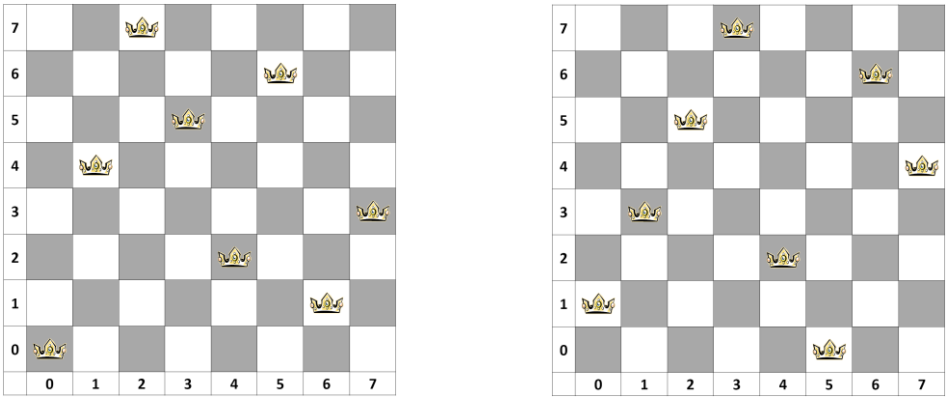
H01_Nyolc vezér probléma [*]

A feladat és leírása

Helyezzünk el a sakktáblán 8 vezért úgy, hogy semelyik kettő se üsse egymást!
A vezérek pillanatnyi helyzetét egy *v* vektorban tároljuk: az *i*-edik vezér az *i*-edik oszlop *v*[*i*]-edik sorában van. (Ezzel kihasználjuk azt, hogy egy oszlopban legfeljebb egy vezér lehet.) Egy adott elhelyezést tehát a *v* vektor értékei adnak meg, pl. [1, 3, 5, 7, 2, 0, 6, 4]. Ez a sorozat tekinthető akár egy nyolcas számrendszerbeli nyolcjegyű számnak is: 13572064_8. (Mind *i*, mind *v*[*i*] számozását 0-tól kezdjük.)



A nyolc vezér probléma két megoldása: [0,4,7,5,2,6,1,3] és [1,3,5,7,2,0,6,4].
A feladatnak mintegy 92 megoldása van.

1. változat: Megoldás lineáris kereséssel

A fentiek alapján a feladat megoldása például a következő lehet. Vesszük egymás után a nyolcjegyű számokat a nyolcas számrendszerben, és minden esetben ellenőrizzük, hogy ütik-e egymást a vezérek. Ha nem, úgy találtunk egy megoldást, és leállíthatjuk a további keresést; de természetesen az sem kizárt, hogy nem találunk ilyen elrendezést. (Lineáris keresés.)
Célszerű a sakktábla méretét pl. egy *n* konstansban tárolni, így a feladatot könnyen általánosíthatjuk. (Megjegyzés: 8×8-as sakktábla esetén a feladatnak 92 db megoldása van.)
Kihasználva azt, hogy van megoldás, a lineáris keresés tételének alkalmazásával a következő algoritmust kapjuk:

```
Változók:
   v: Tömb(0..n-1: Egész)
   üti: Logikai
Nyolcvezér
   v = [0 0 0…]
   üti = igaz
   Ciklus amíg üti
      Növel()
      üti = Ütie()
   Ciklus vége
   Ki: v
Eljárás vége
```

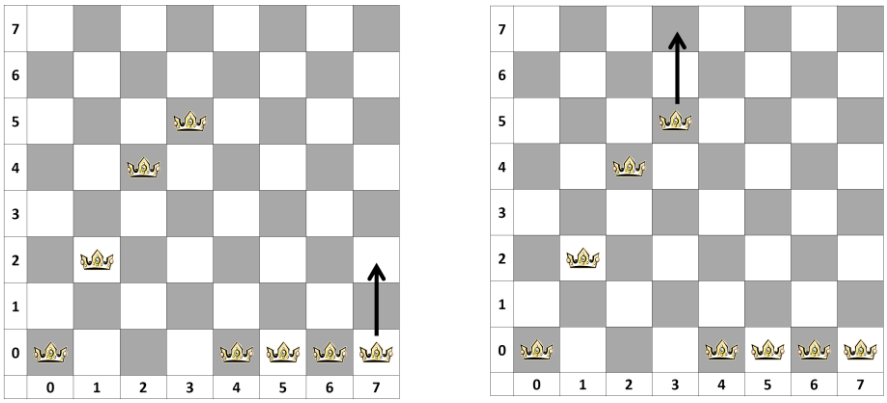
[*] *Forrás:* Farkas Csaba: A programozás alapjai Visual Basic .NET-ben, Jedlik Oktatási Stúdió, 2009 p. 212-216

A *Növel()* eljárás az *n* jegyű *v* „szám” értékét 1-gyel növeli, azaz a *v*-hez „hozzáadja” az [0, 0, 0, 0, 0, 0, 0, 1] „számot”. Például, ha *v* értéke [1, 2, 3, 4, 1, 2, 3, **4**], akkor, ha 1-gyel növeljük, az [1, 2, 3, 4, 1, 2, 3, **5**] „számot” kapjuk. Az eljárás megírása során nehézséget csak az okoz, hogy gyakran átvitel is van. Például, ha *v* értéke [1, 2, 3, 4, 1, 2, **3, 7**], akkor a „számot” 1-gyel növelve az [1, 2, 3, 4, 1, 2, **4, 0**] adódik. Átvitel több helyiértéken is lehet, például [1, **2, 7, 7, 7, 7, 7, 7**] esetén növeléskor az [1, **3, 0, 0, 0, 0, 0, 0**] „számot” kapjuk.

A *Ütie()* függvény azt ellenőrzi, hogy van-e két olyan vezér, amely üti egymást, tehát ezt az eljárást is megírhatjuk lineáris keresésként. Haladunk pl. balról jobbra, és minden esetben megnézzük, hogy az adott oszlopban lévő vezér üti-e valamelyik korábbi oszlopban lévőt. Az eljárás vagy akkor ér véget, amikor már nincs több vizsgálandó vezér, vagy akkor, ha találtunk ütésben lévő párt.

2. változat: A megoldás gyorsítása visszalépéssel

Az algoritmus gyorsítását a következő felismerés teszi lehetővé. Az alábbi ábra szerinti elrendezésben a program éppen a 7. oszlopban lévő vezért lépteti. Amikor az végigért, akkor lépteti egyet előre a 6. oszlopban lévőt, majd újra a 7. oszlopban lévőt viszi végig stb. Látható, hogy az algoritmus sok fölösleges lépés után jut el a ténylegesen rossz helyen álló 3. oszlopban lévő vezérig. A 4..7 oszlopban lévő vezéreknek lényegében nincs is szerepük.



Az algoritmus ebben az állásban a 7. oszlopban álló vezért lépteti, pedig a 3. oszlopban álló van rossz helyen.

Ne tegyük fel kezdetben az összes vezért!
Először csak az első vezért tegyük fel, az nyilván nem lesz ütésben más vezérrel.
Tegyük fel, hogy már sikeresen elhelyeztünk *k* vezért a 0..*(k-1)* oszlopokba! Vegyük fel a következő vezért a *k*. oszlopba, és léptessük mindaddig, amíg ütésben van! Ha végig értünk az oszlopon, és nem találtuk meg a helyét, akkor vegyük le, és léptessük a *(k-1)*. oszlopban lévőt!

Az eljárás C# nyelvű kódja a következő:

Az eljárásban a vizsgált oszlopot nem *k*, hanem az *oszlop* változó adja meg
Azt, hogy az *i*-edik oszlopban még nincs feltéve a vezér a *v*[*i*]=–1 érték jelzi

```
static void BackTrack(int[] v, ref int oszlop, int n)
{
    while (0 <= oszlop && oszlop < n)
    {
        bool uti = true;
        //felléptеti az adott oszlopba a vezért
        //és addig lépteti, amíg ütésben van
        //lényegében lineáris keresés az oszlop soraira
        while (v[oszlop] < n && uti)
        {
            v[oszlop] = v[oszlop] + 1;
```

```
            uti = Utie(v, oszlop);
        }
        //ha talált helyet, továbblép a következő oszlopra
        //ha nem, leveszi a vezért és visszalép az előzőre
        if (v[oszlop] < n)
            oszlop++;
        else
        {
            v[oszlop] = -1;
            oszlop--;
        }
    }
}
```

Az *Ütie()* függvény megvalósításánál elegendő a vizsgált oszloppal való ütközést ellenőrizni, mert az az előttiek már rendben vannak:

```
static bool Utie(int[] v, int oszlop)
{
    //keressük az első olyan oszlopot, amellyel ütésben van
    int i = 0;
    while (i < oszlop && !(v[i] == v[oszlop] ||
             Math.Abs(v[i] - v[oszlop]) == Math.Abs(i - oszlop)))
        i++;
    //ha i<oszlop, akkor találtunk
    return (i < oszlop);
}
```