Algoritma Runut-balik (*Backtracking*) (Bagian 1)

Bahan Kuliah IF2211 Strategi Algoritma

Oleh: Rinaldi M

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika ITB
2026

Pendahuluan

- *Backtracking* dapat dipandang sebagai salah satu dari dua hal berikut:
  1. Sebagai sebuah *fase* di dalam algoritma traversal DFS → Sudah dijelaskan di dalam materi DFS/BFS.

  2. Sebagai sebuah *metode* pemecahan masalah yang sangkil, terstruktur, dan sistematis, baik untuk persoalan optimasi maupun non-optimasi

Backtacking sebagai sebuah metode pemecahan masalah yang sangkil

- Algoritma runut-balik merupakan perbaikan dari *exhaustive search*.
- Pada *exhaustive search*, semua kemungkinan solusi dieksplorasi dan dievaluasi satu per satu.
- Sedangkan pada algoritma *backtracking*, hanya pilihan yang mengarah ke solusi yang dieksplorasi, pilihan yang tidak mengarah ke solusi tidak dipertimbangkan lagi

  → Memangkas (*pruning*) simpul-simpul yang tidak mengarah ke solusi.

- Algoritma runut-balik pertama kali diperkenalkan oleh D. H. Lehmer tahun 1950.
- Kemudian R.J Walker, Golomb, dan Baumert menyajikan uraian umum tentang algoritma runut-balik.

Properti Umum Algoritma Runut-balik

1. **Solusi persoalan.**

- Solusi dinyatakan sebagai vektor dengan *n-tuple*: $X = (x_1, x_2, \ldots, x_n)$, $x_i \in S_i$. Umumnya $S_1 = S_2 = \ldots = S_n$.

- Contoh: Pada persoalan *1/0 knapsack* $S_i = \{0, 1\}$, $x_i = 0$ atau 1

2. Fungsi pembangkit nilai x_k

- Dinyatakan sebagai fungsi $T()$

- $T(x[1], x[2], \ldots, x[k-1])$ membangkitkan nilai untuk x_k, yang merupakan komponen vektor solusi.

3. Fungsi pembatas (*bounding function*)

- Dinyatakan sebagai predikat $B(x_1, x_2, \ldots, x_k)$ yang bernilai true/false

- *B* bernilai *true* jika $(x_1, x_2, \ldots, x_k)$ mengarah ke solusi. Mengarah ke solusi artinya tidak melanggar kendala (*constraints*)

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

Pengorganisasian Solusi

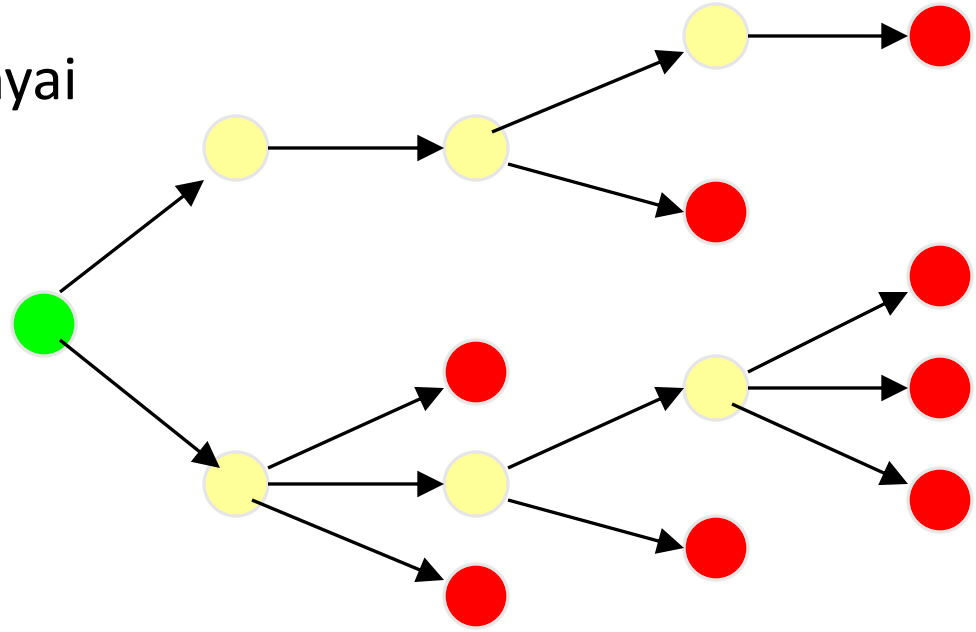
- Semua kemungkinan solusi dari persoalan disebut **ruang solusi** (*solution space*).

- Tinjau *Knapsack* 0/1 untuk $n = 3$.
- Solusi persoalan dinyatakan sebagai $X = (x_1, x_2, x_3)$ dengan $x_i \in \{0,1\}$.
- Ruang solusinya adalah:

$$\{(0, 0, 0),\ (0, 1, 0),\ (0, 0, 1), (1, 0, 0),$$
$$(1, 1, 0), (1, 0, 1), (0, 1, 1), (1, 1, 1)\ \}$$

- Ruang solusi diorganisasikan ke dalam struktur pohon berakar.
- 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*).

Sebuah pohon adalah sekumpulan simpul dan busur yang tidak mempunyai sirkuit



Ada tiga macam simpul:

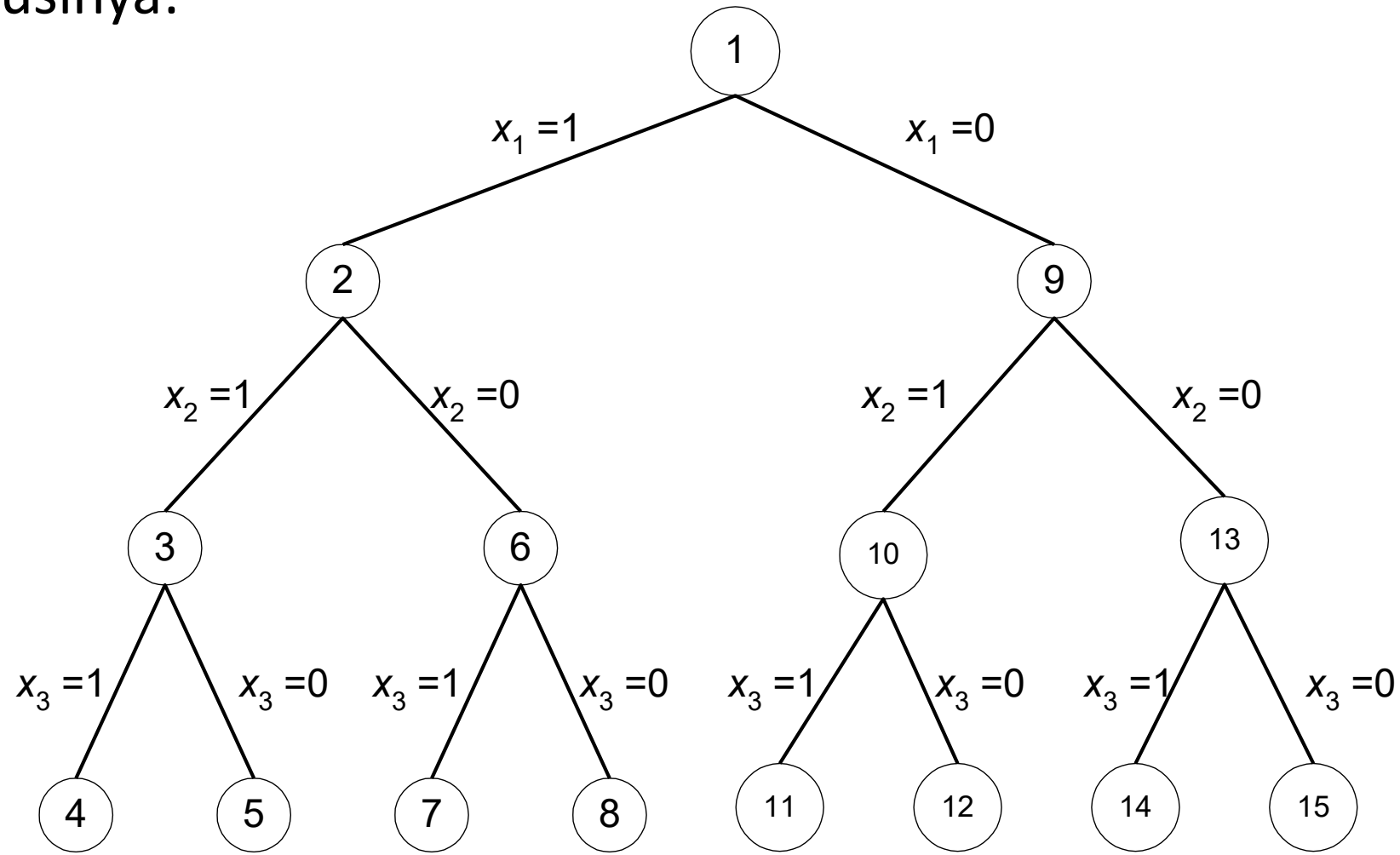
- Simpul akar
- Simpul dalam
- Simpul daun

Backtracking dapat dipandang sebagai pencarian di dalam pohon dari akar menuju daun (simpul solusi)

*) Sumber: *www.cis.upenn.edu/.../35-**backtracking**.ppt*

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

Ruang solusinya:



Catatan: simpul-simpul dinomori dengan aturan *depth-first search*

Prinsip Pencarian Solusi dengan Algoritma Runut-balik

- Solusi dicari dengan membangkitkan simpul-simpul status sehingga menghasilkan lintasan dari akar ke daun.

- Aturan pembangkitan simpul yang dipakai adalah mengikuti aturan *depht-first order* (DFS).

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

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

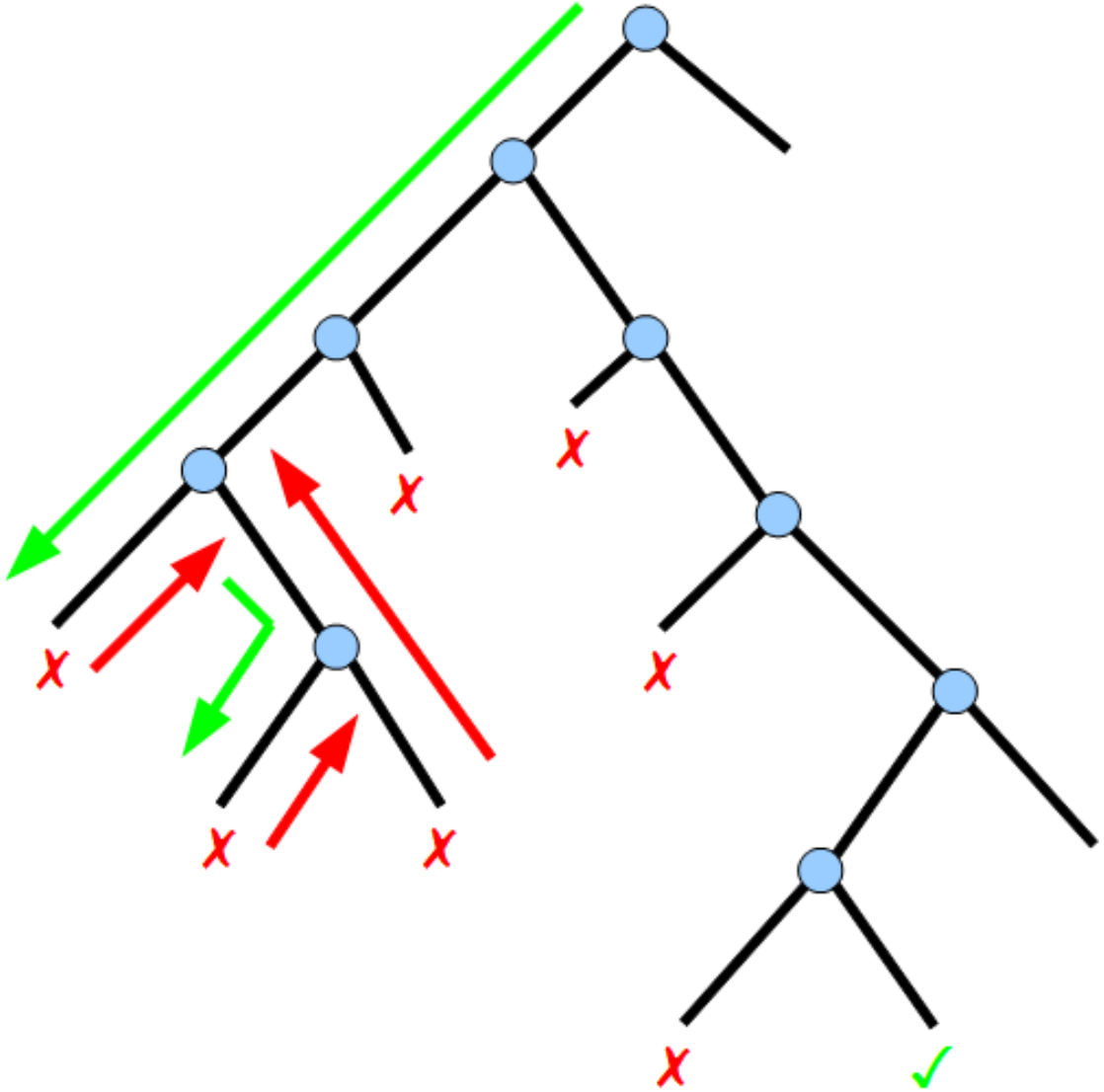
- Tiap kali simpul-E diperluas, lintasan yang dibangun olehnya bertambah panjang.

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

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

- Ketika sebuah simpul mati, maka secara implist kita telah memangkas (*pruning*) simpul-simpul anaknya.

- Jika pembentukan lintasan berakhir dengan simpul mati, maka proses pencarian *backtrack* ke simpul pada aras diatasnya
- Lalu, teruskan dengan membangkitkan simpul anak yang lainnya.
- Selanjutnya simpul ini menjadi simpul-E yang baru.
- Pencarian dihentikan bila kita telah sampai pada *goal node*.



Sumber: http://www.w3.org/2011/Talks/01-14-steven-phenotype/

- Tinjau persoalan 1/0 *Knapsack* 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 konstrain dapat dijadikan sebagai *bounding function*:

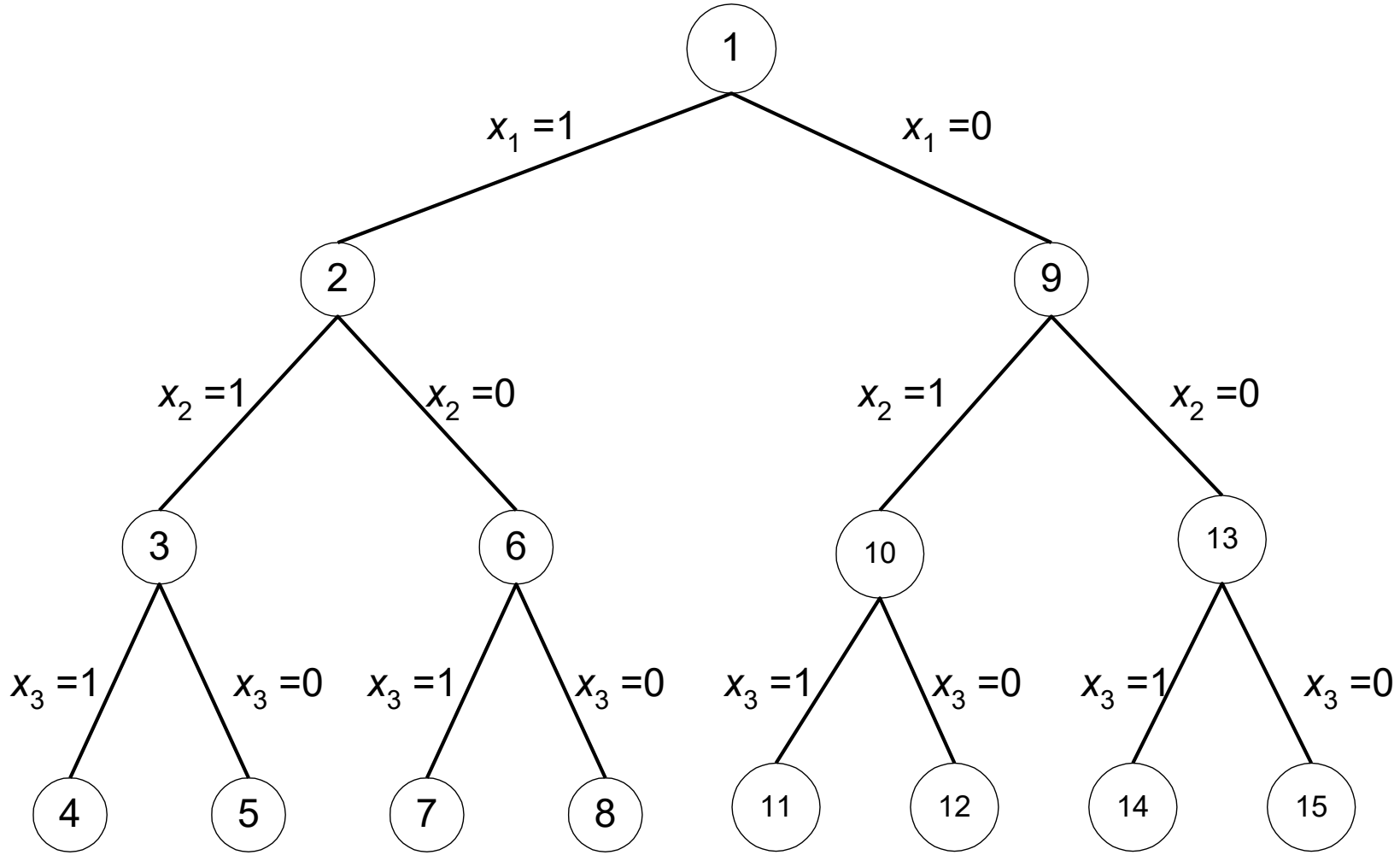
$$\sum_{i=1}^{k} w_i x_i \leq M$$

yaitu,

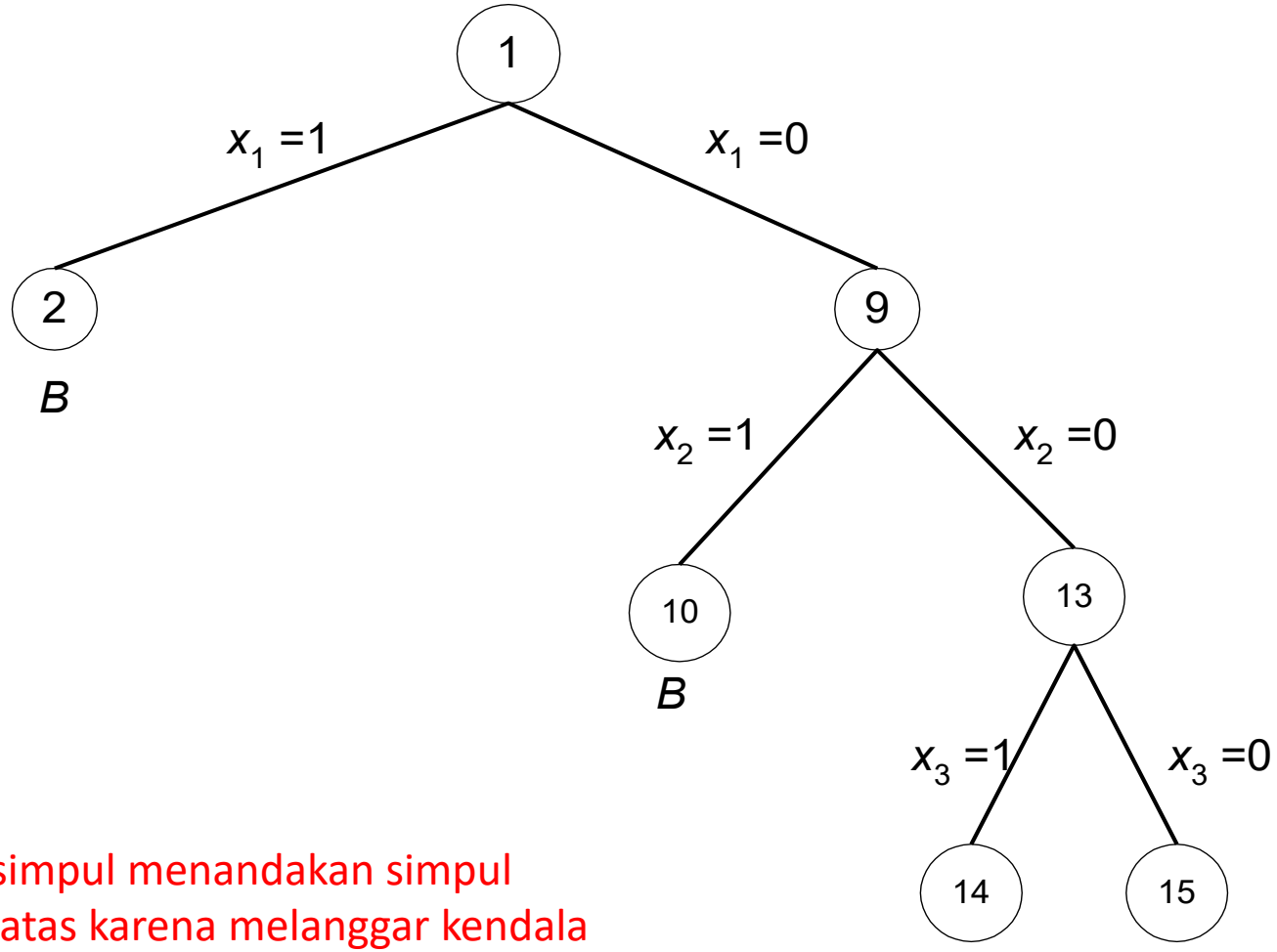
$B(x_1, x_2, \ldots, x_k)$ = *true* jika dan hanya jika $\sum_{i=1}^{k} w_i x_i \leq M$

artinya $x_1, x_2, \ldots, x_k$ mengarah ke simpul solusi (*goal node*) sehingga simpul tersebut dapat diteruskan untuk dikembangkan, tetapi jika $\sum_{i=1}^{k} w_i x_i > M$ maka simpul tersebut dibunuh.

Di bawah ini adalah pohon ruang status lengkap. Setiap lintasan dari akar ke daun merupakan sebuah kemungkinan solusi. Pada metode *exhaustive search* (*brute force*), semua lintasan perlu dievaluasi untuk menentukan solusi optimal

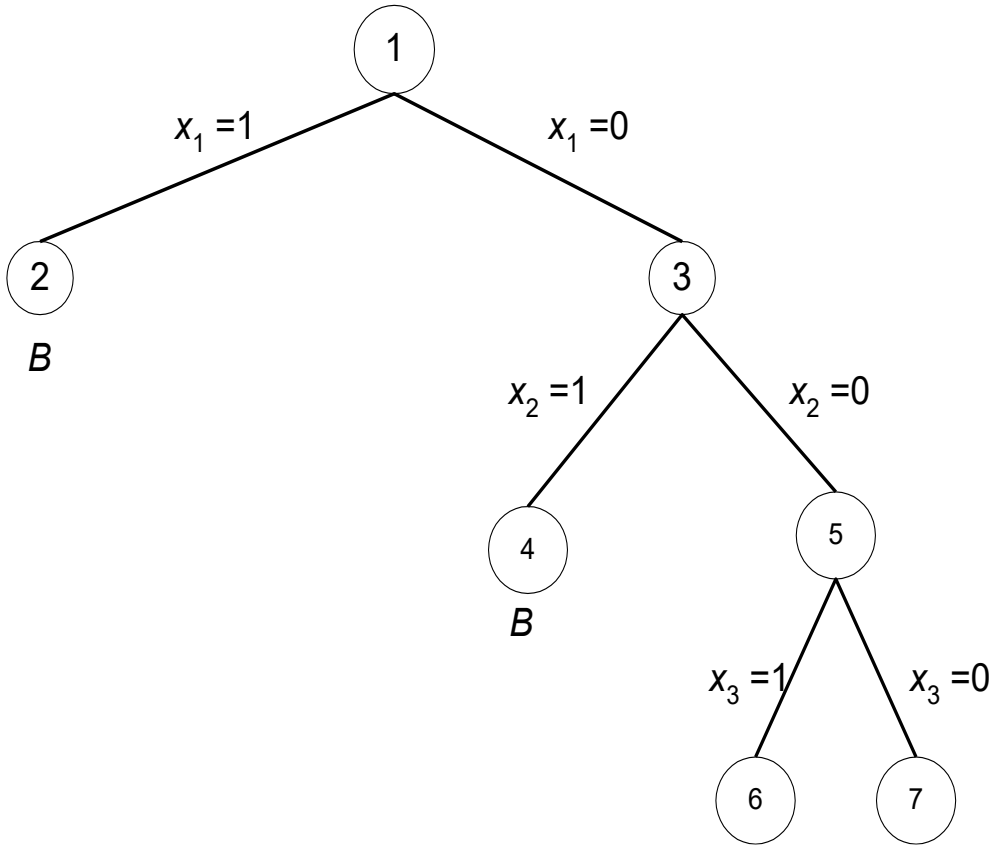


Pada metode *backtracking*, simpul-simpul dibangun satu per satu. Simpul yang memiliki nilai *bounding function* false dimatikan. Di bawah ini adalah pohon dinamis yang dibentuk selama pencarian untuk persoalan 1/0 *Knapsack* dengan $n = 3$, $M = 30$, $w = (35, 32, 25)$ dan $p = (40, 25, 50)$



Keterangan: huruf B di bawah suatu simpul menandakan simpul tersebut dimatikan oleh fungsi pembatas karena melanggar kendala

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]. Diasumsikan x[1], x[2], ..., x[k – 1] sudah
ditentukan nilainya.
Luaran: semua solusi x = (x[1], x[2], ..., x[n])
}
Algoritma:
  for setiap x[k] ∈ T(x[1], x[2], …, x[k – 1]) do
      if B(x[1], x[2], ... ,x[k]) = true then
          if (x[1], x[2], ... , x[k]) adalah lintasan dari akar ke simpul solusi then
              write(x[1], x[2], …, x[k])      { cetak solusi }
          endif
          if k < n then
              RunutBalikR(k+1)    { tentukan nilai untuk x[k+1]}
          endif
      endif
  endfor
```

Pemanggilan pertama kali: RunutBalikR(1)

- Setiap simpul di dalam pohon ruang status (kecuali simpul daun) berasosiasi dengan sebuah pemanggilan rekursif.

- Jika jumlah simpul dalam pohon ruang status adalah 2^n atau $n!$, maka pada 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.

Skema Umum Algoritma Runut-Balik (versi iteratif)

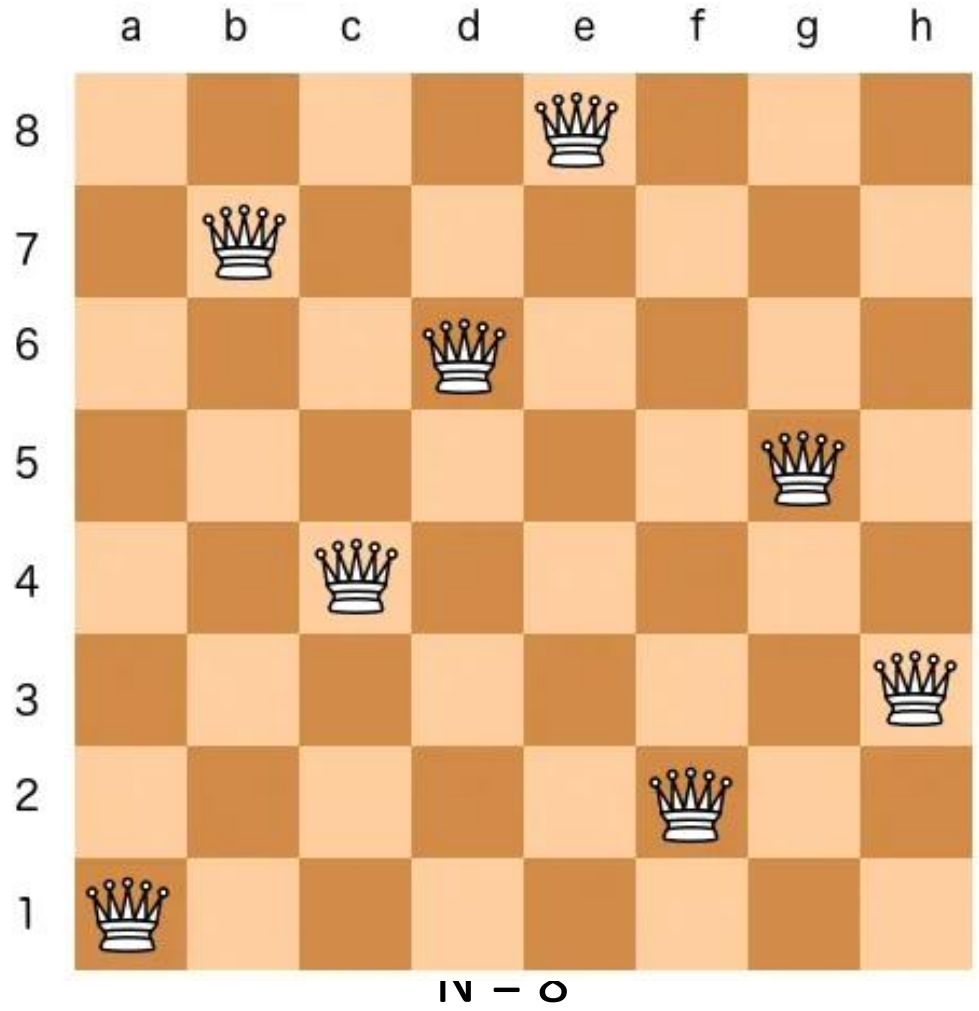
procedure *RunutBalikI*(**input** k : **integer**)
*{Mencari **semua** solusi persoalan dengan metode runut-balik; skema iteratif*
Masukan: k, yaitu indeks komponen vektor solusi, x[k]. Diasumsikan x[1], x[2], ..., x[k – 1] sudah ditentukan nilainya.
Luaran: semua solusi x = (x[1], x[2], ..., x[n])
}
Algoritma:
while $k \neq 0$ **do**
if terdapat nilai $x[k]$ yang belum dicoba sedemikian sehingga $x[k] \in T(x[1], x[2], \ldots, x[k-1])$ **and**
$B(x[1], x[2], \ldots, x[k])$ = **true then**
if $(x[1], x[2], \ldots, x[k])$ adalah lintasan dari akar ke simpul solusi **then**
write$(x[1], x[2], \ldots, x[k])$ *{ cetak solusi }*
endif
$k \leftarrow k + 1$ *{ tentukan nilai x[k] selanjutnya }*
else
$k \leftarrow k - 1$
endif
endwhile

Pemanggilan pertama kali: RunutBalikI(1)

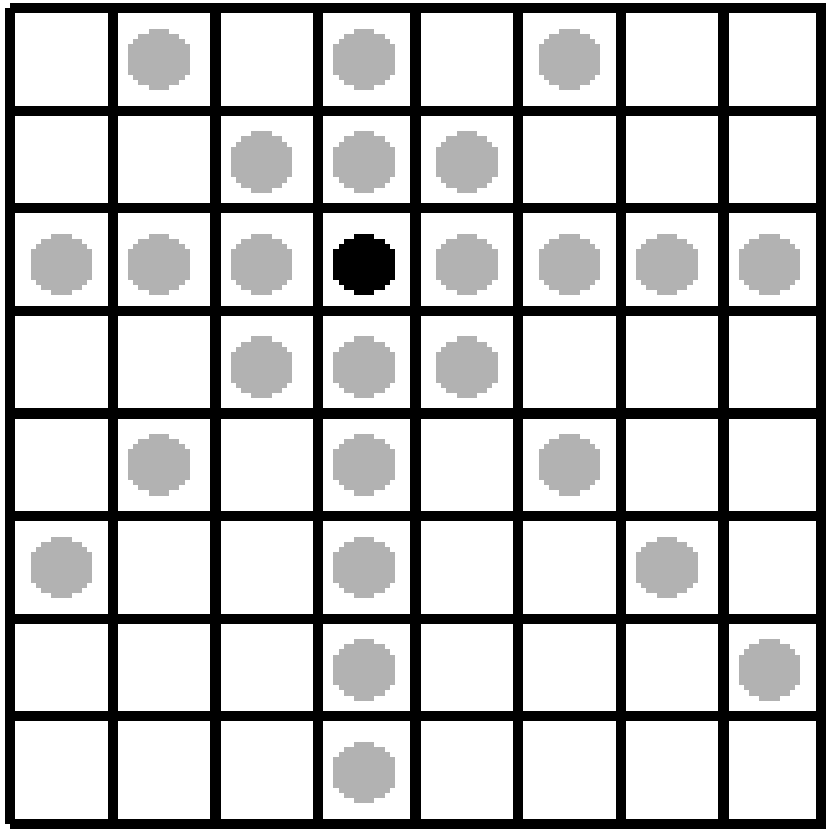
1. Persoalan N-Ratu (*The N-Queens Problem*)

- Diberikan sebuah papan catur yang berukuran $N \times N$ dan N buah ratu. Bagaimanakah cara 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?

  (tidak saling *attack* satu sama lain)



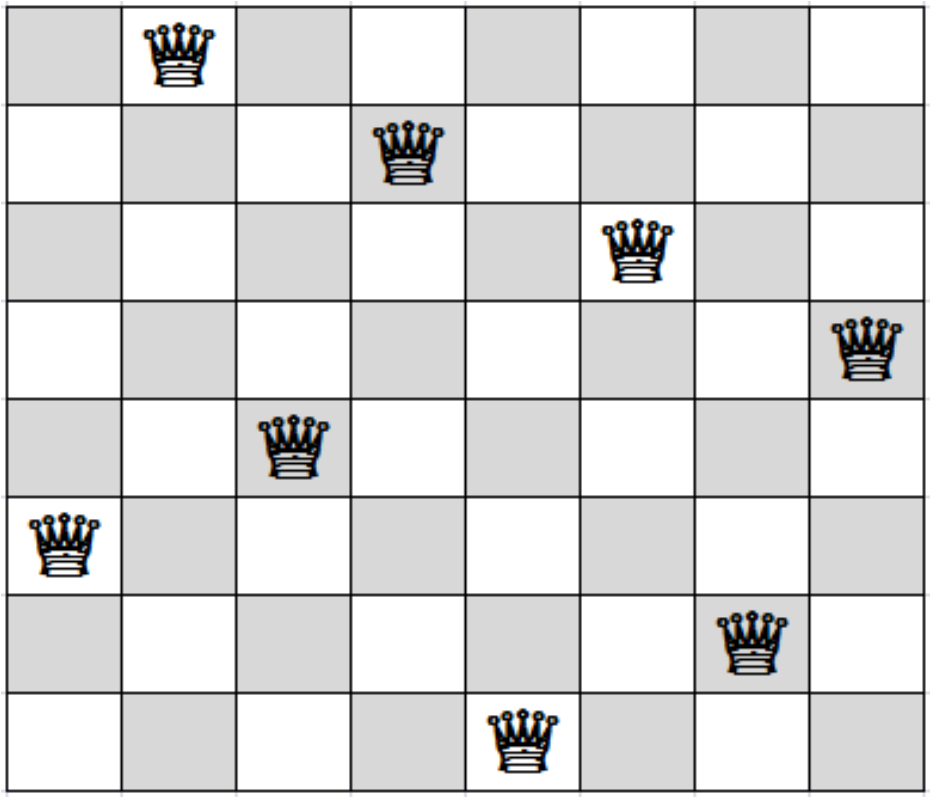
N = 8

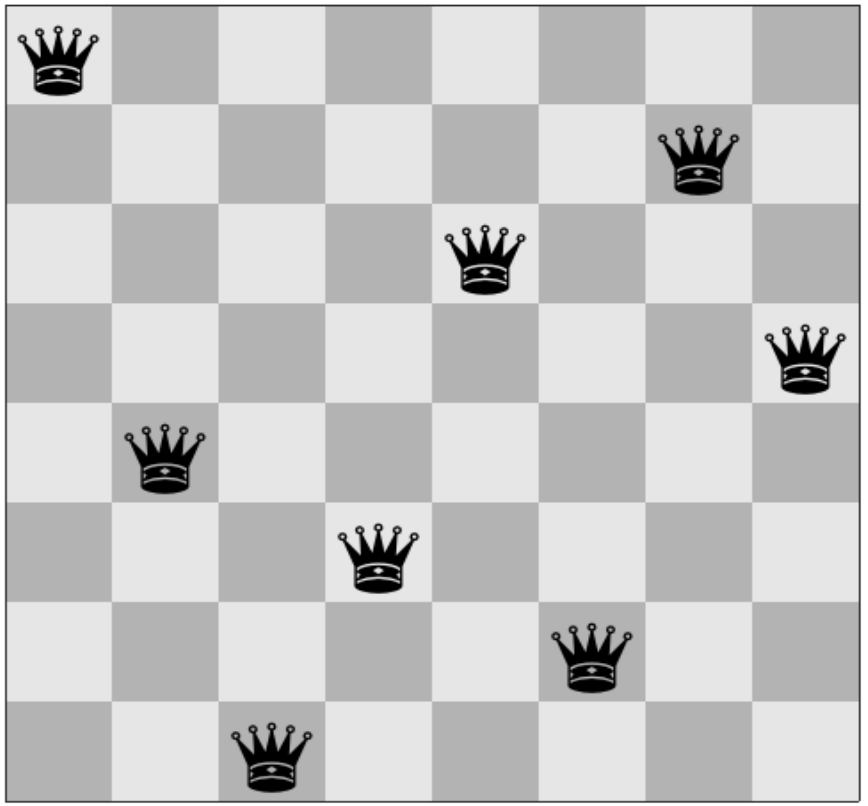


Penempatan ratu (dilambangkan dot hitam) sedemikian sehingga

- tidak boleh ada ratu lain pada baris yang sama,
- tidak boleh ada ratu lain pada kolom yang sama
- tidak boleh ada ratu lain pada diagonal yang sama

Contoh 2 buah solusi *8-queens problem*:





- The puzzle was originally proposed in 1848 by the chess player Max Bezzel, and over the years, many mathematicians, including Gauss, have worked on this puzzle and its generalized N-queens problem. The first solutions were provided by Franz Nauck in 1850. Nauck also extended the puzzle to n-queens problem (on an N×N board—a chessboard of arbitrary size).

- Edsger Dijkstra used this problem in 1972 to illustrate the power of what he called structured programming. He published a highly detailed description of the development of a depth-first backtracking algorithm.[2]

(Sumber: laman Wikipedia)

Contoh *N-queens problem* dalam dunia nyata:

Masalah *N-Queens* adalah abstraksi dari penempatan elemen tanpa konflik seperti:

1. Penjadwalan tugas/shift

- Menjadwalkan pekerja dalam shift kerja sehingga tidak ada dua orang di shift yang sama dengan peran yang sama
- tidak ada tabrakan waktu, setiap shift punya orang yang tepat.
- Ini mirip dengan menempatkan "queen" di papan — satu di setiap baris dan kolom, tanpa bentrok.

2. Penempatan menara pemancar (frekuensi radio / sinyal)

- Dalam jaringan komunikasi, misalnya menara pemancar (seperti BTS), menara harus diletakkan sehingga tidak saling ganggu (interferensi),
- Area cakupan tidak tumpang tindih secara buruk
- Mirip dengan *N-Queens problem*: setiap menara harus "tidak menyerang" yang lain, alias tidak mengganggu sinyal.

3. Penempatan pesawat di bandara

- Menjadwalkan keberangkatan/pendaratan pesawat
- Tidak ada dua pesawat yang menggunakan landasan pada waktu yang sama,
- Jalur lepas landas dan parkir tidak boleh bertabrakan.

5. Penempatan jalur robot

Robot-robot yang harus mengambil barang dari rak-rak berbeda di gudang.

- Robot harus diprogram agar jalurnya tidak tabrakan (*collision-free pa*th),
- Masing-masing robot harus mengakses rak berbeda tanpa konflik.

Penyelesaian N-queens problem dengan algoritma Brute-force (kasus N = 8):

a) Brute Force 1

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

- Penempatan ratu dilakukan secara acak pada petak-petak papan catur

- Terdapat $C(64, 8) = 4.426.165.368$ kemungkinan solusi yang perlu dievaluasi.

b) Brute Force 2

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

- Jumlah kemungkinan solusi yang diperiksa berkurang menjadi

$$8^8 = 16.777.216$$

```
procedure Ratu1
{Mmencari semua solusi penempatan delapan ratu pada petak-petak papan catur yang  berukuran 8 x 8  }
Kamus
   i1, i2, i3, i4, 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  i8←1 to 8 do
                            { periksa apakah i1, i2, i3, i4, i5, i6, i7, i8 merupakan solusi }
                            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*:

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

  x_i menyatakan kolom kedudukan ratu pada baris ke-*i*

  Contoh solusi: X = (2, 4, 6, 8, 3, 1, 7, 5)
  X = (8, 3, 1, 6, 2, 5, 7, 4)

- Vektor solusi X merupakan permutasi bilangan 1, 2, 3, 4, 5, 6, 7, 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  }

Kamus
   X : array[1..n] of integer
   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
         write(x[1], x[2], …, x[k])      { cetak solusi }
      endif

     i←i+1   { ulangi untuk permutasi berikutnya }
  until  i > n
```

Penyelesaian N-queens problem dengan Algoritma Runut-balik:

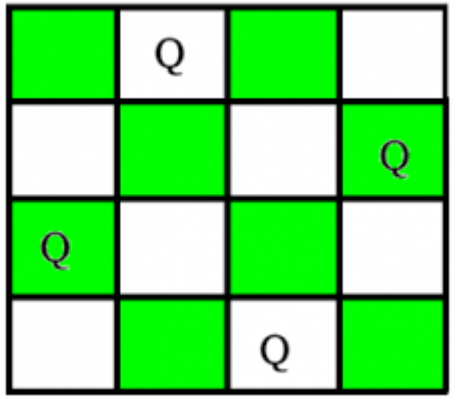
- Algoritma runut-balik memperbaiki algoritma *brute force* 3 (*exhaustive search*).

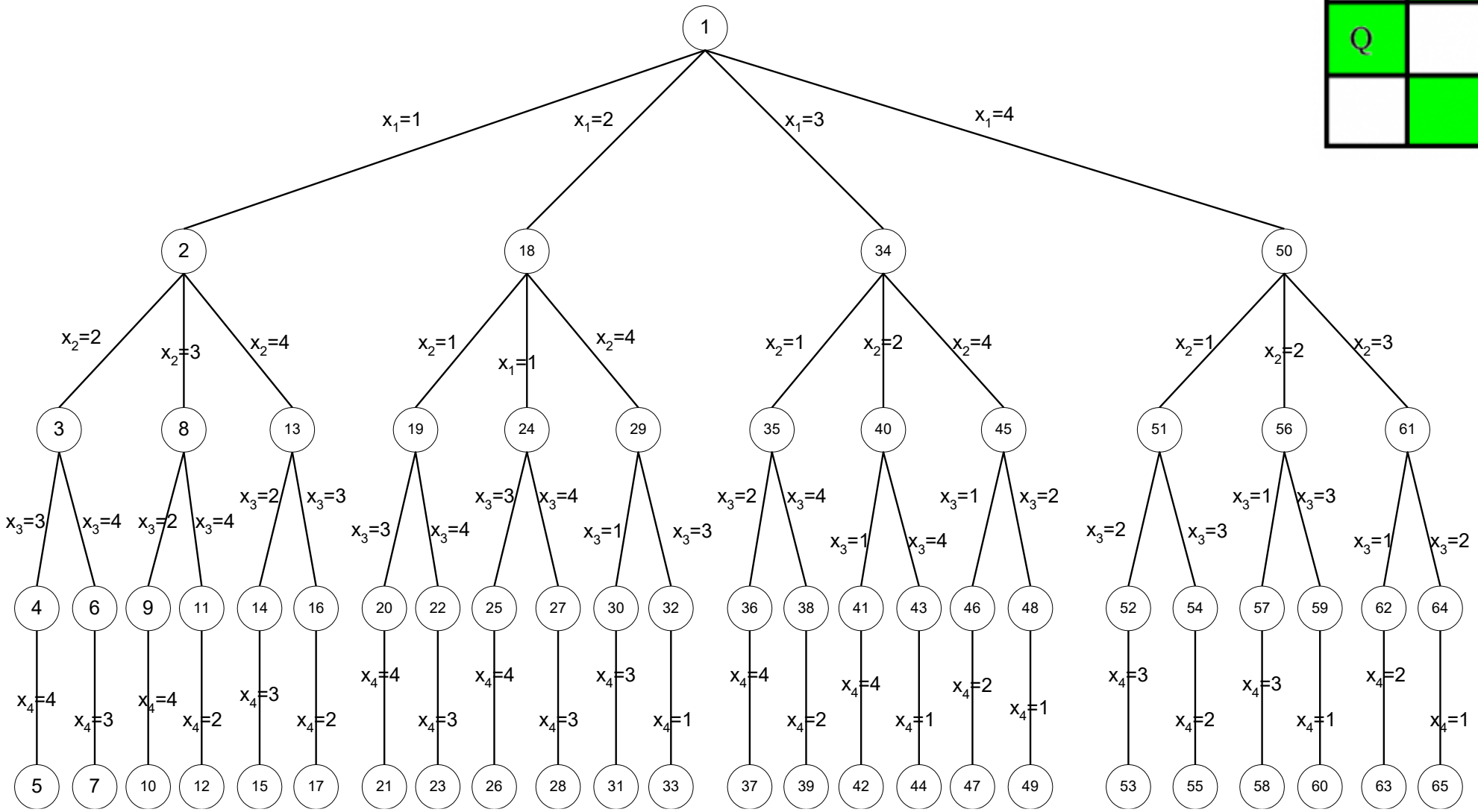
- Ruang solusinya adalah semua permutasi dari angka-angka 1, 2, 3, 4, 5, 6, 7, 8.

- 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.

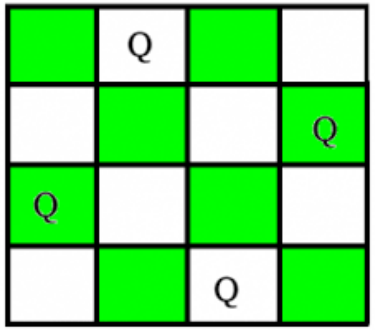
- Untuk penyederhanaan visualisasi, tinjau persoalan 4-Ratu

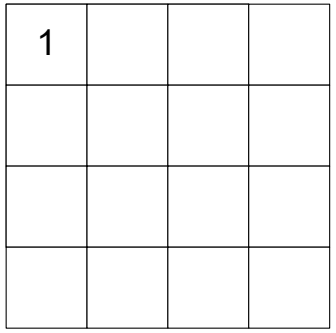
Contoh: Pohon ruang-status lemgkap persoalan 4-Ratu



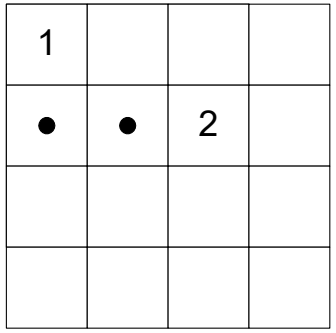


Contoh solusi runut-balik persoalan 4-Ratu:

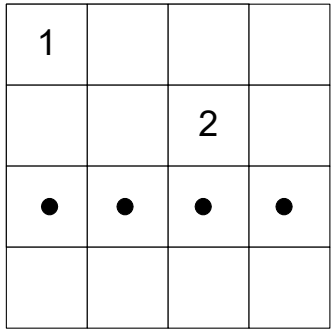




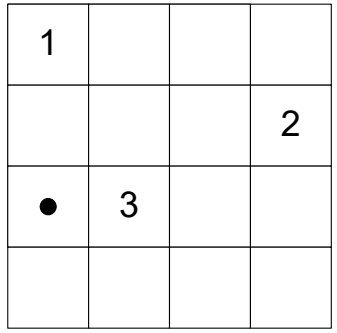
(a)



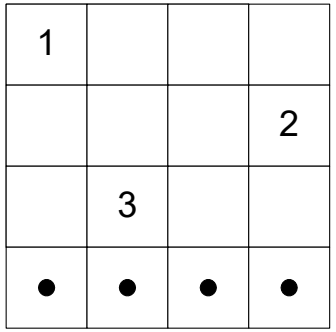
(b)



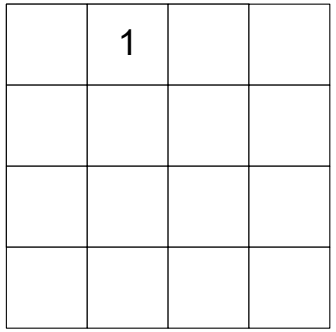
(c)



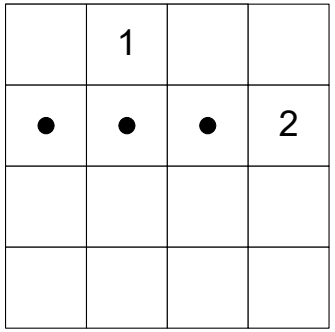
(d)



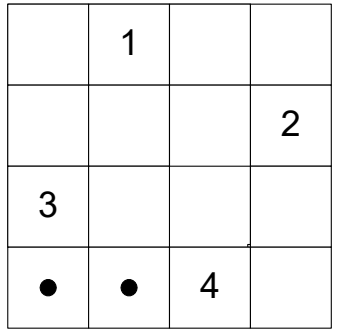
(e)



(f)

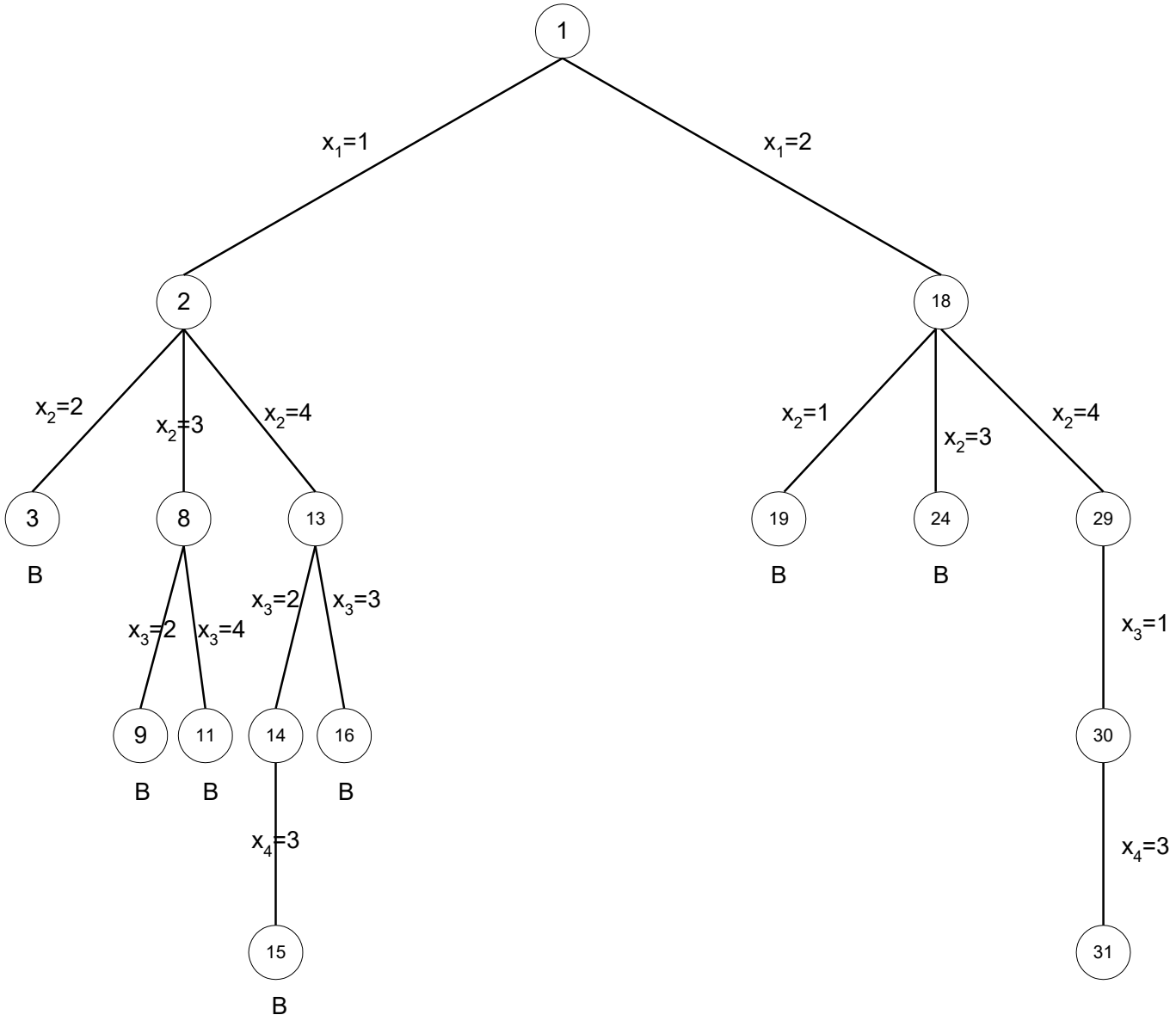


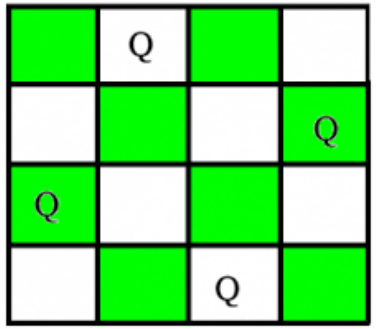
(g)



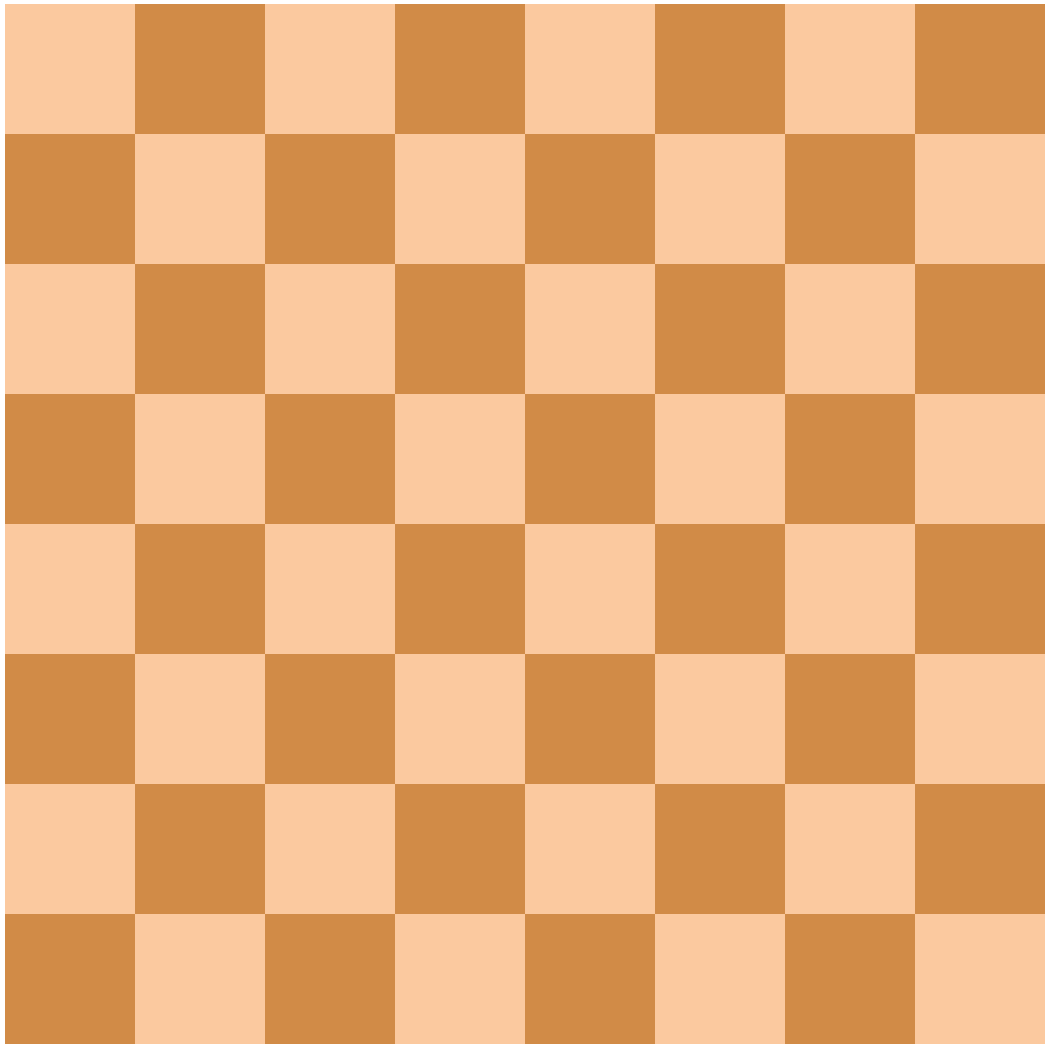
(h)

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



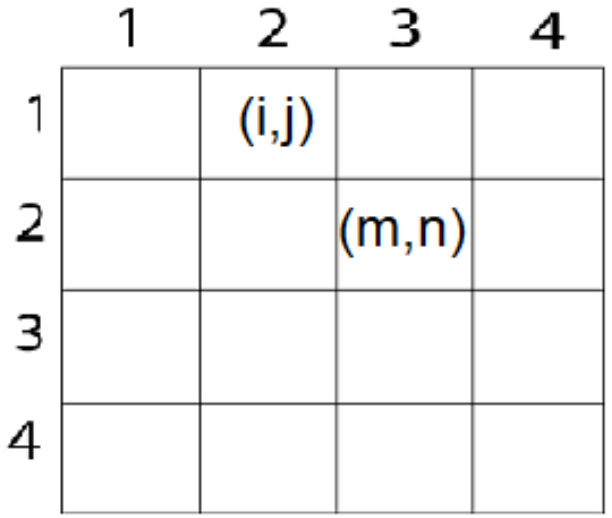


Ket: Huruf B di bawah suatu simpul menyatakan bahwa simpul tersebut dibunuh oleh *bounding function*



Animasi penempatan 8 ratu pada papan catur 8 x 8

Algoritma runut-balik untuk persoalan 8-Ratu



- Tinjau dua posisi ratu pada (i, j) dan (m, n)
- Dua buah ratu terletak pada baris yang sama, berarti

  $i = m$
- Dua buah ratu terletak pada kolom yang sama, berarti

  $j = n$
- Dua buah ratu terletak pada diagonal yang sama, berarti

  ↘ $i - j = m - n$ atau ↙ $i + j = m + n$

  $\Leftrightarrow i - m = j - n$ atau $m - i = j - n$

  $\Leftrightarrow |j - n| = |i - m|$

- Solusi dinyatakan sebagai vector x = (x[1], x[2], …, x[N])

  x[i] menyatakan kolom penempatan ratu pada baris i

- Fungsi pembangkit T(k) :

  $x[k] \leftarrow i, \quad i = 1, 2, 3, \ldots, N$

- Fungsi pembatas B(x[1], x[2], …, x[k]):

  - true jika tidak ada ratu lain pada kolom x[k] atau pada diagonal yang sama
  - false jika ada

```
function Tempat(input k , i : integer)→boolean
{true jika ratu dapat ditempatkan pada baris ke-k dan kolom ke-i, false jika tidak}

Kamus
    j : integer

Algoritma:
  for j ← 1 to k – 1  do      { periksa ratu-ratu mulai dari baris ke-1 sampai baris k – 1 }
      if  (x[j] = i)          {apakah ada dua buah ratu terletak pada kolom yang sama?}
            or                                {atau }
         (ABS(x[j] – i]) = ABS(j – k )) {  apakah ada dua ratu pada diagonal yang sama?}
      then
          return false      { ratu tidak dapat ditempatkan pada baris ke-k dan kolom ke-i }
      endif
  endfor
  return true
```

Function *Tempat* adalah representasi *bounding function*

Algoritma:

- Inisialisasi *x*[1], *x*[2], …, *x*[*N*] dengan 0 sebagai berikut:

for $i \leftarrow 1$ **to** N **do**

$x[i] \leftarrow 0$

endfor

- Panggil prosedur *NratuRec*(1, *N*)

Versi rekursif

```
procedure NratuRec(input k, N : integer)
{ Algoritma rekursif untuk menghasilkan semua solusi penempatan N buah ratu pada petak papan catur N x N
tanpa melanggar kendala;
Masukan: N = jumlah ratu
Luaran: semua solusi x = (x[1], x[2], ..., x[N]) dicetak ke layar. }

Deklarasi
   i : integer

Algoritma:
   for i ← 1 to N do
      if Tempat(k, i) then   { periksa apakah ratu ke-k dapat   ditempatkan pada baris k dan kolom i }
         x[k] ← i
         if k = N then                    { apakah solusi sudah lengkap? }
            write(x[1], x[2], …, x[k])     { cetak solusi }
         else
            Nratu_R(k + 1, N)
         endif
      endif
   endfor
```

Versi iteratif

```
procedure NratuIter(input k : integer, N : integer)
{ Algoritma iteratif untuk menghasilkan semua solusi penempatan N buah ratu pada petak papan catur N x N
tanpa melanggar kendala;
Masukan: N = jumlah ratu
Luaran: semua solusi x = (x[1], x[2], …, x[N]) dicetak ke layar.   }
Deklarasi
    i : integer
    stop : boolean
Algoritma:
    i ← 0
    while k ≠ 0 do
      i ← i + 1
      stop ← false
      while (i <= N) and (not stop)  do
         if Tempat(k, i) then  {periksa apakah ratu dapat ditempatkan pada baris k dan kolom i}
            x[k] ← i
            if k = N  then { apakah solusi sudah lengkap?}
              write(x[1], x[2], …, x[k])      { cetak solusi }
              stop ← true
            else
              k ← k + 1   { menempatkan ratu untuk baris berikutnya}
              i ← 1
            endif
         else
            i ← i+1
         endif
      endwhile  {i>N}
      k ← k - 1     { runut-balik ke baris sebelumnya}
      i ← x[k]
    endwhile
```

Lampiran Program C++

Versi rekursif

```
/ Program N-Queen Problem dengan algoritma backtracking
// Versi rekursif

#include <iostream>
#include <cmath>
using namespace std;

int x[9], sol;

bool Tempat(int k, int i)
{
    int j;

    for(j=1; j<=k-1;j++){
       if ((x[j]==i) || (abs(x[j]-i)==abs(j-k))) {
        return false;
       }
    }
    return true;
}
```

```
void NratuRec(int k, int N) {
  int i,m;

  for(i=1;i<=N;i++) {
    if (Tempat(k,i)){
       x[k]=i;
       if (k==N) {
          sol = sol + 1;
          cout << "Solusi ke-" << sol << ": ";
          for(m=1;m<=N;m++) cout << x[m] << " "; cout << endl;
      }
      else
         NratuRec(k+1,N);
    }
  }
}

int main() {
 int j, N;

 N = 4;
 sol = 0;
 cout << "N = " << N << endl;
 for (j=1;j<=N; j++) { x[j]=0; }
 NratuRec(1,N);
 return 0;
 }
```

Versi iteratif

```
// Program N-Queen Problem dengan algoritma backtracking
// Versi iteratif
#include <iostream>
#include <cmath>
using namespace std;

int x[9], sol;

bool Tempat(int k, int i)
{
    int j;

    for(j=1; j<=k-1;j++){
       if ((x[j]==i) || (abs(x[j]-i)==abs(j-k))) {
        return false;
       }
    }
    return true;
}

void NratuIter(int k, int N)
{
  int i,m;
  bool stop;
```

```
  i = 0;
  sol = 0;
  while (k>0) {
    i = i + 1;
    stop = false;
    while (i<=N && !stop){
      if (Tempat(k,i)) {
        x[k] = i;
        if (k==N) {
          sol = sol + 1;
          cout << "Solusi ke-" << sol << ": ";
          for(m=1;m<=N;m++) cout << x[m] << "  "; cout << endl;
          stop = true;
        }
        else {
          k = k + 1;
          i = 1;
        }
      }
      else
        i = i + 1;
    }
    k = k-1;
    i = x[k];
  }
}
```

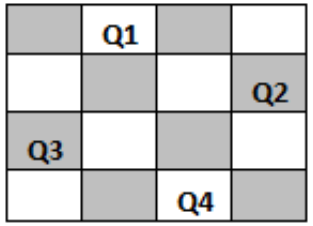
```
int main() {
 int j, N;

 N = 8;
 cout << "N = " << N << endl;
 for (j=1;j<=N; j++) {
    x[j]=0;
 }
 NratuIter(1,N);
 return 0;
 }
```

Hasil *run* program untuk *N* = 4

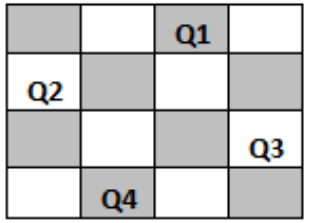
```
Command Prompt

D:\IF2211 Strategi Algoritma\2021>g++ NratuR.cpp

D:\IF2211 Strategi Algoritma\2021>a
N = 4
Solusi ke-1: 2 4 1 3
Solusi ke-2: 3 1 4 2

D:\IF2211 Strategi Algoritma\2021>
```

Cara membaca hasil *run* ini:

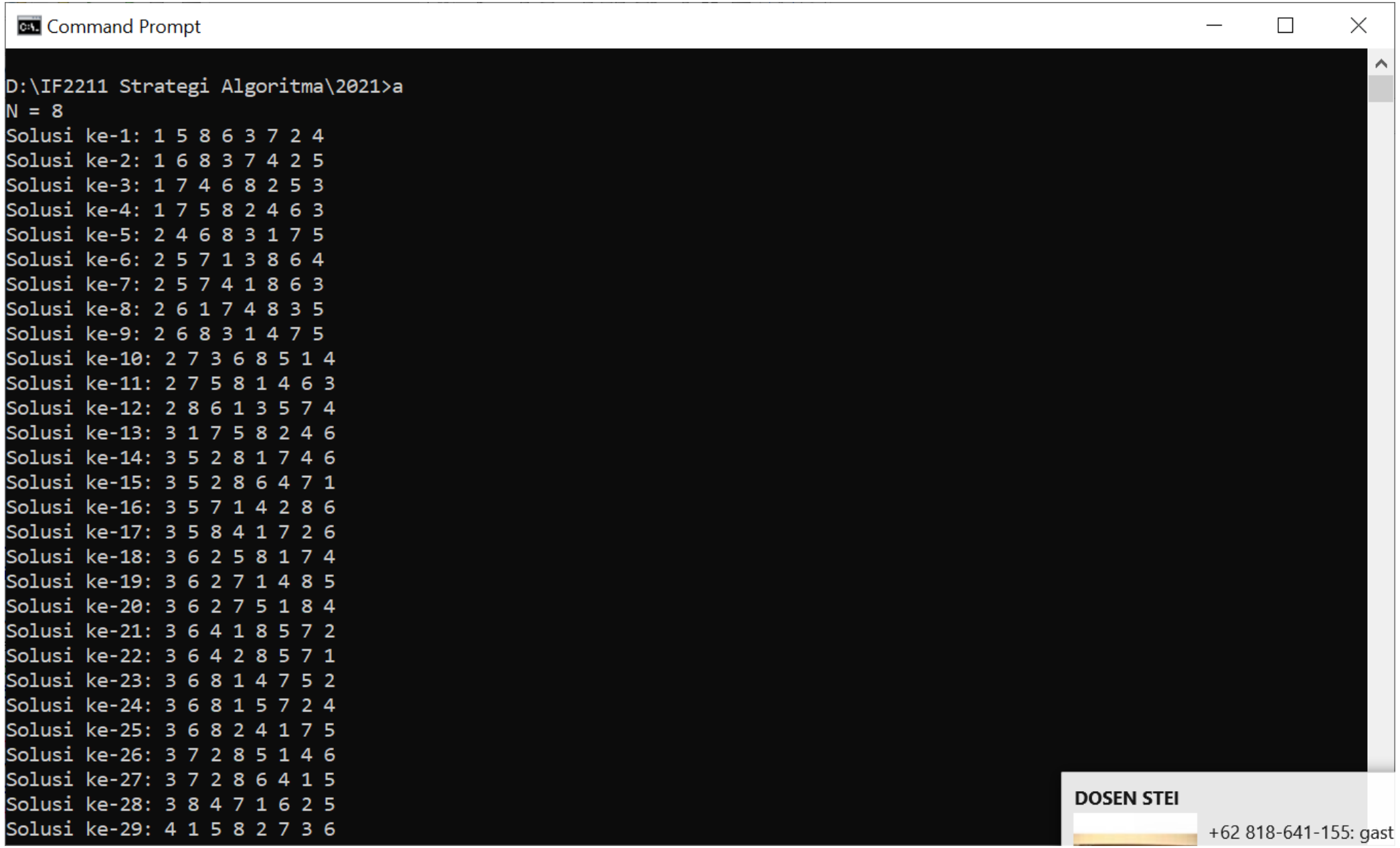
Solusi ke-1: 2 4 1 3, artinya pada baris ke-1 ratu ditempatkan pada kolom ke-2
pada baris ke-2 ratu ditempatkan pada kolom ke-4
pada baris ke-3 ratu ditempatkan pada kolom ke-1
pada baris ke-4 ratu ditempatkan pada kolom ke-3

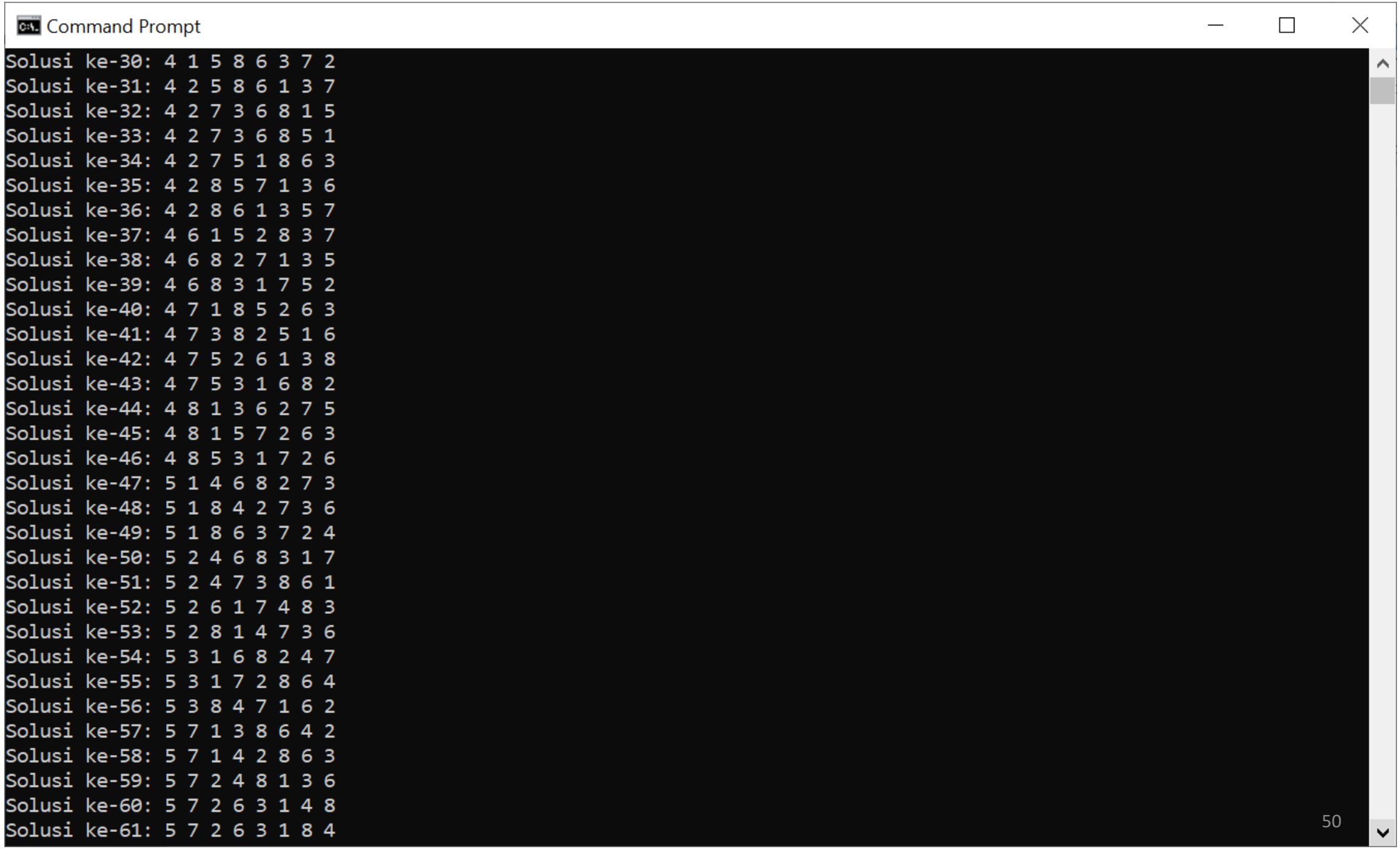


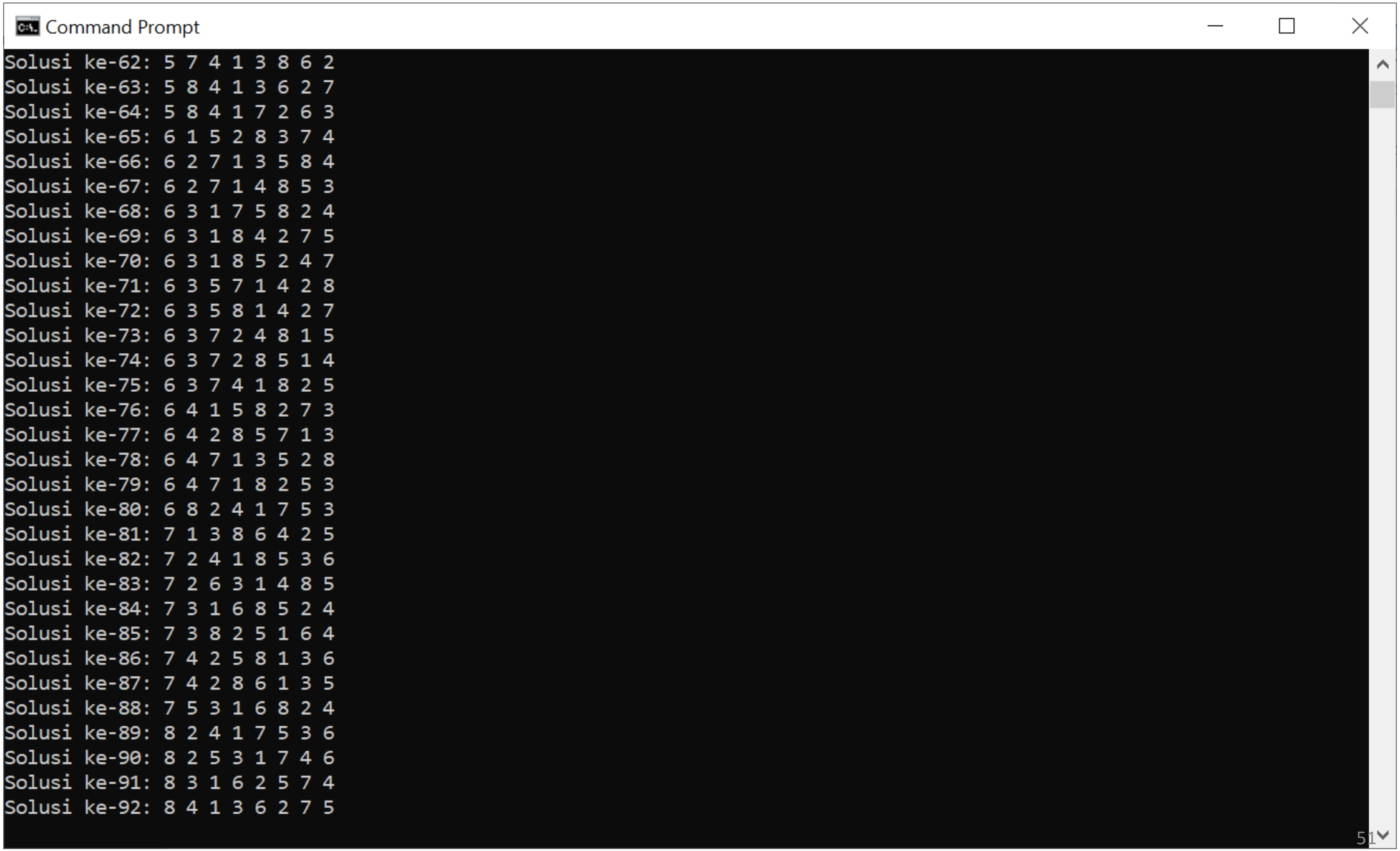
Solusi ke-2: 3 1 4 2, artinya pada baris ke-1 ratu ditempatkan pada kolom ke-3
pada baris ke-2 ratu ditempatkan pada kolom ke-1
pada baris ke-3 ratu ditempatkan pada kolom ke-4
pada baris ke-4 ratu ditempatkan pada kolom ke-2



Hasil *run* program untuk *N* = 8




Command Prompt
Solusi ke-30: 4 1 5 8 6 3 7 2
Solusi ke-31: 4 2 5 8 6 1 3 7
Solusi ke-32: 4 2 7 3 6 8 1 5
Solusi ke-33: 4 2 7 3 6 8 5 1
Solusi ke-34: 4 2 7 5 1 8 6 3
Solusi ke-35: 4 2 8 5 7 1 3 6
Solusi ke-36: 4 2 8 6 1 3 5 7
Solusi ke-37: 4 6 1 5 2 8 3 7
Solusi ke-38: 4 6 8 2 7 1 3 5
Solusi ke-39: 4 6 8 3 1 7 5 2
Solusi ke-40: 4 7 1 8 5 2 6 3
Solusi ke-41: 4 7 3 8 2 5 1 6
Solusi ke-42: 4 7 5 2 6 1 3 8
Solusi ke-43: 4 7 5 3 1 6 8 2
Solusi ke-44: 4 8 1 3 6 2 7 5
Solusi ke-45: 4 8 1 5 7 2 6 3
Solusi ke-46: 4 8 5 3 1 7 2 6
Solusi ke-47: 5 1 4 6 8 2 7 3
Solusi ke-48: 5 1 8 4 2 7 3 6
Solusi ke-49: 5 1 8 6 3 7 2 4
Solusi ke-50: 5 2 4 6 8 3 1 7
Solusi ke-51: 5 2 4 7 3 8 6 1
Solusi ke-52: 5 2 6 1 7 4 8 3
Solusi ke-53: 5 2 8 1 4 7 3 6
Solusi ke-54: 5 3 1 6 8 2 4 7
Solusi ke-55: 5 3 1 7 2 8 6 4
Solusi ke-56: 5 3 8 4 7 1 6 2
Solusi ke-57: 5 7 1 3 8 6 4 2
Solusi ke-58: 5 7 1 4 2 8 6 3
Solusi ke-59: 5 7 2 4 8 1 3 6
Solusi ke-60: 5 7 2 6 3 1 4 8
Solusi ke-61: 5 7 2 6 3 1 8 4
50


Command Prompt
Solusi ke-62: 5 7 4 1 3 8 6 2
Solusi ke-63: 5 8 4 1 3 6 2 7
Solusi ke-64: 5 8 4 1 7 2 6 3
Solusi ke-65: 6 1 5 2 8 3 7 4
Solusi ke-66: 6 2 7 1 3 5 8 4
Solusi ke-67: 6 2 7 1 4 8 5 3
Solusi ke-68: 6 3 1 7 5 8 2 4
Solusi ke-69: 6 3 1 8 4 2 7 5
Solusi ke-70: 6 3 1 8 5 2 4 7
Solusi ke-71: 6 3 5 7 1 4 2 8
Solusi ke-72: 6 3 5 8 1 4 2 7
Solusi ke-73: 6 3 7 2 4 8 1 5
Solusi ke-74: 6 3 7 2 8 5 1 4
Solusi ke-75: 6 3 7 4 1 8 2 5
Solusi ke-76: 6 4 1 5 8 2 7 3
Solusi ke-77: 6 4 2 8 5 7 1 3
Solusi ke-78: 6 4 7 1 3 5 2 8
Solusi ke-79: 6 4 7 1 8 2 5 3
Solusi ke-80: 6 8 2 4 1 7 5 3
Solusi ke-81: 7 1 3 8 6 4 2 5
Solusi ke-82: 7 2 4 1 8 5 3 6
Solusi ke-83: 7 2 6 3 1 4 8 5
Solusi ke-84: 7 3 1 6 8 5 2 4
Solusi ke-85: 7 3 8 2 5 1 6 4
Solusi ke-86: 7 4 2 5 8 1 3 6
Solusi ke-87: 7 4 2 8 6 1 3 5
Solusi ke-88: 7 5 3 1 6 8 2 4
Solusi ke-89: 8 2 4 1 7 5 3 6
Solusi ke-90: 8 2 5 3 1 7 4 6
Solusi ke-91: 8 3 1 6 2 5 7 4
Solusi ke-92: 8 4 1 3 6 2 7 5

Bersambung