Algoritma Runut-balik (*Backtracking*) (Bagian 2)

Bahan Kuliah IF2211 Strategi Algoritma

Oleh: Rinaldi Munir

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika ITB

2021

2. Sum of Subsets Problem

- **Persoalan**: Diberikan n buah bobot (*weight*) berupa bilangan-bilangan positif (*integer*) yang berbeda w_1, w_2, …, w_n dan sebuah bilangan bulat positif m. Tentukan semua himpunan bagian dari n bobot tersebut yang jumlahnya sama dengan m.

  Contoh: $n = 4$; $(w_1, w_2, w_3, w_4) = (11, 13, 24, 7)$, $m = 31$.

  Himpunan bagian yang memenuhi adalah {11, 13, 7} dan {24, 7} .

- Perhatikan, persoalan *sum of subset* mungkin saja tidak memiliki solusi. Misalnya pada contoh di atas, jika $m = 30$, maka tidak ada himpunan bagian yang memenuhi.

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

  $x_i = 1$, artinya w_i dimasukkan ke dalam subset

  $x_i = 0$, artinya w_i tidak dimasukkan ke dalam subset

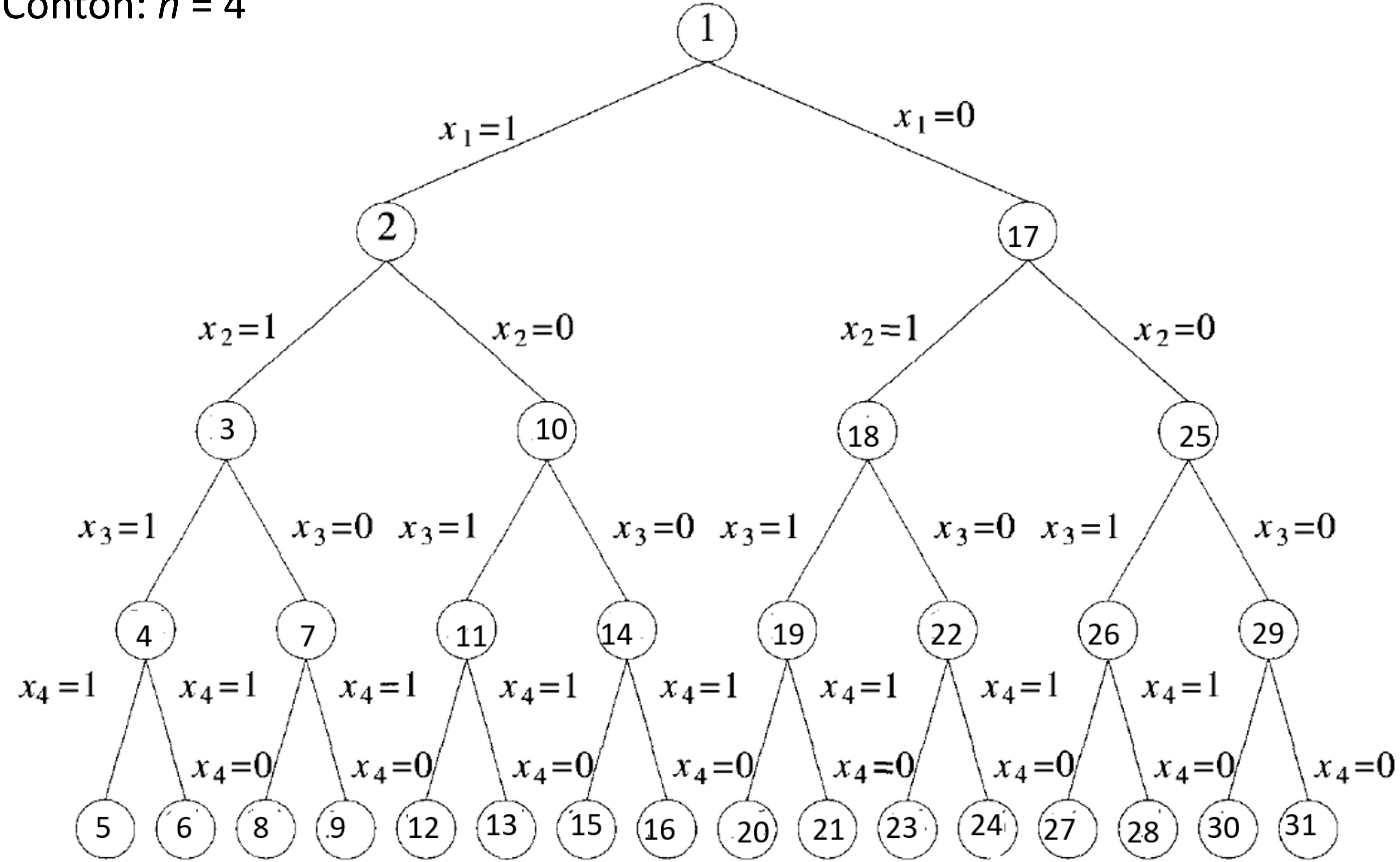
- Pohon ruang status untuk persoalan *sum of subset* berupa pohon biner.

- Sisi pada cabang kiri menyatakan w_i diambil ($x_i = 1$),

- sedangkan sisi pada cabang kanan menyatakan w_i tidak diambil ($x_i = 0$).

- Sembarang lintasan dari akar ke daun menyatakan himpunan bagian (*subset*)

Contoh: n = 4


1
x1=1
x1=0
2
17
x2=1
x2=0
x2=1
x2=0
3
10
18
25
x3=1
x3=0
x3=1
x3=0
x3=1
x3=0
x3=1
x3=0
4
7
11
14
19
22
26
29
x4=1
x4=0
5
6
8
9
12
13
15
16
20
21
23
24
27
28
30
31

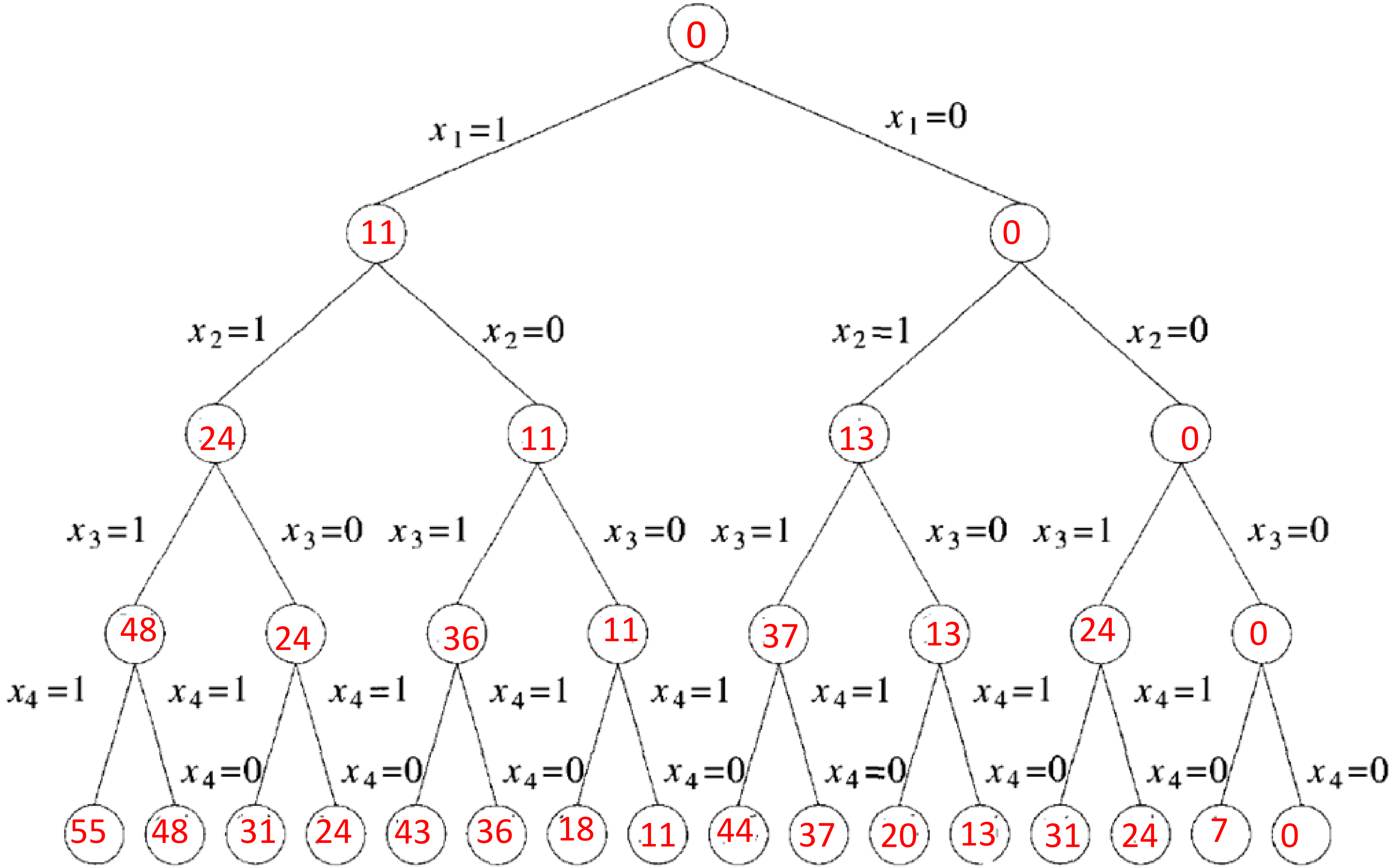
- Sekarang, angka di dalam setiap simpul diganti dengan nilai yang menyatakan jumlah bobot sampai ke simpul tersebut

$w_1 = 11$

$w_2 = 13$

$w_3 = 24$

$w_4 = 7$



- Sebelum dilakukan pencarian solusi, urutkan semua bobot secara menaik dari nilai terkecil hingga nilai yang terbesar.

- Misalkan $x_1, x_2, \ldots, x_{k-1}$ sudah di-*assign* dengan sebuah nilai (0 atau 1). Maka, pada pengisian nilai untuk x_k, kita dapat menggunakan fungsi pembatas (*bounding function*) sebagai berikut:

$$B(x_1, x_2, \ldots, x_k) = \textit{true} \quad \text{jika dan hanya jika} \quad \sum_{i=1}^{k} w_i x_i + \sum_{i=k+1}^{n} w_i \geq m$$

- Ini berarti, $x_1, x_2, \ldots, x_k$ tidak mengarah ke simpul solusi (*goal node*) jika kondisi di atas tidak dipenuhi.

- Perhatikan bahwa $\sum_{i=1}^{k} w_i x_i + \sum_{i=k+1}^{n} w_i \geq m$ artinya jumlah bobot sampai simpul ke-k ditambah dengan bobot-bobot yang tersisa masih lebih besar atau sama dengan m.

- Oleh karena bobot-bobot sudah terurut menaik, maka kita dapat memperkuat fungsi pembatas dengan kondisi bahwa $x_1, x_2, \ldots, x_k$ tidak mengarah ke simpul solusi jika

$$\sum_{i=1}^{k} w_i x_i + w_{k+1} > m$$

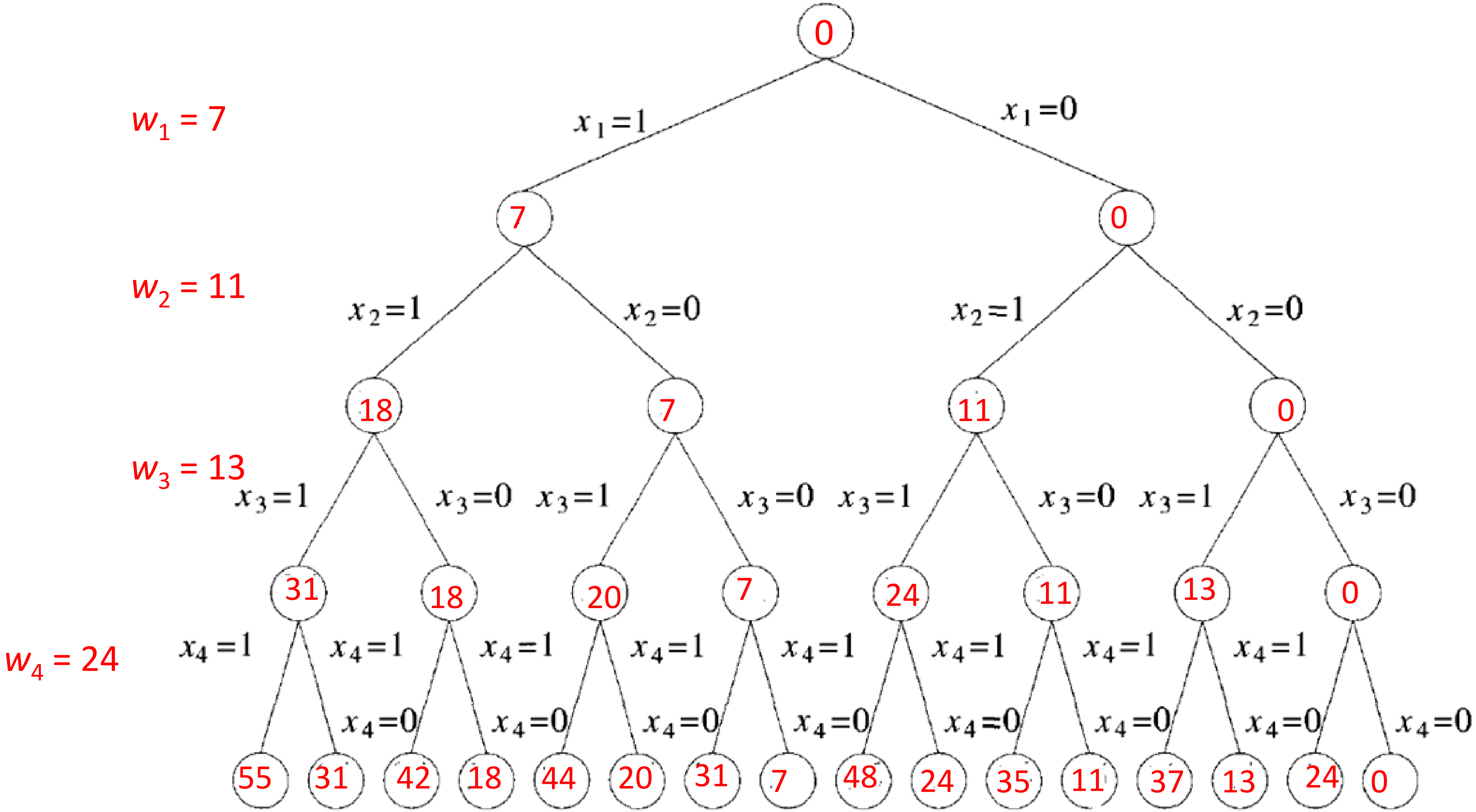
- artinya tidak mengarah ke simpul solusi jumlah bobot sampai simpul ke-*k* ditambah dengan bobot ke-(*k*+1) lebih besar dari *m*.
- Jika jumlah bobot sampai simpul ke-*k* sudah sama dengan *m*, maka STOP.
- Dengan demikian, fungsi pembatas keseluruhan adalah

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

$$\left(\sum_{i=1}^{k} w_i x_i = m \text{ atau } \sum_{i=1}^{k} w_i x_i + w_{k+1} \leq m \right)$$

- Artinya $x_1, x_2, \ldots, x_k$ mengarah ke simpul solusi jika kedua kondisi di atas dipenuhi.

Contoh: $n = 4$; $m = 31$, $(w_1, w_2, w_3, w_4) = (7, 11, 13, 24)$ → sudah diurut menaik



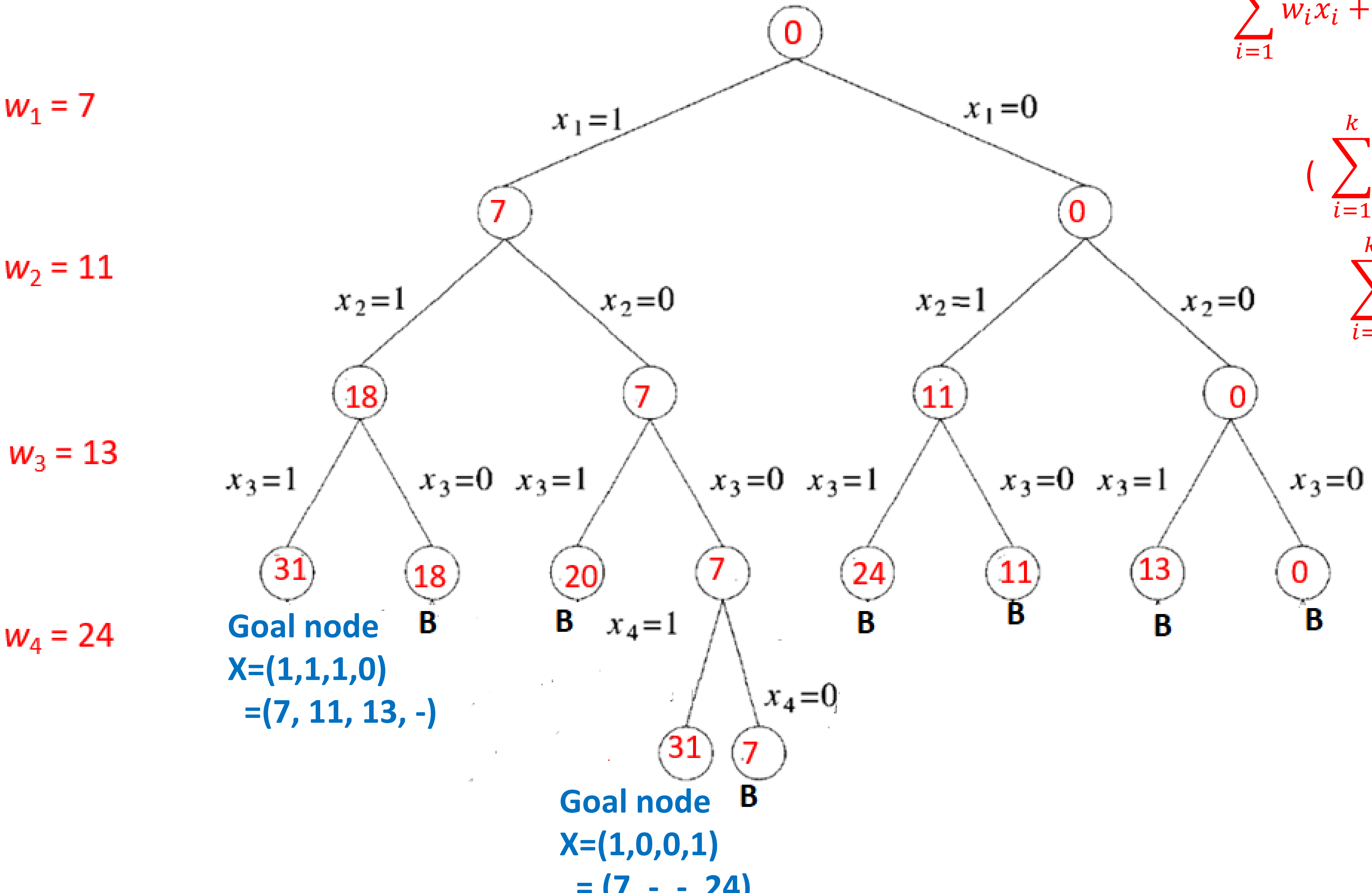
Pencarian solusi: n = 4; (w_1, w_2, w_3, w_4) = (7, 11, 13, 24), m = 31.

$B(x_1, x_2, ..., x_k) = true$ iff

$$\sum_{i=1}^{k} w_i x_i + \sum_{i=k+1}^{n} w_i \geq m$$

dan

$$\left(\sum_{i=1}^{k} w_i x_i = m \text{ atau } \sum_{i=1}^{k} w_i x_i + w_{k+1} \leq m \right)$$



Pseudo-code algoritma *sum-of-subset* dengan *backtracking*

- $Wt = \sum_{i=1}^{k} w_i x_i$
- $sisabobot = \sum_{i=k+1}^{n} w_i$
- Mengarah ke simpul solusi (*promising*) jika

  $(Wt + sisabobot \geq m)$ dan $(Wt = m$ atau $Wt + w_{k+1} \leq m)$

Algoritma *SumofSubset*:

Masukan: n, m, $W = \{w_1, w_2, \ldots, w_n\}$

Luaran: semua himpunan bagian dari W yang jumlahnya sama dengan m

Langkah-Langkah algoritma:

1. Urutkan elemen-elemen W sehingga terurut membesar (dari kecil ke besar)
2. Hitung $total = w_1 + w_2 + \ldots + w_n$
3. Panggil prosedur *SumOfSubset*(0, 0, *total*)

function *promising*(**input** k : **integer**, Wt : **integer**, *sisabobot* : *integer*) $\rightarrow$ **boolean**
{ true jika simpul ke-k mengarah ke goal node, false jika tidak }

Algoritma:
return (($Wt + sisabobot \geq m$) **and** ($Wt = m$ **or** $Wt + w[k+1] \leq m$))

```
procedure SumOfSubsets(input k : integer,  Wt : integer, sisabobot : integer)
{ Mencari semua kombinasi himpunan bagian yang jumlahnya sama dengan m
   Masukan: Wt = jumlah bobot sampai simpul ke-k, sisabobot = jumlah bobot dari k+1 sampai n
   Luaran: semua himpunan bagian yang jumlah bobotnya sama dengan m
}

Algoritma:
      if  promising(k, Wt, sisabobot)  then
           if   Wt = m  then
                write(x[1], x[2], …, x[n])
           else
                x[k+1] = 1           { masukkan w[k+1] }
                SumOfSubsets(k+1, Wt + w[k+1], sisabobot – w[k+1])

                x[k+1] = 0            { w[k+1] tidak dimasukkan  }
                SumOfSubsets(k+1, Wt, sisabobot – w[k+1])
           endif
      endif
```

Program C++ untuk persoalan *Sum of Subset*

```
// Program Sum of Subset Problem

#include <iostream>
using namespace std;

int x[10], w[10];
int N, m;

bool promising(int k, int W, int sisabobot)
{
    return ((W + sisabobot >= m) && (W == m || W + w[k+1] <= m));
}
```

```
void sumofsubsets(int k, int Wt, int sisabobot) {
  int j;

  if (promising(k, Wt, sisabobot)){
      if (Wt==m) {
          for(j=1;j<=N;j++)
             if (x[j]==1) cout << w[j] << " ";
          cout << endl;
      }
      else {
        x[k+1] = 1;
        sumofsubsets(k+1, Wt + w[k+1], sisabobot - w[k+1]);

        x[k+1] = 0;
        sumofsubsets(k+1, Wt, sisabobot - w[k+1]);
      }
  }
}
```

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

 N = 4;
 w[1] = 7; w[2] = 11; w[3] = 13; w[4] = 24;  //semua bobot sudah terurut menaik
 m = 31;
 cout << "N = " << N << endl;
 cout << "m = " << m << endl;
 total = 0;
 for (j=1;j<=N; j++) {
    cout << "w[" << j << "] = " << w[j] << endl;
    total = total + w[j];
 }
 cout << "Solusi:" << endl;
 sumofsubsets(0, 0, total);
 return 0;
}
```

Command Prompt

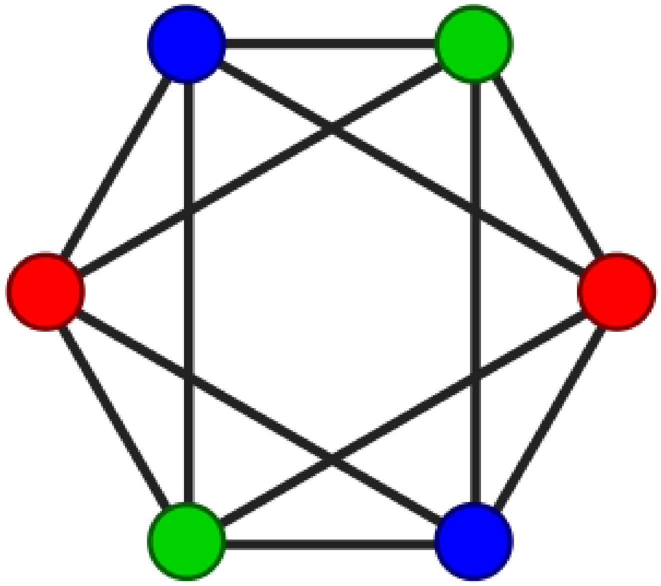
```
D:\IF2211 Strategi Algoritma\2021>g++ sumofsubset.cpp

D:\IF2211 Strategi Algoritma\2021>a
N = 4
m = 31
w[1] = 7
w[2] = 11
w[3] = 13
w[4] = 24
Solusi:
7 11 13
7 24

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

3. Pewarnaan Graf (*Graph Colouring)*

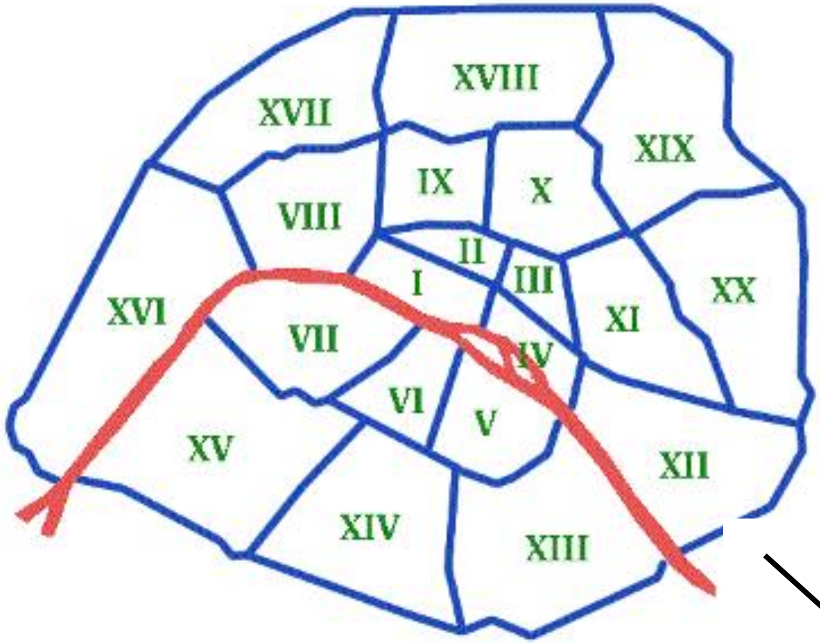
Persoalan:

Diberikan sebuah graf *G* dengan *n* buah simpul dan disediakan *m* buah warna. Bagaimana mewarnai seluruh simpul di dalam graf *G* sedemikian sehingga tidak ada dua buah simpul bertetangga memiliki warna sama?

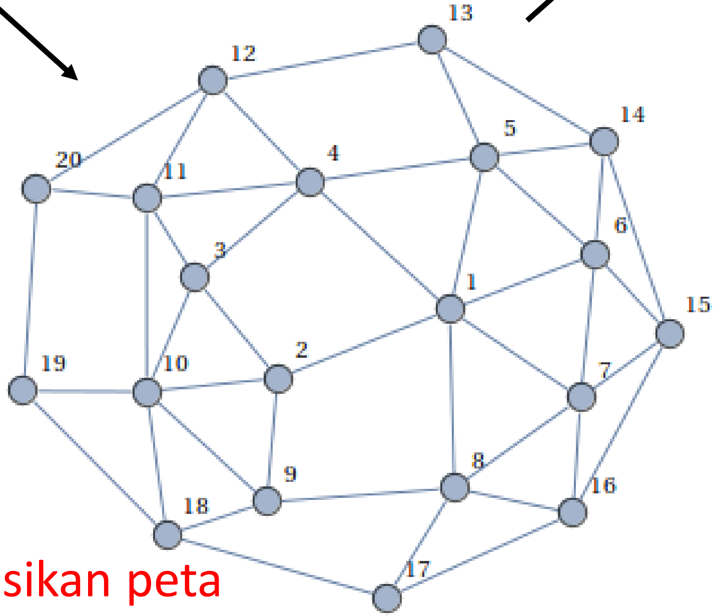
(Perhatikan juga bahwa tidak seluruh warna harus dipakai)



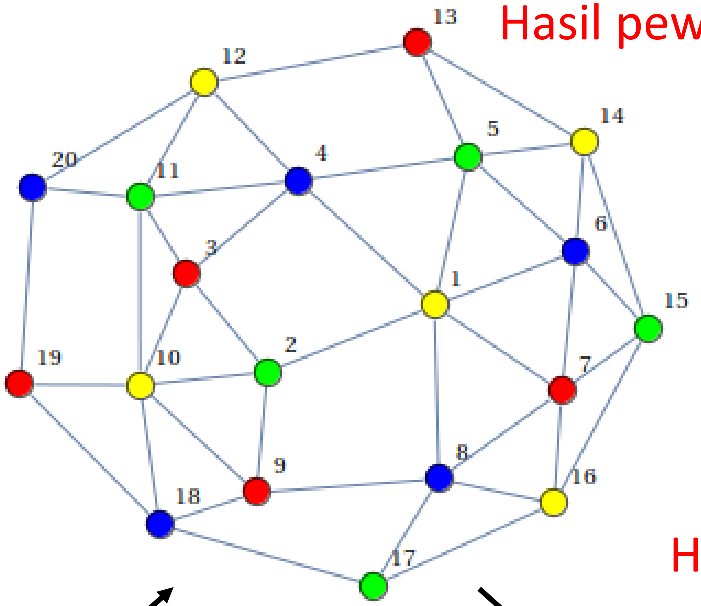
Contoh aplikasi pewarnaan graf: pewarnaan peta



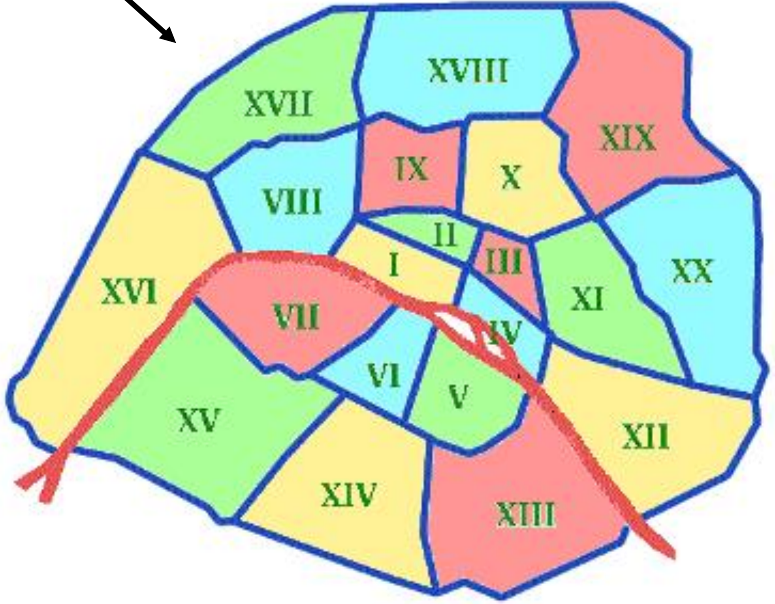
Peta wilayah di kota Paris



Graf yang merepresentasikan peta



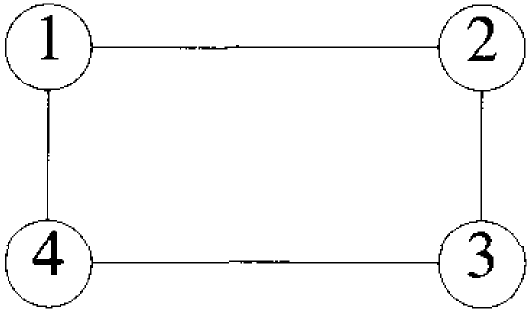
Hasil pewarnaan graf

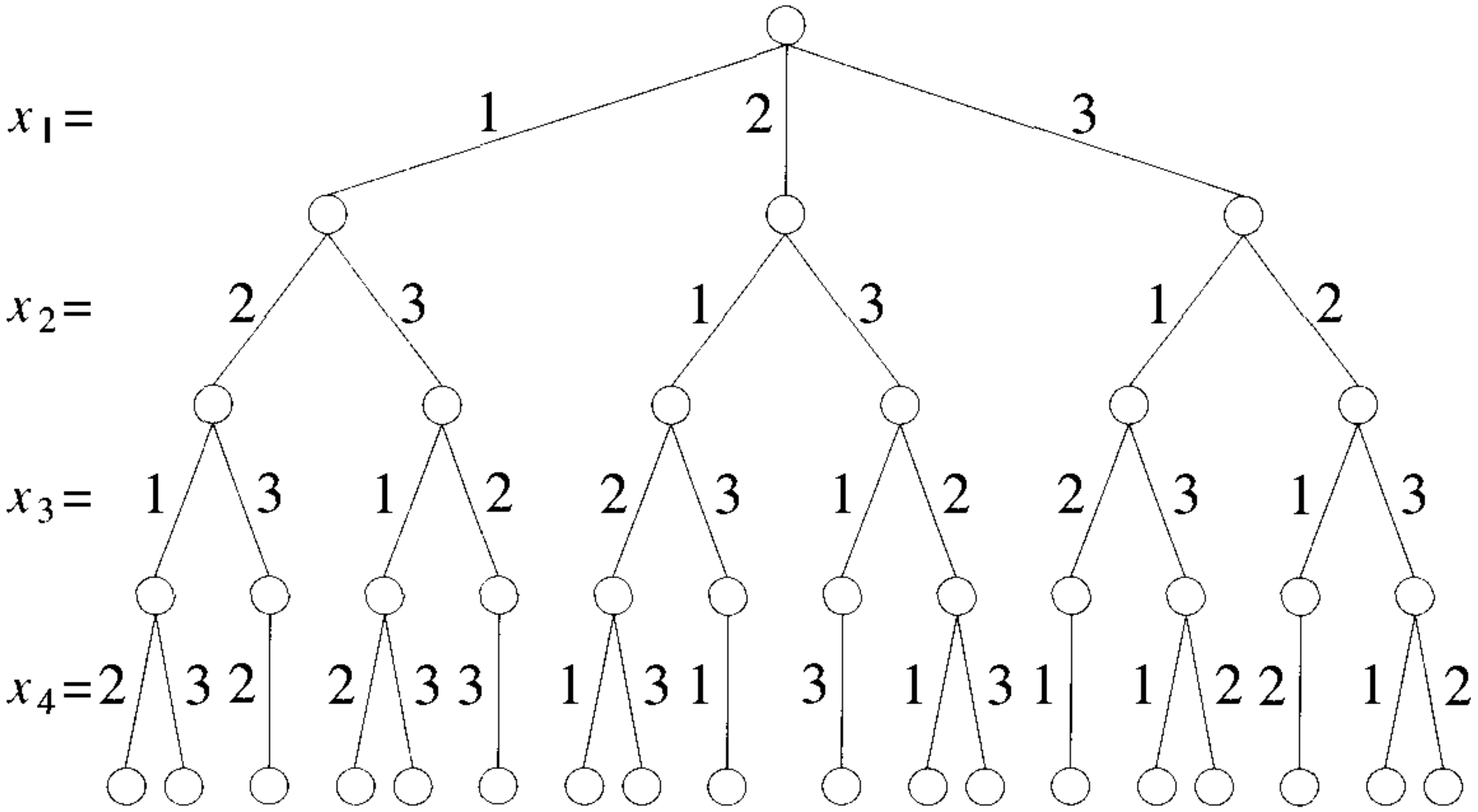


Hasil pewarnaan peta

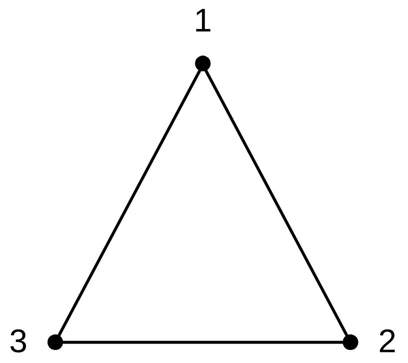
Tinjau untuk $n = 4$ dan $m = 3$.

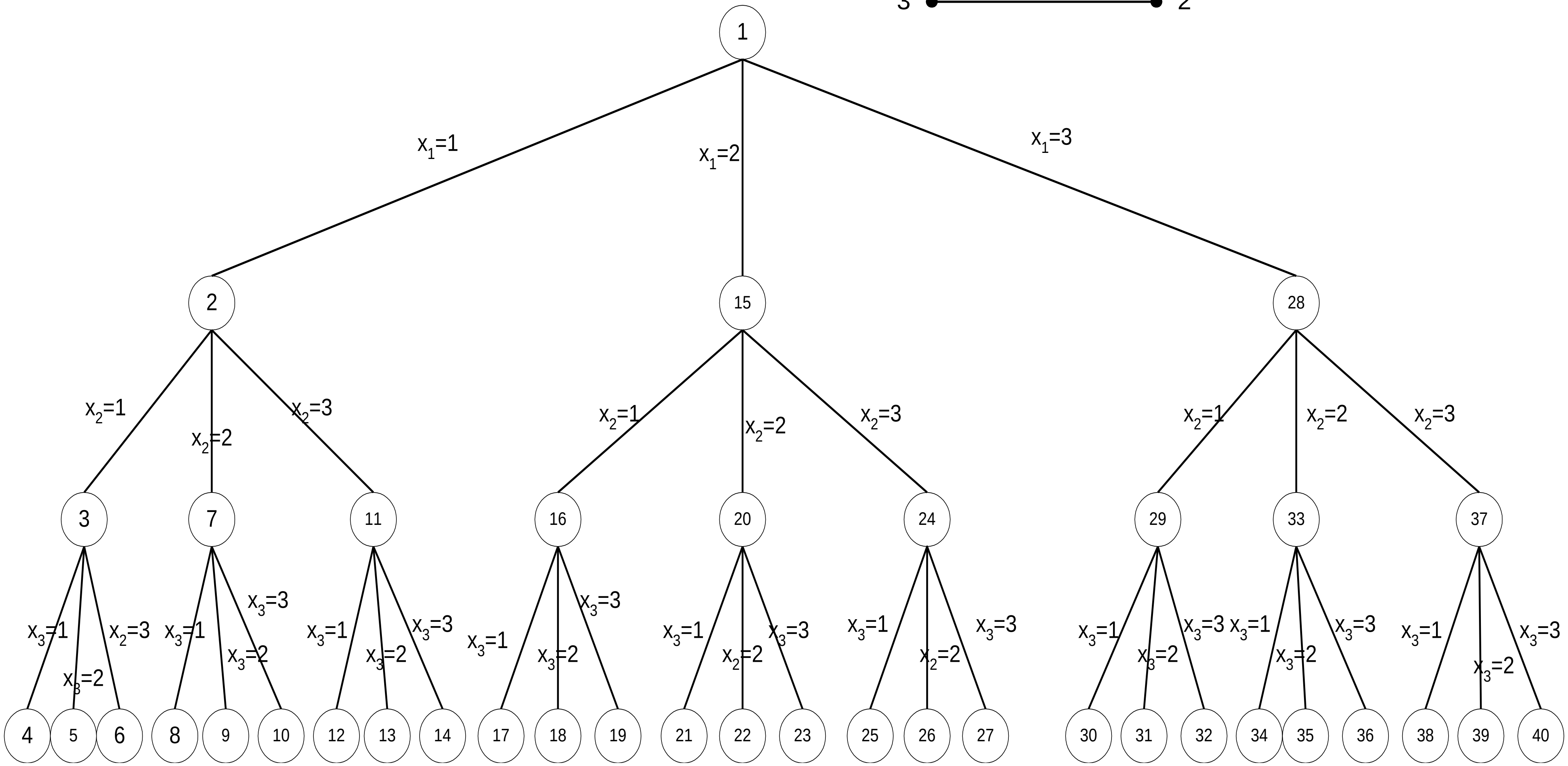
Misalkan warna dinyatakan dengan angka 1, 2, …, m dan solusi dinyatakan sebagai vektor X dengan *n-tuple*: $X = (x_1, x_2, ..., x_n)$, $x_i \in \{1, 2, …, m\}$



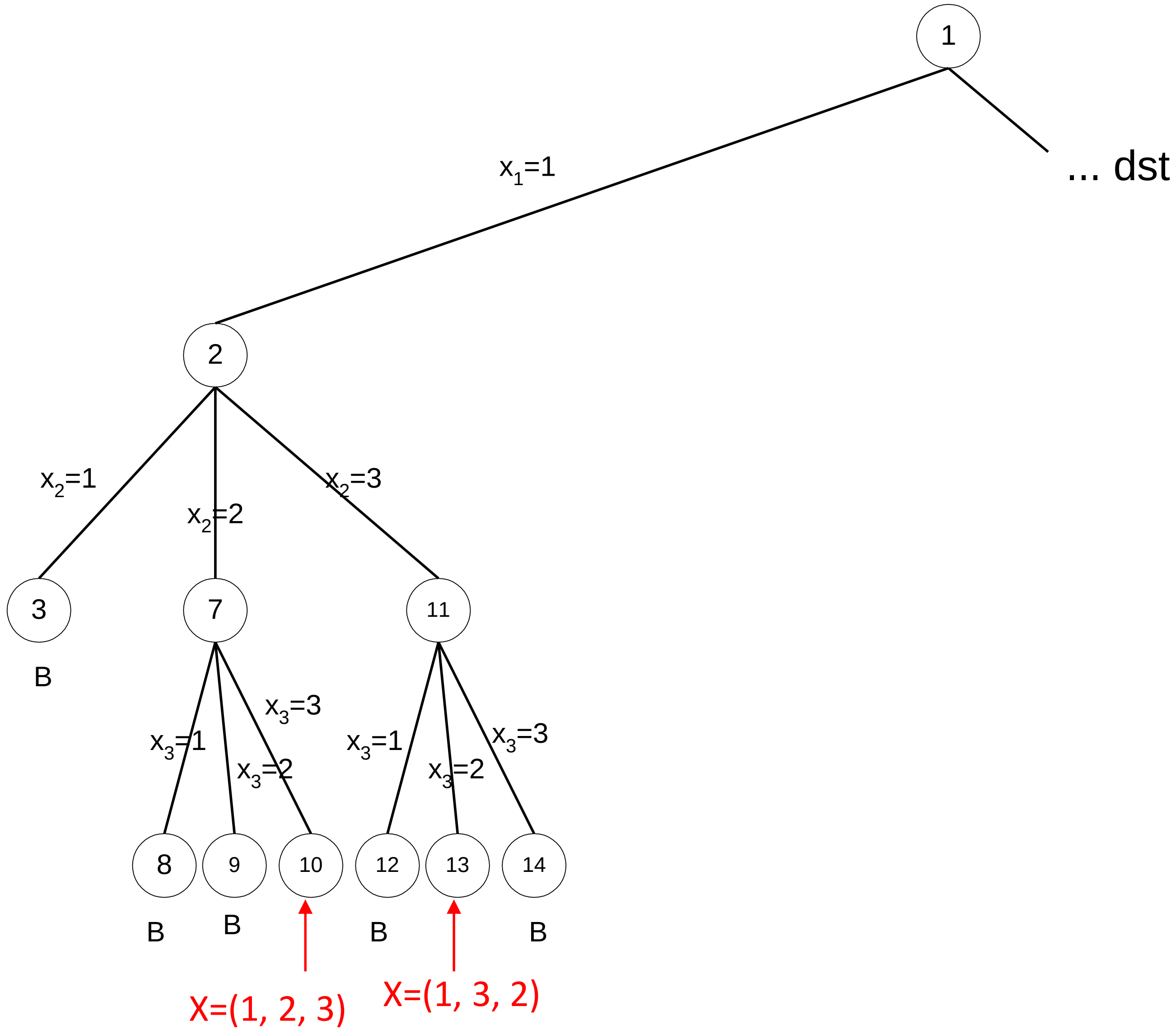


Tinjau untuk *n* = 3 dan *m* = 3.





Pencarian solusi secara *backtracking*:



Algoritma Runut-balik Untuk Pewarnaan Graf

- Masukan:

  1. Matriks ketetanggaan G[1..n, 1..n]

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

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

  2. Warna

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

- Luaran:

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

- <u>Algoritma</u>:

  1. Inisialisasi x[1..n] dengan 0 sebagai berikut:

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

$\quad x[i] \leftarrow 0$

endfor

2. Panggil prosedur *PewarnaanGraf*(1)

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

Algoritma:
  stop ← false
  while not stop do
      WarnaiSimpul(k)      {coba isi x[k] dengan sebuah warna}
      if  x[k] = 0  then        {tidak ada warna lagi yang bisa dicoba,  habis}
          stop ← true
      else
          if  k = n  then                         {apakah seluruh simpul sudah diwarnai?}
              write(x[1], x[2], …, x[k])      { cetak solusi }
          else
              PewarnaanGraf(k + 1)        {warnai simpul berikutnya}
      endif
    endif
  endwhile
```

```
procedure WarnaiSimpul(input k : integer)
{ Menentukan warna untuk simpul k
 Masukan: simpul ke-k
 Luaran: nilai untuk x[k]
}
Deklarasi
    stop, keluar : boolean
    j : integer

Algoritma:
  stop ← false
  while not stop do
       x[k] ← (x[k]+1) mod (m+1)      { bangkitkan warna untuk simpul ke-k}
       if  x[k] = 0  then                      {semua warna telah terpakai}
           stop ← true
       else
           {periksa warna simpul-simpul tetangganya}
           …
```

```
          for  j ← 1 to n do
              if  (G[k, j])       {jika ada sisi dari simpul k ke simpul j}
                    and                 {dan}
                 (x[k] = x[j])    {warna simpul k = warna simpul j }
              then
                  exit loop       {keluar dari kalang}
              endif
          endfor

          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*], $1 \le i \le n$.

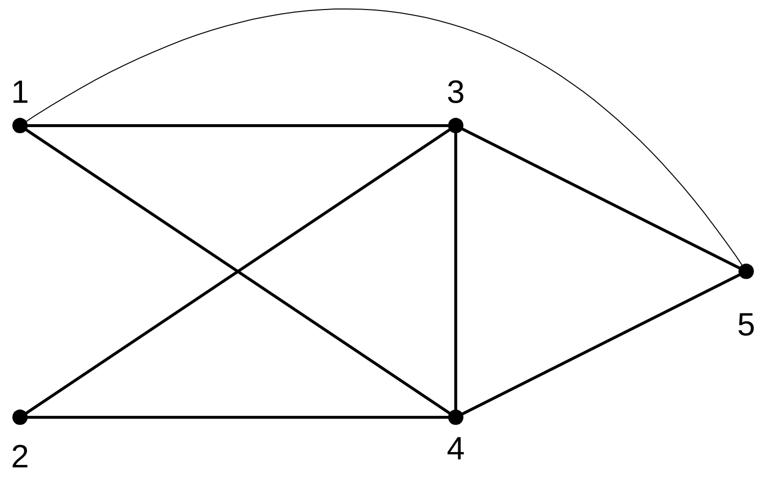
- Simpul pada aras *n* adalah simpul daun. Jumlah simpul internal (simpul bukan daun) adalah $\sum_{i=0}^{n-1} m^i$

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

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

4. Sirkuit Hamilton

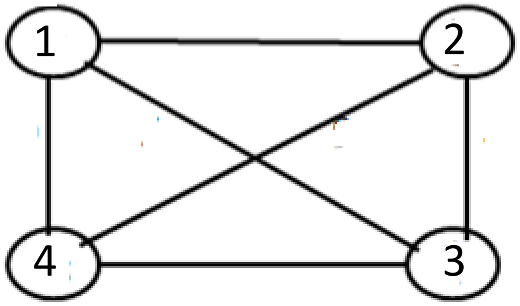
- Persoalan: Diberikan graf terhubung $G = (V, E)$ dengan n buah simpul. Temukan semua sirkuit (atau siklus) Hamilton dalam graf itu. Sirkuit Hamilton adalah perjalanan yang mengunjungi semua simpul tepat satu kali dan kembali lagi ke simpul awal.
- Contoh:



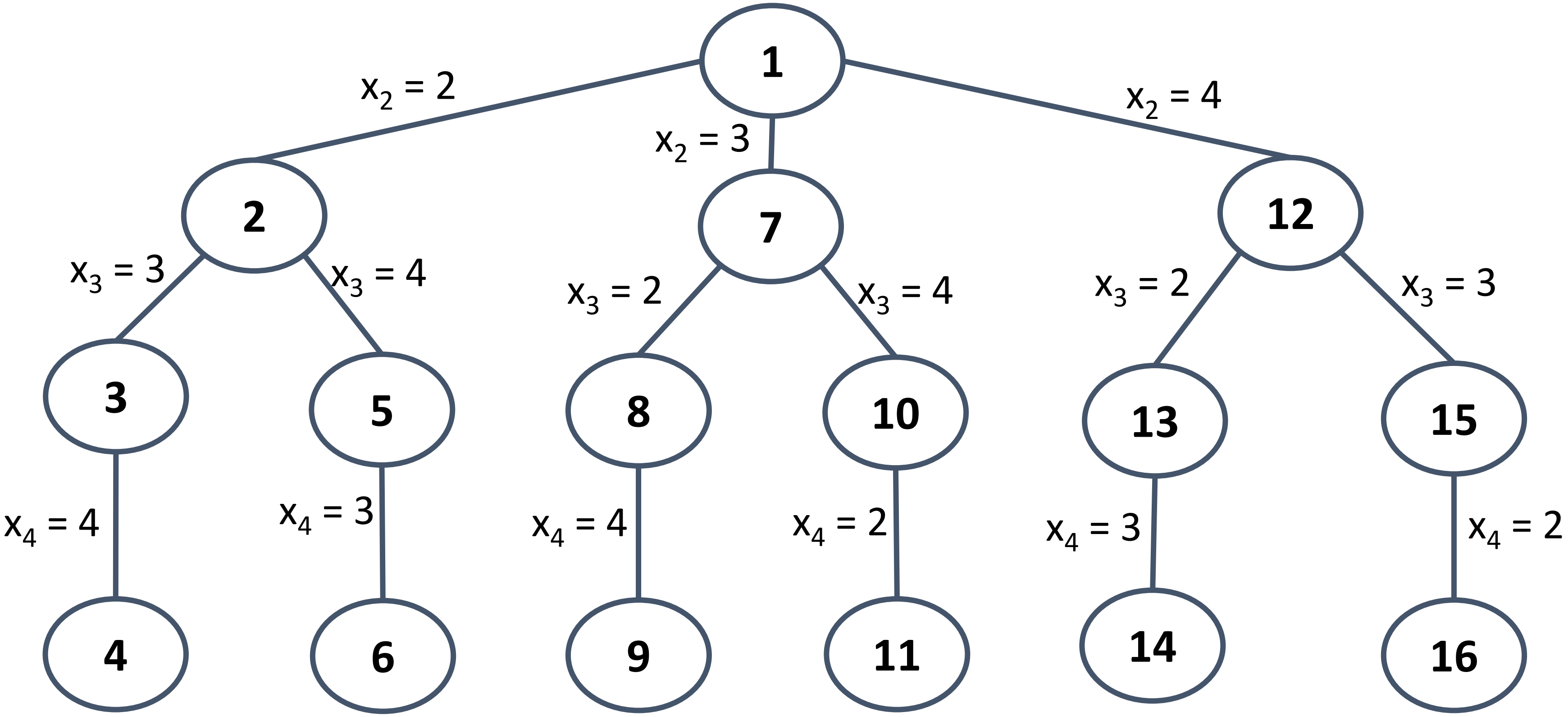
Sirkuit Hamiltonnya adalah (dimulai dari simpul 1:

1, 3, 2, 4, 5, 1
1, 4, 2, 3, 5, 1
1, 5, 4, 2, 3, 1
1, 5, 3, 2, 4, 1

Pohon ruang status berdasarkan graf G (sirkuit Hamilton dimulai dari 1)



Graf G



Algoritma Runut-balik Sirkuit Hamilton

Masukan: Matriks G[1..n, 1..n] { *n* = jumlah simpul graf}
G[i,j] = true jika ada sisi dari simpul i ke simpul j
G[i,j] = false jika tidak ada sisi dari simpul i ke simpul j

Luaran: Vektor X[1..n], yang dalam hal ini, x[i] adalah simpul i di dalam sirkuit Hamilton.

Algoritma:

1. Inisialisasi x[2..n] dengan 0, sedangkan x[1] diisi dengan 1 (karena diasumsikan siklus Hamilton dimulai dari simpul 1) sebagai berikut:.

   $x[1] \leftarrow 1$
   for $i \leftarrow 2$ **to** n **do**
   $\quad x[i] \leftarrow 0$
   endfor

2. Panggil prosedur *SirkuitHamilton*(2)

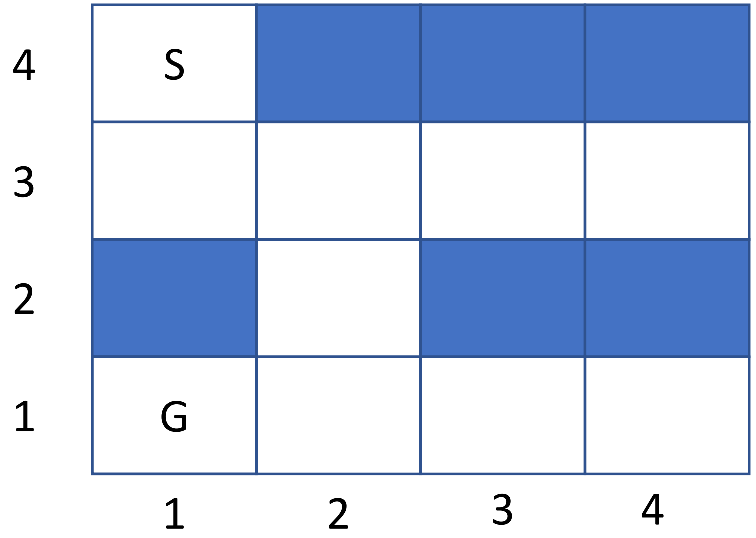
```
procedure SirkuitHamilton(input  k : integer)
{ Menemukan semua sirkuit Hamilton pada graf terhubung. Sirkuit dimulai  dari simpul 1
Masukan: k adalah nomor simpul graf
Luaran: jika solusi ditemukan, solusi dicetak ke piranti keluaran
}
Deklarasi
  stop : boolean

Algoritma:
  stop ← false
  while not stop do
      {tentukan semua nilai untuk x[k] }
      SimpulBerikutnya(k)     {isi x[k] dengan simpul berikutnya}
      if  x[k] = 0  then             {tidak ada simpul lagi, habis}
         stop←true
      else
         if  k = n  then                        {seluruh simpul sudah dikunjungi}
            write(x[1], x[2], …, x[n])  {cetak sirkuit Hamilton}
        else
           SirkuitHamilton(k+1)     {cari simpul berikutnya}
        endif
     endif
  endwhile
```

```
procedure SimpulBerikutnya(input k : integer)
{ Menentukan simpul berikutnya untuk membentuk sirkuit Hamilton
Masukan: k
Luaran: nilai untuk x[k]
Keterangan: x[1], x[2], ..., x[k-1] adalah lintasan yang terdiri atas k – 1 simpul berbeda.
x[k] berisi simpul berikutnya dengan nomor yang lebih tinggi yang:
            (i)  belum terdapat di dalam { x[1], x[2], ..., x[k-1]}
            (ii) terhubung oleh sebuah sisi ke x[k-1]
Jika tidak memenuhi kedua kondisi itu, maka x[k] = 0. Jika k = n,  maka harus diperiksa apakah x[k]
terhubung ke x[1] }
}
Deklarasi
   stop, sama : boolean
   j : integer

Algoritma:
  stop ← false
  while not stop do
     x[k] ← (x[k] + 1) mod (n + 1);       {pembangkitan simpul berikutnya}
     if  x[k] = 0 then
        stop←true
     else
```

```
          if  G[x[k – 1], x[k]]    {ada sisi dari x[k] ke x[k-1]}  then
                { periksa apakah x[k] berbeda dengan simpul-simpul x[1],  x[2], ..., x[k-1] }
                sama ← false
                j←1
                while (j ≤ k – 1) and (not sama) do
                     if  x[j] = x[k] then   sama ← true  else  j ← j + 1 endif
                endwhile
               { j > k – 1  or sama }

               if  not sama {berarti simpul x[k] berbeda} then
                    if  (k < n) {belum semua simpul dikunjungi}
                                    or           { atau }
                           ((k = n)  and (G[x[n], 1]))   {ada sisi dari x[n] ke x[1]}   then
                         stop ← true
                    endif
               endif
          endif
       endif
    endwhile
```

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*.



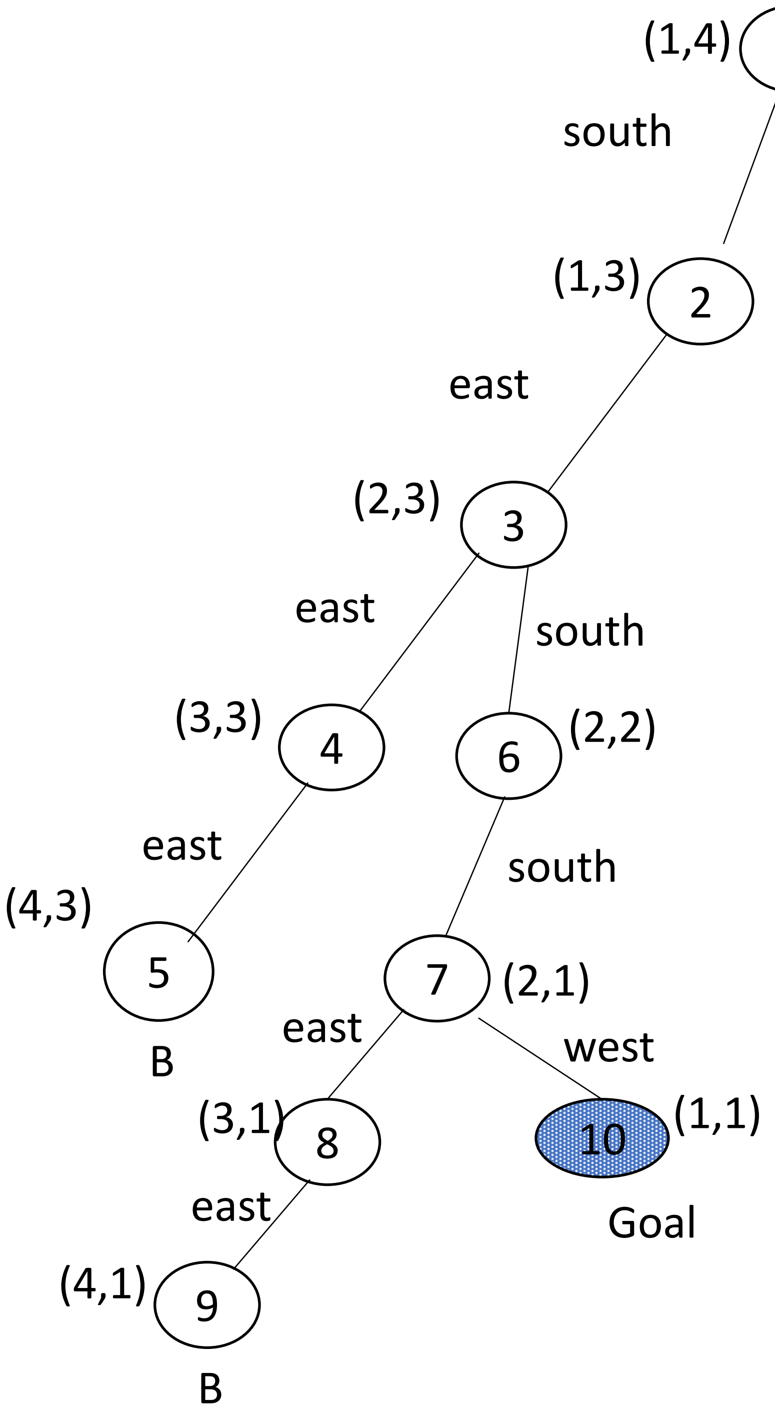
Buatlah pohon pencarian jalur ke titik Goal(4,1)dengan menggunakan Algoritma Backtracking, dimulai dari titik (1,4). Tulislah nomor urutan pembangkitan pada setiap simpul pohon pencarian. Pencarian dihentikan ketika sudah mencapai titik G. Kemudian tuliskan hasil urutan aksiyang dilakukan untuk mencapai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

Penyelesaian:

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

  $x_i \in \{\textit{east}, \textit{south}, \textit{west}, \textit{north}\}$
- Fungsi T(.) mencoba meng-assign x_i dengan urutan *east*, *south*, *west*, *north*

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



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

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