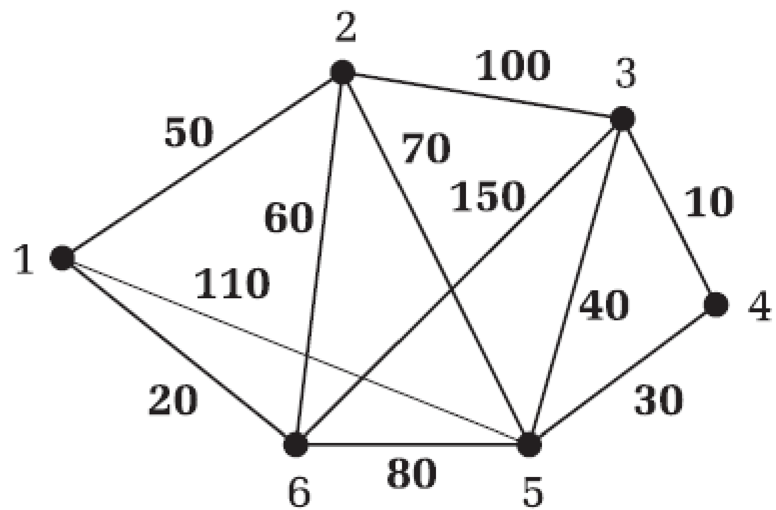
A KRUSKAL algoritmus bonyolultsága $O(m \log m)$.

A továbbiakban bemutatunk egy egyszerűbb implementációt. A `fa[1..n]` tömb fogja nyilvántartani, hogy az egyes csúcspontok az erdőnek éppen melyik fájához tartoznak (az i-edik csúcspont a `fa[i]` fához). Kezdetben minden csúcspont külön fa (az i-edik csúcspont az i-edik fa: `fa[i]=i`).

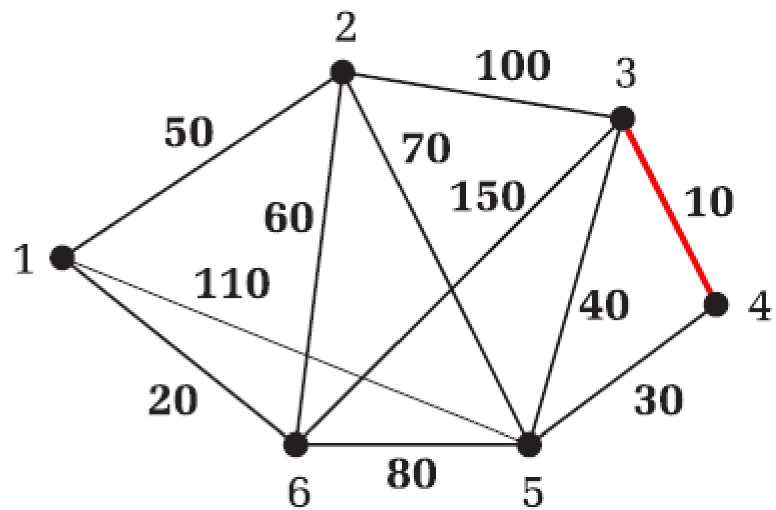
Két fa egyesítését az EGYESÍT eljárás valósítja meg. Az éleket, mint éllistát, az `élek[1..m]` tömb tárolja. Minden élről eltároljuk a kezdőpontját, a végpontját és a súlyát. Az KRUSKAL eljárás kiírja a G gráf minimális súlyú feszítőfáját alkotó éleket, ezek súlyát, illetve a feszítőfa összsúlyát.

```
eljárás KRUSKAL(G)
    minden i ← 1,n végezd
        fa[i] ← i
    vége minden
    RENDEZÉS_SÚLY_SZERINT(élek,m)
    össz_súly ← 0
    minden i ← 1,m végezd
        egyik = fa[élek[i].u]
        másik = fa[élek[i].v]
        ha egyik ≠ másik akkor
            össz_súly ← össz_súly + élek[i].súly
            kiír: élek[i].u, élek[i].v, élek[i].súly
            EGYESÍT(fa,n,egyik,másik)
        vége ha
    vége minden
    kiír: össz_súly
vége KRUSKAL
```

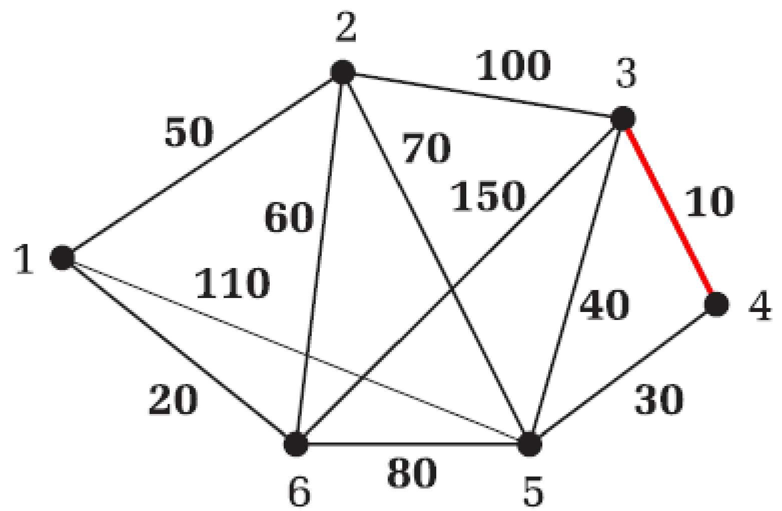
```
eljárás EGYESÍT(fa,n,egyik,másik)
    minden i ← 1,n végezd
        ha fa[i] = másik akkor
            fa[i] ← egyik
        vége ha
    vége minden
vége EGYESÍT
```



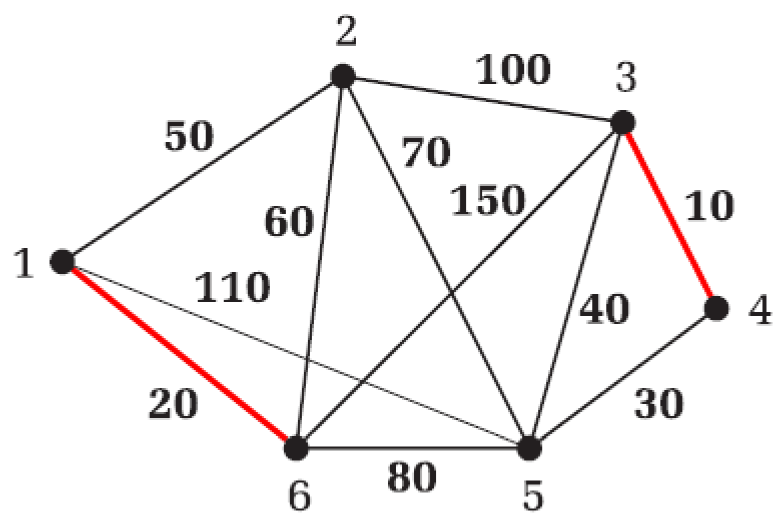
1	2	3	4	5	6
1	2	3	4	5	6



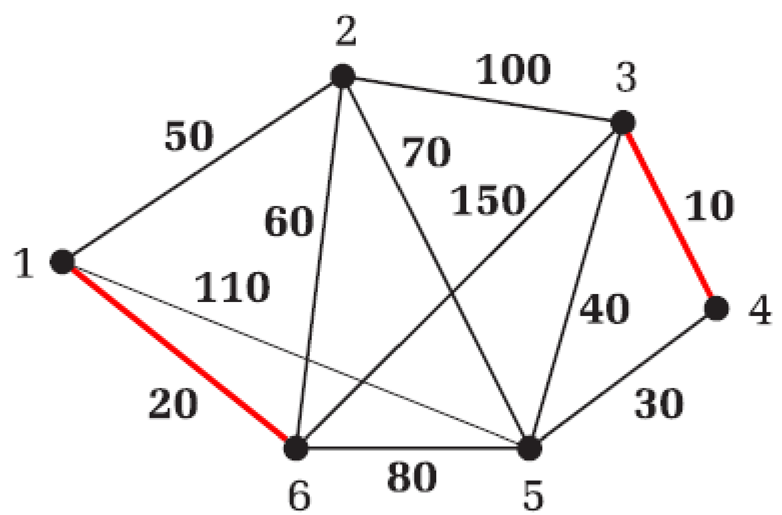
1	2	3	4	5	6
1	2	3	4	5	6



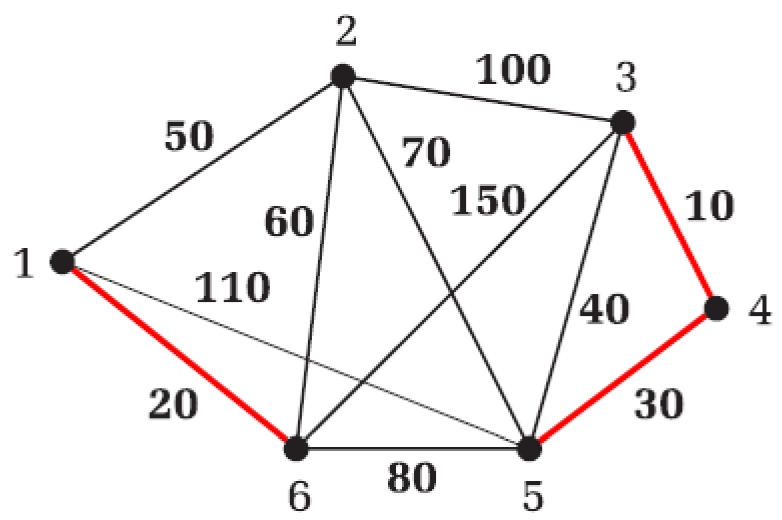
1	2	3	3	5	6
1	2	3	4	5	6



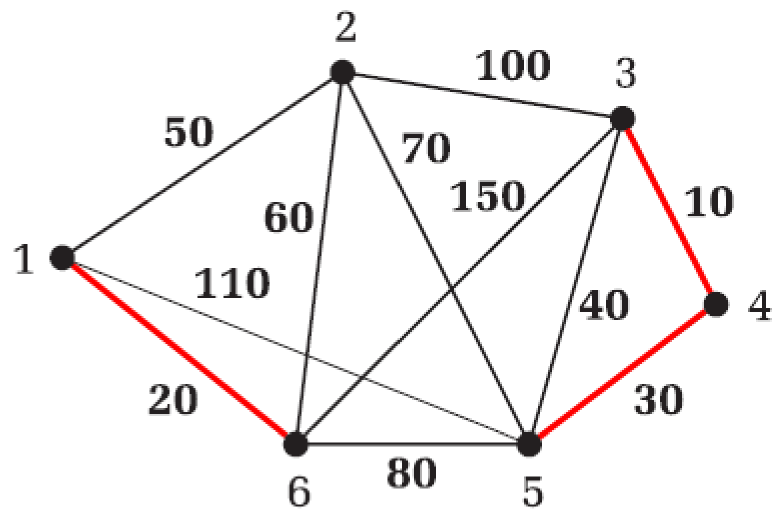
1	**2**	**3**	**3**	**5**	**6**
1	2	3	4	5	6



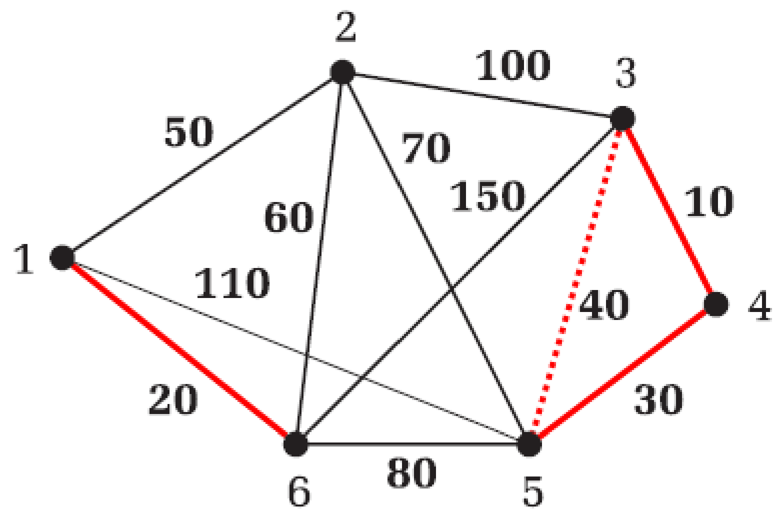
1	2	3	3	5	1
1	2	3	4	5	6



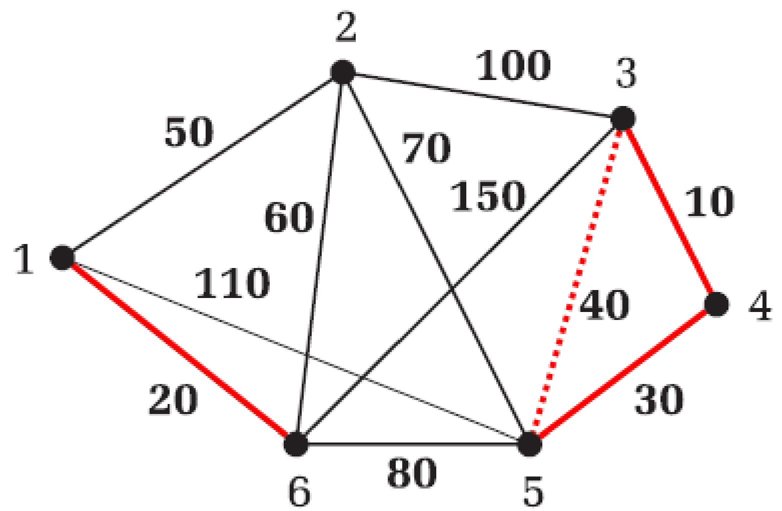
1	**2**	**3**	**3**	**5**	**1**
1	2	3	4	5	6



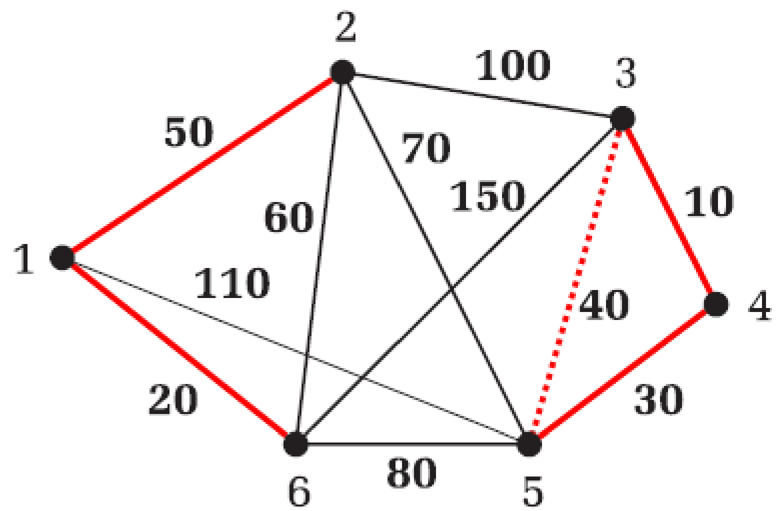
1	**2**	**3**	**3**	**3**	**1**
1	2	3	4	5	6



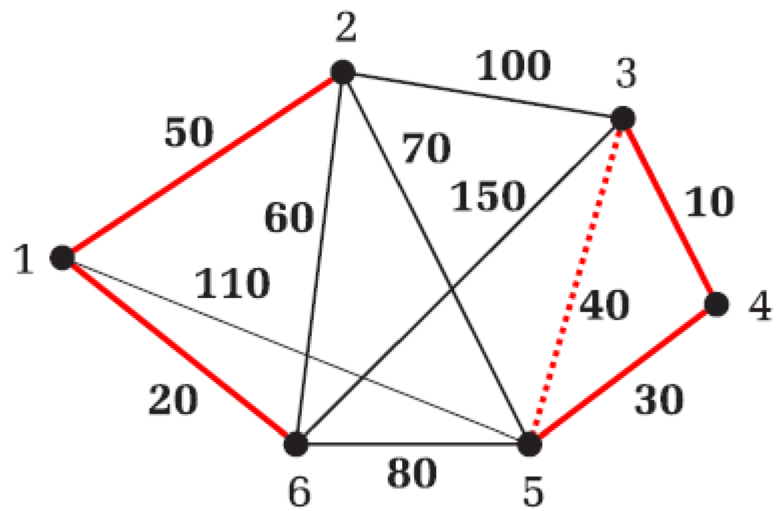
1	**2**	**3**	**3**	**3**	**1**
1	2	3	4	5	6



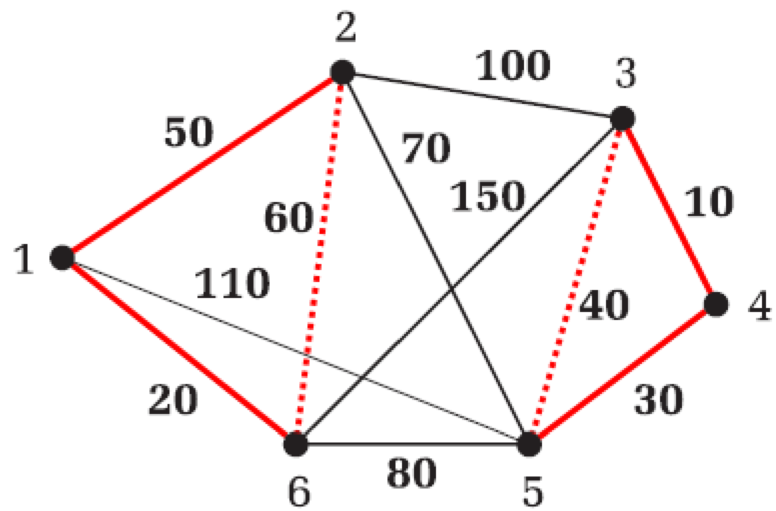
1	**2**	**3**	**3**	**3**	**1**
1	2	3	4	5	6



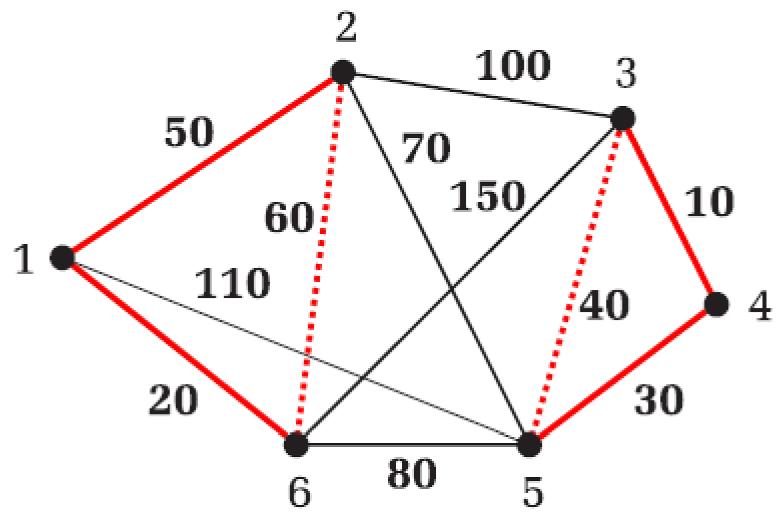
1	**2**	**3**	**3**	**3**	**1**
1	2	3	4	5	6



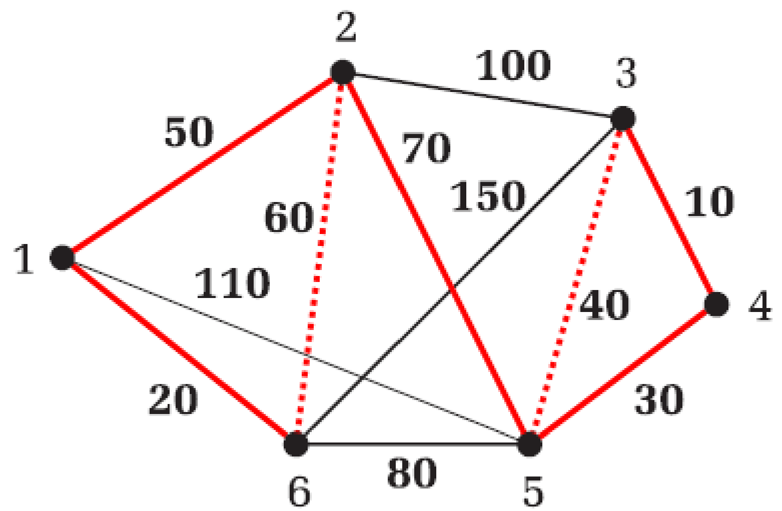
1	1	3	3	3	1
1	2	3	4	5	6



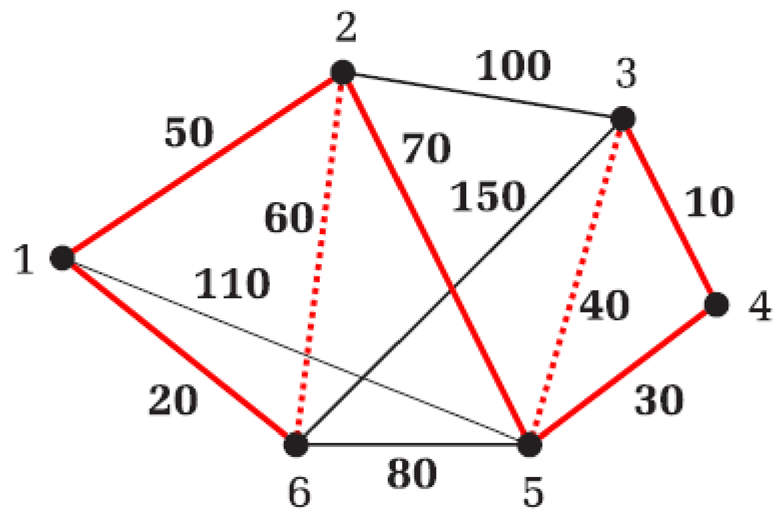
1	**1**	**3**	**3**	**3**	**1**
1	2	3	4	5	6



1	1	3	3	3	1
1	2	3	4	5	6



1	1	3	3	3	1
1	2	3	4	5	6



1	1	1	1	1	1
1	2	3	4	5	6

2) Prim algoritmusa

Stratégia: Gondolatban eltávolítjuk a gráf összes élét, majd a minimális súlyú feszítőfát egy r gyökerű faként építjük fel az élek fokozatos visszahelyezésével. A fa építése az r csúcspontból indul, s akkor fejeződik be, amikor minden csúcs belekerül a fába. Minden lépésben azt a legkisebb súlyú élt rakjuk vissza, amelyik egy fabeli csúcspontot köt össze egy izolált ponttal.

2) Prim algoritmusa

Stratégia: Gondolatban eltávolítjuk a gráf összes élét, majd a minimális súlyú feszítőfát egy r gyökerű faként építjük fel az élek fokozatos visszahelyezésével. A fa építése az r csúcspontból indul, s akkor fejeződik be, amikor minden csúcs belekerül a fába. Minden lépésben azt a legkisebb súlyú élt rakjuk vissza, amelyik egy fabeli csúcspontot köt össze egy izolált ponttal.

Legyen $\mathrm{A} \subset \mathrm{V(G)}$. A $(\mathrm{V} \setminus \mathrm{A}, \mathrm{A})$ halmazpárt a G gráf egy **vágás**ának nevezzük, amely a gráf azon $(u,v) \in \mathrm{E(G)}$ éleit tartalmazza, amelyekre $u \in \mathrm{V} \setminus \mathrm{A}$, $v \in \mathrm{A}$. Tekintsük a $\mathrm{V} \setminus \mathrm{A}$ halmazt a vágás bal oldalának, az A halmazt pedig a vágás jobb oldalának.

2) Prim algoritmusa

Stratégia: Gondolatban eltávolítjuk a gráf összes élét, majd a minimális súlyú feszítőfát egy r gyökerű faként építjük fel az élek fokozatos visszahelyezésével. A fa építése az r csúcspontból indul, s akkor fejeződik be, amikor minden csúcs belekerül a fába. Minden lépésben azt a legkisebb súlyú élt rakjuk vissza, amelyik egy fabeli csúcspontot köt össze egy izolált ponttal.

Legyen $\mathrm{A} \subset \mathrm{V(G)}$. A $(\mathrm{V} \setminus \mathrm{A}, \mathrm{A})$ halmazpárt a G gráf egy **vágás**ának nevezzük, amely a gráf azon $(u,v) \in \mathrm{E(G)}$ éleit tartalmazza, amelyekre $u \in \mathrm{V} \setminus \mathrm{A}$, $v \in \mathrm{A}$. Tekintsük a $\mathrm{V} \setminus \mathrm{A}$ halmazt a vágás bal oldalának, az A halmazt pedig a vágás jobb oldalának.

Egy Q sorban fogjuk tárolni a gráf azon csúcspontjait, amelyeket még nem kapcsoltunk a fához (tehát kezdetben az összes csúcspontot). Így a $\mathrm{V} \setminus \mathrm{Q}$ halmaz fogja tartalmazni a már fához kapcsolt csúcspontokat (kezdetben üres halmaz).

2) Prim algoritmusa

A $(V \setminus Q, Q)$ vágás bal oldalán van a növekvő fa, jobb oldalán pedig a csatolandó pontok.

2) Prim algoritmusa

A $(V \setminus Q, Q)$ vágás bal oldalán van a növekvő fa, jobb oldalán pedig a csatolandó pontok.

A `táv` tömbben fogjuk tárolni a Q halmazbeli csúcspontok fától való távolságát. A `táv[v]` tehát a v csúcspont fához kapcsolódó élei közül a legkisebb súlyú él súlyát fogja megadni. Ha nincs ilyen él, akkor `táv[v]=∞`. Kiindulási helyzetben minden csúcspont ∞ távolságra van a még nem létező fától, kivéve az r kiindulási csúcspontot, amelynek távolságát 0-ra állítjuk (ezzel biztosítjuk be, hogy elsőként az r csúcspont kerül be a fába).

2) Prim algoritmusa

A $(V \setminus Q, Q)$ vágás bal oldalán van a növekvő fa, jobb oldalán pedig a csatolandó pontok.

A `táv` tömbben fogjuk tárolni a Q halmazbeli csúcspontok fától való távolságát. A `táv[v]` tehát a v csúcspont fához kapcsolódó élei közül a legkisebb súlyú él súlyát fogja megadni. Ha nincs ilyen él, akkor `táv[v]=∞`. Kiindulási helyzetben minden csúcspont ∞ távolságra van a még nem létező fától, kivéve az r kiindulási csúcspontot, amelynek távolságát 0-ra állítjuk (ezzel biztosítjuk be, hogy elsőként az r csúcspont kerül be a fába).

Új csúcspontot helyezni a fába annyit jelent, hogy az adott csúcspont a Q halmazból átkerül a $V \setminus Q$ halmazba. Ezután a csúcspontot a Q halmazban maradt szomszédaihoz kapcsoló élek a vágás részeivé válnak, azok az élek pedig, melyek a fához kötötték, kikerülnek a vágásból. Mindez szükségessé teszi a `táv` tömb frissítését.

2) Prim algoritmusa

A KIVESZ_MIN(Q) eljárás meghatározza a `táv` tömb alapján azt a Q halmazbeli csúcspontot, amelyet az aktuális vágás legkisebb súlyú éle kapcsol a fához, és a fához kapcsolja. Az `apa` tömbben fogjuk nyilvántartani, hogy a fának melyik csúcspontjához csatoltuk az új csúcspontot.

2) Prim algoritmusa

A KIVESZ_MIN(Q) eljárás meghatározza a `táv` tömb alapján azt a Q halmazbeli csúcspontot, amelyet az aktuális vágás legkisebb súlyú éle kapcsol a fához, és a fához kapcsolja. Az `apa` tömbben fogjuk nyilvántartani, hogy a fának melyik csúcspontjához csatoltuk az új csúcspontot.

Amíg a v csúcspont a Q halmazban van, az `apa[v]` a v csúcs azon fabeli szomszédját tárolja, amelyikhez az aktuális vágás legkisebb súlyú éle köti. Az `apa[v]` értéket frissíteni kell, ha a v csúcspont egy közelebbi szomszédja kerül a fába. Amikor a v csúcspont bekerül a fába, az `apa[v]` a v fabeli apját fogja tárolni.

2) Prim algoritmusa

A KIVESZ_MIN(Q) eljárás meghatározza a `táv` tömb alapján azt a Q halmazbeli csúcspontot, amelyet az aktuális vágás legkisebb súlyú éle kapcsol a fához, és a fához kapcsolja. Az `apa` tömbben fogjuk nyilvántartani, hogy a fának melyik csúcspontjához csatoltuk az új csúcspontot.

Amíg a v csúcspont a Q halmazban van, az `apa[v]` a v csúcs azon fabeli szomszédját tárolja, amelyikhez az aktuális vágás legkisebb súlyú éle köti. Az `apa[v]` értéket frissíteni kell, ha a v csúcspont egy közelebbi szomszédja kerül a fába. Amikor a v csúcspont bekerül a fába, az `apa[v]` a v fabeli apját fogja tárolni.

Az algoritmus végén az `apa` tömb minden csúcspontnak a minimális súlyú feszítőfában lévő apját fogja tárolni. Az algoritmus futása során a vágás végigvonul a gráfon, minden pillanatban határt képezve a $V \setminus Q$ és Q halmazok között.

```
függvény PRIM(G,r)
   Q ← V(G)
   minden v ∈ Q végezd
      táv[v] ← ∞
   vége minden
   táv[r] ← 0
   apa[r] ← 0
   amíg Q ≠ ∅ végezd
      u ← KIVESZ_MIN(Q)
      minden v ∈ szomszéd(u) végezd
         ha (v ∈ Q ÉS (súly(u,v) < táv[v])) akkor
            apa[v] ← u
            táv[v] ← súly(u,v)
         vége ha
      vége minden
   vége amíg
   vissza apa
vége PRIM
```

```
függvény PRIM(G,r)
   Q ← V(G)
   minden v ∈ Q végezd
      táv[v] ← ∞
   vége minden
   táv[r] ← 0
   apa[r] ← 0
   amíg Q ≠ ∅ végezd
      u ← KIVESZ_MIN(Q)
      minden v ∈ szomszéd(u) végezd
         ha (v ∈ Q ÉS (súly(u,v) < táv[v])) akkor
            apa[v] ← u
            táv[v] ← súly(u,v)
         vége ha
      vége minden
   vége amíg
   vissza apa
vége PRIM
```

A PRIM algoritmus bonyolultsága $O(m \log m)$.

A továbbiakban bemutatunk egy egyszerűbb implementációt. A `fa[1..n]` tömb fogja nyilvántartani, hogy mely csúcspontok tartoznak a növekvő fához, illetve melyik várnak még csatolásra. Ha az i-edik csúcspont már része a fának, akkor `fa[i]=1`, különben `fa[i]=0`.

A továbbiakban bemutatunk egy egyszerűbb implementációt. A `fa[1..n]` tömb fogja nyilvántartani, hogy mely csúcspontok tartoznak a növekvő fához, illetve melyik várnak még csatolásra. Ha az i-edik csúcspont már része a fának, akkor `fa[i]=1`, különben `fa[i]=0`.

A gráf éleit az `élek[1..m]` bejegyzés-tömb tárolja mint éllistát. Minden élről eltároljuk a kezdőpontját (u mező), végpontját (v mező) és súlyát (*súly* mező), illetve azt, hogy része-e az aktuális vágásnak vagy nem (*vágás* mező). Az `élek` tömböt az élek súlya szerint növekvő sorrendbe rendezzük.

A továbbiakban bemutatunk egy egyszerűbb implementációt. A `fa[1..n]` tömb fogja nyilvántartani, hogy mely csúcspontok tartoznak a növekvő fához, illetve melyik várnak még csatolásra. Ha az i-edik csúcspont már része a fának, akkor `fa[i]=1`, különben `fa[i]=0`.

A gráf éleit az `élek[1..m]` bejegyzés-tömb tárolja mint éllistát. Minden élről eltároljuk a kezdőpontját (u mező), végpontját (v mező) és súlyát (*súly* mező), illetve azt, hogy része-e az aktuális vágásnak vagy nem (*vágás* mező). Az `élek` tömböt az élek súlya szerint növekvő sorrendbe rendezzük.

A gráfot szomszédsági listaként (`SZ_L[1..n]` bejegyzés-tömb) ábrázoljuk, ahol a `SZ_L[i].fokszám` mező az i-edik pont fokszámát tárolja, az `SZ_L[i].szomszéd[]` tömb-mező pedig az i-edik pont szomszédainak listáját.

A továbbiakban bemutatunk egy egyszerűbb implementációt. A `fa[1..n]` tömb fogja nyilvántartani, hogy mely csúcspontok tartoznak a növekvő fához, illetve melyik várnak még csatolásra. Ha az i-edik csúcspont már része a fának, akkor `fa[i]=1`, különben `fa[i]=0`.

A gráf éleit az `élek[1..m]` bejegyzés-tömb tárolja mint éllistát. Minden élről eltároljuk a kezdőpontját (u mező), végpontját (v mező) és súlyát (*súly* mező), illetve azt, hogy része-e az aktuális vágásnak vagy nem (*vágás* mező). Az `élek` tömböt az élek súlya szerint növekvő sorrendbe rendezzük.

A gráfot szomszédsági listaként (`SZ_L[1..n]` bejegyzés-tömb) ábrázoljuk, ahol a `SZ_L[i].fokszám` mező az i-edik pont fokszámát tárolja, az `SZ_L[i].szomszéd[]` tömb-mező pedig az i-edik pont szomszédainak listáját.

Felépítünk egy szomszédsági mátrixot is (`SZ_M[1..n,1..n]`), ahol a tömbelemek az egyes élek éllistabeli sorszámát tárolják.

Kiindulási helyzetben a fa az r csúcspontból áll, az aktuális vágás pedig az r csúcspontra illeszkedő éleket tartalmazza. Minden lépésben (összesen $n-1$ darab van) három műveletet hajt végre az algoritmus:

Kiindulási helyzetben a fa az r csúcspontból áll, az aktuális vágás pedig az r csúcspontra illeszkedő éleket tartalmazza. Minden lépésben (összesen $n - 1$ darab van) három műveletet hajt végre az algoritmus:

1. **Kiválasztja az aktuális vágás legkisebb súlyú élét.** Mivel az `élek` tömb rendezett, a minimális súlyú vágásbeli él az `élek` tömb vágáshoz tartozó elemei közül az első lesz (KIVESZ_VÁGÁSBÓL_MIN eljárás).

Kiindulási helyzetben a fa az r csúcspontból áll, az aktuális vágás pedig az r csúcspontra illeszkedő éleket tartalmazza. Minden lépésben (összesen $n-1$ darab van) három műveletet hajt végre az algoritmus:

1. **Kiválasztja az aktuális vágás legkisebb súlyú élét.** Mivel az `élek` tömb rendezett, a minimális súlyú vágásbeli él az `élek` tömb vágáshoz tartozó elemei közül az első lesz (KIVESZ_VÁGÁSBÓL_MIN eljárás).
2. **A fához csatolja a kiválasztott élt.** Az illető élen keresztül a fához csatoljuk a legközelebb eső csúcspontot. A `fa` tömb megfelelő elemét 1-re állítjuk.

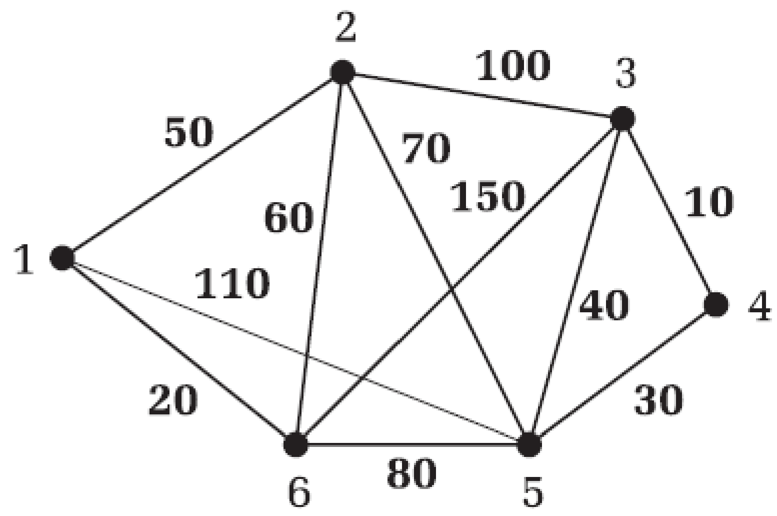
Kiindulási helyzetben a fa az r csúcspontból áll, az aktuális vágás pedig az r csúcspontra illeszkedő éleket tartalmazza. Minden lépésben (összesen $n - 1$ darab van) három műveletet hajt végre az algoritmus:

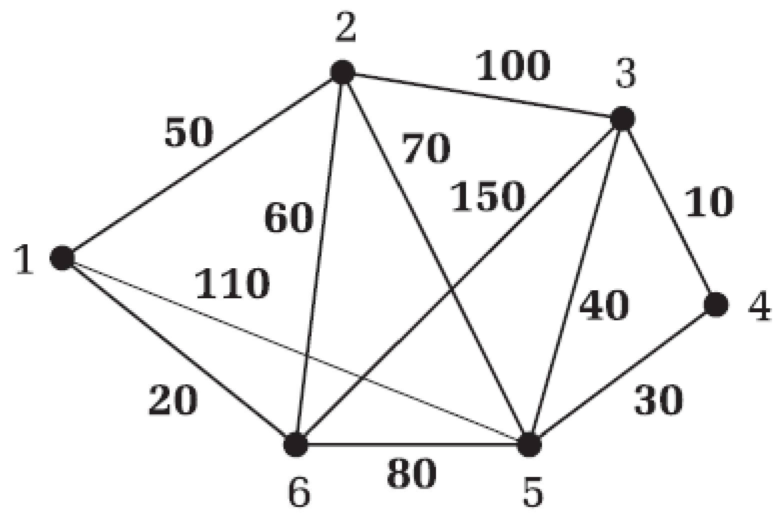
1. **Kiválasztja az aktuális vágás legkisebb súlyú élét.** Mivel az `élek` tömb rendezett, a minimális súlyú vágásbeli él az `élek` tömb vágáshoz tartozó elemei közül az első lesz (KIVESZ_VÁGÁSBÓL_MIN eljárás).
2. **A fához csatolja a kiválasztott élt.** Az illető élen keresztül a fához csatoljuk a legközelebb eső csúcspontot. A `fa` tömb megfelelő elemét 1-re állítjuk.
3. **Aktualizálja a vágást.** Kikerülnek a vágásból azok az élek, amelyek az új csúcspontot a fában lévő szomszédaihoz kötik, és bekerülnek azok az élek, amelyek a vágás jobb oldalán maradt szomszédaihoz vezetnek.

```
eljárás PRIM(G,r)
                                  // a fa tömb inicializálása
   minden v∈Q végezd
      fa[v] ← 0
   vége minden
   fa[r] ← 1
   apa[r] ← 0
   RENDEZÉS_SÚLY_SZERINT(élek,m)
        // az SZ_M mátrix felépítése; a vágás inicializálása
   minden i = 1,m végezd
      SZ_M[élek[i].u][élek[i].v] ← i
      SZ_M[élek[i].v][élek[i].u] ← i
      ha (élek[i].u = r) VAGY (élek[i].v = r) akkor
         élek[i].vágás ← 1
      különben
         élek[i].vágás ← 0
      vége ha
   vége minden
```

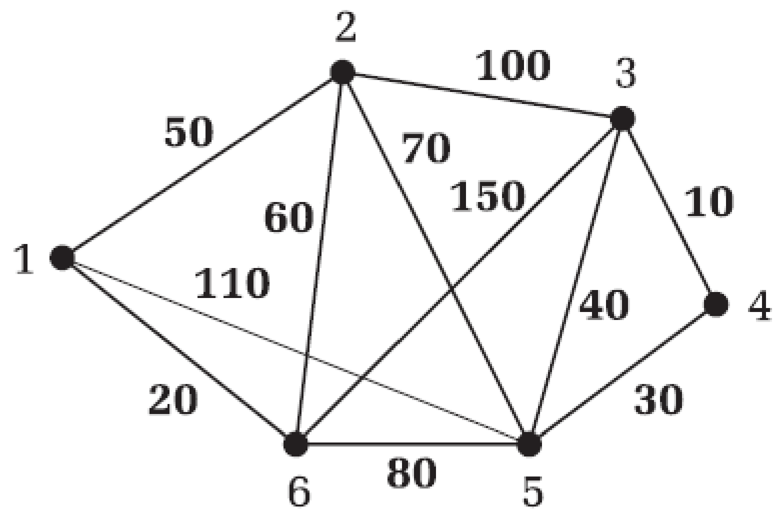
```
                    // Az algoritmus magva (n-1) lépésben
összsúly = 0
minden lépés = 1,n-1 végezd
                                              // 1. művelet
   i ← KIVESZ_VÁGÁSBÓL_MIN(élek,m)
   kiír: élek[i].u, élek[i].v, élek[i].súly
   össze_súly = össz_súly + élek[i].súly
                                              // 2. művelet
   ha fa[élek[i].u] = 1 akkor
      csatolt_pont ← élek[i].v
      apa[csatolt_pont] ← élek[i].u
   különben
      csatolt_pont ← élek[i].u
      apa[csatolt_pont] ← élek[i].v
   vége ha
   fa[csatolt_pont] ← 1
```

```
                                              // 3. művelet
      minden j ← 1,Sz_L[csatolt_pont].fokszám végezd
         sz_pont ← Sz_L[csatolt_pont].szomszéd[j]
         sz_él ← SZ_M[csatolt_pont][sz_pont]
         ha (fa[sz_pont] = 1) akkor
            élek[sz_él].vágás ← 0
         különben
            élek[sz_él].vágás ← 1
         vége ha
      vége minden
   vége minden
   kiír: össz_súly
vége PRIM
```

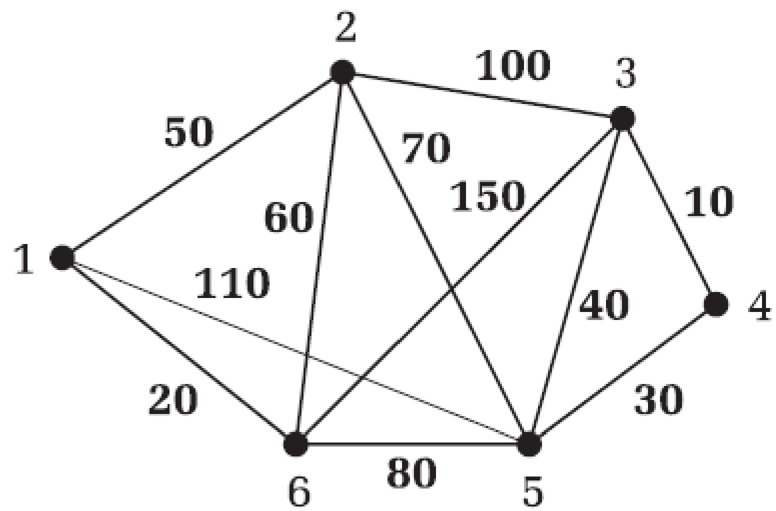

2
100
3
50
70
150
10
60
1
110
40
4
20
30
6
80
5



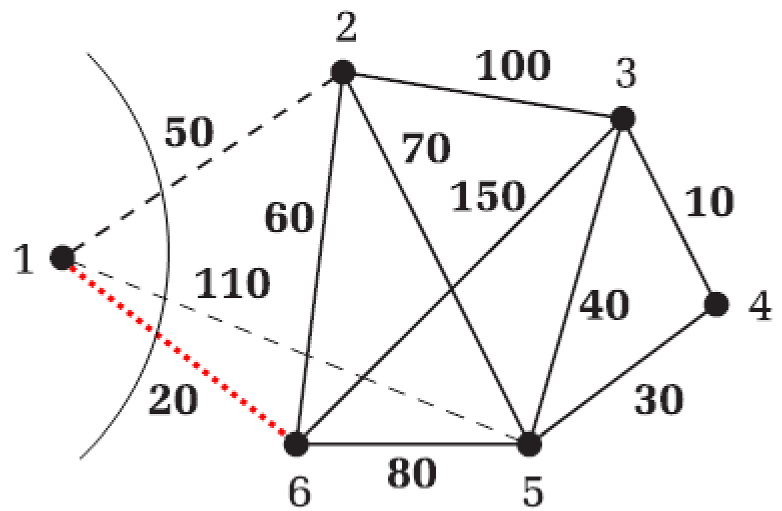
táv	0	∞	∞	∞	∞	∞
apa	0	0	0	0	0	0
	1	2	3	4	5	6



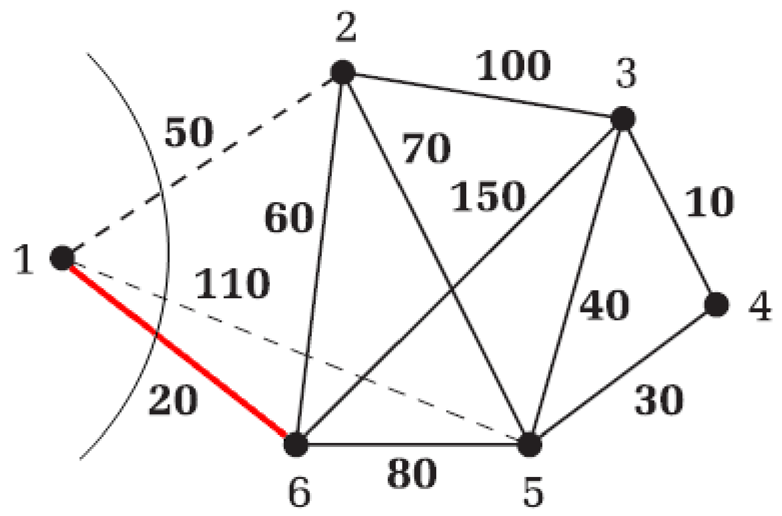
táv	0	∞	∞	∞	∞	∞
apa	0	0	0	0	0	0
	1	2	3	4	5	6



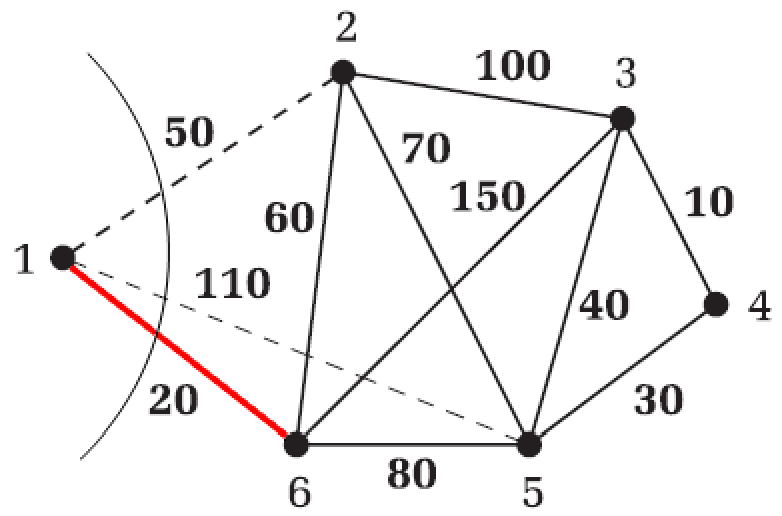
táv	0	50	∞	∞	110	20
apa	0	1	0	0	1	1
	1	2	3	4	5	6



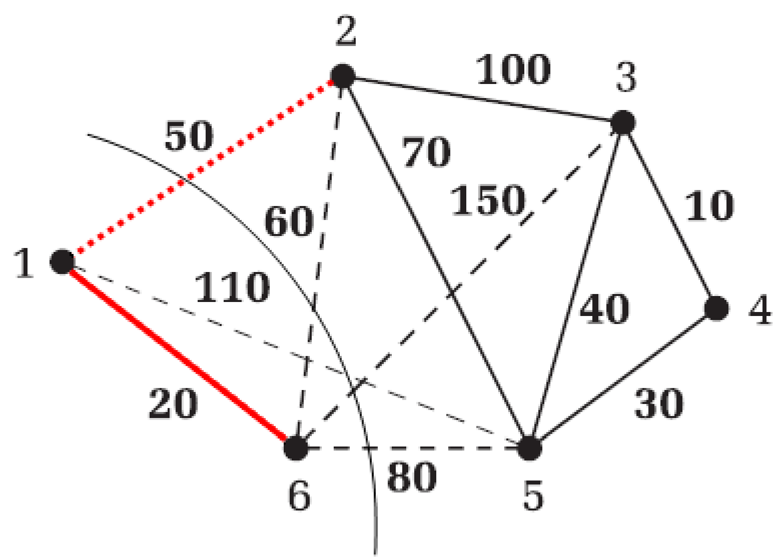
táv	0	50	∞	∞	110	20
apa	0	1	0	0	1	1
	1	2	3	4	5	6



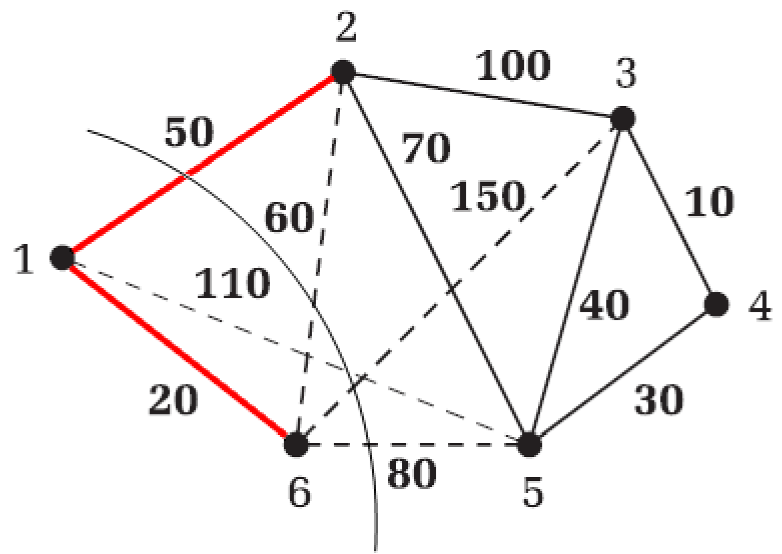
táv	0	50	∞	∞	110	0
apa	0	1	0	0	1	1
	1	2	3	4	5	6



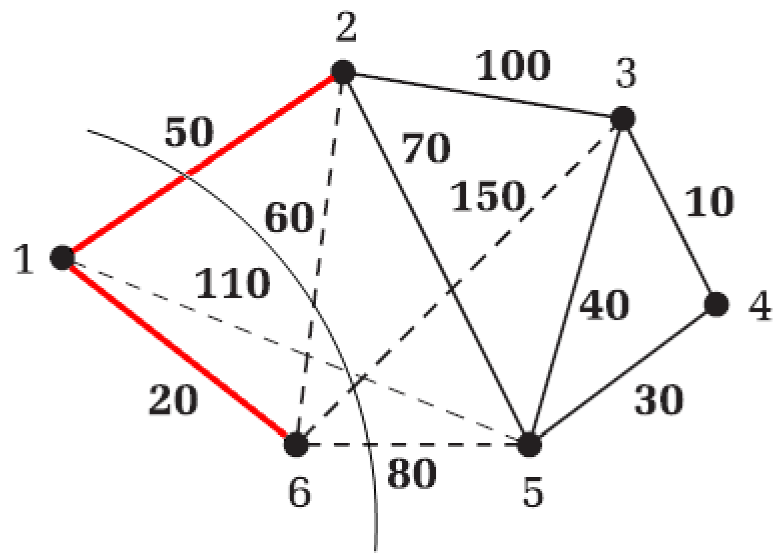
táv	0	50	150	∞	80	0
apa	0	1	6	0	6	1
	1	2	3	4	5	6



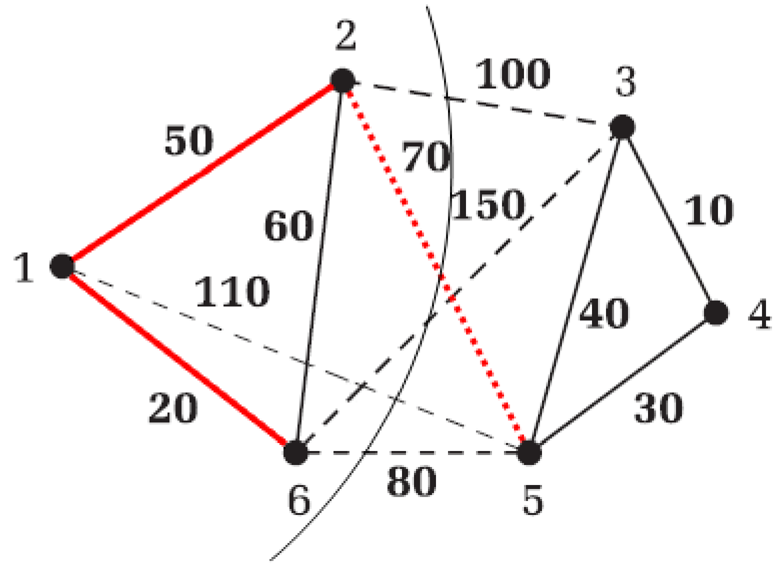
	1	2	3	4	5	6
táv	0	50	150	∞	80	0
apa	0	1	6	0	6	1



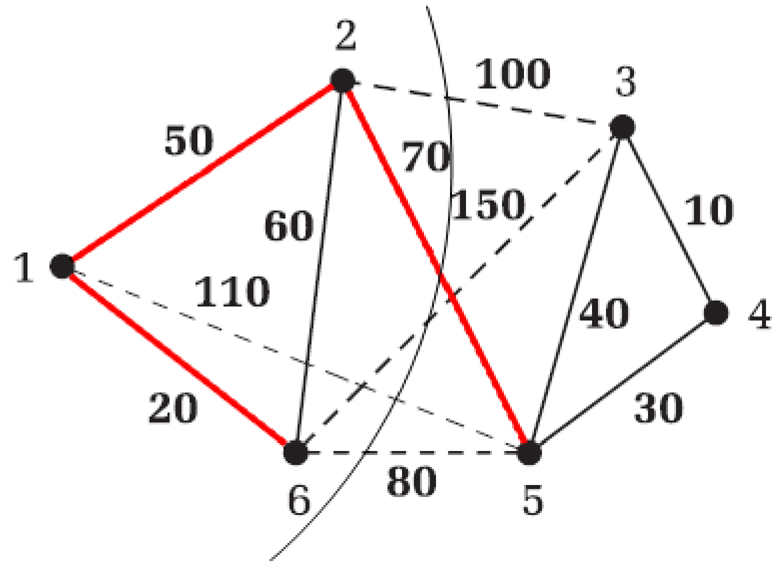
	1	2	3	4	5	6
táv	0	0	150	∞	80	0
apa	0	1	6	0	6	1



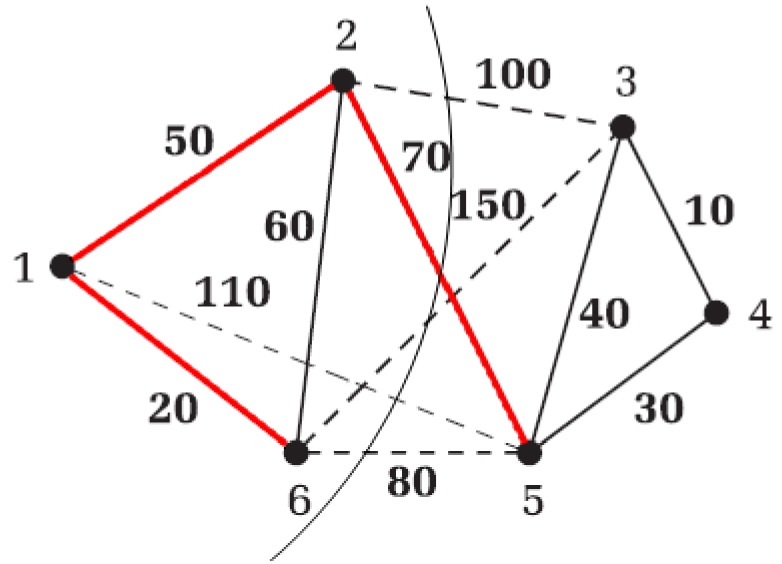
táv	0	0	100	∞	70	0
apa	0	1	2	0	2	1
	1	2	3	4	5	6



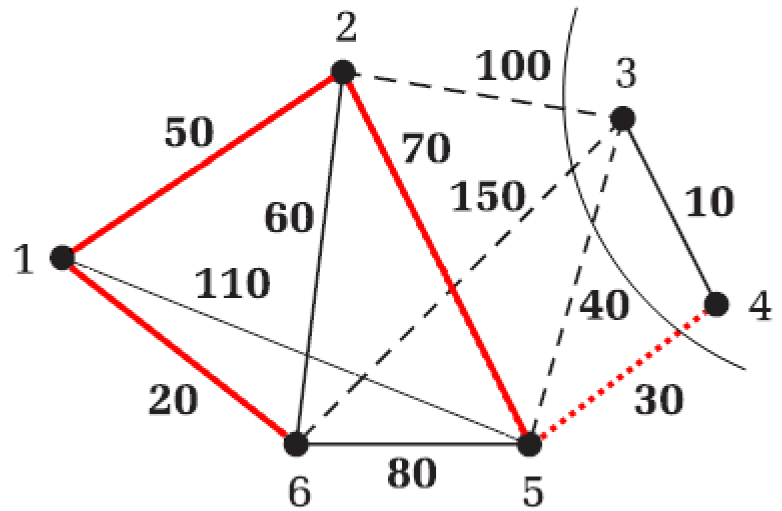
táv	0	0	100	∞	70	0
apa	0	1	2	0	2	1
	1	2	3	4	5	6



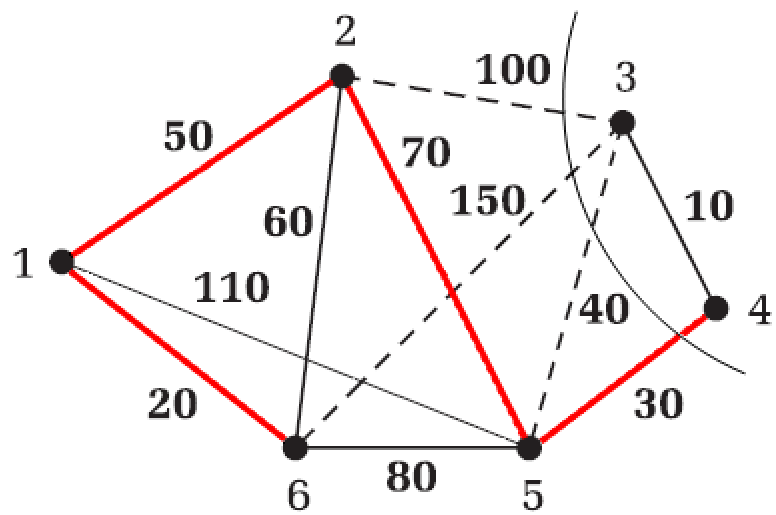
táv	0	0	100	∞	0	0
apa	0	1	2	0	2	1
	1	2	3	4	5	6



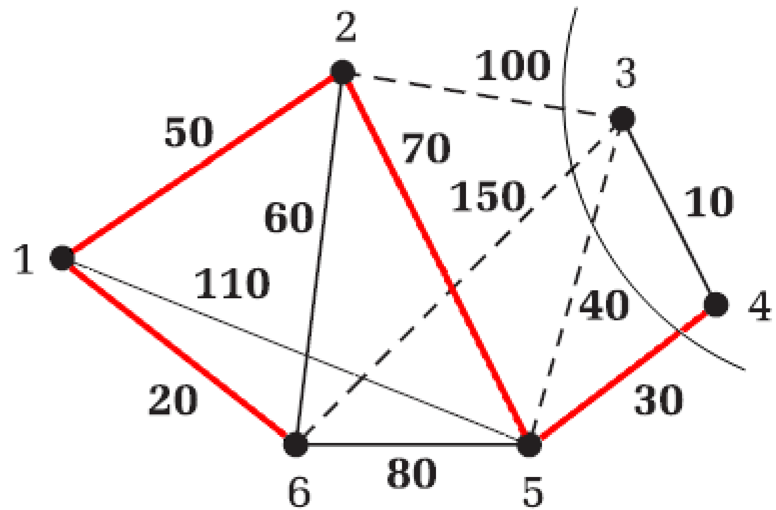
táv	0	0	40	30	0	0
apa	0	1	5	5	2	1
	1	2	3	4	5	6



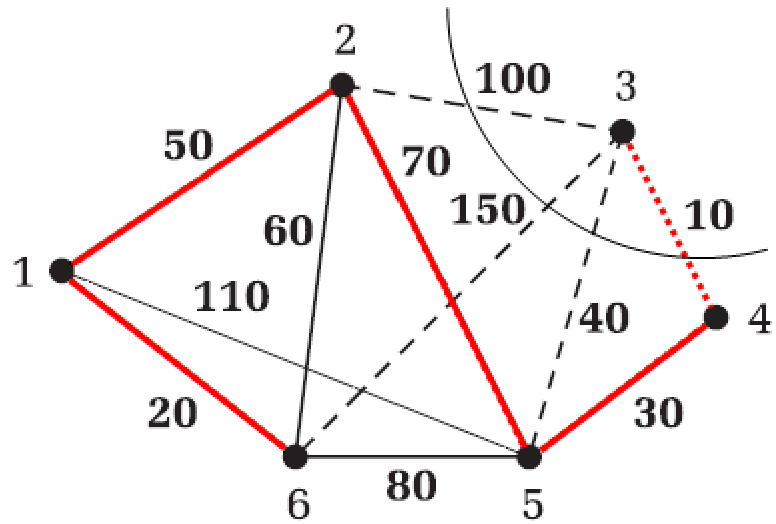
táv	0	0	40	30	0	0
apa	0	1	5	5	2	1
	1	2	3	4	5	6



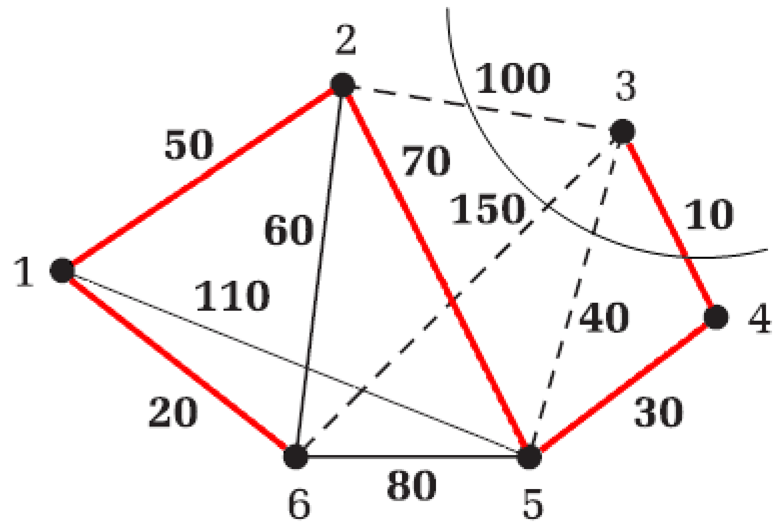
táv	0	0	40	0	0	0
apa	0	1	5	5	2	1
	1	2	3	4	5	6



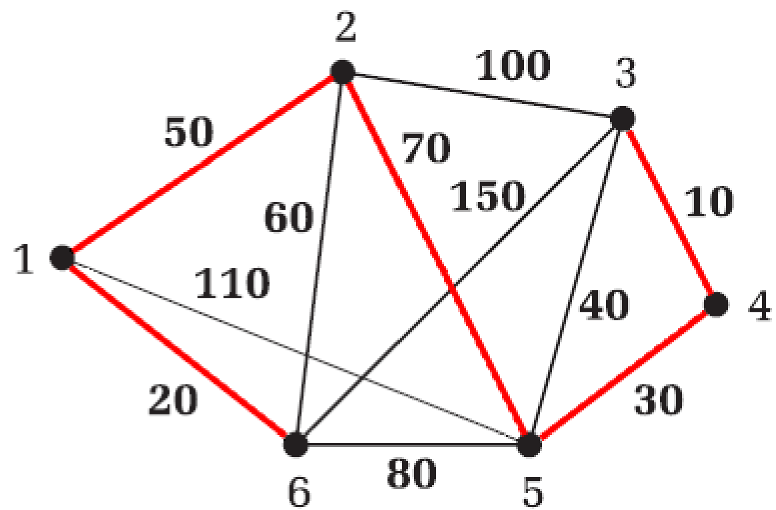
	1	2	3	4	5	6
táv	0	0	10	0	0	0
apa	0	1	4	5	2	1



táv	0	0	10	0	0	0
apa	0	1	4	5	2	1
	1	2	3	4	5	6



	1	2	3	4	5	6
táv	0	0	0	0	0	0
apa	0	1	4	5	2	1



táv	0	0	0	0	0	0
apa	0	1	4	5	2	1
	1	2	3	4	5	6

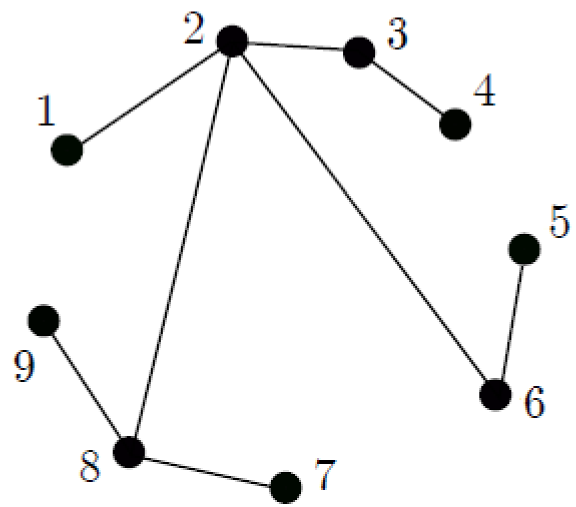
Prüfer-kód

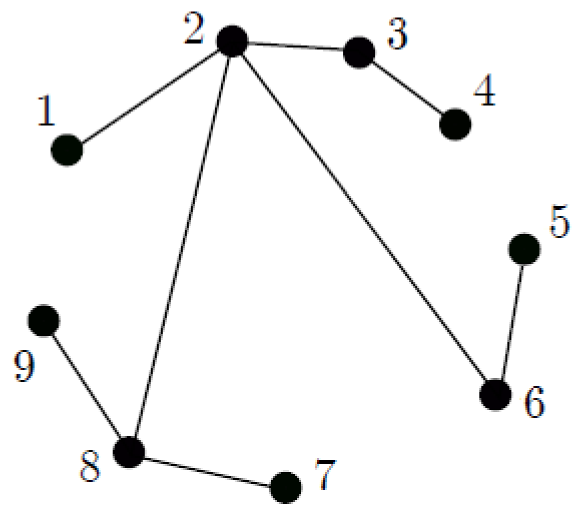
Minden n csúcspontú fa, amelynek a csúcspontjait tetszőleges módon az $1, 2, \ldots, n$ természetes számokkal címkéztük meg, kódolható egy $(v_1, v_2, \ldots, v_{n-1})$ sorozattal, ahol $v_i \in \{1, 2, \ldots, n\}$.

Prüfer-kód

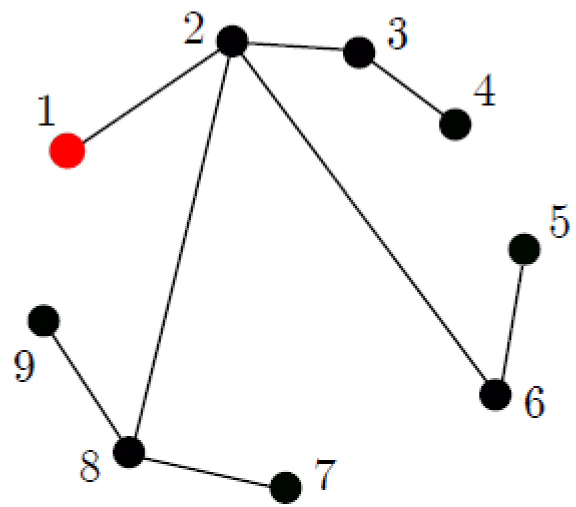
Minden n csúcspontú fa, amelynek a csúcspontjait tetszőleges módon az $1, 2, \ldots, n$ természetes számokkal címkéztük meg, kódolható egy $(v_1, v_2, \ldots, v_{n-1})$ sorozattal, ahol $v_i \in \{1, 2, \ldots, n\}$.

Kódolási folyamat ($n - 1$ lépéses eljárás)**:** Minden lépésben eltávolítjuk a fa legkisebb címkéjű levelét, és beírjuk a sorozatba annak a csúcspontnak a címkéjét, amelyről levágtuk. Az eredményül kapott sorozat a fa **Prüfer-kód**ja. A kód utolsó eleme a gyökér címkéje.

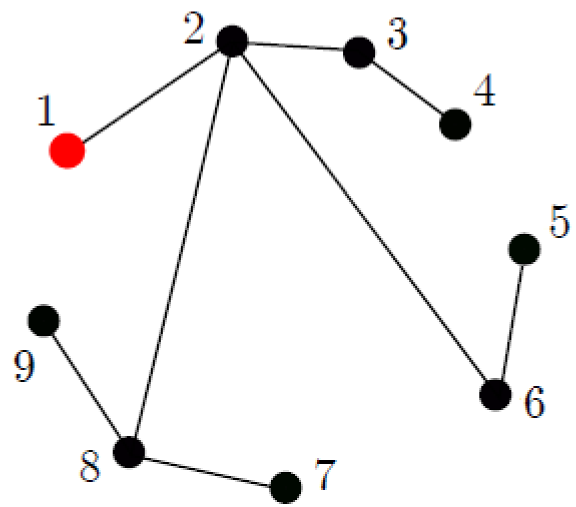

2
3
4
1
5
9
6
8
7



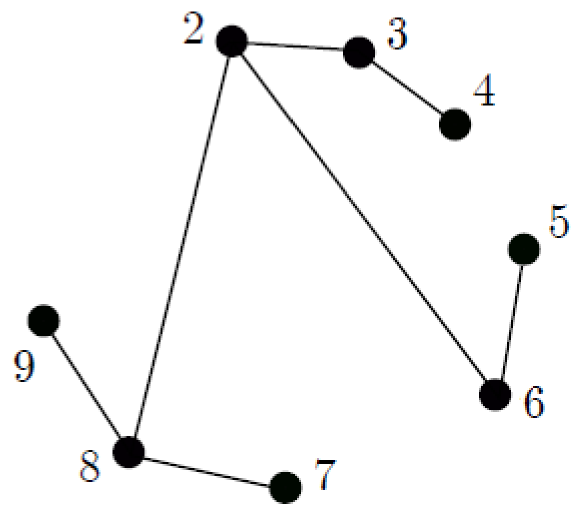
i	1	2	3	4	5	6	7	8
v_i								



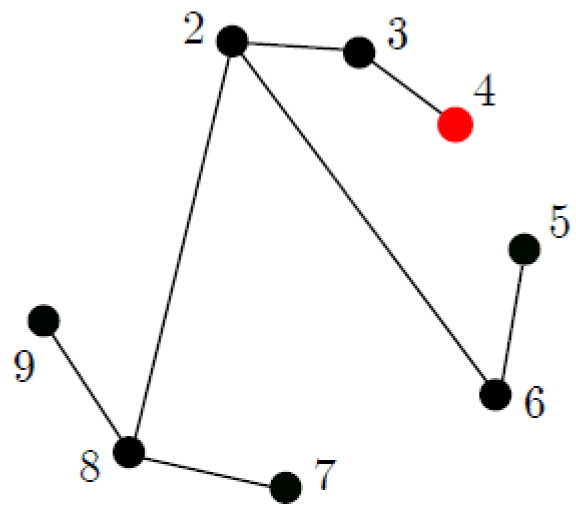
i	1	2	3	4	5	6	7	8
v_i								



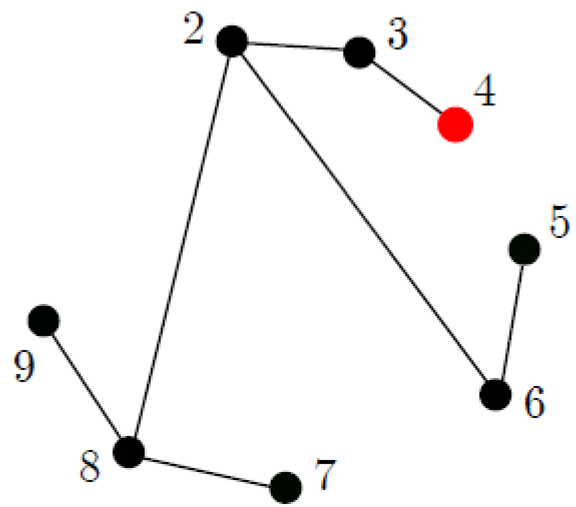
i	1	2	3	4	5	6	7	8
v_i	2							



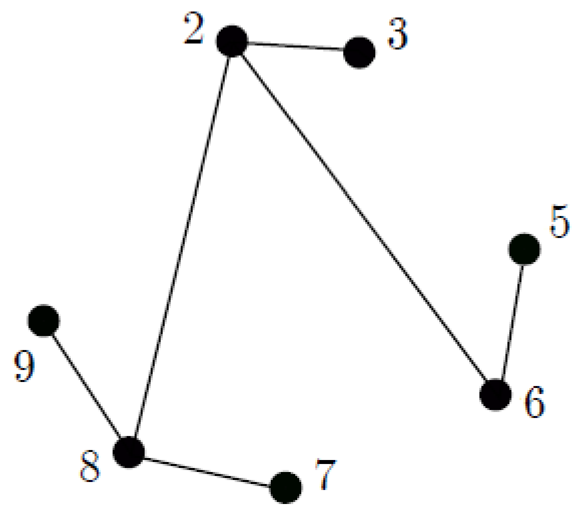
i	1	2	3	4	5	6	7	8
v_i	2							



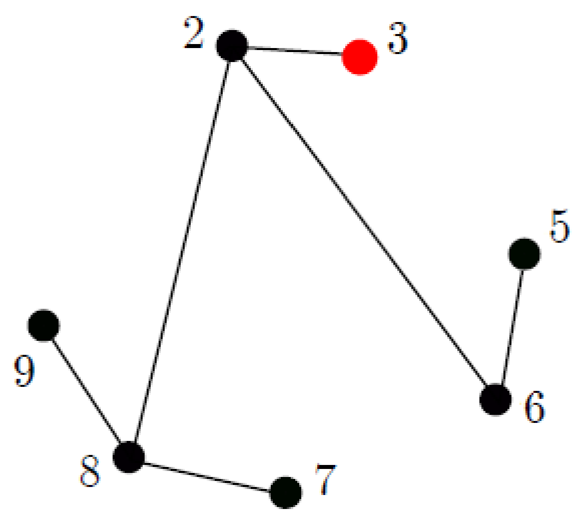
i	1	2	3	4	5	6	7	8
v_i	2							



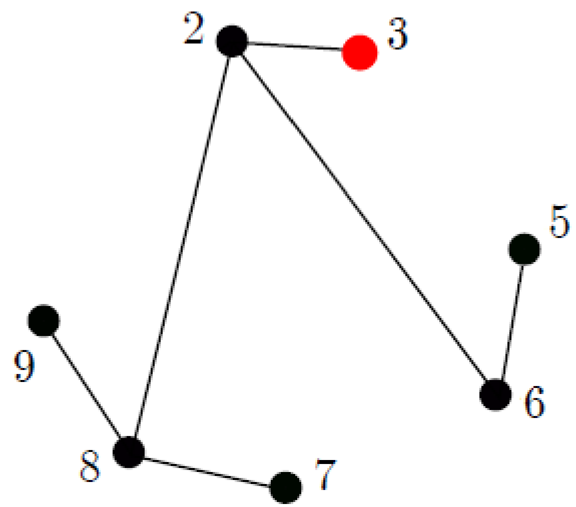
i	1	2	3	4	5	6	7	8
v_i	2	3						



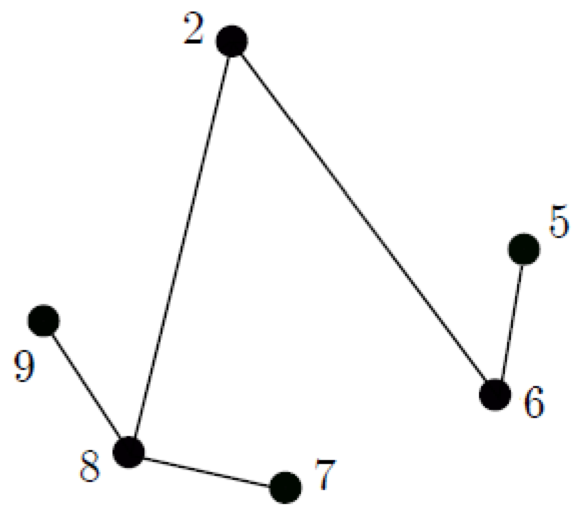
i	1	2	3	4	5	6	7	8
v_i	2	3						



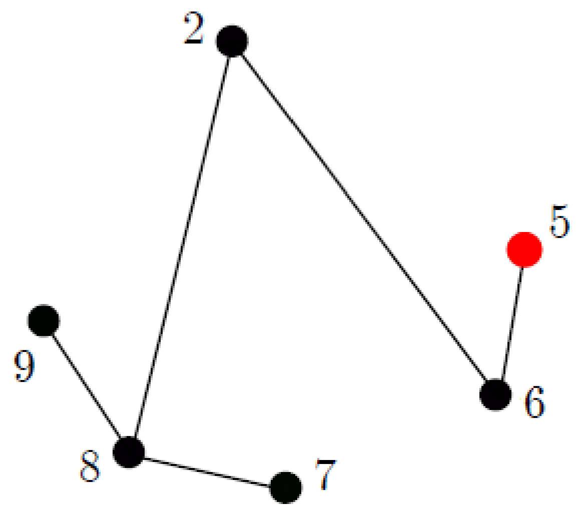
i	1	2	3	4	5	6	7	8
v_i	2	3						



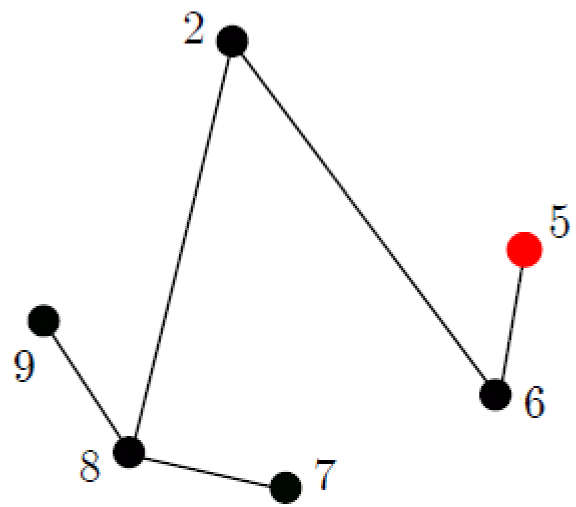
i	1	2	3	4	5	6	7	8
v_i	2	3	2					



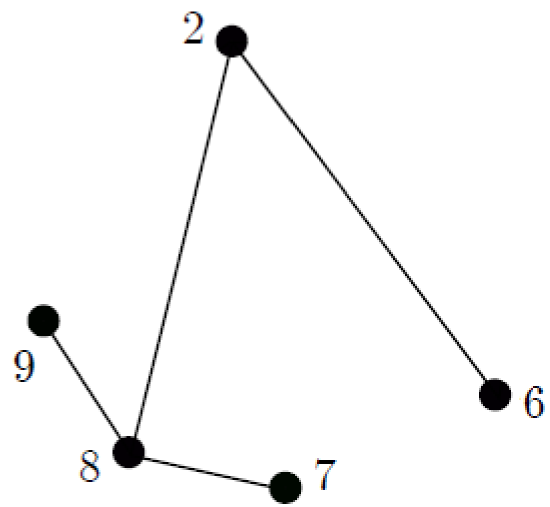
i	1	2	3	4	5	6	7	8
v_i	2	3	2					



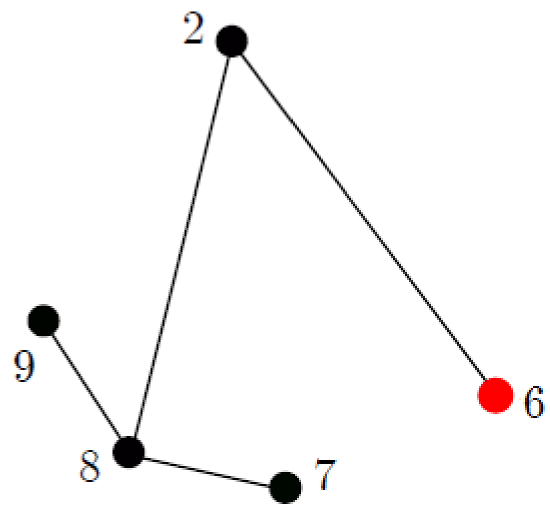
i	1	2	3	4	5	6	7	8
v_i	2	3	2					



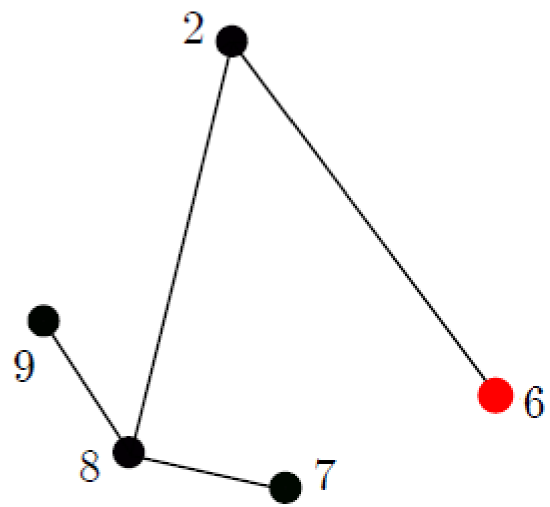
i	1	2	3	4	5	6	7	8
v_i	2	3	2	6				



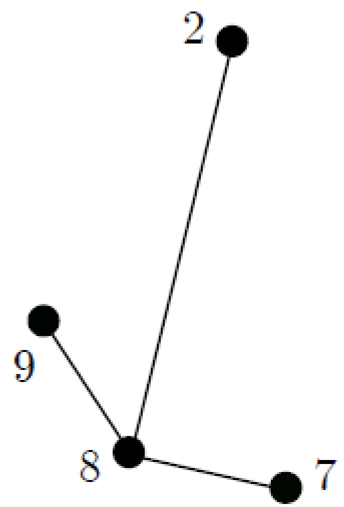
i	1	2	3	4	5	6	7	8
v_i	2	3	2	6				



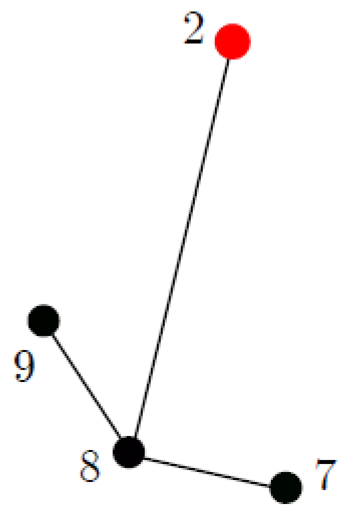
i	1	2	3	4	5	6	7	8
v_i	2	3	2	6				



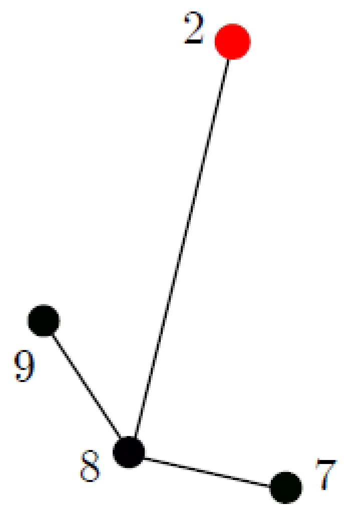
i	1	2	3	4	5	6	7	8
v_i	2	3	2	6	2			



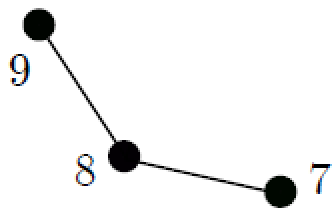
i	1	2	3	4	5	6	7	8
v_i	2	3	2	6	2			



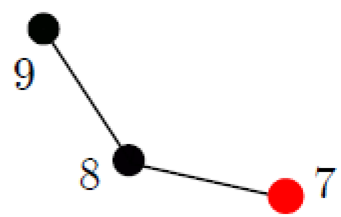
i	1	2	3	4	5	6	7	8
v_i	2	3	2	6	2			



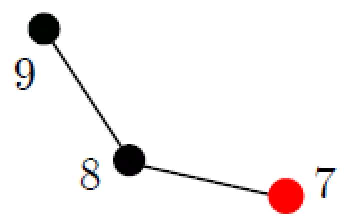
i	1	2	3	4	5	6	7	8
v_i	2	3	2	6	2	8		



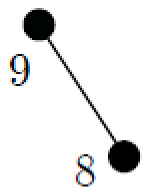
i	1	2	3	4	5	6	7	8
v_i	2	3	2	6	2	8		



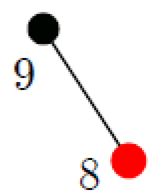
i	1	2	3	4	5	6	7	8
v_i	2	3	2	6	2	8		



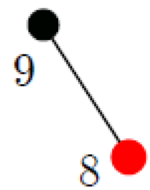
i	1	2	3	4	5	6	7	8
v_i	2	3	2	6	2	8	8	



i	1	2	3	4	5	6	7	8
v_i	2	3	2	6	2	8	8	

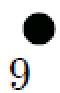


i	1	2	3	4	5	6	7	8
v_i	2	3	2	6	2	8	8	



i	1	2	3	4	5	6	7	8
v_i	2	3	2	6	2	8	8	9

i	1	2	3	4	5	6	7	8
v_i	2	3	2	6	2	8	8	9



i	1	2	3	4	5	6	7	8
v_i	2	3	2	6	2	8	8	9

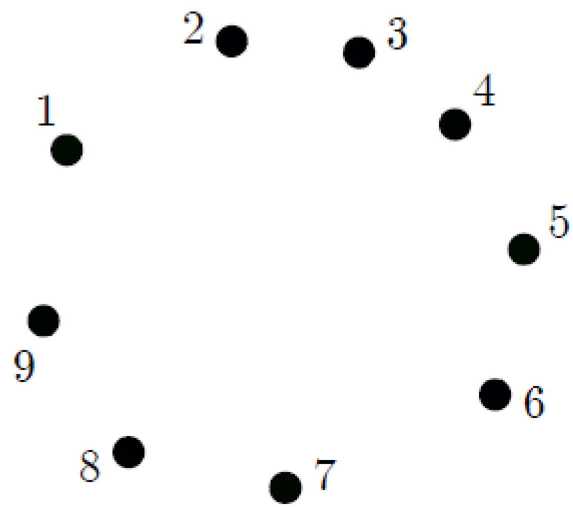
PrüferKódolás(F)

1. legyen K üres sorozat
2. **while** F nemcsak gyökérből áll **do**
3.   legyen v a legkisebb címkéjű levél F-ben
4.   írjuk be K-ba v ősét
5.   töröljük v-t F-ből
6. **return** K

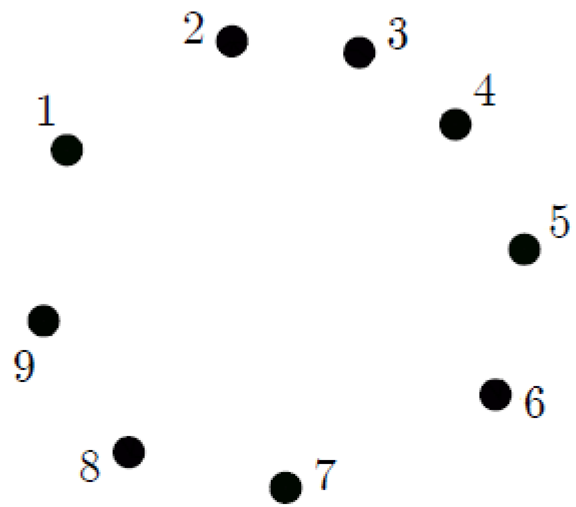
Prüfer-kód

Dekódolási folyamat ($n-1$ lépéses eljárás)**:** A Prüfer-kód egy véges számsorozat. Legyen a az első szám a sorozatban. Keressük meg azt a legkisebb b számot, amely nincs benne a sorozatban. Rajzoljunk élt a fában az a csúcsból a b csúcsba. Töröljük ki a sorozat első elemét (a-t), majd írjuk be a végére a b-t. Folytassuk az eljárást mindaddig, amíg el nem fogynak az eredeti sorozat elemei.

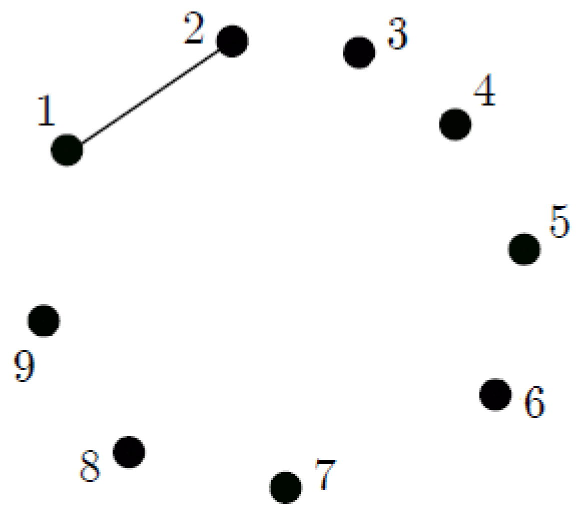
2	3	2	6	2	8	8	9	



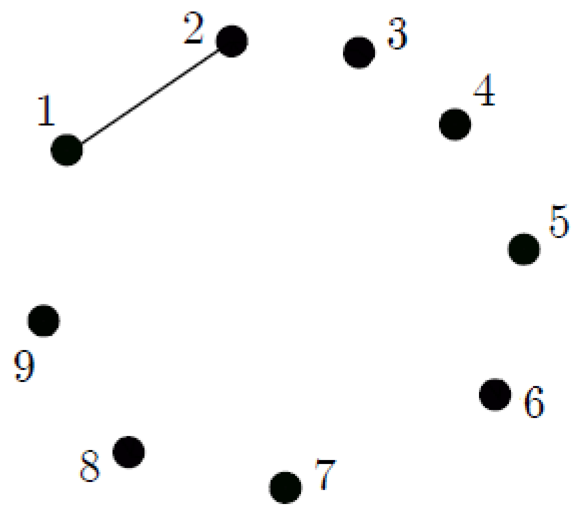
2	3	2	6	2	8	8	9	



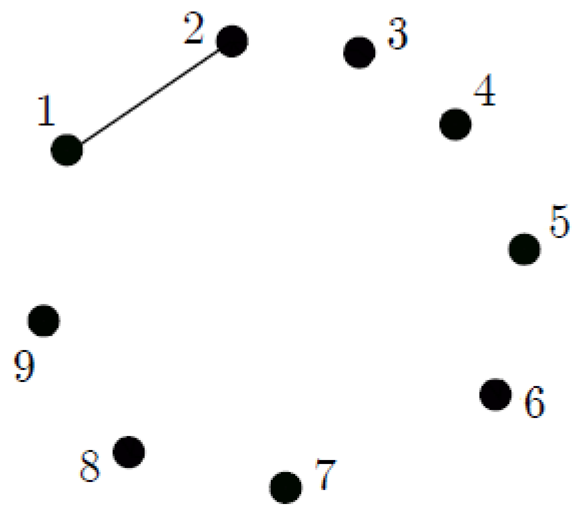
2	3	2	6	2	8	8	9	



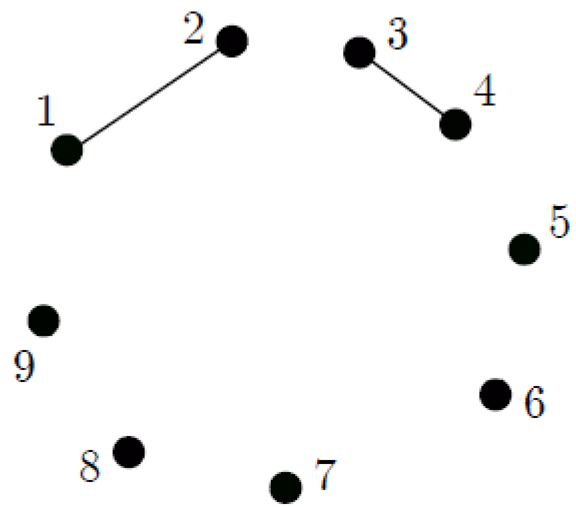
2	3	2	6	2	8	8	9	



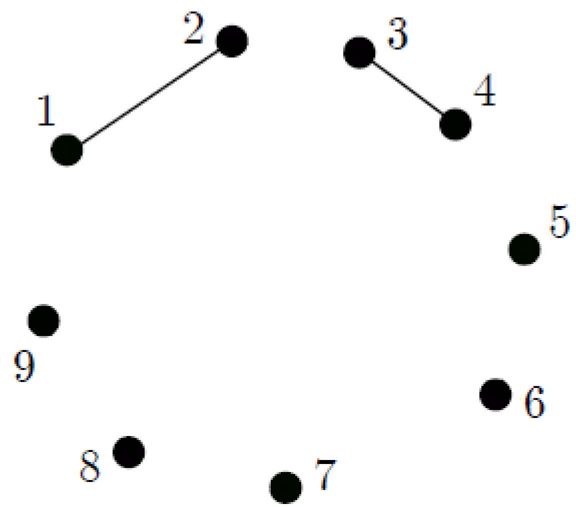
	3	2	6	2	8	8	9	1



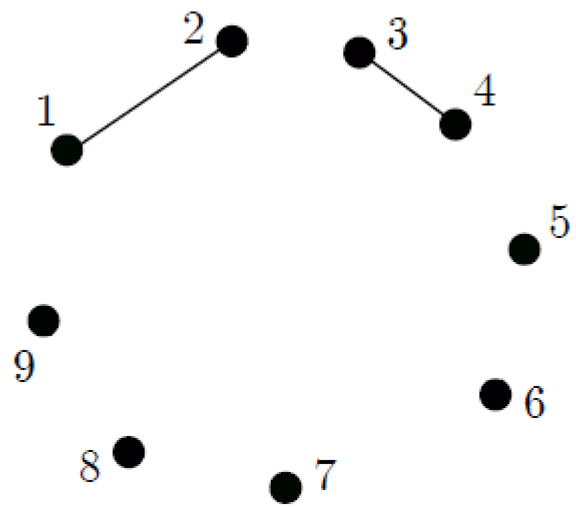
	3	2	6	2	8	8	9	1



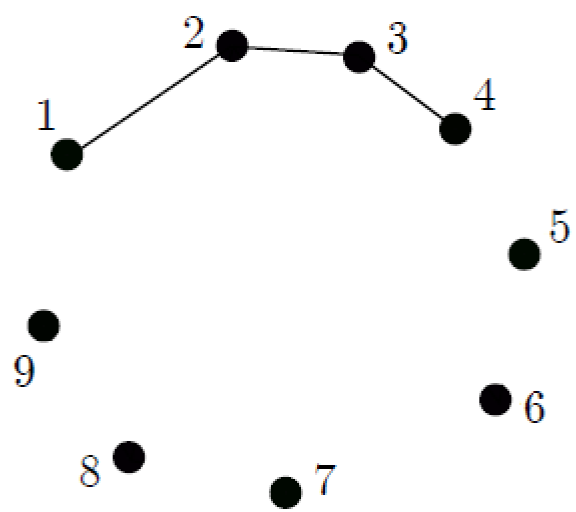
	3	2	6	2	8	8	9	1



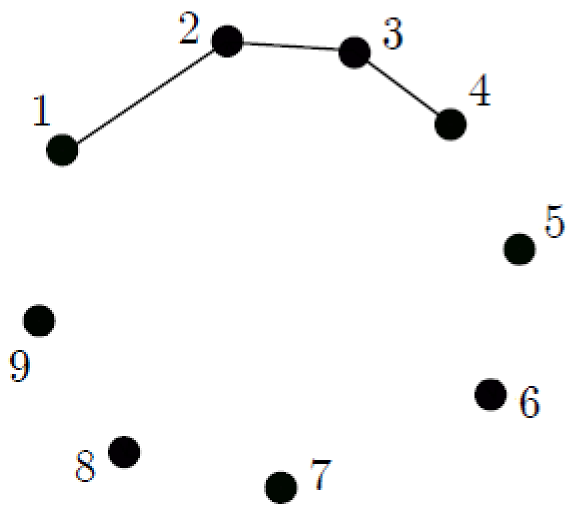
	2	6	2	8	8	9	1	4



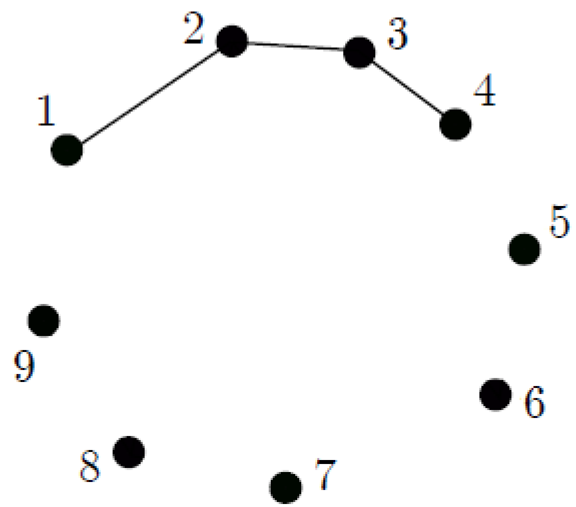
	$\boxed{2}$	6	2	8	8	9	1	4



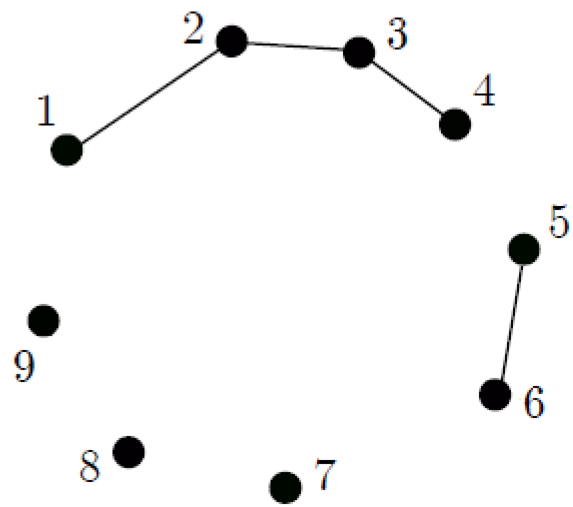
	$\boxed{2}$	6	2	8	8	9	1	4



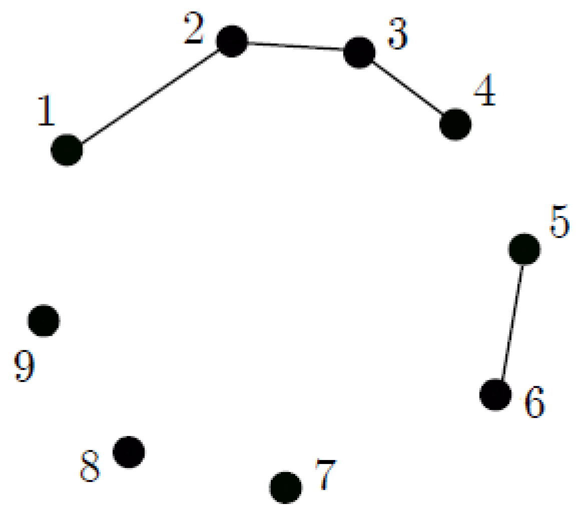
	6	2	8	8	9	1	4	3



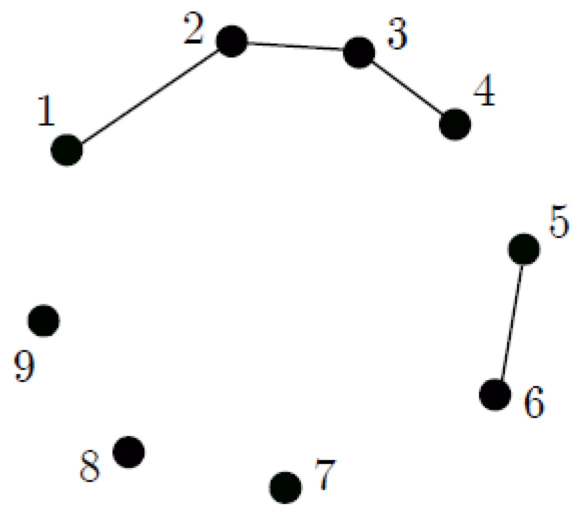
	6	2	8	8	9	1	4	3



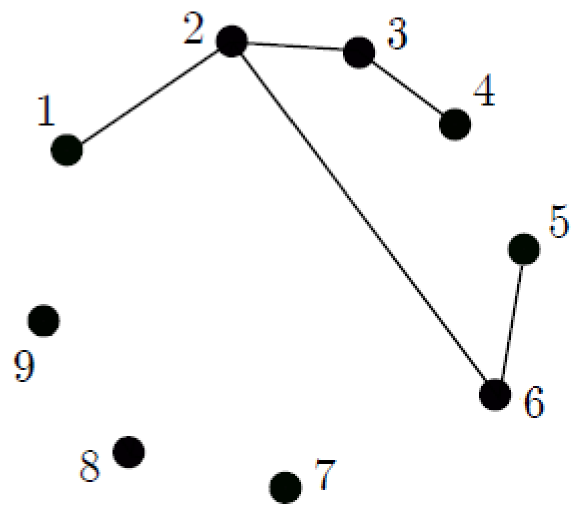
	6	2	8	8	9	1	4	3



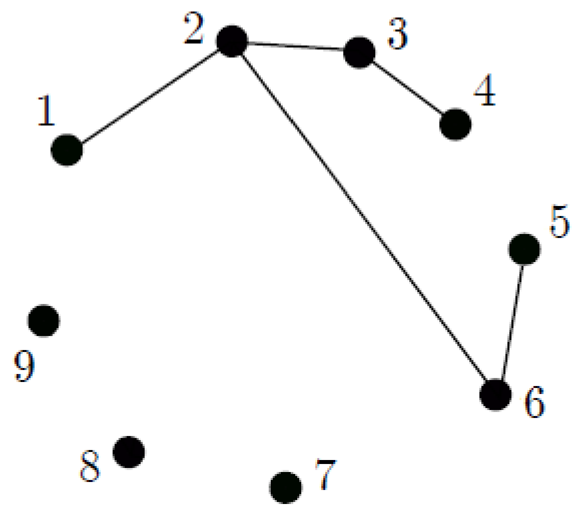
	2	8	8	9	1	4	3	5



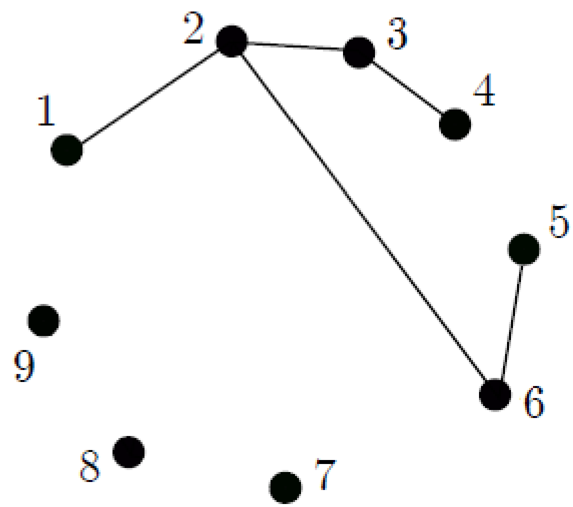
	2	8	8	9	1	4	3	5



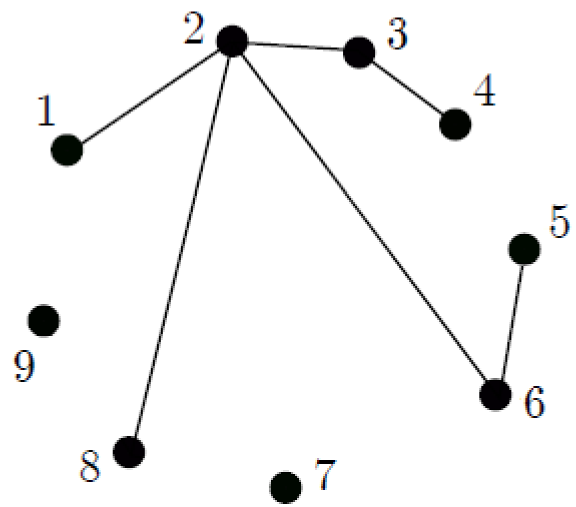
	2	8	8	9	1	4	3	5



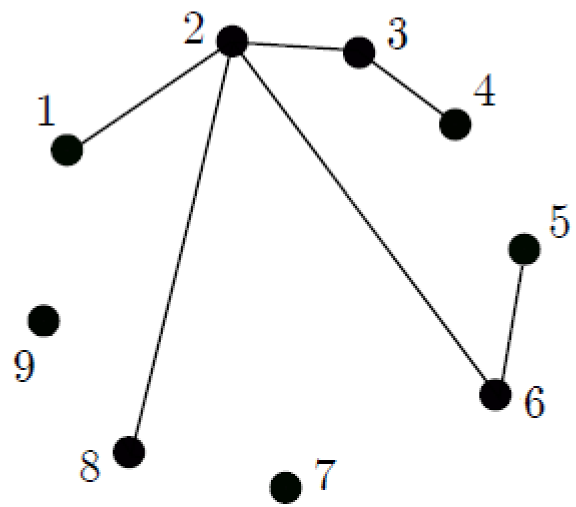
	8	8	9	1	4	3	5	6



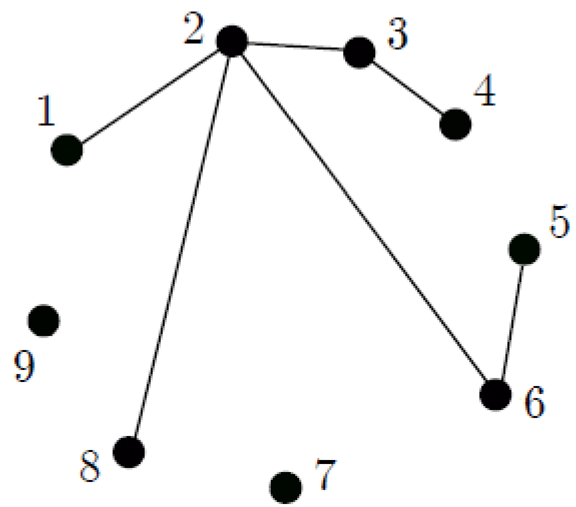
	8	8	9	1	4	3	5	6



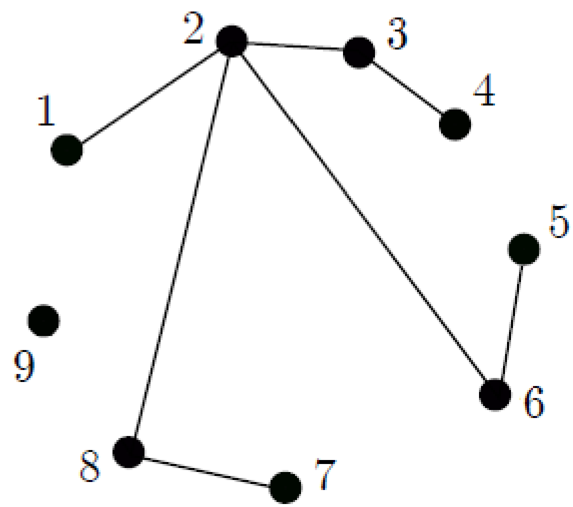
	8	8	9	1	4	3	5	6



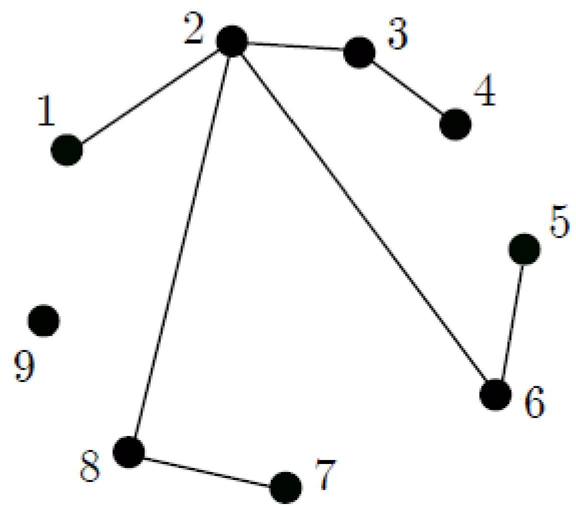
	8	9	1	4	3	5	6	2



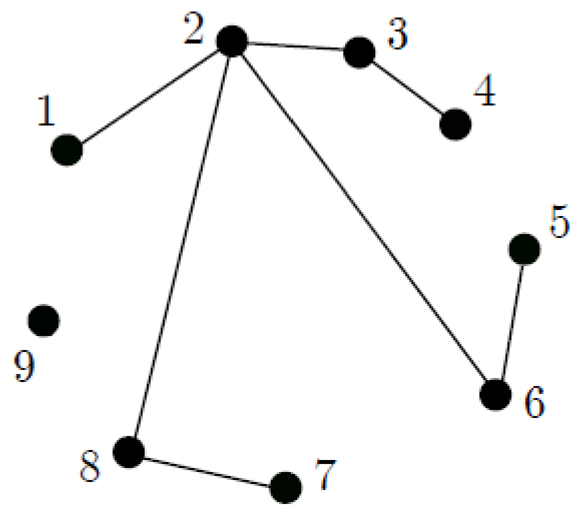
	8	9	1	4	3	5	6	2



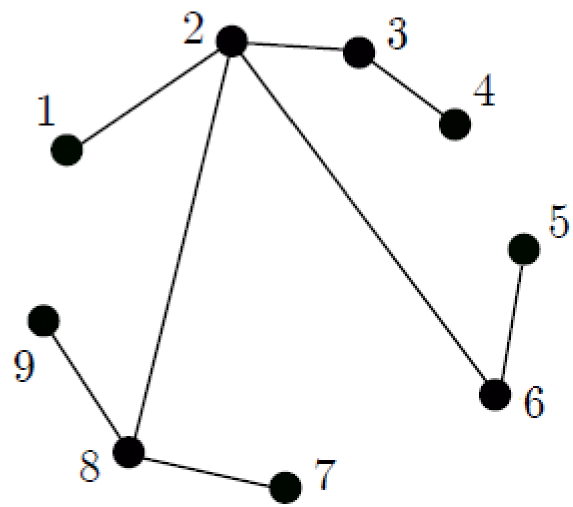
	8	9	1	4	3	5	6	2



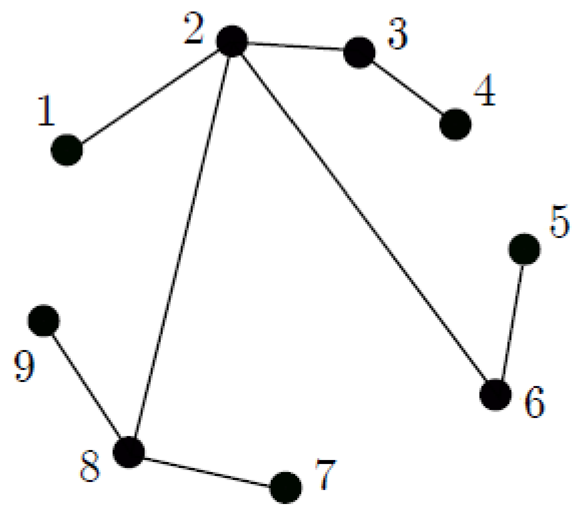
	9	1	4	3	5	6	2	7



	9	1	4	3	5	6	2	7



	9	1	4	3	5	6	2	7



	1	4	3	5	6	2	2	8

PrüferDekódolás(K, n)

legyen F egy üres gráf
for $i = 1, 2, \ldots, n-1$ **do**
    legyen x a K sorozat első eleme
    legyen y a legkisebb természetes szám, amely nincs benne K-ban
    rajzoljunk egy (x, y) élt F-be
    töröljük x-et a K elejéről, és adjuk hozzá a végére y-t
return F

4.1 tétel: (Cayley tétele)

Egy n csúcspontú teljes gráfnak n^{n-2} különböző feszítőfája van.

4.1 tétel: (Cayley tétele)

Egy n csúcspontú teljes gráfnak n^{n-2} különböző feszítőfája van.

Bizonyítás:

Címkézzük meg a gráf csúcspontjait az első n természetes számmal. Tudjuk, hogy egy n csúcspontú fa Prüfer-kódja $n-1$ elemből áll, ahol az utolsó elem mindig a gyökér címkéje, az első $n-2$ elem pedig az első n természetes szám bármelyike lehet.

4.1 tétel: (Cayley tétele)

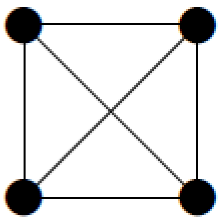
Egy n csúcspontú teljes gráfnak n^{n-2} különböző feszítőfája van.

Bizonyítás:

Címkézzük meg a gráf csúcspontjait az első n természetes számmal. Tudjuk, hogy egy n csúcspontú fa Prüfer-kódja $n-1$ elemből áll, ahol az utolsó elem mindig a gyökér címkéje, az első $n-2$ elem pedig az első n természetes szám bármelyike lehet.

Összesen n^{n-2} ilyen ismétléses variáció van, s ez megegyezik az n csúcspontú teljes gráf összes feszítőfájának számával □

A K_4 gráfnak összesen 16 különböző feszítőfája van:



K_4

A K_4 gráfnak összesen 16 különböző feszítőfája van: