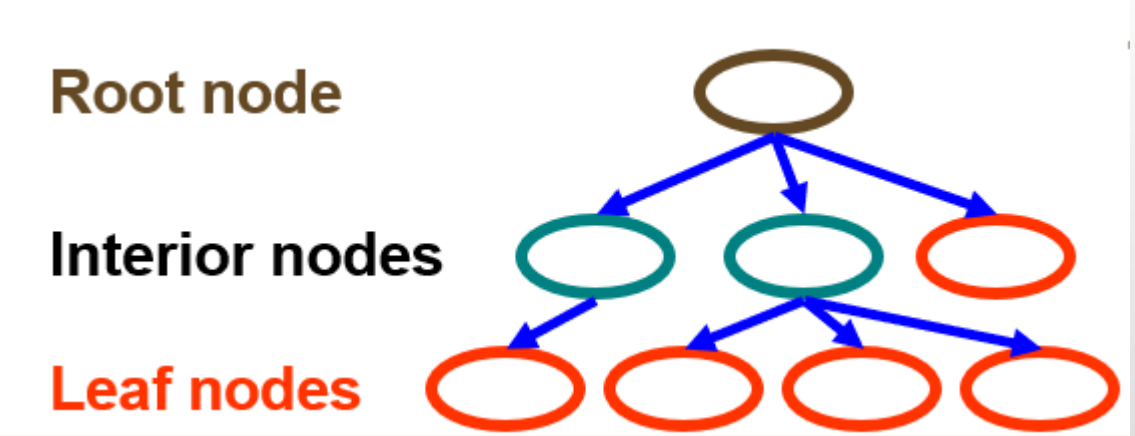
IN244 – Strategi Algoritmik

o8 - Backtracking

MARANATHA CHRISTIAN UNIVERSITY
INTEGRITY
CARE
EXCELLENCE

Struktur Data

- Algoritma backtracking memakai struktur tree dalam pencarian solusi
- Tree : struktur graf yang tidak mengandung sirkuit
- Traversal yang dipakai dalam backtracking
  - Pencarian Mendalam
  - Depth First Search



Algoritma Backtracking

- Runut-balik (*backtracking*) :
  - algoritma yang berbasis pada DFS untuk mencari solusi persoalan secara lebih efisien
- Backtracking :
  - perbaikan dari algoritma *brute-force*,
  - secara sistematis mencari solusi persoalan di antara semua kemungkinan solusi yang ada.
- Dengan metode runut-balik,
  - Tidak perlu memeriksa semua kemungkinan solusi yang ada.
  - Hanya pencarian yang mengarah ke solusi saja yang selalu dipertimbangkan.
  - Dengan memangkas simpul-simpul yang tidak mengarah ke solusi
- Banyak diterapkan untuk
  - program *games* (seperti permainan *tic-tac-toe*, menemukan jalan keluar dalam sebuah labirin, catur, dll)
  - masalah-masalah pada bidang kecerdasan buatan (*artificial intelligence*).

PROPERTI UMUM

- Solusi persoalan dinyatakan sebagai vektor dengan *n-tuple*:
  - $X = (x_1, x_2, ..., x_n)$, $x_i \in$ himpunan berhingga S_i.
  - Mungkin saja $S_1 = S_2 = ... = S_n$.
  - Contoh: $S_i = \{0, 1\}$, $x_i = 0$ atau 1
- **Fungsi pembangkit** nilai x_k
  - Dinyatakan sebagai $T()$
  - $T(x[1],x[2], ... x[n])$ membangkitkan nilai untuk x_k yang merupakan komponen vektor solusi.
- **Fungsi pembatas** (pada beberapa persoalan fungsi ini dinamakan fungsi kriteria)
  - Dinyatakan sebagai $B(x_1, x_2, ..., x_k)$
  - Fungsi pembatas menentukan apakah $(x_1, x_2, ..., x_k)$ mengarah ke solusi.
  - Jika bernilai True, maka pembangkitan nilai untuk x_{k+1} dilanjutkan,
  - jika False, maka $(x_1, x_2, ..., x_k)$ dibuang dan tidak dipertimbangkan lagi dalam pencarian solusi.

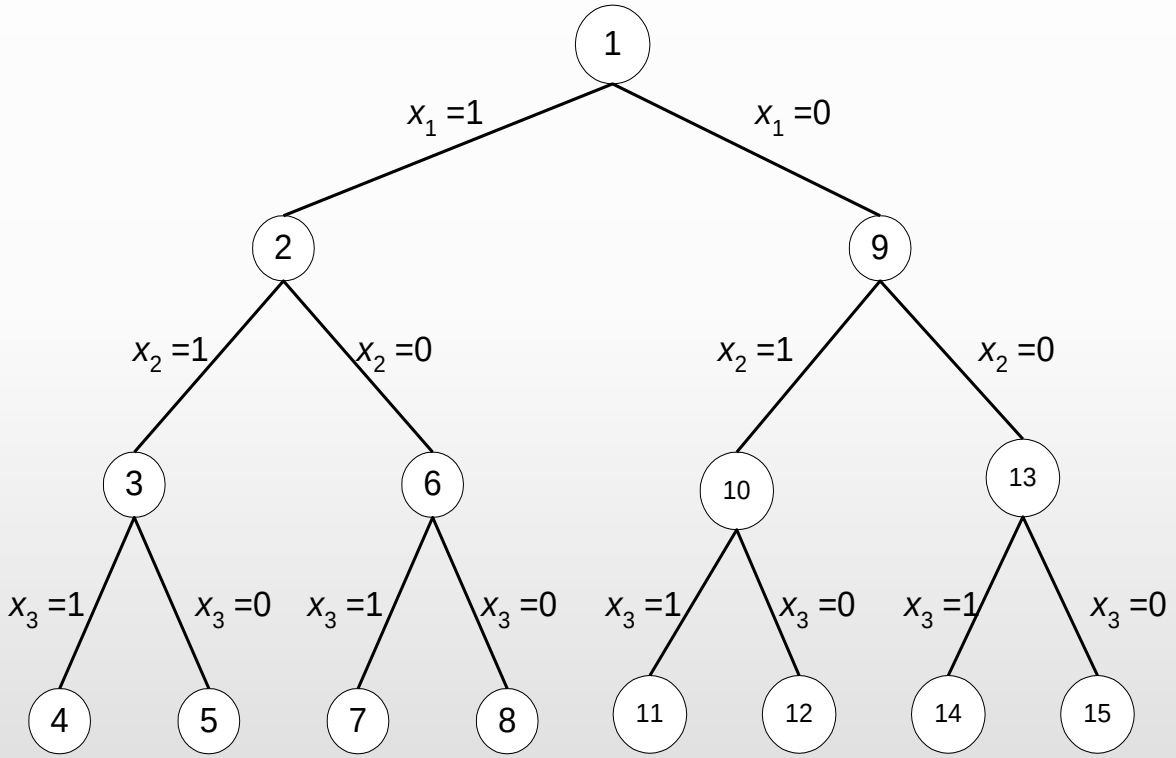
Organisasi Solusi

- 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 persoalan *Knapsack* 0/1 untuk $n = 3$.
  - Solusi persoalan dinyatakan sebagai vektor (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 persoalan *Knapsack* 0/1 dengan $n = 3$ terdapat $2^n = 2^3 = 8$ kemungkinan solusi

Organisasi Solusi

- Ruang solusi diorganisasikan ke dalam struktur pohon.
- Tiap simpul pohon menyatakan status (*state*) persoalan, sedangkan sisi (cabang) diberi label 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*).

Ruang Solusi utk Knapsack 0/1 dengan n=3



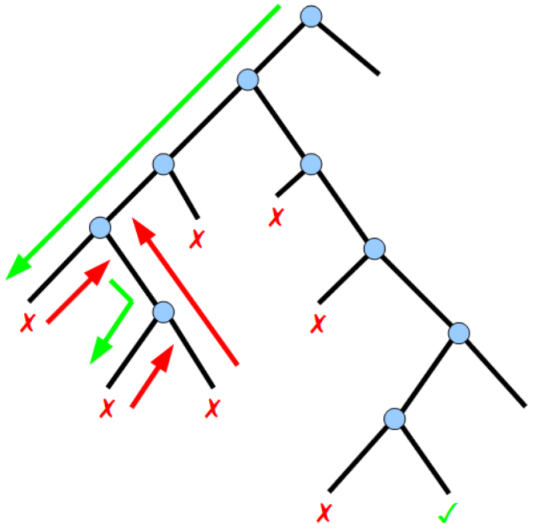
- Simpul Akar : simpul 1
- Simpul Daun :
  - Simpul 4, 5, 7, 8, 11,12, 14,15

Prinsip Pencarian Solusi

- 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.

Prinsip Pencarian Solusi

- 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.



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

Contoh Knapsack 0/1

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

  $n = 3$

  $(w_1, w_2, w_3) = (32, 25, 20)$

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

  $M = 30$

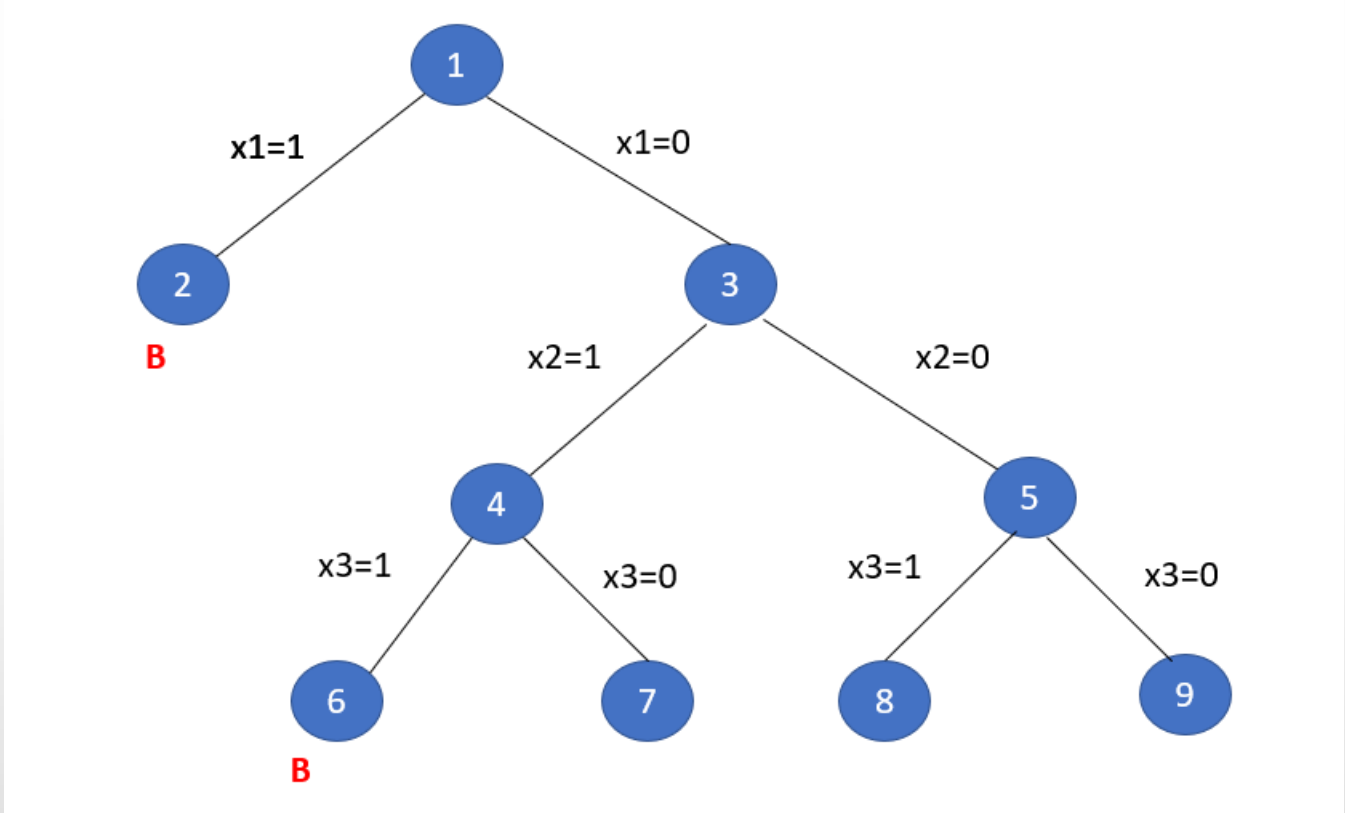
- Solusi dinyatakan sebagai $X = (x_1, x_2, x_3)$, $x_i \in \{0, 1\}$.

- Fungsi pembatas:

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

Contoh Knapsack 0/1

- Pohon pencarian solusi yang terbentuk
- Solusi yang layak :
  - *X1* = (0, 1, 0) dan *F* = 25
  - *X2* = (0, 0, 1) dan *F* = 50
- Solusi optimum:
  - *X2* = (0, 0, 1) dan *F* = 50



Algo Backtracking (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
```

Pemanggilan prosedur pertama kali: **RunutBalikR(1)**

Algo Backtracking (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
```

- Pemanggilan prosedur pertama kali: **RunutBalikI(n)**

Komputasi Backtracking

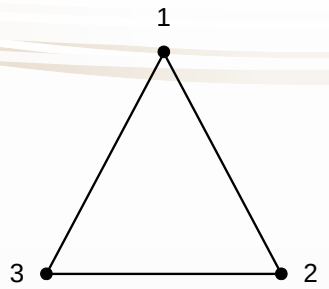
- Setiap simpul dalam pohon ruang status berasosiasi dengan sebuah pemanggilan rekursif.
- Jika jumlah simpul dalam pohon ruang status adalah 2^n atau $n!$, maka untuk kasus terburuk,
  - algoritma runut-balik membutuhkan waktu dalam $O(p(n)2^n)$ atau $O(q(n)n!)$,
  - dengan $p(n)$ dan $q(n)$ adalah polinom derajat n yang menyatakan waktu komputasi setiap simpul

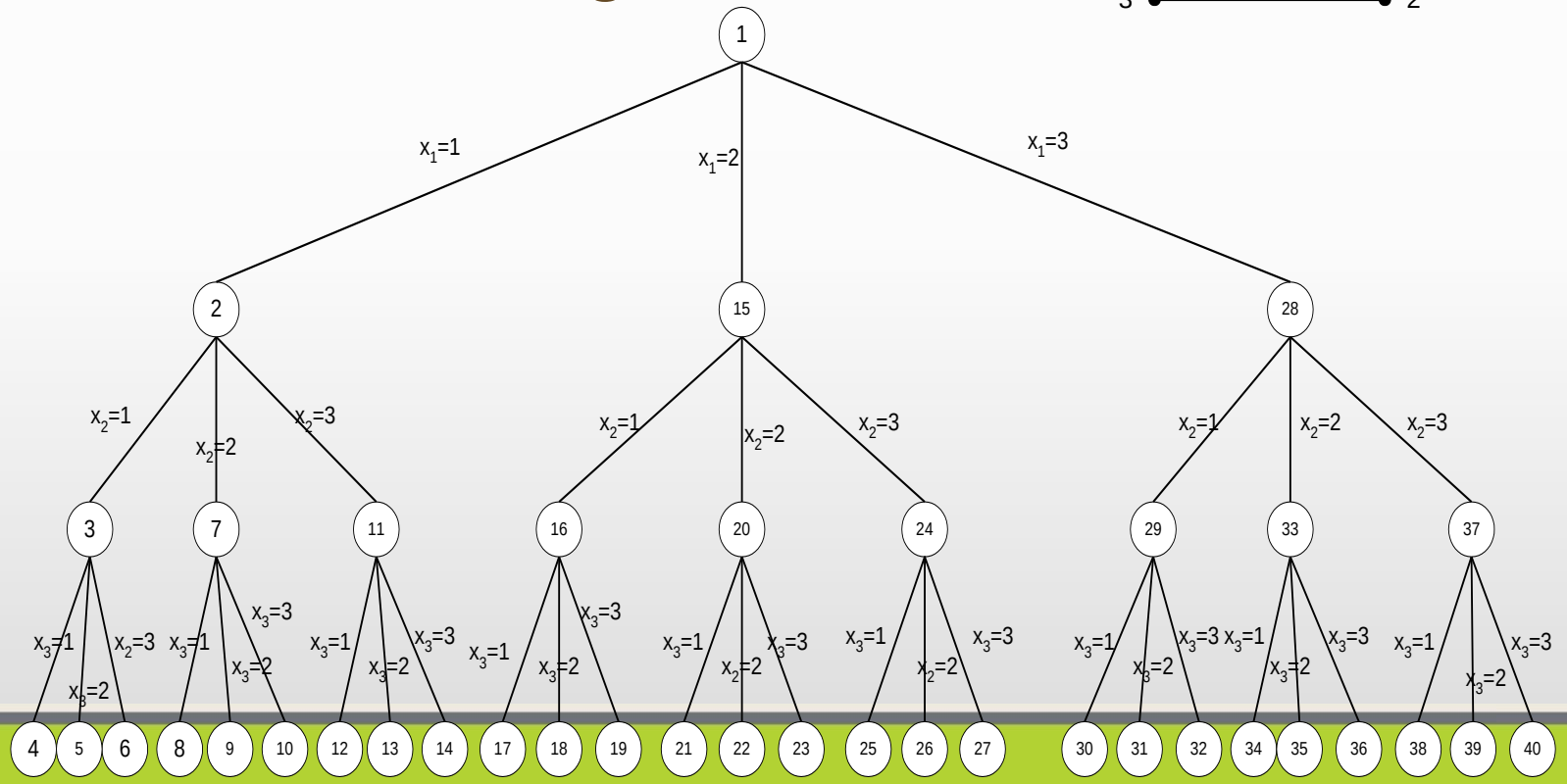
Graph colouring (pewarnaan graf)

- Diberikan sebuah graf *G* dengan *n* buah simpul dan disediakan *m* buah warna.
- Warnailah seluruh simpul graf *G* sedemikian sehingga tidak ada dua buah simpul bertetangga yang mempunyai warna sama
- Perhatikan juga bahwa tidak seluruh warna harus dipakai
- Aplikasi : pewarnaan peta

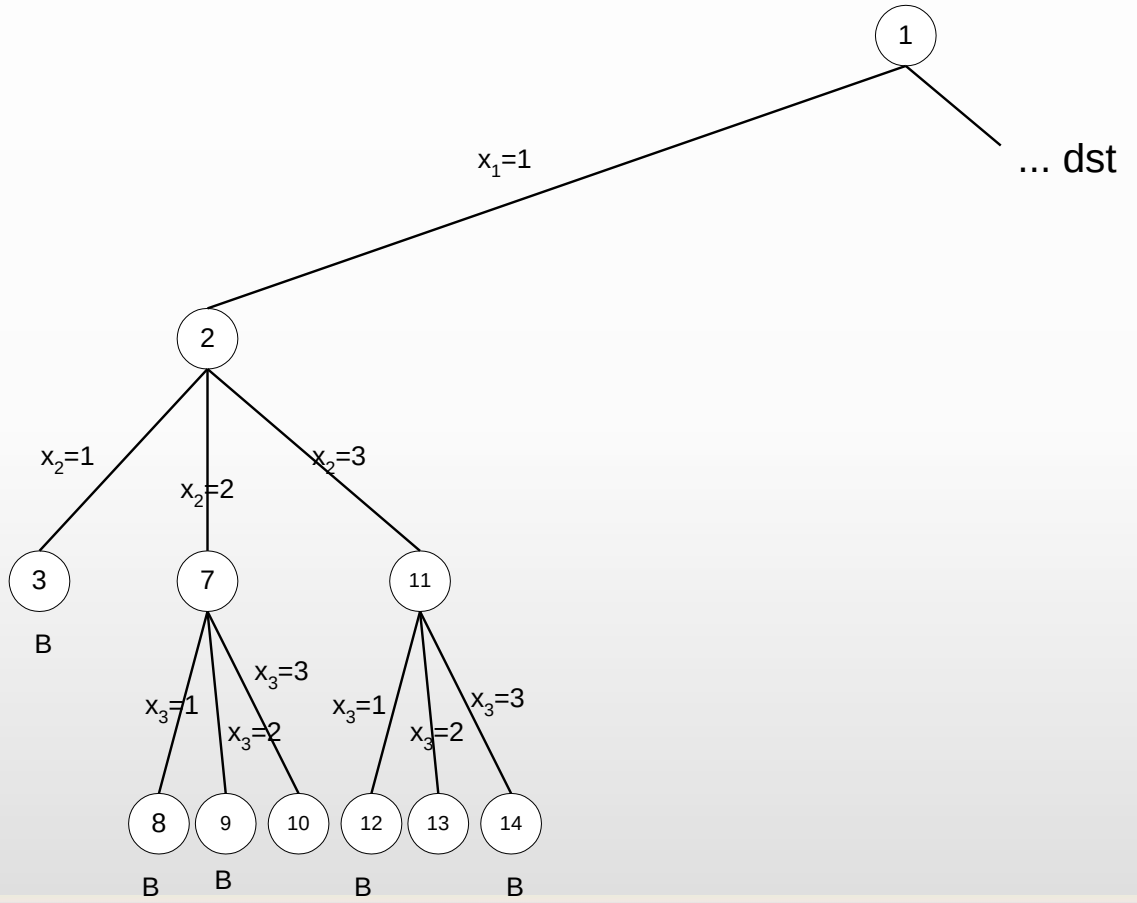
Contoh Persoalan

- Graf dengan n = 3, m = 3
- Pohon ruang status statis

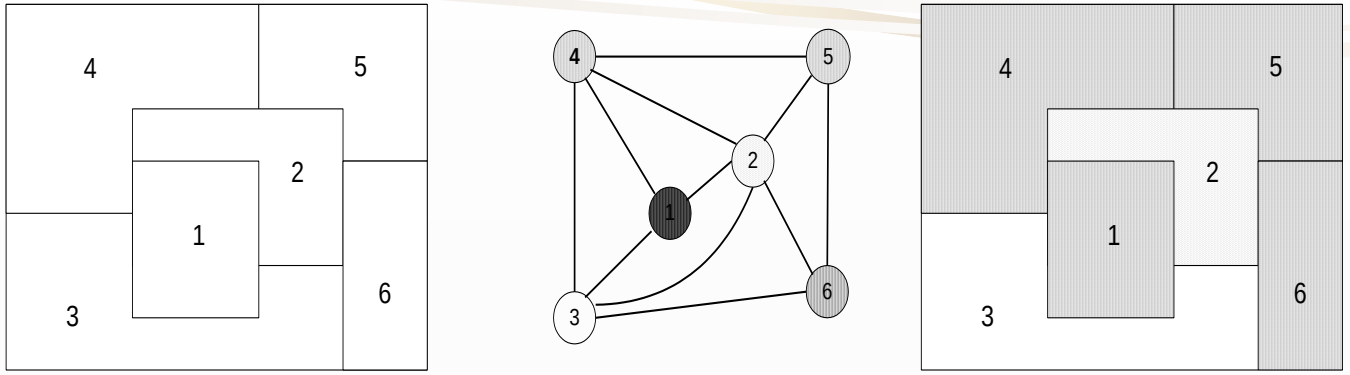




- Pohon ruang status dinamis pd saat backtrack



Pewarnaan peta

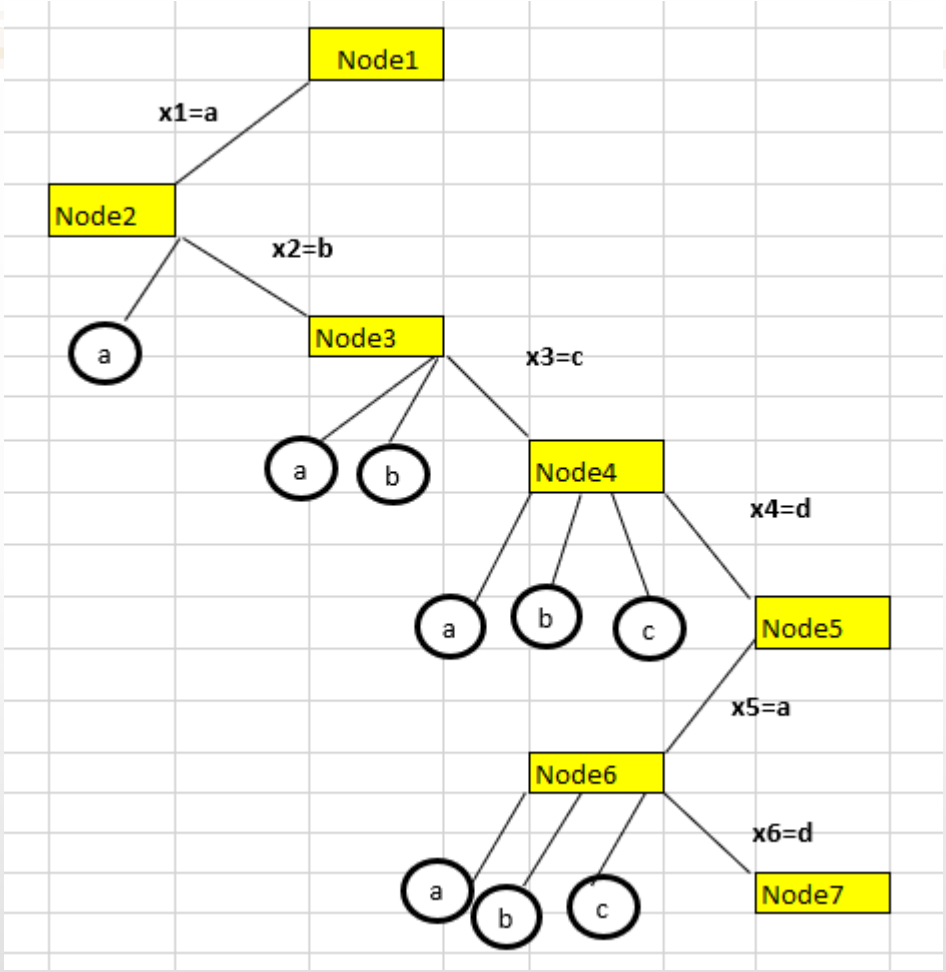


(a) (b) (c)

- Contoh : n (simpul) = 6, m (warna) = 4
- Gambar (a) : Peta dgn 6 wilayah
- Gambar (b) : Graf representasi peta
- Gambar (c) : Peta yg sudah diwarnai

Pewarnaan peta

- Pohon pencarian solusi yang terbentuk
- Warna : a,b,c,d
- Solusi :
  - $X1$ = a
  - $X2$ = b
  - $X3$ = c
  - $X4$ = d
  - $X5$ = a
  - $X6$ = d



Algoritma Backtrack utk Pewarnaan Graf

- Masukan:
  - 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)
  - Warna
    - Dinyatakan dengan integer 1, 2, ...,m
- Keluaran:
  - Tabel X[1..n],
  - x[i] adalah warna untuk simpul i.

Algoritma Pewarnaan Graf

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

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

  - Panggil prosedur PewarnaanGraf(1)
- Kompleksitas algoritma : $O(nm^n)$

Deklarasi Data

```
{ Kamus global }
Deklarasi
    const n = 100 { jumlah simpul graf }
    const m = 4 { jumlah warna yang disediakan }
    type matriks = array[1..n,1..n] of boolean
    type tabel = array[1..n] of integer;
    GRAF : matriks
    x : tabel
```

```
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  (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
```

Referensi

- Anany Levitin, Introduction to The Design and Analysis of Algorithms, Pearson, 2012.
- S.Dasgupta, et al. Algorithms, 2006.
- Cormen, et al. Algorithms, MGH,2009.
- Strategi Algoritma – Rinaldi Munir (ITB) :
  - https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/stmik.htm
- Khan Academy, Computer Science: Algorithm, https://www.khanacademy.org/computing/computer-science/algorithms