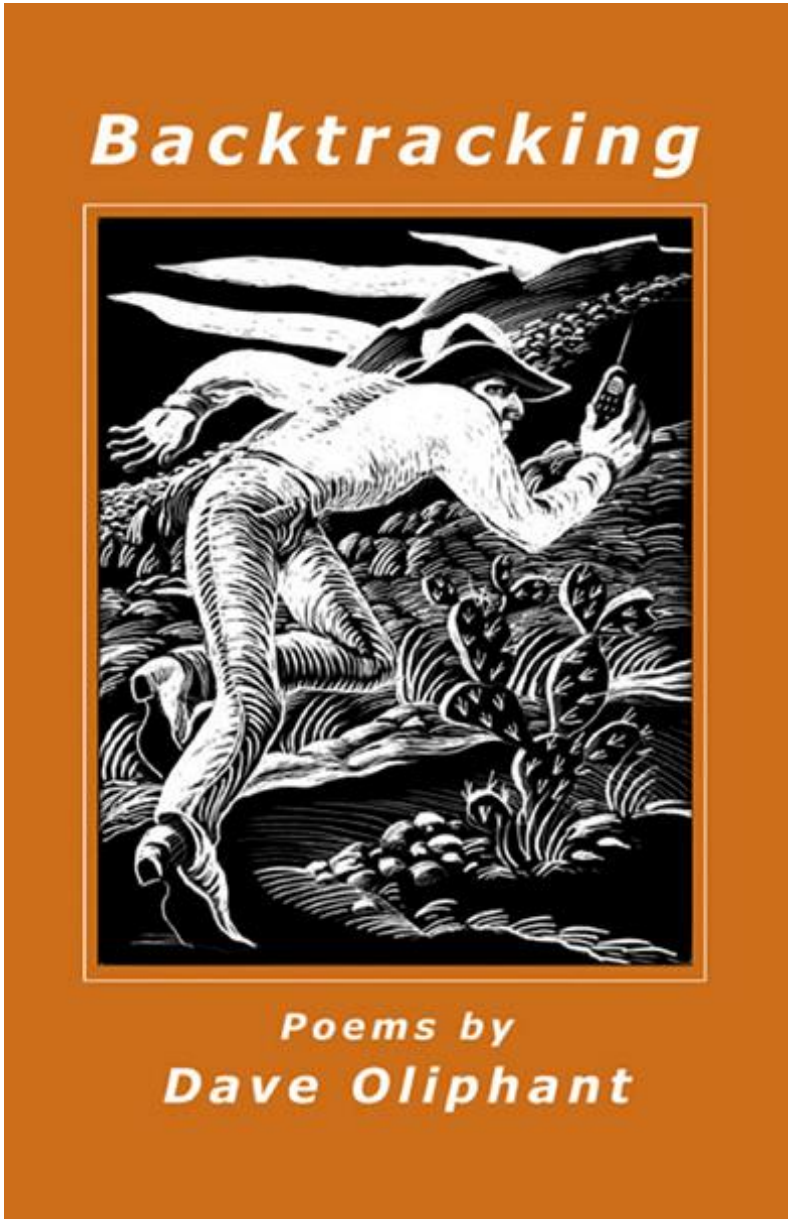
Algoritma Runut-balik (*Backtracking*)

Bahan Kuliah IF2211 Strategi Algoritma

Oleh: Rinaldi Munir

Program Studi Informatika STEI-ITB


Backtracking
Poems by
Dave Oliphant

Pendahuluan

- *Backtracking* dapat dipandang sebagai salah satu dari dua hal berikut:

  1. Sebagai sebuah fase di dalam algoritma traversal DFS → Sudah dijelaskan pada materi DFS/BFS.

  2. Sebagai sebuah metode pemecahan masalah yang mangkus, terstruktur, dan sistematis

Backtacking sebagai sebuah metode pemecahan masalah yang mangkus

- Algoritma runut-balik merupakan perbaikan dari *exhaustive search*.

- Pada *exhaustive search*, semua kemungkinan solusi dieksplorasi satu per satu.

- Pada *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 pada tahun 1950.

- R.J Walker, Golomb, dan Baumert menyajikan uraian umum tentang algoritma 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 predikat:

   $T(k)$

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

3. Fungsi pembatas

- Dinyatakan sebagai predikat

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

- 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*).
- 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:

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

- Ruang solusi diorganisasikan ke dalam struktur pohon.

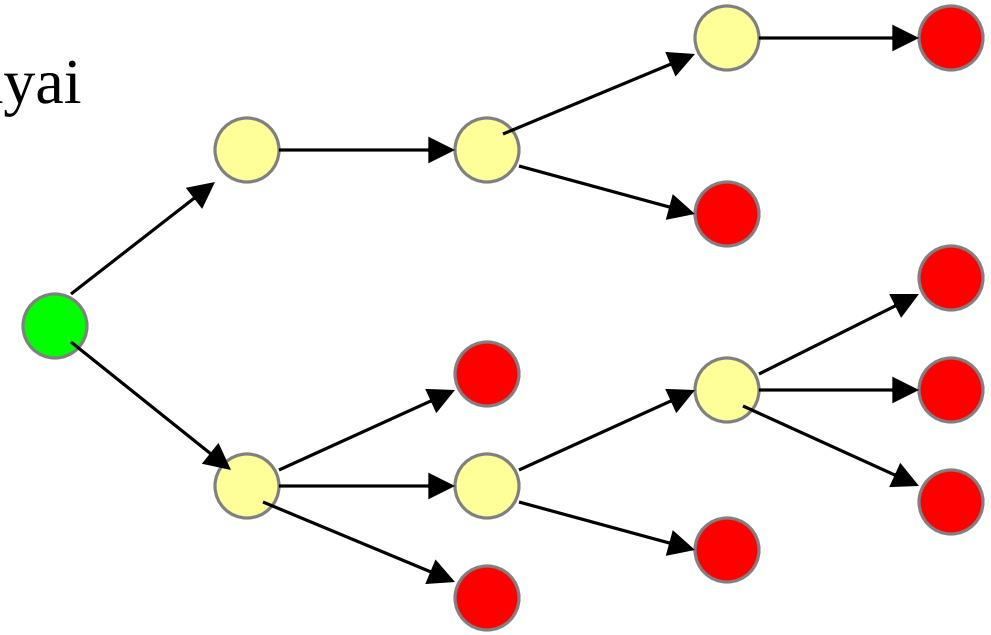
- Tiap simpul pohon menyatakan status (*state*) persoalan, sedangkan sisi (cabang) dilabeli dengan nilai-nilai x_i.

- 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:

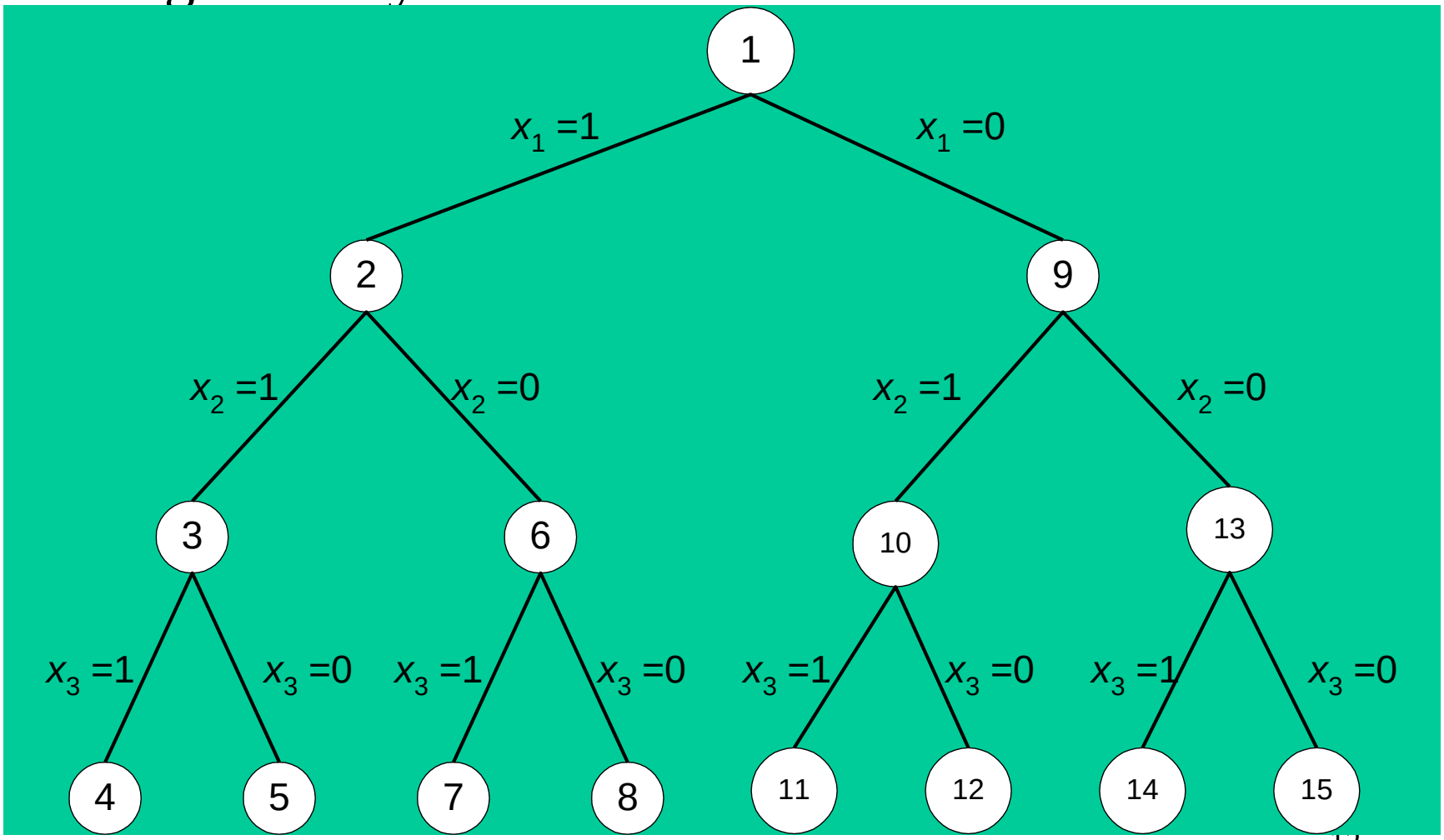
- Simpul akar
- Simpul dalam
- Simpul daun

Backtracking dapat dipandang sebagai pencarian di dalam pohon menuju simpul daun (goal) tertentu

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

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 *depht-first order* (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*).

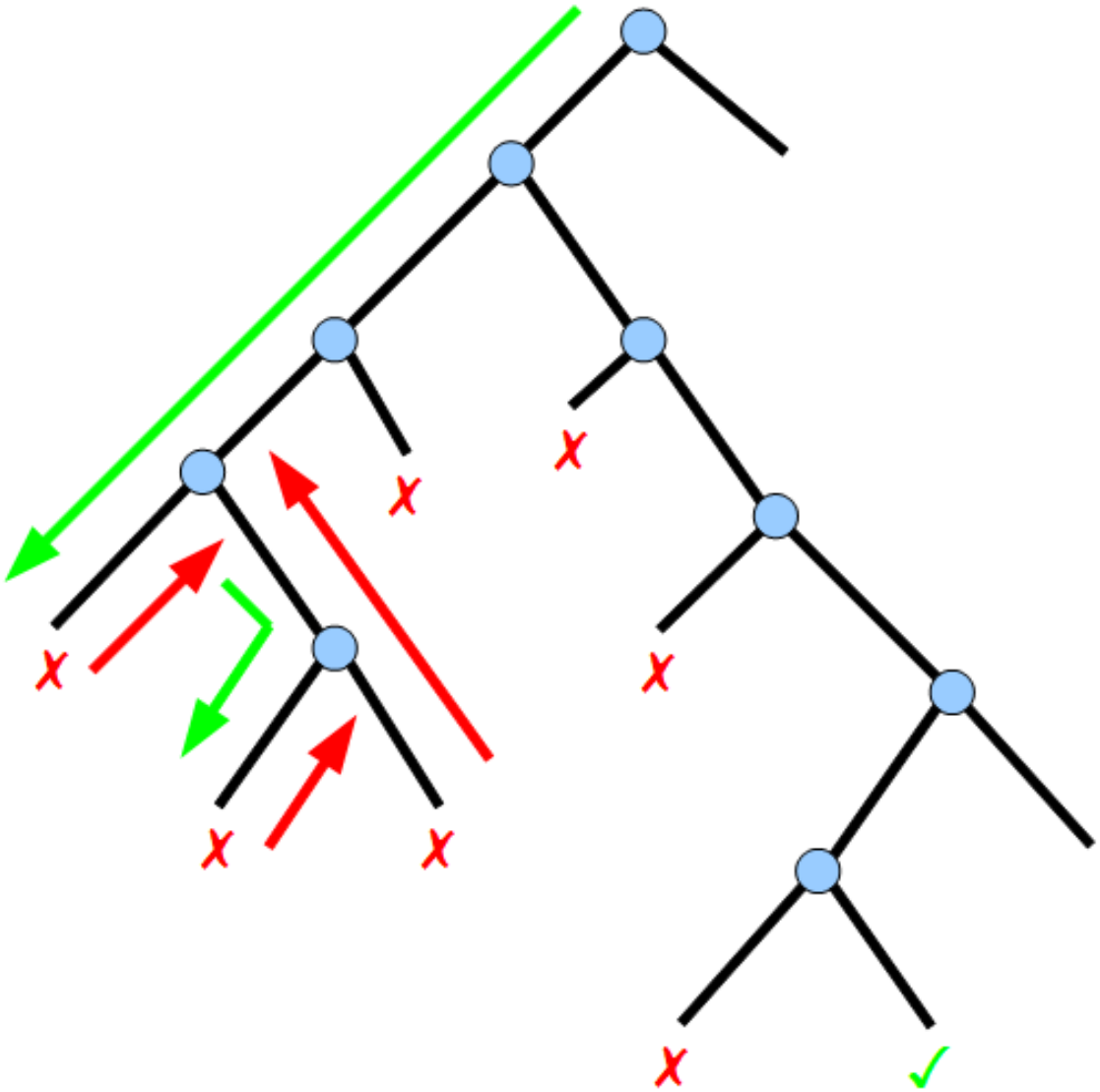
- Simpul yang sudah mati tidak akan pernah diperluas lagi.

- Jika pembentukan lintasan berakhir dengan simpul mati, maka proses pencarian *backtrack* ke simpul 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 *Knapsack* 0/1 dengan instansiasi:

  $n = 3$

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

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

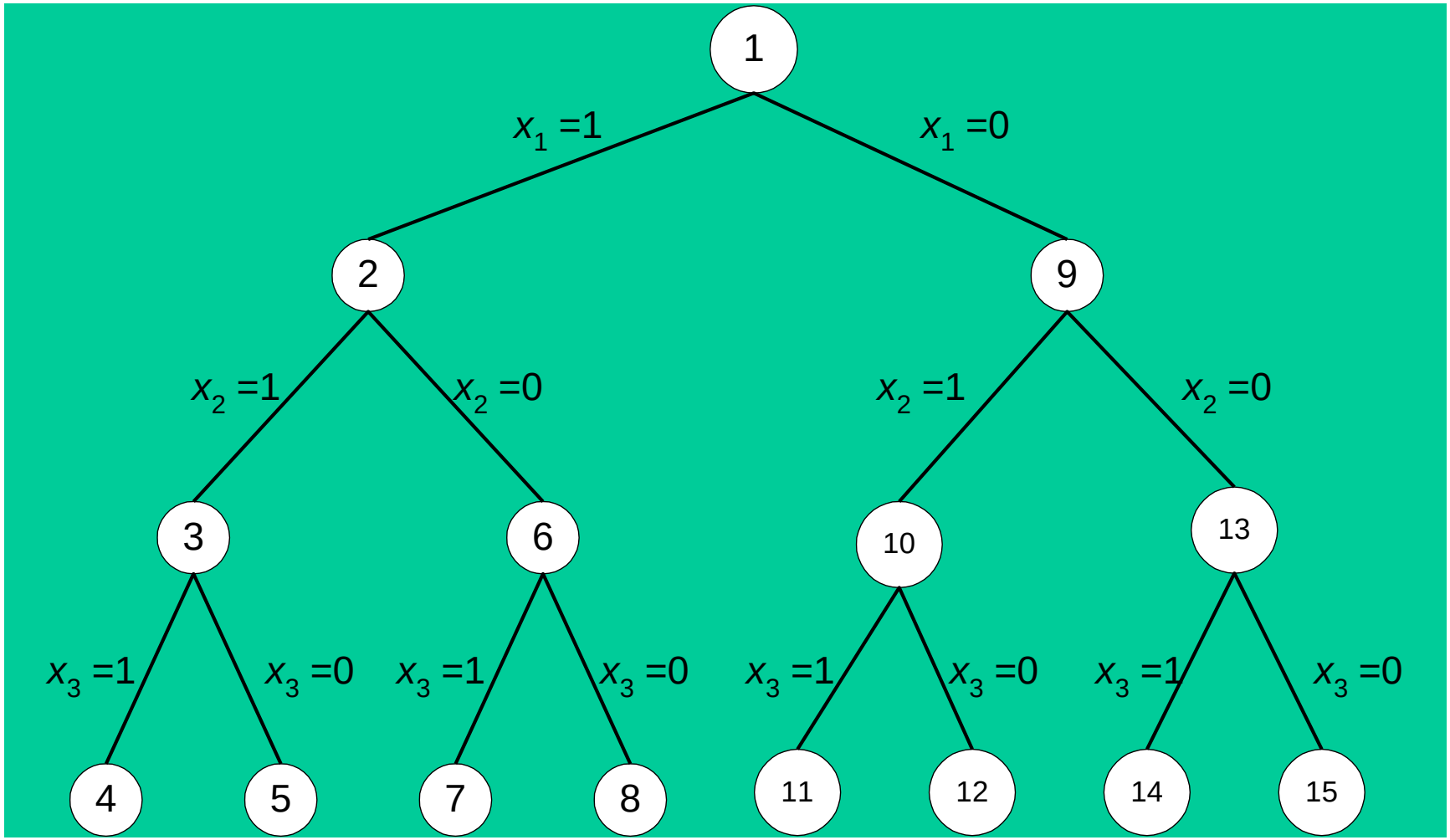
  $M = 30$

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

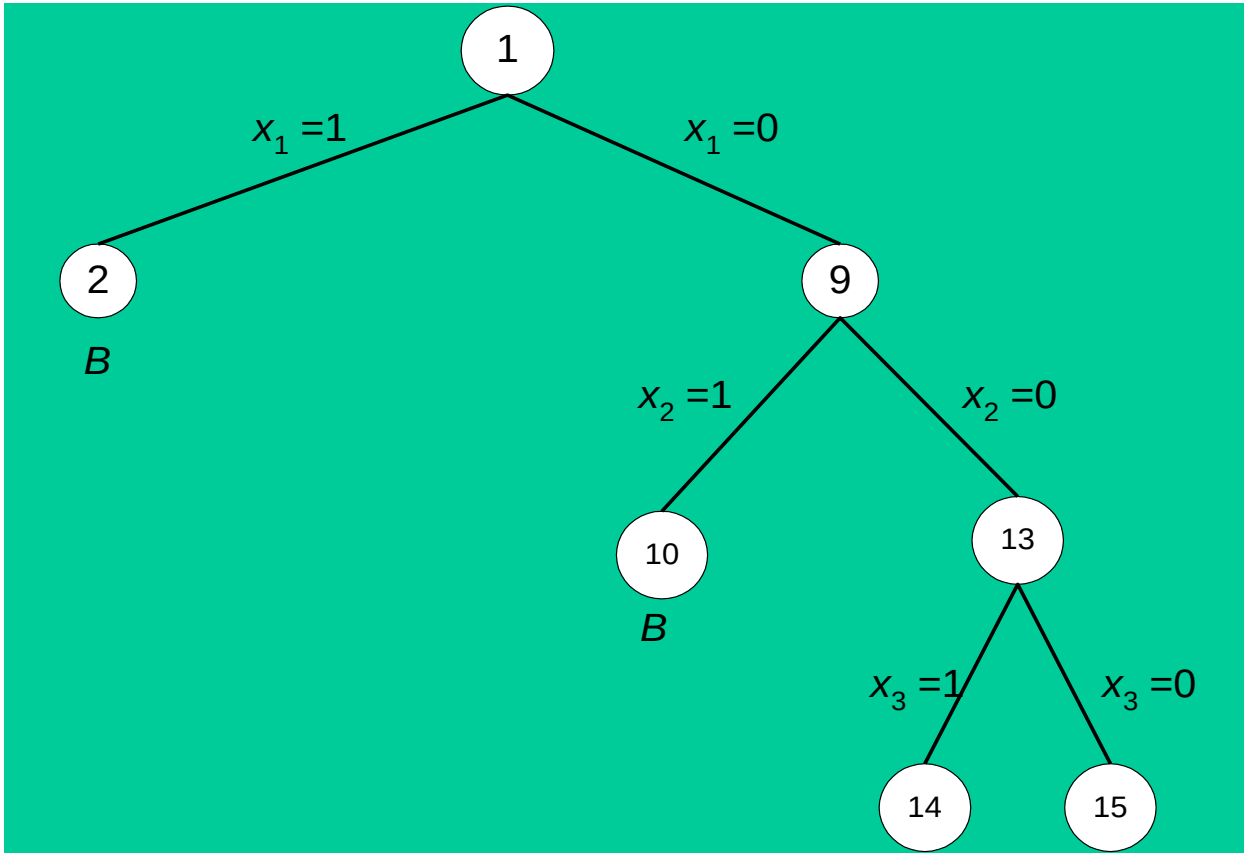
- Fungsi konstrain (dapat dianggap sebagai *bounding function*):

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

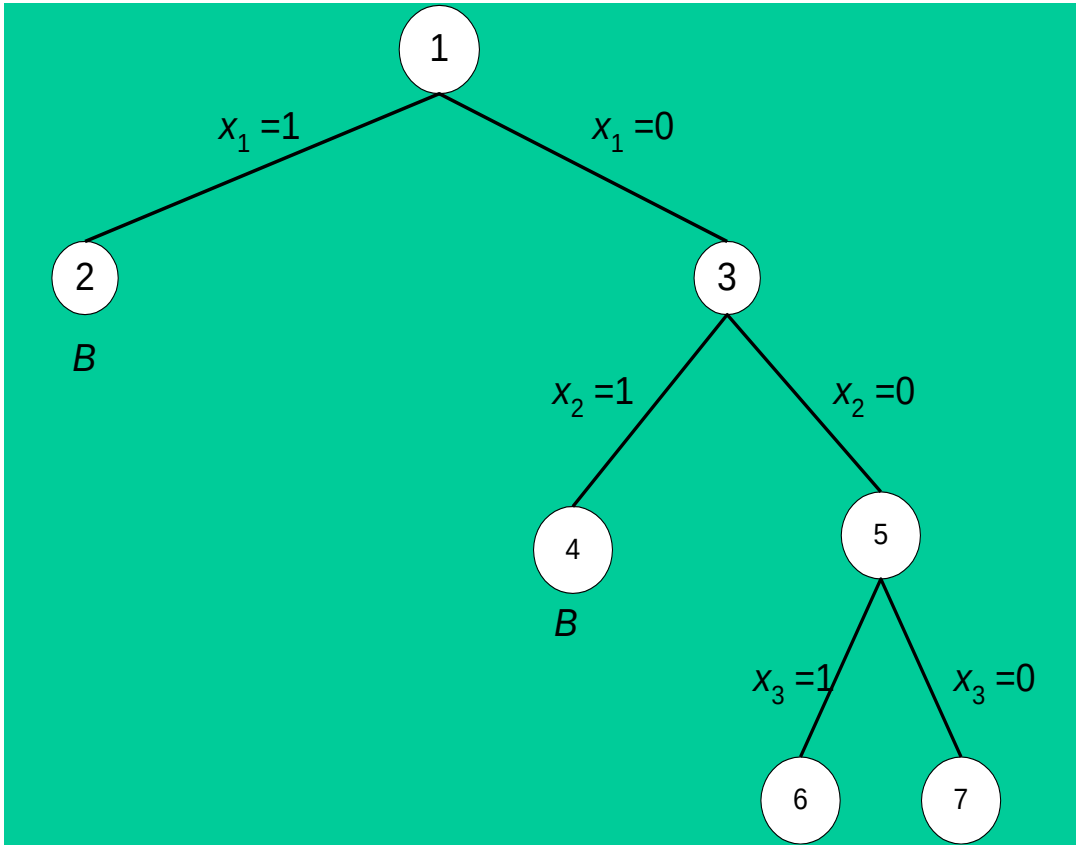
Pada metode *brute force*, semua lintasan dari akar ke daun dievaluasi



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
```

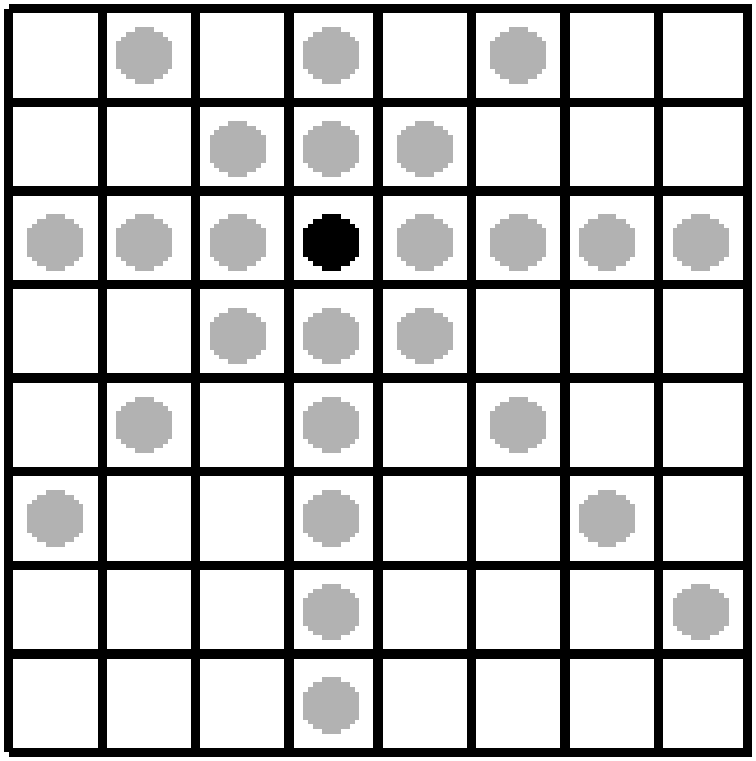
- Setiap simpul dalam pohon ruang status berasosiasi dengan sebuah pemanggilan rekursif.

- 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 $O(q(n)n!)$,

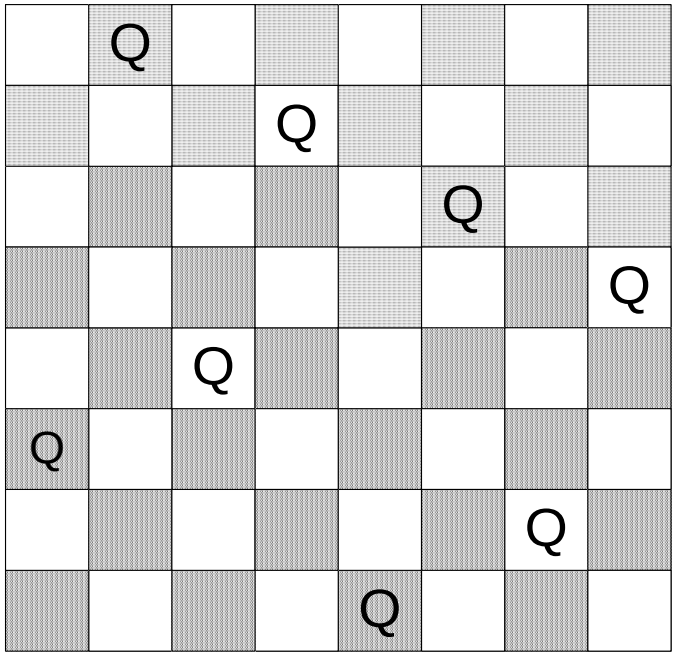
- dengan $p(n)$ dan $q(n)$ adalah polinom derajat n yang menyatakan waktu komputasi setiap simpul.

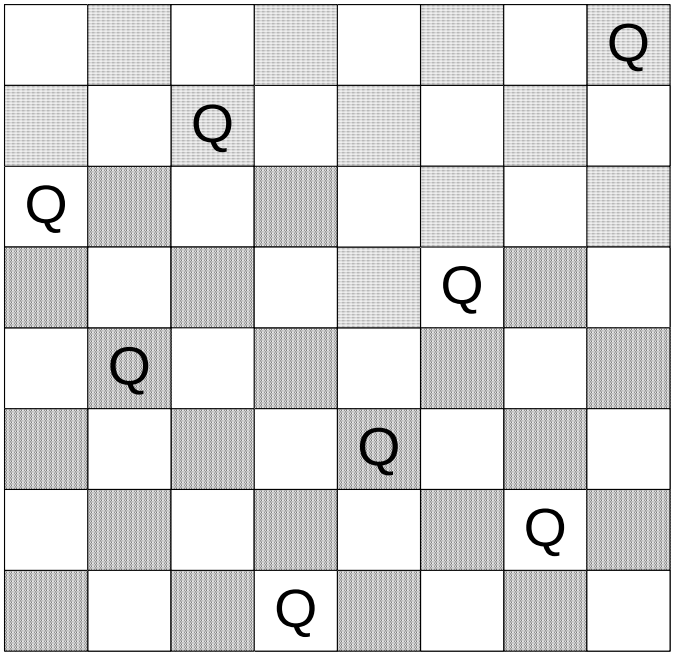
Persoalan N-Ratu (*The N-Queens Problem*)

- 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*:





- 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)

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.

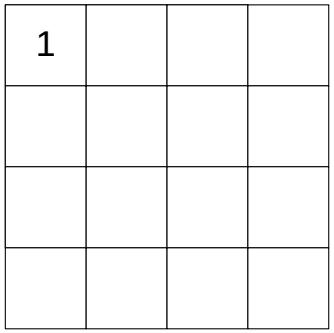
Penyelesaian dengan Algoritma Runut-balik:

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

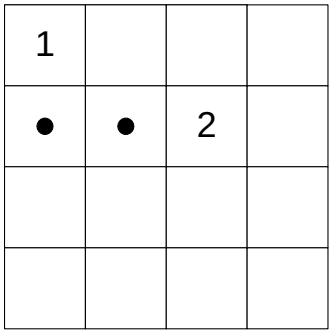
- Ruang solusinya adalah semua permutasi dari angka-angka 1, 2, 3, 4, 5, 6, 7, 8.

- 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*.

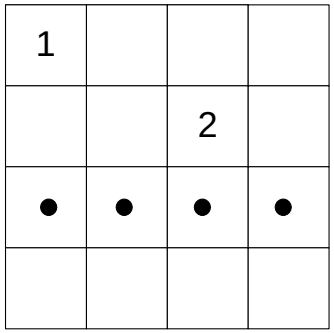
Contoh solusi runut-balik persoalan 4-Ratu:



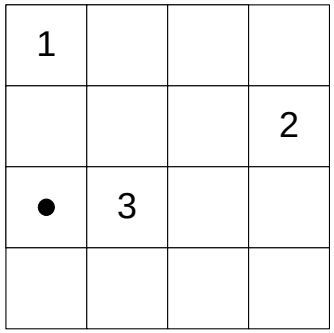
(a)



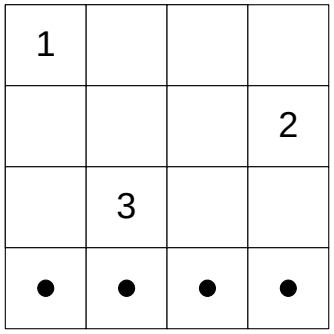
(b)



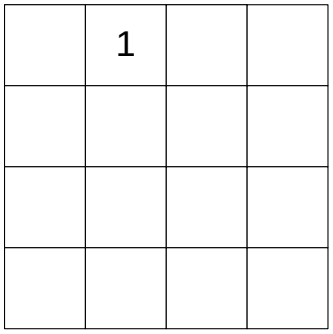
(c)



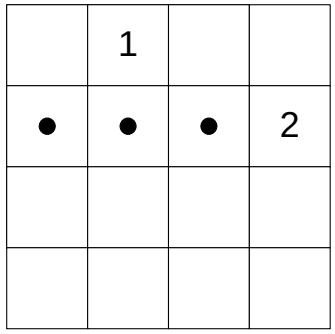
(d)



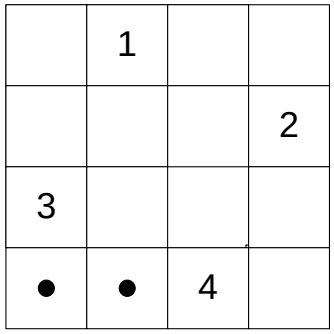
(e)



(f)

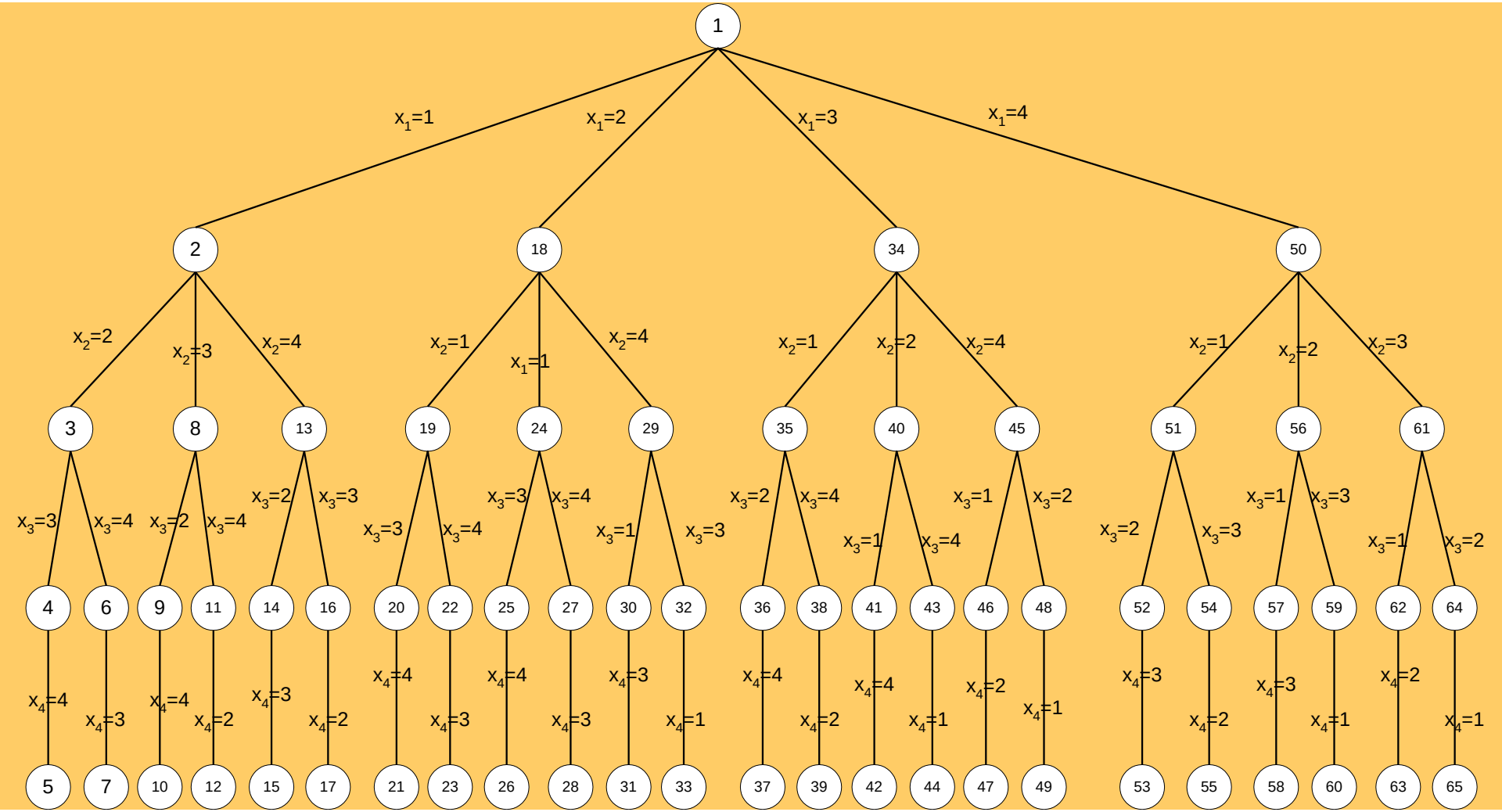


(g)

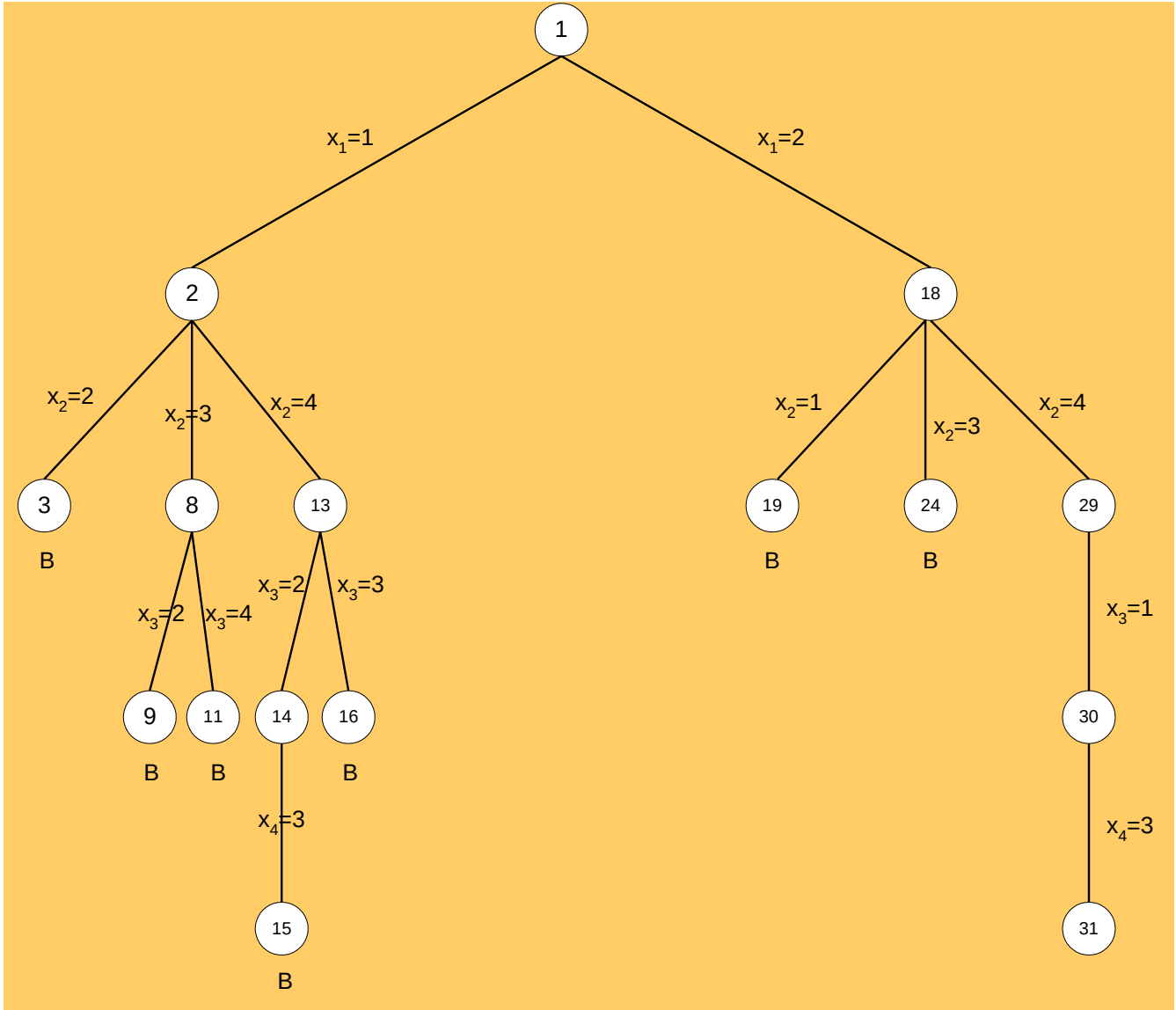


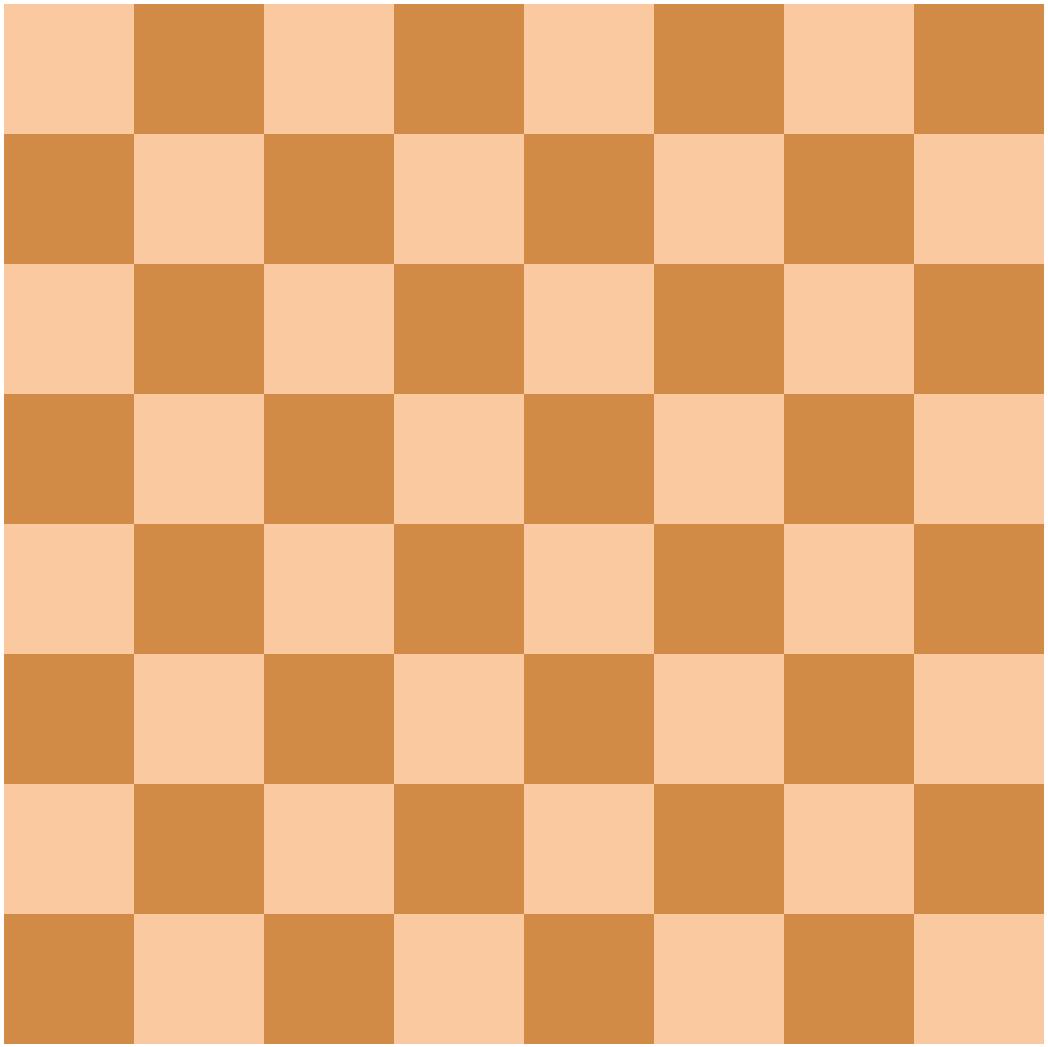
(h)

Contoh: Pohon ruang-status persoalan 4-Ratu



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





Algoritma Runut-balik untuk Persoalan 8-Ratu

- Tinjau dua posisi ratu pada (i, j) dan (k, l)
- 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 ↙ $i + j = k + l$

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

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

```
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
```

Algoritma:

- Inisialisasi $x[1], x[2], \ldots, x[N]$ dengan 0

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

- Panggil prosedur N_RATU_R(1)

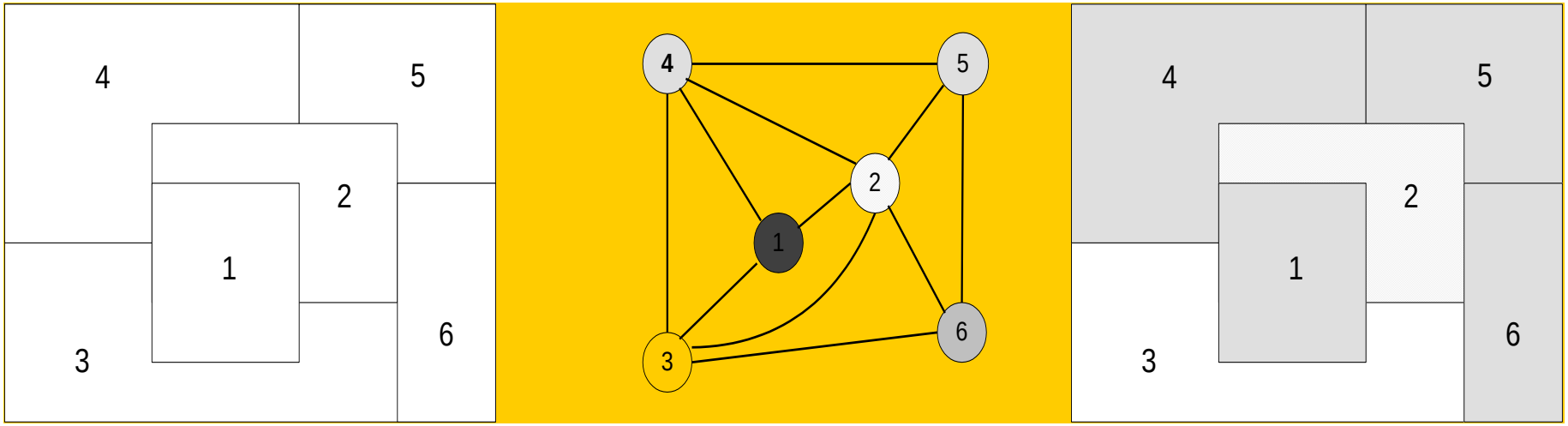
```
procedure N_RATU_R(input k:integer)
{ Menempatkan ratu pada baris ke-k pada petak papan catur N x N
  tanpa melanggar kendala; versi rekursif

  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)*

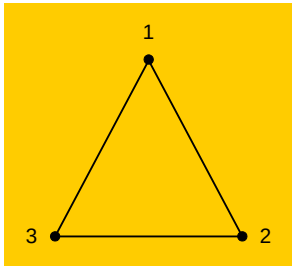
Persoalan:

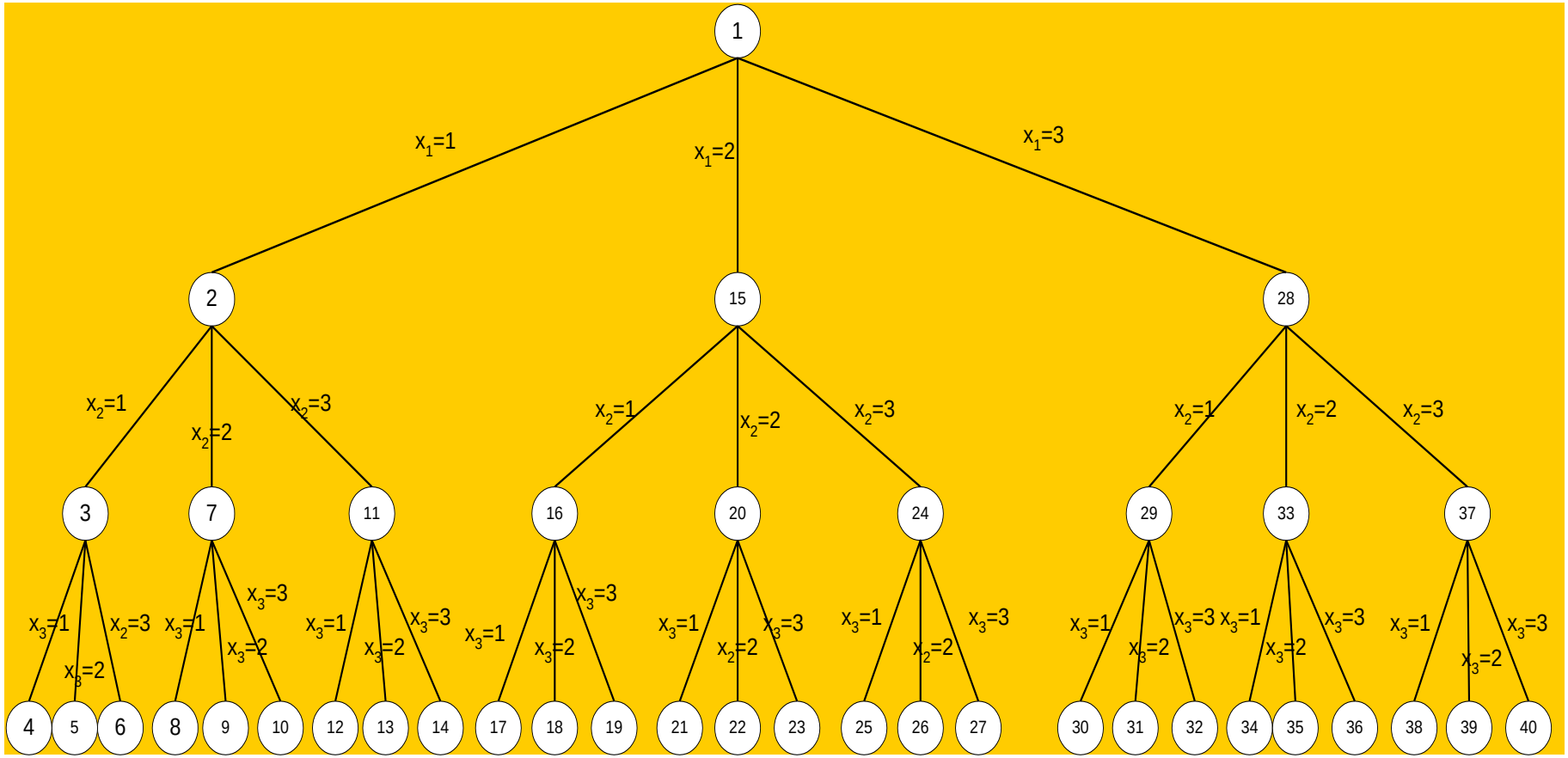
- Diberikan sebuah graf *G* dengan *n* buah simpul dan disediakan *m* buah warna. Bagaimana mewarnai seluruh simpul graf *G* sedemikian sehingga tidak ada dua buah simpul bertetangga yang mempunyai warna sama (Perhatikan juga bahwa tidak seluruh warna harus dipakai)

Contoh aplikasi: pewarnaan peta



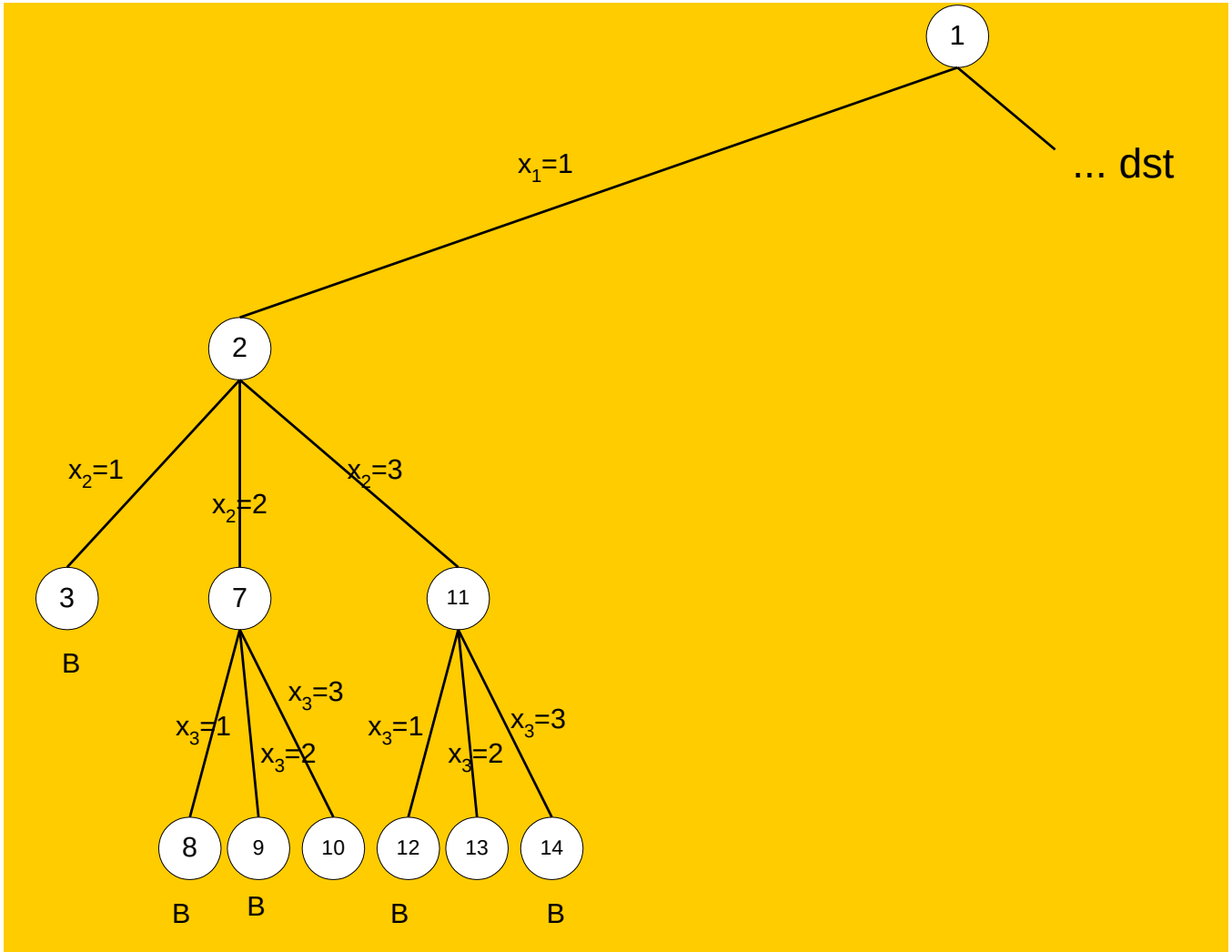
Tinjau untuk $n = 3$ dan $m = 3$.





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 ketetanggaan 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* 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

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

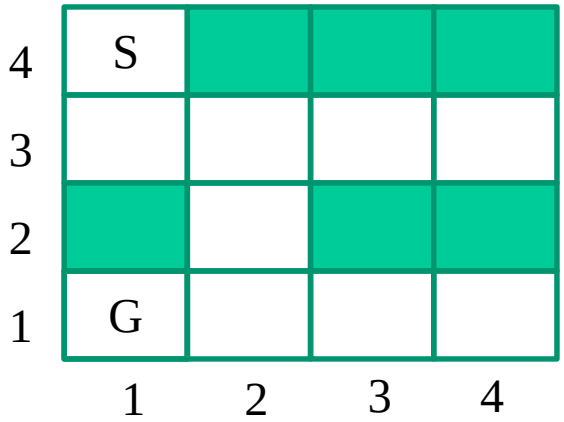
Latihan Soal
Algoritma Backtracking

IF2211 Strategi Algoritma

Program Studi Teknik Informatika ITB
2020

Latihan 1: Soal UAS 2019

Terdapat sebuah labirin sederhana seperti pada Gambar 1. Titik S (Start) berada pada posisi (1,4), dan titik G (*Goal*) berada pada posisi (4,1). Sel yang diarsir adalah sel yang tidak bisa dilewati.Persoalan yang akan diselesaikan adalah menemukan jalur dari S menuju G dengan menggunakan Algoritma Backtracking. Jarak dari satu titik ke titik berikutnya adalah 1 (satu) satuan jarak. Operasi yang bisa dilakukan adalah bergerak *east*(posisi x bertambah 1), *south*(posisi y berkurang 1), *west*(posisi x berkurang 1), dan *north*(posisi y bertambah 1). Jika diperlukan, urutan prioritas operasiyang dilakukan adalah *east*, *south*, *west*, *north*.



Buatlah pohon pencarian jalur ke titik Goal(4,1)dengan menggunakan Algoritma Backtracking, dimulai dari titik (1,4). Tulislah nomor urutan pembangkitan pada setiap simpul pohon pencarian. Pencarian dihentikan ketika sudah mencapai titik G. Kemudian tuliskan hasil urutan aksiyang dilakukan untuk mencapai G dari S.

Penyelesaian:

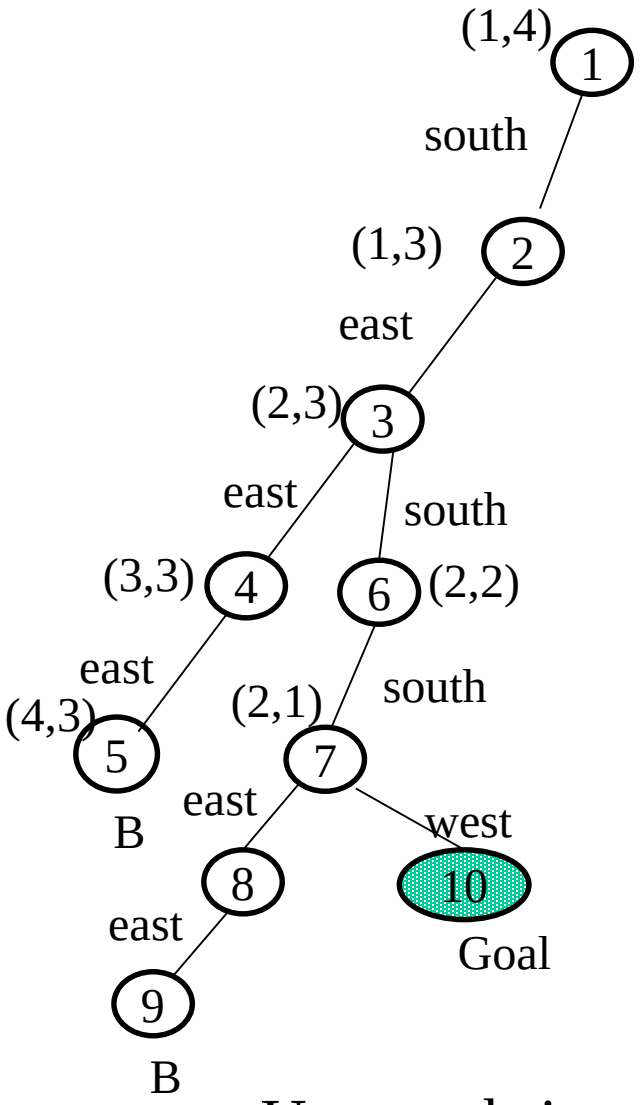
- Solusi dinyatakan sebagai vector $X = (x_1, x_2, \ldots, x_m)$

  $x_i \in \{east, south, west, north\}$

-

- Fungsi T(.) mencoba meng-assign x_i dengan urutan *east*, *south*, *west*, *north*

- Fungsi pembatas B memeriksa apakah koordinat (x, y) belum mencapai batas labirin, yaitu $1 < x < 4$ dan $1 < y < 4$ atau masih bisa berpindah ke sel lain. Jika true, ekspansi simpul, jika false, matikan simpul.



Urutan aksi: south – east – south – south – west

Solusi: X = (south, east, south, south, west)

Latihan 2: *Sum of Subset Problem*

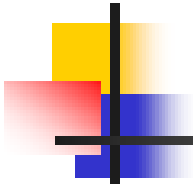
- Diberikan *n* buah bilangan positif berbeda w_1, w_2, …, w_n dan sebuah bilangan bulat positif *W*. Tentukan semua himpunan bagian dari *n* integer tersebut yang jumlahnya sama dengan *W*.

  Contoh: $w_1 = 3$ $w_2 = 4$, $w_3 = 5$, $w_4 = 6$, $W = 13$.

  Himpunan bagian yang memenuhi hanya satu, yaitu:

  $$\{3, 4, 6\}$$

  Selesaikan dengan algoritma runut-balik!



5.4 The Sum-of-Subsets Problem

- create a state space tree
  - See Fig. 5.7 pp. 215
- each ***left*** edge denotes we ***include*** wi (weight wi)
- each ***right*** edge denotes we ***exclude*** wi (weight 0)
- any path from root to a leaf forms a subset
- See Fig. 5.8 pp. 216

5.4 The Sum-of-Subsets Problem

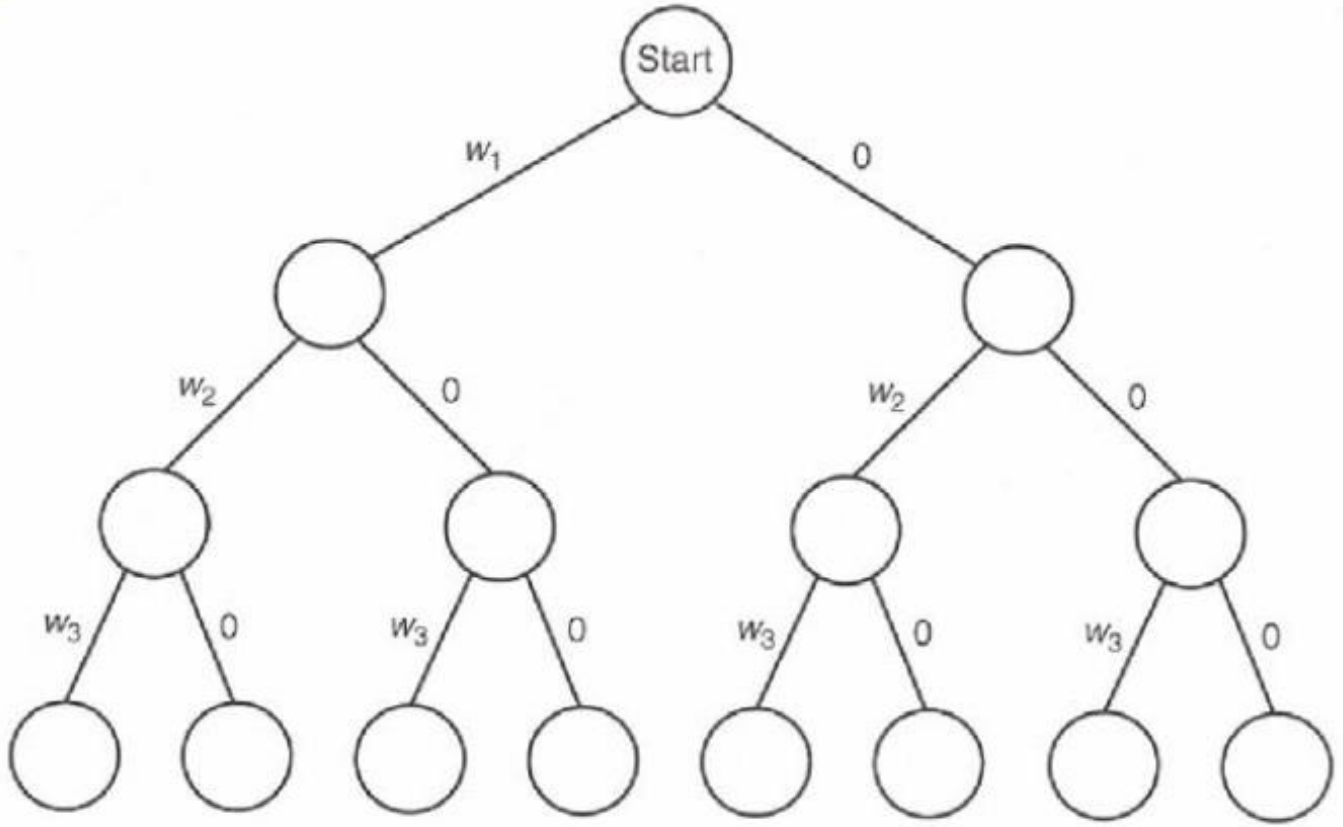


Figure 5.7 • A state space tree for instances of the Sum-of-Subsets problem in which $n = 3$.

5.4 The Sum-of-Subsets Problem

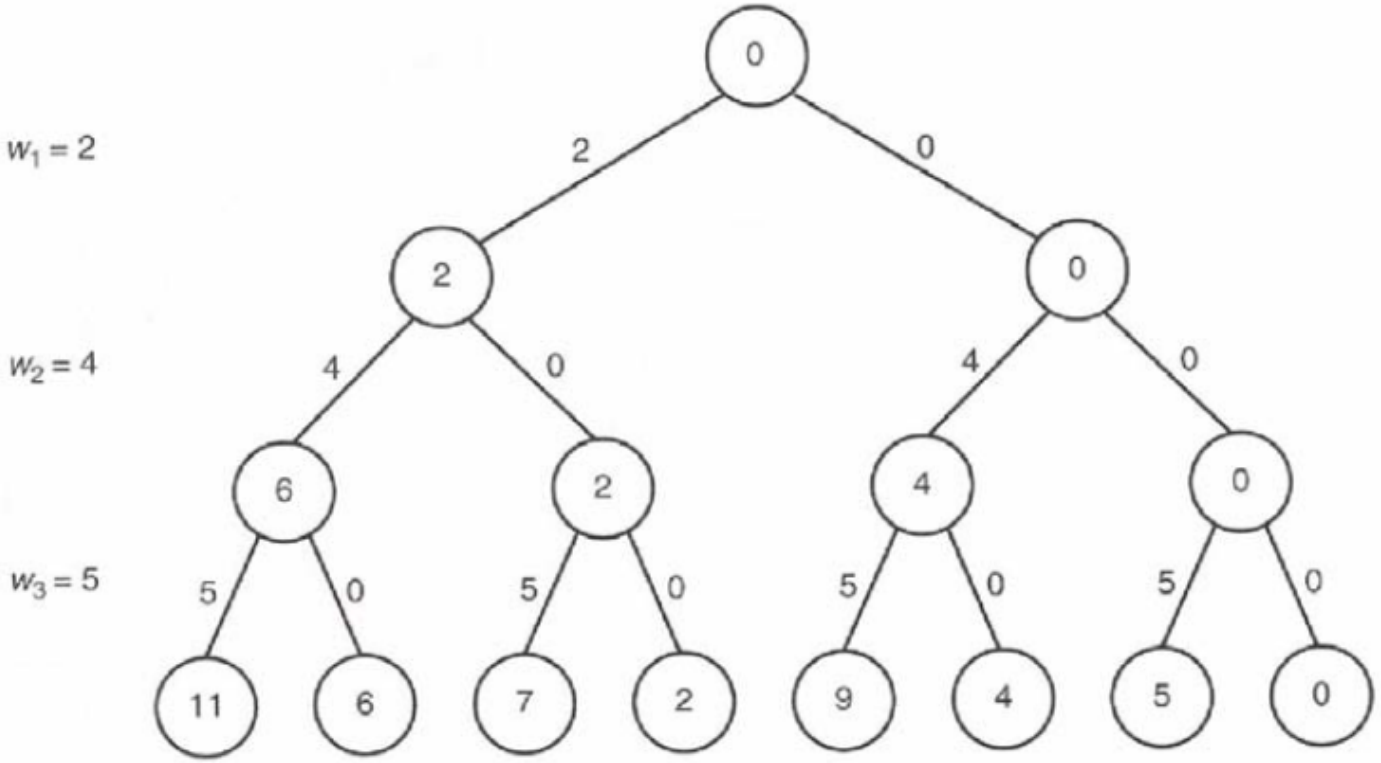


Figure 5.8 • A state space tree for the Sum-of-Subsets problem for the instance in Example 5.3. Stored at each node is the total weight included up to that node.

5.4 The Sum-of-Subsets Problem

- ***significant signs (backtracking)***
  - sorting the weights in nondecreasing order
  - weight be the subtotal from root to node i at level I
- weight $+w_{i+1} > W$
  - any descendant of node i will be nonpromising (because is w_{i+1} the lightest weight remaining)
- weight + all remaining items < W
  - any descendant of node i will be nonpromising
- Example 5.4 and Fig. 5.9 pp. 217
- See Algorithm 5.4

5.4 The Sum-of-Subsets Problem

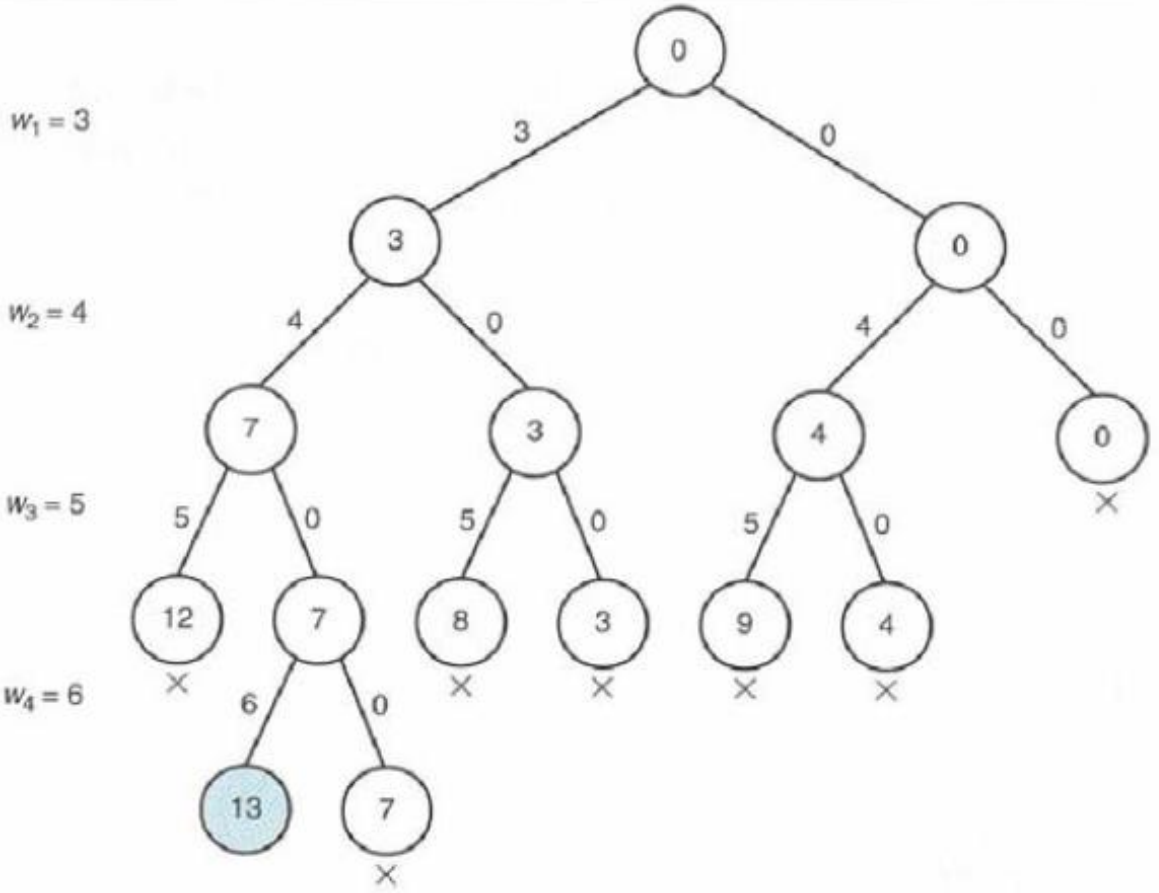


Figure 5.9 • The pruned state space tree produced using backtracking in Example 5.4. Stored at each node is the total weight included up to that node. The only solution is found at the shaded node. Each nonpromising node is marked with a cross.

Algorithm 5.4

The Backtracking Algorithm for the Sum-of-Subsets Problem

Problem: Given n positive integers (weights) and a positive integer W, determine all combinations of the integers that sum to W.

Inputs: positive integer n, sorted (nondecreasing order) array of positive integers w indexed from 1 to n, and a positive integer W.

Outputs: all combinations of the integers that sum to W.

```
void sum_of_subsets (index i,
                     int weight, int total)
{
   if (promising(i))
      if (weight == W)
         cout << include[1] through include[i];
      else {
         include[i + 1] = "yes";                    // Include w[i + 1].
         sum_of_subsets(i + 1, weight + w[i + 1], total - w[i + 1]);
         include[i + 1] = "no";                     // Do not include w[i + 1].
         sum_of_subsets(i + 1, weight, total - w[i + 1]);
      }
}

bool promising (index i);
{
return (weight + total >= W) && (weight == W || weight + w[i + 1] <= W);
}
```