
Algoritma Traversal di dalam Graf

- Traversal di dalam graf berarti mengunjungi simpul-simpul dengan cara yang sistematik.

- Algoritma traversal untuk graf:

  1. Pencarian Melebar (*Breadth First Search* atau BFS),

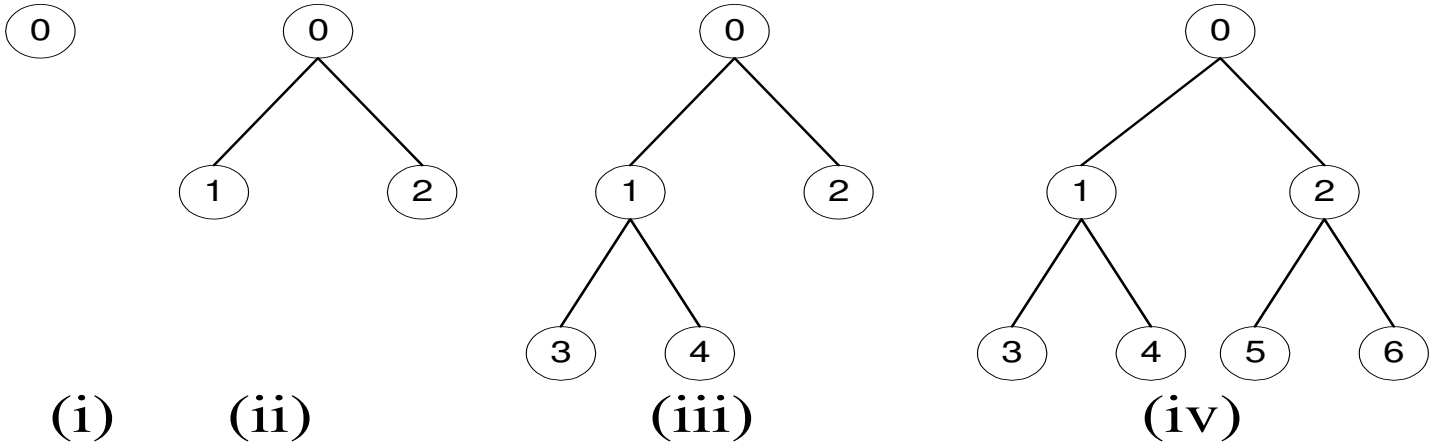
  2. Pencarian Mendalam (*Depth First Search* atau DFS).

Algoritma Pencarian Melebar (*BFS*)

- Traversal dimulai dari simpul *v*.
- Algoritma:
  1. Kunjungi simpul *v*,
  2. Kunjungi semua simpul yang bertetangga dengan simpul *v* terlebih dahulu.
  3. Kunjungi simpul yang belum dikunjungi dan bertetangga dengan simpul-simpul yang tadi dikunjungi, demikian seterusnya.

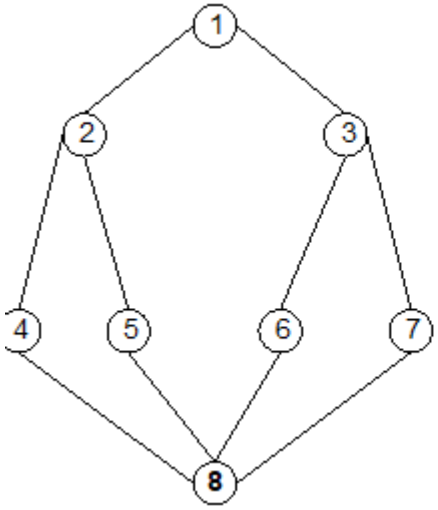
- Jika graf berbentuk pohon berakar, maka semua simpul pada aras *d* dikunjungi lebih dahulu sebelum mengunjungi simpul-simpul pada aras *d* + 1.

Metode Pencarian Melebar (BFS)

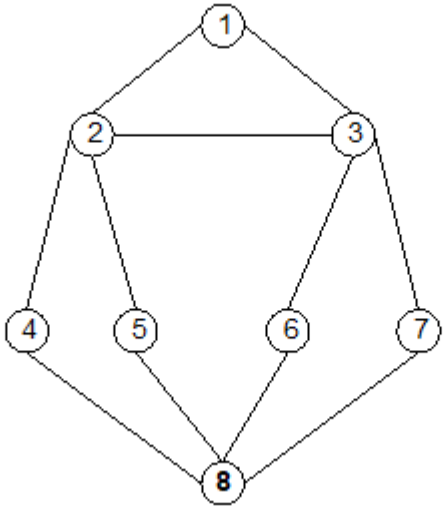


Gambar 6.4. Tahapan pembentukan pohon BFS

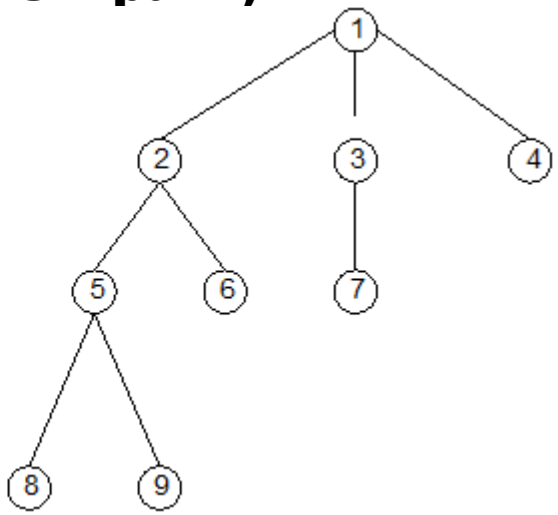
Contoh (misalkan traversal dimulai dari simpul 1)



(a)



(b)



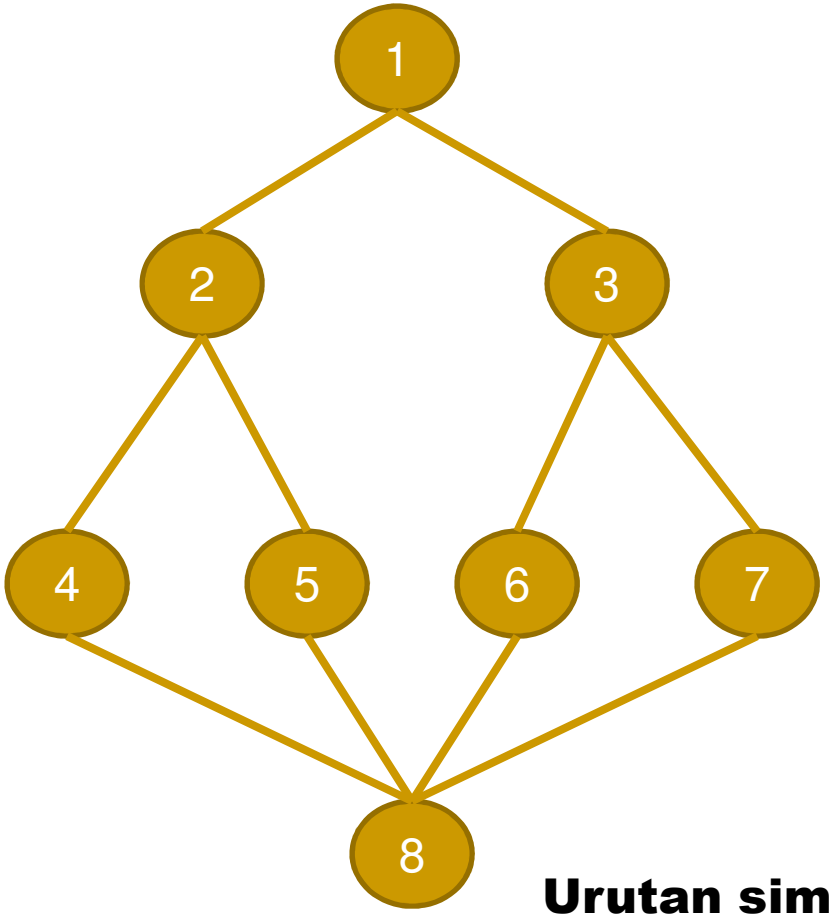
(c)

Gambar (a) BFS(1): 1, 2, 3, 4, 5, 6, 7, 8.

Gambar (b) BFS(1): 1, 2, 3, 4, 5, 6, 7, 8

Gambar (c) BFS(1): 1, 2, 3, 4, 5, 6, 7, 8, 9

BFS: Ilustrasi



Iterasi	V	Q	dikunjungi							
			1	2	3	4	5	6	7	8
Inisialisasi	1	{1}	T	F	F	F	F	F	F	F
Iterasi 1	1	{2,3}	T	T	T	F	F	F	F	F
Iterasi 2	2	{3,4,5}	T	T	T	T	T	F	F	F
Iterasi 3	3	{4,5,6,7}	T	T	T	T	T	T	T	F
Iterasi 4	4	{5,6,7,8}	T	T	T	T	T	T	T	T
Iterasi 5	5	{6,7,8}	T	T	T	T	T	T	T	T
Iterasi 6	6	{7,8}	T	T	T	T	T	T	T	T
Iterasi 7	7	{8}	T	T	T	T	T	T	T	T
Iterasi 8	8	{}	T	T	T	T	T	T	T	T

Urutan simpul2 yang dikunjungi: 1, 2, 3, 4, 5, 6, 7, 8

- *Pseudo-code* algoritma:
- Diperlukan:
  1. Matriks ketetanggaan $A = [a_{ij}]$ yang berukuran $n \times n$,

     $a_{ij} = 1$, jika simpul *i* dan simpul *j* bertetangga,

     $a_{ij} = 0$, jika simpul *i* dan simpul *j* tidak bertetangga.

  2. Antrian *q* untuk menyimpan simpul yang telah dikunjungi.

3. Tabel *boolean* yang bernama `dikunjungi`

```
dikunjungi : array[1..n] of boolean
```

dikunjungi[i] = true jika simpul *i* sudah dikunjungi
dikunjungi[i] = false jika simpul *i* belum dikunjungi

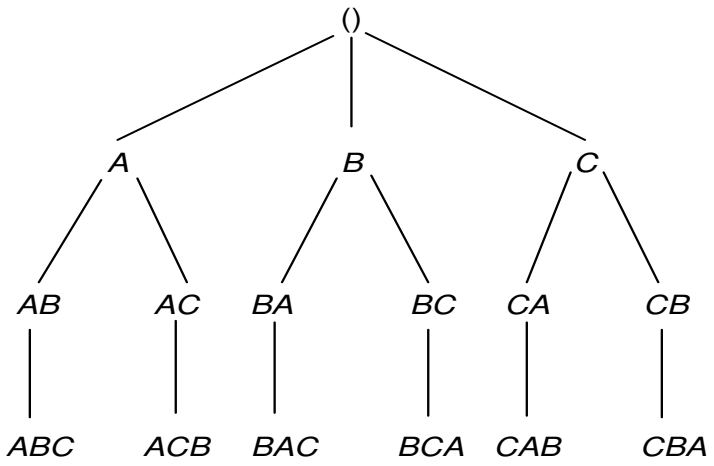
Inisialisasi tabel:

```
for i←1 to n do
    dikunjungi[i]← false
endfor
```

```
procedure BFS(input v:integer)
{ Traversal graf dengan algoritma pencarian BFS.

  Masukan: v adalah simpul awal kunjungan
  Keluaran: semua simpul yang dikunjungi dicetak ke layar
}
Deklarasi
    w : integer
    q : antrian;

    procedure BuatAntrian(input/output q : antrian)
    { membuat antrian kosong, kepala(q) diisi 0 }

    procedure MasukAntrian(input/output q:antrian, input v:integer)
    { memasukkan v ke dalam antrian q pada posisi belakang }

    procedure HapusAntrian(input/output q:antrian,output v:integer)
    { menghapus v dari kepala antrian q }

    function AntrianKosong(input q:antrian) → boolean
    { true jika antrian q kosong, false jika sebaliknya }

Algoritma:
    BuatAntrian(q)          { buat antrian kosong }

    write(v)                { cetak simpul awal yang dikunjungi }
    dikunjungi[v]←true      { simpul v telah dikunjungi, tandai dengan
                              true}
    MasukAntrian(q,v)       { masukkan simpul awal kunjungan ke dalam
                              antrian}

   { kunjungi semua simpul graf selama antrian belum kosong }
    while not AntrianKosong(q) do
         HapusAntrian(q,v)   { simpul v telah dikunjungi, hapus dari
                               antrian }
         for tiap simpul w yang bertetangga dengan simpul v do
                if not dikunjungi[w] then
                   write(w)     {cetak simpul yang dikunjungi}
                   MasukAntrian(q,w)
                   dikunjungi[w]←true
                endif
       endfor
    endwhile
   { AntrianKosong(q) }
```

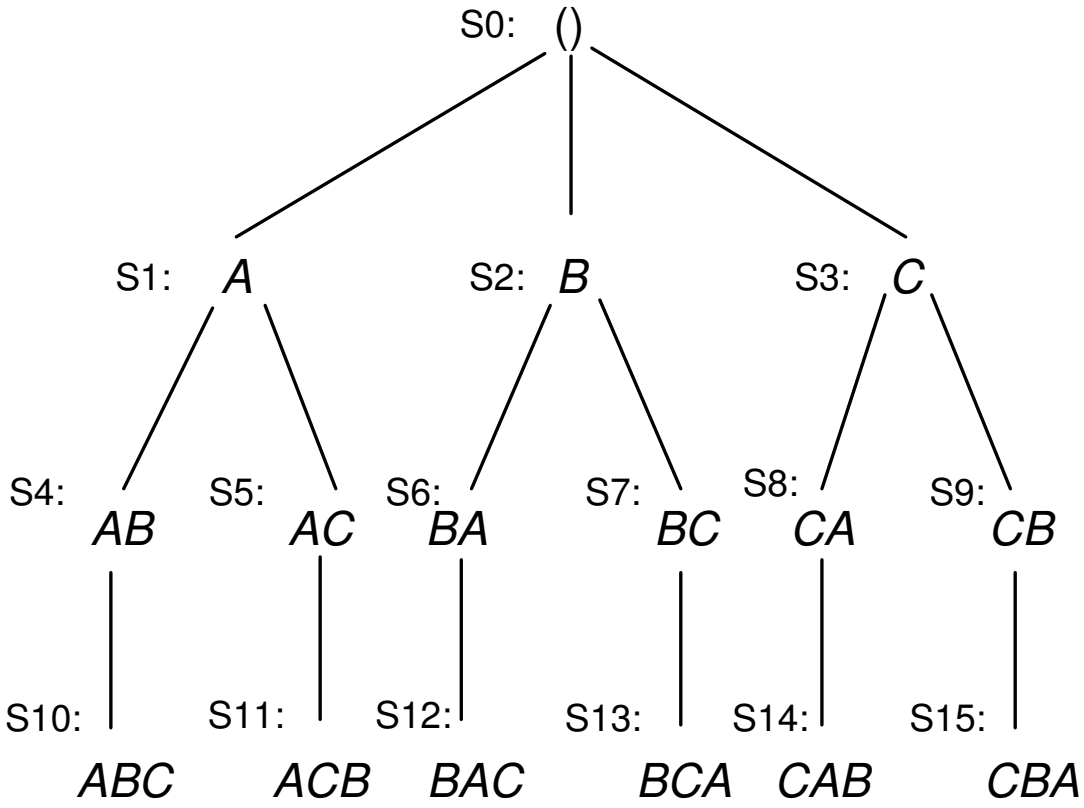
```
procedure BFS(input v:integer)
{ Traversal graf dengan algoritma pencarian BFS.

  Masukan: v adalah simpul awal kunjungan
  Keluaran: semua simpul yang dikunjungi dicetak ke layar
}
Deklarasi
   w : integer
   q : antrian;

   procedure BuatAntrian(input/output q : antrian)
   { membuat antrian kosong, kepala(q) diisi 0 }

   procedure MasukAntrian(input/output q:antrian, input v:integer)
   { memasukkan v ke dalam antrian q pada posisi belakang }

   procedure HapusAntrian(input/output q:antrian,output v:integer)
   { menghapus v dari kepala antrian q }

   function AntrianKosong(input q:antrian) → boolean
   { true jika antrian q kosong, false jika sebaliknya }

Algoritma:
   BuatAntrian(q)          { buat antrian kosong }

   write(v)                { cetak simpul awal yang dikunjungi }
   dikunjungi[v]←true      { simpul v telah dikunjungi, tandai dengan
                             true}
   MasukAntrian(q,v)       { masukkan simpul awal kunjungan ke dalam
                             antrian}

  { kunjungi semua simpul graf selama antrian belum kosong }
   while not AntrianKosong(q) do
       HapusAntrian(q,v)   { simpul v telah dikunjungi, hapus dari
                             antrian }
       for w←1 to n do
          if A[v,w] = 1 then      { v dan w bertetangga }
             if not dikunjungi[w] then
                write(w)     {cetak simpul yang dikunjungi}
                MasukAntrian(q,w)
                dikunjungi[w]←true
             endif
          endif
       endfor
   endwhile
  { AntrianKosong(q) }
```

Contoh 6.1. Pohon ruang status yang dibangkitkan untuk menghasilkan semua permutasi *A*, *B*, *C*:



Ket: () = status kosong

- Setiap simpul di dalam pohon menyatakan status persoalan.
- Status awal adalah akar yang berupa sebuah "kosong".
- Setiap daun pada pohon tersebut (ABC, ACB, BAC, BCA, CAB. Dan CBA) menyatakan status solusi, dan semua daun adalah ruang solusi.



Gambar 6.5 Pembentukan pohon ruang status persoalan pembangkitan permutasi *A*, *B*, *C* dengan metode BFS

Contoh 6.2. Permainan 8-*puzzle*:

2	1	6
4		8
7	5	3

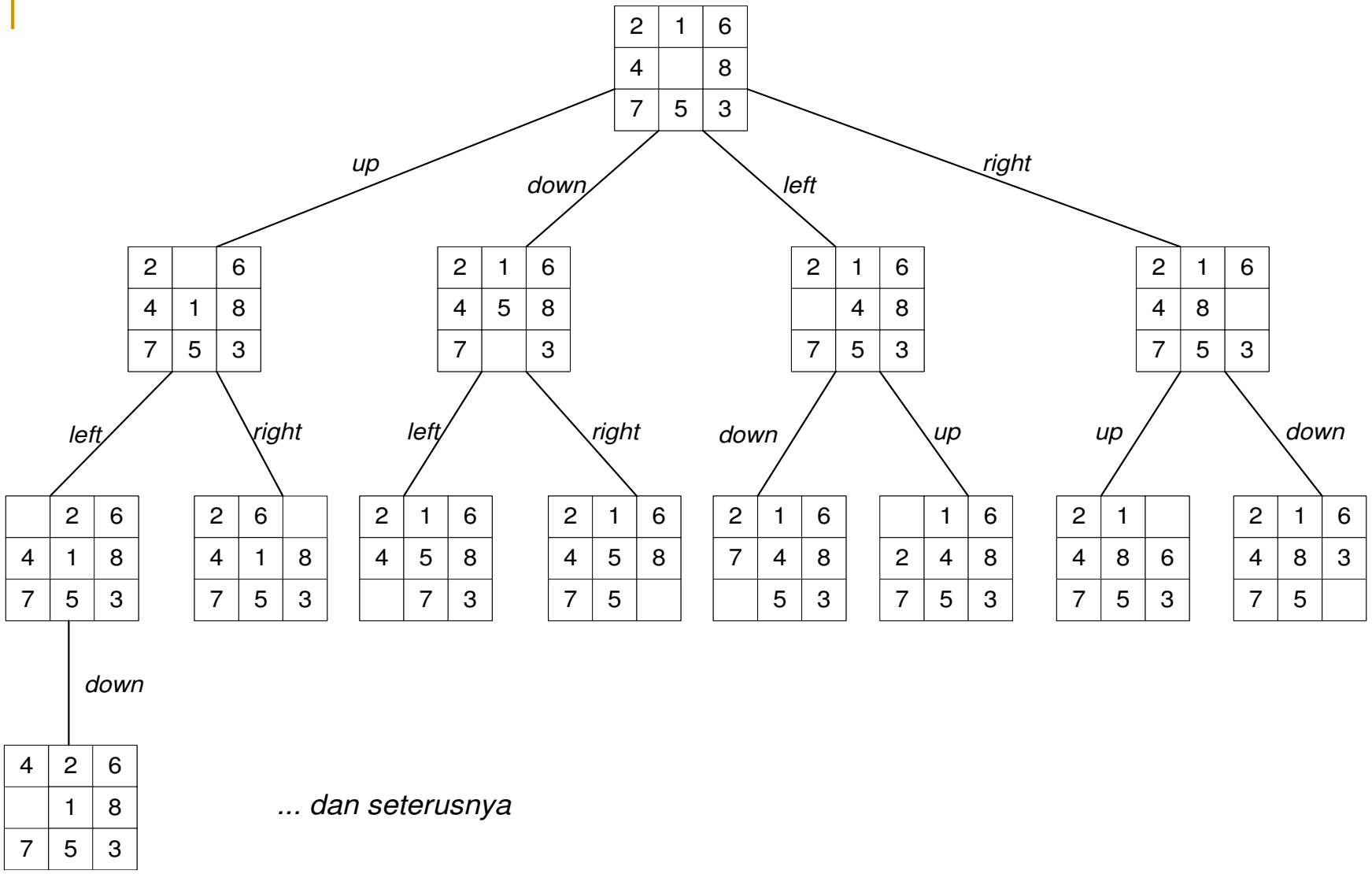
(a) Susunan awal (*initial state*)

1	2	3
8		4
7	6	5

(b) Susunan akhir (*goal state*)

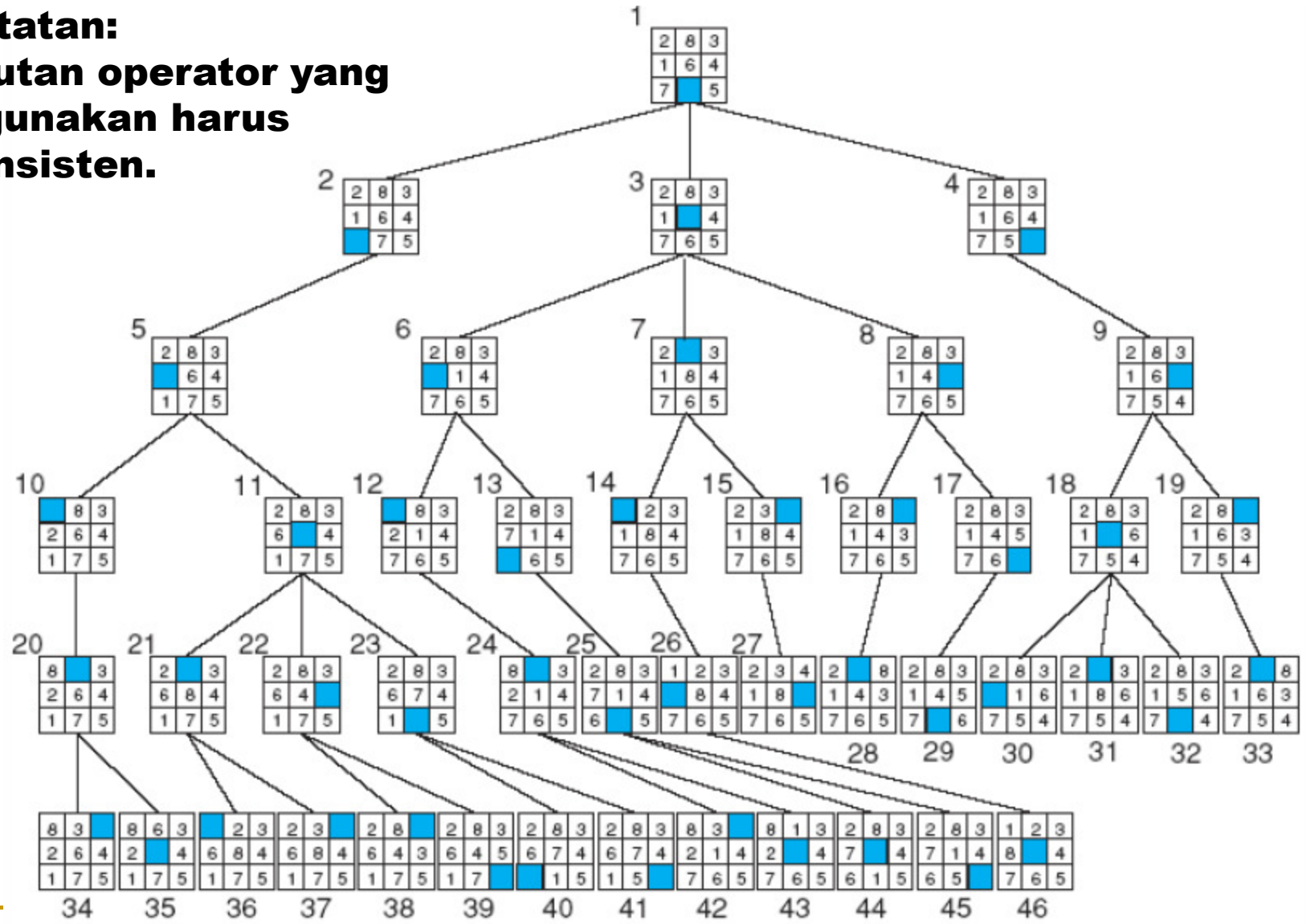
State berdasarkan ubin kosong (*blank*)
Operator: *up*, *down*, *left*, *right*

8-Puzzle: Pohon Ruang Status

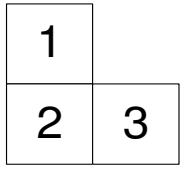


BFS untuk 8-Puzzle

Catatan:
Urutan operator yang digunakan harus konsisten.



Contoh 6.3. Sebuah mainan yang terdiri atas 3 buah blok (dinomori 1, 2, dan 3).

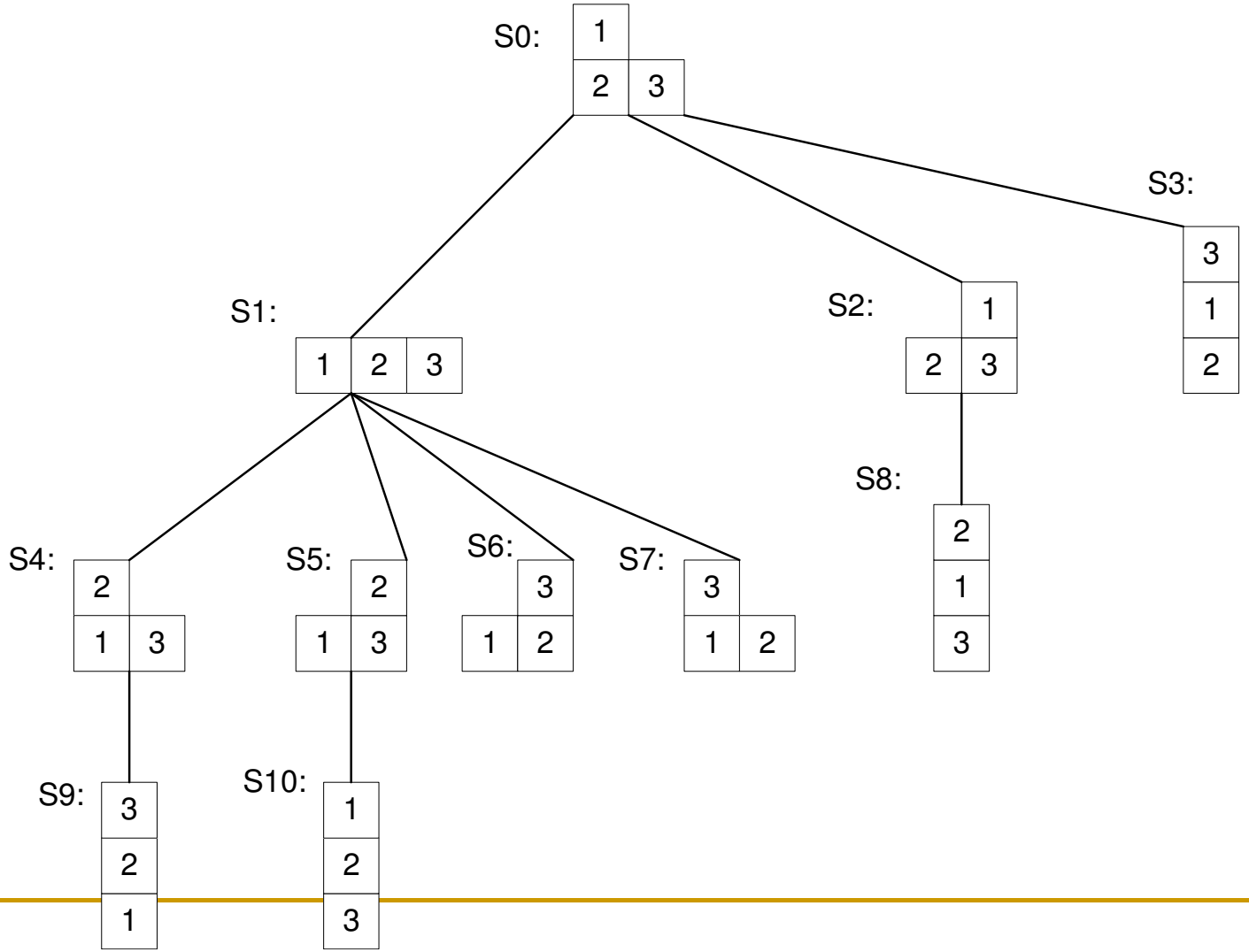


(a) Susunan awal

(b) Susunan akhir

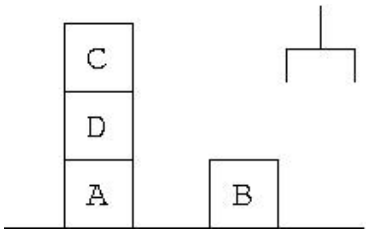
- Operator perpindahan: "PINDAHKAN X ke Y", yang berarti memindahkan objek X ke atas objek yang lain.
- Pada setiap saat, hanya satu buah blok yang boleh dipindahkan.
- Operator tidak digunakan untuk membangkitkan status yang sama lebih dari satu kali.

Pohon ruang status yang dibentuk selama pencarian solusi dengan metode BFS:

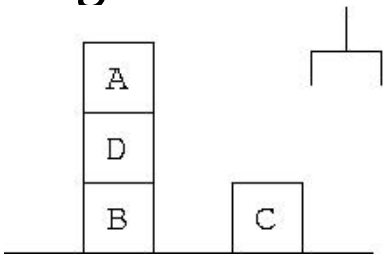


Block World Problem

- Terdapat beberapa buah balok berbentuk kubus yang ditempatkan di atas meja atau di atas balok yang lain sehingga membentuk sebuah kofigurasi. Sebuah robot yang memiliki lengan bercapit harus memindahkan balok-balok kubus tersebut sehingga membentuk konfigurasi lain dengan jumlah perpindahan yang minimum. Persyaratannya adalah hanya boleh memindahkan satu balok setiap kali ke atas balok lain atau ke atas meja. Gambarkan pohon ruang status pencarian solusi secara BFS dan DFS untuk *initial state* dan *goal state* di bawah ini. Setiap status digambarkan sebagai tumpukan balok kubus setelah pemindahan satu balok. Beri nomor setiap status sesuai aturan BFS dan DFS. Hitung berapa banyak status yang dibangkitkan sampai ditemukan *goal state.*



Initial state

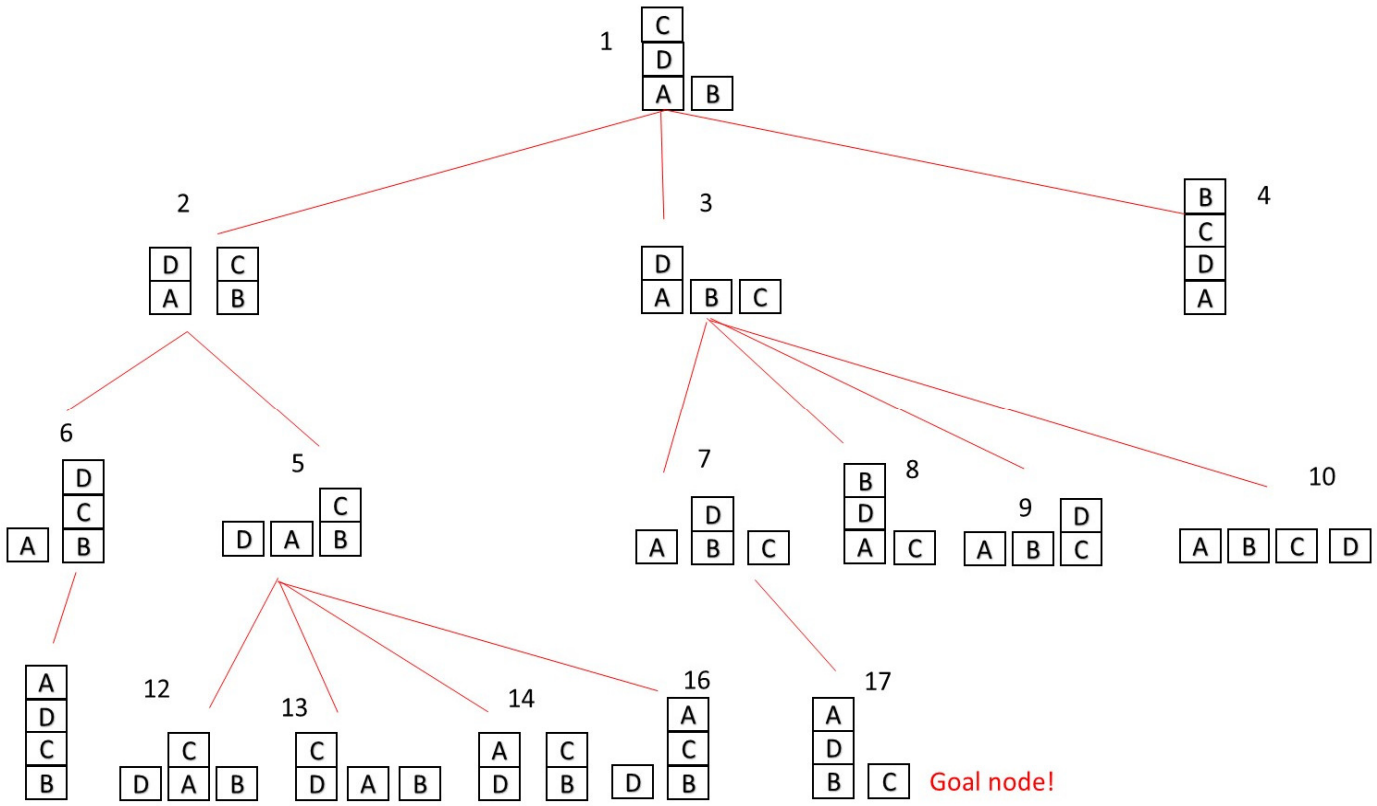


Goal state

Penyelesaian:

(a) BFS

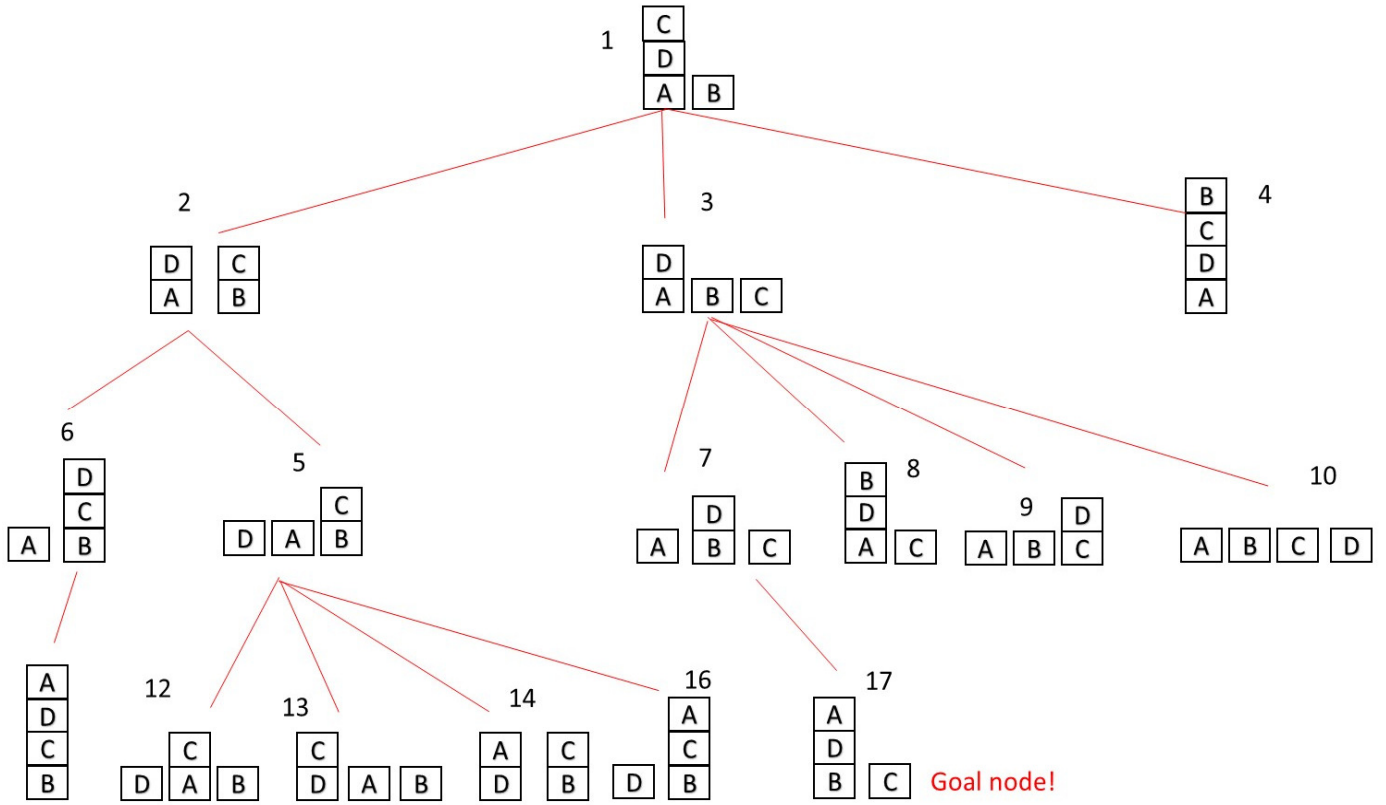
Salah satu kemungkinan solusi:



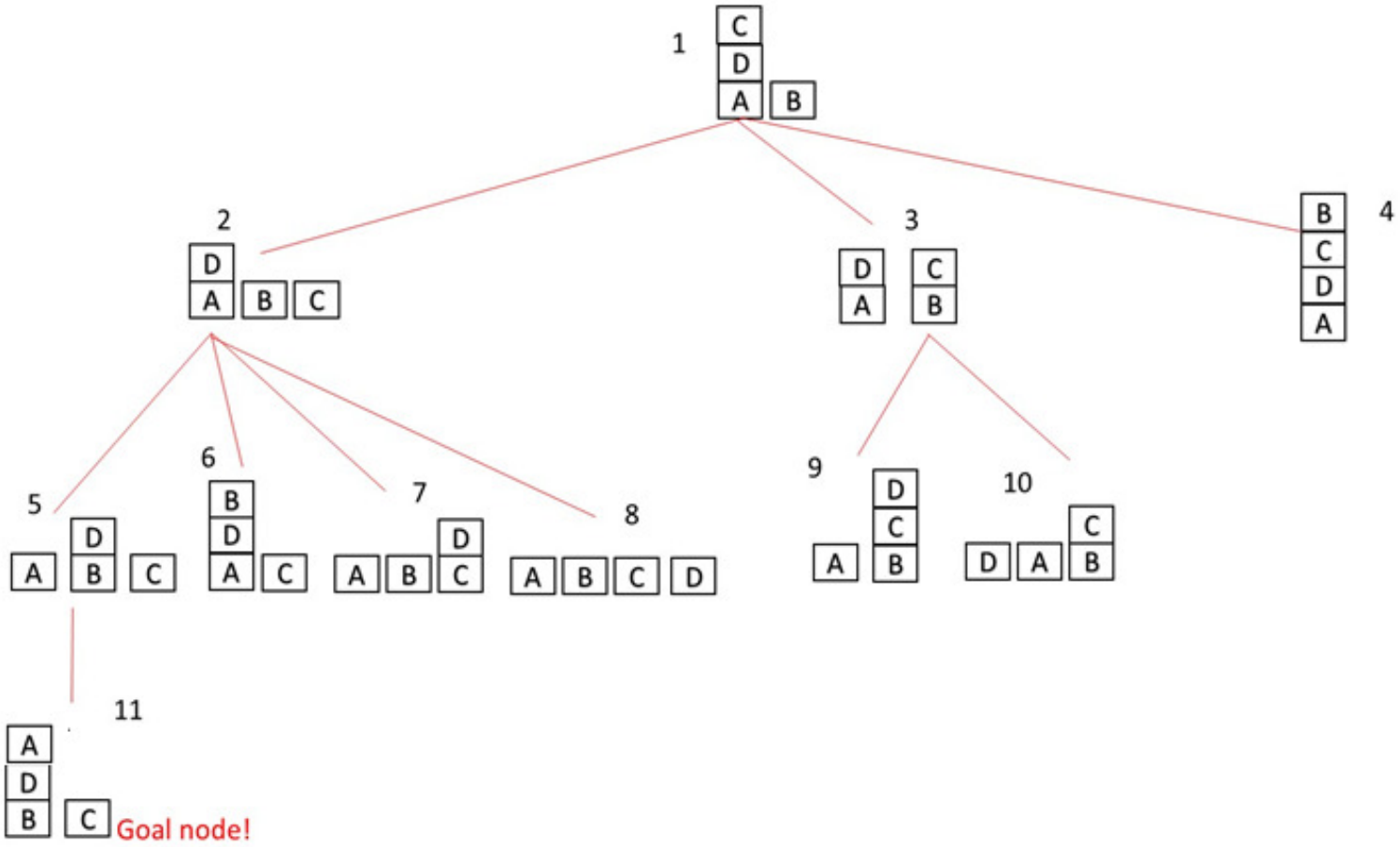
Penyelesaian:

(a) BFS

Salah satu kemungkinan solusi:



Kemungkinan lain:



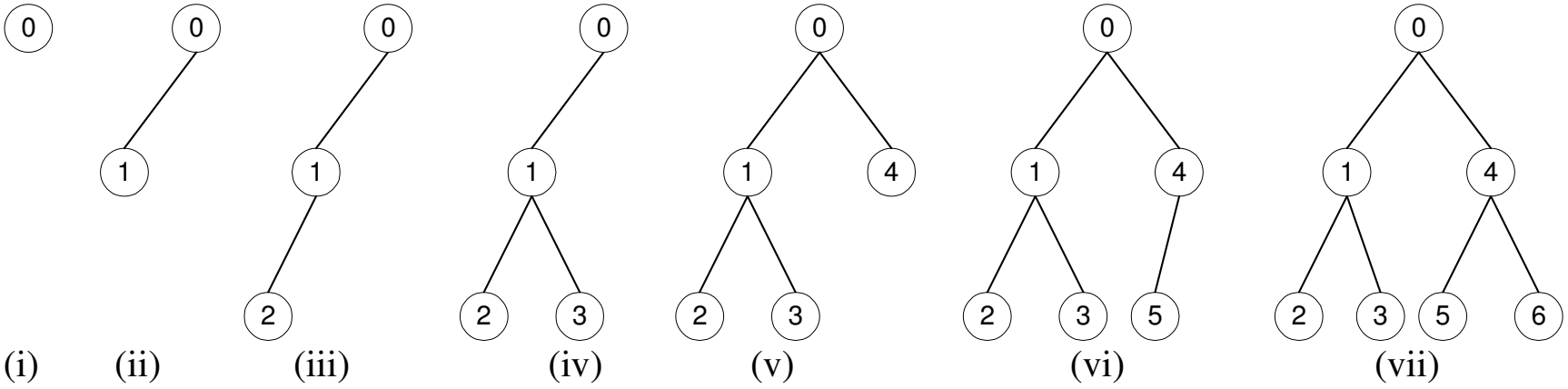
Metode Pencarian Mendalam (DFS)

- Traversal dimulai dari simpul *v*.
- Algoritma:
  1. Kunjungi simpul *v*,
  2. Kunjungi simpul *w* yang bertetangga dengan simpul *v*.
  3. Ulangi DFS mulai dari simpul w.

4. Ketika mencapai simpul *u* sedemikian sehingga semua simpul yang bertetangga dengannya telah dikunjungi, pencarian dirunut-balik (*backtrack*) ke simpul terakhir yang dikunjungi sebelumnya dan mempunyai simpul *w* yang belum dikunjungi.

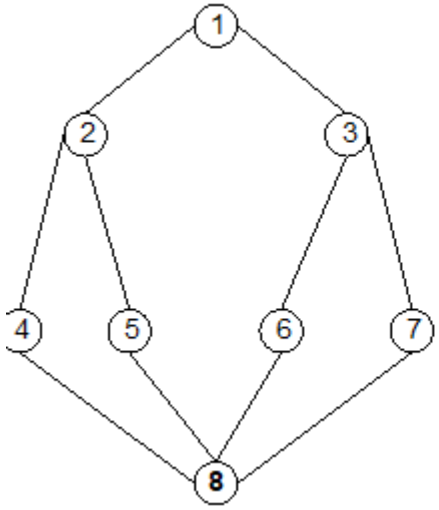
4. Pencarian berakhir bila tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi.

Metode Pencarian Mendalam (DFS)

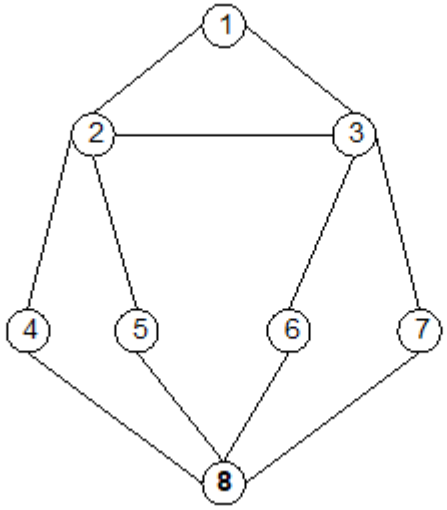


Gambar 6.9. Tahapan pembentukan pohon DFS

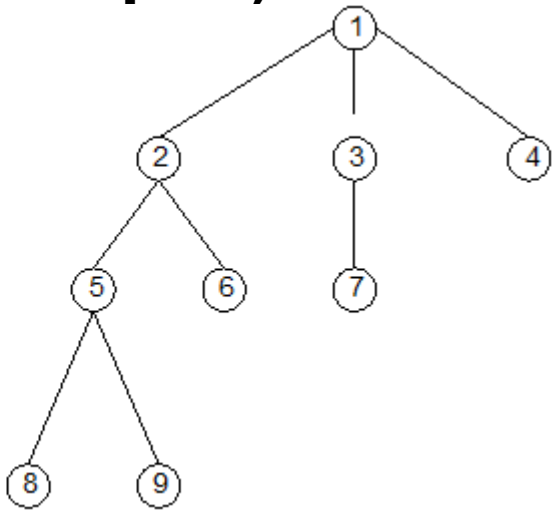
Contoh 2: (misalkan traversal dimulai dari simpul 1)



(a)



(b)



(c)

Gambar (a) DFS(1): 1, 2, 4, 8, 5, 6, 3, 7

Gambar (b) DFS(1): 1, 2, 3, 6, 8, 4, 5, 7

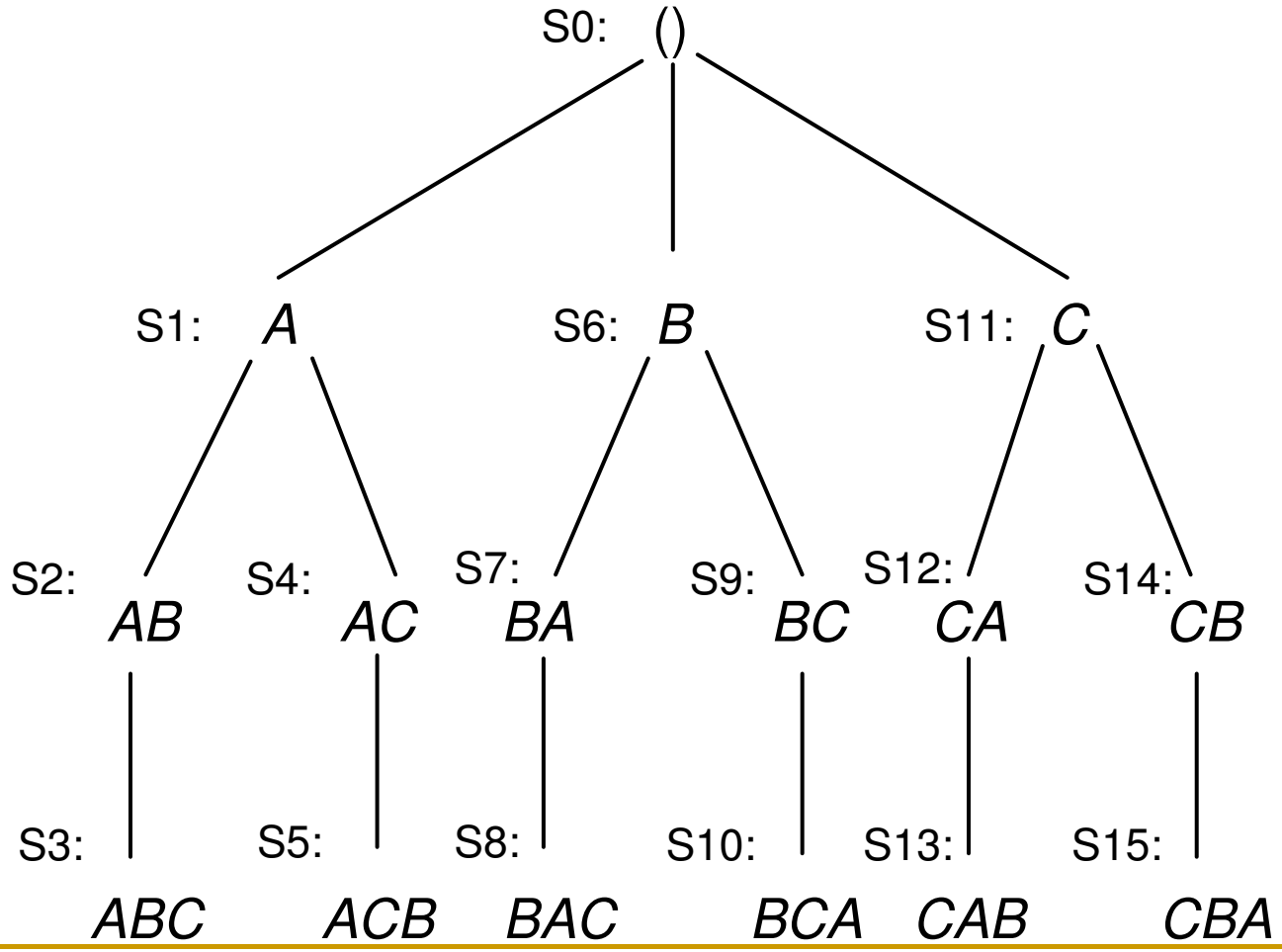
Gambar (c) DFS(1): 1, 2, 5, 8, 9, 6, 3, 7, 4

```
procedure DFS(input v:integer)
{Mengunjungi seluruh simpul graf dengan algoritma pencarian DFS

Masukan: v adalah simpul awal kunjungan
Keluaran: semua simpulyang dikunjungi ditulis ke layar
}
Deklarasi
   w : integer
Algoritma:
   write(v)
   dikunjungi[v]←true
   for tiap simpul w yang bertetangga dengan simpul v do
      if not dikunjungi[w] then
          DFS(w)
      endif
   endfor
```

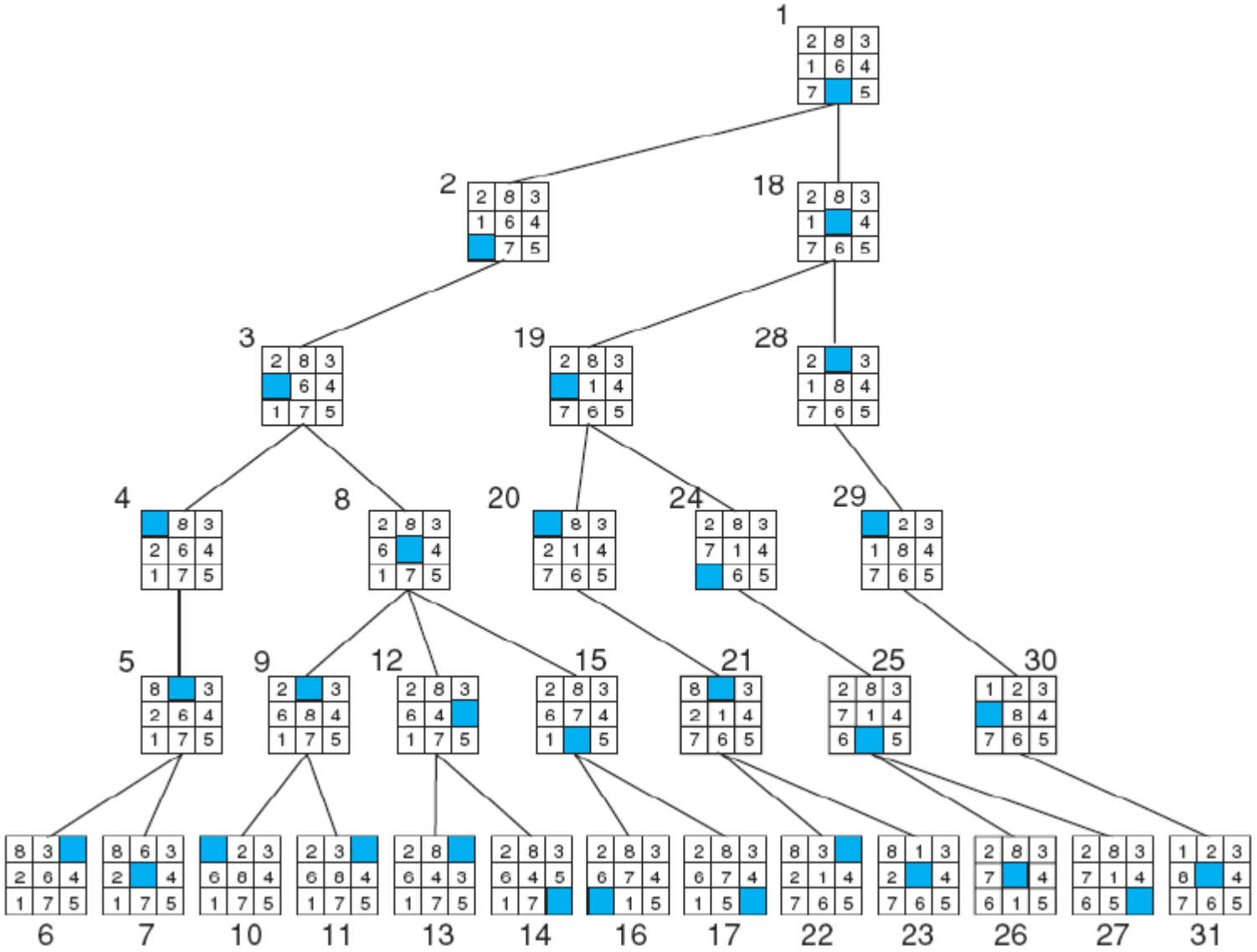
Algoritma DFS selengkapnya adalah:

```
procedure DFS(input v:integer)
{Mengunjungi seluruh simpul graf dengan algoritma pencarian DFS

Masukan: v adalah simpul awal kunjungan
Keluaran: semua simpulyang dikunjungi ditulis ke layar
}
Deklarasi
   w : integer

Algoritma:
   write(v)
   dikunjungi[v]←true
   for w←1 to n do
      if A[v,w]=1 then {simpul v dan simpul w bertetangga }
          if not dikunjungi[w] then
             DFS(w)
          endif
      endif
   endfor
```

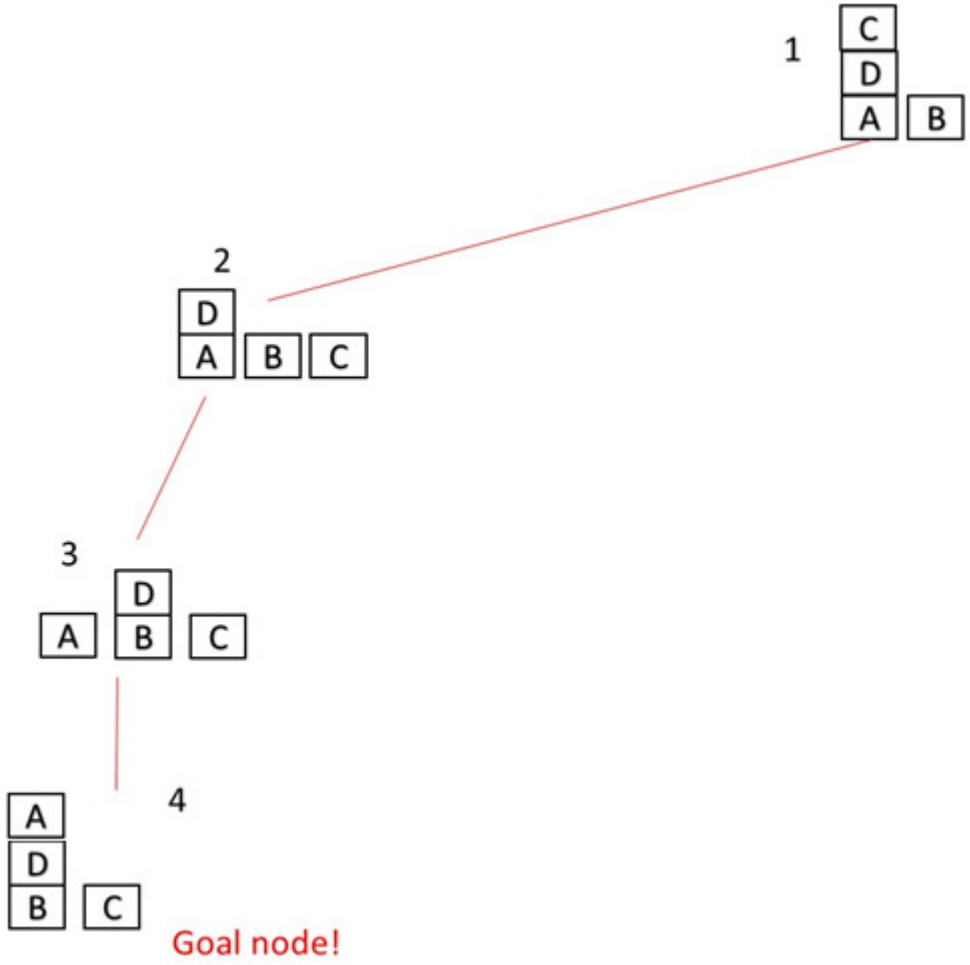
Pembentukan pohon ruang status persoalan pembangkitan permutasi *A*, *B*, *C* dengan metode DFS



DFS untuk 8-Puzzle



(b) DFS



Aplikasi DFS dan BFS

1. Search Engine (*google*, *yahoo*, *altavista*)

Komponen *search engine*:

1. *Spider* : program penjelajah *web* (*web surfer*)
2. *Index*: basisdata yang menyimpan kata-kata penting pada setiap halaman *web*
3. *Query*: pencarian berdasarkan string yang dimasukkan oleh pengguna (*end- user*)

Secara periodik (setiap jam atau setiap hari), *spider* menjejalahi *internet* untuk mengunjungi halaman-halaman *web*, mengambil kata-kata penting di dalam *web*, dan menyimpannya di dalam *index*.

Query dilakukan terhadap *index*, bukan terhadap *website* yang aktual.

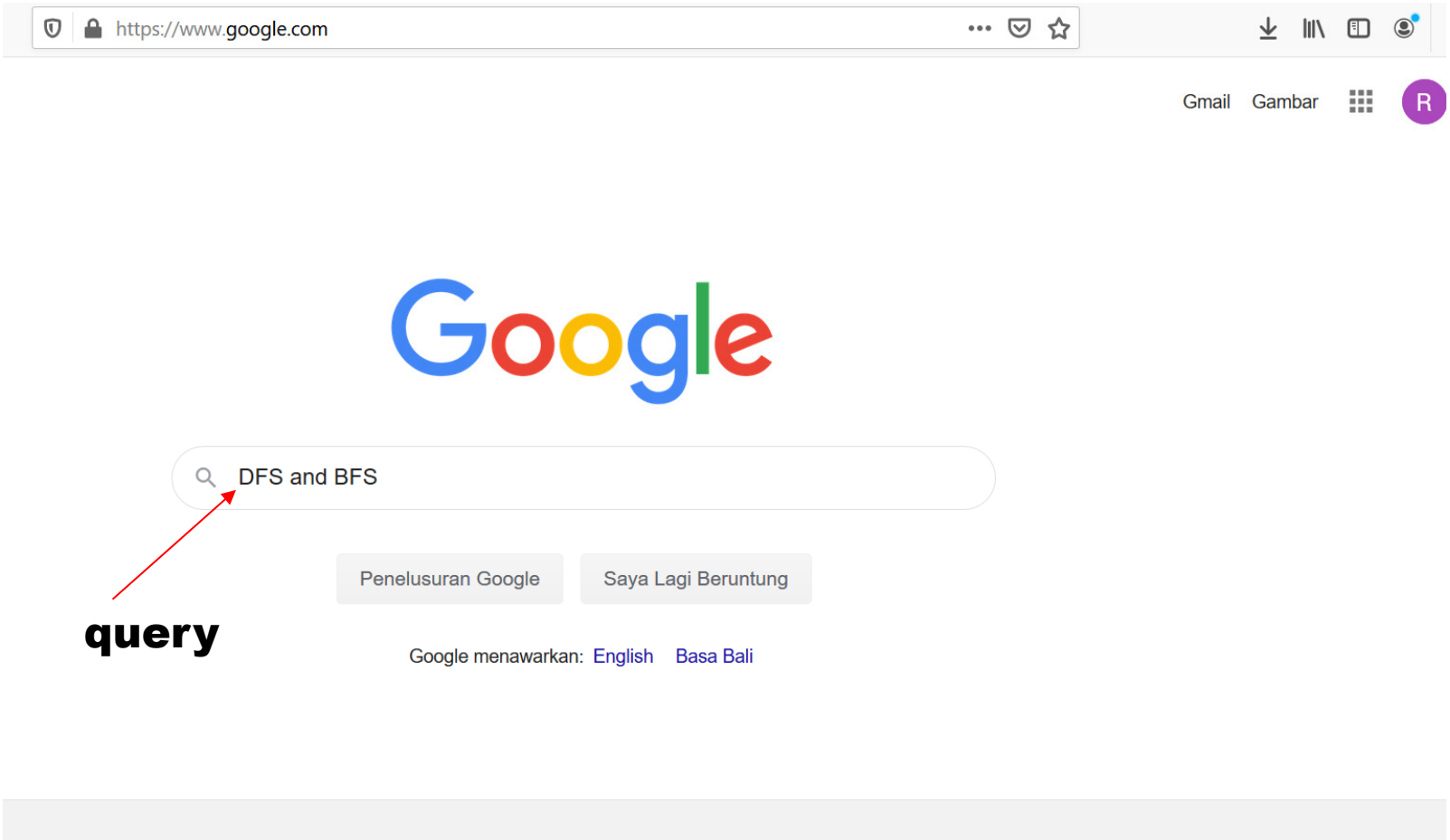
Bagaimana *spider* menjelajahi (*surfing*) *web?*

Simpul menyatakan halaman (*page*)

Sisi menyatakan *link* ke halaman (*page*)

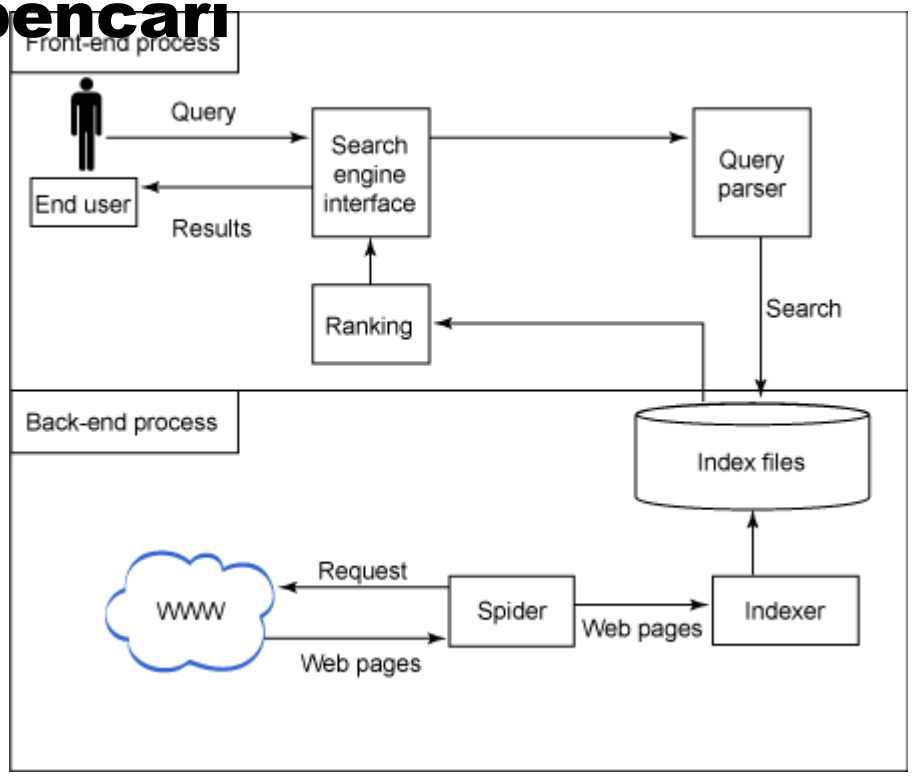
Bagaimana teknik menjelajahi *web*? Secara DFS atau BFS?

Penerapan BFS dan DFS: Web Spider



Penerapan BFS dan DFS: Web Spider

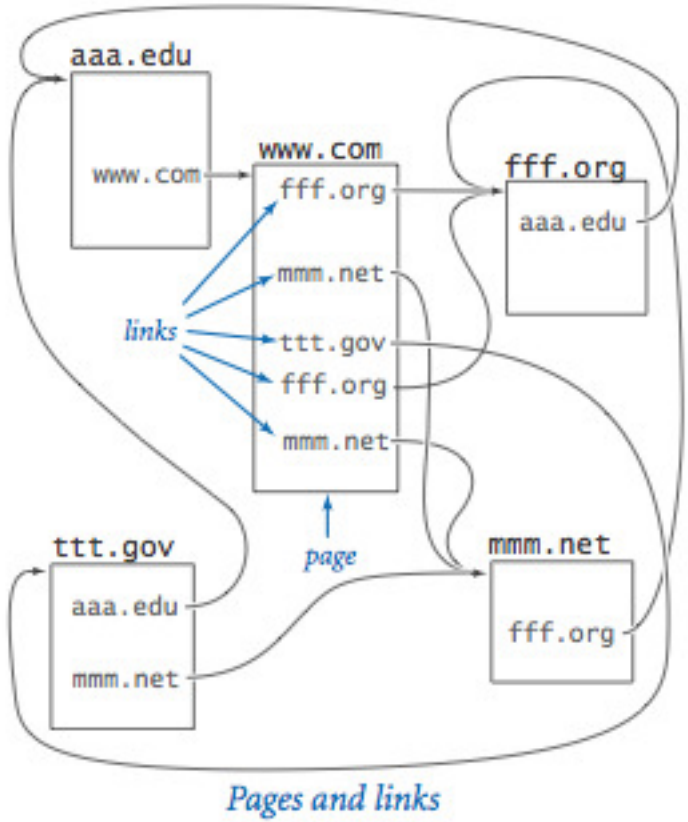
Arsitektur umum mesin pencari



http://www.ibm.com/developerworks/web/library/wa-lucene2/

- Secara periodik, web spider menjejalahi internet untuk mengunjungi halaman-halaman web

Web Spider: Penjelajahan Web

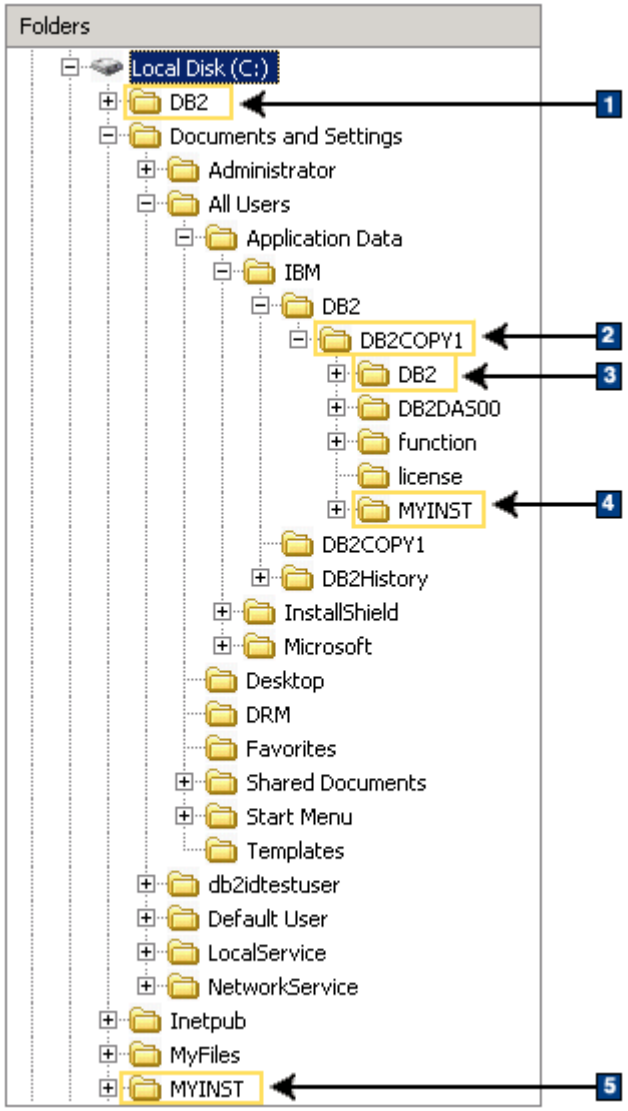


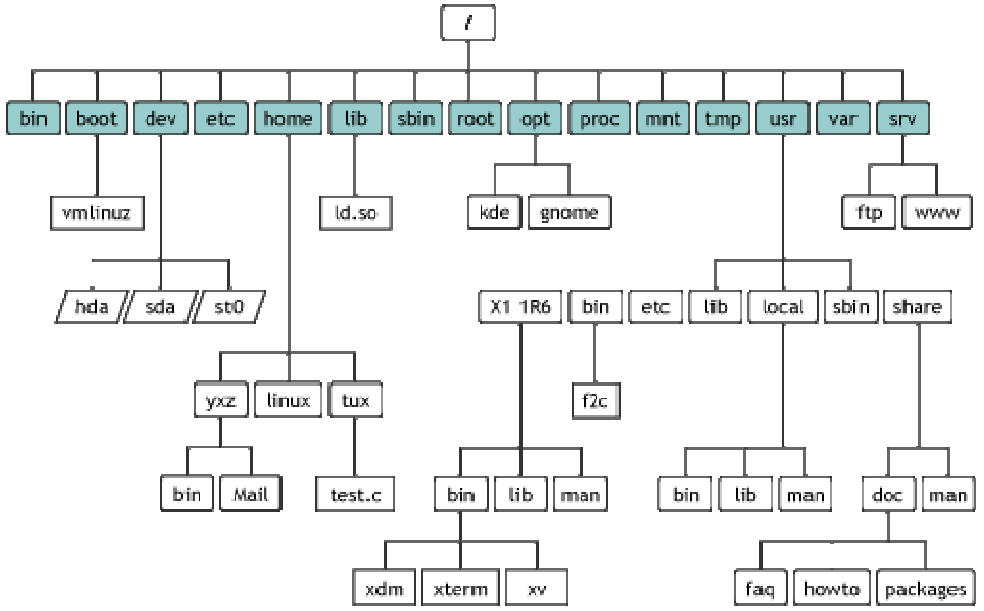
Pages and links

- Halaman web dimodelkan sebagai graf berarah
  - Simpul menyatakan halaman web (web page)
  - Sisi menyatakan link ke halaman web
- Bagaimana teknik menjelajahi web? Secara DFS atau BFS
- Dimulai dari web page awal, lalu setiap link ditelusuri secara DFS sampai setiap web page tidak mengandung link.

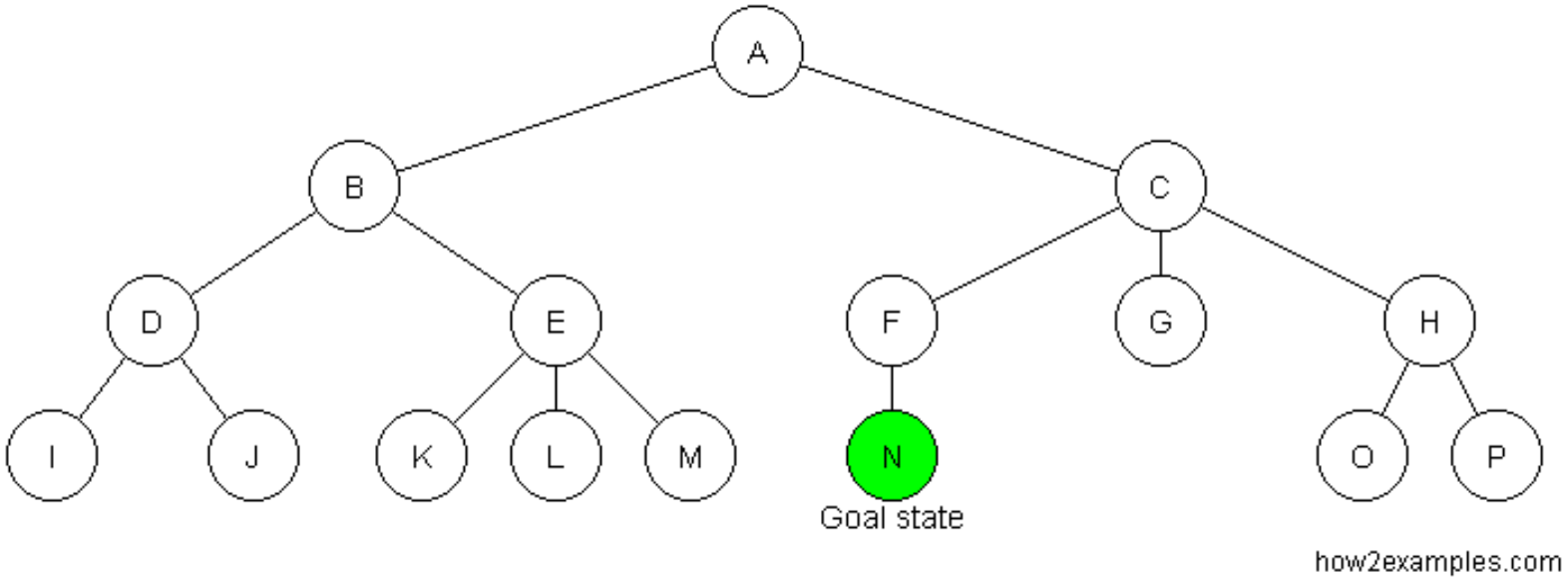
http://introcs.cs.princeton.edu/java/16pagerank/

DFS dan BFS untuk penelusuran direktori (folder)





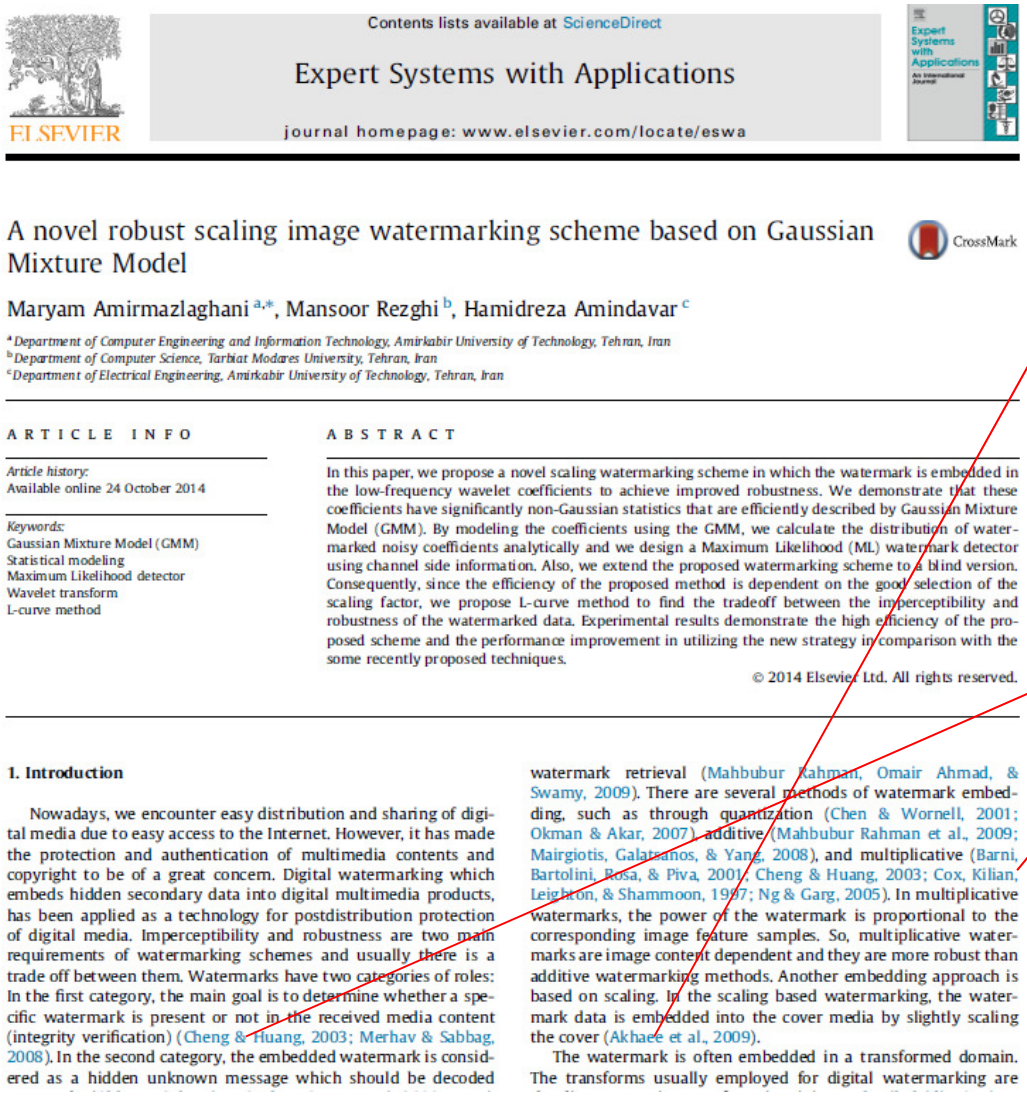
Pencarian dokumen di dalam direktori (folder) secara BFS



2. Referensi pada Makalah/Jurnal

- Pada setiap makalah, ada acuan ke literatur yang digunakan.
- Pada literatur tsb, ada acuan ke makalah/literatur yang lain?
- Bagaimana menelusuri acuan-acuan pada makalah? Secara DFS atau BFS?

Penerapan BFS dan DFS: Citation Map



Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

A novel robust scaling image watermarking scheme based on Gaussian Mixture Model

Maryam Amirmazlaghani [a,*], Mansoor Rezghi [b], Hamidreza Amindavar [c]

[a] Department of Computer Engineering and Information Technology, Amirkabir University of Technology, Tehran, Iran
[b] Department of Computer Science, Tarbiat Modares University, Tehran, Iran
[c] Department of Electrical Engineering, Amirkabir University of Technology, Tehran, Iran

ARTICLE INFO

Article history:
Available online 24 October 2014

Keywords:
Gaussian Mixture Model (GMM)
Statistical modeling
Maximum Likelihood detector
Wavelet transform
L-curve method

ABSTRACT

In this paper, we propose a novel scaling watermarking scheme in which the watermark is embedded in the low-frequency wavelet coefficients to achieve improved robustness. We demonstrate that these coefficients have significantly non-Gaussian statistics that are efficiently described by Gaussian Mixture Model (GMM). By modeling the coefficients using the GMM, we calculate the distribution of watermarked noisy coefficients analytically and we design a Maximum Likelihood (ML) watermark detector using channel side information. Also, we extend the proposed watermarking scheme to a blind version. Consequently, since the efficiency of the proposed method is dependent on the good selection of the scaling factor, we propose L-curve method to find the tradeoff between the imperceptibility and robustness of the watermarked data. Experimental results demonstrate the high efficiency of the proposed scheme and the performance improvement in utilizing the new strategy in comparison with the some recently proposed techniques.

1. Introduction

Nowadays, we encounter easy distribution and sharing of digital media due to easy access to the Internet. However, it has made the protection and authentication of multimedia contents and copyright to be of a great concern. Digital watermarking which embeds hidden secondary data into digital multimedia products, has been applied as a technology for postdistribution protection of digital media. Imperceptibility and robustness are two main requirements of watermarking schemes and usually there is a trade off between them. Watermarks have two categories of roles: In the first category, the main goal is to determine whether a specific watermark is present or not in the received media content (integrity verification) (Cheng & Huang, 2003; Merhav & Sabbag, 2008). In the second category, the embedded watermark is considered as a hidden unknown message which should be decoded watermark retrieval (Mahbubur Rahman, Omair Ahmad, & Swamy, 2009). There are several methods of watermark embedding, such as through quantization (Chen & Wornell, 2001; Okman & Akar, 2007), additive (Mahbubur Rahman et al., 2009; Mairgiotis, Galatsanos, & Yang, 2008), and multiplicative (Barni, Bartolini, Rosa, & Piva, 2001; Cheng & Huang, 2003; Cox, Kilian, Leighton, & Shammoon, 1997; Ng & Garg, 2005). In multiplicative watermarks, the power of the watermark is proportional to the corresponding image feature samples. So, multiplicative watermarks are image content dependent and they are more robust than additive watermarking methods. Another embedding approach is based on scaling. In the scaling based watermarking, the watermark data is embedded into the cover media by slightly scaling the cover (Akhaee et al., 2009).

The watermark is often embedded in a transformed domain. The transforms usually employed for digital watermarking are

References

Akhaee, M. A., Sahraeian, S. M. E., & Marvasti, F. (2010). Contourlet-based image watermarking using optimum detector in a noisy environment. *IEEE Transactions on Image Processing, 19*, 967–980.

Akhaee, M. A., Sahraeian, S. M. E., Sankur, B., & Marvasti, F. (2009). Robust scaling-based image watermarking using maximum-likelihood decoder with optimum strength factor. *IEEE Transactions on Multimedia, 11*, 822–833.

Allili, M. S. (2012). Wavelet modeling using finite mixtures of generalized Gaussian distributions: Application to texture discrimination and retrieval. *IEEE Transactions on Image Processing, 21*, 1452–1464.

Amirmazlaghani, M., Amindavar, H., & Moghaddamjoo, A. R. (2009). Speckle suppression in SAR images using the 2-D GARCH model. *IEEE Transactions on Image Processing, 18*, 250–259.

Barni, M., Bartolini, F., Rosa, A. D., & Piva, A. (2001). A new decoder for the optimum recovery of nonadditive watermarks. *IEEE Transactions on Image Processing, 10*, 755–766.

Barni, M., Bartolini, F., Rosa, A. D., & Piva, A. (2003). Optimum decoding and detection of multiplicative watermarks. *IEEE Transactions on Signal Processing, 51*, 1118–1123.

Cheng, Q., & Huang, T. S. (2003). Robust optimum detection of transform domain multiplicative watermarks. *IEEE Transactions on Signal Processing, 51*, 906–924.

Chen, B., & Wornell, G. W. (2001). Quantization index modulation: A class of provably good methods for digital watermarking and information embedding. *IEEE Transactions on Information Theory, 47*, 1423–1443.

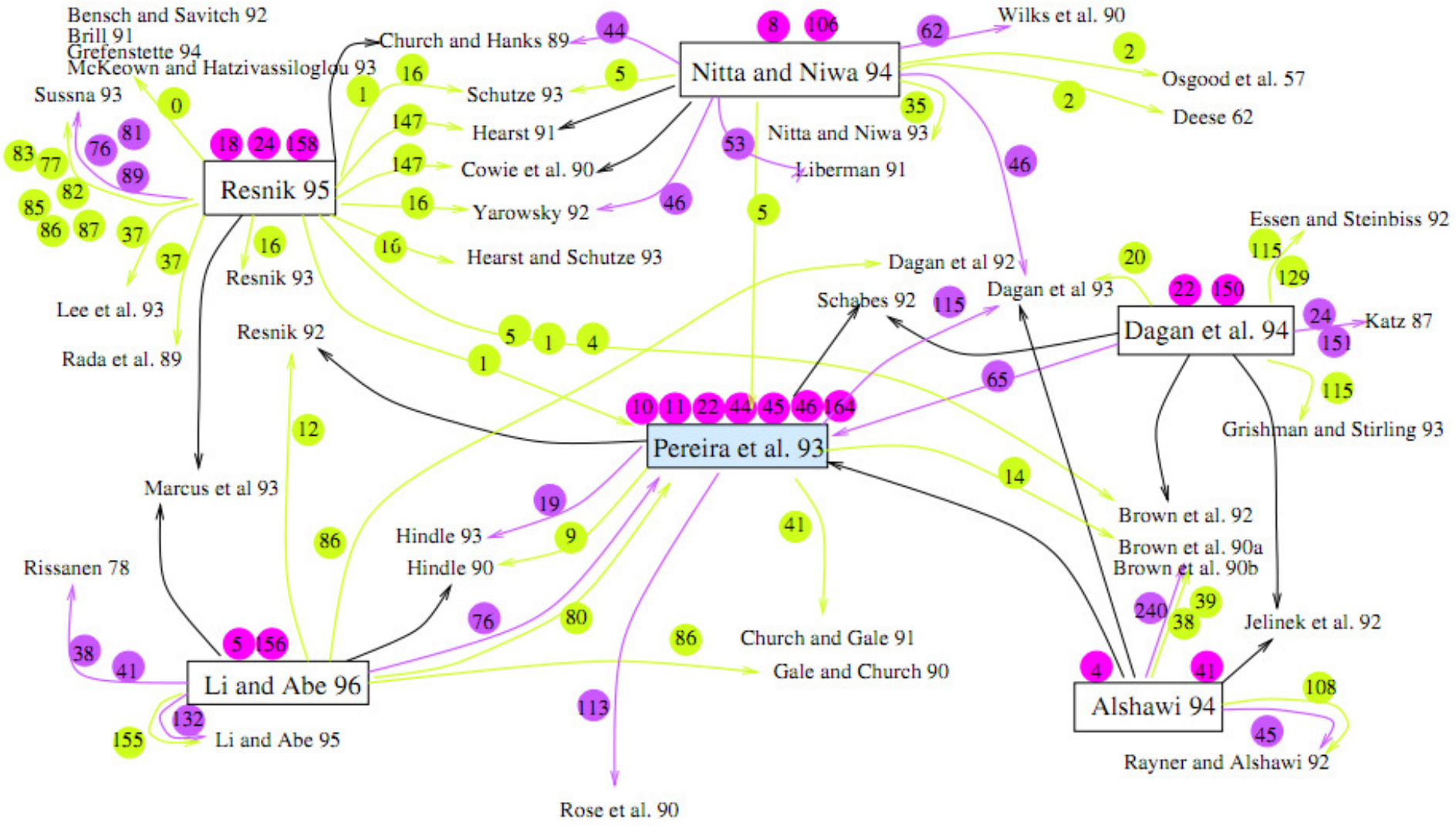
Cox, I., Kilian, J., Leighton, T., & Shammoon, T. (1997). Secure spread spectrum watermarking for multimedia. *IEEE Transactions on Image Processing, 6*, 1673–1687.

Doncel, V. R., Nikolaidis, N., & Pitas, I. (2007). An optimal detector structure for the Fourier descriptors domain watermarking of 2D vector graphics. *IEEE Transactions on Visualization and Computer Graphics, 13*, 851–863.

Donoho, D. L. (1994). Ideal spatial adoption by wavelet shrinkage. *Biometrika, 81*, 425–455.

Gembicki, F., & Haimes, Y. (1975). Approach to performance and sensitivity multiobjective optimization: The goal attainment method. *IEEE Transactions on Automatic Control, AC-20*, 769–771.

Citation Map:



Sumber: Teufel (1999), Argumentative Zoning

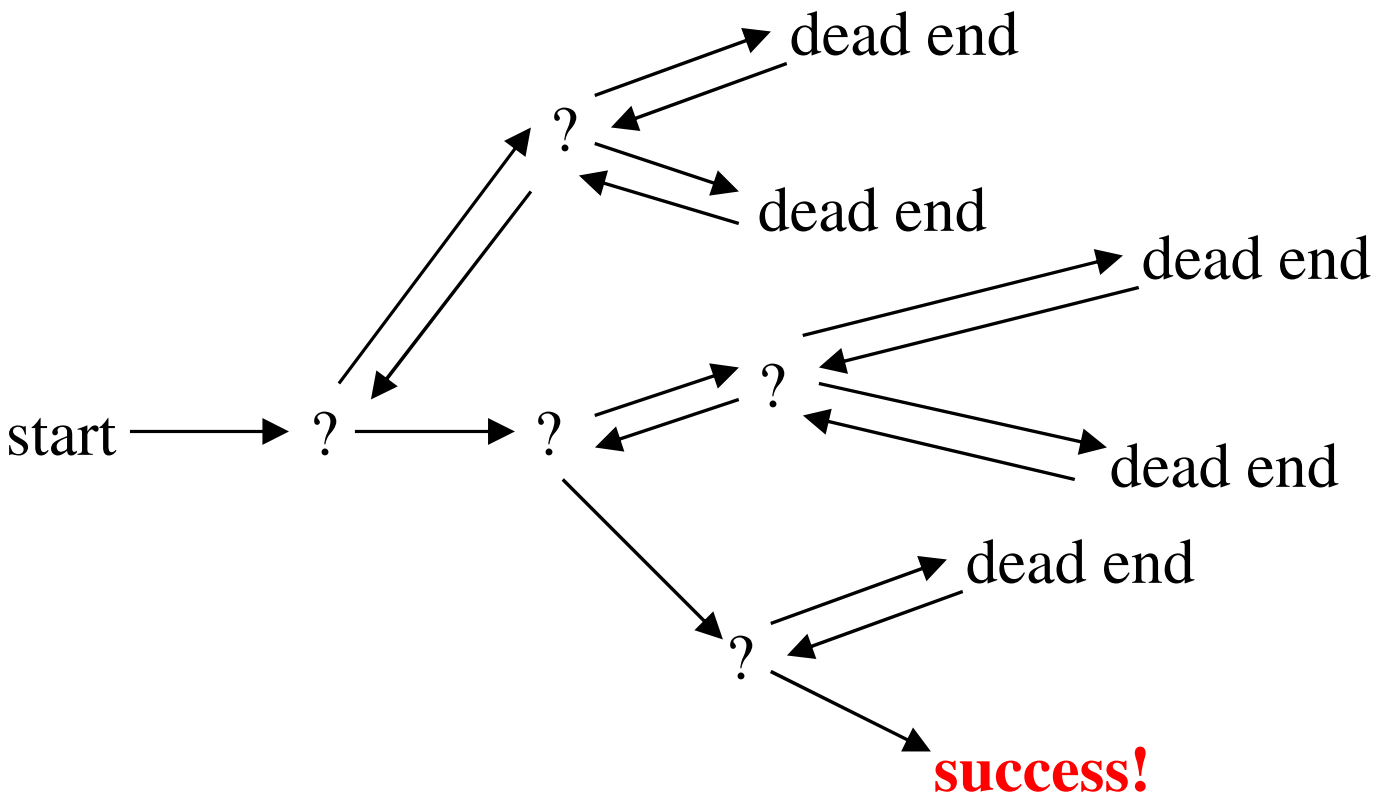
Aplikasi *Backtracking* di dalam DFS untuk Pemecahan Persoalan Pencarian Solusi

- Karakteristik *backtracking* di dalam algoritma DFS berguna untuk memecahkan persoalan pencarian solusi yang memiliki banyak alternatif pilihan selama pencarian.

- Solusi diperoleh dengan mengekspansi pencarian menuju *goal* dengan aturan "depth-first"
  - Anda tidak punya cukup informasi untuk mengetahui apa yang akan dipilih
  - Tiap keputusan mengarah pada sekumpulan pilihan baru
  - Beberapa sekuens pilihan (bisa lebih dari satu) mungkin merupakan solusi persoalan

- Backtracking di dalam algoritma DFS adalah cara yang metodologis mencoba beberapa sekuens keputusan,

- sampai Anda menemukan sekuens yang "bekerja"

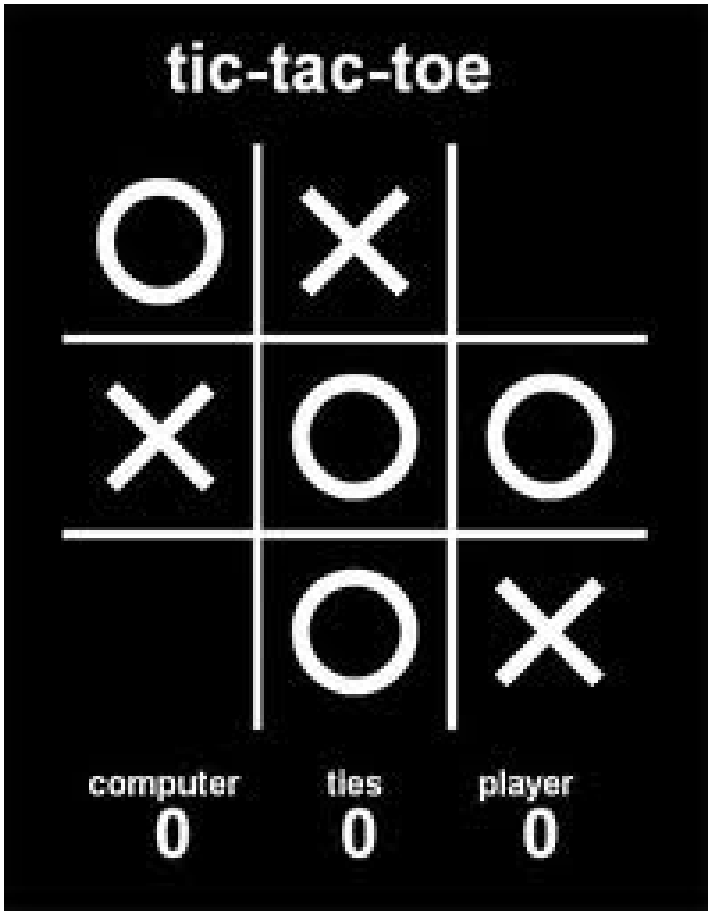


*) Sumber: *www.cis.upenn.edu/.../35-**backtracking**.ppt*

- Backtracking di dalam algoritma DFS banyak diterapan untuk mencari solusi persoalan games seperti:
  - permainan *tic-tac-toe*,
  - menemukan jalan keluar dalam sebuah labirin,
  - Catur, *crossword puzzle*, *sudoku*, dan masalah-masalah pada bidang kecerdasan buatan (*artificial intelligence*).

Crossword puzzle:

Tic-Tac-Toe



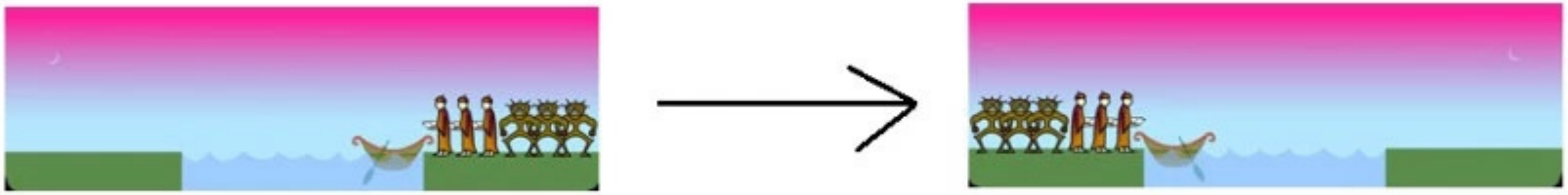
Sudoku

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Catur

Persoalan Missionaris dan Kanibal

- Terdapat 3 misionaris dan 3 kanibal yang harus menyebrang ke sisi sungai menggunakan sebuah perahu. Perahu hanya boleh berisi penumpang maksimal 2 orang. Jumlah kanibal tidak boleh lebih banyak dari jumlah misionaris di salah satu sisi, jika



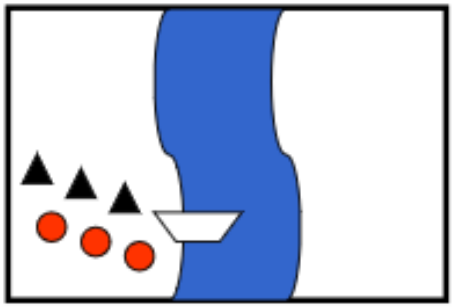
Initial State

Goal State

- sehingga semuanya selamat sampai di sisi sungai seberangnyya? Selesaikan dengan BFS dan DFS

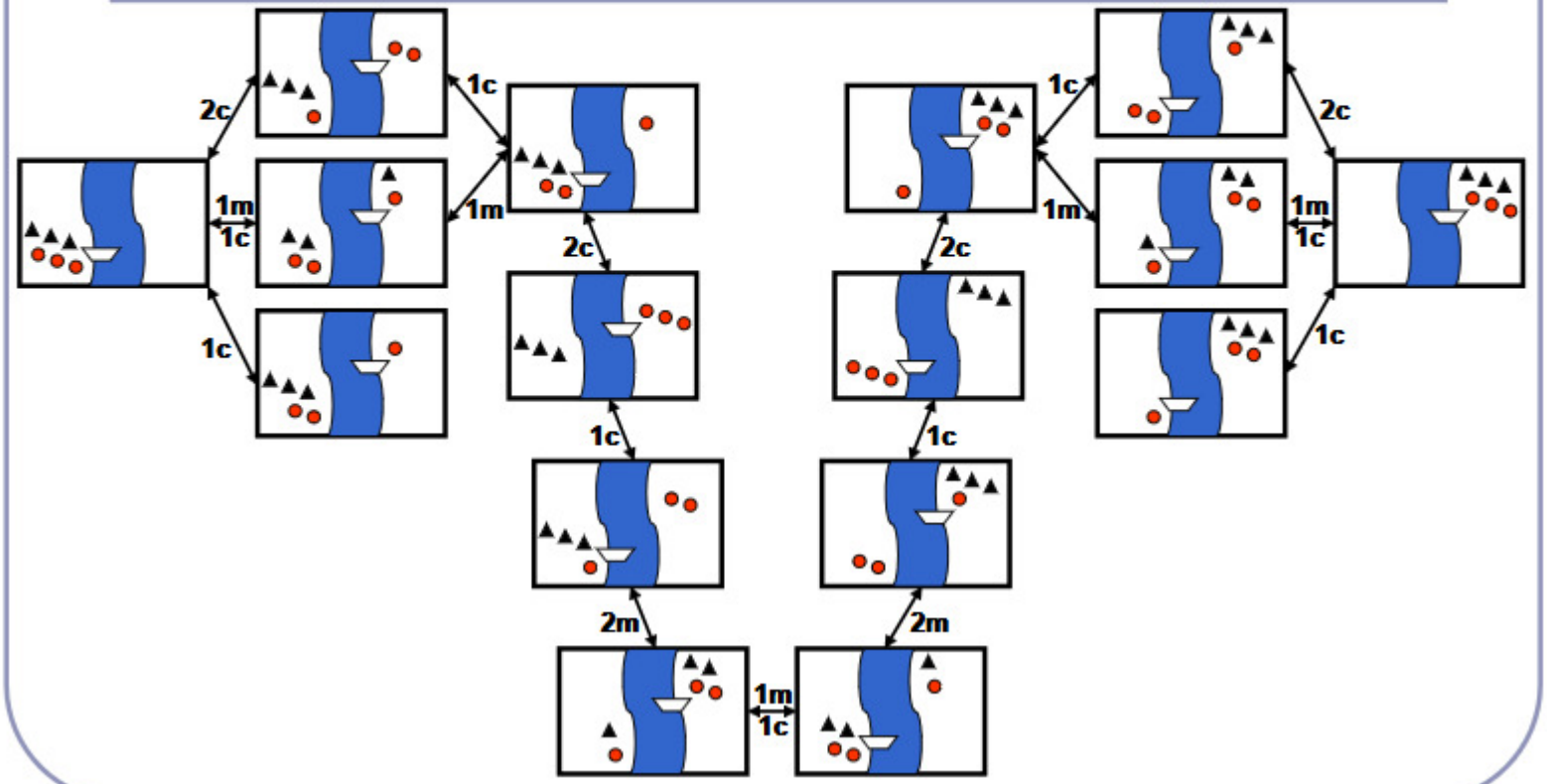
Missionaries and Cannibals: Initial State and Actions

- initial state:
  - all missionaries, all cannibals, and the boat are on the left bank

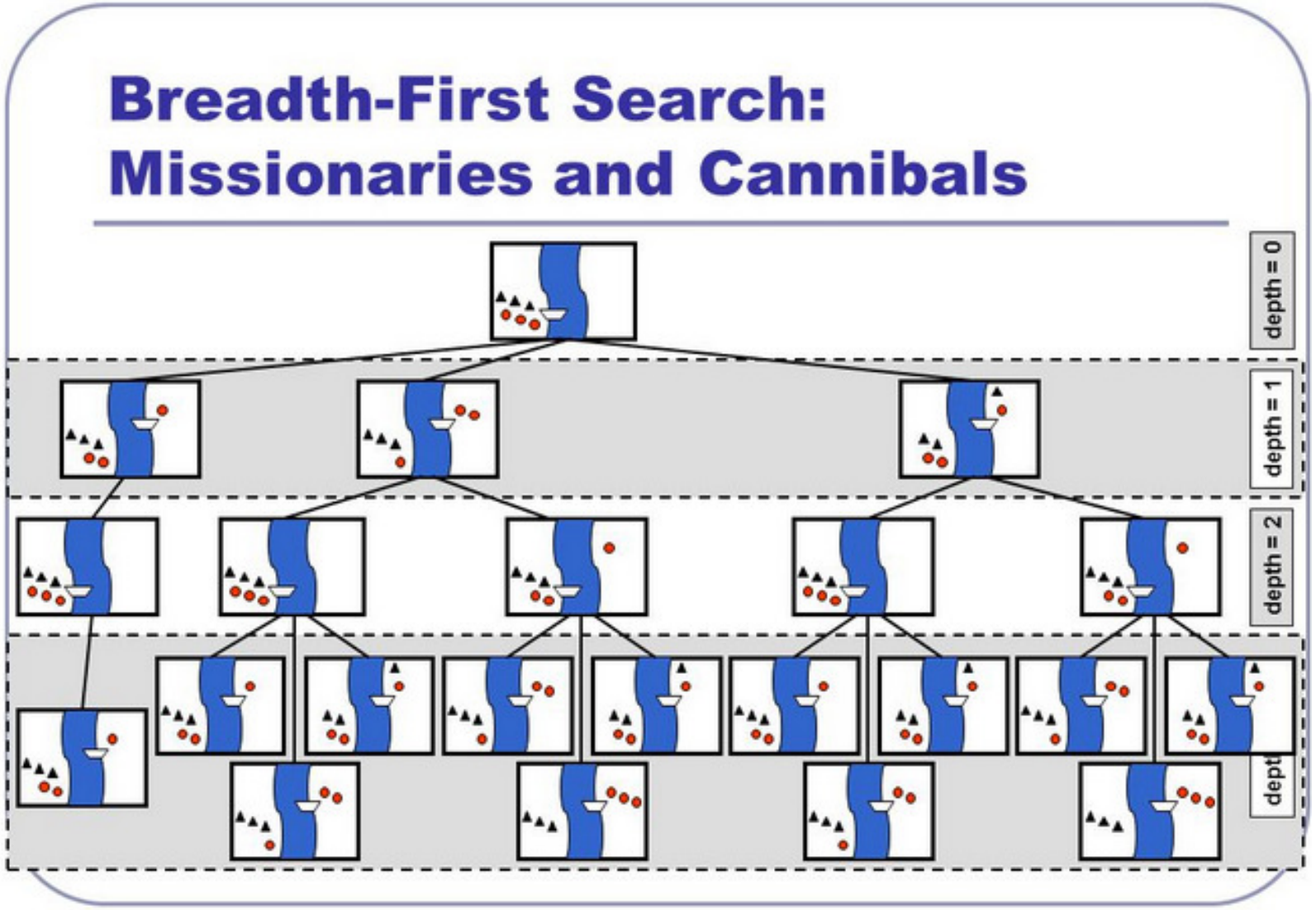


- 5 possible actions:
  - one missionary crossing
  - one cannibal crossing
  - two missionaries crossing
  - two cannibals crossing
  - one missionary and one cannibal crossing

Missionaries and Cannibals: State Space

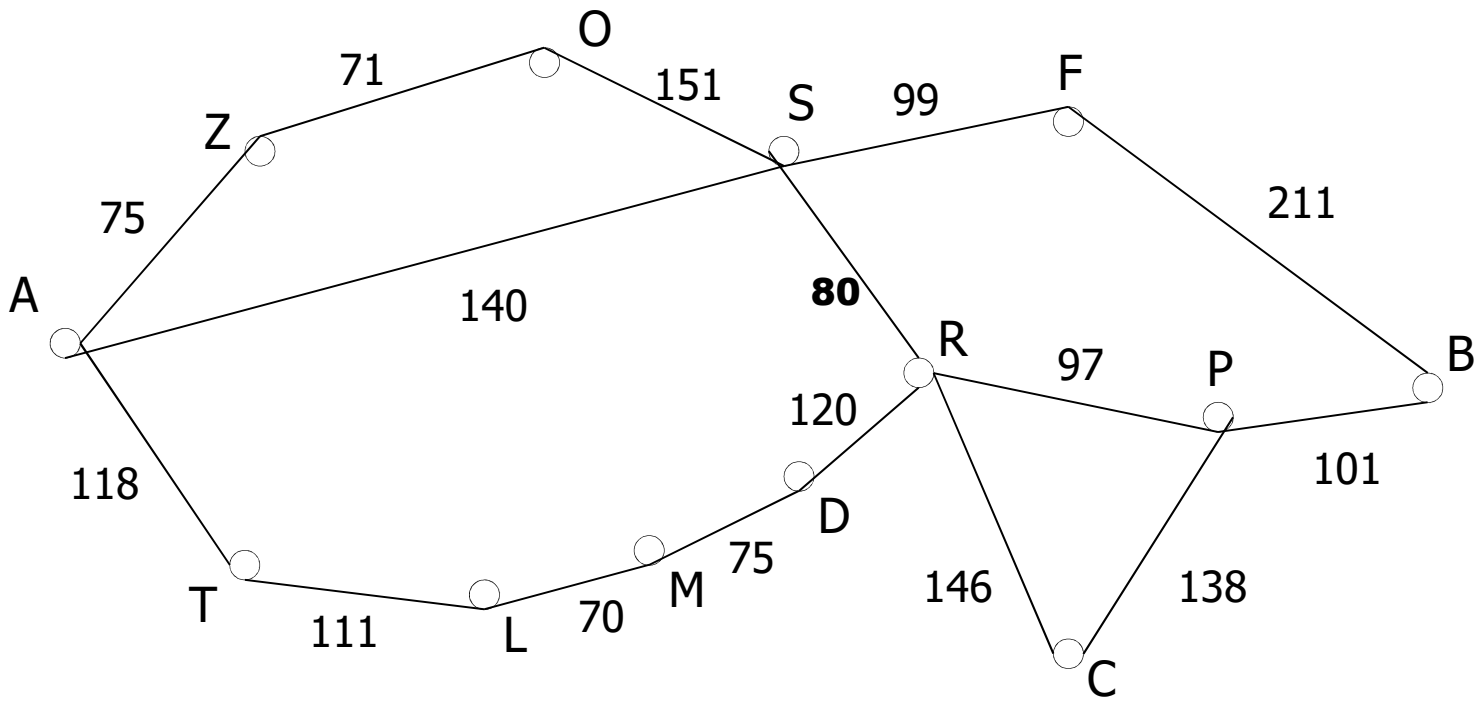


Sebagian pohon ruang status dengan BFS



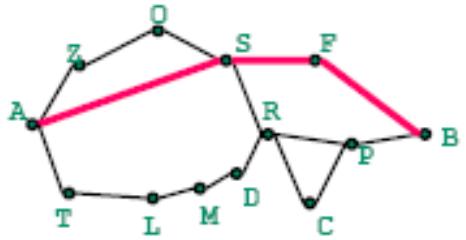
Search

Source: Russell's book

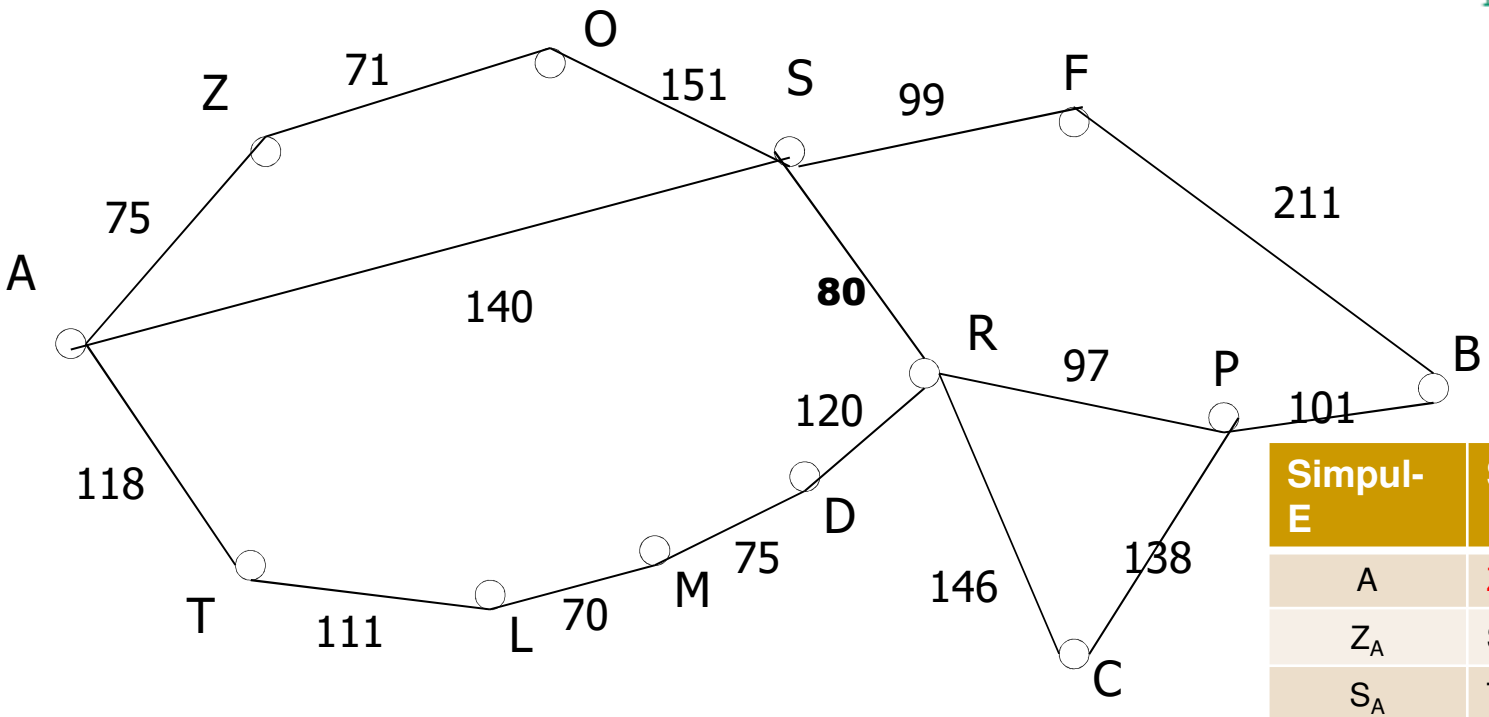


S: set of cities
i.s: A (Arad)
g.s: B (Bucharest)
Goal test: s = B ?
Path cost: time ~ distance

Breadth-First Search (BFS)



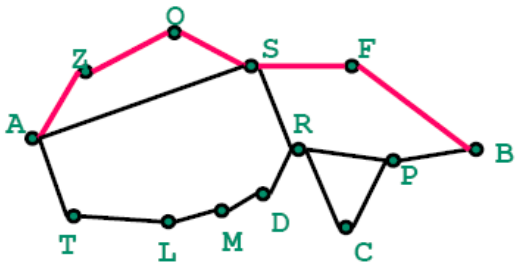
Treat agenda as a queue (FIFO)



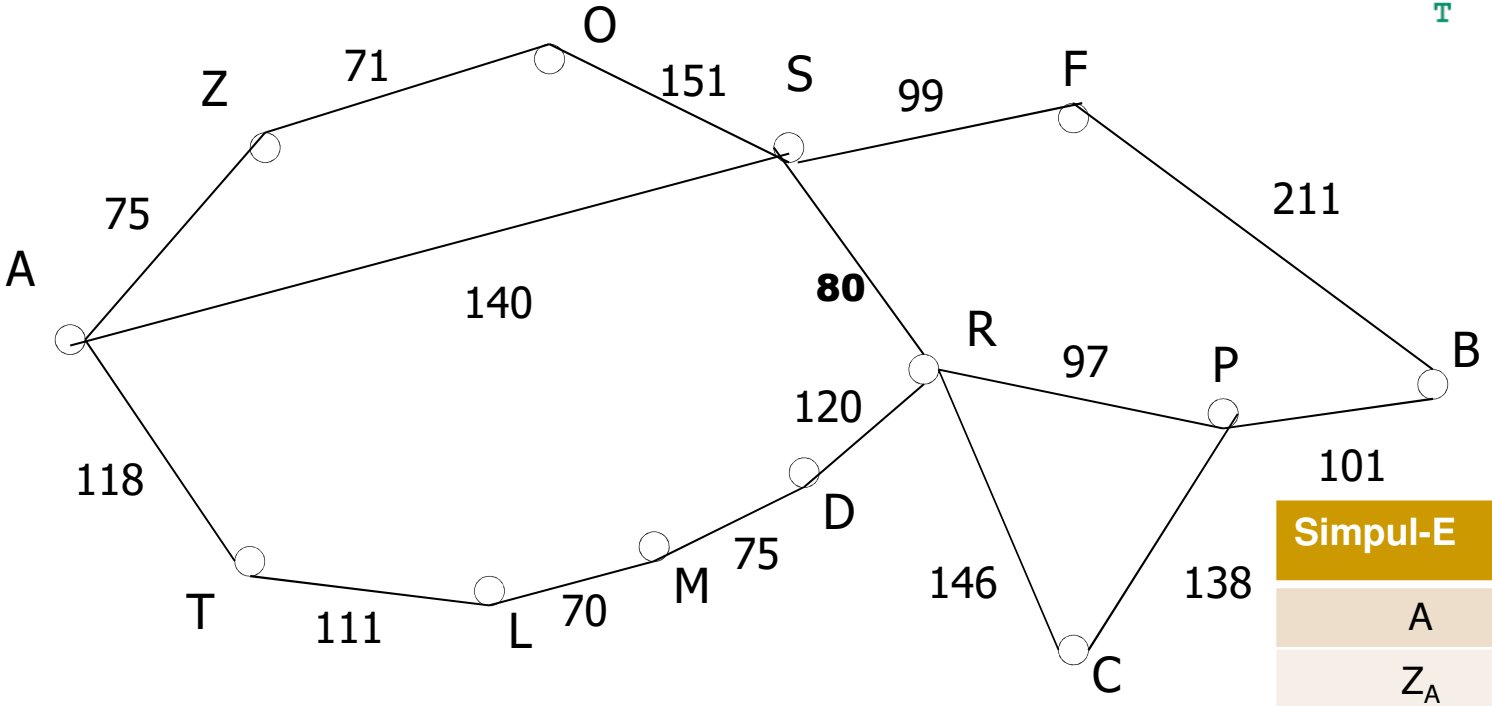
Path: A→ S → F → B,
Path-cost = 450

Simpul-E	Simpul Hidup
A	Z_A, S_A, T_A
Z_A	S_A, T_A, O_{AZ}
S_A	$T_A, O_{AZ}, O_{AS}, F_{AS}, R_{AS}$
T_A	$O_{AZ}, O_{AS}, F_{AS}, R_{AS}, L_{AT}$
O_{AZ}	$O_{AS}, F_{AS}, R_{AS}, L_{AT}$
O_{AS}	F_{AS}, R_{AS}, L_{AT}
F_{AS}	R_{AS}, L_{AT}, B_{ASF}
R_{AS}	$L_{AT}, B_{ASF}, D_{ASR}, C_{ASR}, P_{ASR}$
L_{AT}	$B_{ASF}, D_{ASR}, C_{ASR}, P_{ASR}, M_{ATL}$
B_{ASF}	Solusi ketemu

Depth-First Search (DFS)



Treat agenda as a stack (LIFO)



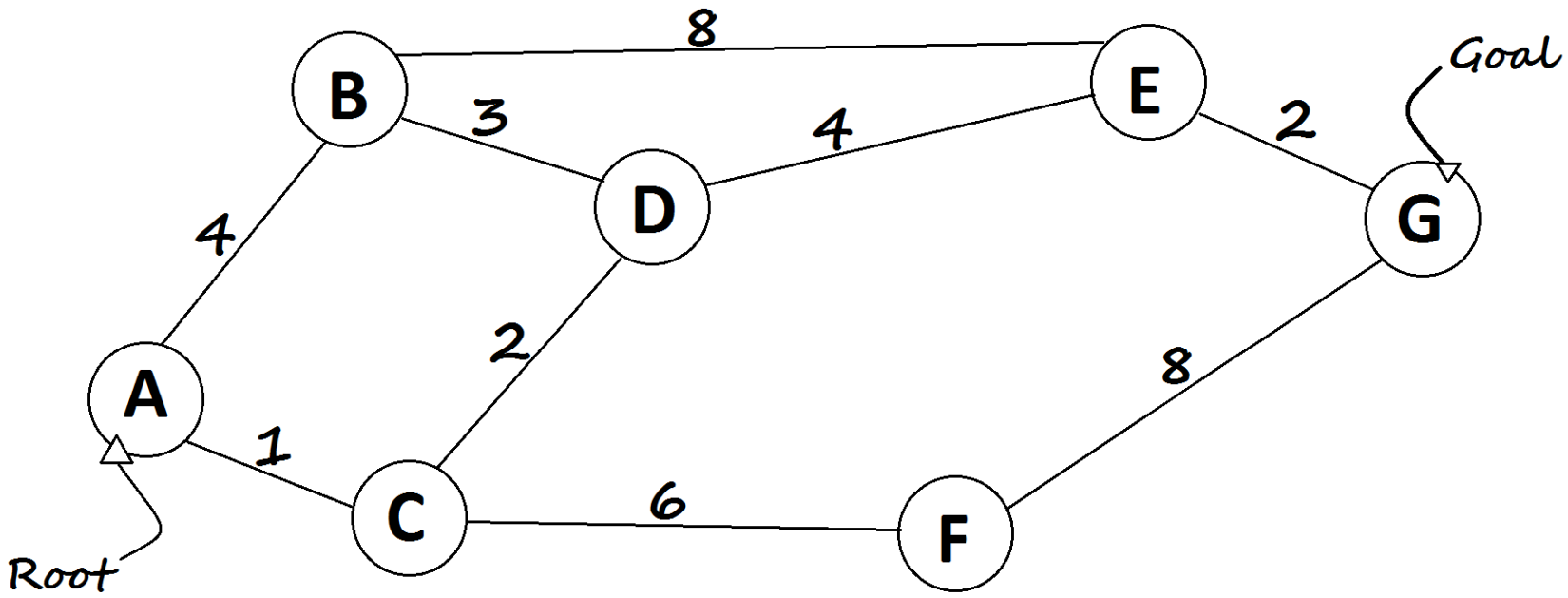
Path: A→ Z → O → S → F → B
Path-cost = 607

Simpul-E	Simpul Hidup
A	Z_A,S_A,T_A
Z_A	O_{AZ}, S_A,T_A
O_{AZ}	S_{AZO},S_A,T_A
S_{AZO}	F_{AZOS}, R_{AZOS},S_A,T_A
F_{AZOS}	B_{AZOSF}, R_{AZOS},S_A,T_A
B_{AZOSF}	Solusi ketemu

Contoh *path finding* lainnya:

a. BFS

b. DFS



Sumber:https: //medium.com/omarelgabrys-blog/path-finding-algorithms-f65a8902eb40

Referensi

- Materi kuliah IF3170 Inteligensi Buatan Teknik Informatika ITB, Course Website:
  http://kuliah.itb.ac.id → STEI → Teknik Informatika → IF3170
- Stuart J Russell & Peter Norvig, Artificial Intelligence: A Modern Approach, 3rd Edition, Prentice-Hall International, Inc, 2010, Textbook Site: http://aima.cs.berkeley.edu/ (2nd edition)
- Free online course materials | MIT OpenCourseWare Website:
  Site: http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/