Bevezetés a programozásba

12. Előadás: 8 királynő

A 8 királynő feladat

- Egy sakktáblára tennénk 8 királynőt, úgy, hogy ne álljon egyik sem ütésben
- Ez nem triviális feladat, a lehetséges 64*63*62*61*60*59*58*57/8!=4'426'165'368 esetből csak 92 jó
- Először tehát gondolkodni kell, hogyan lehetne elkerülni a felesleges számolást, és felesleges memóriahasználatot

A 8 királynő feladat

- Első lépés: helyes reprezentáció választása
- Emlékeztető: a jó reprezentáció minden lehetséges esetet (most sakktábla-állást) lehetővé tesz, de többet lehetőleg nem
- A naiv reprezentáció: egy 8x8 mátrix, ahol 0 jelenti, hogy nincs királynő, és 1 jelenti, hogy van
- Ennél van jobb reprezentáció

A 8 királynő feladat

- Használjuk ki, hogy tudjuk, hogy ha két királynő van egy oszlopban, az biztosan rossz
- Tehát egy oszlopban csak egy királynőt kell tudni reprezentálni!
- Ez pedig könnyű: egy számmal, hogy hányadik sorban áll
- Az állás reprezentálása tehát egészek vektora
- Mellesleg ~~4'426'165'368~~ 16'777'216!

Királynő ütésszabály

- Két egész koordinátával megadott királynő pozícióról eldönthető, hogy egymást ütik-e:

```
bool kiralyno(int x1, int y1, int x2, int y2) {
    return x1==x2 || y1==y2
     || x1+y2==x2+y1
     || x1+y1==x2+y2;
}
```

Sakktábla ütésszabály

- Két egymásba ágyazott lineáris keresést alkalmazunk (elhanyagolva a „hol”-t)

```
bool utesben(vector<int>& v) {
    int s=v.size();
    bool res=false;
    for (int i=0;i<s && !res;i++) {
        for (int j=i+1;j<s && !res;j++) {
            res = kiralyno(i,v[i],j,v[j]);
        }
    }
    return res;
}
```

A „brute force” algoritmus

- Keresési feladatoknál
- Az elv: nézd meg az összes lehetséges megoldást
- Előnye: nem kell gondolkodni
- Hátránya: lassú. Legtöbbször használhatatlanul
- „*when in doubt use brute force*” Ken Thompson
- Összefoglalva: érdemes ezzel kezdeni
  - Az optimalizált programot legyen mivel hasonlítani
  - Ha kell egyáltalán optimalizálni

Brute force

```
int main() {
  vector<int> v(8);
  for (int i1=0;i1<8;i1++) {
   for (int i2=0;i2<8;i2++) {
    … … … … … … … … … … … … … …
    for (int i8=0;i8<8;i8++) {
        v[0]=i1;v[1]=i2;v[2]=i3;v[3]=i4;
        v[4]=i5;v[5]=i6;v[6]=i7;v[7]=i8;
        if (!utesben(v)) {
            talalat(v);
        }
    }
    … … … … … … … … … … … … … …
   }
  }
}
```

Brute force eredmények

```
1299852 lepesbol: 0 4 7 5 2 6 1 3
1551565 lepesbol: 0 5 7 2 6 3 1 4
1695331 lepesbol: 0 6 3 5 7 1 4 2
 … … … … … … … …
15081886 lepesbol: 7 1 4 2 0 6 3 5
15225652 lepesbol: 7 2 0 5 1 4 6 3
15477365 lepesbol: 7 3 0 2 5 1 6 4
```

- Nincs kecmec, a lehetséges 16 millió esetből kiválogattuk a megfelelőket
- Kicsit lassú

Hogyan lehet gyorsabb algoritmust csinálni?

- Főtétel : „Nincs ingyen leves”
- Az általánosság és a hatékonyság ellentétes:
  - Ha egy algoritmus semmit sem használ ki a feladatból, a hatékonysága megegyezik a véletlen találgatással
- Ilyenkor gondolkodni kell
  - keresni a problémának olyan vonását, amely kihasználható

8 királynő

- Mit nem használtunk ki eddig?
- A brute force megoldás egy olyan esetben, ahol i1 és i2 már ütik egymást, feleslegesen néz meg 8^6 esetet
- Más szavakkal: abból, hogy két bábu az első n bábu közül már ütésben van, tudható, hogy tetszőleges n-en túli elrendezés sem lesz jó
- Hogyan használjuk ezt ki?

Rekurzióval

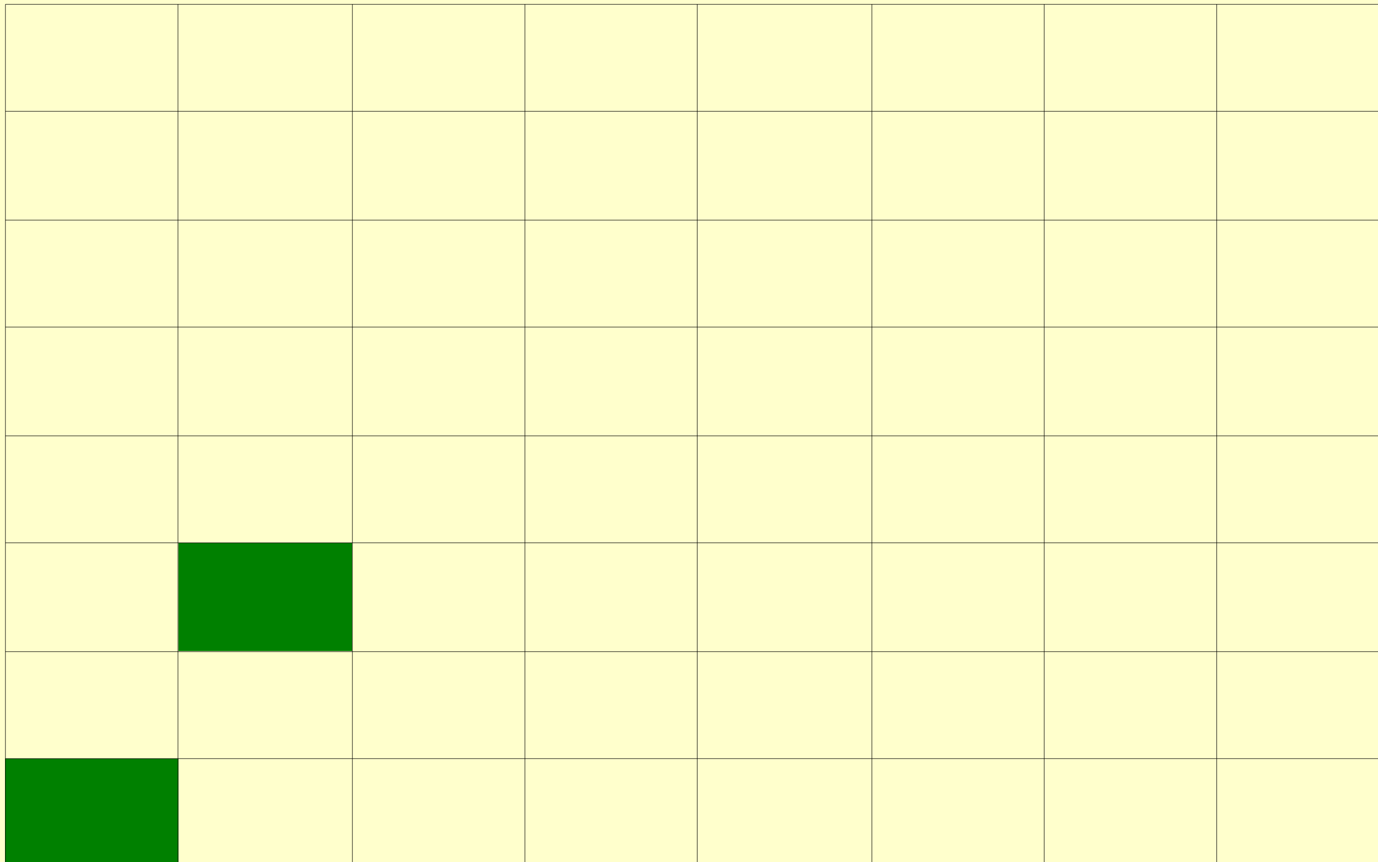
```
kereső(állás, mélység, méret) {
    HA állás ütésben van, return

    //eddig jó
    HA mélység = méret, találat(állás)

    //Még nem elég hosszú, de eddig jó
    bővítsük az állást
    CIKLUS i a helyekre a mélység-edik oszlopban
        állás[mélység]=i;
        kereső(állás,mélység+1,méret);
    }
}
```

8 királynő

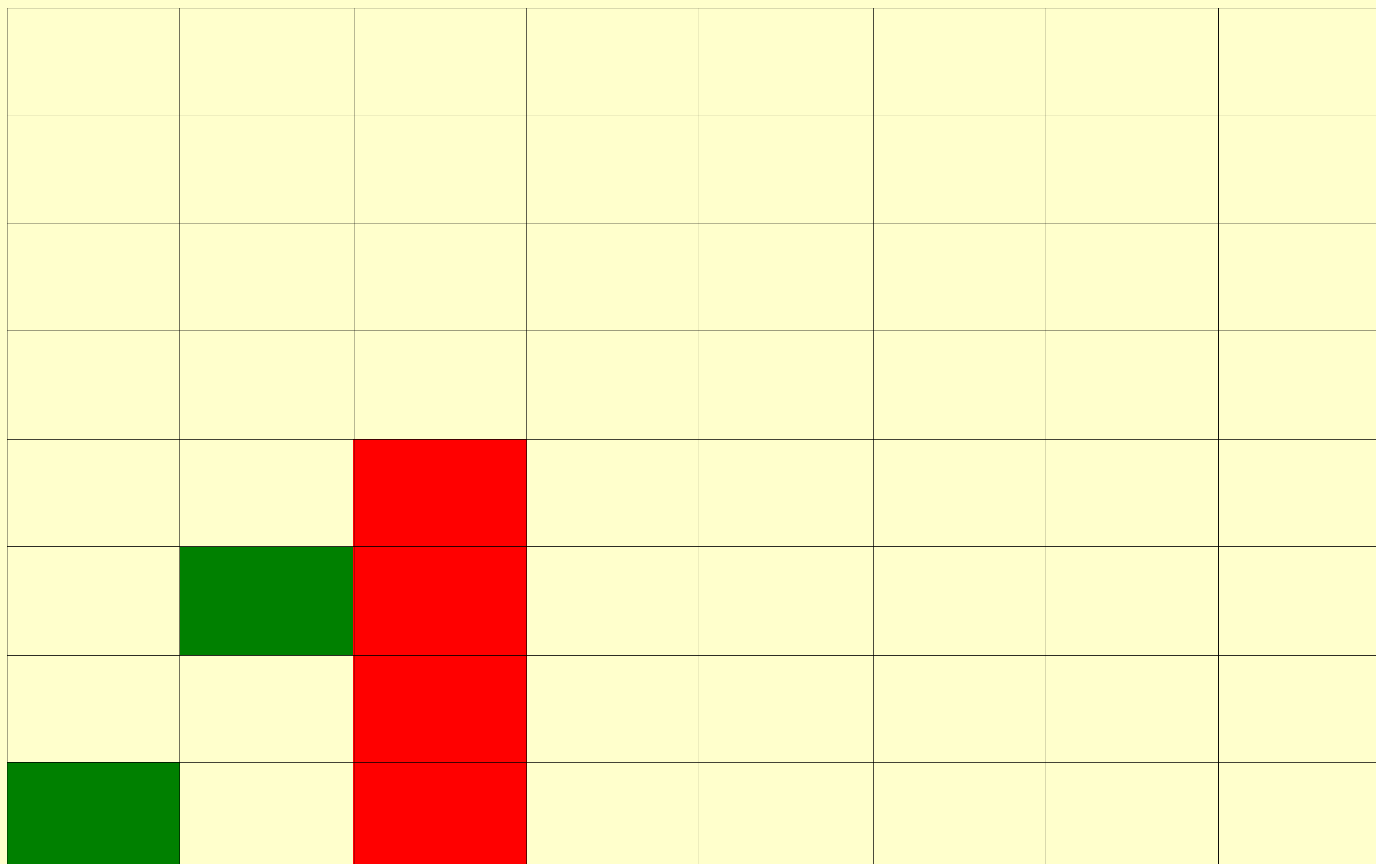
8 királynő

8 királynő

8 királynő

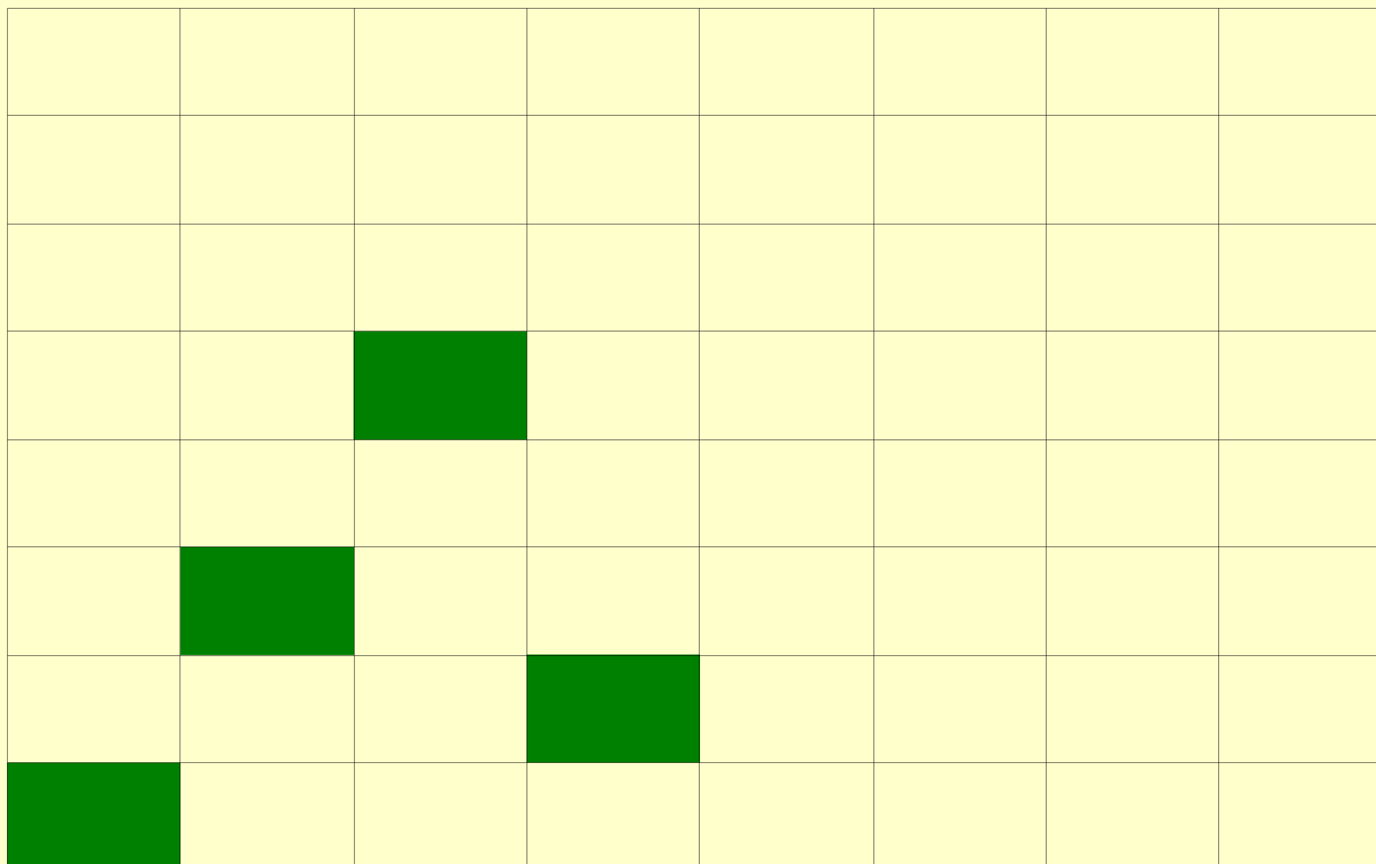


8 királynő



8 királynő

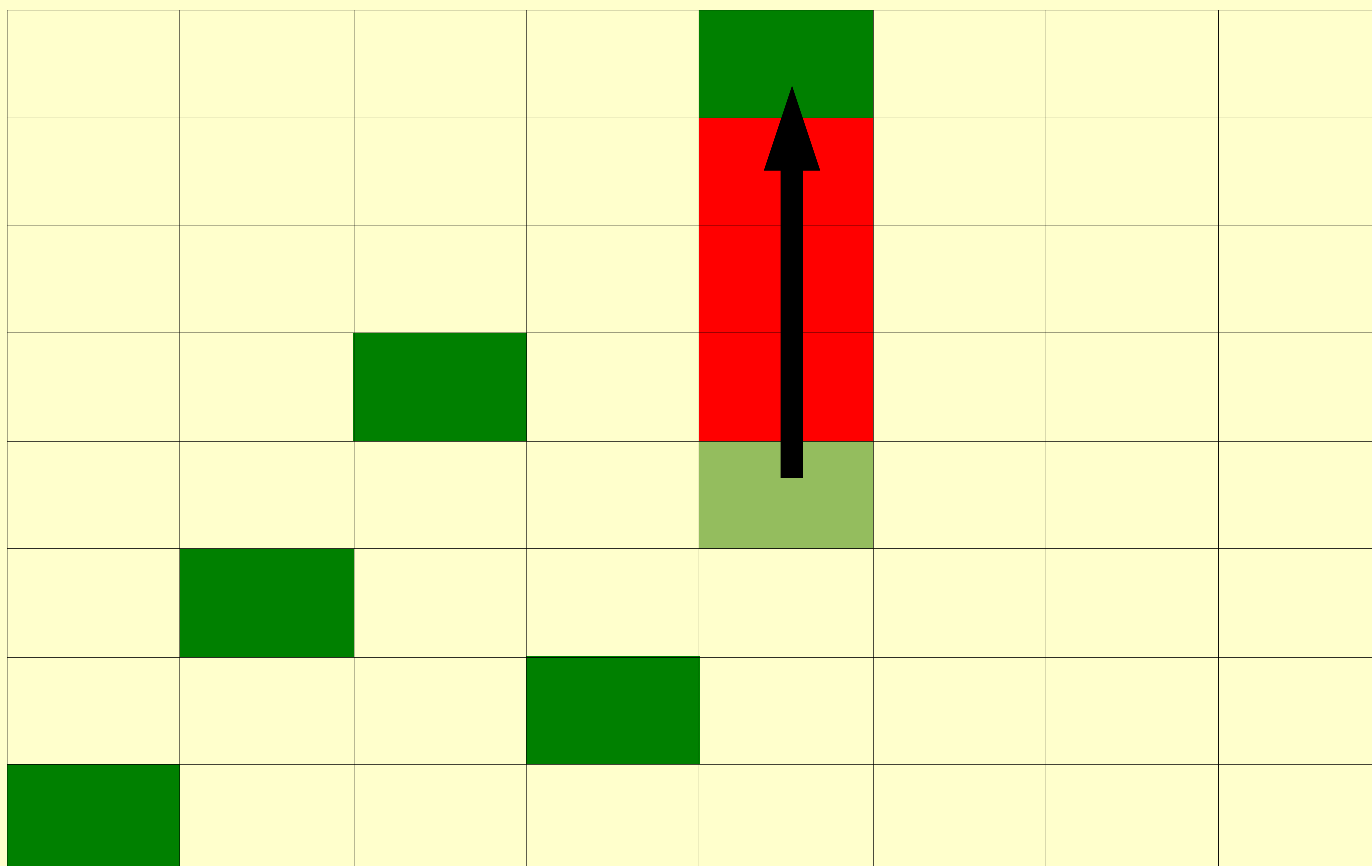
8 királynő



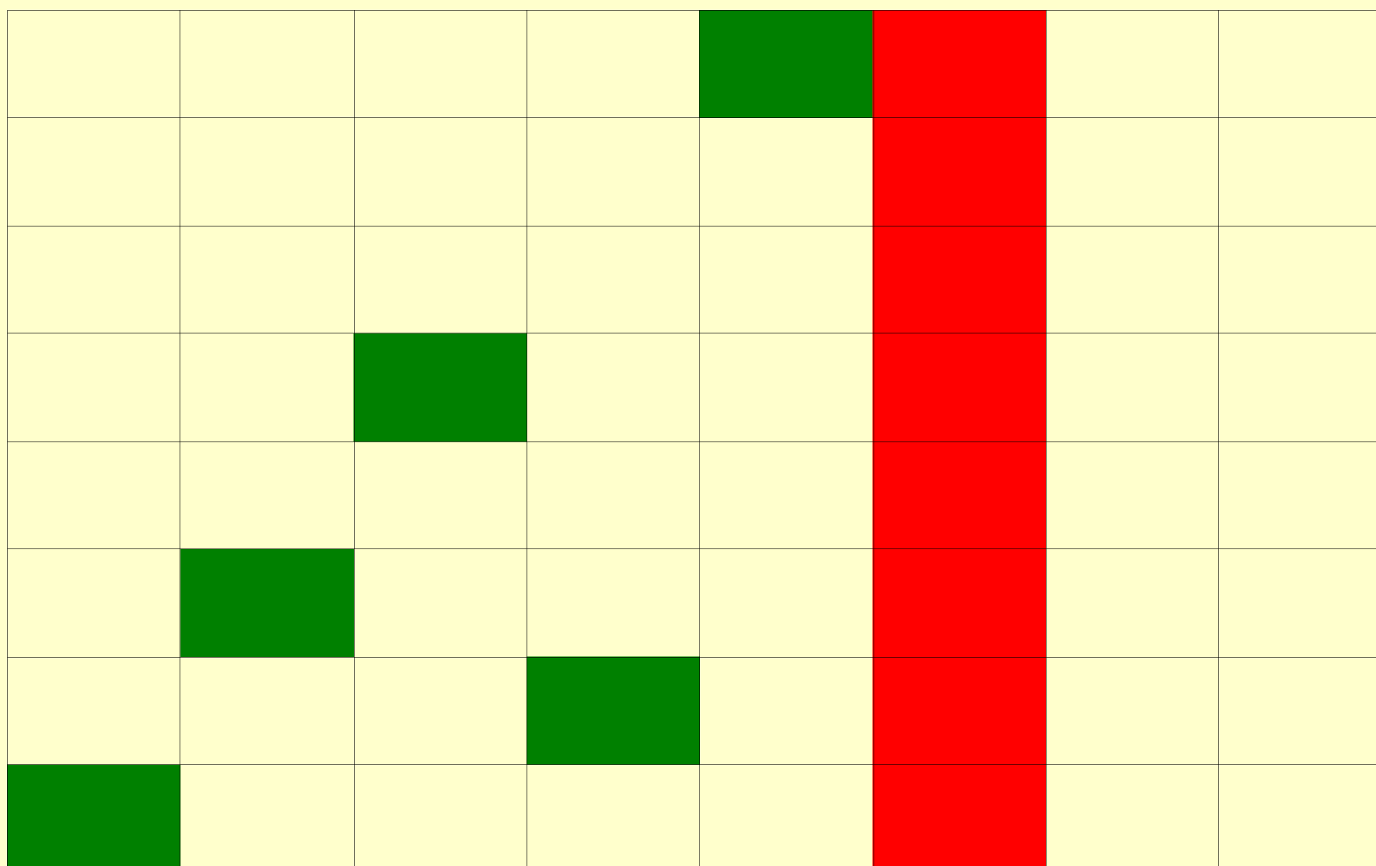
8 királynő

8 királynő

8 királynő

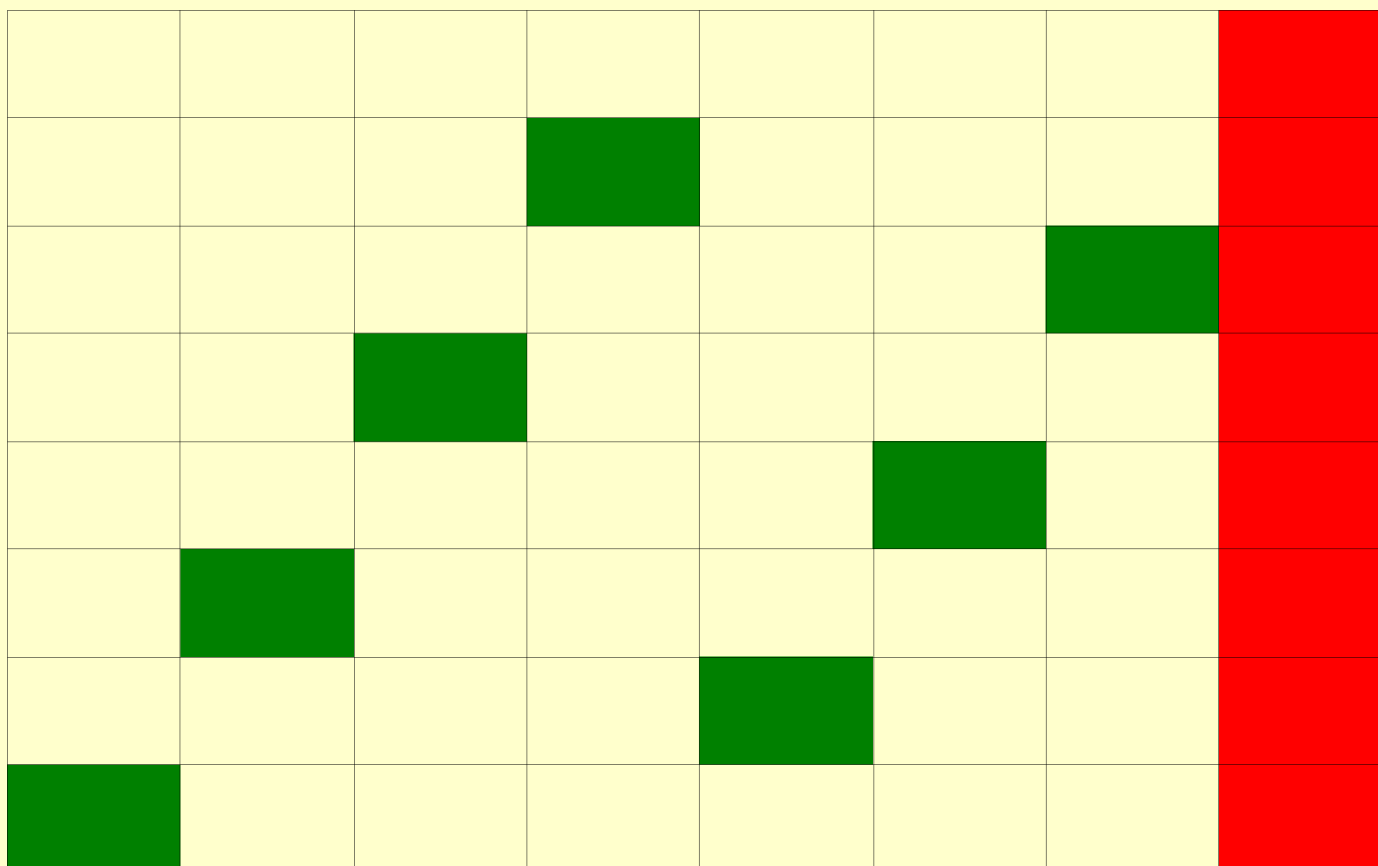


8 királynő

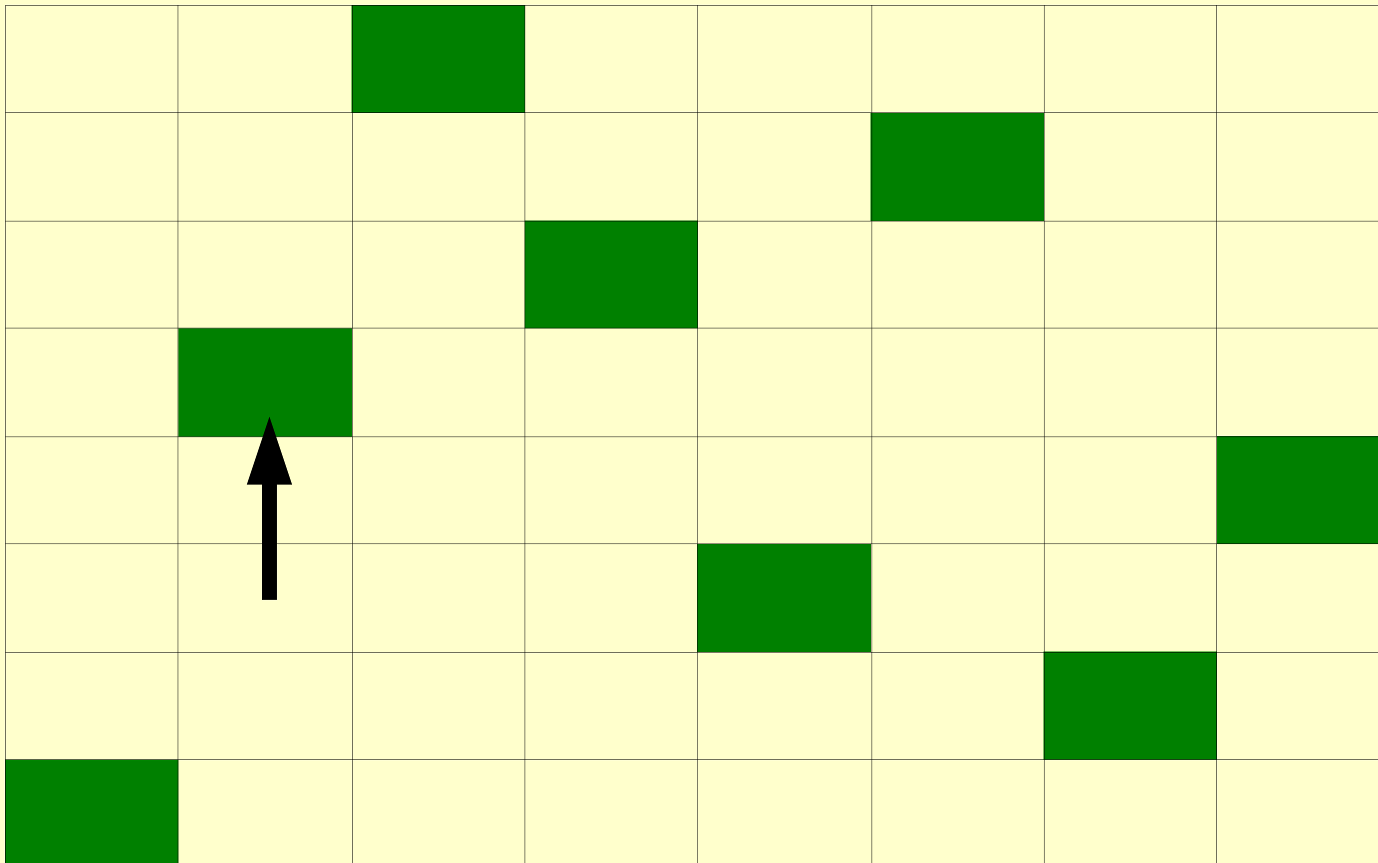


8 királynő

8 királynő



8 királynő : az első megoldás



8 királynő: visszalépéses keresés

```
void keres(vector<int> v, int a, int kir) {
    if (utesben(v)) return;
    if (v.size()==kir) {
        talalat(v);
    }
    //Még nem elég hosszú, de eddig jó
    v.push_back(0);
    for (int i=0;i<kir;i++) {
        v[a]=i;
        keres(v,a+1,kir);
    }
}
```

- Érték szerinti paraméterátadás: nincs „pop_back()”

Hatékonyság

```
1299852 lepesbol: 0 4 7 5 2 6 1 3
1551565 lepesbol: 0 5 7 2 6 3 1 4
1695331 lepesbol: 0 6 3 5 7 1 4 2
 … … … … … … … …
15081886 lepesbol: 7 1 4 2 0 6 3 5
15225652 lepesbol: 7 2 0 5 1 4 6 3
15477365 lepe
```

```
877 lepesbol: 0 4 7 5 2 6 1 3
1149 lepesbol: 0 5 7 2 6 3 1 4
1357 lepesbol: 0 6 3 5 7 1 4 2
 ... ... ...
15101 lepesbol: 7 1 4 2 0 6 3 5
15309 lepesbol: 7 2 0 5 1 4 6 3
15581 lepesbol: 7 3 0 2 5 1 6 4
```

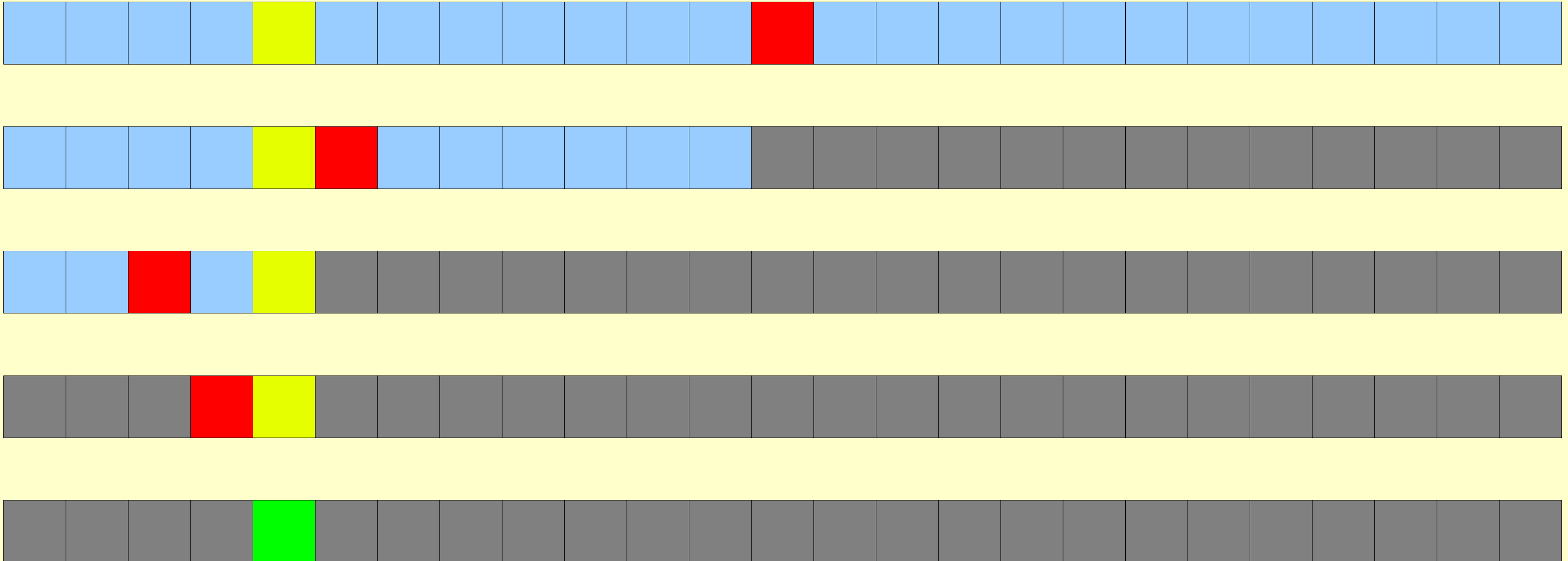
Hatékonyság

- Láttuk, hogy van különbség a brute force és a visszalépéses keresés között
- Az előny körülbelül ezerszeres volt ebben a példában
- Ez sok?
- Sok, de nem valódi előny:
  - 8x8 tábla helyett n x n táblán a futás ideje mindkét algoritmusnak körülbelül exponenciális

Bináris keresés

- A feladat ugyanaz, mint a lineáris keresésben: adott elemet keresünk sok elem között
- De előfeltétel: az elemek vektorban vannak, és rendezett állapotban
- Így az intervallum felezésével minden lépésben felezzük a maradék megnézendő elemeket, tehát $\log_2(n)$ lépésben végzünk
- Ez általános értelemben is gyorsabb, mint a lineáris keresés

Bináris keresés



24 elem, 5 lépés

$2^5 = 32$

$\log_2(n)$ felfelé kerekítve

Hatékonyság elemzés

- Eszköz: „Ordo”
- Definíció: az algoritmus, aminek a bemenete p_1, p_2, ..., p_n paraméterektől függ, $O(f(p_1, p_2, .., p_n))$ akkor és csak akkor, ha létezik F, hogy *futásidő* $< F*f(p_1, p_2, ..., p_n)$ bármely elegendően nagy p_1, p_2, ..., p_n-re
- Példa: n elemű sorozat összegzése O(n)
- n elemű sorozat buborékrendezése $O(n^2)$
- „Legfeljebb konstansszorosa”

Hatékonyság

- Jellegzetes O(x) kategóriák
  - O(1) : konstans idejű művelet. Ilyen az értékadás, vektor egy elemének kiválasztása, fix hosszú vektoron végzett tetszőleges művelet
  - O(log(n)) : logaritmikus idejű művelet, például a bináris keresés
  - O(n): lineáris idejű művelet, az összes tétel ilyen a bemeneten kapott sorozat szerint
  - O(n*log(n)): a gyors rendezőalgoritmusok ilyenek
  - $O(n^2)$: lassú rendezőalgoritmusok
  - $O(2^n)$: exponenciális idejű műveletek

Kitekintés

- „brute force” a biztonságtechnikában
- Párhuzamos architektúrákon érdemes megkülönböztetni azokat az eseteket, ha „n” egy nagyságrendbe esik a feldolgozóegységek számával
  - O(n) -> O(1), ha van n processzor, és függetlenül kezelhetőek
  - A valóság ennél sajnos bonyolultabb
- A jó algoritmushoz jó adatszerkezetre is szükség lesz