Algoritma Runut-balik (*Backtracking*)

Bahan Kuliah IF2251 Strategi Algoritmik

Oleh: Rinaldi Munir

Pendahuluan

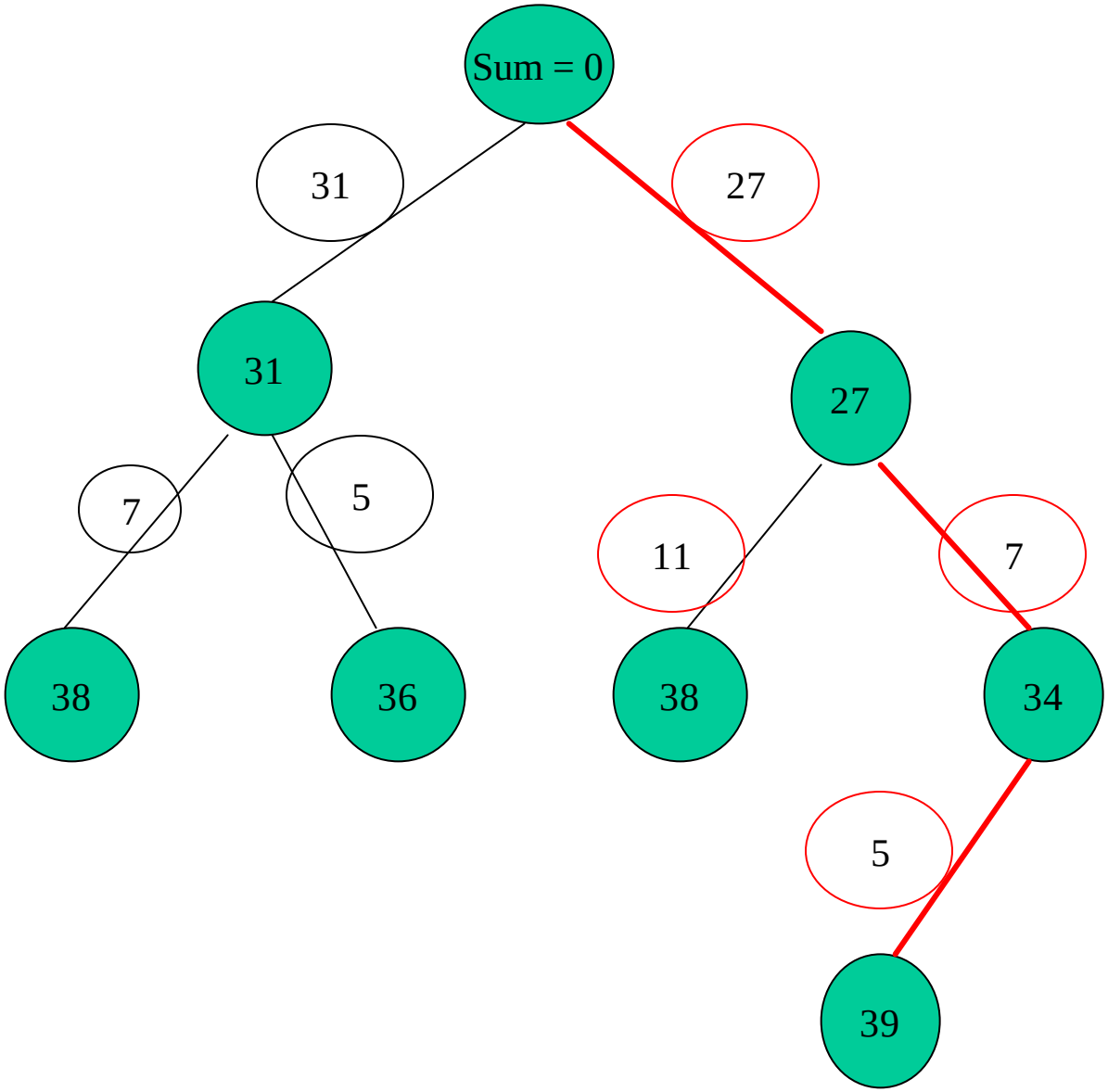
- Runut-balik (*backtracking*) adalah algoritma yang berbasis pada DFS untuk mencari solusi persoalan secara lebih mangkus.
- Runut-balik, yang merupakan perbaikan dari algoritma *brute-force*, secara sistematis mencari solusi persoalan di antara semua kemungkinan solusi yang ada.
- Memangkas (*pruning*) simpul-simpul yang tidak mengarah ke solusi.

- Algoritma runut-balik banyak diterapkan untuk program *games* :
  - permainan *tic-tac-toe*,
  - menemukan jalan keluar dalam sebuah labirin,
  - Catur,
  - dan masalah-masalah pada bidang kecerdasan buatan (*artificial intelligence*).

Contoh Pengantar

(Soal UTS 2007)

(*Sum of subsets*). Diberikan *A* adalah sebuah himpunan yang berisi sejumlah bilangan bulat, x_1, x_2, ..., x_n. Asusmikan bahwa himpunan A sudah terurut menurun. Anda diminta menentukan himpunan bagian dari A yang mempunyai jumlah *M*. Misalnya, jika *A* = {31, 27, 15, 11, 7, 5} dan *M* = 27, maka himpunan bagian tersebut adalah {5, 15, 7}. Perhatikan bahwa solusi masalah ini bisa lebih dari 1 (solusi lainnya adalah {27}). Gambarkan pembentukan pohon *DFS* secara dinamis untuk mencari himpunan bagian pertama dari *A* yang mempunyai jumlah *M* = 39.



Solusi: [27, 7, 5]

- Istilah Runut-balik pertama kali diperkenalkan oleh D. H. Lehmer pada tahun 1950.

- R.J Walker, Golomb, dan Baumert menyajikan uraian umum tentang runut-balik.

Properti Umum Metode Runut-balik

1. **Solusi persoalan.**

- Solusi dinyatakan sebagai vektor dengan *n-tuple*: $X = (x_1, x_2, \ldots, x_n)$, $x_i \in S_i$.
- Mungkin saja $S_1 = S_2 = \ldots = S_n$.
- Contoh: $S_i = \{0, 1\}$, $x_i = 0$ atau 1

2. **Fungsi pembangkit** nilai x_k

   Dinyatakan sebagai:

   $T(k)$

   $T(k)$ membangkitkan nilai untuk x_k, yang merupakan komponen vektor solusi.

3. Fungsi pembatas

- Dinyatakan sebagai

  $B(x_1, x_2, \ldots, x_k)$

- *B* bernilai *true* jika $(x_1, x_2, \ldots, x_k)$ mengarah ke solusi.

- Jika *true*, maka pembangkitan nilai untuk x_{k+1} dilanjutkan, tetapi jika *false*, maka $(x_1, x_2, \ldots, x_k)$ dibuang.

Pengorganisasian Solusi

- Semua kemungkinan solusi dari persoalan disebut **ruang solusi** (*solution space*).

- Jika $x_i \in S_i$, maka $S_1 \times S_2 \times \ldots \times S_n$ disebut **ruang solusi**.

- Jumlah anggota di dalam ruang solusi adalah $|S_1| \cdot |S_2| \cdot \ldots \cdot |S_n|$.

- Tinjau *Knapsack* 0/1 untuk $n = 3$.

- Solusi persoalan dinyatakan sebagai (x_1, x_2, x_3) dengan $x_i \in \{0,1\}$.

Ruang solusinya adalah

$\{0,1\} \times \{0,1\} \times \{0,1\} = \{(0, 0, 0), (0, 1, 0), (0, 0, 1), (1, 0, 0), (1, 1, 0), (1, 0, 1), (0, 1, 1), (1, 1, 1)\}$.

- Pada *Knapsack* 0/1 dengan $n = 3$ terdapat $2^n = 2^3 = 8$ kemungkinan solusi, yaitu:

  (0, 0, 0), (0, 1, 0), (0, 0, 1),
  (1, 0, 0), (1, 1, 0), (1, 0, 1),
  (0, 1, 1), dan (1, 1, 1).

- Penyelesaian secara *exhaustive search* adalah dengan menguji setiap kemungkinan solusi.

- Ruang solusi diorganisasikan ke dalam struktur pohon.

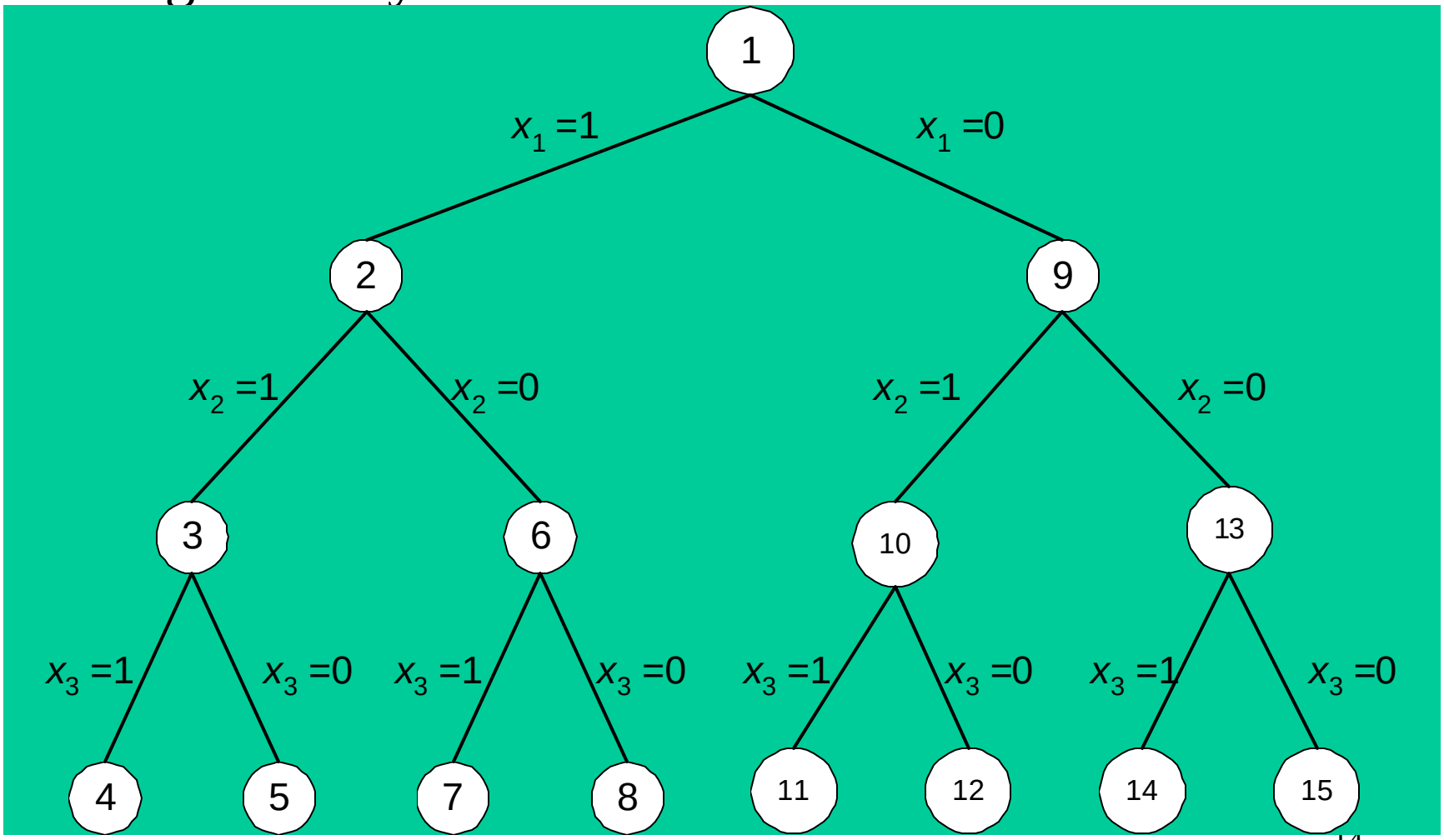
- Tiap simpul pohon menyatakan status (*state*) persoalan, sedangkan sisi (cabang) dilabeli dengan nilai-nilai x_i.

- Lintasan dari akar ke daun menyatakan solusi yang mungkin.

- Seluruh lintasan dari akar ke daun membentuk ruang solusi. Pengorganisasian pohon ruang solusi diacu sebagai **pohon ruang status** (*state space tree*).

Tinjau persoalan *Knapsack* 1/0 untuk $n = 3$.

Ruang solusinya:



Prinsip Pencarian Solusi dengan Metode Runut-balik

- Solusi dicari dengan membentuk lintasan dari akar ke daun. Aturan pembentukan yang dipakai adalah mengikuti aturan pencarian mendalam (DFS).

- Simpul-simpul yang sudah dilahirkan dinamakan **simpul hidup** (*live node*).

- Simpul hidup yang *sedang* diperluas dinamakan **simpul-E** (*Expand-node*).

- Tiap kali simpul-E diperluas, lintasan yang dibangun olehnya bertambah panjang.

- Jika lintasan yang sedang dibentuk tidak mengarah ke solusi, maka simpul-E tersebut “dibunuh” sehingga menjadi **simpul mati** (*dead node*).

- Fungsi yang digunakan untuk membunuh simpul-E adalah dengan menerapkan **fungsi pembatas** (*bounding function*).

- Simpul yang sudah mati tidak akan pernah diperluas lagi.

- Jika pembentukan lintasan berakhir dengan simpul mati, maka proses pencarian diteruskan dengan membangkitkan simpul anak yang lainnya.

- Bila tidak ada lagi simpul anak yang dapat dibangkitkan, maka pencarian solusi dilanjutkan dengan melakukan runut-balik (*backtrack*) ke simpul hidup terdekat (simpul orangtua).

- Selanjutnya simpul ini menjadi simpul-E yang baru.

- Pencarian dihentikan bila kita telah menemukan solusi atau tidak ada lagi simpul hidup untuk runut-balik.

- Tinjau persoalan *Knapsack* 0/1 dengan instansiasi:

  $n = 3$

  $(w_1, w_2, w_3) = (35, 32, 25)$

  $(p_1, p_2, p_3) = (40, 25, 50)$

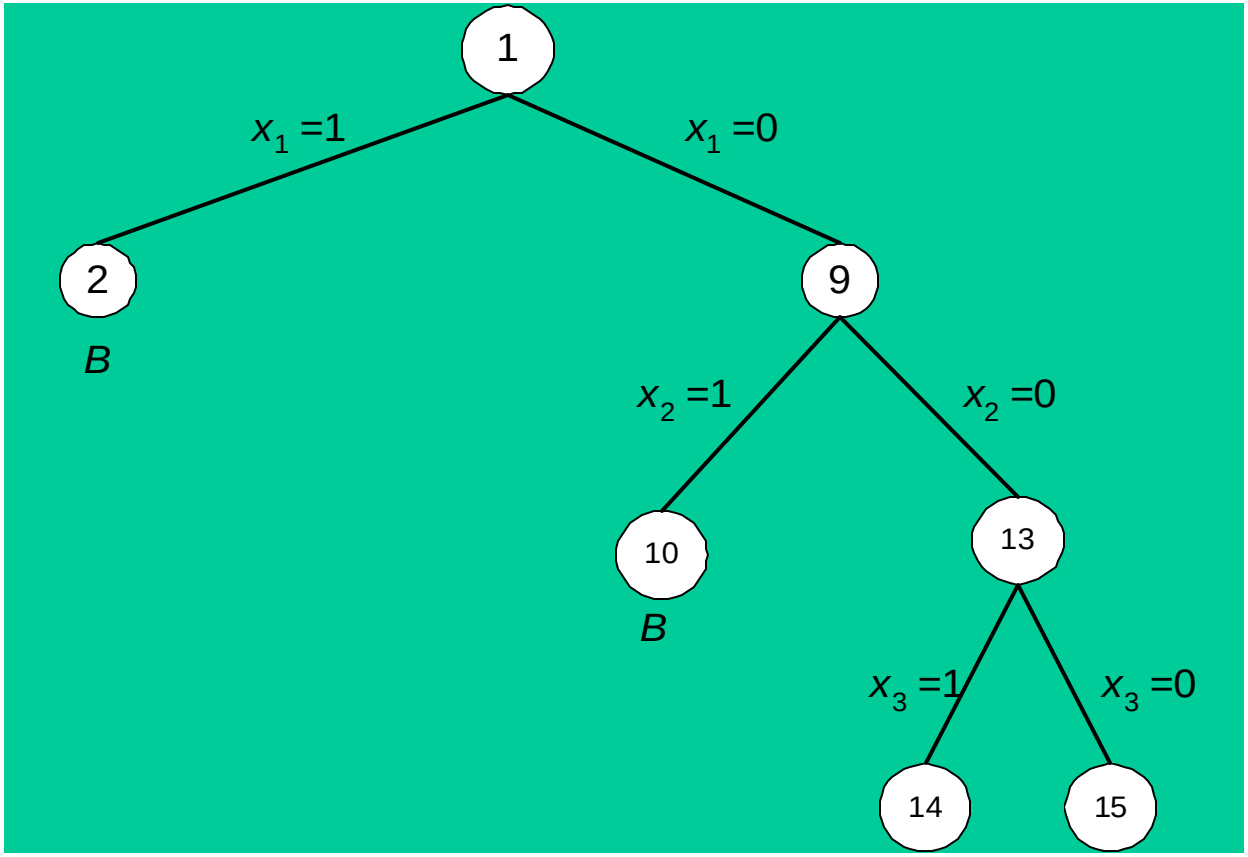
  $M = 30$

- Solusi dinyatakan sebagai $X = (x_1, x_2, x_3)$, $x_i \in \{0, 1\}$.

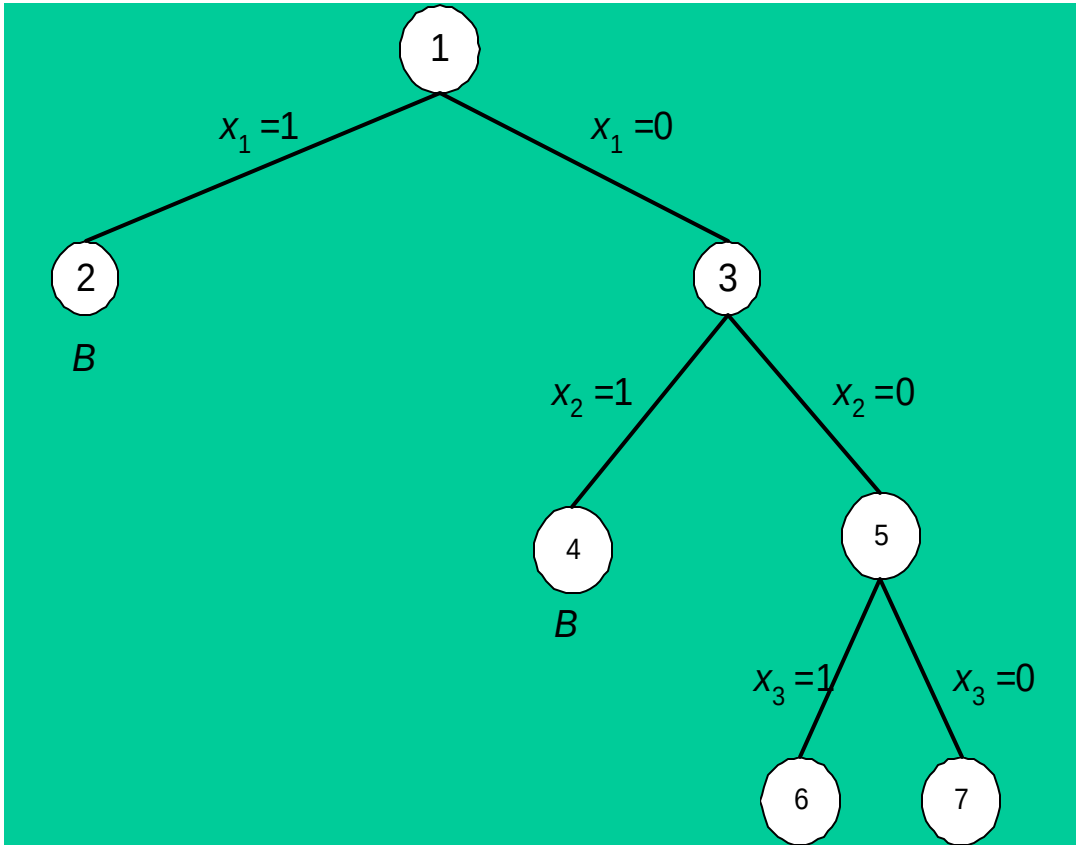
- Fungsi pembatas:

$$\sum_{i=1}^{k} w_i x_i \leq M$$

Pohon dinamis yang dibentuk selama pencarian untuk persoalan *Knapsack* 0/1 dengan $n = 3$, $M = 30$, $w = (35, 32, 25)$ dan $p = (40, 25, 50)$



Penomoran ulang simpul-simpul sesuai urutan pembangkitannya



Solusi optimumnya adalah X = (0, 0, 1) dan F = 50.

Skema Umum Algoritma Runut-Balik (versi rekursif)

```
procedure RunutBalikR(input k:integer)
{Mencari semua solusi persoalan dengan metode runut-balik; skema rekursif
 Masukan: k, yaitu indeks komponen vektor solusi, x[k]
 Keluaran: solusi x = (x[1], x[2], …, x[n])
}
Algoritma:
   for tiap x[k] yang belum dicoba sedemikian sehingga
        ( x[k]←T(k)) and B(x[1], x[2], ... ,x[k])= true do
      if (x[1], x[2], ... ,x[k]) adalah lintasan dari akar ke daun
      then
        CetakSolusi(x)
      endif
      RunutBalikR(k+1)    { tentukan nilai untuk x[k+1]}
   endfor
```

Skema Umum Algoritma Runut-Balik (versi iteratif)

```
procedure RunutBalikI(input n:integer)
{Mencari semua solusi persoalan dengan metode runut-balik; skema iteratif.
 Masukan: n, yaitu panjang vektor solusi
 Keluaran: solusi x = (x[1], x[2], …, x[n])
}
Delarasi:
   k : integer

Algoritma:
   k←1
   while k > 0 do
      if (x[k] belum dicoba sedemikian sehingga x[k]←T(k)) and
          (B(x[1], x[2], ... ,x[k])= true)
      then
         if (x[1],x[2],...,x[k]) adalah lintasan dari akar ke daun
         then
           CetakSolusi(x)
         endif
         k←k+1   {indeks anggota tupple berikutnya}
      else  {x[1], x[2], …, x[k] tidak mengarah ke simpul solusi }
         k←k-1   {runut-balik ke anggota tupple sebelumnya}
      endif
   endwhile
  { k = 0 }
```

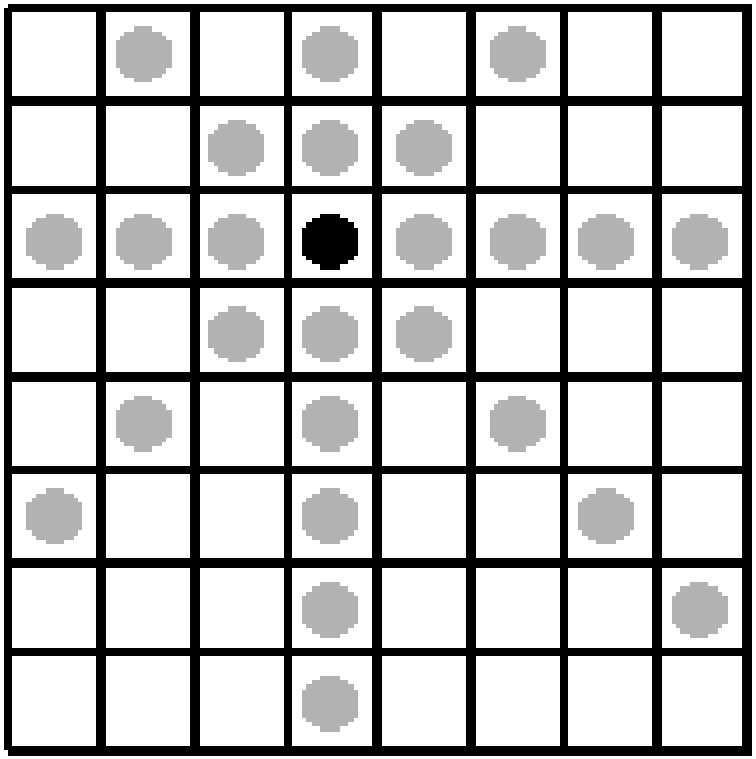
- Setiap simpul dalam pohon ruang status berasosiasi dengan sebuah pemanggilan rekursif.

- Jika jumlah simpul dalam pohon ruang status adalah 2^n atau $n!$, maka untuk kasus terburuk, algoritma runut-balik membutuhkan waktu dalam $O(p(n)2^n)$ atau $O(q(n)n!)$,

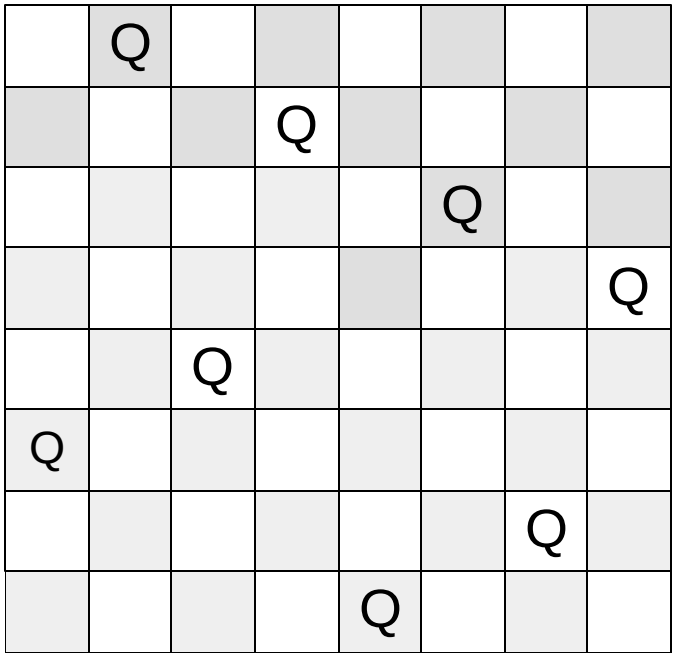
- dengan $p(n)$ dan $q(n)$ adalah polinom derajat n yang menyatakan waktu komputasi setiap simpul.

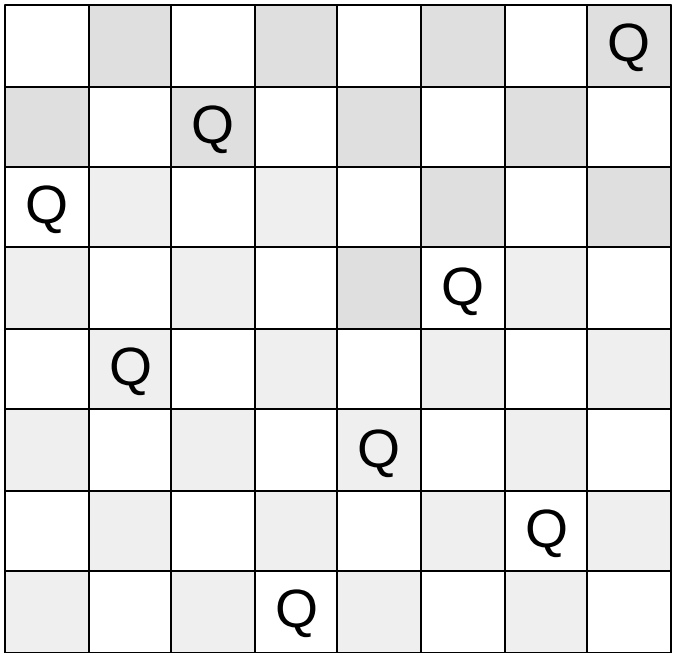
Persoalan N-Ratu (*The N-Queens Problem*)

- 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?



Contoh 2 buah solusi *8-queen problem*:





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

$$8^8 = 16.777.216$$

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.

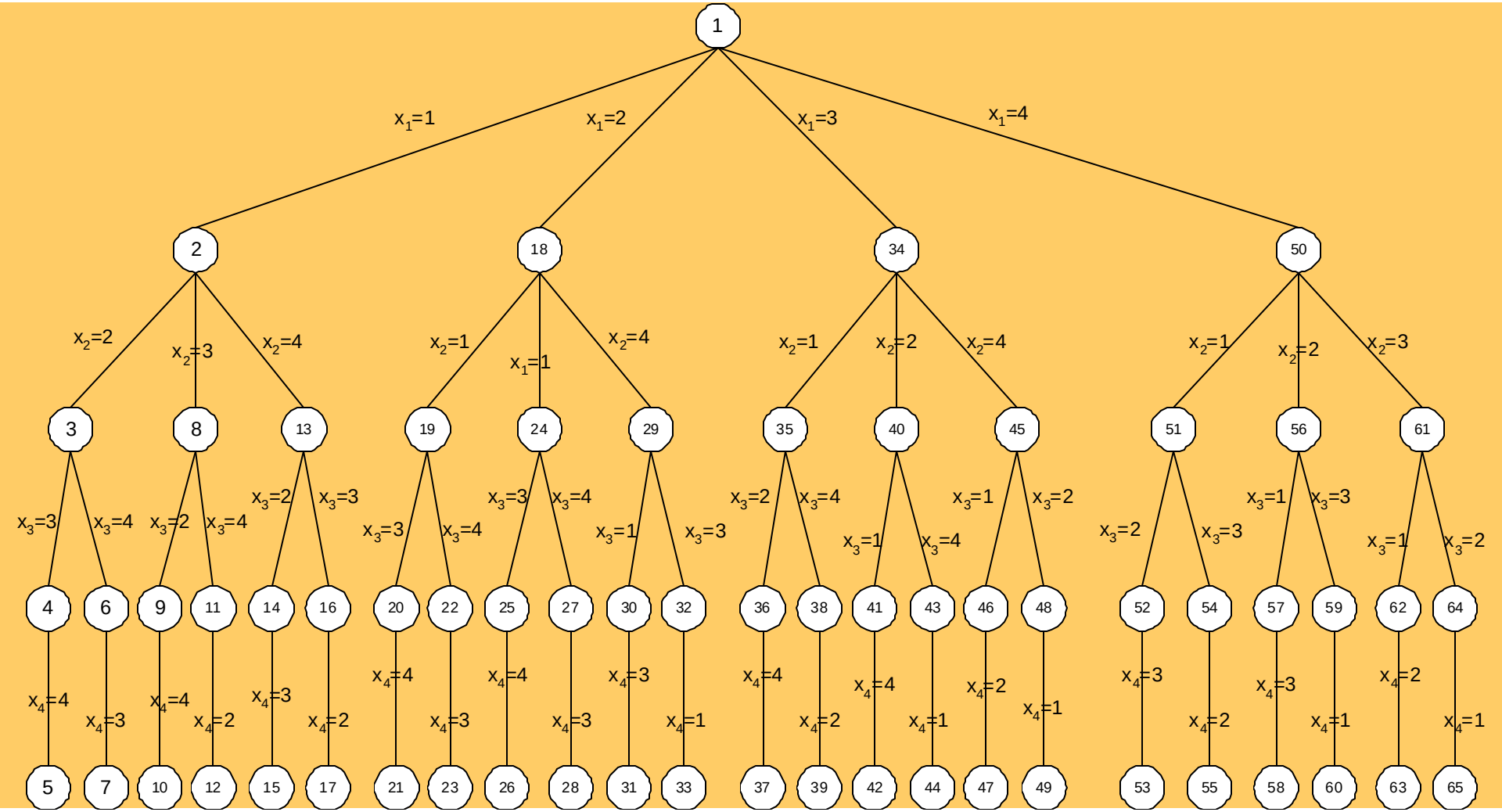
Penyelesaian dengan Algoritma Runut-balik:

- Algoritma runut-balik memperbaiki algoritma *brute force* 3 (*exhaustive search*).

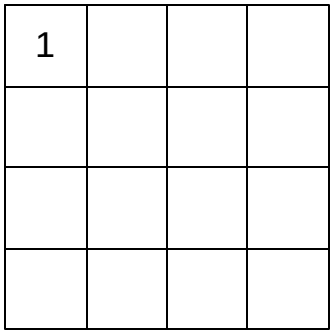
- Ruang solusinya adalah semua permutasi dari angka-angka 1, 2, 3, 4, 5, 6, 7, 8.

- Setiap permutasi dari 1, 2, 3, 4, 5, 6, 7, 8 dinyatakan dengan lintasan dari akar daun. Sisi-sisi pada pohon diberi label nilai *xi*.

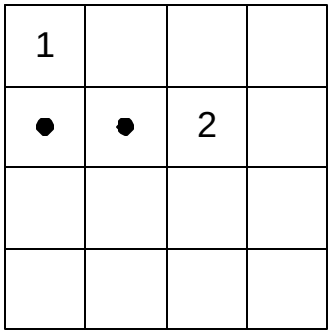
Contoh: Pohon ruang-status persoalan 4-Ratu



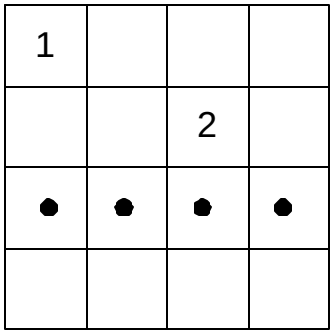
Contoh solusi runut-balik persoalan 4-Ratu:



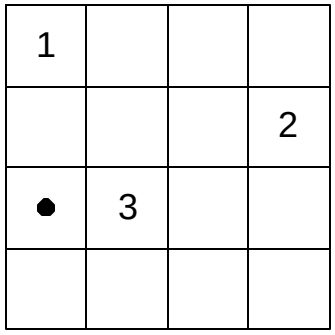
(a)



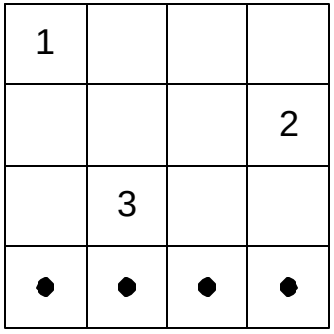
(b)



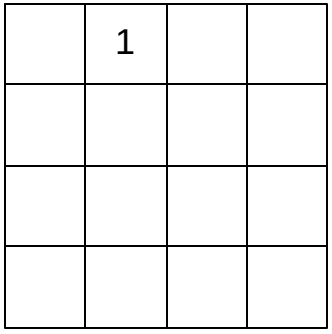
(c)



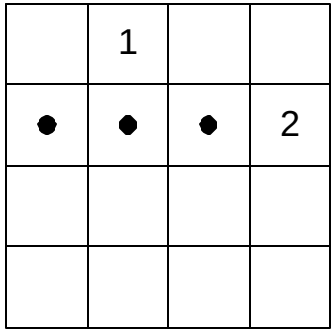
(d)



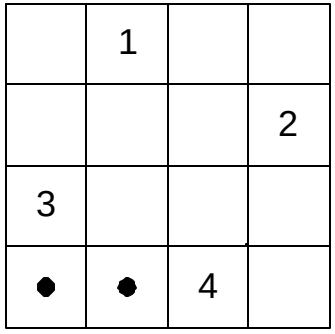
(e)



(f)

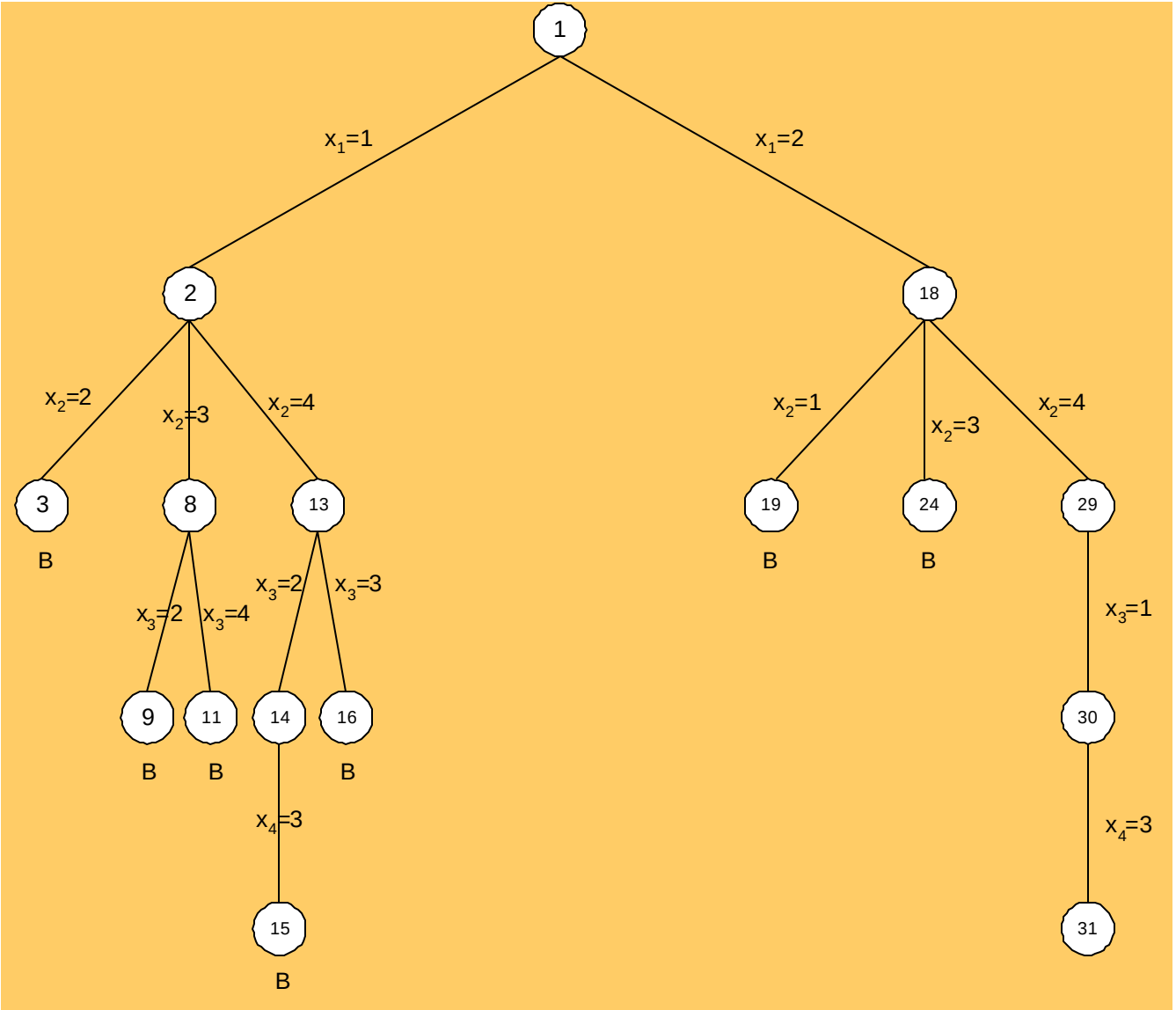


(g)



(h)

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



Algoritma Runut-balik untuk Persoalan 8-Ratu

(a) Versi iteratif

- *Dua buah ratu terletak pada baris yang sama, berarti*

  $i = k$

- *Dua buah ratu terletak pada kolom yang sama, berarti*

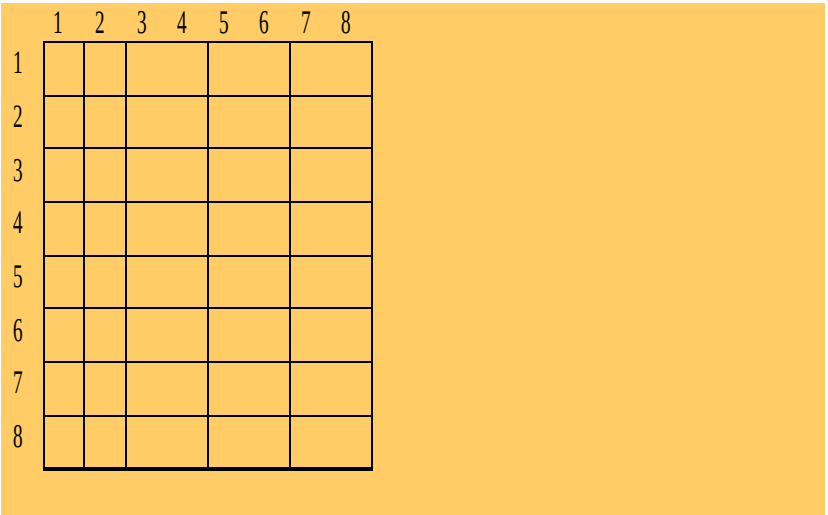
  $j=l$

- *Dua buah ratu terletak pada diagonal yang sama, berarti*

  ↘ $i-j=k-l$ *atau* ↙ $i+j=k+l$

  Û $i-k=j-l$ *atau* $k-i=j-l$

  Û $|j-l| = |i-k|$



```
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
```

(b) 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)

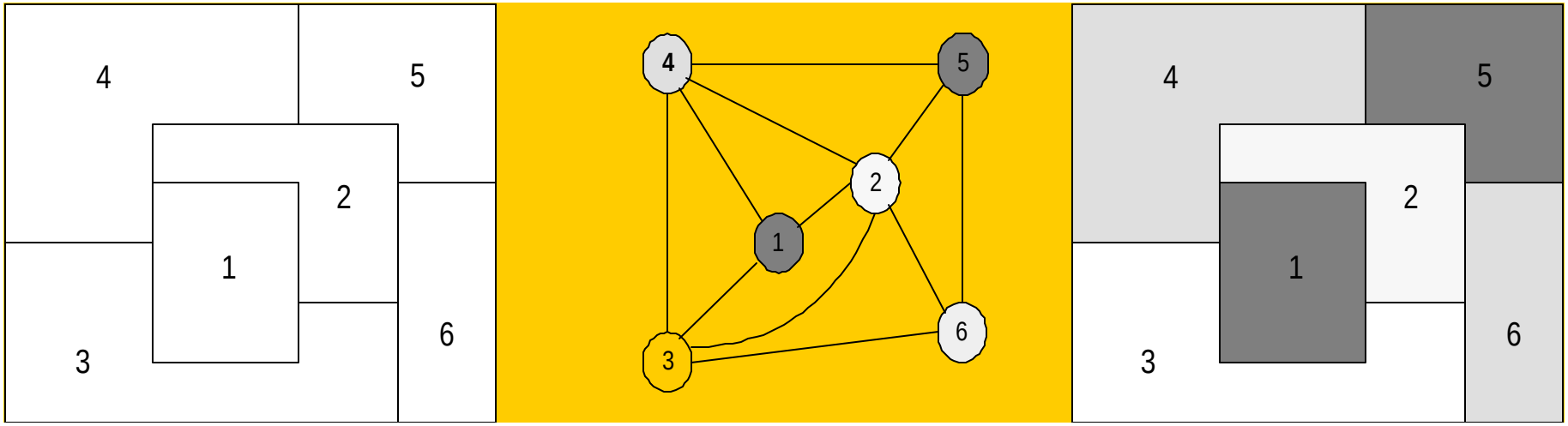
```
procedure N_RATU_R(input k:integer)
{ Menempatkan ratu pada baris ke-k pada petak papan catur N x N
  tanpa melanggar kendala; versi rekursif

  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}
```

Pewarnaan Graf
(*Graph Colouring)*

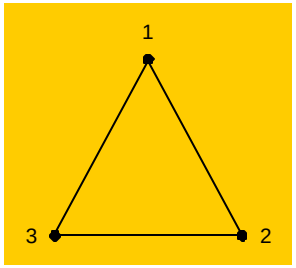
Persoalan:

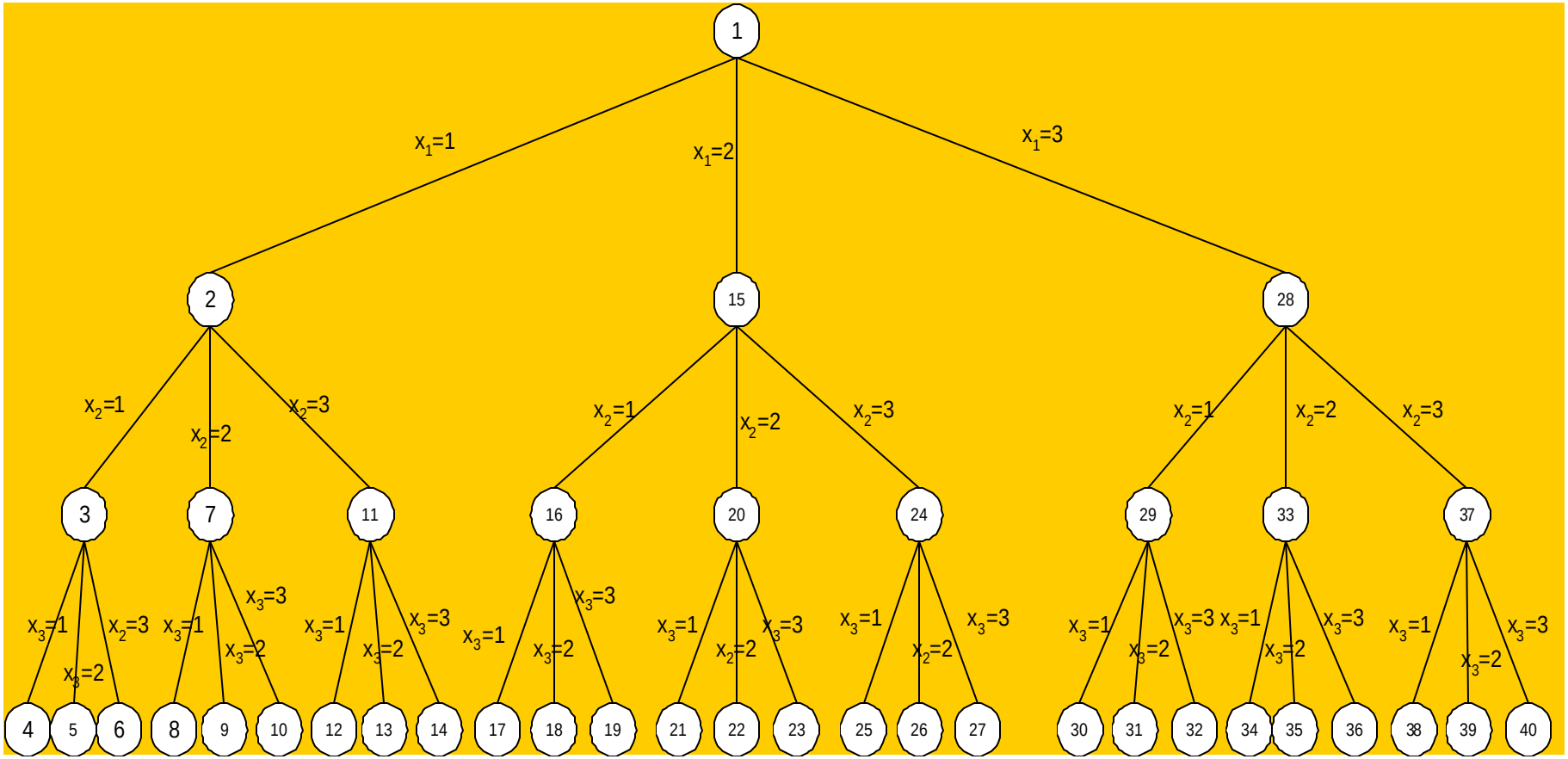
- Diberikan sebuah graf *G* dengan *n* buah simpul dan disediakan *m* buah warna. Bagaimana mewarnai seluruh simpul graf *G* sedemikian sehingga tidak ada dua buah simpul bertetangga yang mempunyai warna sama (Perhatikan juga bahwa tidak seluruh warna harus dipakai)

Contoh aplikasi: pewarnaan peta



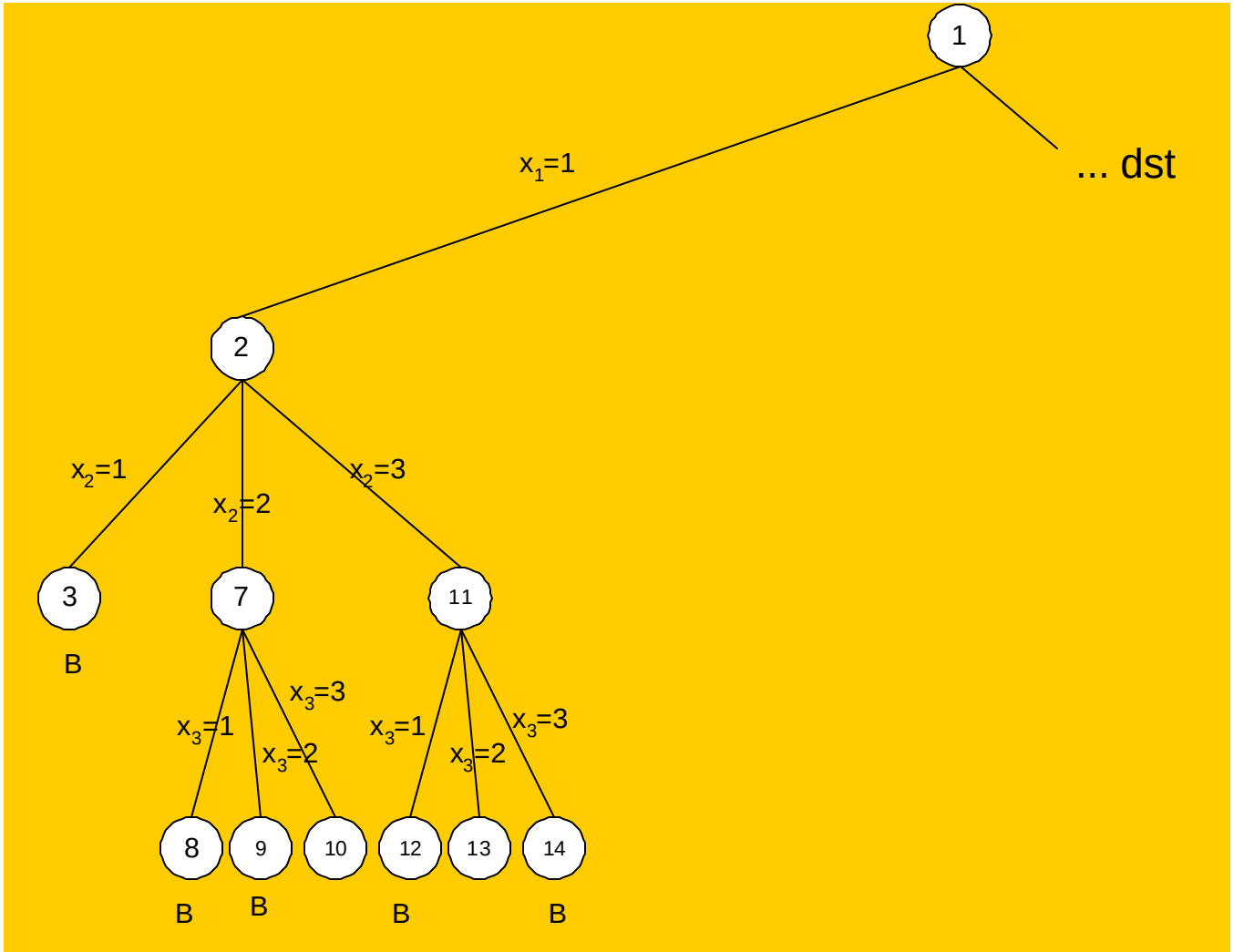
Tinjau untuk $n = 3$ dan $m = 3$.





Misalkan warna dinyatakan dengan angka 1, 2, …, *m* dan solusi dinyatakan sebagai vektor X dengan *n-tuple*:

$X = (x_1 , x_2 , ..., x_n) , \quad x_i \in \{ 1, 2, \ldots, m\}$



Algoritma Runut-balik Untuk Pewarnaan Graf

- Masukan:

  1. Matriks ketetanggan GRAF[1..n, 1..n]

     GRAF[i,j] = true jika ada sisi (i,j)

     GRAF[i,j] = false jika tidak ada sisi (i,j)

  2. Warna

     Dinyatakan dengan integer 1, 2, ...,m

- Keluaran:

  1. Tabel X[1..n], yang dalam hal ini, x[i] adalah warna untuk simpul i.

- Algoritma:

  1. Inisialisasi x[1..n] dengan 0

```
for i←1 to n do
  x[i]←0
endfor
```

2. Panggil prosedur PewarnaanGraf(1)

```
procedure PewarnaanGraf(input k : integer)
{ Mencari semua solusi solusi pewarnaan graf; rekursif
  Masukan: k adalah nomor simpul graf.
  Keluaran: jika solusi ditemukan, solusi dicetak ke piranti
keluaran
}
Deklarasi
  stop : boolean

Algoritma:
  stop←false
  while not stop do
     {tentukan semua nilai untuk x[k] }
     WarnaBerikutnya(k) {isi x[k] dengan sebuah warna}
     if x[k] = 0 then    {tidak ada warna lagi, habis}
      stop←true
     else
      if k=n then       {apakah seluruh simpul sudah diwarnai?}
         CetakSolusi(X,n)
      else
         PewarnaanGraf(k+1)   {warnai simpul berikutnya}
       endif
    endif
  endwhile
```

```
procedure WarnaBerikutnya(input k:integer)
{ Menentukan warna untuk simpul k

 Masukan: k
 Keluaran: nilai untuk x[k]

 K.Awal: x[1], x[2], ... , x[k-1] telah diisi dengan warna dalam
himpunan {1,2, …, m} sehingga setiap simpul bertetangga mempunyai
warna berbeda-beda.
 K.Akhir: x[k] berisi dengan warna berikutnya apabila berbeda dengan
warna simpul-simpul tetangganya. Jika tidak ada warna yang dapat
digunakan, x[k] diisi dengan nol
}
Deklarasi
  stop, keluar : boolean
  j : integer

Algoritma:
  stop←false
  while not stop do
     x[k]←(x[k]+1) mod (m+1) {warna berikutnya}
     if x[k]=0  then          {semua warna telah terpakai}
      stop←true
     else
        {periksa warna simpul-simpul tetangganya}
       j←1
       keluar←false
        while (j≤n) and (not keluar) do

           if (GRAF[k,j]) {jika ada sisi dari simpul k ke simpul j}
              and             {dan}
            (x[k] = x[j])     {warna simpul k = warna simpul j }
           then
            keluar←true {keluar dari kalang}
           else
            j←j+1      {periksa simpul berikutnya}
         endif
        endwhile
        { j > n or keluar}

        if j=n+1 {seluruh simpul tetangga telah diperiksa dan
                 ternyata warnanya berbeda dengan x[k] }
        then
           stop←true   {x[k] sudah benar, keluar dari kalang}
        endif

     endif
  endwhile
```

Kompleksitas Waktu algoritma PewarnaanGraf

- Pohon ruang status yang untuk persoalan pewarnaan graf dengan *n* simpul dan *m* warna adalah pohon *m-ary* dengan tinggi $n + 1$.

- Tiap simpul pada aras *i* mempunyai *m* anak, yang bersesuaian dengan *m* kemungkinan pengisian $x[i]$, $1 \leq i \leq n$.

- Simpul pada aras $n+1$ adalah simpul daun. Jumlah simpul internal (simpul bukan daun) ialah $\sum_{i=0}^{n-1} m^i$.

- Tiap simpul internal menyatakan pemanggilan prosedur `WarnaBerikutnya` yang membutuhkan waktu dalam $O(mn)$. Total kebutuhan waktu algoritma `PewarnaanGraf` adalah

$$\sum_{i=1}^{n} m^i n = \frac{n(m^{n+1}-1)}{(m-1)} = O(nm^n)$$