1. témakör: alapfogalmak, hatékonyság-, bonyolultságelemzés

Az algoritmus fogalma: nem definiáljuk!
Jellemzői:

1. Egyértelmű leírás
2. Véges lépések sorozata
3. Teljes, azaz az adott feladatkörhöz tartozó összes feladat megoldására jó. Még a szélső esetekre is.

Bonyolultság (hatékonyság): Az O (ordó) függvény.
Elemi lépés: végrehajtási ideje nem függ a végrehajtásban szereplő adatok nagyságától, illetve megadható egy olyan korlát, amely mindig nagyobb, mint a lépés végrehajtási ideje.
Egy algoritmus elemi lépéseinek száma becsülhető. A becslés eredményét az algoritmusra jellemző O függvénnyel adhatjuk meg.
Megkülönböztetünk:

1. Átlagos hatékonyságot.
2. Legrosszabbeset-hatékonyságot.
3. Optimáliseset-hatékonyságot.

A gyakorlatban az átlagos hatékonyság (bonyolultság) a mérvadó, kivétel néhány speciális eset.
Néhány példa bonyolultságra:

1. Konstans bonyolultság O(1): másodfokú egyenlet megoldása.
2. Polinomiális bonyolultság $O(n^k)$:
    a. n szám összeadása: O(n)
    b. Rendezések: Legrosszabb eset általában $O(n^2)$ néhány esetben $O(n\log_2(n))$, átlagos eset $O(n^2)$, némelynél $O(n\log_2(n))$. (Vizsgált rendezések: minimumkiválasztásos, buborékos rendezés, gyorsrendezés. Mindegyiknél átlagos és legrosszabbeset-vizsgálat. Lineáris hatékonyságú rendezési algoritmus valamelyike.)
3. NP-teljesség (csak szemléletesen: még nem ismerünk polinomiális hatékonyságú algoritmust a megoldásukra). Továbbiakban ezeket nehéz feladatnak hívjuk.

2. témakör: gráfok bejárása

A gráfokkal kapcsolatos alapfogalmak felidézése: mi a gráf, irányított, irányítatlan eset, fagráf, összefüggőség, stb.

Gráfbejárások:

1. (Irányított) fagráfokra:
    a. mélységi bejárás:

    input: a fagráf a gyökérpontjával azonosítva.
    output: a fagráf, minden pontján az elérést jelző címkével.
    segédszerkezet: verem.

    1. lépés: Tegyük a verembe a gyökérpontot, és jelöljük meg.
    2. lépés: Ha a verem üres, vége.
    3. lépés: Vegyük ki a veremből a soron következő pontot.
    4. lépés: A kivett pont minden gyerekét jelöljük meg, és tegyük a verembe.
    5. lépés: Újra a 2. lépés következzen.

    b. széltében való bejárás:

    input: a fagráf a gyökérpontjával azonosítva.
    output: a fagráf, minden pontján az elérést jelző címkével.
    segédszerkezet: sor.

    1. lépés: Tegyük a sor végére a gyökérpontot, és jelöljük meg.
    2. lépés: Ha a sor üres, vége.
    3. lépés: Vegyük ki a sor elejéről a soron következő pontot.
    4. lépés: A kivett pont minden gyerekét jelöljük meg, és tegyük a sor végére.
    5. lépés: Újra a 2. lépés következzen.

    c. egyenletes bejárás:

    input: a fagráf a gyökérpontjával azonosítva.
    output: a fagráf, minden pontján az elérést jelző címkével.
    Ismert egy súlyfüggvény, amely a fagráf minden éléhez egy nemnegatív súlyértéket rendel. Az algoritmusban egy ilyen függvényt használunk. A függvény értékeinek kiszámítását az algoritmusból leolvashatjuk.

    1. lépés: Rendeljük a gyökérponthoz 0 súlyértéket és jelöljük meg.
    2. lépés: Ha nincs jelöletlen pont, vége.
    3. lépés: Keressük meg azt a jelöletlen pontot, amely a legközelebb van a jelölt pontokhoz, azaz amelyre a jelöletlen pont jelölt ősének súlya és a ponthoz az ősből vezető él súlyának összege minimális.
    4. lépés: A megtalált pont súlya legyen a megtalált minimum, jelöljük meg a pontot.
    5. lépés: Újra a 2. lépés következzen.

2. (Irányított) általános gráfokra:
    a. mélységi bejárás:

    input: a gráf, és a bejárás kezdőpontja.
    output: a gráf, minden pontján az elérést jelző címkével.
    segédszerkezet: verem.

    1. lépés: Tegyük a verembe a kezdőpontot, és jelöljük meg.
    2. lépés: Ha a verem üres, vége.
    3. lépés: Vegyük ki a veremből a soron következő pontot.
    4. lépés: A kivett pont minden jelöletlen szomszédját jelöljük meg, és tegyük a verembe.
    5. lépés: Újra a 2. lépés következzen.

    b. széltében való bejárás:

    input: a gráf, és a bejárás kezdőpontja.
    output: a gráf, minden pontján az elérést jelző címkével.
    segédszerkezet: sor.

    1. lépés: Tegyük a sor végére a kezdőpontot, és jelöljük meg.
    2. lépés: Ha a sor üres, vége.
    3. lépés: Vegyük ki a sor elejéről a soron következő pontot.
    4. lépés: A kivett pont minden jelöletlen szomszédját jelöljük meg, és tegyük a sor végére.
    5. lépés: Újra a 2. lépés következzen.

    c. egyenletes bejárás:

    input: a gráf és a bejárás kezdőpontja.
    output: a gráf, minden pontján az elérést jelző címkével.
    Ismert egy súlyfüggvény, amely a gráf minden éléhez egy nemnegatív súlyértéket rendel. Az algoritmusban használt súlyfüggvény értékeinek kiszámítási módját az algoritmus tartalmazza.

    1. lépés: Rendeljük a gyökérponthoz 0 súlyértéket és jelöljük meg.
    2. lépés: Ha nincs jelöletlen pont, vége.
    3. lépés: Keressük meg azt a jelöletlen pontot, amely a legközelebb van a jelölt pontokhoz, azaz azt a pontot, amelyhez vezet jelölt pontból él, az ilyen jelölt él súlyának, valamint az él kezdőpontja súlyának összege a lehető legkisebb.
    4. lépés: A megtalált pont súlya legyen a megtalált minimum, jelöljük meg a pontot.
    5. lépés: Újra a 2. lépés következzen.

Útkeresés a gráfban:

1. Adott pontból minden más pontba:
   Adott pontból adott pontba való út megkeresése nem egyszerűbb!
   Bármelyik gráfbejáró algoritmus alkalmas útkeresésre, ha az elért pontokat megfelelően címkézzük.

A címkézés legyen:

A kezdőpontot címkézzük meg önmagával.

Minden más pontra kerüljön fel címkeként annak a pontnak az azonosítója, ahonnan az adott pontot elértük.

A megtalált út felírásának algoritmusa:

Az algoritmus paramétere: végpont azonosítója.

A működés:

1. lépés: Ha a végpont címkéje nem egyezik meg a végpont azonosítójával, akkor írjuk fel az utat a végpont címkéjéig.
2. lépés: Írjuk fel a végpont azonosítóját.

Az eljárás végrehajtása után a kezdőpontból a végpontig az úthoz tartozó pontok azonosítóinak listája látszik az érintés sorrendjében.

2. Legrövidebb út adott pontból minden más pontba:

Adott pontból adott pontba való út megkeresése nem egyszerűbb!

Kétféleképpen definiálható a legrövidebb út:

a. legkevesebb élet tartalmazzon.
   A széltében való bejárás a megismert címkézéssel a megoldás!
b. ha ismerjük az élek hosszát, az út éleinek összhossza a lehető legkisebb legyen.
   Az egyenletes bejárás a megismert címkézéssel egy lehetséges megoldás.
   Ezen kívül számtalan speciális algoritmus ismert (pl.: Disktra...)

3. Legrövidebb út minden pontból minden pontba:

Mátrixalgoritmus:

Egy mátrixsorozat előállítása, amelynek utolsó eleme a legrövidebb utak hosszát tartalmazza.

input: a gráf élhosszait tartalmazó C mátrix, n a mátrix mérete.
output: a legrövidebb utak hosszát tartalmazó T mátrix.

1. lépés: T := C, $l := 1$.
2. lépés: Túj legyen a következő:
   $túj_{i,j}$:=min($t_{i,j}$, $t_{i,l}$+$t_{l,j}$) minden $1 <= i <= n$ és $1 <= j <= n$ értékekre.
   T := Túj
3. lépés: inc(l). Ha $l > n$, akkor vége, különben a 2. lépés

Az algoritmus kiegészítve megfelelő címkézéssel, nemcsak a minimális élek hosszát adja meg, de a minimális utak felírását is lehetővé teszi.

3. témakör: folyamhálózatok

Egy irányított gráfot folyamhálózatnak tekintünk, ha élein értelmezve van két függvény a következő módon:
k kapacitásfüggvény:
$\mathrm{k} : \mathbf{E} \to \mathbf{R}^+ \cup \{0\}$
f folyamfüggvény:
$\mathrm{f} : \mathbf{E} \to \mathbf{R}^+ \cup \{0\}$, és $\mathrm{f} \le \mathrm{k}$

Az általánosság megszorítása nélkül feltehetjük, hogy a gráfunk teljes, ugyanis ha nem az, nulla kapacitással felvehetünk fiktív éleket, amivel a gráf teljessé tehető. Ez a bővítés a folyamfüggvényre annyiban hat, hogy a fiktív éleken a folyam nulla!

A folyamhálózat valamely *i* pontja forrás, ha $\sum_k \mathrm{f}(k,i) - \sum_k \mathrm{f}(i,k) < 0$ nyelő, ha $\sum_k \mathrm{f}(k,i) - \sum_k \mathrm{f}(i,k) > 0$.
Egy folyamhálózat az általánosság megtartásával mindig átalakítható úgy, hogy egyetlen nyelője és egyetlen forrása legyen; elérhető az is, hogy $\sum_i \mathrm{f}(i,1) = 0$. A továbbiakban ilyen hálózatokról beszélünk, és forrásnak 1-es, nyelőnek az *n*-dik pontot tekintjük.
Egy fenti folyamhálózat folyamértékén a $\sum_i \mathrm{f}(1,i)$ értéket értjük.

Maximálisfolyam-problémán a következő optimumfeladatot értjük:
Keressük azt a folyamfüggvényt, amelyre a folyamérték a maximális:

A megoldó algoritmus (Ford-Fulkerson):

Input: a folyamhálózat ismert folyamfüggvénnyel.
Output: a folyamhálózat egy optimális folyamfüggvénnyel.

1. lépés: Mentsük el a folyamhálózat kapacitásfüggvényét!
2. lépés: Változtassuk meg a kapacitásfüggvényt a következő módon:
   $\mathrm{k}(i,j) := \mathrm{k}(i,j) - \mathrm{f}(i,j) + \mathrm{f}(j,i)$
3. lépés: Keressünk legkevesebb élből álló utat az 1. pontból az *n*-edikbe olyan éleket felhasználva, amelyeken a kapacitás nem nulla!
4. lépés: Ha nincs ilyen út, akkor tegyük vissza a hálózatra az elmentett kapacitásfüggvényt, módosítsuk az f függvényt a következőképpen:
   Minden 1 <= *i* < *j* <= *n* pontpárra $m := \min(\mathrm{f}(i,j), \mathrm{f}(j,i))$
   $\mathrm{f}(i,j) := \mathrm{f}(i,j) - m$, $\mathrm{f}(j,i) := \mathrm{f}(j,i) - m$. A keletkező f maximális folyamfüggvény, és vége.
5. lépés: Ha van ilyen út, számítsuk ki az út élei kapacitásának minimumát (*m*)!
6. lépés: Változtassuk meg a folyamfüggvényt és a kapacitásfüggvényt a következőképpen: $\mathrm{f}(i,j) := \begin{cases} \mathrm{f}(i,j), & \text{ha } i,j \text{ él nincs az úton} \\ \mathrm{f}(i,j) + m, & \text{ha } i,j \text{ él az úton van} \end{cases}$,

$$\mathrm{k}(i,j) := \begin{cases} \mathrm{k}(i,j), \text{ ha sem } i, j, \text{ sem } j,i \text{ nincs az úton} \\ \mathrm{k}(i,j) - m, \text{ ha } i, j, \text{ az úton van} \\ \mathrm{k}(i,j) + m, \text{ ha } j,i \text{ az úton van} \end{cases}$$

, majd folytassuk a 3. lépéssel.

4. témakör: optimumkeresés véges halmazokon

Egy rendszer változtatható paramétereinek lehetséges valós értékeit jelölje rendre $\mathbf{X}_1$, $\mathbf{X}_2$, ... $\mathbf{X}_n$ halmaz, $n > 0$ egész szám, F az $\mathbf{X}_1 \times \mathbf{X}_2 \times \ldots \times \mathbf{X}_n$ halmazon értelmezett $\mathbf{B}^m$ értékkészletű függvény (**B** a Boole-halmaz), c az $\mathbf{X}_1 \times \mathbf{X}_2 \times \ldots \times \mathbf{X}_n$ halmazon értelmezett valós értékű függvény. *tr* eleme $\mathbf{B}^m$-nek és minden koordinátája ***true.***

Optimum-feladatnak nevezzük a következő problémát:
Keressük azt az *x* vektort, amelyre

$$x \in \mathbf{X}_1 \times \mathbf{X}_2 \times \ldots \mathbf{X}_n$$

$$\mathrm{F}(x) = tr$$

$$\mathrm{c}(x) \to \max \quad (\text{vagy} \quad \mathrm{c}(x) \to \min)$$

Egy probléma lehetséges megoldásának nevezzük azt az *x* vektort, amely eleme $\mathbf{X}_1 \times \mathbf{X}_2 \times \ldots \times \mathbf{X}_n$-nek. Jelöljük a lehetséges megoldások halmazát **L**-lel.
Egy probléma megengedett megoldásának nevezzük azt az *x* vektort, amely lehetséges megoldás, és $\mathrm{F}(x) = tr$. Jelöljük a megengedett megoldások halmazát **M**-mel.
Egy probléma optimális megoldása az *x* vektor, ha *x* megengedett megoldás, és minden megengedett *x'* megoldásra $\mathrm{c}(x) \geq \mathrm{c}(x')$ (vagy $\mathrm{c}(x) \leq \mathrm{c}(x')$). Jelöljük az optimális megoldások **O**-val.
Nyilván $\mathbf{L} \supseteq \mathbf{M} \supseteq \mathbf{O}$.

Az F és a c függvények alakja lényegesen befolyásolja a feladat megoldhatóságát! Csak speciális F és c függvényekre vannak hatékony megoldó algoritmusaink. Más esetekben csak úgy van esélyünk a probléma optimális megoldásai közül valamelyiket megtalálni, hogy vesszük a **L** elemeit, egyenként megvizsgáljuk, hogy elemei-e **M**-nek, ha igen, feljegyezzük az elemet és a hozzá tartozó c függvényértéket, ha ez jobb, mint a már megvizsgált **M**-beli elem. Ha **L** összes elemét megvizsgáltuk, akkor a feljegyzett elem az egyik legjobb.
Ez a módszer a leszámlálás módszere.

Megoldásfa: Az **L** halmaz elemeinek egyfajta felsorolása, illetve az **L** egyfajta reprezentációja:
Az egyik lehetséges felépítése:
1. szint: a gyökérpontból annyi élet indítunk, ahány eleme van $\mathbf{X}_1$-nek, az élekre rendre ráírjuk $\mathbf{X}_1$ elemeit.
i-edik szint: az $(i - 1)$. szint minden pontjából annyi élet indítunk, ahány eleme van az $\mathbf{X}_i$-nek, és minden pontnál minden élre rendre ráírjuk az $\mathbf{X}_i$ elemeit. $(1 < i <= n)$

Ebben a fában minden, a gyökértől a levélig vezető út egy lehetséges megoldást reprezentál az éleire írt értékek segítségével.
A megoldásfa egy mélységi bejárása egy leszámlálást valósít meg, ha a bejárást kiegészítjük a levél elérésekor azzal, hogy itt nézze meg, hogy a megoldás megengedett-e, ha igen számolja ki a c függvényértéket, és jegyezze fel a megoldással együtt, ha szükséges. A bejárás után a feljegyzett megoldás az egyik optimális megoldás. Ezt a bejárást Back Track bejárásnak is szokás hívni.
Ha van megoldásfánk, Back Track módszerrel mindig tudunk optimumot előállítani.

Az egészértékű optimumfeladatok másik megoldó módszere a Branch and Bound módszer.

Legyen d a $\mathcal{P}(\mathbf{L})$ halmazon parciálisan értelmezett függvény, értékkészlete a $\mathcal{P}(\mathcal{P}(\mathbf{L}))$.
A d függvény osztályozó függvény **L**-en, ha d **L**-re értelmezve van, és ha valamely $\mathbf{\Omega} \in \mathcal{P}(\mathbf{L})$-re értelmezve van, akkor d(**Ω**) elemei páronként diszjunktak, egyesítésük **Ω**, valamint d d(**Ω**) elemeire is értelmezve van.
Legyen k a $\mathcal{P}(\mathbf{L})$ halmazon parciálisan értelmezett függvény, értékkészlete a valós számok halmaza.
A k függvény korlátozó függvény, ha értelmezve van mindenhol, ahol d osztályozó függvény értelmezve van, valamint k(**Ω**) nem kisebb (nagyobb)**,** mint bármely $x \in \mathbf{\Omega}$ esetén c(x). Ha **Ω** egyelemű, és x megengedett, akkor k(**Ω**)=c(x), különben k(**Ω**) = $-\infty$ ($+\infty$).

A BB eljárás:

1. lépés: Álljon egy fagráf egyetlen pontból, a gyökérpontból, amely jelképezze **L**-et és címkézzük meg k(**L**)-lel.
2. lépés: Keressük meg a fa legnagyobb (legkisebb) címkéjű levelét! Jelölje a jelképezett halmazt **Ω.**
3. lépés: Ha ez a levél egyelemű halmazt jelképez, és a címkéje nem $-\infty$ ($+\infty$), akkor ennek a halmaznak az eleme az optimum-feleadat egyik optimális megoldása, és az algoritmusnak vége.
4. lépés: Ha nem egyelemű a halmaz, és a d függvény nem értelmezett erre a halmazra, akkor az algoritmus nem tudja megoldani az optimum-feladatot.
5. lépés:Ha nem egyelemű a halmaz, és a d függvény értelmezett erre a halmazra, akkor bővítsük a fát ezen a levélen keresztül úgy, hogy annyi levelet teszünk hozzá, ahány eleme van a d(**Ω**)-nak, mindegyik levél egy-egy d(**Ω**)-beli elemet jelképezzen, címkézzük meg mindegyiket a k függvény megfelelő értékével, majd menjünk a 2. lépésre.

Könnyű látni, hogy a lépéssorozat valóban az optimális megoldását adja az optimum-feladatnak, ha egyáltalán befejeződik. A megállás ugyanis nemcsak a d függvénytől, hanem az **L** halmaztól is függ. Végtelen elemszámú **L** esetén a BB-fa végtelen méretűvé is nőhet, aminek következménye, hogy a fenti lépéssorozat a 2. és 5. lépések között végtelen sokszor ismétlődhet, azaz megállására nincs garancia. Biztosan befejeződik a lépéssorozat, azaz algoritmusnak nevezhető, ha **L** véges. A BB algoritmus szélsőséges esetben ugyancsak egyfajta leszámláláshoz vezet, de általában a leszámlálásnál lényegesen kevesebb elem vizsgálata után megkapjuk az optimumot.
(A módszer leírásában az alternatív értékek az optimum-feladat azon változatára szólnak, amelyben a minimális célfüggvényértéket keressük.)

Mivel **L** elemeinek száma legtöbbször nagy, sok esetben nincs lehetőség az összes elem megvizsgálására, sőt a BB által nyújtott részleges vizsgálatra sem. Ilyenkor „célirányosan” vizsgálunk, tulajdonképpen felépítünk egyetlen **L**-beli elemet, és ezt tekintjük optimumnak. Mivel nem garantált, hogy valóban az optimális megoldások egyikét kapjuk, kvázioptimumról szoktunk ebben az esetben beszélni, a keresőmódszert pedig heurisztikus módszernek hívjuk. A heurisztikus módszerek tulajdonképpen valamely megoldásfának a gyökérből az egyik levélhez vezető útját építik fel.

Heurisztikus módszerek eredményeinek vizsgálata:
Fontos tudni, hogy az eredménytől mit várhatunk, mennyire tér el az optimumtól.

Legrosszabb eset és átlagos eset vizsgálata.
Matematikai eszközökkel néha nehéz a vizsgálat, és az eredmények is szerények lehetnek, ezért empirikus vizsgálattal, a futási eredmények összevetésével jellemezhetjük a heurisztikus módszereink „jóságát”.

Metaheurisztikus módszerek: A kvazioptimumot nem egyetlen megengedett megoldással adják meg, hanem iteratív módon, a megengedett megoldásokon haladva javítanak, amíg valamilyen stacionárius állapotot vagy valamilyen időkorlátot el nem érnek.

Néhány egyszerű nehéz feladat, és megoldó heurisztikus algoritmusuk:

Hátizsák-probléma:
Adott egy doboz, aminek ismerjük a kapacitását (a kapacitás lehet térfogat, tömeg, súly stb.), valamint adott véges sok dolog, amelyeknek ismerjük a használati értékét, valamint ismerjük azt is, hogy a doboz kapacitásának mekkora hányadát foglalja el, ha a dobozba tesszük. Pakoljuk meg a dobozt úgy, hogy a benne lévő tárgyak használati összértéke a lehető legnagyobb legyen.

A megoldó heurisztikus algoritmus:

Input: a doboz kapacitása és a tárgyak adatai.
Output: a dobozba került tárgyak sorozata.

1. lépés: Számítsuk ki a tárgyak értékének és kapacitáshányadának hányadosát!
2. lépés: Rendezzük sorba a hányadosok szerint a tárgyakat, és menjünk a sor elejére!
3. lépés: A soron következő tárgyat vizsgáljuk meg: ha belefér a dobozba, tegyük bele, he nem ne! lépjük a következő tárgyra, ha nincs vége, ha van, ismételjük meg a 3. lépést!

A harmadik lépésben mindig azt a tárgyat tesszük a dobozba, amely relatívan a legnagyobb mértékben növeli a doboz összértékét abban a reményben, hogy ezzel a legjobban töltjük ki a dobozt! Az ilyen algoritmusokat, amelyek lokális optimum felhasználásával próbálják a globális optimumot elérni, mohó algoritmusoknak hívjuk. (Ilyen mohó algoritmus volt az egyenletes bejárás felhasználásával megvalósított minimálisútkereső-algoritmus is.) A mohó algoritmusok egy része valóban optimális megoldást szolgáltat, más része nem garantálja az optimum előállítását.

Ládapakolás:
Ismert egy gyártási rendszer, amelyben az előállított termékek a végfeldolgozás után véletlenszerűen követik egymást. Ismerjük, hogy az egyes termékek egy egységnyi kapacitású ládának hányad részét foglalják el. Pakoljuk ládákba a termékeket úgy, hogy a lehető legkevesebb ládát használjuk fel a pakoláshoz.

1. lépés: Nyissunk egy üres ládát.
2. lépés: Nézzük meg, hogy a soron következő termék belefér-e a nyitott ládába, ha igen beletesszük és folytatjuk a 2. lépésen, ha nem bezárjuk a ládát, veszünk egy új nyitott ládát és újra a 2. lépéssel folytatjuk a pakolást.

Az algoritmus egyszerű, sok ötlettel módosítható lenne. (Pl. több, de adott korlát alatt maradó nyitott ládával dolgozunk.) A tapasztalat és az elméleti eredmények is azt mutatják, hogy ezek a módosítások alig javítanak a ládafelhasználáson!

5. témakör: Euler-kör keresése

Tekintsük a következő problémát:
Adott egy úthálózat. Egy útkereszteződésből kiindulva járjuk végig az úthálózatot úgy, hogy minden útszakaszon pontosan egyszer menjük végig, és érjünk vissza a kiindulási pontba. A bejárásnál ügyeljük a közlekedési korlátok betartására (pl.: egyirányú utca, stb.). Mivel a feladat ilyen megfogalmazása kínai eredetű, ezért ezt a problémát kínai postásproblémának is szokás nevezni. Bár érdemes megjegyezni, hogy a kőnigsbergi hidak problémájaként is ismert. Ezt az utóbbi feladatot Euler oldotta meg!

Modell:
Az úthálózatot jelképezze az a gráf, amelynek csomópontjai az útkereszteződésnek, élei az útkereszteződések között lévő útszakaszok lesznek. Mindkét irányba járható útszakasz irányítatlan éllel egyirányban járható útszakasz irányított éllel szerepeljen! Keressünk a gráfban olyan körsétát, amelyben minden él pontosan egyszer szerepel!
Megoldható-e?
Euler-feltétele irányítatlan gráfra: Akkor és csak akkor megoldható, ha a gráf minden pontja páros fokszámú.
Euler-feltétele irányítatlan gráfra kiterjesztve: Akkor és csak akkor megoldható, ha minden pont kifoka megegyezik a befokkal.
Euler-feltétele vegyes gráfra kiterjesztve: Akkor és csak akkor megoldható, ha minden pontra igaz: irányítatlan élszám–| kifok–befok| nemnegatív és páros.

Megoldó algoritmus az irányítatlan esetre:

input: a gráf.
output: a körséta élei egy sorban.

1. lépés: Tegyük jelöletlenné a gráf éleit, a sor legyen üres.
2. lépés: Keressünk a gráfban olyan pontot, amelyhez csatlakozik jelöletlen él. és ha sor nem üres, akkor jelölt él is.
3. lépés: Ha nem találtunk ilyet, vége és a sor a keresett körsétát tartalmazza.
4. lépés: Ha a sor nem üres, keressünk a sorban egy olyan élet, amelynek végpontja az előbb megtalált pont. nevezzük ezt az élet aktuálisnak.
5. lépés: Vegyünk a pontból egy jelöletlen élet, ha a sor nem üres, szúrjuk be a sorba az aktuális él mögé, ha a sor üres volt, akkor az első helyre. A most beszúrt él legyen az aktuális. Jelöljük meg ezt az élet, és lépjük ennek az élnek a végpontjába.
6. lépés: Ha ebben a pontban van jelöletlen él, akkor az 5. lépés következzen, ha nincs, akkor a 2. lépés.

Irányított, illetve vegyes gráfok esetén értelemszerűen módosítva az eljárást a körséta előállítható.

Mi van akkor, ha nem teljesül az Euler feltétel?
Módosított postás probléma: Hogyan járható be az úthálózat úgy, hogy visszaérjünk a kiindulási pontba, és minden útszakaszt legalább egyszer érintve a lehető legrövidebb utat járjuk be.

A matematikai modellben olyan körsétát keresünk, amelyben minden él legalább egyszer szerepel, és a séta hossza minimális.

Ez nehéz feladat!

Megoldása heurisztikus módszerrel:

1. lépés: Keressük meg azokat a pontokat, amelyre nem teljesül az Euler-feltétel.
2. lépés: Keressük meg ezen pontok között a legrövidebb utakat az összes lehetséges módon.
3. lépés: Válasszuk ki a lehető legtöbb pontpárt, úgy, hogy a kiválasztott pontpárokhoz tartózó minimális utak összhossza a lehető legkisebb legyen. (Ez a lépés nem egyszerű, de azokban az esetekben, amikor kevés az Euler-feltételt nem teljesítő pont – pl. csak két ilyen van –, megoldható.)
4. lépés: Vegyük sorra ezeket a minimális utakat, és vegyük hozzá új élként a gráfhoz ezeknek az utaknak az éleit (következményképpen: két pont között több él is lehet).
5. lépés: Ha az új gráfban nem teljesül az Euler-feltétel, akkor újra az 1. lépés
6. lépés: Alkalmazzuk a postásprobléma megoldó algoritmusát az új gráfra.

6. témakör: hozzárendelési feladat és a magyar-módszer

A következő egészértékű lineáris programozási feladat a hozzárendelési feladat,

$x_{i,j} \in \{0,1\}$ minden i, j-re

$\sum_i x_{i,j} = 1$ minden j-re

$\sum_j x_{i,j} = 1$ minden i-re

$\sum_{i,j} x_{i,j} c_{i,j} \to \min$

Vegyük észre, hogy a feladatot a $c_{i,j}$ értékek határozzák meg! Két hozzárendelési feladatot ekvivalensnek mondunk, ha az egyik optimális megoldása a másiknak is optimális megoldása, és fordítva.

Megoldási módszere König és Egerváry eredményein alapul, kidolgozója Kuhn, 1956. Kuhn magyar-módszernek nevezte el. A módszer lényeg: ekvivalens hozzárendelési feladatokon keresztül olyan feladathoz jutni, amelyről egy optimális megoldás könnyen leolvasható.
Ilyen hozzárendelési feladat: a $c_{i,j}$ értékei között n darab független nulla van!

Egy mátrix nullái független nulla rendszer alkotnak, ha ezeket a nullákat megjelölve egyetlen sorban és oszlopban sem lesz egynél több nulla megjelölve.
Egy mátrixban maximális független nulla rendszer a független nulla rendszer, ha nulláinak száma nem kisebb, mint a mátrix bármely független nulla rendszerben lévő nullák száma.
Nyilván könnyen leolvasható egy optimális megoldás abban a hozzárendelési feladatban, amelyhez tartozó **C** mátrixban n független nulla van! (A független nulláknak megfelelő x-ek legyenek 1 értékűek, a többi legyen 0 értékű!)
A megoldó módszer olyan – az eredetivel ekvivalens – hozzárendelési feladatot állít elő, amelyben van n független nulla!

Az algoritmus:

Input: a $c_{i,j}$ értékeket tartalmazó **C** mátrix.
Output: a hozzárendelést definiáló **X** mátrix.

1. lépés: Vonjuk ki a C mátrix minden sorának elemeiből a sor legkisebb értékét!
2. lépés: Vonjuk ki a C mátrix minden oszlopának elemeiből az oszlop legkisebb értékét!
3. lépés: Oszloponként haladva keressünk a **C** mátrixban független nulla rendszert úgy, hogy a lehető legkisebb indexű nullát vegyük be a független nullák közé! Csillagozzuk meg a kiválasztott nullákat!
4. lépés: Ha a megcsillagozott nullák száma n, akkor legyen
   $x_{i,j} = \begin{cases} 1 \text{ ha az } c_{i,j} = 0 \text{ és csillagos} \\ 0 \text{ különben} \end{cases}$, és vége.
5. lépés: Jelöljük meg + jellel a csillagos nullák oszlopát!

6. lépés: Keressünk sorfolytonosan szabad nullát! (olyan nullát, amelynek sem sora, sem oszlopa, sem önmaga nincs megjelölve!)
7. lépés: Ha találtunk, akkor keressünk a sorában csillagos nullát! Ha van, folytassuk a 9. lépéssel, ha nincs, akkor a 10. lépéssel.
8. lépés: Ha nem találtunk szabad nullát, akkor számítsuk ki azoknak az elemeknek a minimumát, amelyeknek sem a sora, sem az oszlopa nincs megjelölve. Ezt a minimumot vonjuk ki minden ilyen elemből, és adjuk hozzá minden olyan elemhez, amelyeknek mind a sora, mind az oszlopa meg van jelölve, majd folytassuk a 6. lépéssel.
9. lépés: Jelöljük meg + jellel a szabad nulla sorát, vegyük le a jelölést a csillagos nulla oszlopáról, tegyünk , jelet a szabad nullára, majd folytassuk a 6. lépéssel.
10. lépés: A szabad nullára tegyünk , jelet.
11. lépés: Képezzünk láncot ebből a nullából kiindulva a következőképpen: a lánc első szeme a most megvesszőzött nulla legyen.
12. lépés: Az eddig felírt lánc utolsó szemének oszlopában keressünk csillagos nullát!
13. lépés: Ha nincs, a lánc szemein végighaladva cseréljük le a vesszős nullák vesszőit csillagra, majd töröljük le az összes vesszőjelet a nullákról, töröljük le a + jeleket az összes sorról és oszlopról, majd folytassuk a 4. lépéssel.
14. lépés: Ha van, akkor tegyük a láncba ezt a nullát, keressük meg a sorában lévő vesszős nullát, tegyük a lánc végére, és folytassuk a 12. lépéssel.

7. témakör: Hamilton-körök keresése

A probléma hétköznapi megfogalmazásban:
Ismerünk egy városokból álló rendszert. A városok egy része vagy mindegyike utakkal közvetlenül össze van kötve egymással. Az egyik városban egy ügynök lakik, akinek az a feladata, hogy látogassa meg az összes várost úgy, hogy minden városban pontosan egyszer forduljon meg, a lehető legkevesebbet utazzon és az útja végén abba a városba érkezzen, ahol lakik.
A probléma modellje: Tekintsük azt a gráfot, amelynek pontosan annyi pontja van, ahány város, tehát minden városhoz hozzárendelhető egy pont. Ezek közül a pontok közül azok vannak élekkel összekötve, amelyekhez tartozó városokat közvetlen út köt össze. Minden élhez legyen hozzárendelve annak az útnak a hossza, amelyet az él jelképez. Keressük a gráfban azt a legrövidebb kört, amely minden pontot érint. (Egy gráf minden pontját tartalmazó kört Hamilton-körnek nevezzük.)
Nehéz probléma!

Lineáris programozási modellje:

$x_{i,j} \in \{0,1\}$ minden i, j-re

$\sum_i x_{i,j} = 1$ minden j-re

$\sum_j x_{i,j} = 1$ minden i-re

$\sum_{i \in \mathbf{Q}, j \in \mathbf{Q}', \mathbf{Q} \cup \mathbf{Q}' = \mathbf{P}, \mathbf{Q} \cap \mathbf{Q}' = \emptyset} x_{i,j} \geq 1$ minden, legalább egyelemű **Q** és **Q'**-re.

$\sum_{i,j} x_{i,j} h_{i,j} \rightarrow \min$

Vegyük észre, hogyha elhagyjuk a negyedik feltételt, akkor egy hozzárendelési feladatot kapunk!

Megoldása heurisztikus módszerekkel:
1. heurisztika: Ismert kezdőkör bővítése a körhöz legközelebb lévő pont hozzáadásával.

Input: a gráf.
Output: a kész Hamilton-kör.

1. lépés: A kezdőkör megadása. (Legegyszerűbb: legyen a kezdőkör a gráf egy tetszőleges pontja!)
2. lépés: Ha a kör minden pontot tartalmaz, vége!
3. lépés: Keressük meg a körhöz legközelebb lévő pontot! (Azaz válasszuk ki azt a legrövidebb élet, amelynek egyik végpontja a körön van, a másik nem.)
4. lépés: Szúrjuk be a körbe a megtalált él körön kívüli végpontját a körön lévő végpont mögé! Folytassuk a 2. lépéssel.

2. heurisztika: Ismert kezdőkör bővítése a körhöz legtávolabbi pont beszúrásával.

Input: a gráf.
Output: a kész Hamilton-kör.

1. lépés: A kezdőkör megadása. (Legegyszerűbb: legyen a kezdőkör a gráf egy tetszőleges pontja!)
2. lépés: Ha a kör minden pontot tartalmaz, vége!
3. lépés: Keressük meg a körhöz legtávolabb lévő pontot! (Azaz válasszuk ki azt a leghosszabb élet, amelynek egyik végpontja a körön van, a másik nem.)
4. lépés: keressük meg a körön azt a két egymást követő pontot, amelyek közé a legtávolabbi pontot beszúrva a kör hossza a legkisebb mértékben növekszik!
5. lépés: Szúrjuk be a két pont közé a legtávolabbi pontot, és folytassuk a 2. lépéssel!

3. heurisztika: Ismert kezdőkör bővítése a legközelebbi pont beszúrásával.

A módszer majdnem teljesen megegyezik az előzővel. A különbség csak annyi, hogy a harmadik lépésben a legtávolabbi pont helyett a legközelebbit választjuk.

BB algoritmus:
Input: egy ***G*** gráf.
Output: a kész Hamilton-kör.

1. Egészítsük ki a ***G*** gráfot + ∞ hosszúságú élekkel azokra az *i*, *j* pontokra, amelyekre *i*-ből nem vezet él *j*-be.
2. lépés: Álljon egy fagráf egyetlen pontból, a gyökérpontból, amely jelképezze a ***G*** gráfot és címkézzük meg a ***G***-hez tartozó hozzárendelési feladat célfüggvényértékével.
3. lépés: Keressük meg a fa legkisebb címkéjű levelét! Jelölje a jelképezett gráfot ***G'***.
4. lépés: Ha a címke hossza + ∞ akkor nincs a ***G*** gráfban Hamilton-kör és az algoritmusnak vége.
5. lépés: Ha ***G'***-höz tartozó hozzárendelési feladat optimális megoldása Hamilton-kör és véges hosszúságú, akkor ez lesz az eredeti ***G*** gráf optimális Hamilton-köre, és az algoritmusnak vége.
6. lépés: Ha nem Hamilton-kör a ***G'***-höz tartotó hozzárendelési feladat megoldása, akkor keressük meg az optimális megoldás által megadott körrendszer legkevesebb élet tartalmazó körét, e kör éleiből sorban egyet-egyet cseréljük + ∞ hosszúságúra. A kapott gráfokhoz tartozó hozzárendelési feladatot megoldva rendeljük mindegyikhez a kapott célfüggvényértéket. Bővítsük a fát a ***G'*** jelű pontjában annyi éllel és levéllel, ahány gráf keletkezett, és mindegyik levél jelképezzen egy-egy ilyen gráfot megcímkézve a megfelelő célfüggvényértékkel, majd folytassuk a 3. lépésben.

Ha biztosan tudjuk, hogy a gráfban van Hamilton-kör, akkor a 4. lépés szükségtelen, valamint célszerű kiegészíteni 6. lépést azzal, hogy a végtelen célfüggvényértékű gráfok ne vegyenek részt a bővítésben. Az algoritmus természetesen evvel a kiegészítéssel sem lesz hatékonyabb, viszont a keletkező BB-fa mérete csökkenthető, így az esetlegesen felmerülő memóriaproblémák talán kikerülhetők.
A 6. lépésben a legkevesebb élet tartalmazó kör választása a BB-fa méretének a lehető legkisebb bővítése miatt történik. A fa „túlburjánzásának" megakadályozására sok javaslat született, ezek alkalmazása csökkentheti a memóriaigényt és a futási időt. Itt nem célunk ezek

ismertetése, hiszen BB algoritmus elvi működését nem befolyásolják ezek a takarékos megoldást segítő tényezők. Persze egy gyakorlatban is használható módszernél mindenképpen figyelembe veendők.

A fenti algoritmusokon kívül számtalan heurisztikus és BB algoritmus létezik, ezekkel nem foglakozunk.