Opcionális feladat **5p.**

A négyoszlopos hanoi tornyok feladata

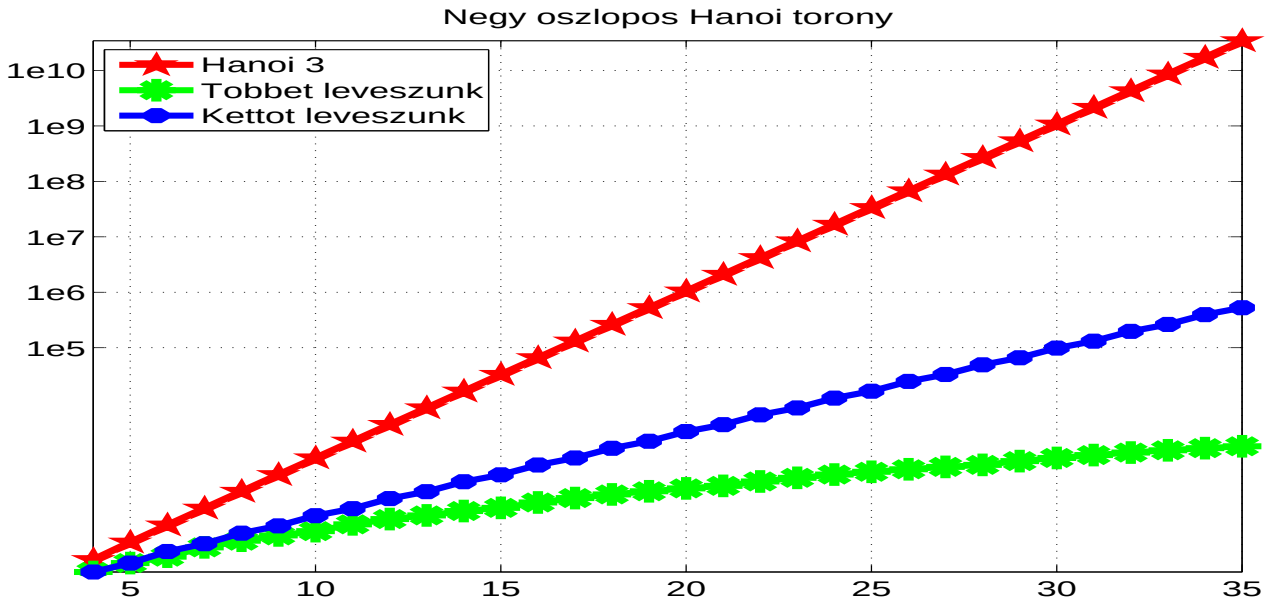
A „klasszikus” három oszlopos hanoi feladatnál három oszlopunk adott, a harmadik oszlopon D korong csökkenő sorrendben. A célunk a korongoknak az első oszlopra való helyezése úgy, hogy szintén csökkenő sorrendben legyenek az oszlopon és az áthelyezés során nem helyezünk nagyobb korongot kisebbre.

```
% harom oszloppal
hanoi(1,I,J,_,[I->J]) :- !.
hanoi(D,I,J,K,LepLista) :-
    D1 is D - 1,
    hanoi(D1,I,K,J,L1),
    hanoi(1,I,J,K,EgyLep),
    append(L1,EgyLep,Lt),
    hanoi(D1,K,J,I,L2),
    append(Lt,L2,LepLista).
```

Mellékelve a háromoszlopos Hanoi tornyot megoldó Prolog kód látható. A megoldás a predikátumok negyedik argumentumában található lépések listája. Belátható, hogy a lista hossza $2^D - 1$.

Feladat, hogy írjunk kódot, mely a hanoi tornyok feladatát négy oszlop esetén oldja meg. A cél, hogy minél kevesebb lépésszámmal tudjuk a *negyedik* oszlopról az elsőre áthelyezni a korongokat. Alább található egy ábra, mely a négy oszlopos hanoi tornyok feladatát összehasonlítja. Az első a klasszikus megoldás, ahol a negyedik oszlopot nem vesszük figyelembe. A másik két sor két más megoldás, mely sokkal kevesebb idő alatt végzi el az áthelyezést (figyeljük meg a logaritmikus skálát).

Írjunk programot, mely az alábbi grafikonok által megjelenített időnél gyorsabban elvégzi az áthelyezést.



Kötelező feladat **8p.**

Bűvös négyzet

Bűvös négyzet az az $N \times N$-es négyzet, melyben az elemek összege megegyezik sorok, oszlopok, valamint a két átló szerint.
Minden pozícióban az $1 \ldots N^2$ számok valamelyike van. Mivel minden oszlopban ugyanaz az összeg, a soronkénti összeget a következő:

$$S_{sor} = \frac{1}{N} \sum_{n=1}^{N^2} n = \frac{1}{N} \cdot \frac{N^2 \left(N^2 + 1\right)}{2} = \frac{N \left(N^2 + 1\right)}{2}$$

Feladatunk, hogy ábrázoljuk a bűvös négyzetek keresését gráf-kiterjesztési feladatként:

- építsük fel a feladat állapotterét (definiáljunk a gráfot a helyes megoldásokat eredményező kitöltések folyamataként)
- definiáljunk egy gráfkiterjesztési procedúrát a feladatra;
- keressük meg az összes lehetséges megoldást gráfkereső (?backtracking?) módszerrel.

Követelmények:

- Dokumentáció, mely tartalmazza a
  1. paraméterterét a feladatnak,
  2. a gráfkiterjesztés lépéseit,
  3. a gráfbejárás sorrendjét.
- Program, mely az N szám ismeretében kiírja (egy TXT állományba) az összes megoldást valamint kiírja a képernyőre a megoldások számát.

Opcionális feladat **10p.**

A Rubik-kígyó $3 \times 3 \times 3$ kocka elemeinek összefűzése az alábbi ábrán látható. Az ábrán kiterített kígyót össze lehet göngyölni egy kockává, melynek minden oldala három kockából áll. Az ábrán látható „konfiguráció" egy síkbeli kiterítése a kockának: olyan konfiguráció, ahol a harmadik koordináta azonos (például nulla). Feladatunk, hogy **alakítsuk vissza** a kígyót egy kockává.

A göngyölést csak megengedett műveletekkel tudjuk végrehajtani: csak olyan hajlatot fordíthatunk, mely a fordítás során – az alatt és annak eredményeként – nem vezet két kocka ütközéséhez.

1. Ábrázoljuk a feladatot Prolog nyelven: találjunk egy ábrázolást, mely kompakt és ugyanakkor a kígyó teljes állapotterét tárolni képes;

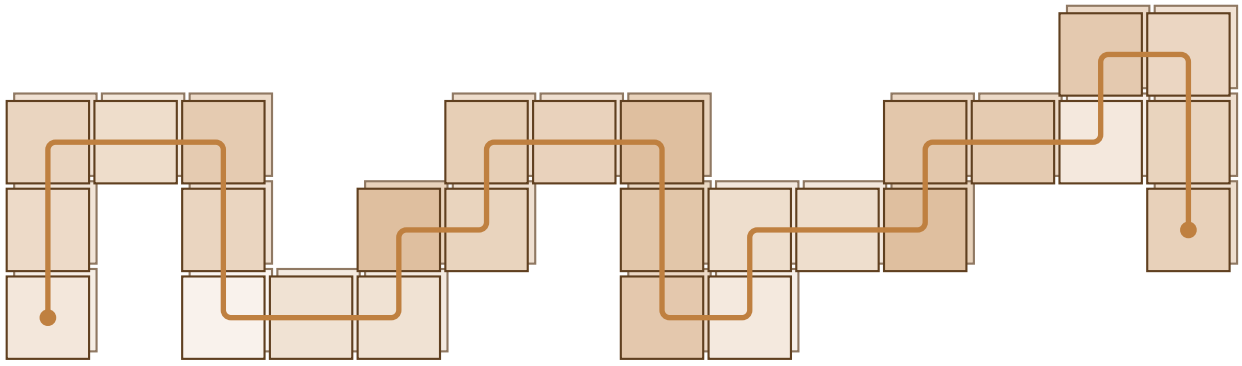
   3p.

2. Fogalmazzunk meg predikátumokat úgy, hogy a kezdeti konfigurációból – amint azt az alábbi ábrán is látjuk – az óhajtott konfigurációba eljutó **parancs-listát** generál. A cél-konfiguráció a kígyó kompakt állapot, amely egy $3 \times 3 \times 3$-as kockát teljesen kitölt. Kitétel a parancs-listánál az, hogy az átalakítások során az elemek ne ütközzenek.

   4p.

3. **program** Írjunk egy megjelenítő programot, mely a fentebb kapott lépések szerint a hajtogatást elvégzi.

   3p.



Opcionális feladatok **Össz: 6p.**

Számjegyek A háromjegyű számok halmazán értelmezzük a következő feladatot: jussunk el egy I számból egy adott G-hez a következő szabályok szerint:

1. Egy számjegyet eggyel növelünk vagy csökkentünk,
2. Egymásutáni lépésekben különböző a számjegyeket módosítunk,
3. A 9-es számjegyhez nem lehet hozzáadni; a 0-ból nem lehet kivonni,
4. Nem tehető olyan lépés, mely a tiltott halmazba tartozó számot állítana elő (a tiltott lista adott).

A feladat megoldásához **használjuk az** A^* **algoritmust**, ahol a kiértékelő függvény a következő:

$$h(I) = |G_1 - I_1| + |G_2 - I_2| + |G_3 - I_3|$$

Forrás: Szalay **2p.**

Particionálás Vizsgáljuk meg, hogy létezik-e az $\{1, \ldots, N\}$, $N = 10$ halmaznak a következő tulajdonságokkal rendelkező $k = 8$ darab részhalmaza:

1. bármely két (2) elem legtöbb két (2) halmazban szerepel, illetve
2. bármely két (2) halmaz metszete legkevesebb két (2) elemből áll.

Feladat:

- írjunk egy programot, mely megkeresi az első ilyen részhalmazrendszert. A program paraméterezhető kell, hogy legyen, a paraméterei az (N, k) páros.
- **Nagyobb teszt:** létezik az $(N = 15\ ,\ k = 12)$ esetre megoldás?

http://www.maths.qmw.ac.uk/~pjc/oldprob.html **3p.**

Pénz Legyen a következő „összeadás”:

$$\begin{array}{cccc} S & E & N & D \\ M & O & R & E \\ \hline M & O & N & E & Y \end{array}$$

ahol:

- A betűk mindegyike számjegy.
- „értelmes” számok: S és M nem lehet 0.

Feladat: Írjunk egy programot, mely a keresést végrehajtja.

Forrás: Szalay **1p.**

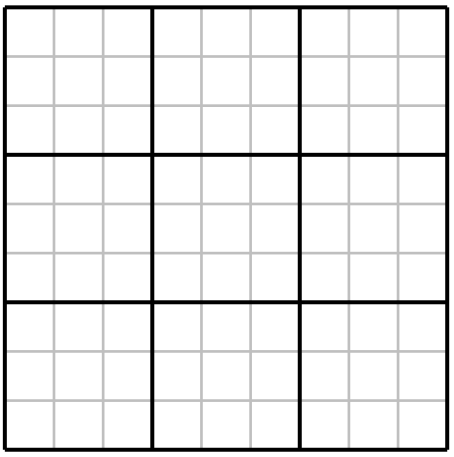
Kötelező feladat **10p.**

Sudoku.

A sudoku egy japán játék, melyben célunk, hogy *számokat* helyezzünk el egy négyzetrácsban úgy, hogy bizonyos szabályoknak eleget tegyenek.

A rács mérete $N^2 \times N^2$, ahol N egy természetes szám és a nagy rács fel van bontva kis négyzetekre, melyek méretei egyenként $N \times N$ (lásd ábra). A szabályok:

- egy oszlopban illetve egy sorban egy szám egyszer szerepelhet;
- minden kisebb négyzetben (ezek száma N^2) egy szám egyszer szerepelhet.



1. Tekintsük az $N = 2$ esetet. Mi lesz ekkor a paramétertér?

   1p.

2. Szintén az $N = 2$ esetre generáljuk a megoldások halmazát egy listában (javasolt a *prolog* alkalmazása).

   1p.

3. Írjunk egy modult, mely „szépen” megjelenít egy megoldást a $N = 2$ illetve a $N = 3$ esetekre.

   1p.

Sudoku rejtvény megoldása $N = 3$ – azaz 9×9 – esetre:

A sudoku rejtvény egy olyan feladat, melyben csak adott pozíciókban vannak számjegyek, és feltételezzük, hogy létezik egy – az összes szabályt kielégítő - megoldása a részlegesen kitöltött táblázatnak.

4 Írjunk programot, mely megtalálja egy részlegesen kitöltött sudoku rejtvény kitöltött változatát (a rejtvényt egy TXT file-ből olvassuk be). Ellenőrizzük, hogy csak egy megoldás van.

3p.

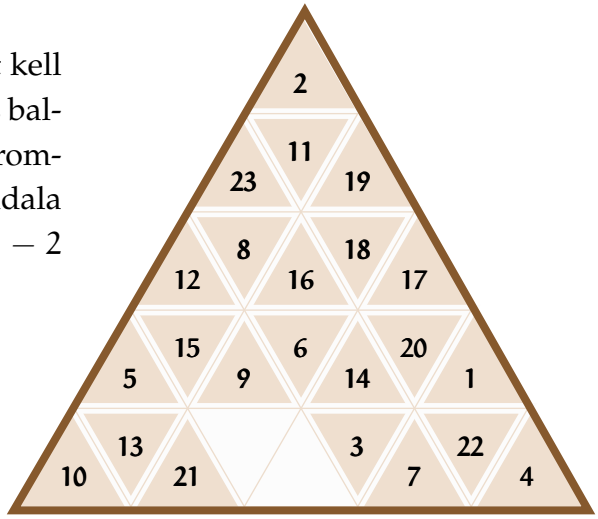
5 Írjunk feladatot, mely generál egy sudoku rejtvényt. Azaz generál egy részlegesen kitöltött feladatot, melynek **csak egy** megoldása van.

4p.

Opcionális feladat **12p.**

A háromszög-puzzle

A mellékelt ábrán háromszög-alapú puzzle-t kell kiraknunk, a helyes sorrend a fentről lefele és balról jobbra haladó számozás, az utolsó két háromszög helye üres. Amennyiben a keret egy oldala az alapháromszögek K-szorosa, összesen $K^2 - 2$ számozott háromszögünk van.



Feladat:

1. Határozzuk meg a feladat állapotterét. Írjunk függvényt, mely egy állapotból megadja az összes lehetséges lépéseket.

   1pt.

2. Írjunk programot, mely egy adott állapotból a cél-állapotba vivő lépések sorozatát adja meg.

   2pt.

3. Írjunk egy **grafikus megjelenítőt**. Írjunk egy programot, mely

   (a) megjeleníti a puzzle-t;

   1pt.

   (b) generál egy puzzle-t (olyan konfigurációt, melyből el *lehet* jutni a célba);

   2pt.

   (c) generálja a megoldás lépéseit, majd azokat animálva végrehajtja;

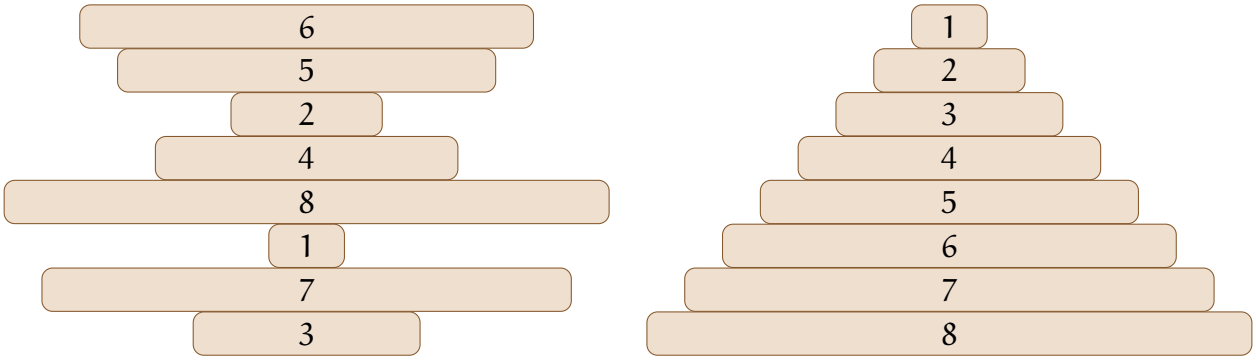
   2pt.

   (d) amennyiben egy üres mező szomszédjára kattintunk, az illető kocka a szabad mezőre „megy”; jelzi a felhasználónak azt, hogy cél-állapotban vagyunk-e.

   4pt.

Opcionális feladat **7pt.**

Forgatós rendezés http://www.inf.unideb.hu/~jeszy



A játék szabálya, hogy egy *köteg* korongot tudunk mozgatni: a legfelül elhelyezkedő k darabot. Célunk, hogy a jobb oldalon látható sorrendet kapjuk.

Feladat:

1. Határozzuk meg a feladat állapotterét. Írjunk függvényt, mely egy állapotból megadja az összes lehetséges lépést.

   1pt.

2. Írjunk programot, mely egy adott állapotból a cél-állapotba vivő lépések sorozatát határozza meg.

   2pt.

3. Írjunk egy **grafikus megjelenítőt**. A program a következőket kell, hogy tudja:

   (a) generál egy kezdőállapotot, melyet megjelenít;

   2pt.

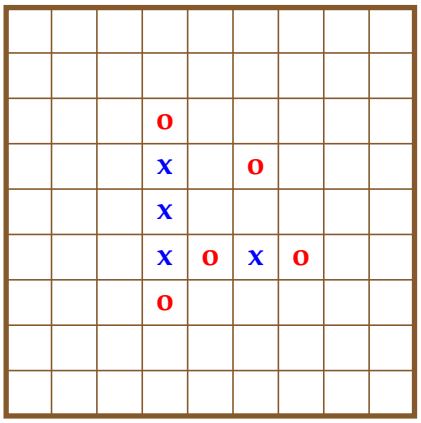
   (b) generálja a megoldás lépéseit, majd azokat animálva végrehajtja;

   2pt.

Kötelező feladat **10p. (+5p. opc)**

Az amőba-játék

Az amőba-játékban a feladatunk egy négyzetekből álló mezőre felváltva rajzolni köröket és kereszteket úgy, hogy előbb nekünk gyűljön ki $J = 5$ darab azonos jel egy sorban vagy egy átló mentén.
Tegyük fel, hogy egy $K \times K$ méretű mezőn játszunk, ahol $K > 4$. Ha a K értéke nagy, akkor a játékot nem tudjuk *teljesen kiterjeszteni*, közelítő becsléseket kell használnunk. Ha $J = K = 3$, akkor bizonyítható, hogy racionális játék során nem lesz győztes.



Feladatok:

1. Implementáljuk az amőba játékot, ahol be tudjuk állítani a J és K értékeket, valamint tudjuk ellenőrizni, hogy egy állapotban egyik játékos nyert-e vagy mehet tovább a játék.

   2pt.

2. A $J = K = 3$ esetre implementáljuk a BOT játékost, amely racionálisan játszik. A lépésekhez használjuk a nyerő stratégia keresésének algoritmusát.

   3pt.

3. Implementáljuk az automata játékost a $K = 4, J = 4$ konfigurációra. Ezekre a paraméterekre a játékos kezdése mindig nyerést jelent. Ha mi kezdünk és nem játszunk jól, akkor is a BOT játékos nyer. Használjuk az ALFA-BÉTA vágást, mellyel gyorsíthatjuk a programunkat.

   4pt.

4. Próbáljuk kiterjeszteni a játékot a $K = 5$ és $J = 4$ esetre.

   1pt.

5. Írjunk programot a $J = 5$ esetre, amikor a K mérete nagyobb, mint 8.

   Opcionális rész: ***5pt.**

Opcionális feladat **9p.**

Egyszerű kugli-játék.

Az alábbi egyszerűsített kuglijátékban minden bábu (teke) egy sorban van elhelyezve, ez – közelítően – merőleges arra az irányra, ahonnan a kugligolyó érkezik. A golyó mérete akkora, hogy vagy egyszerre egy bábút vagy két szomszédosat képes leütni. Vesztes az a játékos, aki nem üt le – nem tud leütni – bábut.

Feladat:

1. Határozzuk meg a játék állapotterét $K = 3$ bábura. Minden lépés előtt határozzuk meg, hogy a lépő játékos nyer vagy veszít ($K = 3$).

   1pt.

2. Számítsuk ki, hogy a kezdő játékos nyertes-e $K = 7$ esetén.

   1pt.

3. Írjunk programot, mely meghatározza, hogy a kezdő játékos nyer-e tetszőleges K esetén.

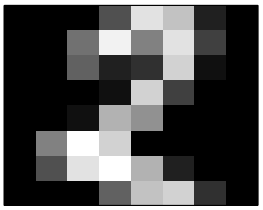
   3pt.

4. Írjunk programot, mely **játszik** a felhasználóval és minden lépésnél optimális lépés szerint cselekszik.

   4pt.

Kötelező feladat **12p. (+5p. opc)**

Felügyelt gépi tanulás.



Egy kettes számjegy.

Az **OCR** tanulási halmazban kézzel írott számjegyek képei találhatók. Az tanulási és teszt adatokat egyenként $8 \times 8 + 1 = 65$ hosszúságú vektorok alakjában tároljuk, ahol az első 64 szám a 8×8-as bittérkép szürke-árnyalatának a kódja, az utolsó érték pedig az osztály kódja 0 és 9 között.

Az adathalmaz a következő állományokból áll:

- `optdigits.tra` – tanulási adatok (3823 darab);
- `optdigits.tes` – teszt adatok (1797 darab);
- `optdigits.tra` – az adatok leírása.

Feladat:

1. Írjunk egy távolságszámoló függvényt, amely két, egyenként $64 = 8 \times 8$ hosszú adatra megadja azok euklideszi távolságát. **2pt.**

2. Az adatok terében definiáljunk egy általánosított skaláris szorzatot – kernel függvényt: $\langle \mathbf{x}, \mathbf{y} \rangle \stackrel{\text{def}}{=} k(\mathbf{x}, \mathbf{y})$ – majd definiáljuk a skaláris szorzat függvényeként a távolságot:

$$\|\mathbf{x} - \mathbf{y}\|^2 = \langle \mathbf{x} - \mathbf{y}, \mathbf{x} - \mathbf{y} \rangle = k(\mathbf{x}, \mathbf{x}) + k(\mathbf{y}, \mathbf{y}) - 2\, k(\mathbf{x}, \mathbf{y}).$$

   Implementáljuk a lineáris, polinomiális és Gauss-féle kerneleket. **2pt.**

3. Implementáljuk a kNN (k-Nearest Neighbor – k legközelebbi szomszéd) algoritmust úgy, hogy az tetszőleges kernelt használhasson. **3pt.**

4. Implementáljuk a centroid módszert úgy, hogy az tetszőleges kernelt használhasson. **3pt.**

5. Számítsuk ki a tanulási és a teszt hibát minden osztályra. **2pt.**

6. 5-szörös kereszt-megerősítést (*cross-validation*) használva határozzuk meg a kernelek optimális paramétereit a kNN és centroid módszerekhez. **opc *5pt.**

Skaláris szorzatok:

$$k_{\text{lin}}(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{d} x_i y_i, \quad k_{\text{pol}}(\mathbf{x}, \mathbf{y};\, p) = \left(1 + \sum_{i=1}^{d} x_i y_i\right)^p, \quad k_{\text{gauss}} = \exp\left(-\frac{\|\mathbf{x} - \mathbf{y}\|^2}{2\sigma^2}\right)$$

Adatbázis: *Optical Recognition of Handwritten Digits*

http://archive.ics.uci.edu/ml/machine-learning-databases/optdigits