Backtracking

wijanarto

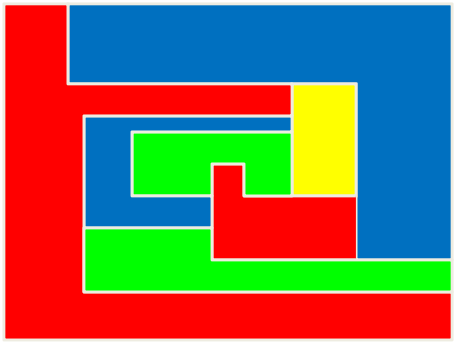
backtracking

- Misal anda harus membuat beberapa *keputusan* diantara banyak *pilihan,* Dimana :
  - Anda tidak memiliki informasi untuk mengeahui apa yang harus di pilih
  - Tiap keputusan, memiliki himpunan pilihan lagi
  - Beberapa urutan pilihan (mungkin lebih dari satu) mungkin merupakan soulusi bagi masalah anda
- Backtracking merupakan cara secara metodologis untuk menemukan beberapa urutan hingga kita menemukan yang tepat .

Solving Maze
(dibahas pada slide lainnya)

- Diberikan sebuah maze, cari jalur dari awal hingga akhir
- Tiap interseksi, anda harus menentukan antara 3 atau kurang ilihan :
  - Terus
  - Ke kiri
  - Ke kanan
- Anda tidak punya cukup informasi untuk memilih secara benar
- Tiap pilihan mempunyai himpunan pilihan lainnya
- Satu atau lebih urutan pilihan mungkin merupakan solusi bagi anda

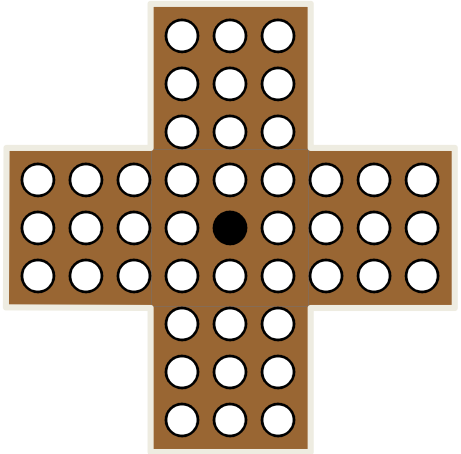
Coloring a map



- You wish to color a map with not more than four colors
  - red, yellow, green, blue
- Adjacent countries must be in different colors
- You don’t have enough information to choose colors
- Each choice leads to another set of choices
- One or more sequences of choices may (or may not) lead to a solution
- Many coloring problems can be solved with backtracking

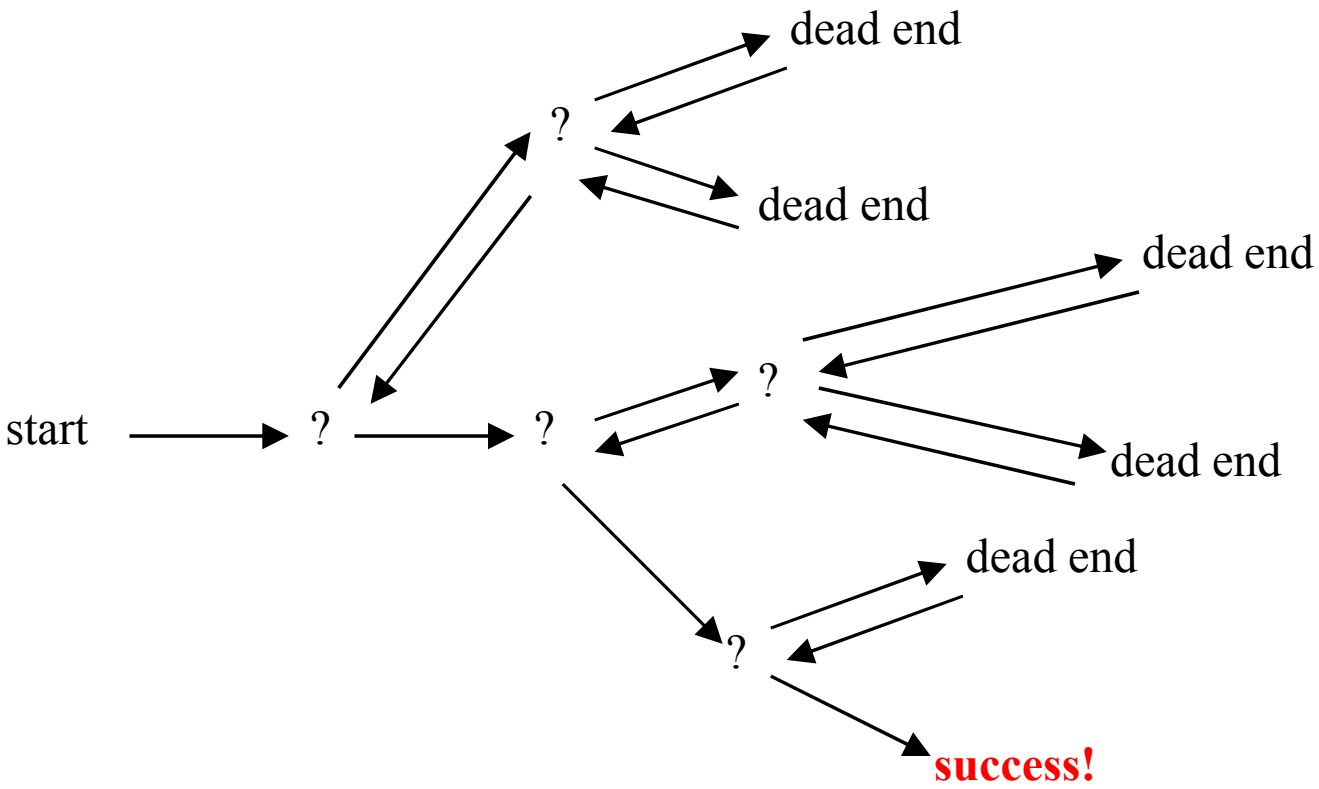
Solving a puzzle

- In this puzzle, all holes but one are filled with white pegs
- You can jump over one peg with another
- Jumped pegs are removed
- The object is to remove all but the last peg



- You don’t have enough information to jump correctly
- Each choice leads to another set of choices
- One or more sequences of choices may (or may not) lead to a solution
- Many kinds of puzzle can be solved with backtracking

Animasi backtracking



Algoritma Backtracking

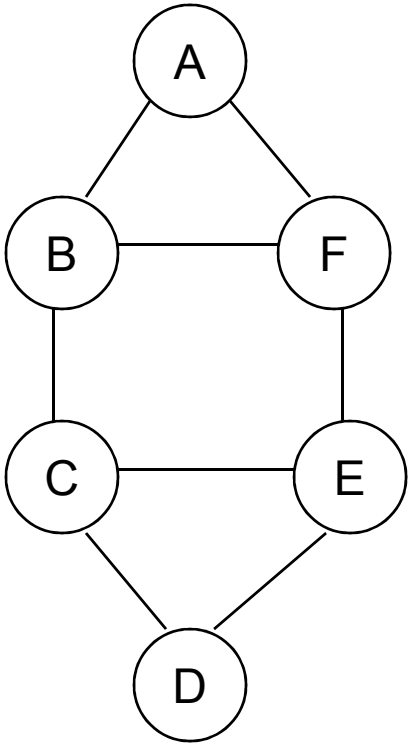
- Ide : “explore” tiap node :
- Untuk meng-”explore” node N:
    1. If N = goal node, return “success”
    2. If N = leaf node, return “failure”
    3. For anak C dalam N,
        3.1. Explore C
            3.1.1. If C = success, return “success”
    4. Return “failure”

Map Coloring

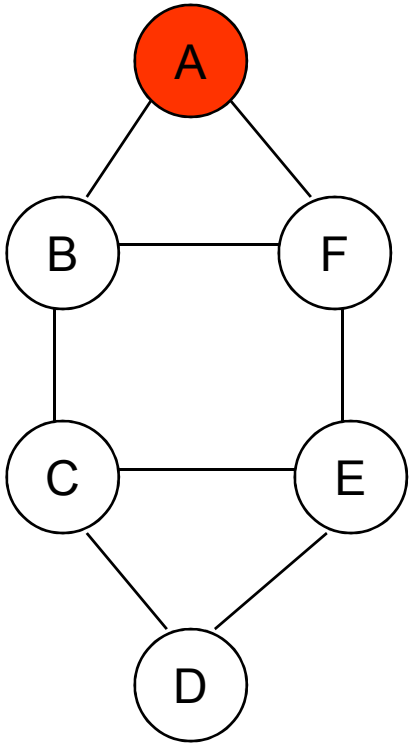
- Dalam Four Color Theorem dinyatakan bahwa tiap map pada suatu plane dapat diwarnai dengan tidak lebih dari 4 warna, jadi tidak ada 2 negara dengan batas negara yang berwarna sama
- Dalam map cari pewarnaan yang mudah
- Dalam map, dapat sulit untuk menemukan warna
- Kita akan konstruksikan contoh ini dalam java

Graph Coloring

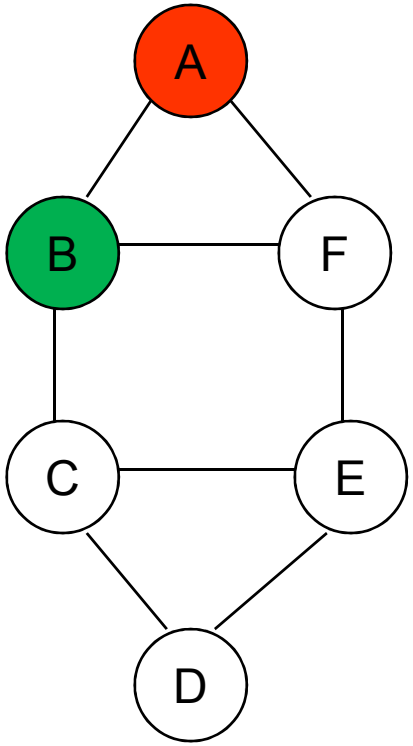
- Contoh:
  - Urutan vertek A-F
  - Warna yang diberikan adalah : R, G, B



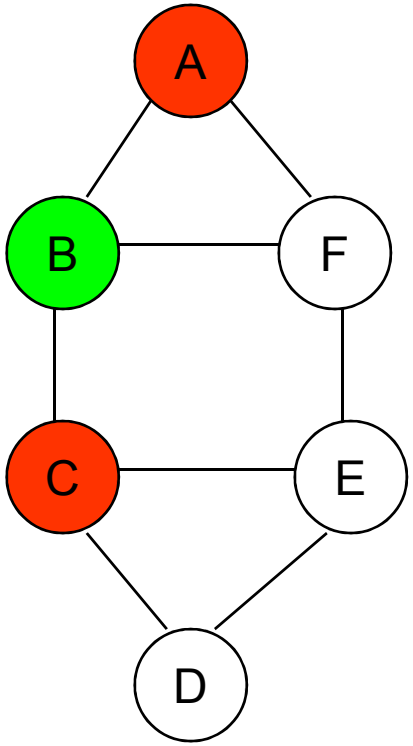
Graph Coloring



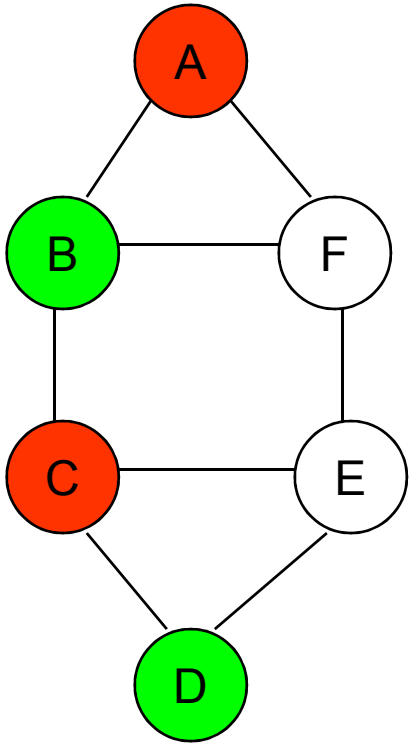
Graph Coloring



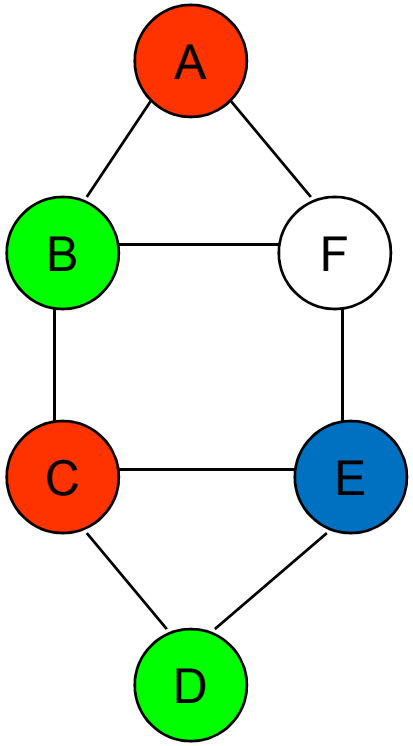
Graph Coloring



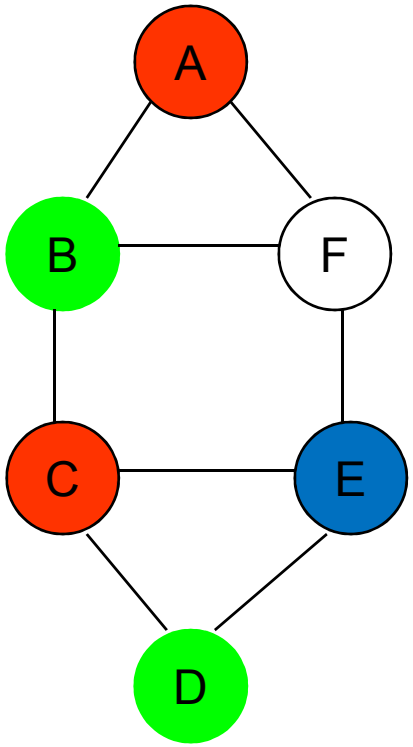
Graph Coloring



Graph Coloring

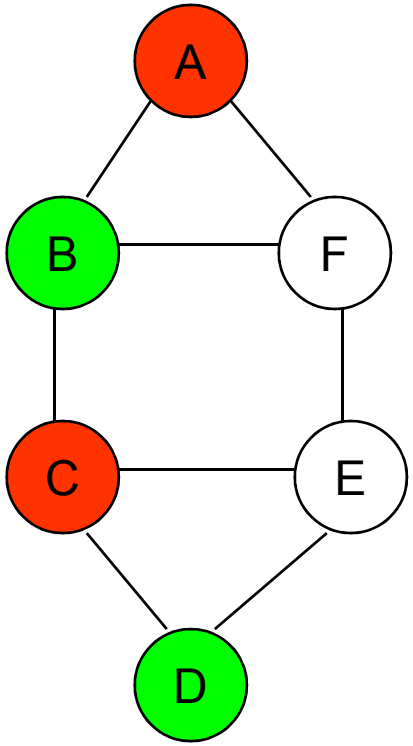


Graph Coloring

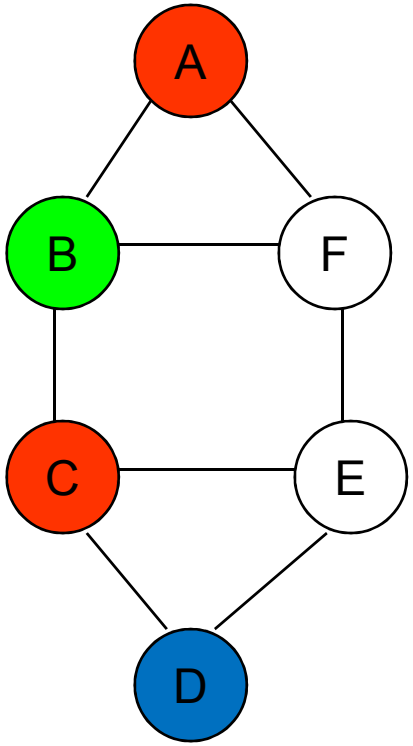


Stuck!

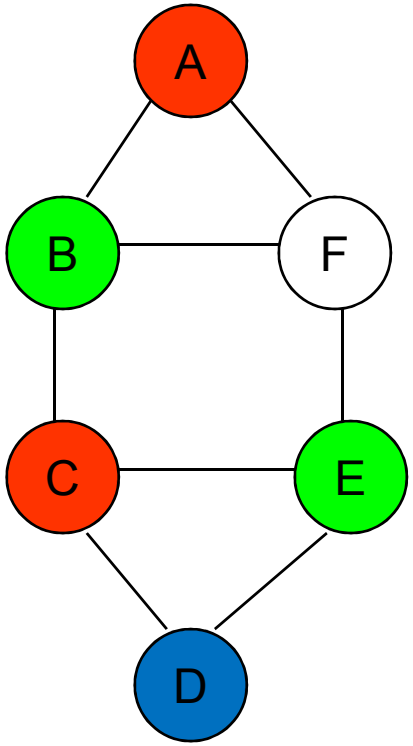
Graph Coloring



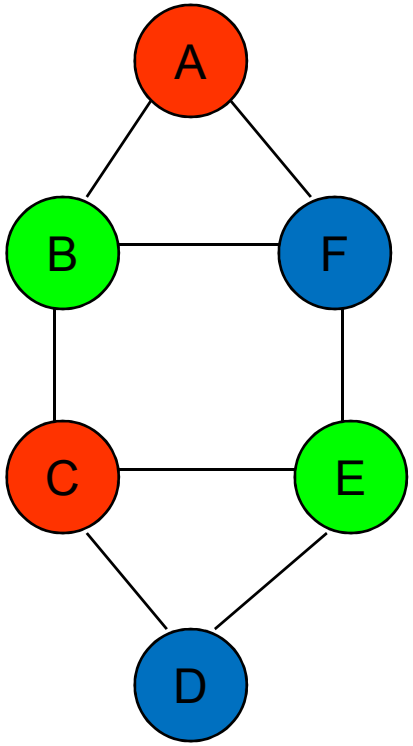
Graph Coloring



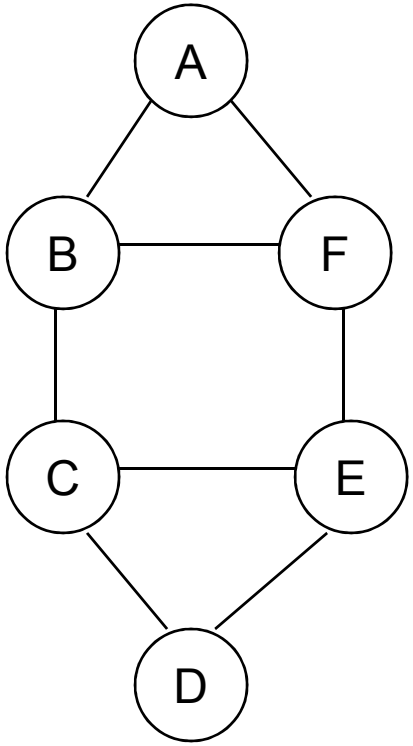
Graph Coloring



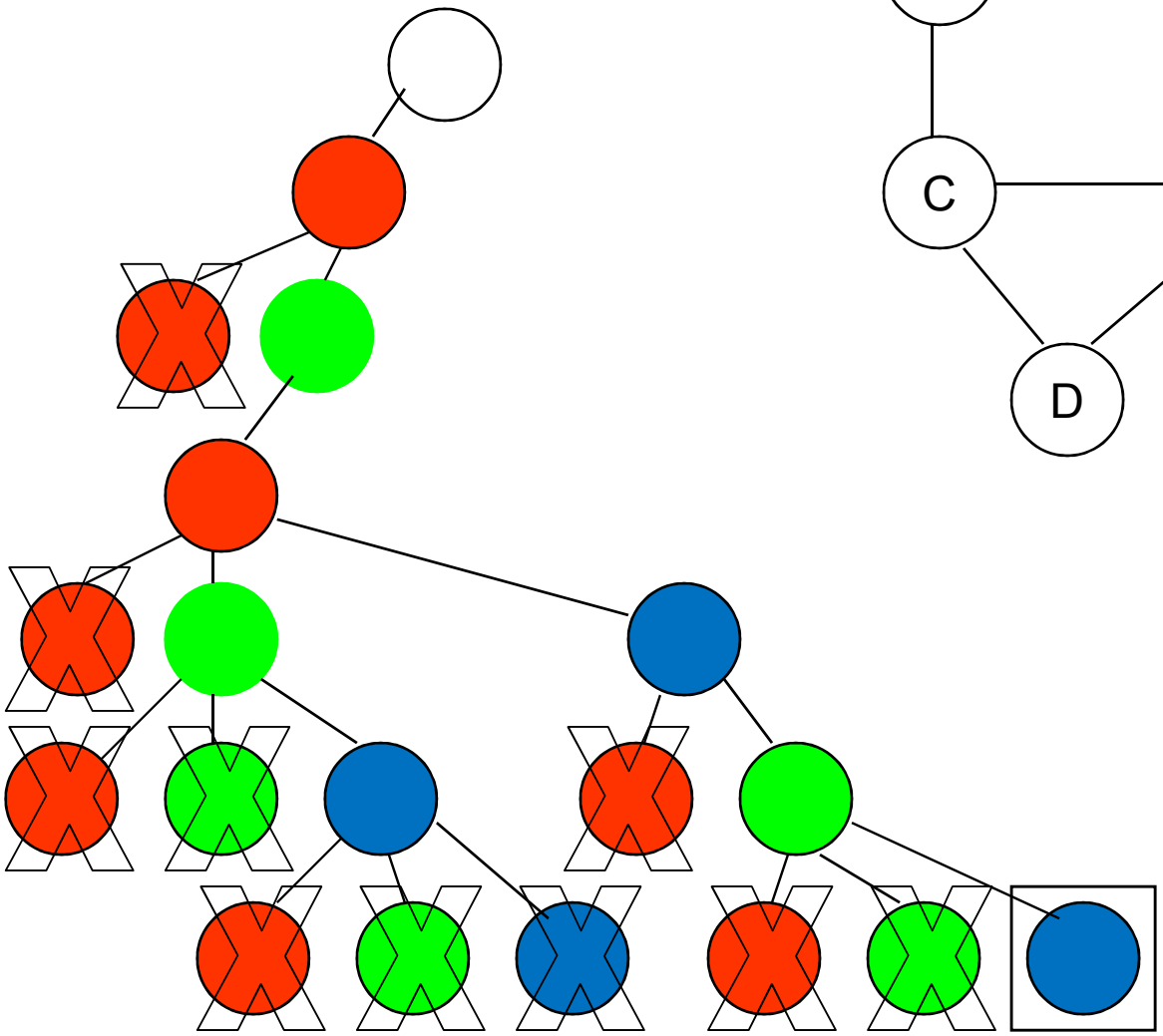
Graph Coloring



Graph Coloring



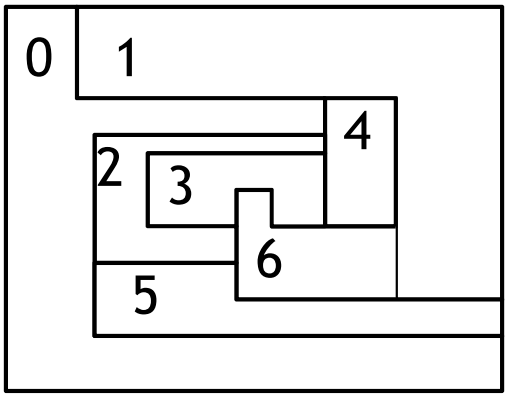
A
B
C
D
E
F



Struktur Data

- Struktur Data
  - Setting warna untuk tiap negara
  - Tiap negara, fapat menemukan semua negara tetangganya
- Lakukan dengan 2 array
  - Array “warna”, dimana WarnaNegara[i] adalah warna negara ke-i
  - Array negara tetangga, map[i][j] merupakan negara tetangga ke-j dari negara I
    - Misal: map[5][3]==8 : negara tetangga ke-3 pada negara 5 adalah negara 8

Buat Map




```
int map[][];

void createMap() {
    map = new int[7][];
    map[0] = new int[] { 1, 4, 2, 5 };
    map[1] = new int[] { 0, 4, 6, 5 };
    map[2] = new int[] { 0, 4, 3, 6, 5 };
    map[3] = new int[] { 2, 4, 6 };
    map[4] = new int[] { 0, 1, 6, 3, 2 };
    map[5] = new int[] { 2, 6, 1, 0 };
    map[6] = new int[] { 2, 3, 4, 1, 5 };
}
```

Seting Initial color

```
static final int NONE = 0;
static final int MERAH= 1;
static final int KUNING = 2;
static final int HIJAU = 3;
static final int BIRU = 4;

int mapColors[] = { NONE, NONE, NONE, NONE,
                    NONE, NONE, NONE };
```

Main

(The name of the enclosing class is ColoredMap)

```
public static void main(String args[]) {
    ColoredMap m = new ColoredMap();
    m.createMap();
    boolean result = m.explore(0, MERAH);
    System.out.println(result);
    m.printMap();
}
```

Metode backtrack

```
boolean explore(int country, int color) {
    if (country >= map.length) return true;
    if (warnai(country, color)) {
        mapColors[country] = color;
        for (int i = MERAH; i <= BIRU; i++) {
            if (explore(country + 1, i)) return true;
        }
    }
    return false;
}
```

Periksa warna yang dapat di pakai

```
boolean warnai(int country, int color) {
    for (int i = 0; i < map[country].length;
    i++) {
        int AdjCountry = map[country][i];
        if (mapColors[AdjCountry] == color) {
            return false;
        }
    }
    return true;
}
```

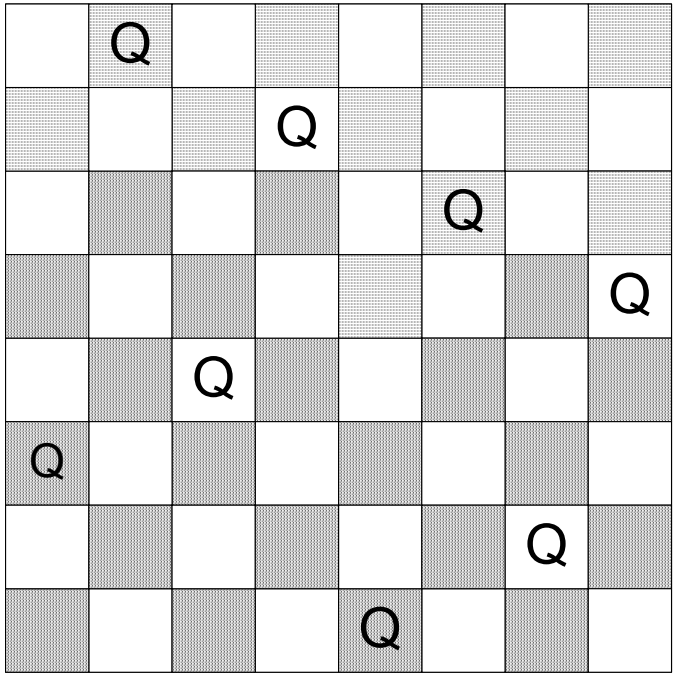
Cetak hasil

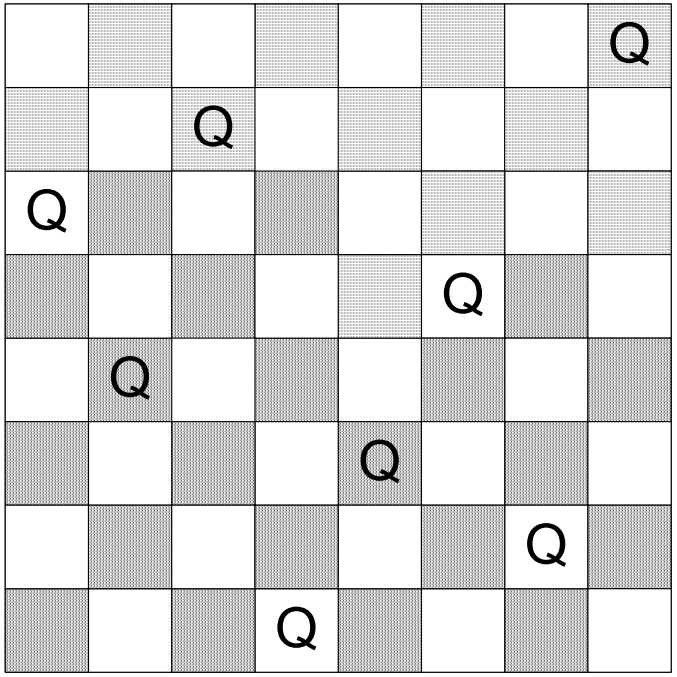
```
void printMap() {
      for (int i = 0; i < mapColors.length; i++) {
         System.out.print("map[" + i + "] is ");
         switch (mapColors[i]) {
            case NONE:    System.out.println("none");   break;
            case RED:    System.out.println(“merah");     break;
            case YELLOW: System.out.println(“kuning"); break;
            case GREEN:   System.out.println(“hijau");  break;
            case BLUE:     System.out.println("biru");     break;
         }
      }
}
```

Persoalan N-Ratu
(*The N-Queens Problem*)

- Diberikan sebuah papan catur yang berukuran $N \times N$ dan delapan buah ratu. Bagaimanakah menempatkan N buah ratu (Q) itu pada petak-petak papan catur sedemikian sehingga tidak ada dua ratu atau lebih yang terletak pada satu baris yang sama, atau pada satu kolom yang sama, atau pada satu diagonal yang sama ?

8-Ratu





Penyelesaian dengan Algoritma Brute Force

- *a) Brute Force 1*

Mencoba semua kemungkinan solusi penempatan delapan buah ratu pada petak-petak papan catur.

Ada $C(64, 8) = 4.426.165.368$ kemungkinan solusi.

- *b) Brute Force 2*

Meletakkan masing-masing ratu hanya pada baris-baris yang berbeda. Untuk setiap baris, kita coba tempatkan ratu mulai dari kolom 1, 2, …, 8.

- Jumlah kemungkinan solusi yang diperiksa berkurang menjadi $8^8 = 16.777.216$

```
procedure Ratu1
{Mencari semua solusi penempatan delapan ratu pada petak-petak papan catur
yang  berukuran 8 x 8 }
Deklarasi
  i1, i2, 13, 14, i5, i6, i7, i8 : integer
Algoritma:
  for i1←1 to 8 do
   for i2←1 to 8 do
     for i3←1 to 8 do
         for i4←1 to 8 do
            for i5←1 to 8 do
              for i6←1 to 8 do
                  for i7←1 to 8 do
                     for i1←1 to 8 do
                        if Solusi(i1, i2, i3, i4, i5, i6, i7, i8) then
                           write(i1, i2, i3, i4, i5, i6, i7, i8)
                      endif
                   endfor
               endfor
            endfor
          endfor
       endfor
    endfor
  endfor
 endfor
```

- *c) Brute Force 3 (exhaustive search)*

Misalkan solusinya dinyatakan dalam vektor 8-*tupple*:

$$X = (x_1 , x_2 , \ldots , x_8)$$

Vektor solusi merupakan permutasi dari bilangan 1 sampai 8.

Jumlah permutasi bilangan 1 sampai 8 adalah $P(1, 8) = 8! = 40.320$ buah.

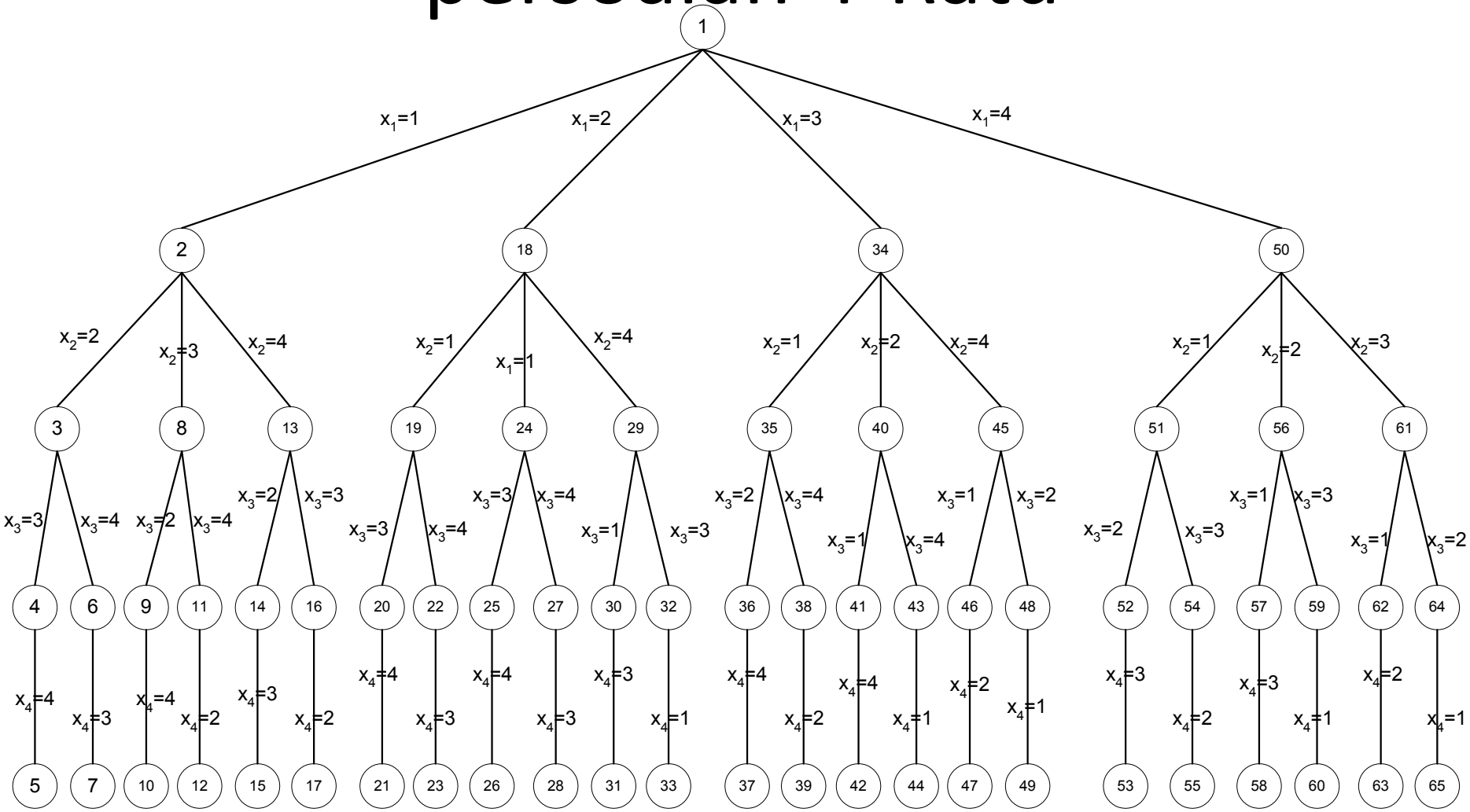
```
procedure Ratu2
{Mencari semua solusi penempatan delapan ratu pada petak-petak
papan catur yang berukuran 8 x 8 }
Deklarasi
  X : vektor_solusi
  n,i : integer
Algoritma:
  n←40320  { Jumlah permutasi (1, 2, …, 8) }
  i←1
  repeat
    X←Permutasi(8)   { Bangkitan permutasi (1, 2, …, 8) }

    { periksa apakah X merupakan solusi }
    if Solusi(X) then
      TulisSolusi(X)
    endif
    i←i+1   { ulangi untuk permutasi berikutnya }
  until i > n
```

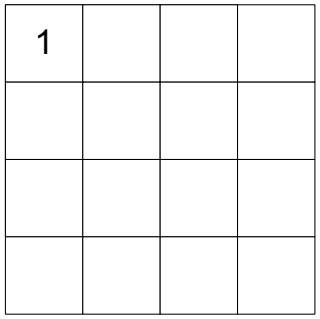
Penyelesaian dengan Algoritma Runut-balik

- Algoritma runut-balik memperbaiki algoritma *brute force* 3 (*exhaustive search*).
- Ruang solusinya adalah semua permutasi dari angka-angka 1, 2, 3, 4, 5, 6, 7, 8. Setiap permutasi dari 1, 2, 3, 4, 5, 6, 7, 8 dinyatakan dengan lintasan dari akar daun. Sisi-sisi pada pohon diberi label nilai x_i.
- Contoh: Pohon ruang-status persoalan 4-Ratu

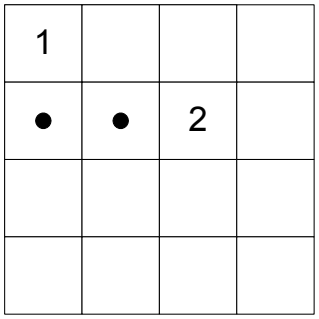
Pohon ruang status statis persoalan 4-Ratu



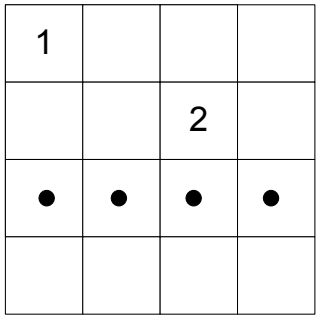
solusi runut-balik persoalan 4-Ratu



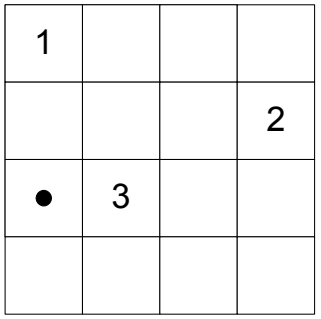
(a)



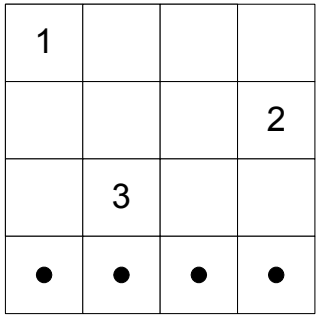
(b)



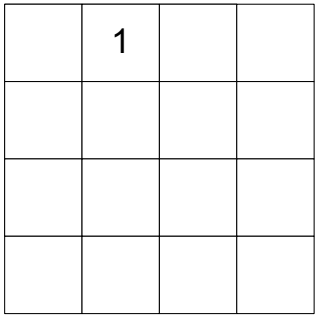
(c)



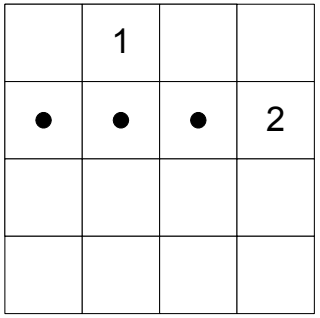
(d)



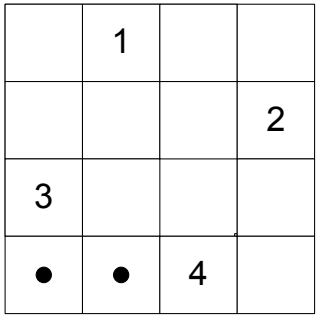
(e)



(f)

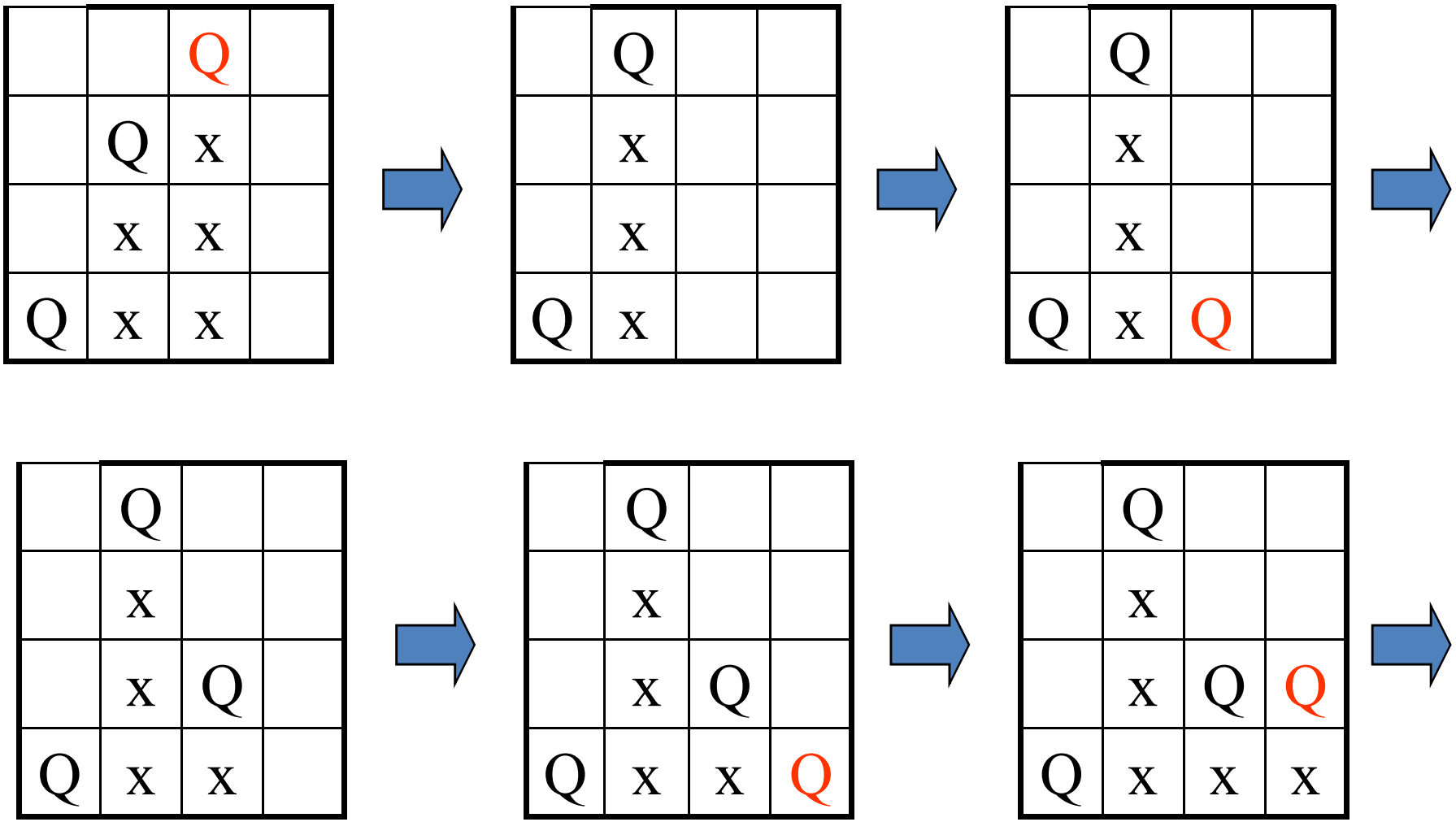


(g)

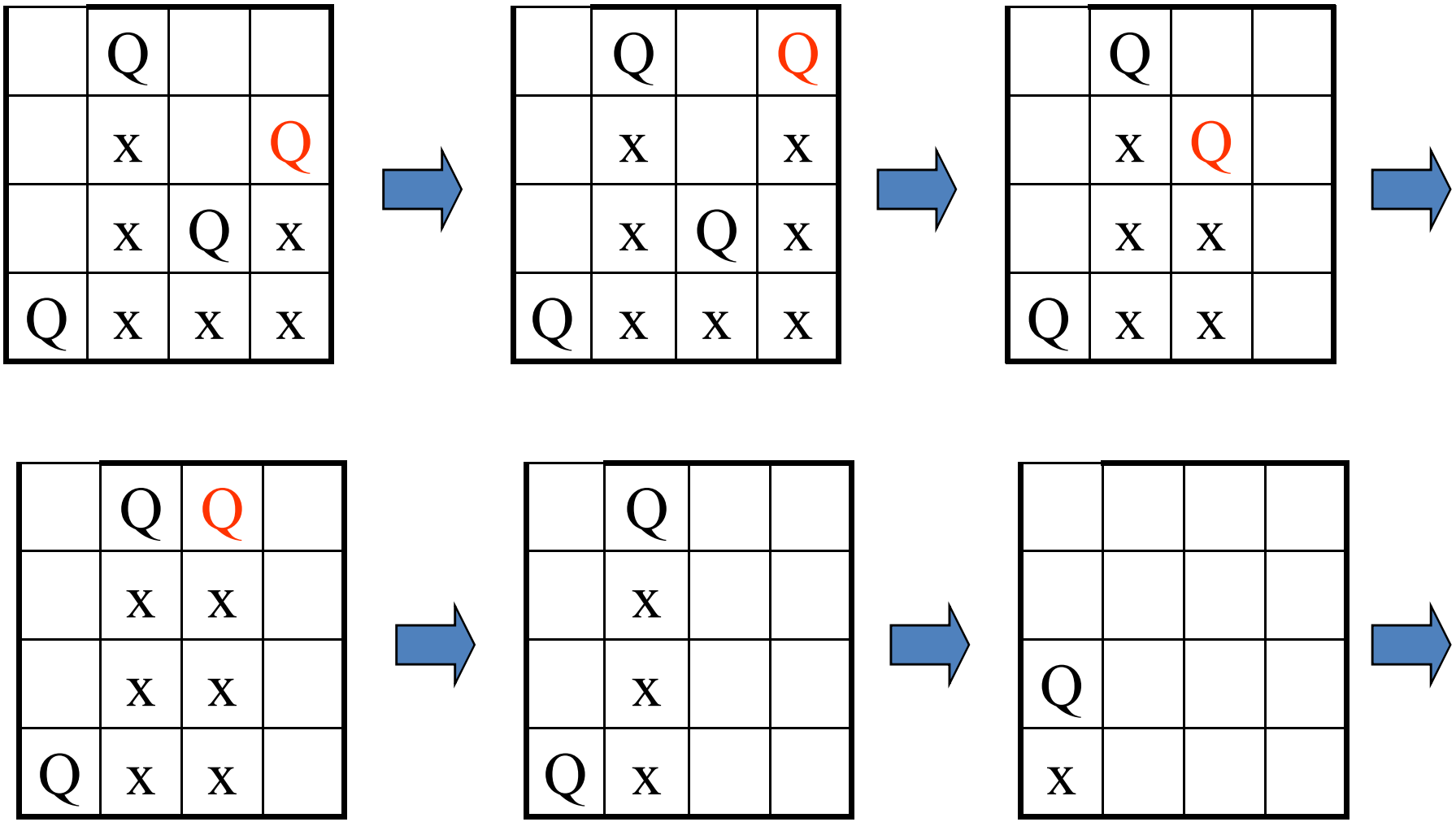


(h)

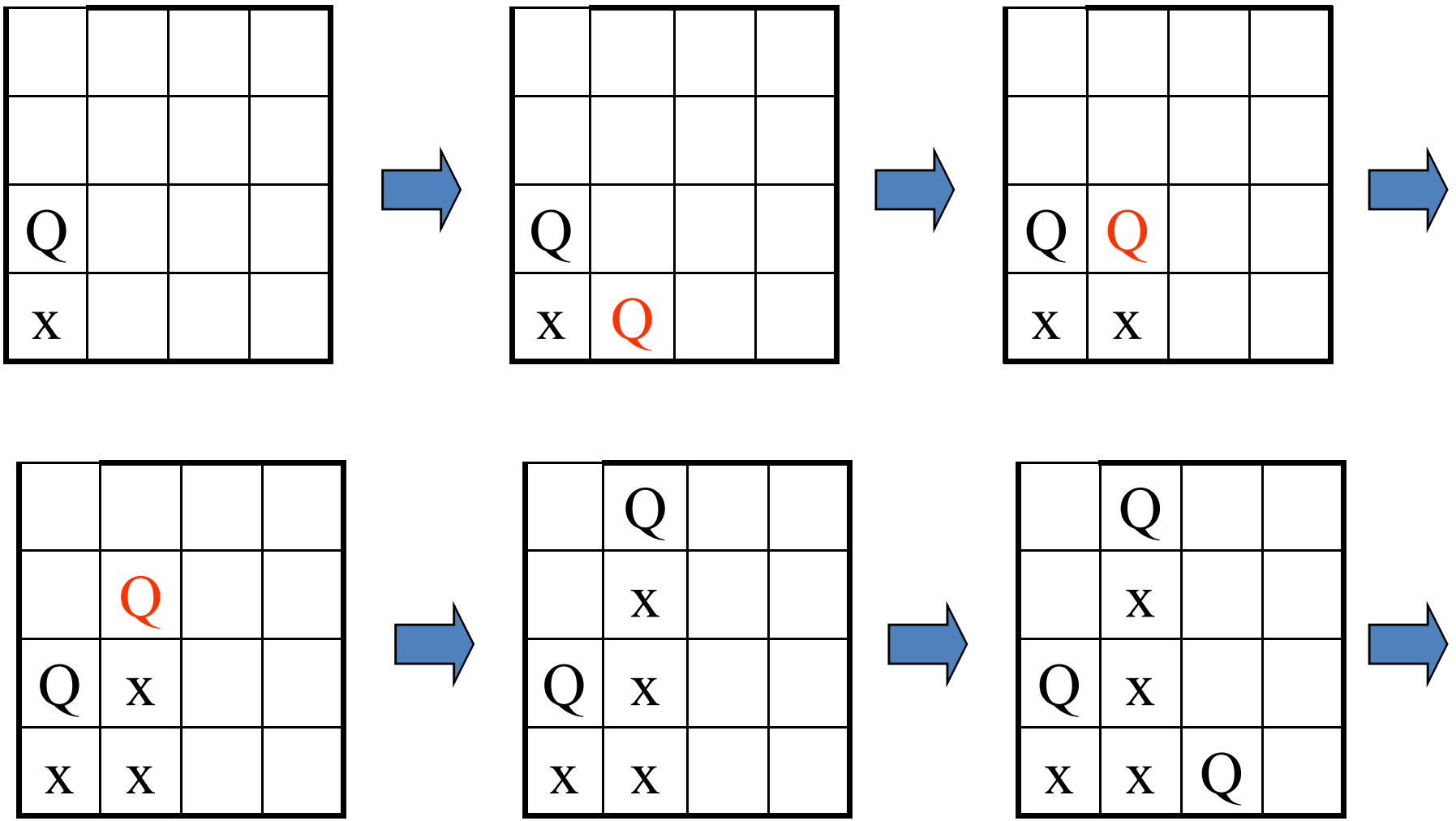
4-Queens



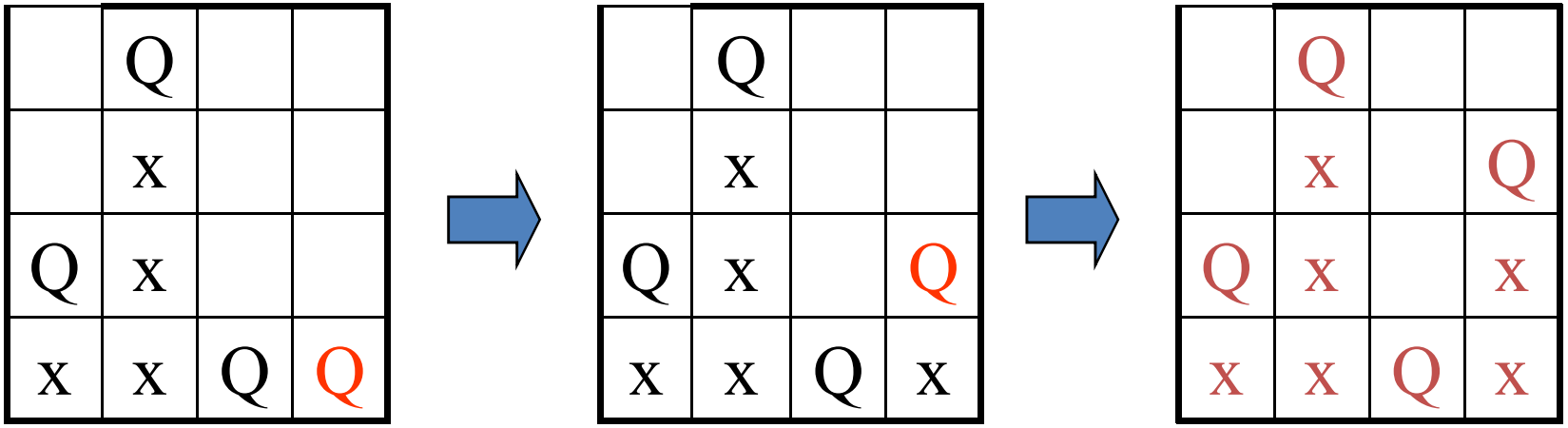
4-Queens



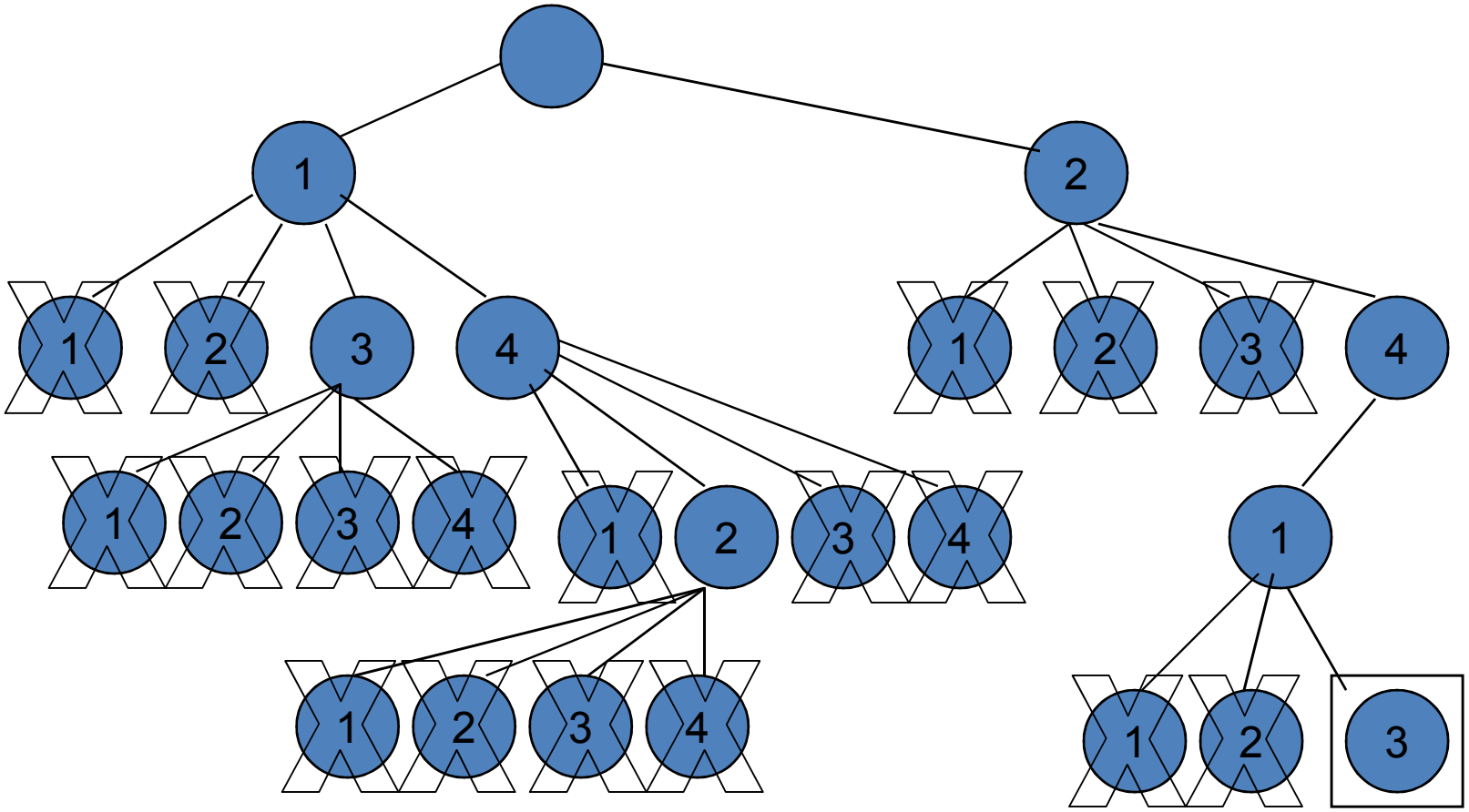
4-Queens



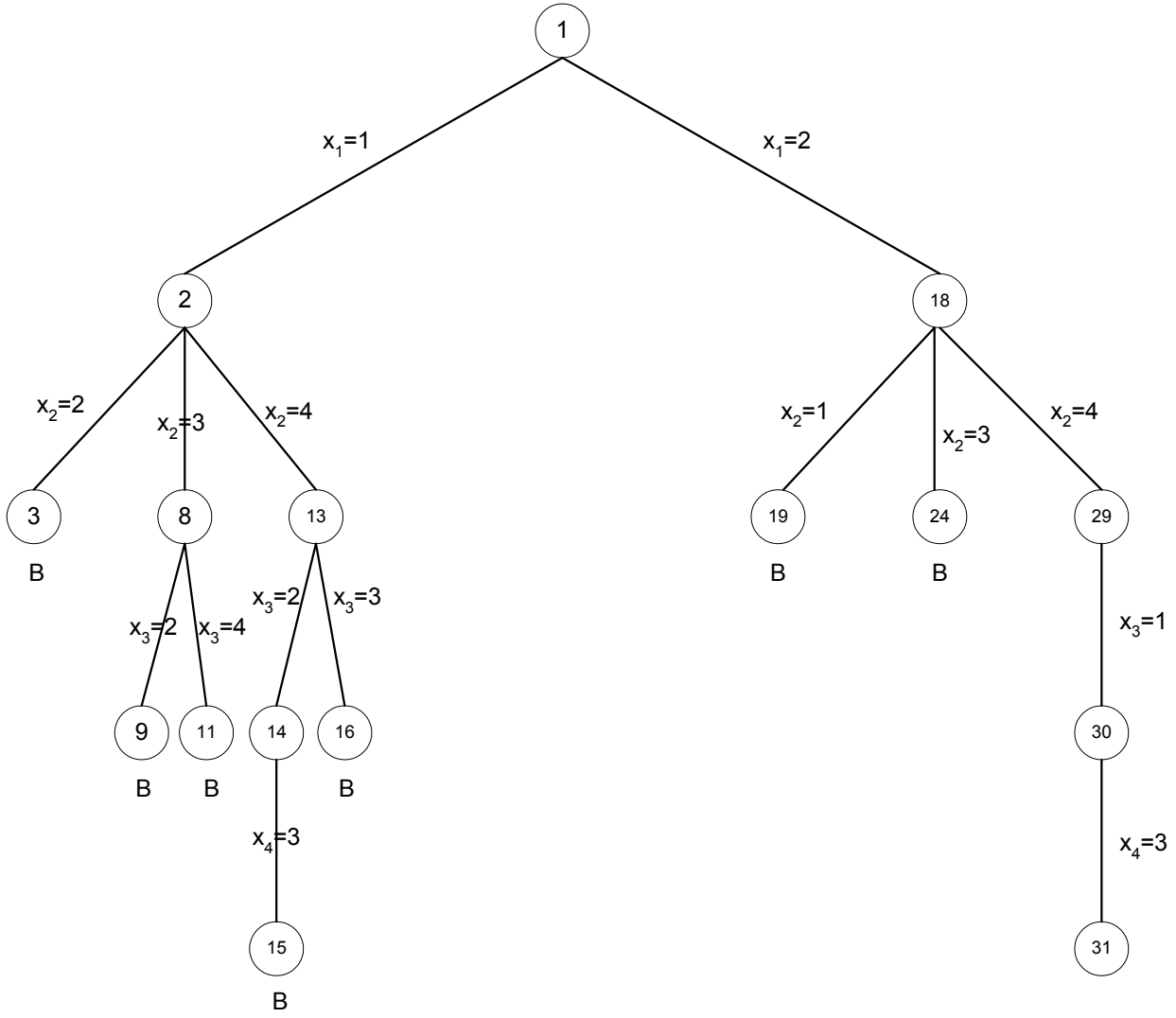
4-Queens



4-Queens

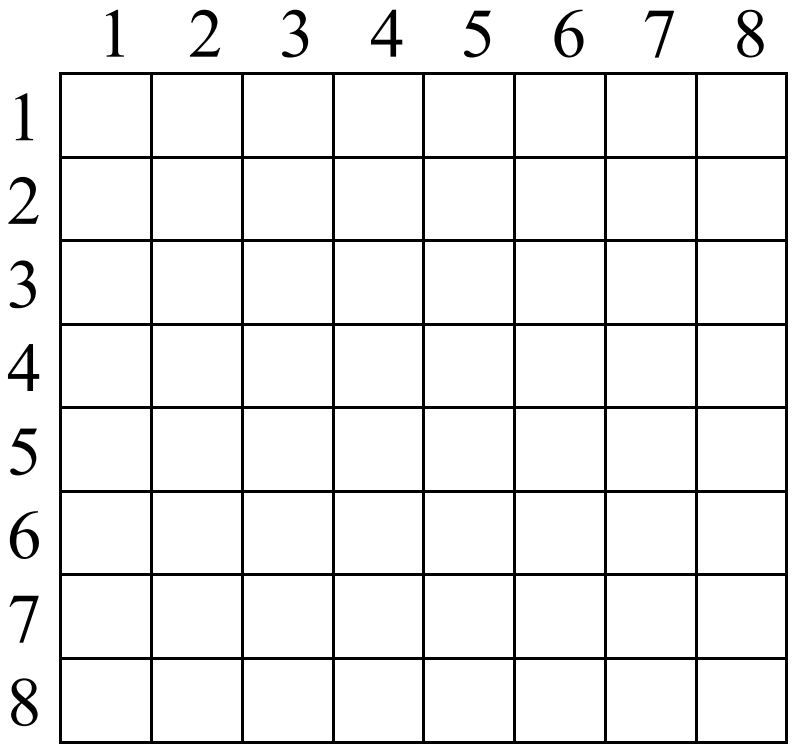


Pohon ruang status dinamis persoalan 4-Ratu yang dibentuk selama pencarian



Algoritma Runut-balik untuk Persoalan 8-Ratu (iteratif)

- Matrik papan catur



Dua buah ratu terletak pada baris yang sama, berarti $i=k$

Dua buah ratu terletak pada kolom yang sama, berarti $j=l$

Dua buah ratu terletak pada diagonal yang sama, berarti

$\searrow$ $i-j=k-l$ atau $\swarrow$ $i+j=k+l$

$\Leftrightarrow$ $i-k=j-l$ atau $k-i=j-l$

$\Leftrightarrow$ $|j-l|=|i-k|$

Skema iteratif : N_Ratu_I(8)

```
procedure N_RATU_I(input N:integer)
{ Mencetak semua solusi penempatan N buah ratu pada
  petak papan catur N x N tanpa melanggar kendala; versi iteratif
 Masukan: N = jumlah ratu
  Keluaran: semua solusi x = (x[1], x[2], …, x[N]) dicetak ke layar.
}
Deklarasi
  k : integer
Algoritma:
  k←1              {mulai pada baris catur ke-1}
  x[1]←0           {inisialisasi kolom dengan 0}
  while k > 0 do
     x[k]←x[k]+1  {pindahkan ratu ke kolom berikutnya}
     while (x[k] ≤ N) and (not TEMPAT(k)) do
        {periksa apakah ratu dapat ditempatkan pada kolom x[k]}
        x[k]:=x[k] + 1
     endwhile
     {x[k] > n or TEMPAT(k) }
     if x[k]≤ n then   { kolom penempatan ratu ditemukan }
        if k=N then   { apakah solusi sudah lengkap?}
          CetakSolusi(x,N)       { cetak solosi}
        else
          k←k+1   {pergi ke baris berikutnya}
          x[k]←0  {inisialisasi kolom dengan 0}
        endif
     else
        k←k-1     { runut-balik ke baris sebelumnya}
    endif
  endwhile
  { k = 0 }
```

```
function TEMPAT(input k:integer)→boolean
{true jika ratu dapat ditempatkan pada kolom x[k], false jika tidak}
Deklarasi
   i : integer
   stop : boolean
Algoritma:
   kedudukan←true    { asumsikan ratu dapat ditempatkan pada kolom x[k] }
   { periksa apakah memang ratu dapat ditempatkan pada kolom x[k] }
   i←1        { mulai dari baris pertama}
   stop←false
   while (i<k) and (not stop) do
      if  (x[i]=x[k]){apakah ada dua buah ratu pada kolom yang sama?}
                    or                                          { atau}
          (ABS(x[i]-x[k])=ABS(i-k)) {dua ratu pada diagonal yang sama?}
      then
        kedudukan←false
        keluar←true
      else
          i←i+1            { periksa pada baris berikutnya}
      endif
   endwhile
   { i = k or keluar }

   return kedudukan
```

Versi rekursif

Algoritma:

Inisialisasi *x*[1], *x*[2], …, *x*[*N*] dengan 0

```
  for i←N to n do
        x[i]←0
   endfor
```

Panggil prosedur N_RATU_R(1)

source1 simulasi1 source2 simulasi2

```
procedure N_RATU_R(input k:integer)
{ Menempatkan ratu pada baris ke-k pada petak papan catur N x N
  tanpa melanggar kendala; versi rekursif
  Masukan: N = jumlah ratu
  Keluaran: semua solusi x = (x[1], x[2], …, x[N]) dicetak ke layar. }

Deklarasi
  stop : boolean
Algoritma:
  stop←false
  while not stop do
     x[k]←x[k]+1  { pindahkan ratu ke kolom berikutnya }
     while (x[k] ≤ n) and (not TEMPAT(k)) do
     { periksa apakah ratu dapat ditempatkan pada kolom x[k] }
         x[k]←x[k]+1
     endwhile
     { x[k] > n or TEMPAT(k) }
     if x[k] ≤ N then        { kolom penempatan ratu ditemukan }
        if k=N then               { apakah solusi sudah lengkap? }

          CetakSolusi(x,N)  { cetak solusi }
        else
          N_RATU_R(k+1)
        else          { x[k] > N → gagal, semua kolom sudah dicoba }
          stop←true
          x[k]←0
      endif
   endwhile
  {stop}
```