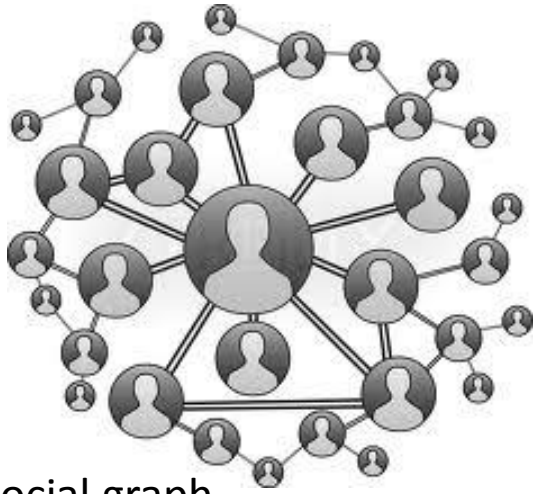
Breadth/Depth First Search (BFS/DFS)

Bahan Kuliah IF2211 Strategi Algoritmik
Oleh: Rinaldi Munir
Update: Nur Ulfa Maulidevi
2 Maret 2015

Traversal Graf

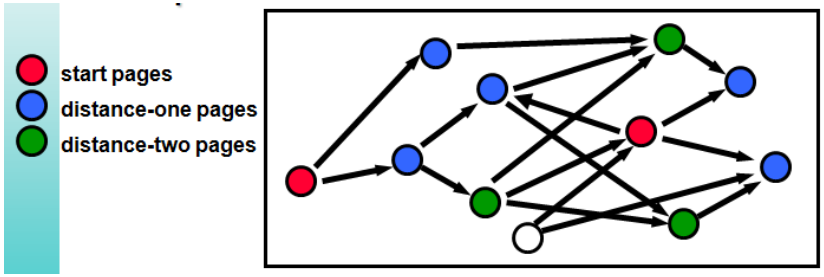
- Algoritma traversal graf: mengunjungi simpul dengan cara yang sistematik
  - Pencarian melebar (breadth first search/BFS)
  - Pencarian mendalam (depth first search/DFS)
  - Asumsi: graf terhubung
- Graf: representasi persoalan ➔ Traversal graf: pencarian solusi



social graph

http://www.oreilly.de/catalog/9780596518172/toc.html

Web page network



Algoritma Pencarian

- Tanpa informasi (*uninformed/blind search*)
  - Tidak ada informasi tambahan
  - Contoh: **DFS, BFS**, *Depth Limited Search*, *Iterative Deepening Search*, *Uniform Cost Search*
- Dengan informasi (*informed Search*)
  - Pencarian berbasis heuristik
  - Mengetahui non-goal state "lebih menjanjikan" daripada yang lain
  - Contoh: Best First Search, A*

Representasi Graf dalam Proses Pencarian

Dalam proses pencarian solusi, terdapat dua pendekatan:

- Graf statis: graf yang sudah terbentuk sebelum proses pencarian dilakukan

  - graf direpresentasikan sebagai struktur data

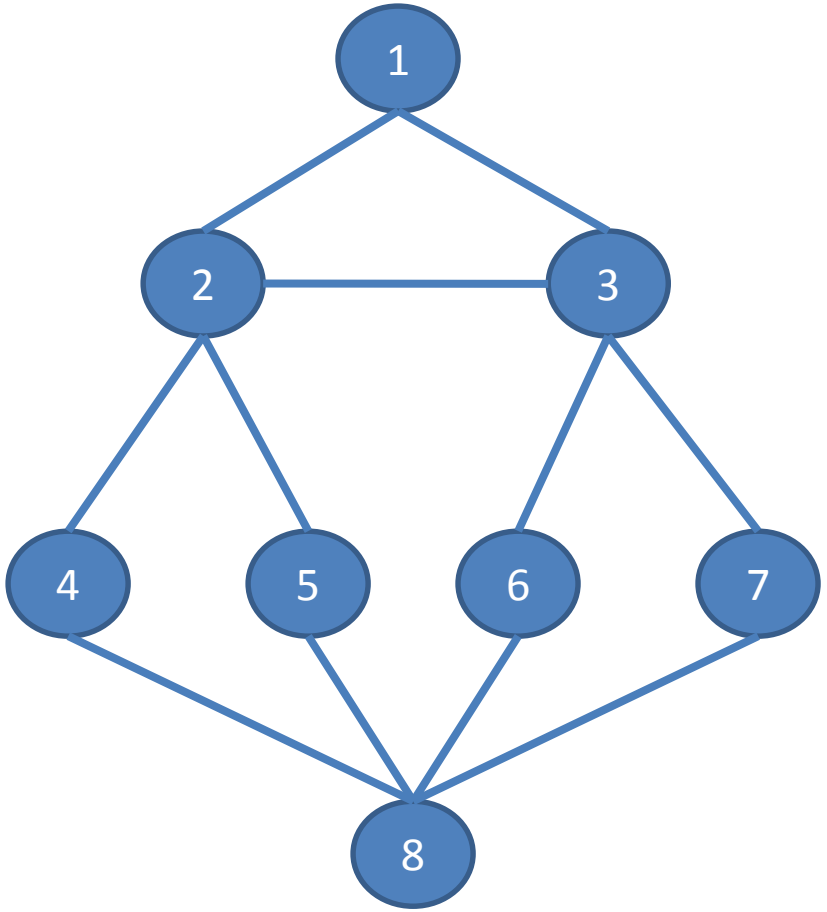
- Graf dinamis: graf yang terbentuk saat proses pencarian dilakukan

  - graf tidak tersedia sebelum pencarian, graf dibangun selama pencarian solusi

GRAF STATIS

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.



BFS: Struktur Data

1. Matriks ketetanggaan $A = [a_{ij}]$ yang berukuran nxn,
   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, diberi nama "dikunjungi"

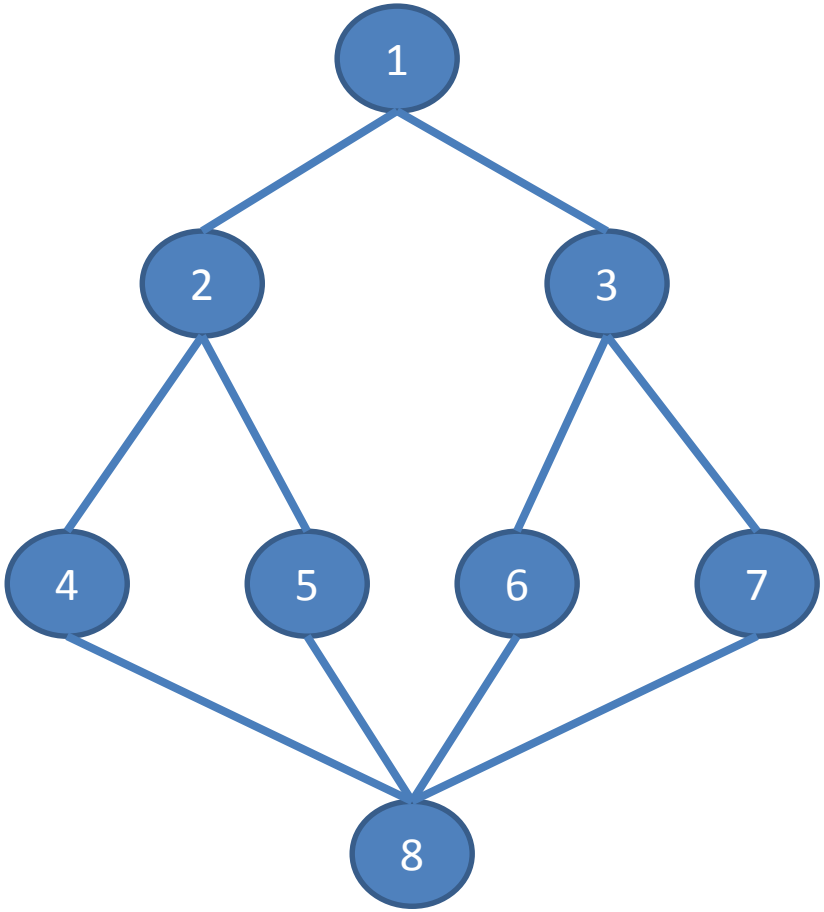
   ```
   dikunjungi : array[1..n] of boolean
   ```

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

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

```
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) }
```

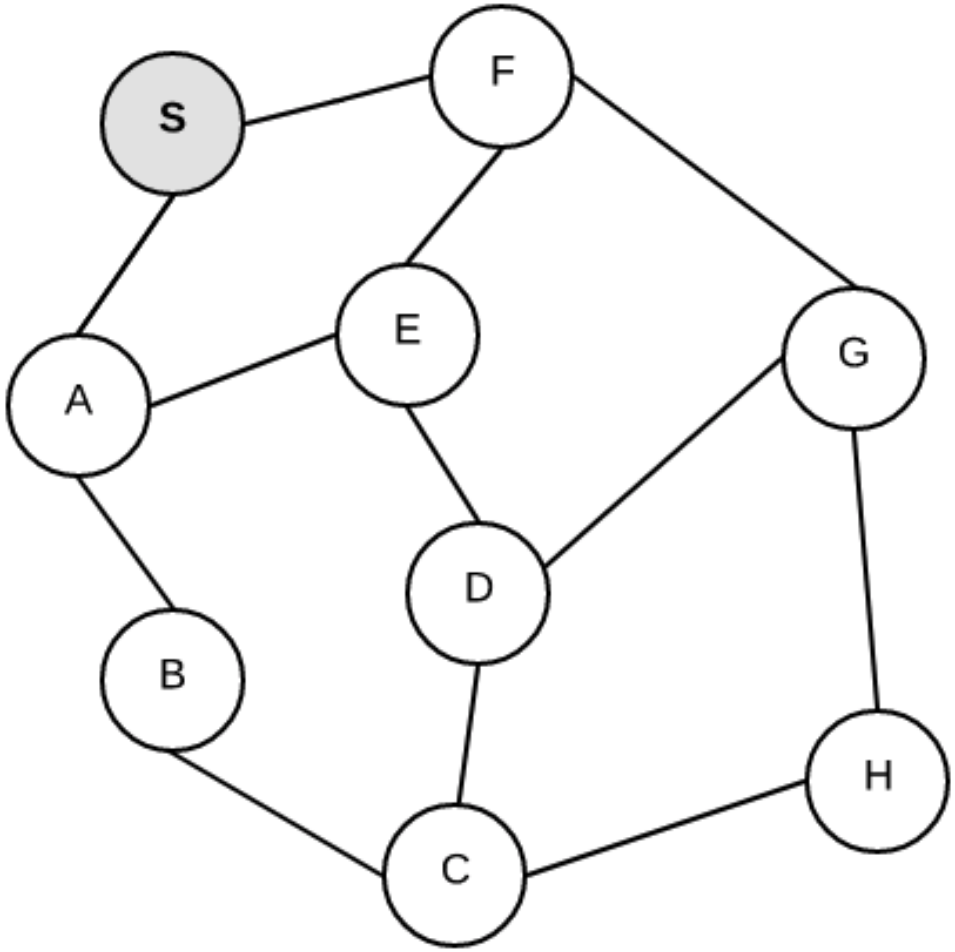
BFS: Ilustrasi

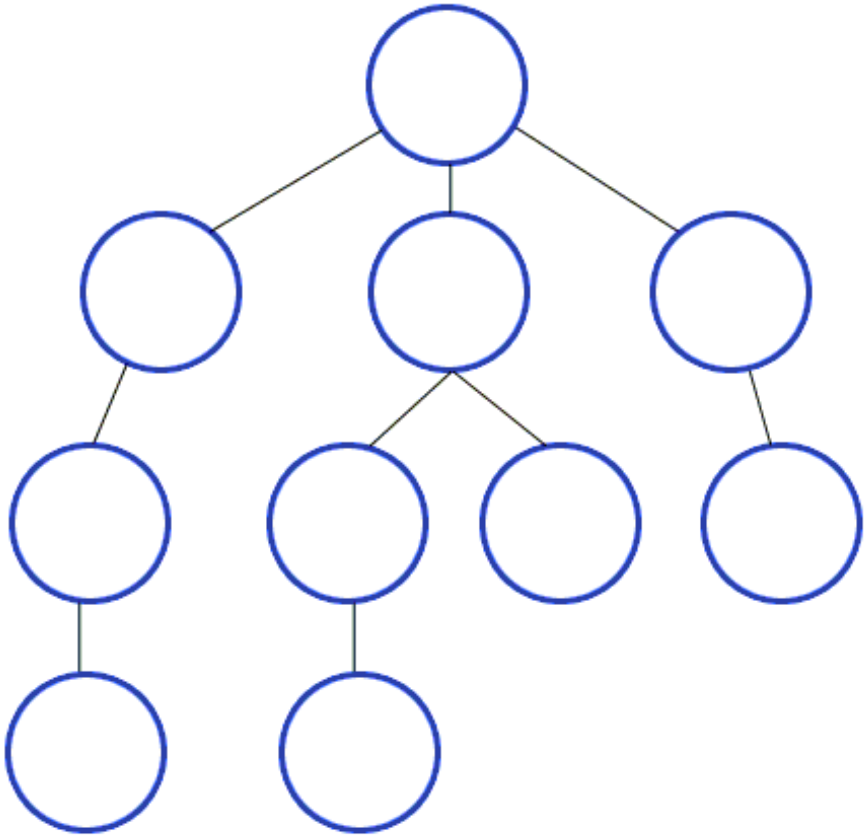


Iterasi	V	Q	dikunjungi							
			1	2	3	4	5	6	7	8
Inisialisasi	1	{1}	T	F	F	F	F	F	F	F
Iterasi 1	1	{2,3}	T	T	T	F	F	F	F	F
Iterasi 2	2	{3,4,5}	T	T	T	T	T	F	F	F
Iterasi 3	3	{4,5,6,7}	T	T	T	T	T	T	T	F
Iterasi 4	4	{5,6,7,8}	T	T	T	T	T	T	T	T
Iterasi 5	5	{6,7,8}	T	T	T	T	T	T	T	T
Iterasi 6	6	{7,8}	T	T	T	T	T	T	T	T
Iterasi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

Fig 2

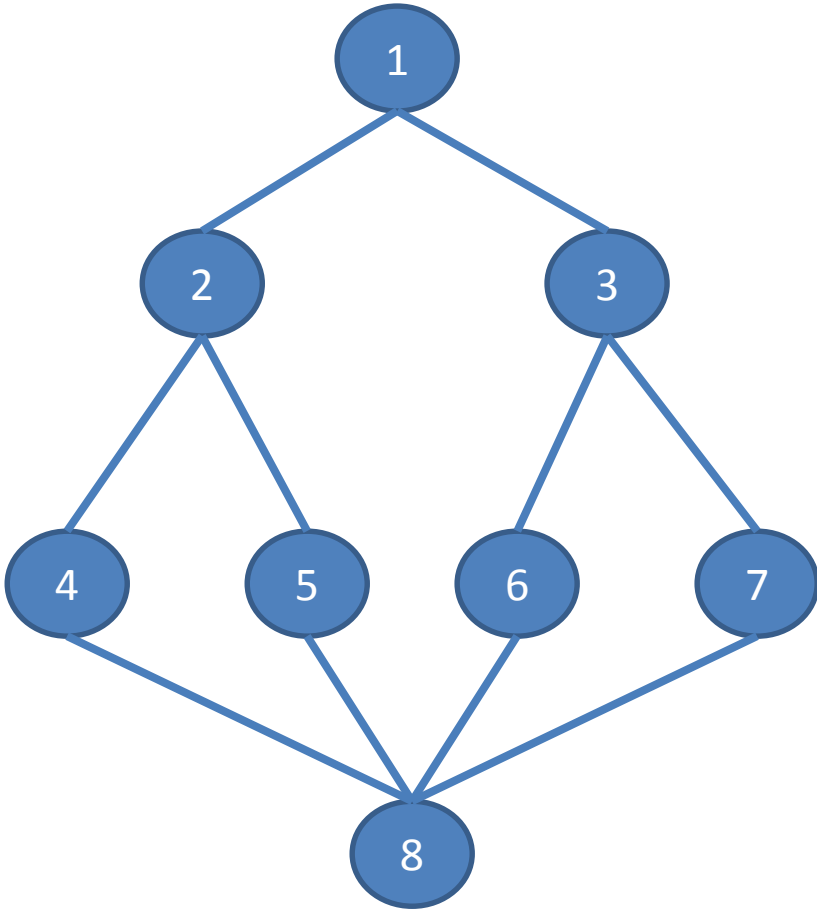




Pencarian Mendalam (DFS)

- Traversal dimulai dari simpul *v*.
- Algoritma:

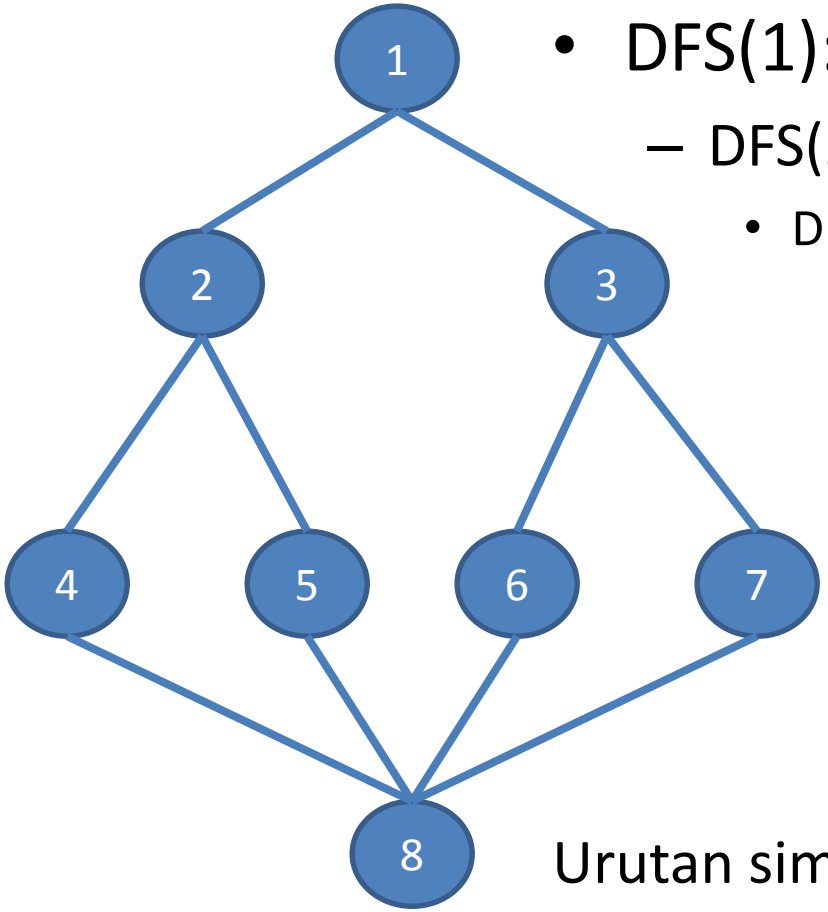
1. Kunjungi simpul *v*
2. Kunjungi simpul w yang bertetangga dengan simpul *v*.
3. Ulangi DFS mulai dari simpul *w*.
4. Ketika mencapai simpul *u* sedemikian sehingga semua simpul yang bertetangga dengannya telah dikunjungi, pencarian dirunut-balik (*backtrack*) ke simpul terakhir yang dikunjungi sebelumnya dan mempunyai simpul w yang belum dikunjungi.
5. Pencarian berakhir bila tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi.



DFS

```
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
```

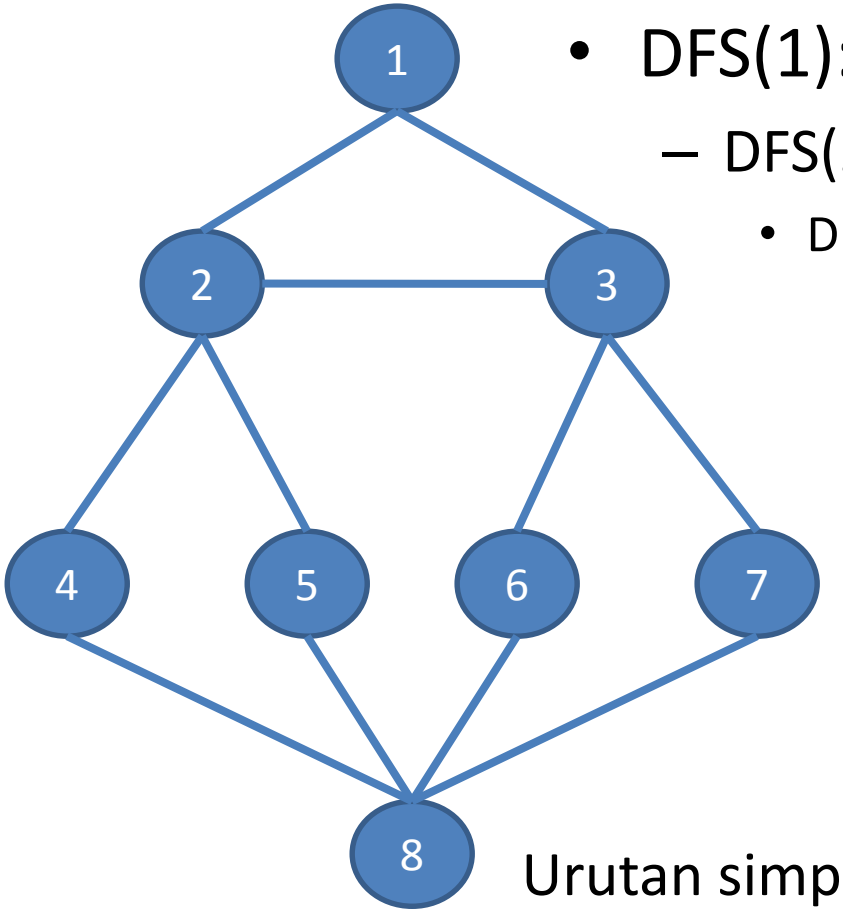
DFS: Ilustrasi 1



- DFS(1): v=1; dikunjungi[1]=true; DFS(2)
  - DFS(2): v=2; dikunjungi[2]=true; DFS(4)
    - DFS(4): v=4; dikunjungi[4]=true; DFS(8)
      - DFS(8): v=8; dikunjungi[8]=true; DFS(5)
        - » DFS(5): v=5; dikunjungi[5]=true
        - » DFS(6): v=6; dikunjungi[6]=true; DFS(3)
          - DFS(3): v=3; dikunjungi[3]=true; DFS(7)
            - DFS(7): v=7; dikunjungi[7]=true

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

DFS: Ilustrasi 2

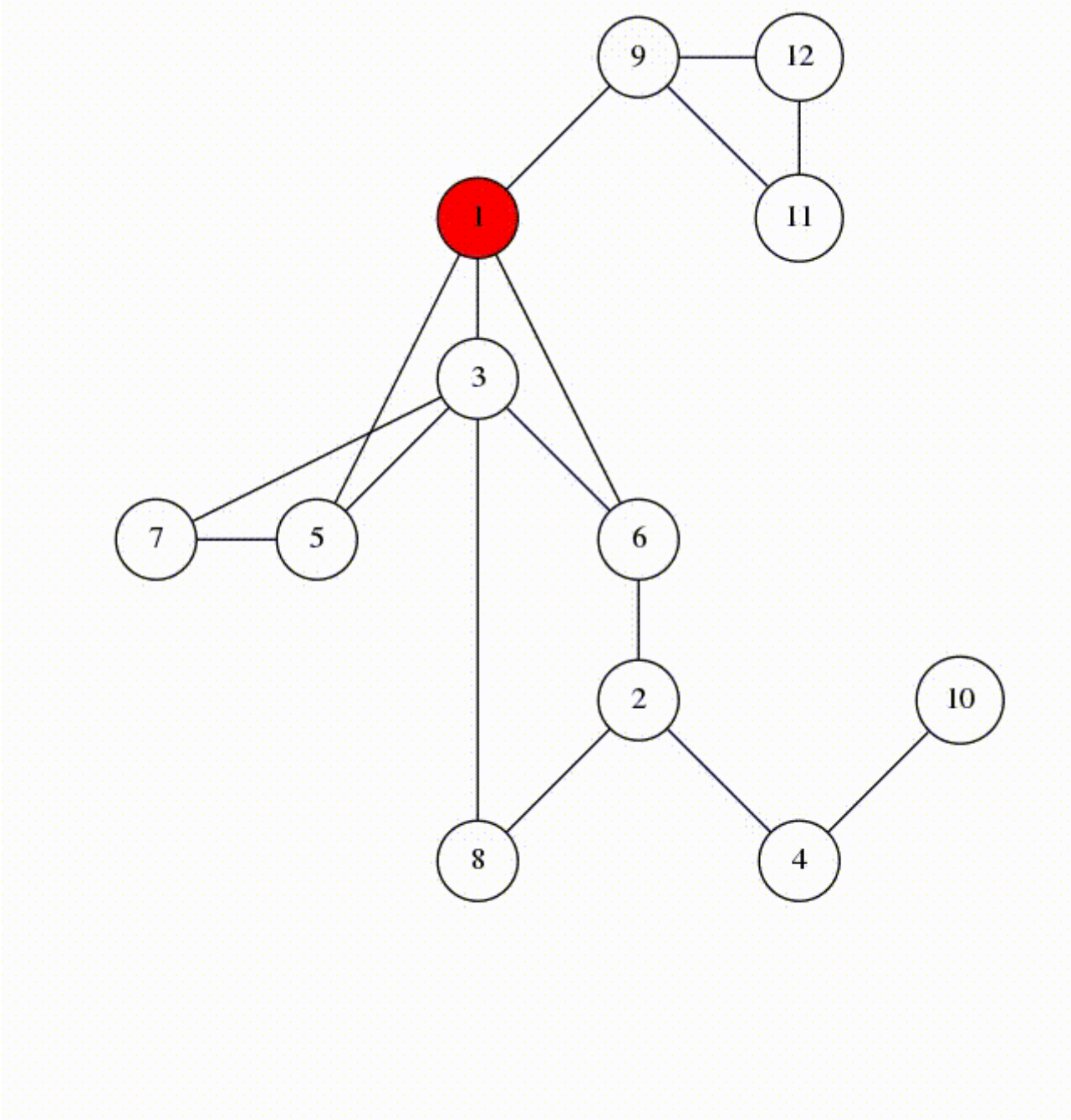


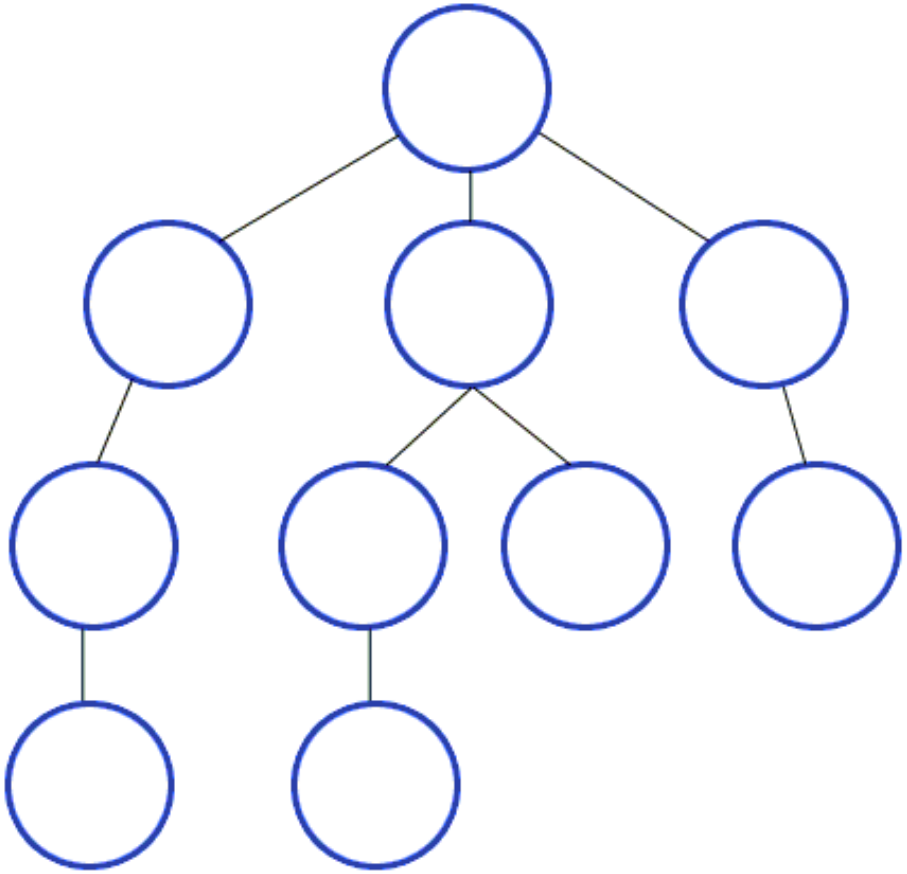
- DFS(1): v=1; dikunjungi[1]=true; DFS(2)
  - DFS(2): v=2; dikunjungi[2]=true; DFS(3)
    - DFS(3): v=3; dikunjungi[3]=true; DFS(6)
      - DFS(6): v=6; dikunjungi[6]=true; DFS(8)
        - » DFS(8): v=8; dikunjungi[8]=true; DFS(4)
          - DFS(4): v=4; dikunjungi[4]=true;

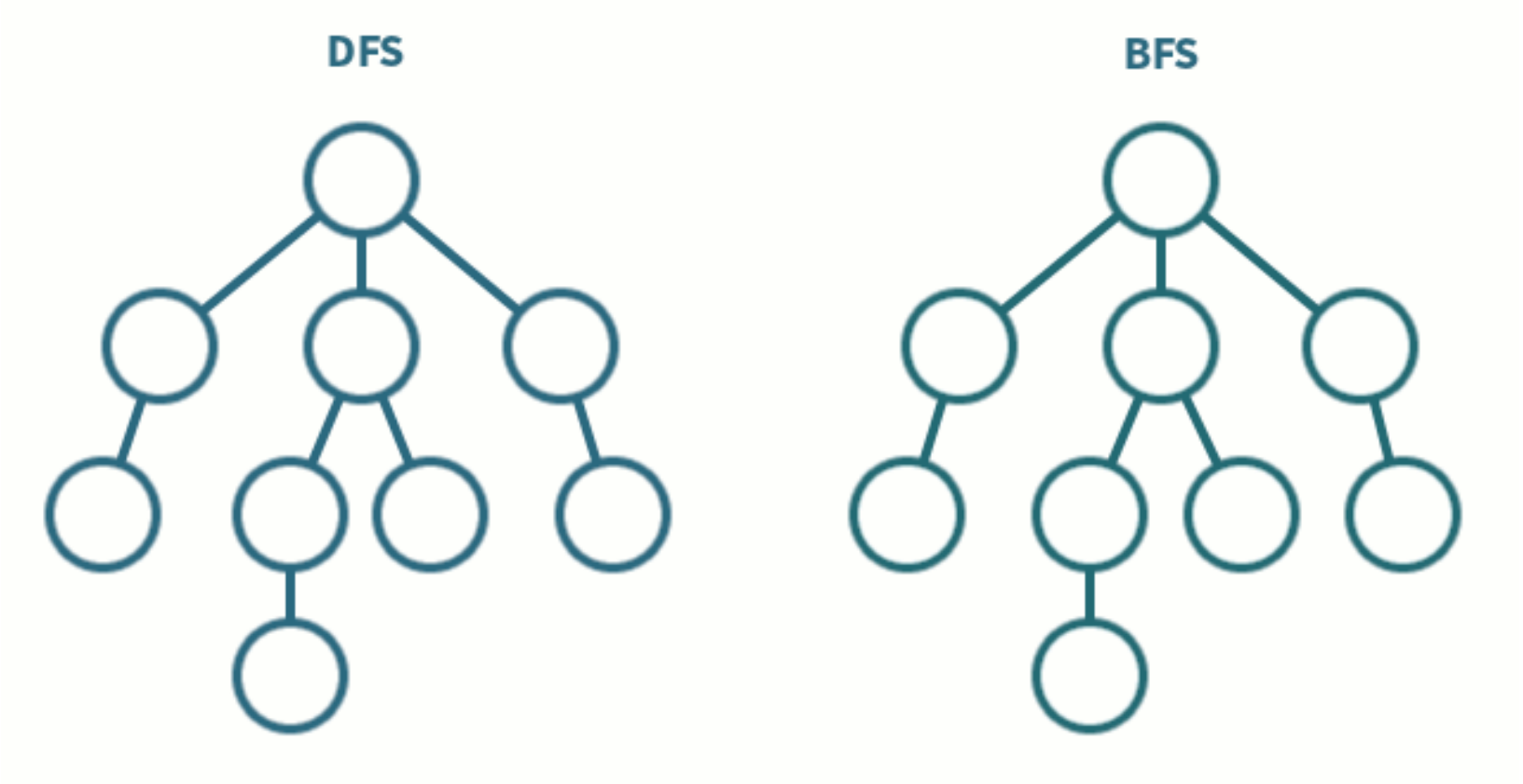
        DFS(8): DFS(5)
          - DFS(5): v=5; dikunjungi[5]=true

        DFS(8): DFS(7)
          - DFS(7): v=7; dikunjungi[7]=true

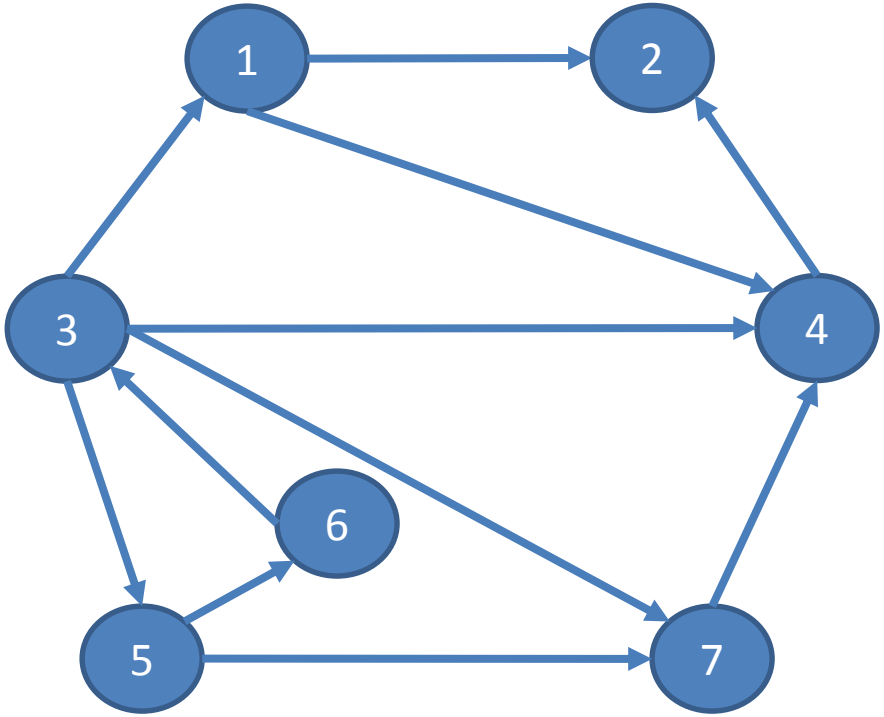
Urutan simpul2 yang dikunjungi: 1, 2, 3, 6, 8, 4, 5, 7


9
12
1
11
3
7
5
6
2
10
8
4




DFS
BFS

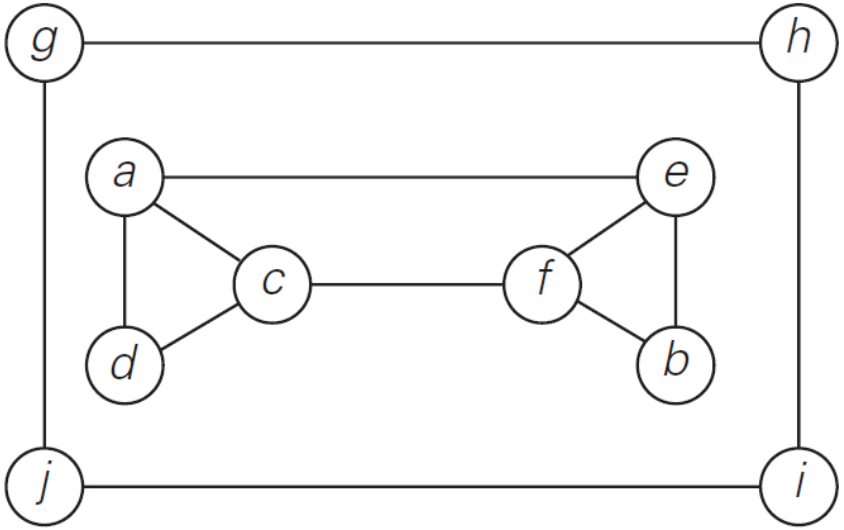
Contoh (hal 113)



- Khusus untuk graf berarah, beberapa simpul mungkin tidak dapat dicapai dari simpul awal. Coba dengan simpul yang belum dikunjungi sebagai simpul awal. (hal 113)

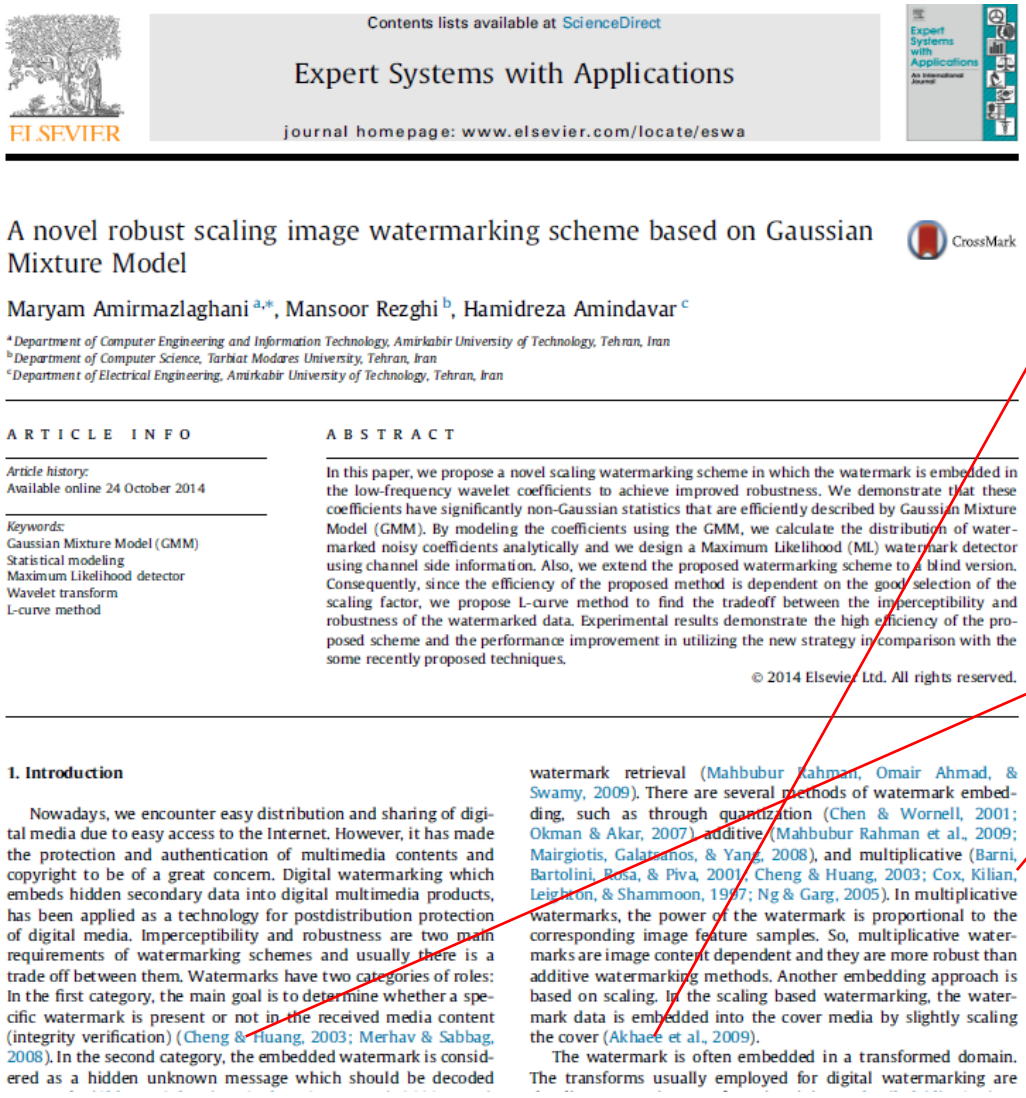
- DFS (1): 1-2-4-3-5-6-7
- BFS (1): 1-2-4-3-5-7-6

Contoh Lain



- Bagaimana penelusuran graf dengan BFS?
- Bagaimana Penelusuran graf dengan DFS

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.

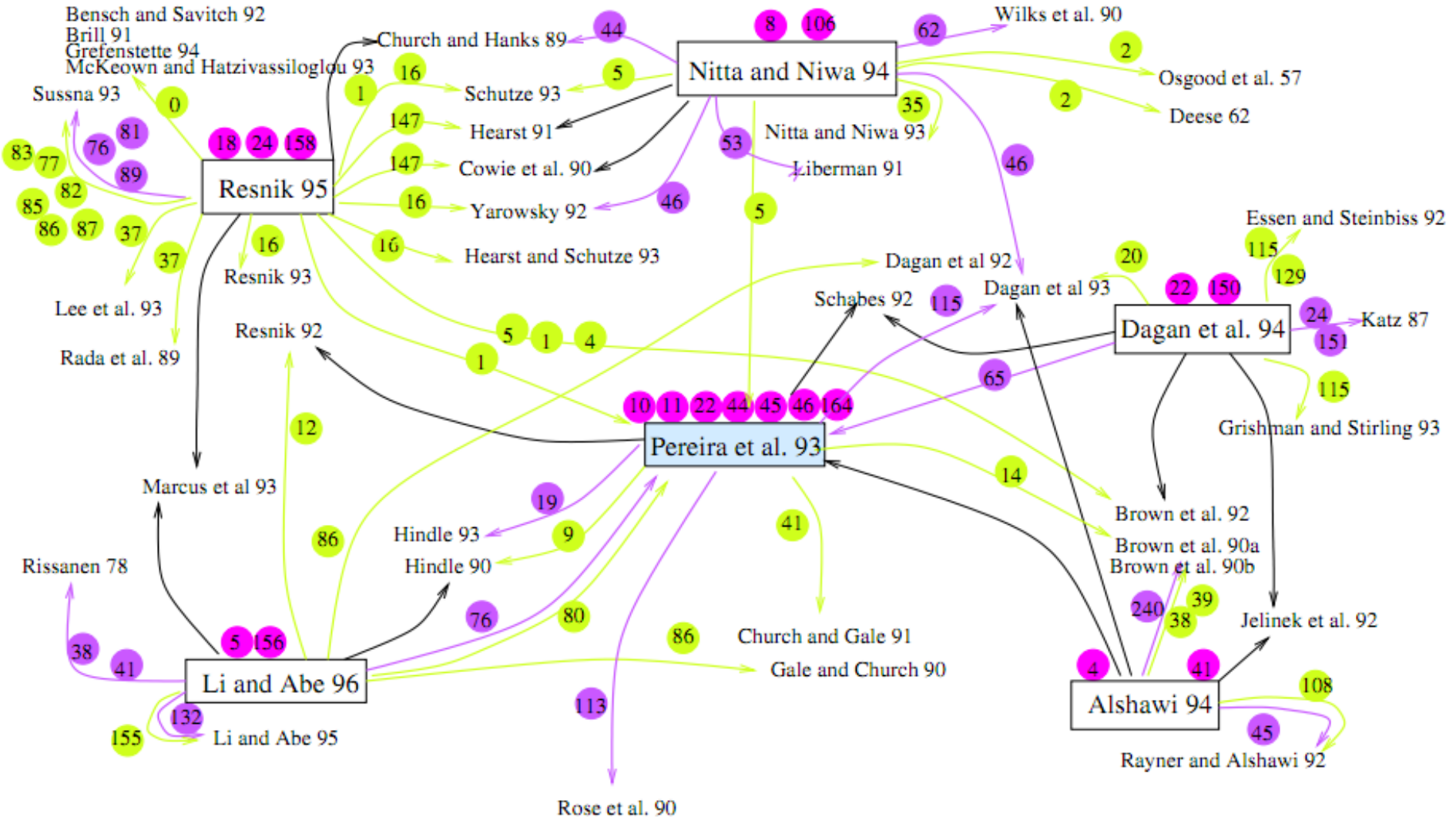
Cox, I., Kilian, J., Leighton, T., & Shammoon, T. (1997). Secure spread spectrum watermarking for multimedia. *IEEE Transactions on Image Processing, 6*, 1673–1687.

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.

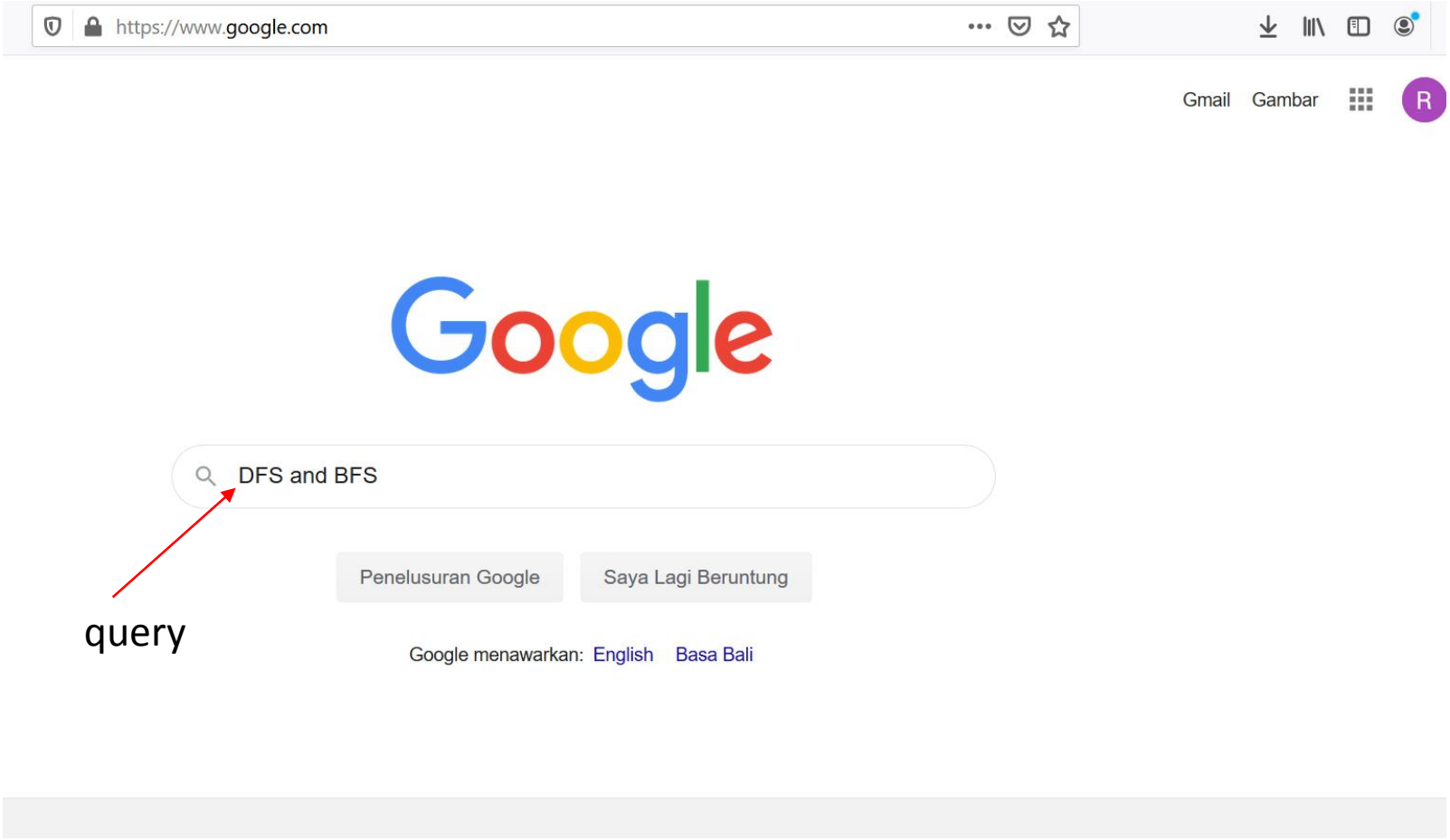
Gembicki, F., & Haimes, Y. (1975). Approach to performance and sensitivity multiobjective optimization: The goal attainment method. *IEEE Transactions on Automatic Control, AC-20*, 769–771.

Citation Map:



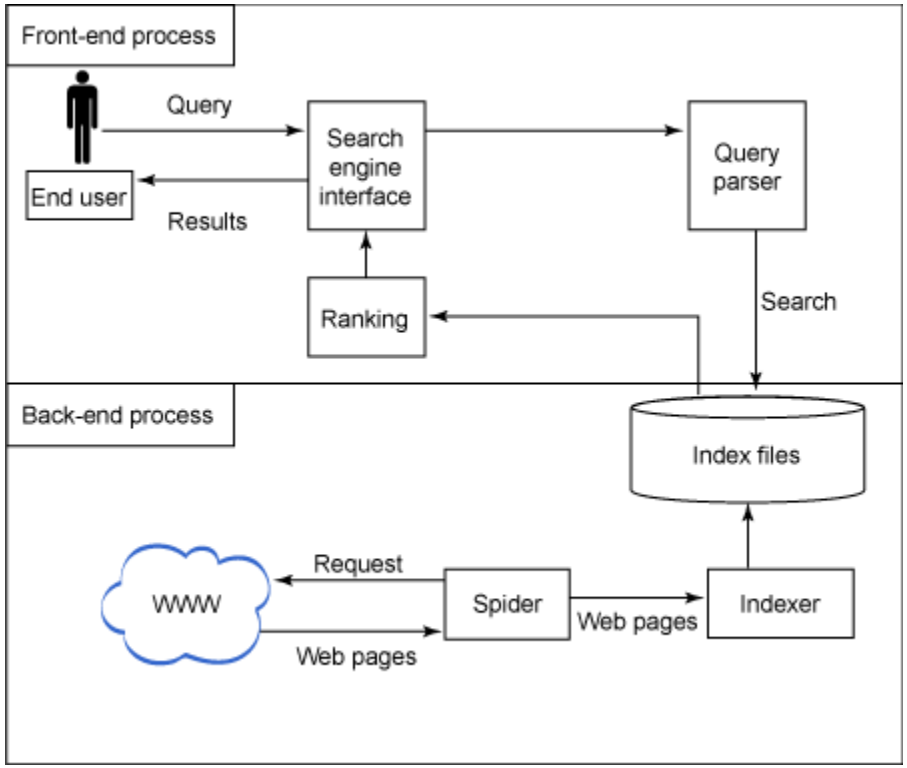
Sumber: Teufel (1999), Argumentative Zoning

Penerapan BFS dan DFS: Web Spider



Penerapan BFS dan DFS: Web Spider

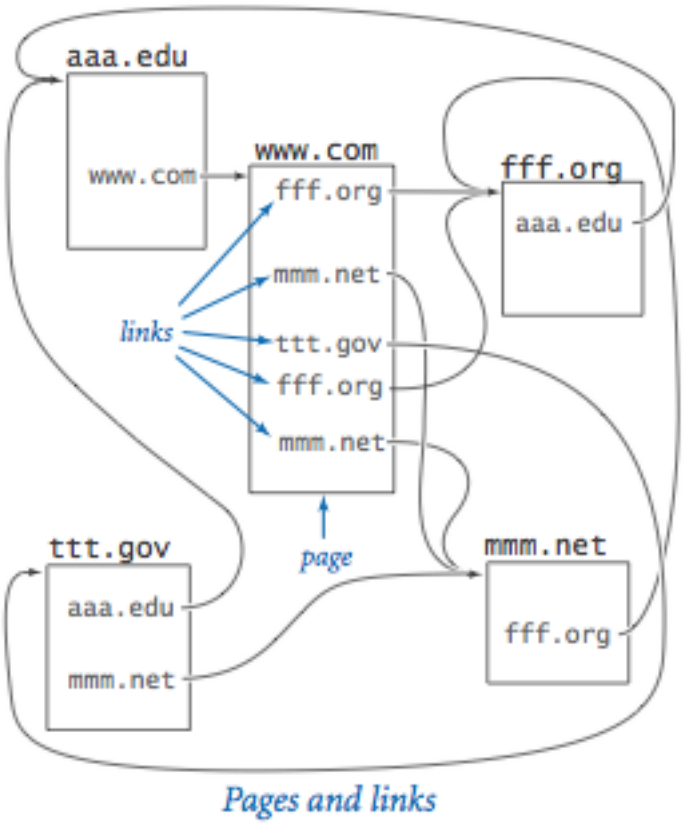
Arsitektur umum mesin pencari



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

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

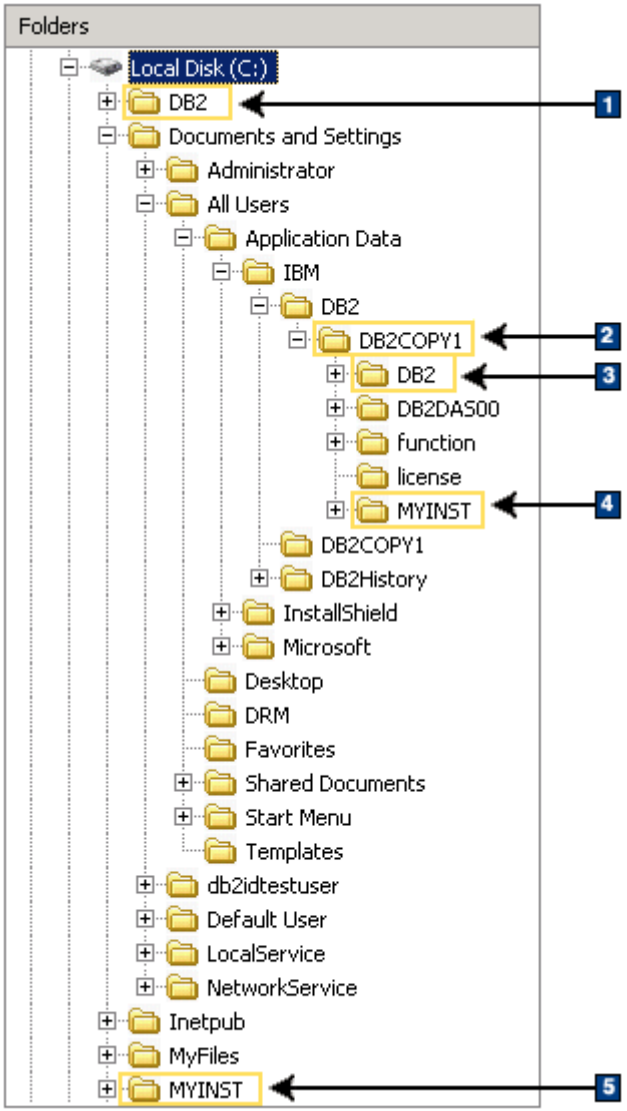
Web Spider: Penjelajahan Web

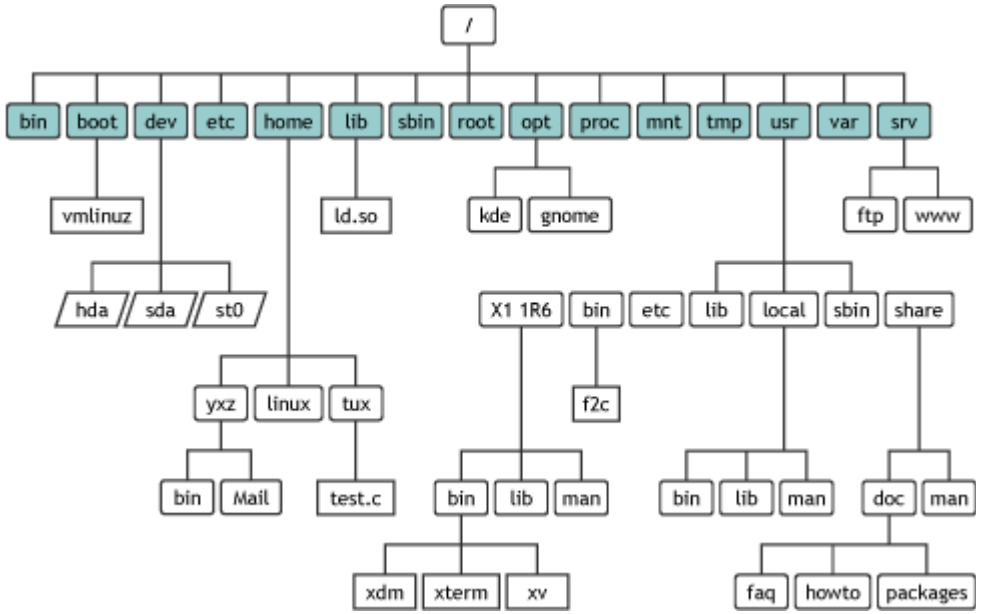


http://introcs.cs.princeton.edu/java/16pagerank/

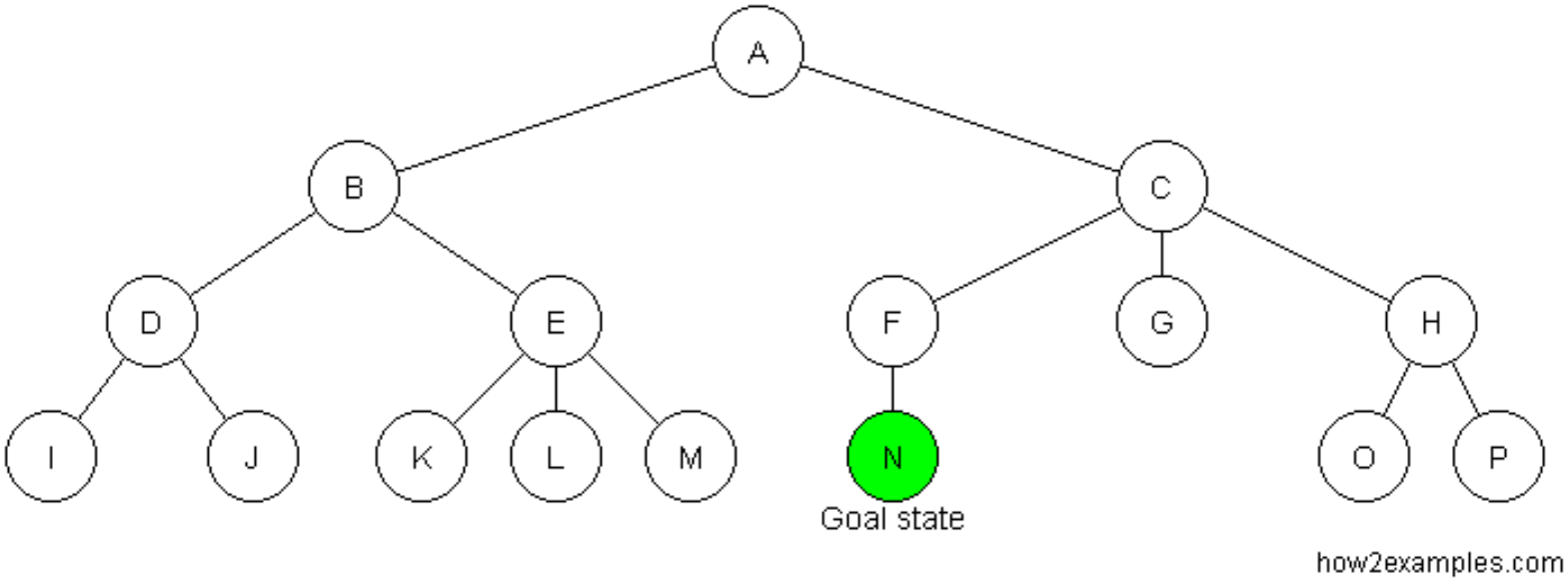
- Halaman web dimodelkan sebagai graf berarah
  - Simpul menyatakan halaman web (web page)
  - Sisi menyatakan link ke halaman web
- Bagaimana teknik menjelajahi web? Secara DFS atau BFS
- Dimulai dari web page awal, lalu setiap link ditelusuri secara DFS sampai setiap web page tidak mengandung link.

DFS dan BFS untuk penelusuran direktori (folder)

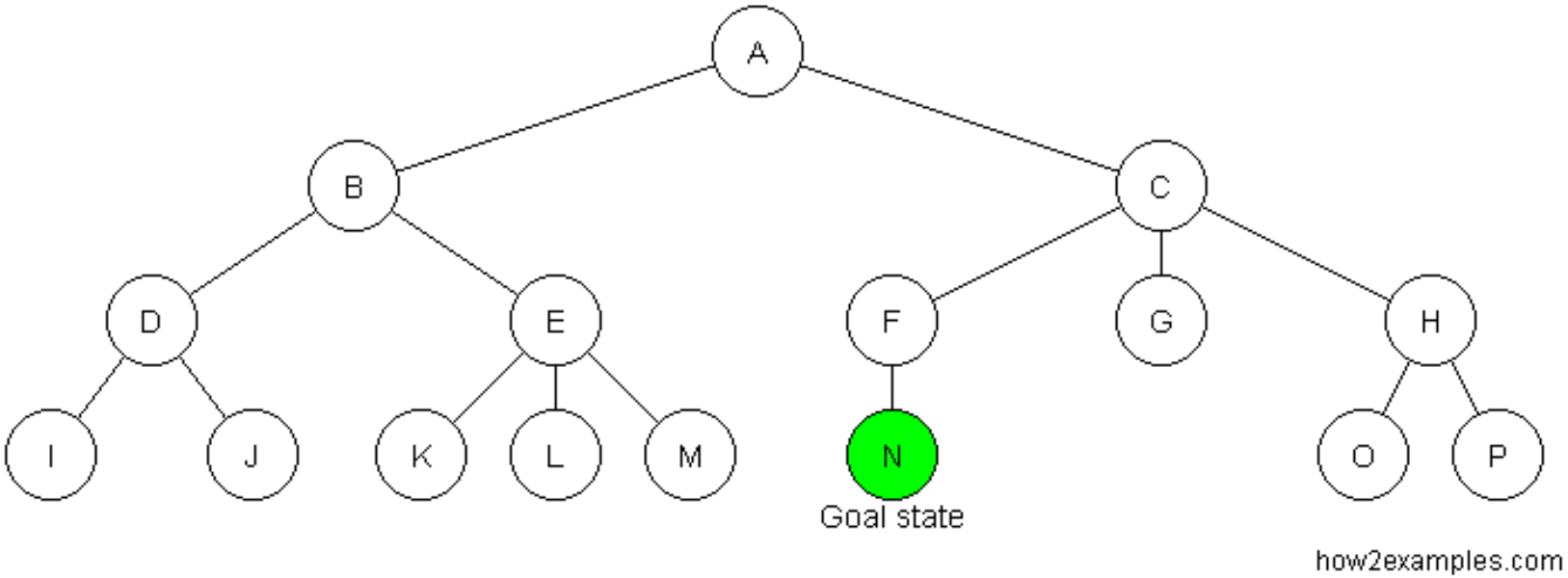




Pencarian dokumen di dalam direktori (folder) secara BFS



Pencarian dokumen di dalam direktori (folder) secara DFS

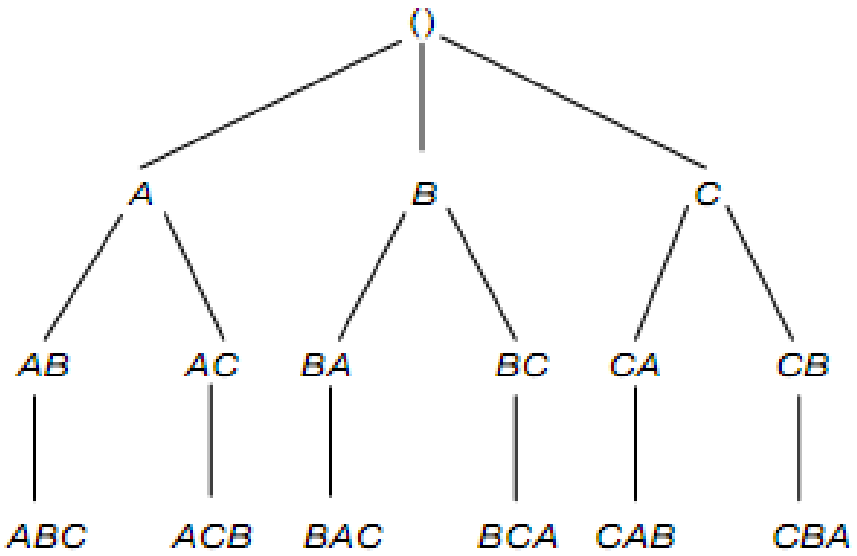


GRAF DINAMIS

Pencarian Solusi dengan BFS/DFS

- Menyelesaikan persoalan dengan melakukan pencarian
- Pencarian solusi ≈ pembentukan pohon dinamis
  - Setiap simpul diperiksa apakah solusi (goal) telah dicapai atau tidak. Jika simpul merupakan solusi , pencarian dapat selesai (satu solusi) atau dilanjutkan mencari solusi lain (semua solusi).
- Representasi pohon dinamis:
  - Pohon ruang status (*state space tree*)
  - Simpul: *problem state* (layak membentuk solusi)
    - Akar: *initial state*
    - Daun: *solution/goal state*
  - Cabang: operator/langkah dalam persoalan
  - Ruang status (*state space*): himpunan semua simpul
  - Ruang solusi: himpunan status solusi
- Solusi: path ke status solusi

Pohon Dinamis: Permutasi A,B,C



Ket: () = status kosong

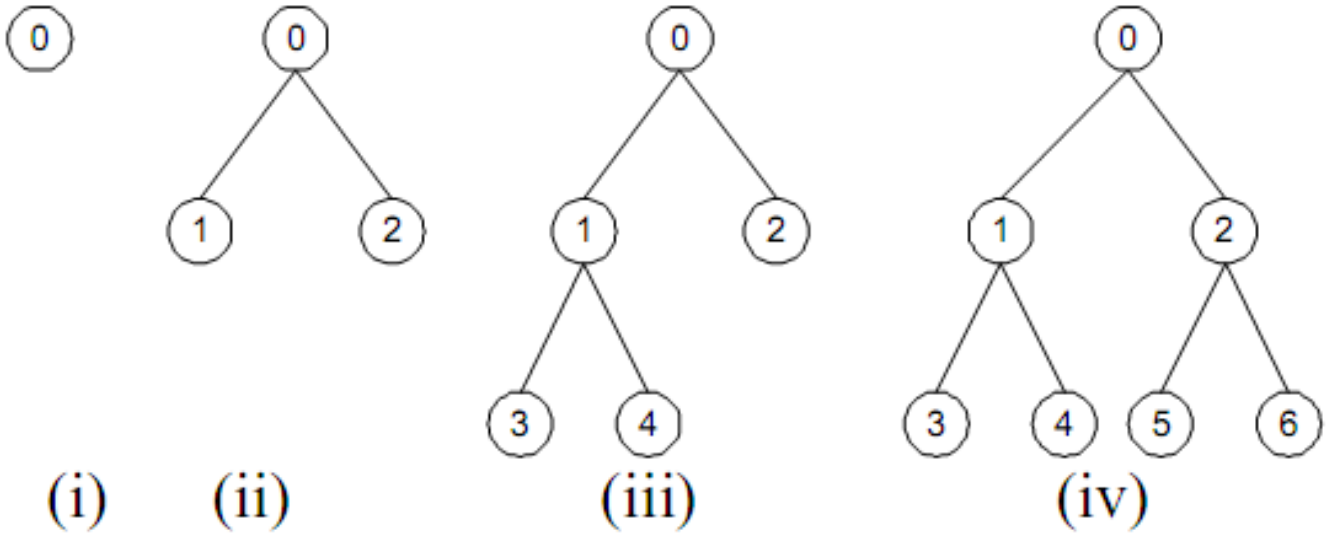
Pohon ruang status

- Operator: add X
- Akar : status awal (status "kosong")
- Simpul: problem state
  - Status persoalan (*problem state*): simpul-simpul di dalam pohon dinamis yang memenuhi kendala (constraints).
- Daun: status solusi
  - Status solusi (*solution state*): satu atau lebih status yang menyatakan solusi persoalan.
- Ruang solusi:
  - Ruang solusi (*solution space*): himpunan semua status solusi.
- Ruang status (*state space*): Seluruh simpul di dalam pohon dinamis dan pohonnya dinamakan juga pohon ruang status (*state space tree*).

Pembangkitan Status

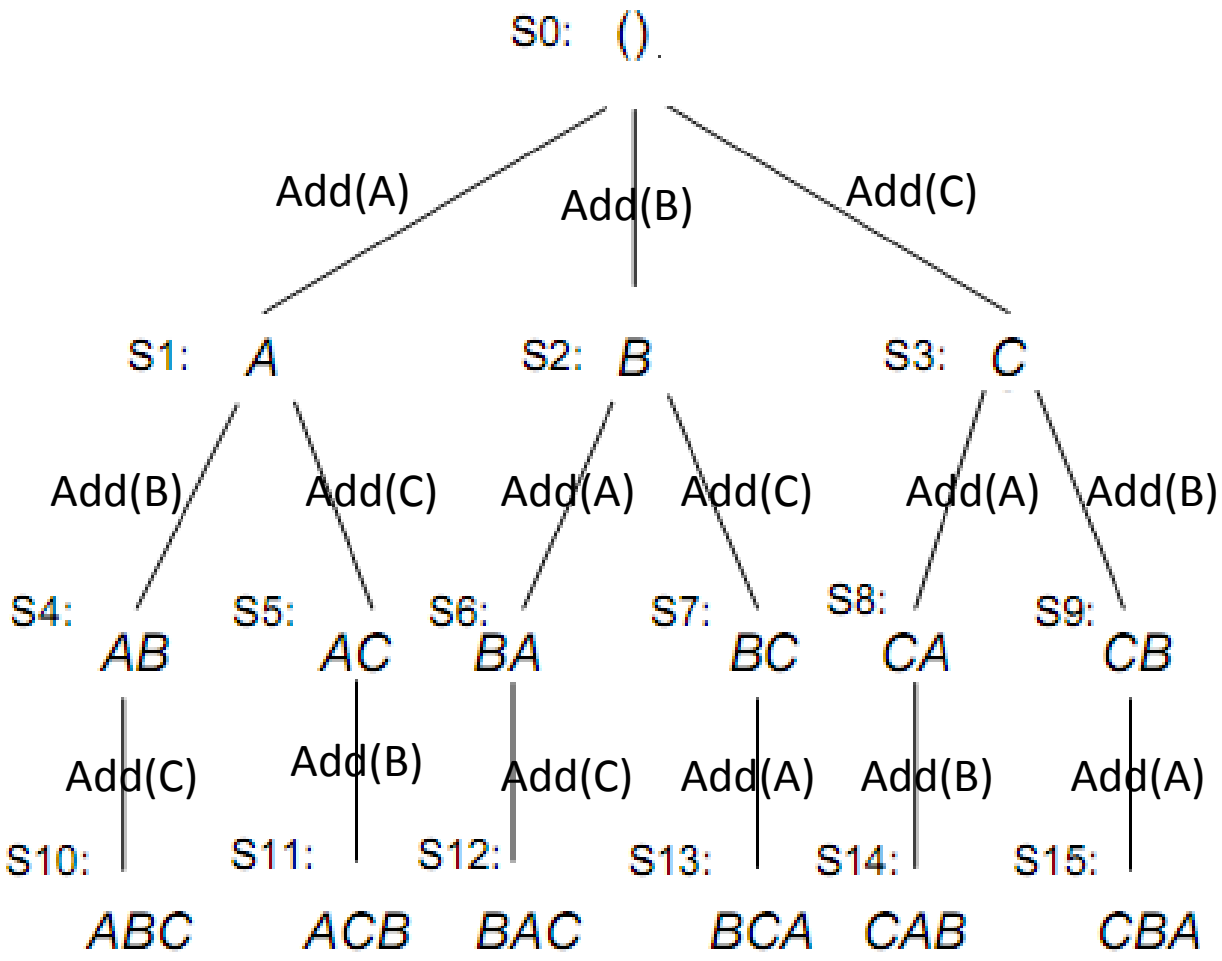
- Pembangkitan status baru dengan cara mengaplikasikan operator (langkah legal) kepada status (simpul) pada suatu jalur
- Jalur dari simpul akar sampai ke simpul (daun) goal berisi rangkaian operator yang mengarah pada solusi persoalan

BFS untuk Pembentukan Pohon Ruang Status



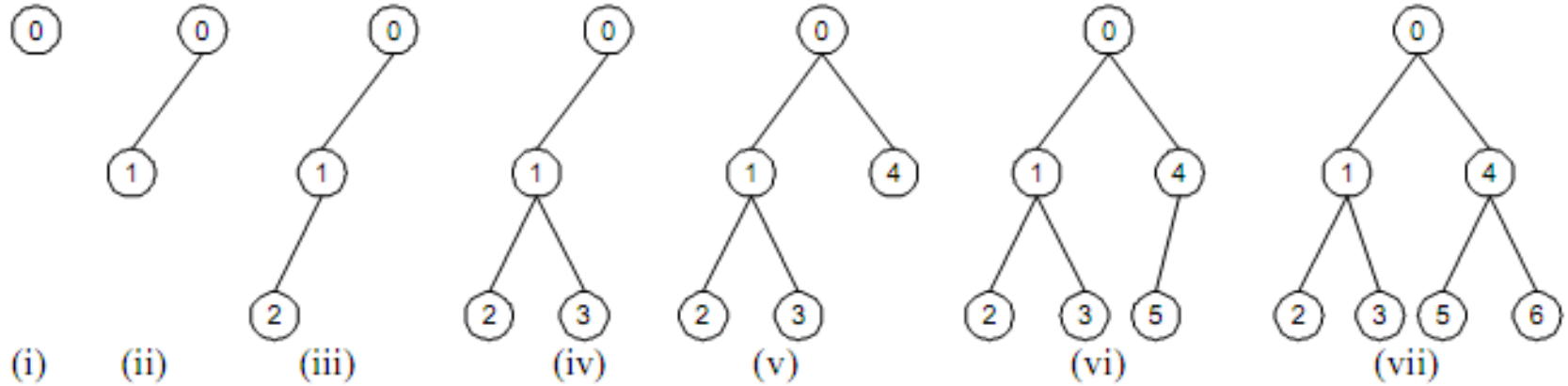
- Inisialisasi dengan status awal sebagai akar, lalu tambahkan simpul anaknya, dst.
- Semua simpul pada level d dibangkitkan terlebih dahulu sebelum simpul-simpul pada level d+1

BFS untuk Permutasi

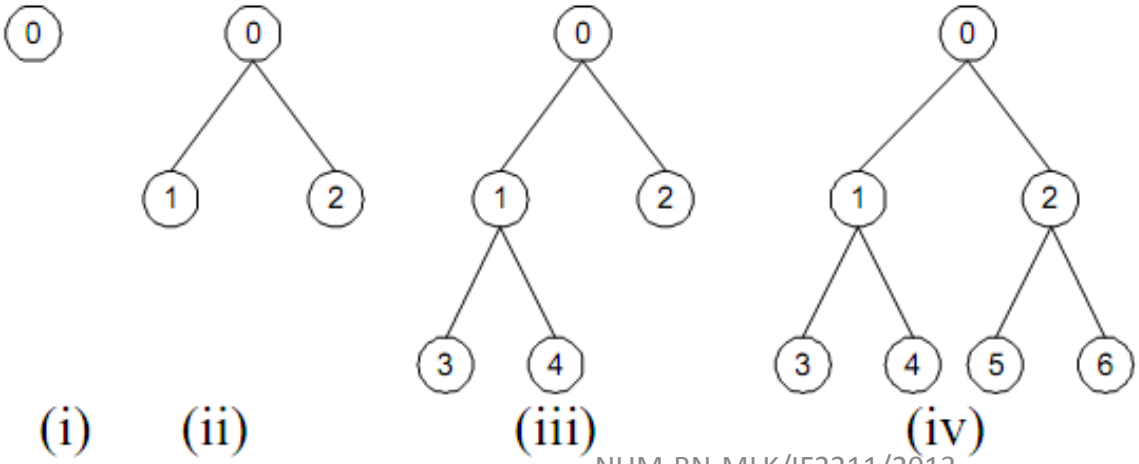


DFS untuk Pembentukan Pohon Ruang Status

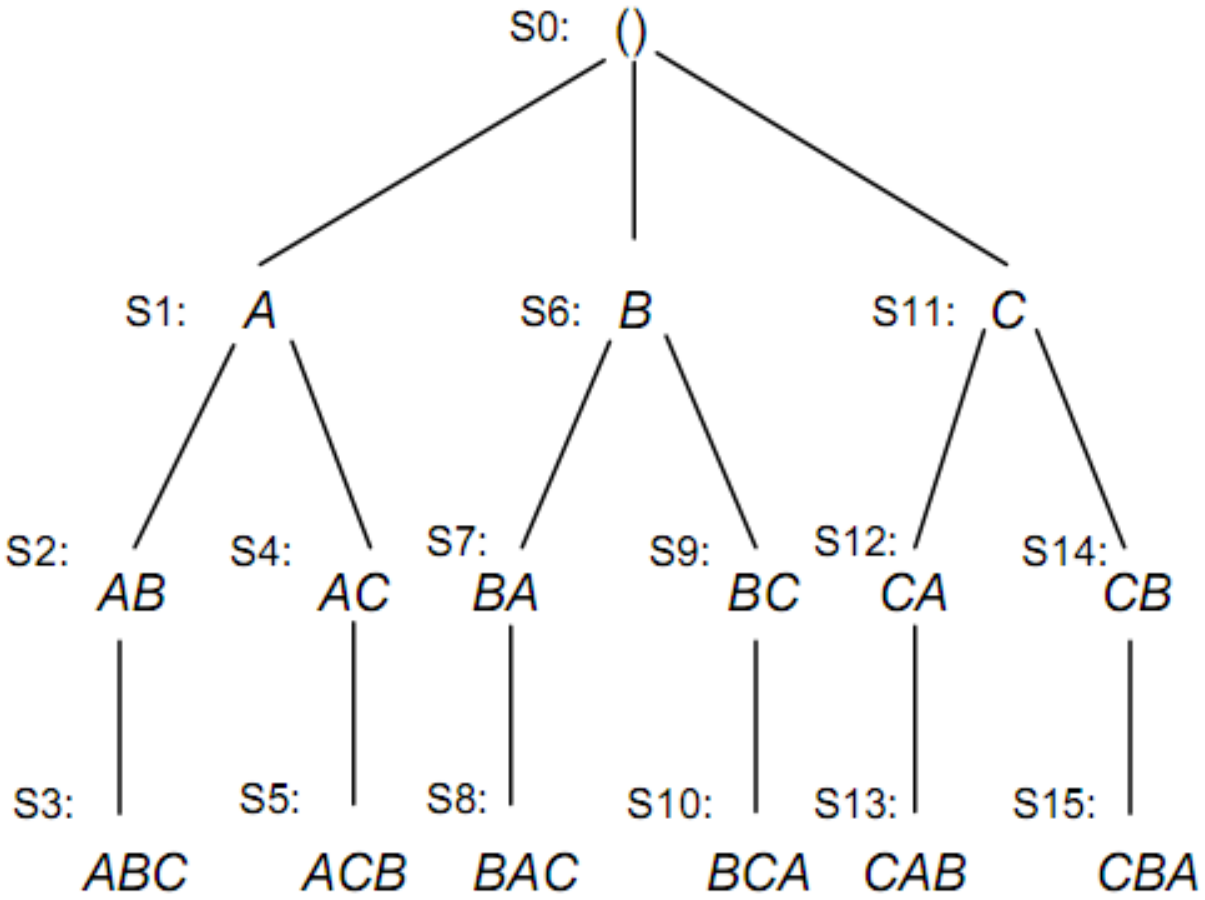
DFS:



BFS:



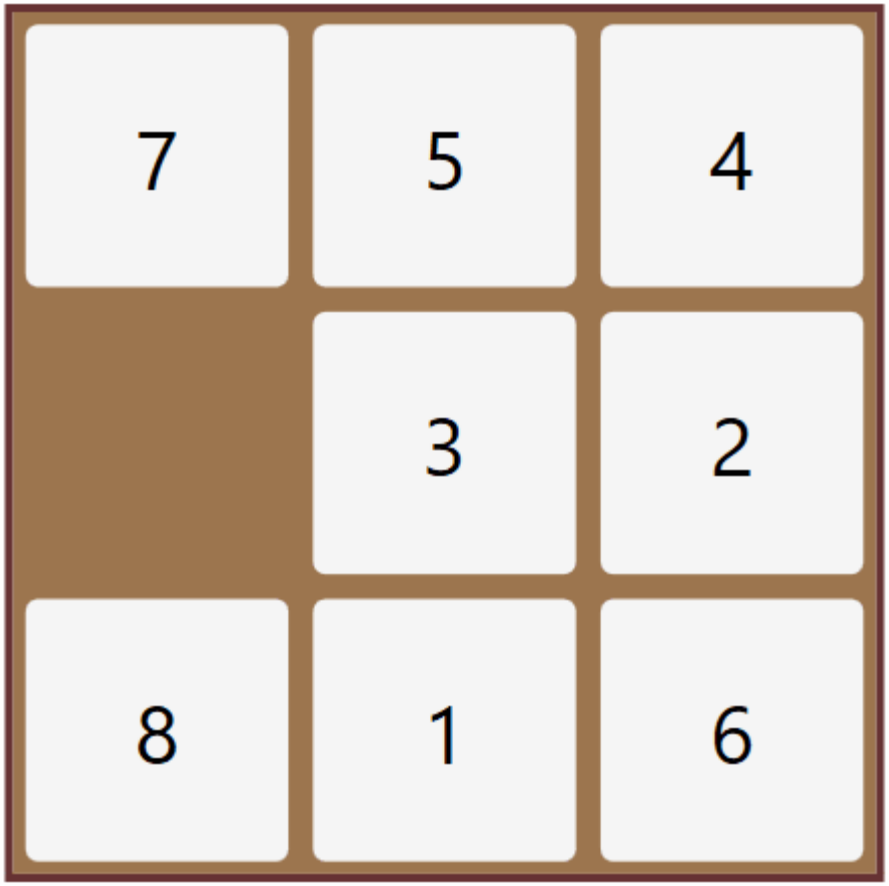
DFS Permutasi A,B,C



Evaluasi dari Teknik Pencarian

- Aspek untuk melihat seberapa 'baik' suatu teknik pencarian
  - *Completeness:* apakah menjamin ditemukannya solusi jika memang ada
  - *Optimality:* apakah teknik menjamin mendapatkan solusi yang optimal *(e.g: lowest path cost)?*
  - *Time Complexity:* waktu yang diperlukan untuk mencapai solusi
  - *Space Complexity:* memory yang diperlukan ketika melakukan pencarian
- Kompleksitas waktu dan ruang diukur dengan menggunakan istilah berikut.
  - *b*: (*branching factor*) maksimum pencabangan yang mungkin dari suatu simpul
  - *d*: (*depth*) kedalaman dari solusi terbaik (*cost* terendah)
  - *m*: maksimum kedalaman dari ruang status (bisa ∞)

Permainan 8-*Puzzle*



2	1	6
4		8
7	5	3

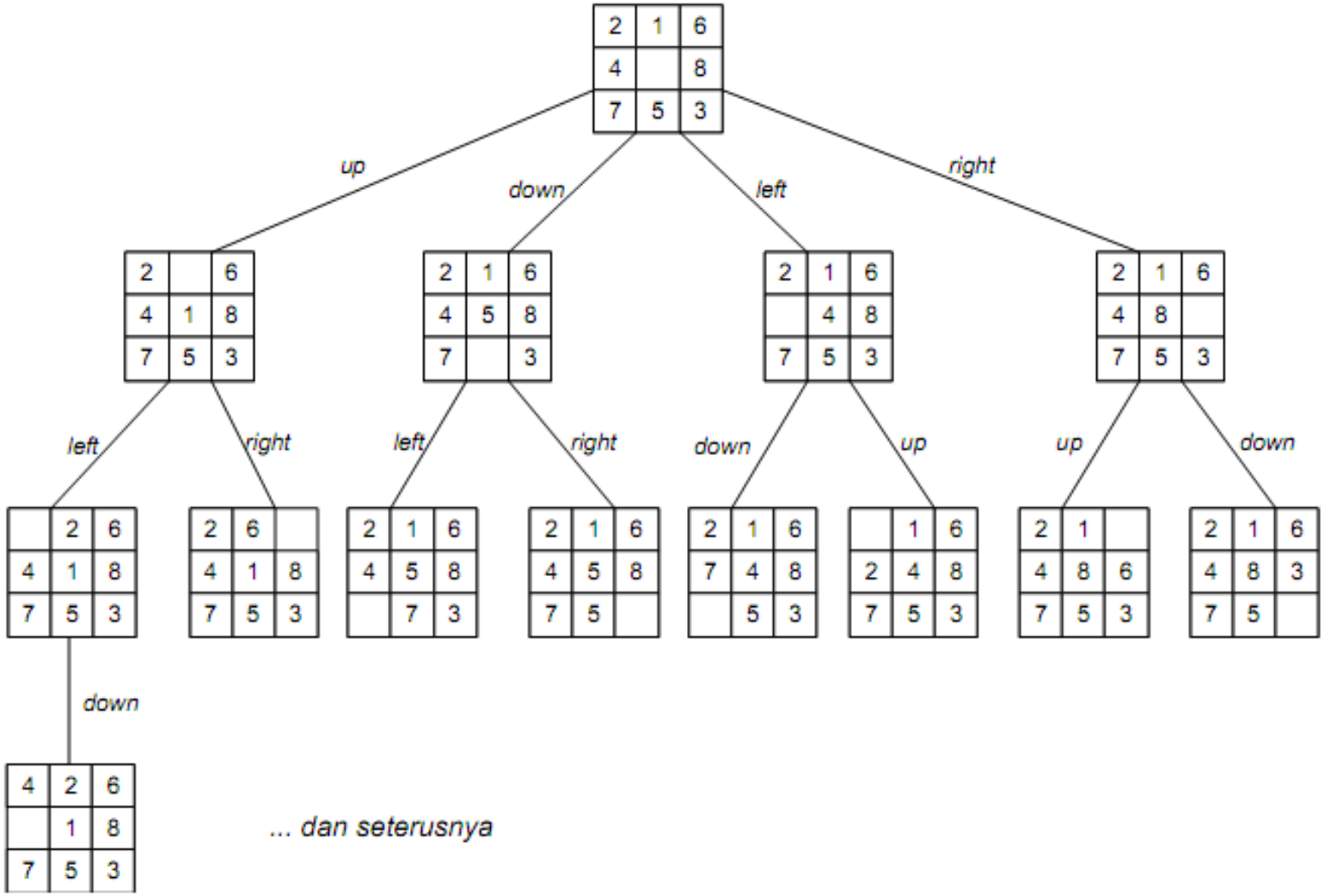
(a) Susunan awal (*initial state*)

1	2	3
8		4
7	6	5

(b) Susunan akhir (*goal state*)

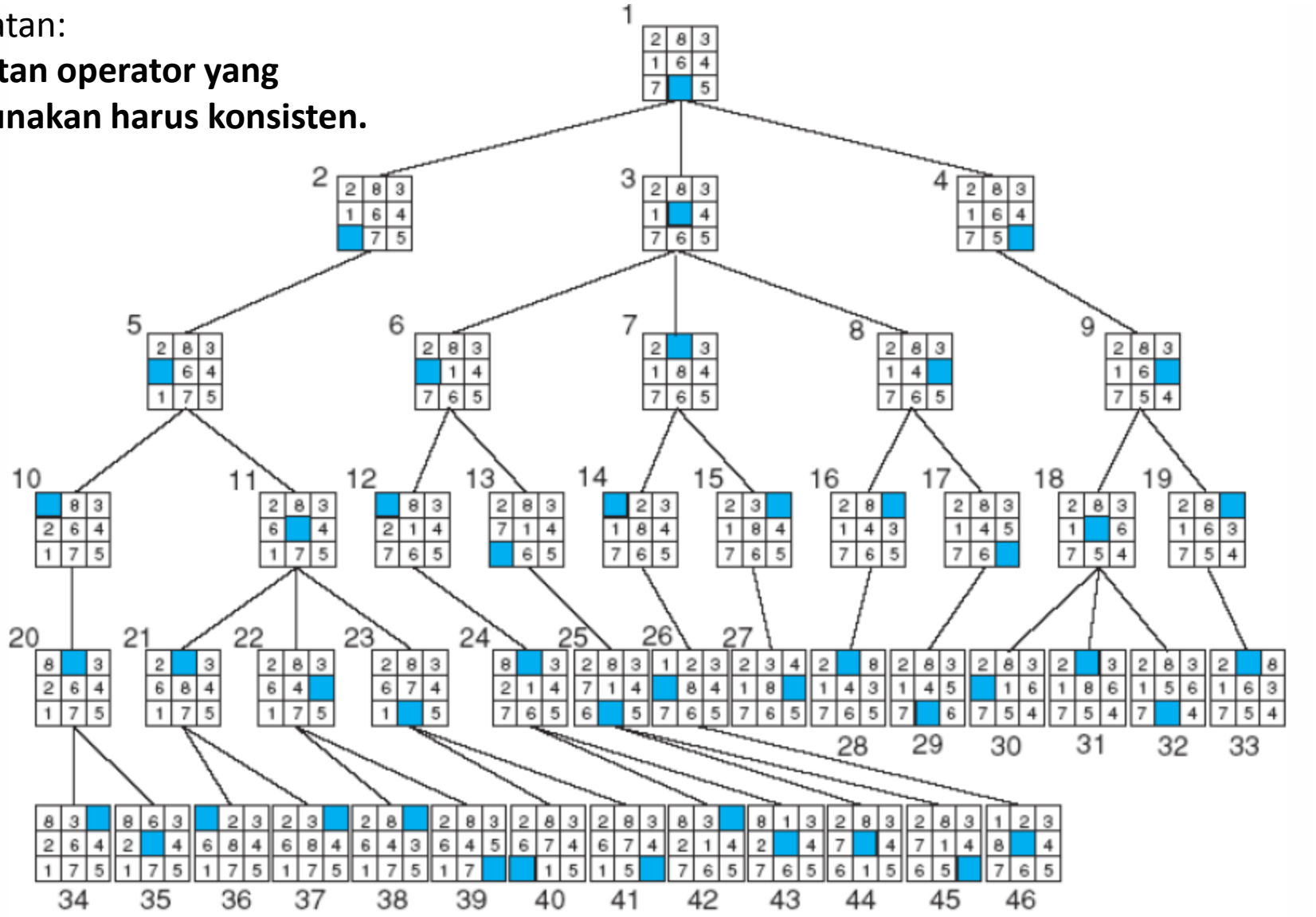
- State berdasarkan ubin kosong (*blank*)
- Operator: *up*, *down*, *left*, *right*

8-Puzzle: Pohon Ruang Status



BFS untuk 8-Puzzle

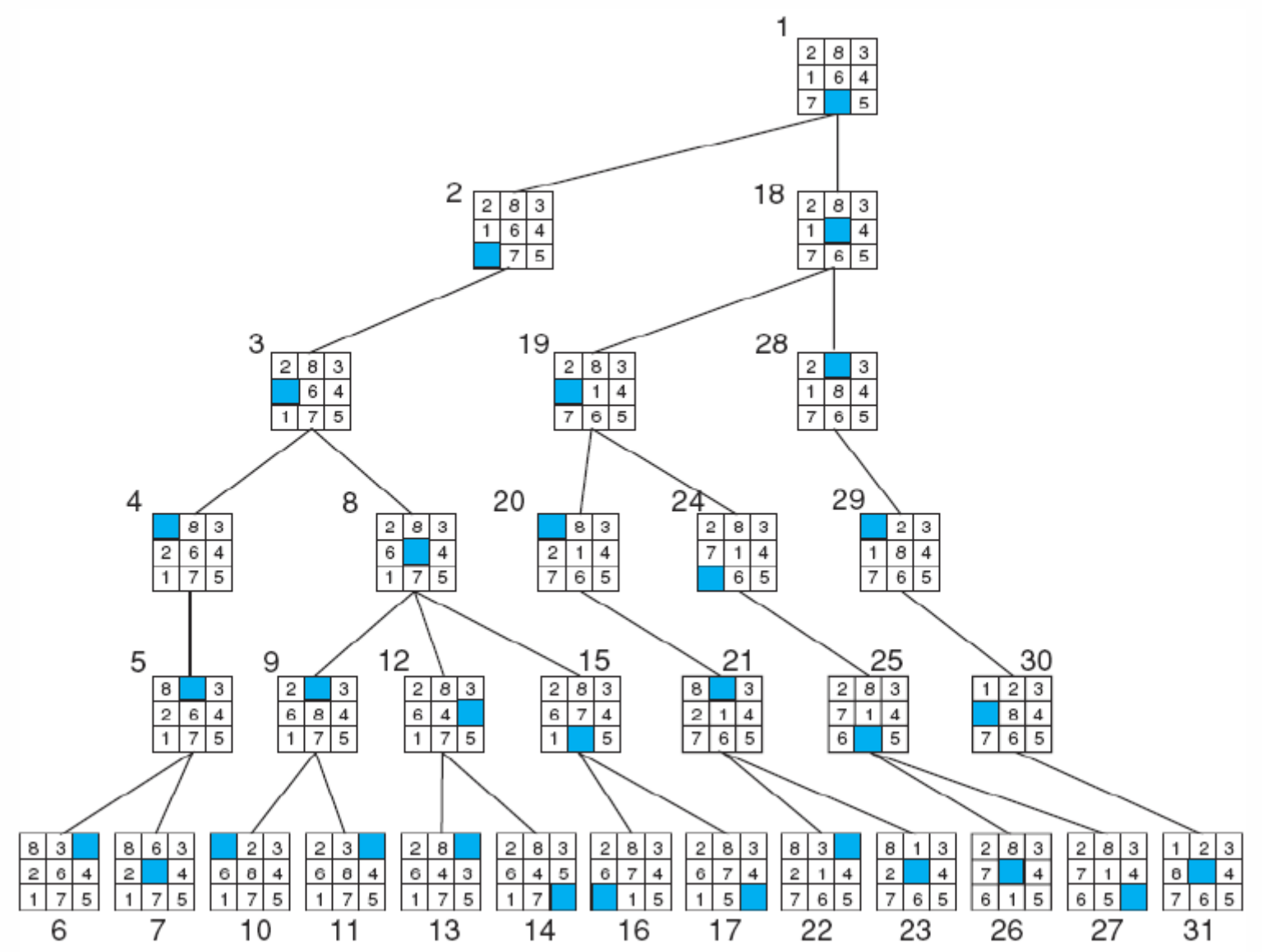
Catatan:
Urutan operator yang digunakan harus konsisten.



Bagaimana property dari BFS?

- *Completeness*?
  - Ya (selama nilai *b* terbatas)
- Optimality?
  - Ya, jika langkah = biaya
- Kompleksitas waktu:
  - $1+b+b^2+b^3+\ldots+b^d = O(b^d)$
- Kompleksitas ruang:
  - $O(b^d)$
- Kurang baik dalam kompleksitas ruang

DFS untuk 8-Puzzle



Bagaimana property dari DFS?

- *Completeness*?
  - Ya (selama nilai *b* terbatas, dan ada penanganan '*redundant paths*' dan '*repeated states*')
- *Optimality*?
  - Tidak
- Kompleksitas waktu:
  - O(b^m)
- Kompleksitas ruang:
  - O(*bm*)

- Kurang baik dalam kompleksitas waktu, lebih baik dalam kompleksitas ruang

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

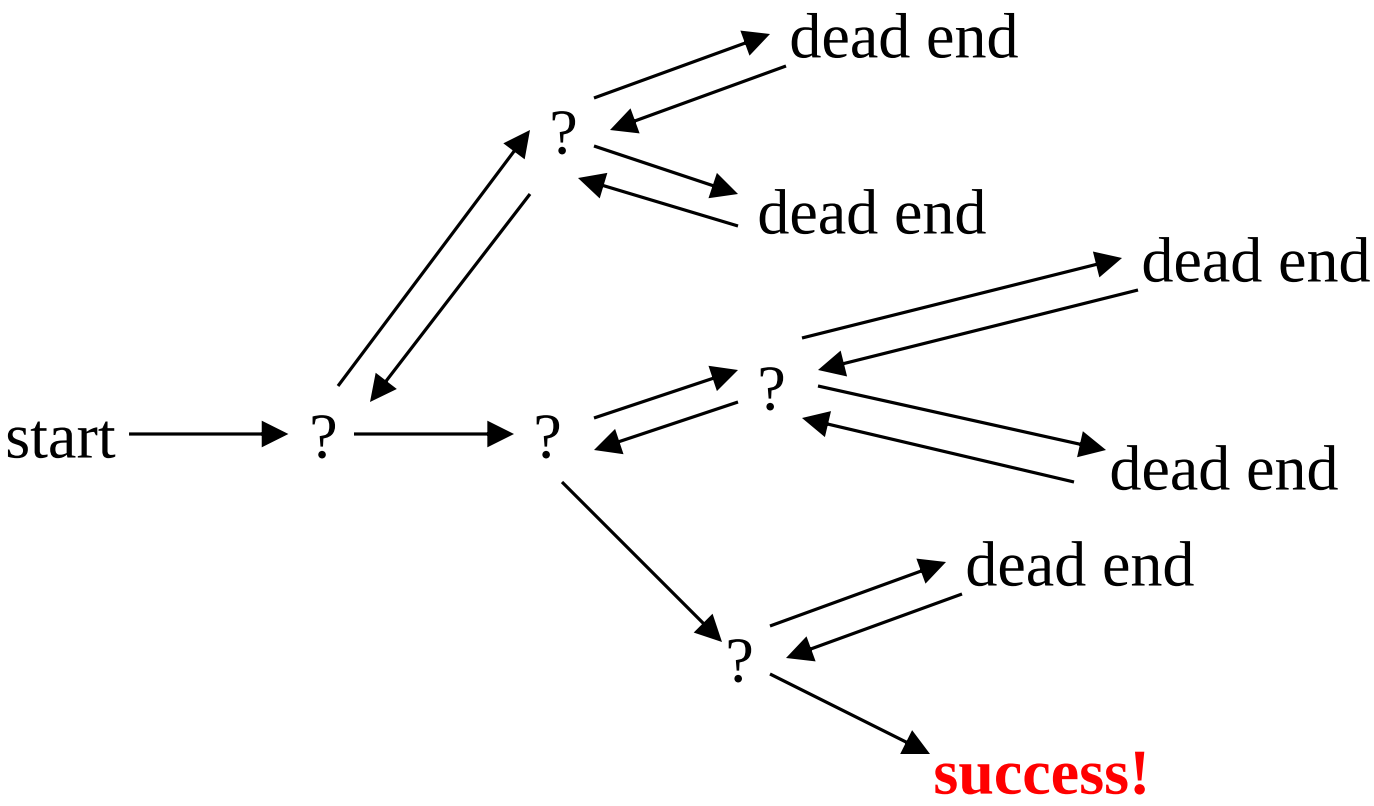
- Karakteristik *backtracking* di dalam algoritma DFS berguna untuk memecahkan persoalan pencarian solusi yang memiliki banyak alternatif pilihan selama pencarian.

- 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"

Animasi *Backtracking* *)

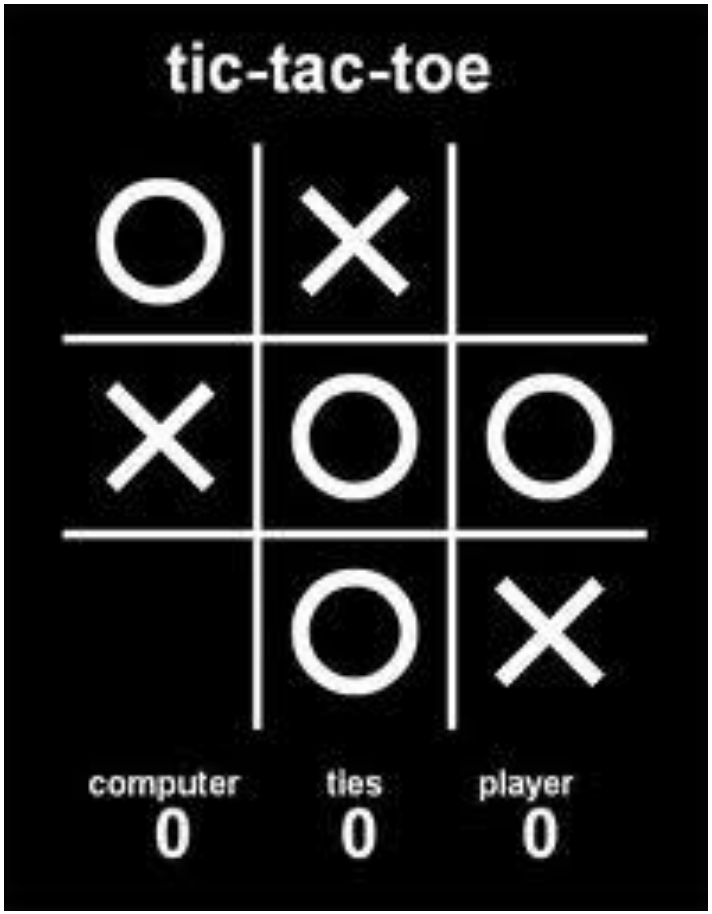


*) Sumber: *www.cis.upenn.edu/.../35-**backtracking**.ppt*

- 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

Permainan Bidak

Contoh. Sebuah bidak (pion) bergerak di dalam sebuah matriks pada Gambar 6.11. Bidak dapat memasuki elemen matriks mana saja pada baris paling atas. Dari elemen matriks yang berisi 0, bidak dapat bergerak ke bawah jika elemen matriks di bawahnya berisi 0; atau berpindah horizontal (kiri atau kanan) jika elemen di bawahnya berisi 1. Bila bidak berada pada elemen yang berisi 1, ia tidak dapat bergerak kemanapun. Tujuan permainan ini adalah mencapai elemen matriks yang mengandung 0 pada baris paling bawah.

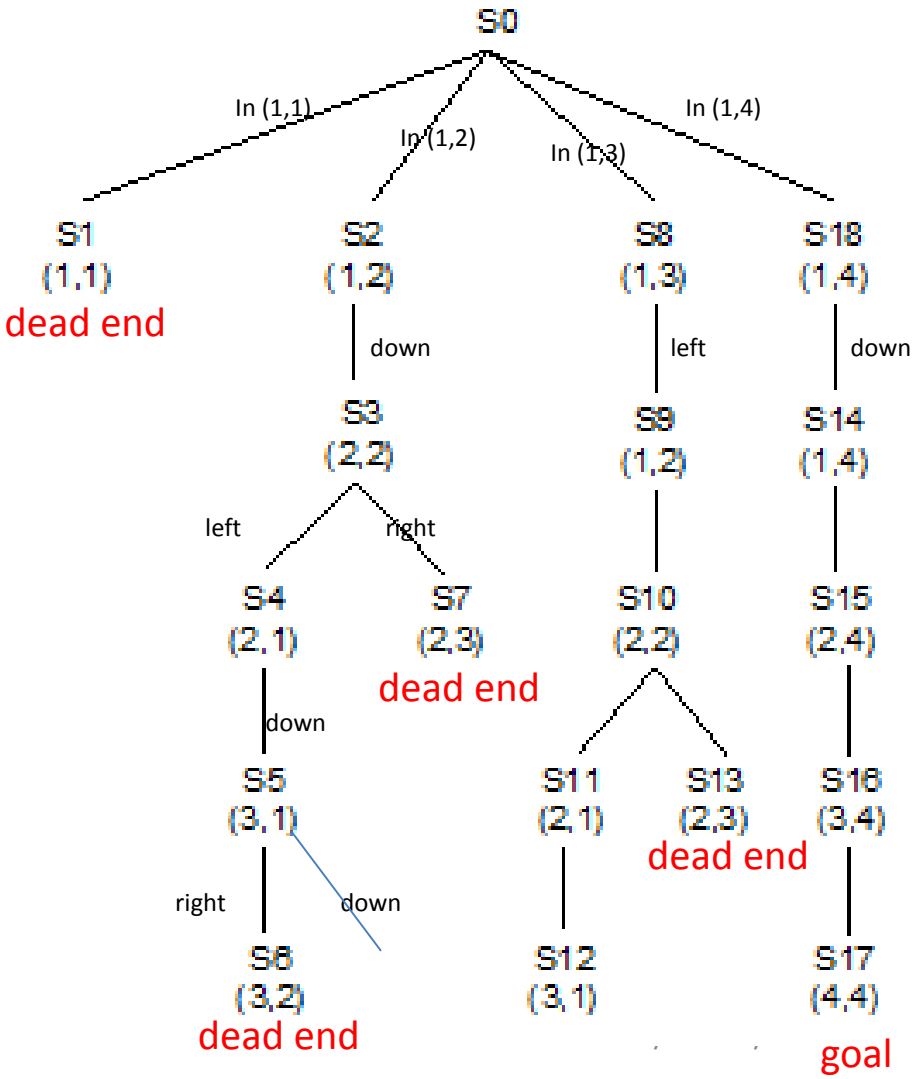
	1	2	3	4
1	1	0	0	0
2	0	0	1	0
3	0	1	0	0
4	1	0	0	0

Gambar 6.11 Matriks bidak

DOWN pindahkan bidak satu posisi ke bawah
LEFT pindahkan bidak satu posisi ke kiri
RIGHT pindahkan bidak satu posisi ke kanan

Batas kedalaman maksimum pohon ruang status diandaikan 5.

Pohon Ruang Status

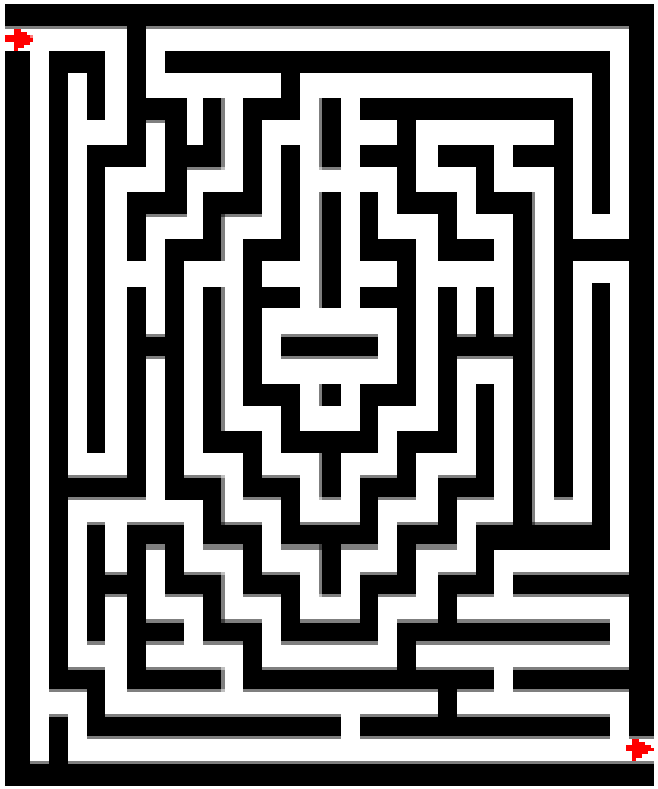


	1	2	3	4
1	1	0	0	0
2	0	0	1	0
3	0	1	0	0
4	1	0	0	0

Gambar 6.11 Matriks bidak

Maze Problem

Diberikan sebuah labirin (*maze*), temukan lintasan dari titik awal sampai titik akhir

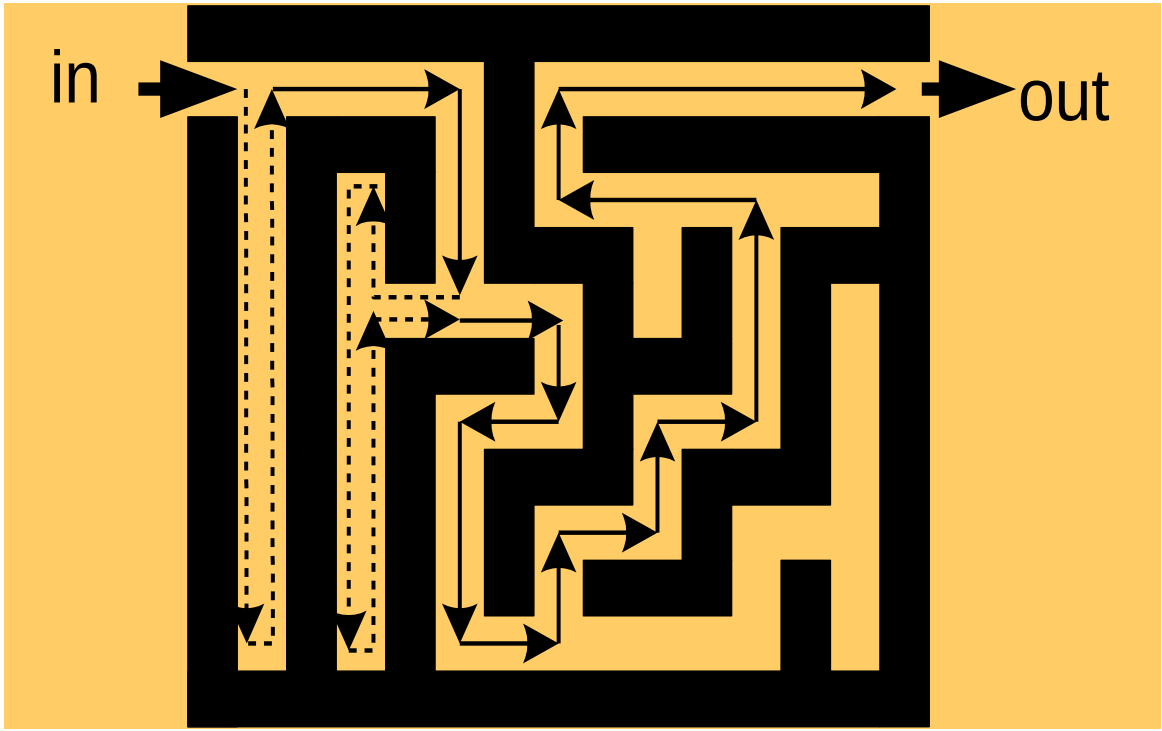


- Pada tiap perpotongan, anda harus memutuskan satu diantara tiga pilihan:
  - Maju terus
  - Belok kiri
  - Belok kanan

- Anda tidak punya cukup informasi untuk memilih pilihan yang benar (yang mengarah ke titik akhir)

- Tiap pilihan mengarah ke sekumpulan pilihan lain

- Satu atau lebih sekuens pilihan mengarah ke solusi.



Contoh runut-balik pada sebuah labirin. Runut-balik diperlihatkan dengan garis putus-putus.

Penyelesaian dengan DFS:

- Bagi lintasan menjadi sederetan langkah.
- Sebuah langkah terdiri dari pergerakan satu unit sel pada arah tertentu.
- Arah yang mungkin: lurus (*straight*), kiri (*left*), ke kanan (*right*).

Garis besar algoritma DFS:

```
while belum sampai pada tujuan do
  if terdapat arah yang benar sedemikian sehingga kita belum pernah
     berpindah ke sel pada arah tersebut
  then
     pindah satu langkah ke arah tersebut
  else
     backtrack langkah sampai terdapat arah seperti yang disebutkan
     di atas
  endif
endwhile
```

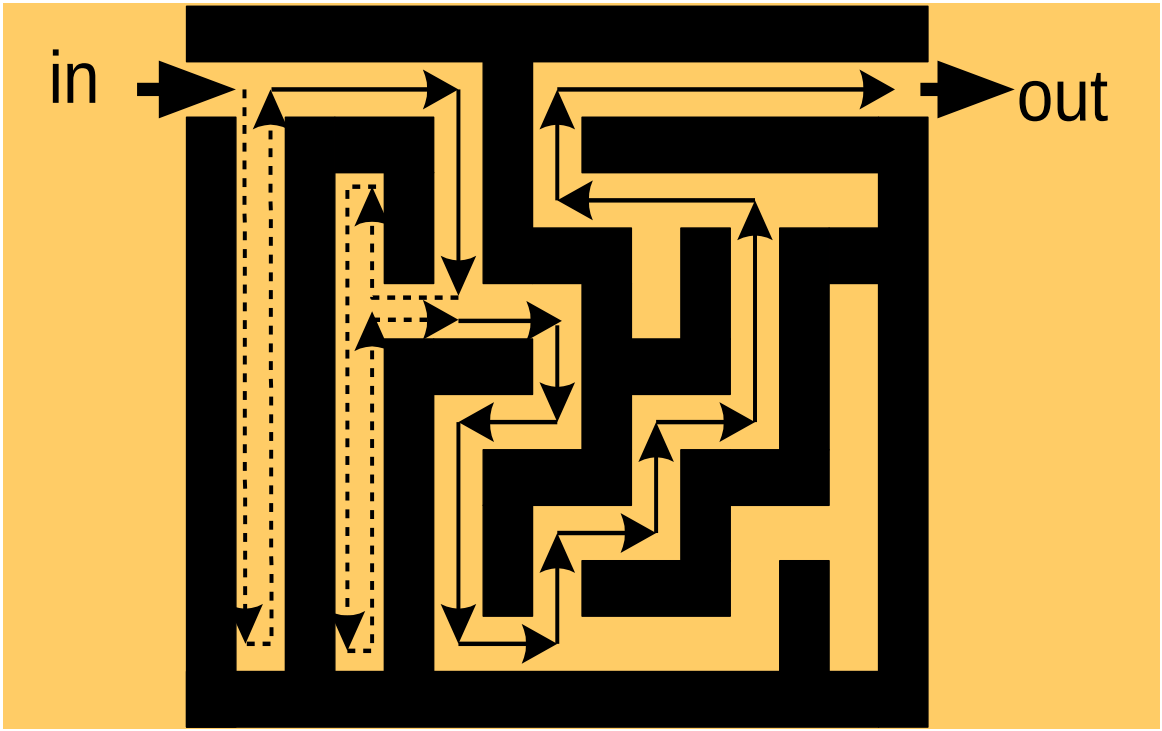
- Bagaimana mengetahui langkah yang mana yang perlu dijejaki kembali?

- Ada dua solusi untuk masalah ini:

  1. Simpan semua langkah yang pernah dilakukan, atau
  2. gunakan rekursi (yang secara implisit menyimpan semua langkah).

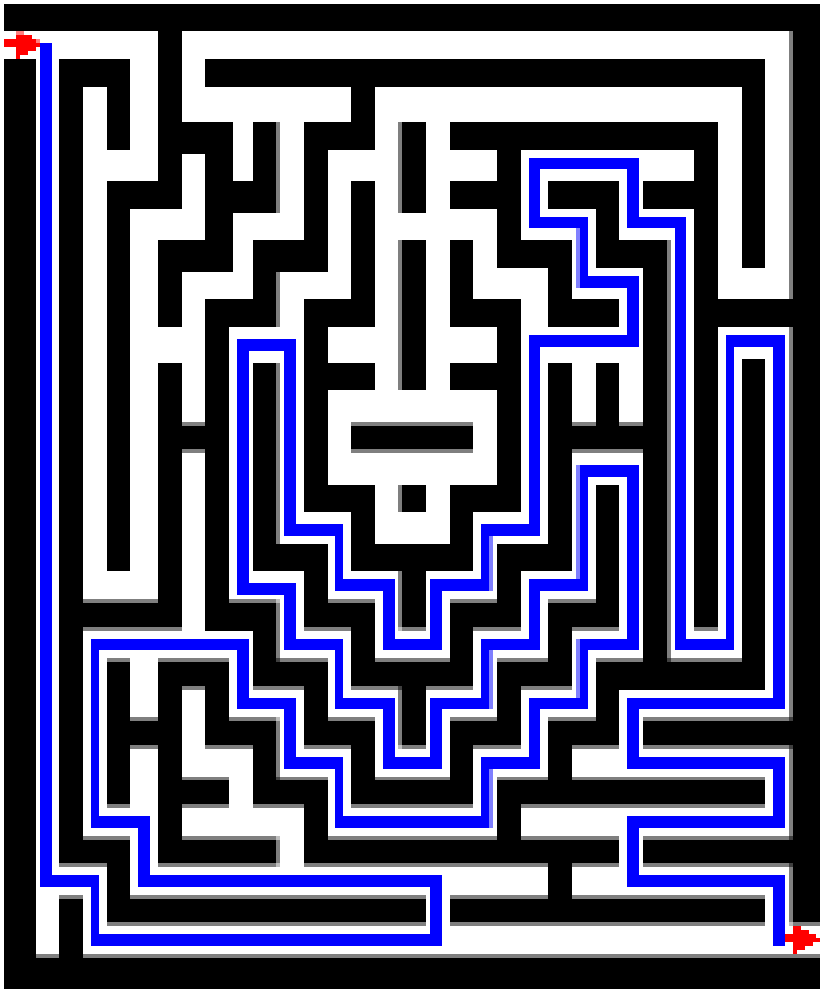
- Rekursi adalah solusi yang lebih mudah.

```
function SolveMaze(input M : labirin)→boolean
{ true jika pilihan mengarah ke solusi }

Deklarasi
   arah : integer  { up = 1, down, 2, left = 3, right = 4
}

Algoritma:
   if pilihan arah merupakan solusi then
     return true
   else
     for tiap arah gerakan (lurus, kiri, kanan) do
        move(M, arah)  { pindah satu langkah (satu sel)
                         sesuai arah  tersebut }
        if SolveMaze(M) then
          return true
        else
          unmove(M, arah)  { backtrack }
        endif
     endfor
     return false     { semua arah sudah dicoba, tetapi
                        tetap buntu,  maka
                        kesimpulannya: bukan solusi }
   endif
```



Contoh runut-balik pada sebuah labirin. Runut-balik diperlihatkan dengan garis putus-putus.

Contoh lainnya:



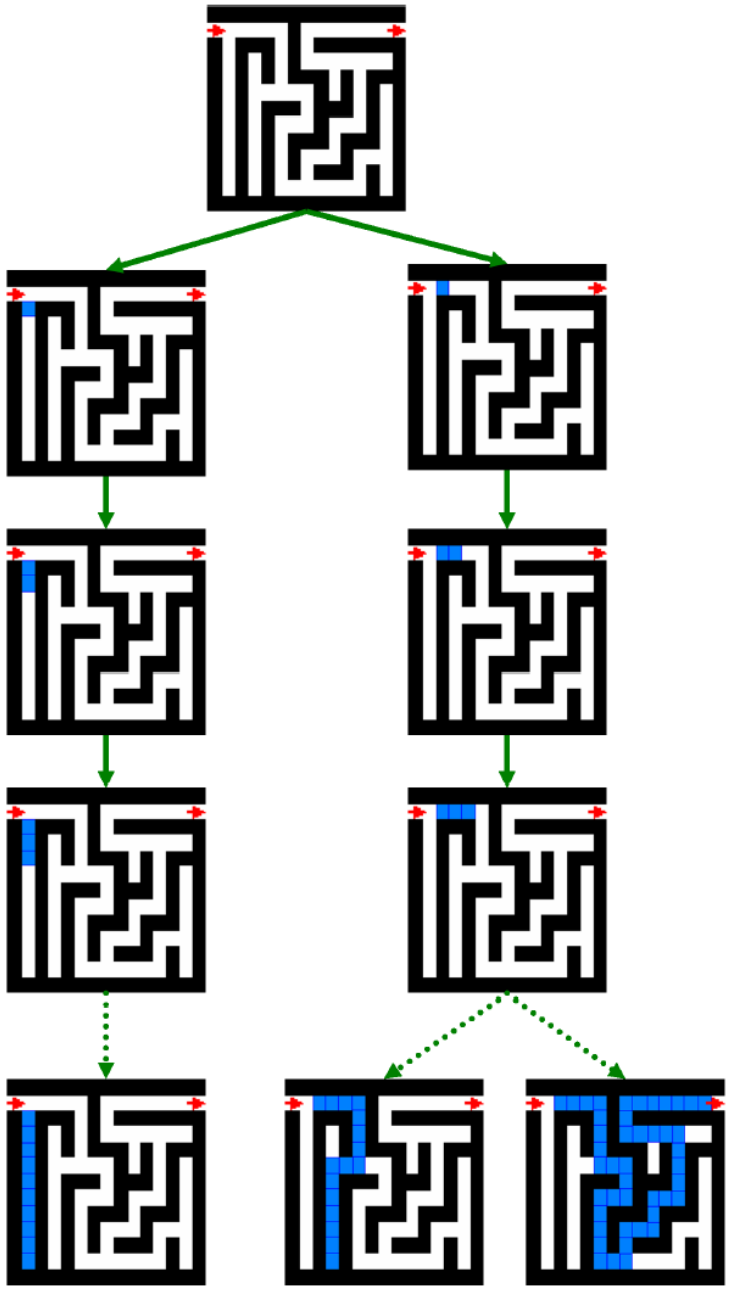
Jika kita menggambarkan sekuens pilihan yang kita lakukan, maka diagram berbentuk seperti pohon.

Simpul daun merupakan:

1. Titik *backtrack*, atau
2. Simpul *goal*

Pada titik *backtrack*, simpul tersebut menjadi mati (tidak bisa diekspansi lagi)

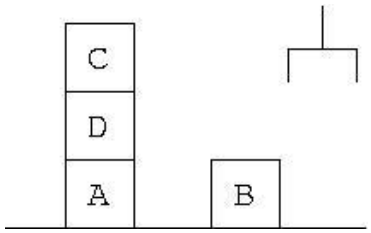
Aturan pembentukan simpul: DFS



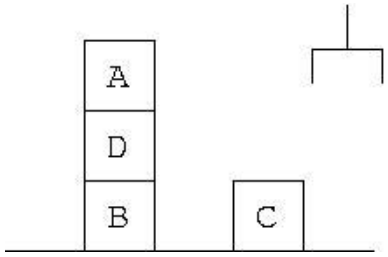
Block World Problem

(Soal UTS 2018)

- 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

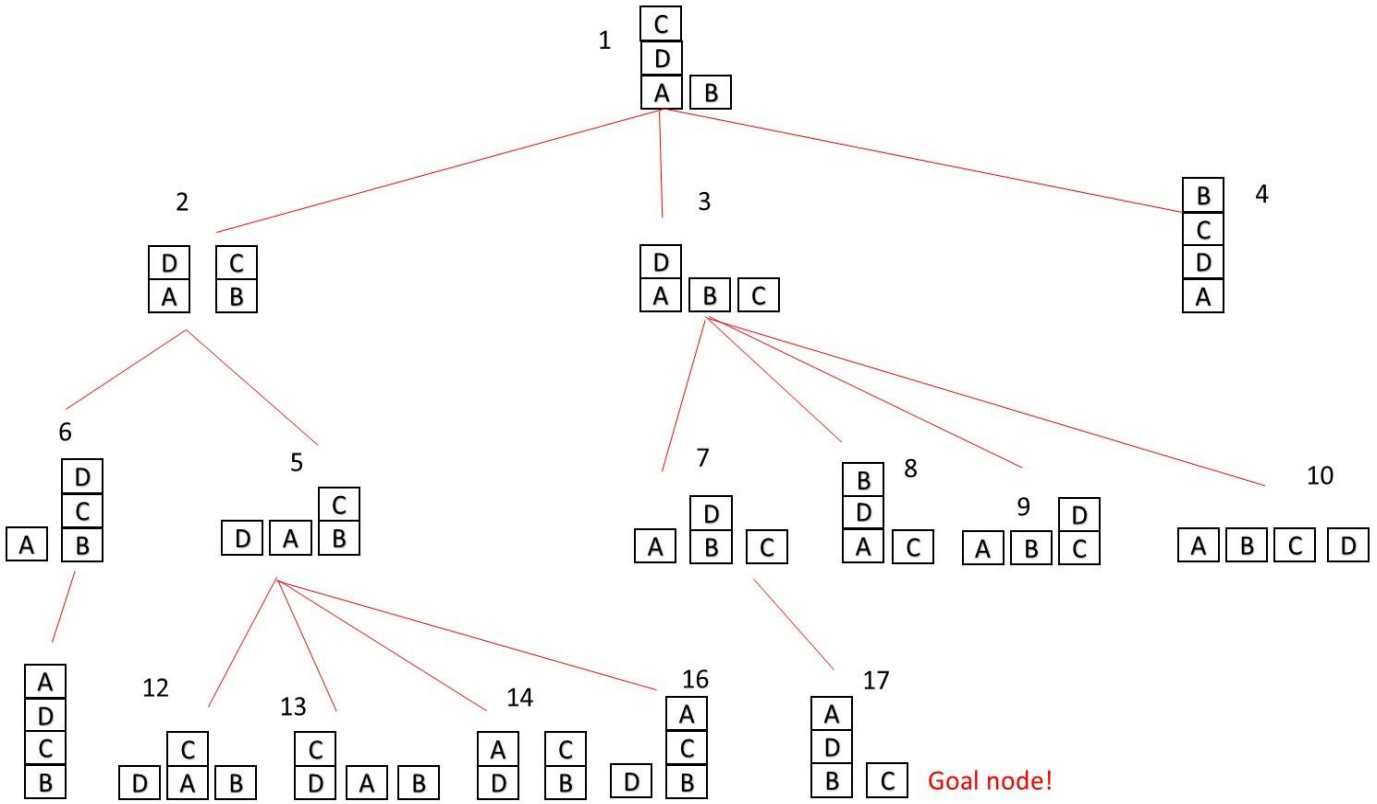


Goal state

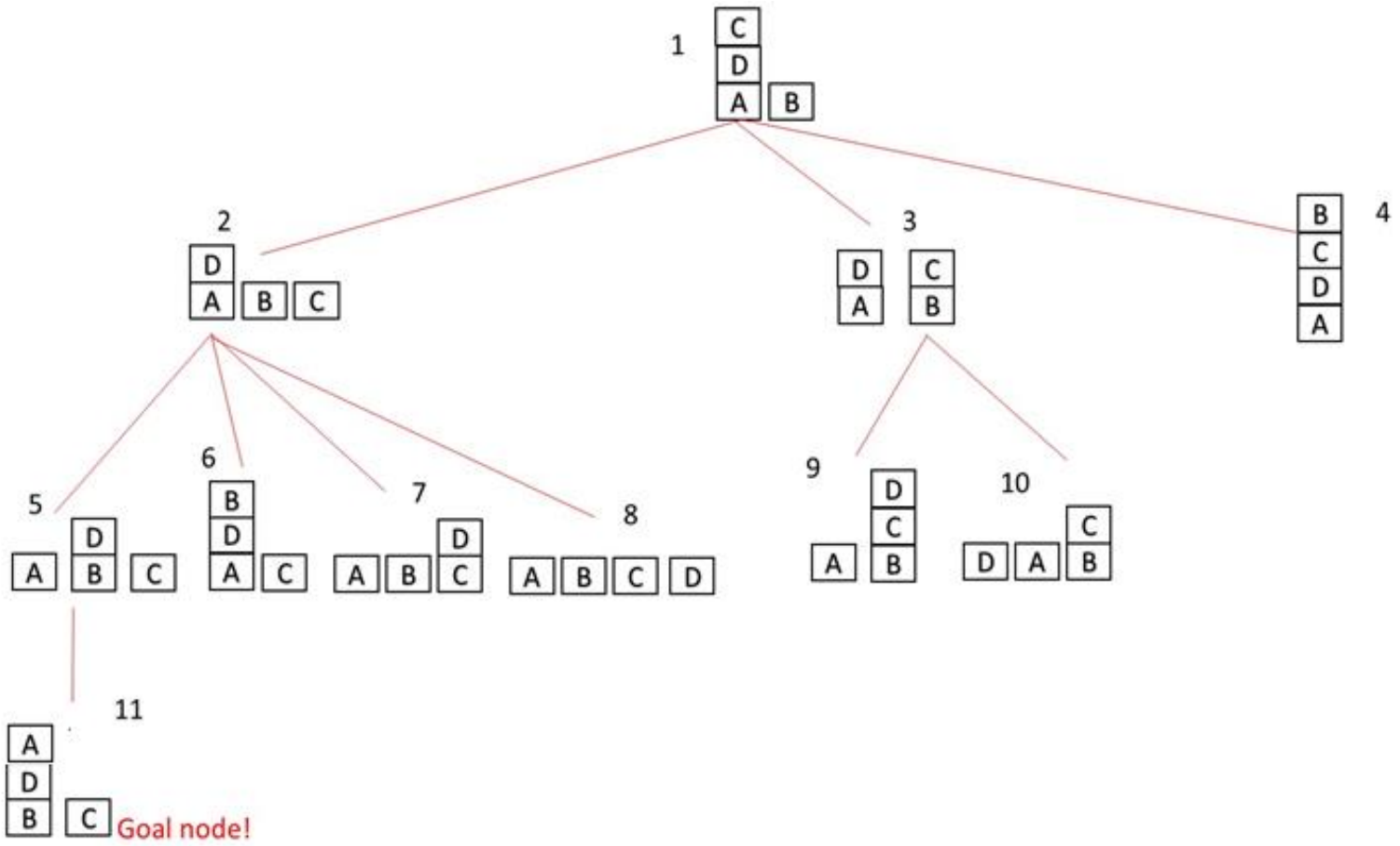
Penyelesaian:

(a) BFS

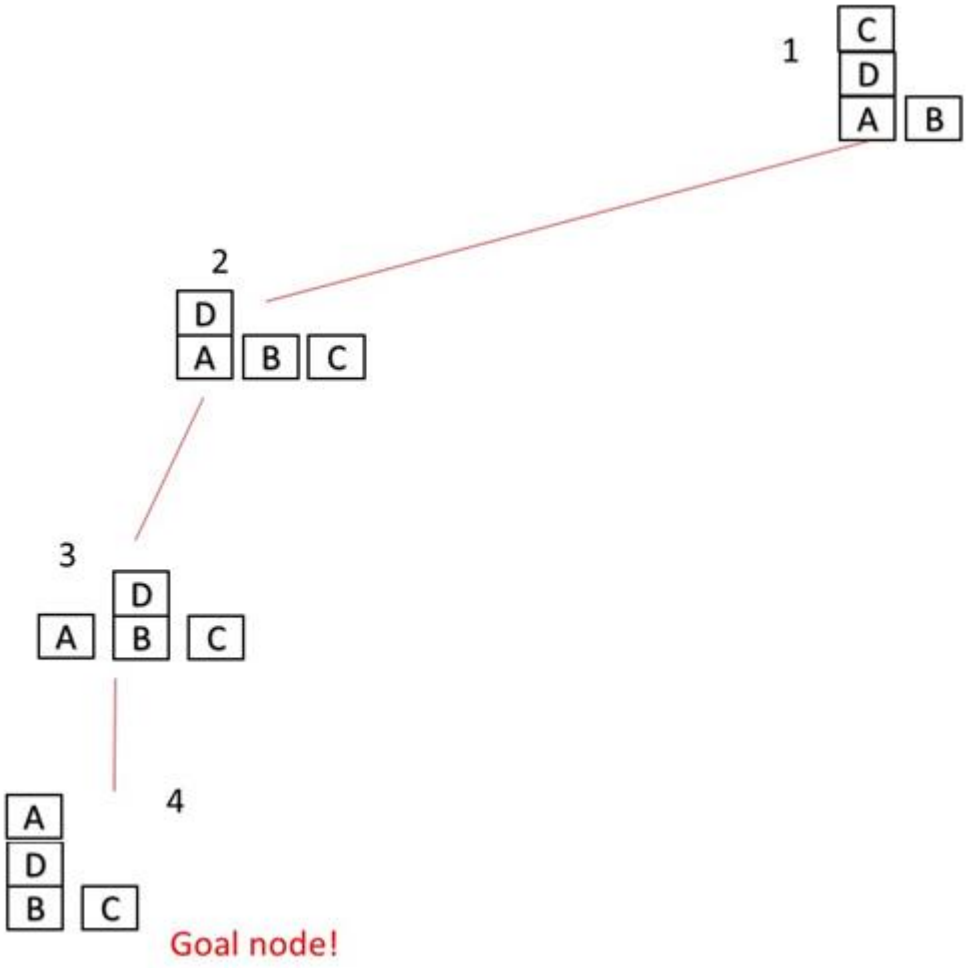
Salah satu kemungkinan solusi:



Kemungkinan lain:

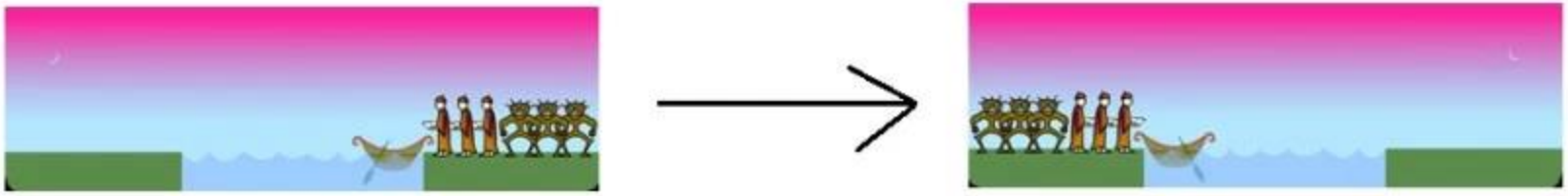


(b) DFS



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



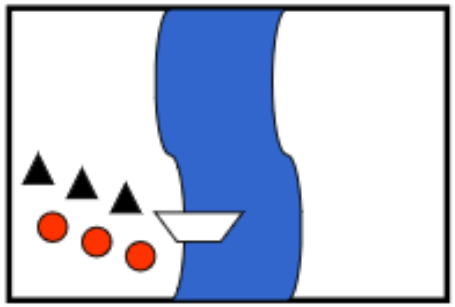
Initial State

Goal State

- Bagaimana menyeberangkan keenam orang tadi sehingga semuanya selamat sampai di sisi sungai seberangnyya? Selesaikan dengan BFS dan DFS

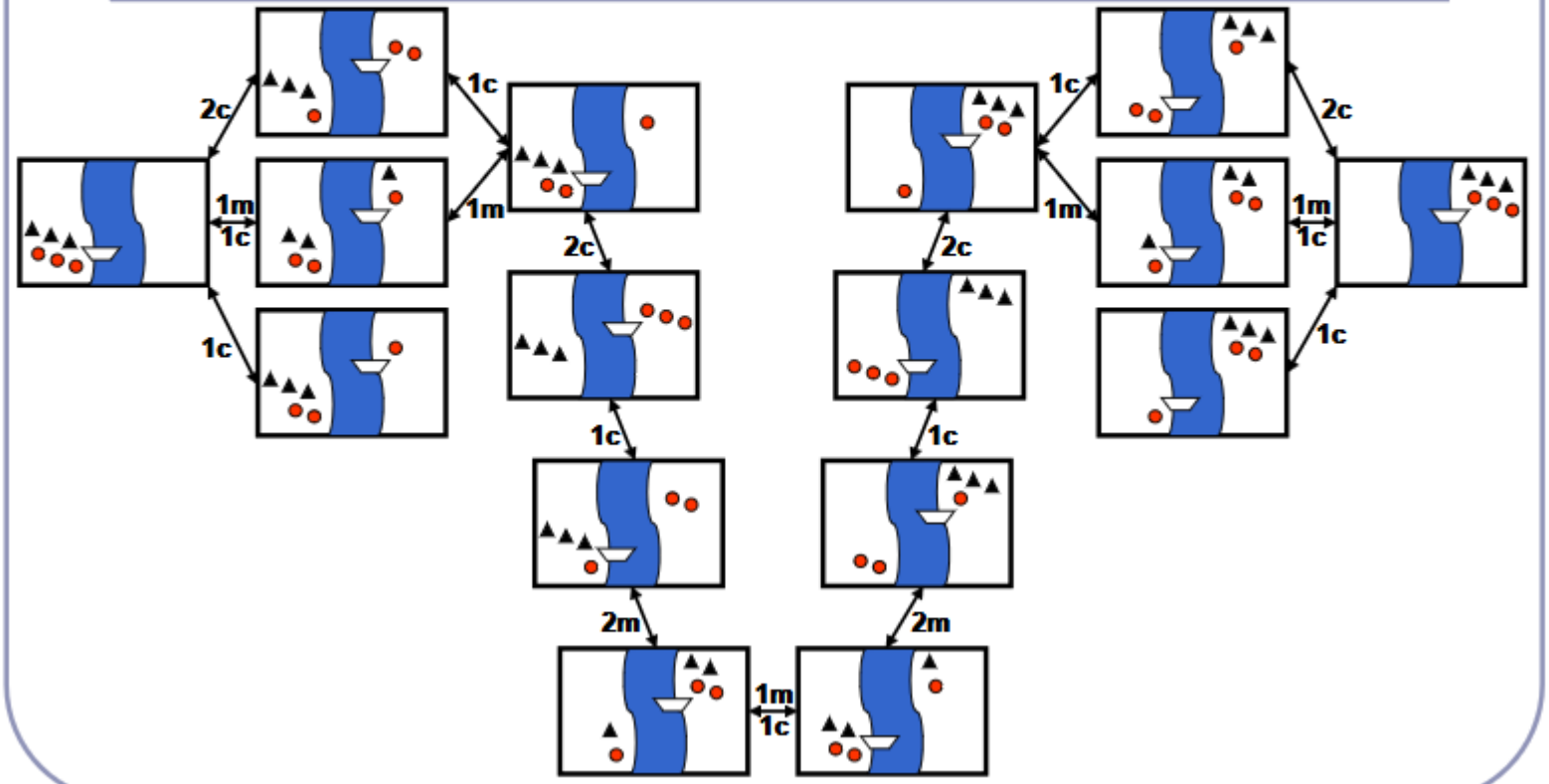
Missionaries and Cannibals: Initial State and Actions

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

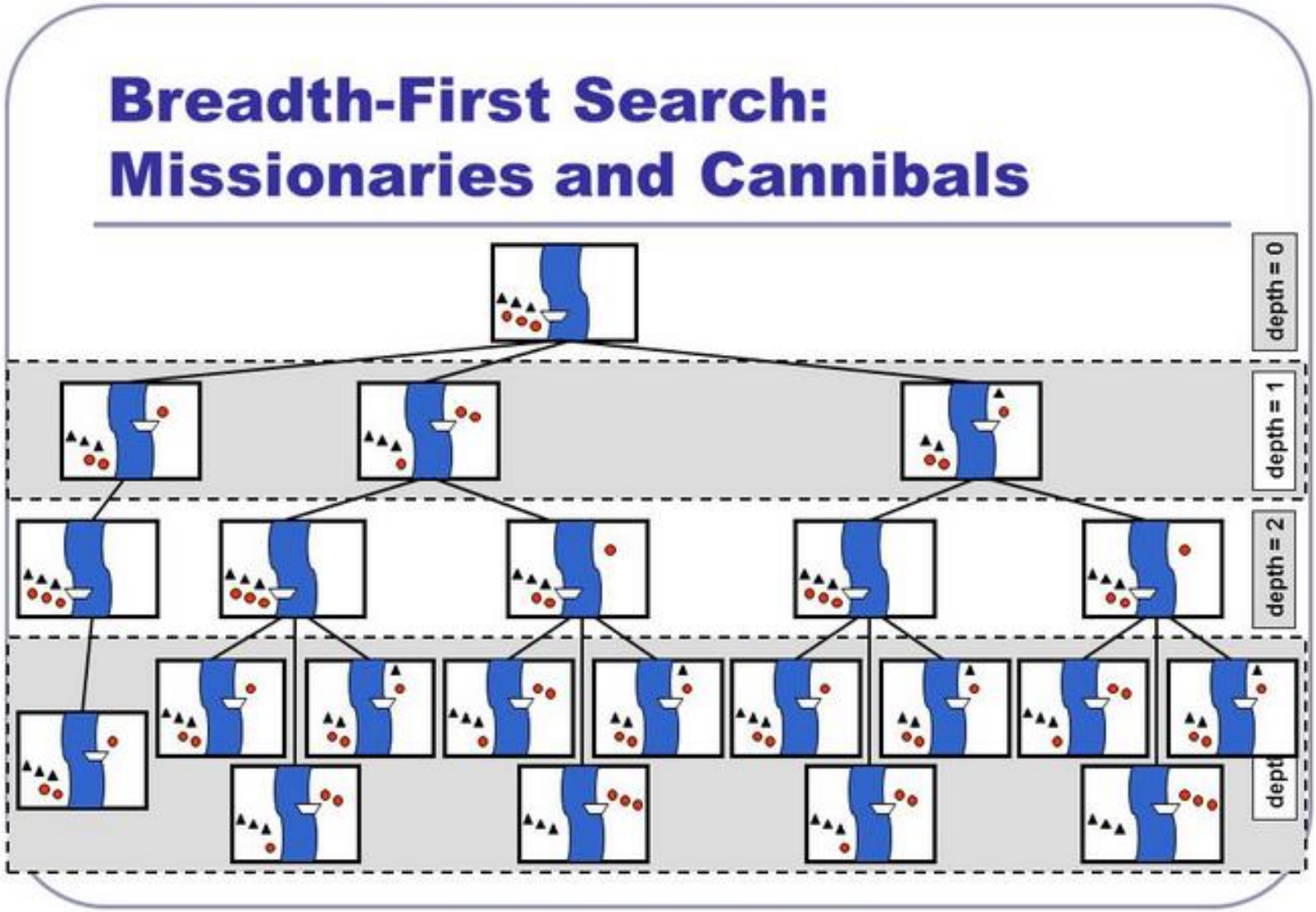


- 5 possible actions:
  - one missionary crossing
  - one cannibal crossing
  - two missionaries crossing
  - two cannibals crossing
  - one missionary and one cannibal crossing

Missionaries and Cannibals: State Space



Sebagian pohon ruang status dengan BFS



Algoritma Pencarian Lainnya

- Depth-limited search
- Iterative deepening search

Depth-Limited Search

- BFS: dijamin menemukan path dgn langkah minimum tapi membutuhkan ruang status yang besar
- DFS: efisien, tetapi tidak ada jaminan solusi dgn langkah minimum
  - DFS dapat memilih langkah yang salah, sehingga path panjang bahkan infinite. Pemilihan langkah sangat penting
- Salah satu solusi: DFS-limited search
  - DFS dengan pembatasan kedalaman sampai l
  - Simpul pada level l dianggap tidak memiliki successor
  - Masalah: penentuan batas level ($\geq$ shallowest goal)

DLS Algorithm

```
Function DLS (problem, limit)
→ rec_DLS(make_node(init_state),problem,limit)

Function Rec_DLS (node,problem, limit)
  if isGoal(node) then → solution(node)
  else if depth(node)=limit then → cutoff
  else
      for each successor in Expand(node,problem) do
        result ←rec_DLS(successor,problem,limit)
        if result=cutoff then cutoff_occured← true
        else if result≠failure then → result
  if cutoff_occured then → cutoff
  else → failure
```

Bagaimana property dari DLS?

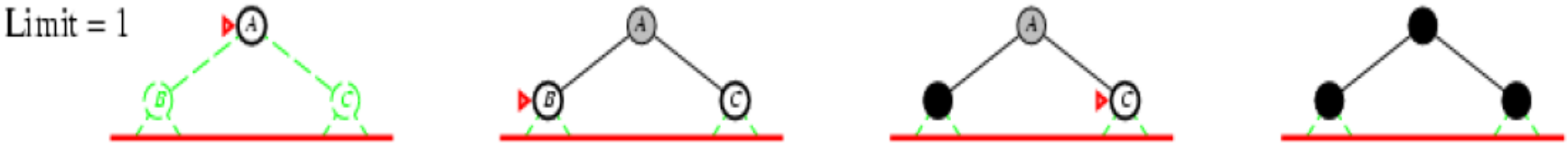
- *Completeness*?
  - Tidak
- *Optimality*?
  - Tidak
- Kompleksitas waktu:
  - $O(b^l)$
- Kompleksitas ruang:
  - $O(bl)$

Iterative Deepening Search (IDS)

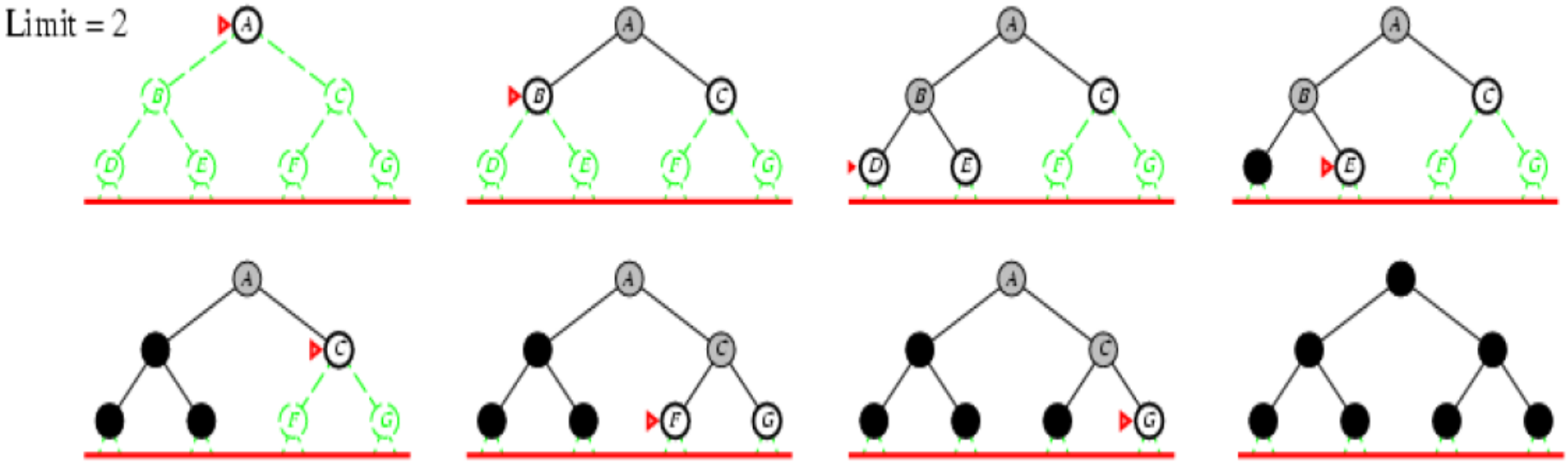
- IDS: melakukan serangkaian DFS, dengan peningkatan nilai kedalaman-cutoff, sampai solusi ditemukan
- Asumsi: simpul sebagian besar ada di level bawah, sehingga tidak menjadi persoalan ketika simpul pada level-level atas dibangkitkan berulang kali

```
Depth ← 0
Iterate
  result← DLS(problem,depth)
stop: result ≠ cutoff
  depth← depth+1
→ result
```

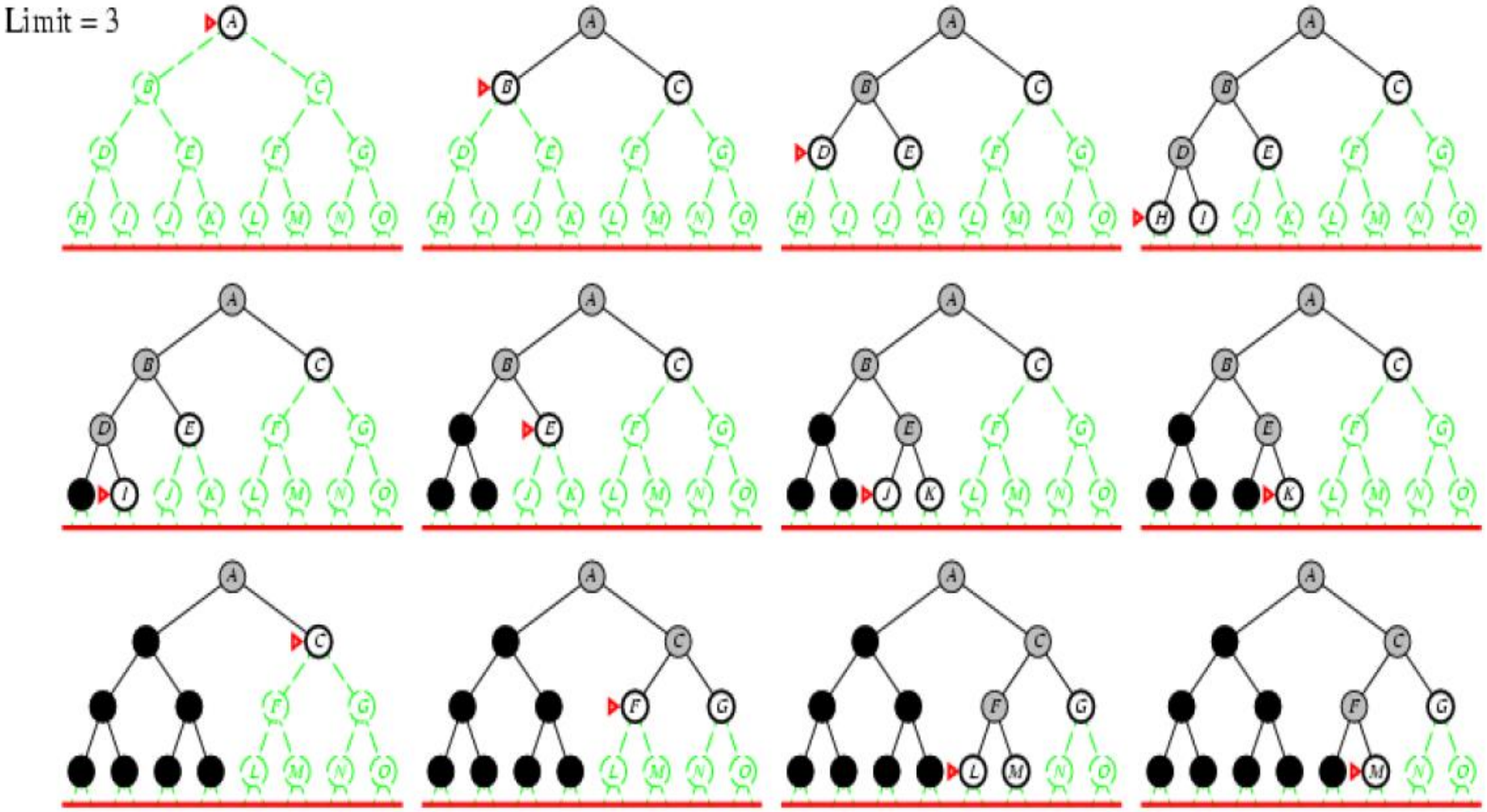
IDS dengan d=1



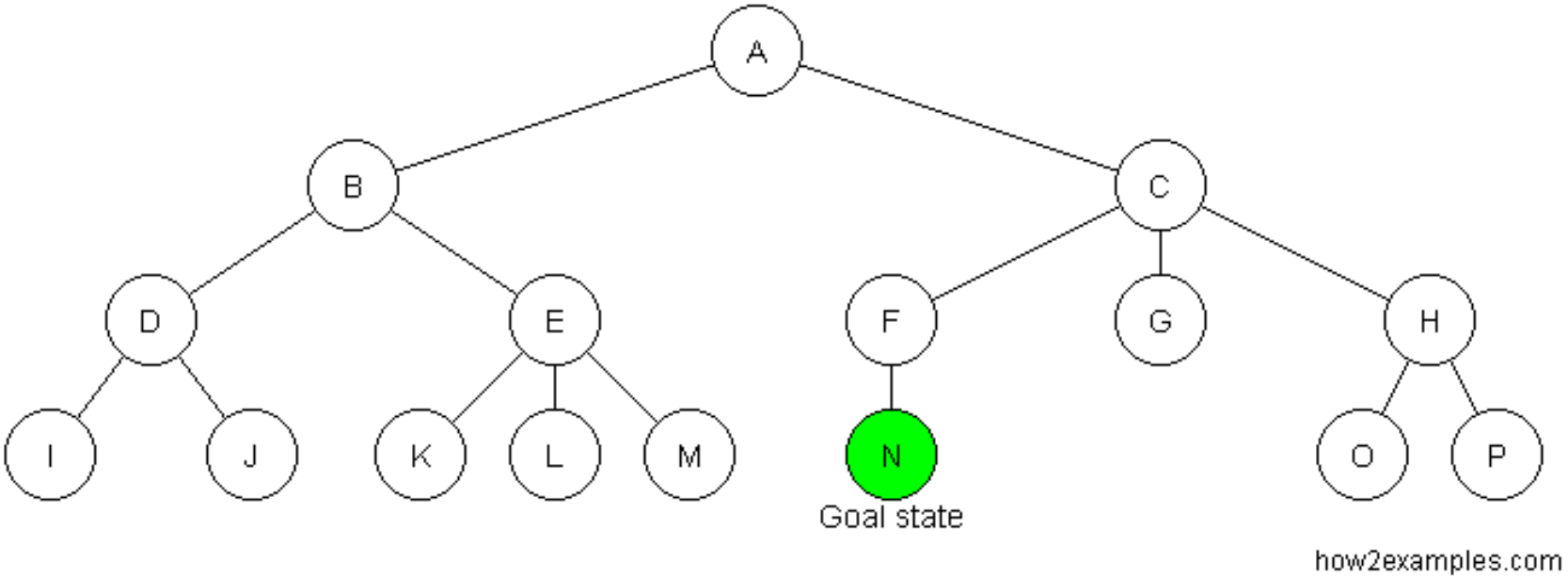
IDS dengan d=2



IDS dengan d=3



IDS (animasi)



Bagaimana property dari IDS?

- Completeness?
  - Ya, jika b terbatas
- Optimality?
  - Ya, jika langkah = biaya
- Kompleksitas waktu:
  - $O(b^d)$
- Kompleksitas ruang:
  - O(bd)

Route/Path Planning

Materi Kuliah IF2211 – Strategi Algoritma

Teknik Informatika - ITB

Referensi

- Materi kuliah IF3170 Inteligensi Buatan Teknik Informatika ITB, Course Website:

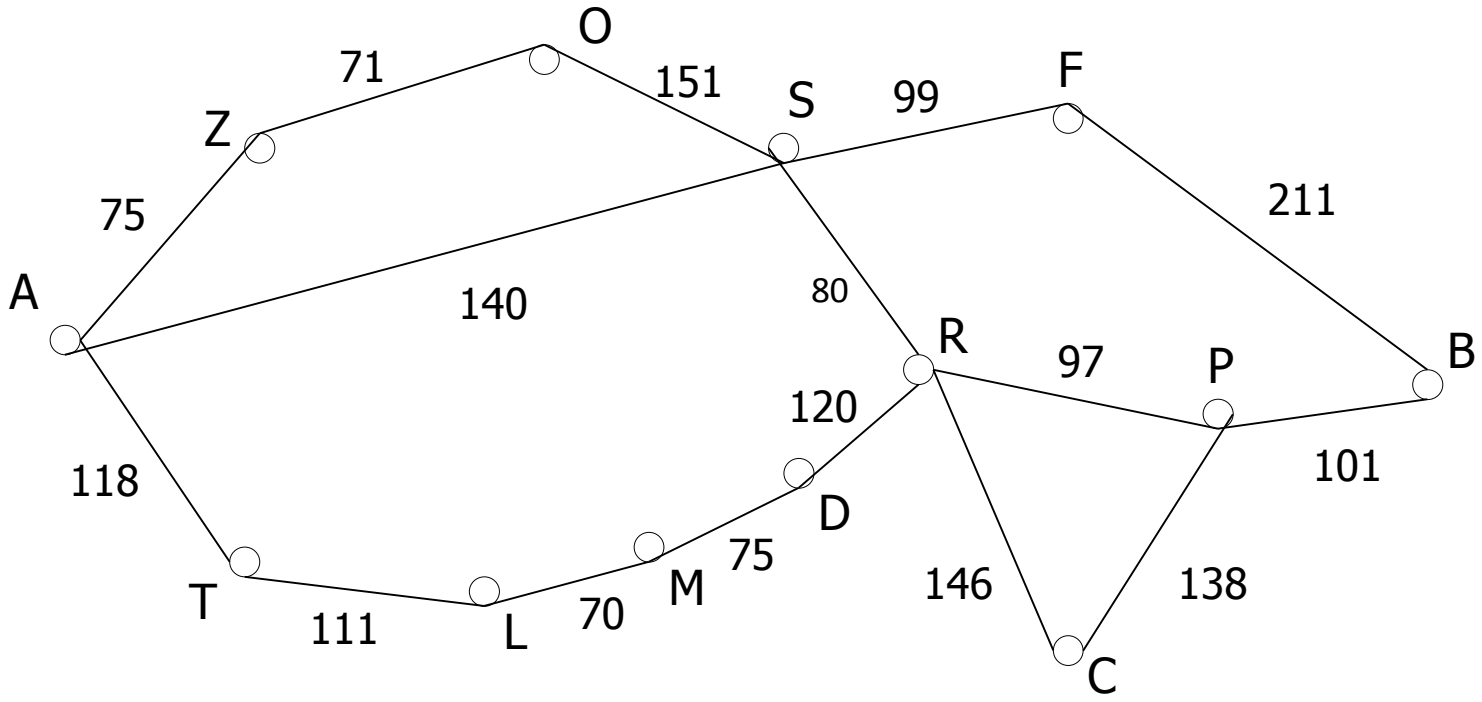
  http://kuliah.itb.ac.id → STEI → Teknik Informatika → IF3170
- Stuart J Russell & Peter Norvig, Artificial Intelligence: A Modern Approach, 3rd Edition, Prentice-Hall International, Inc, 2010, Textbook

  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/

Route Planning

Search

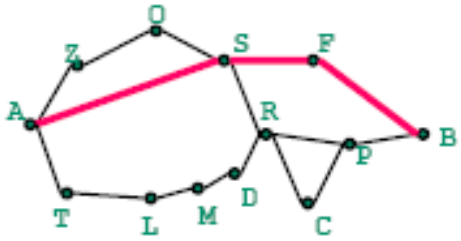
Source: Russell's book

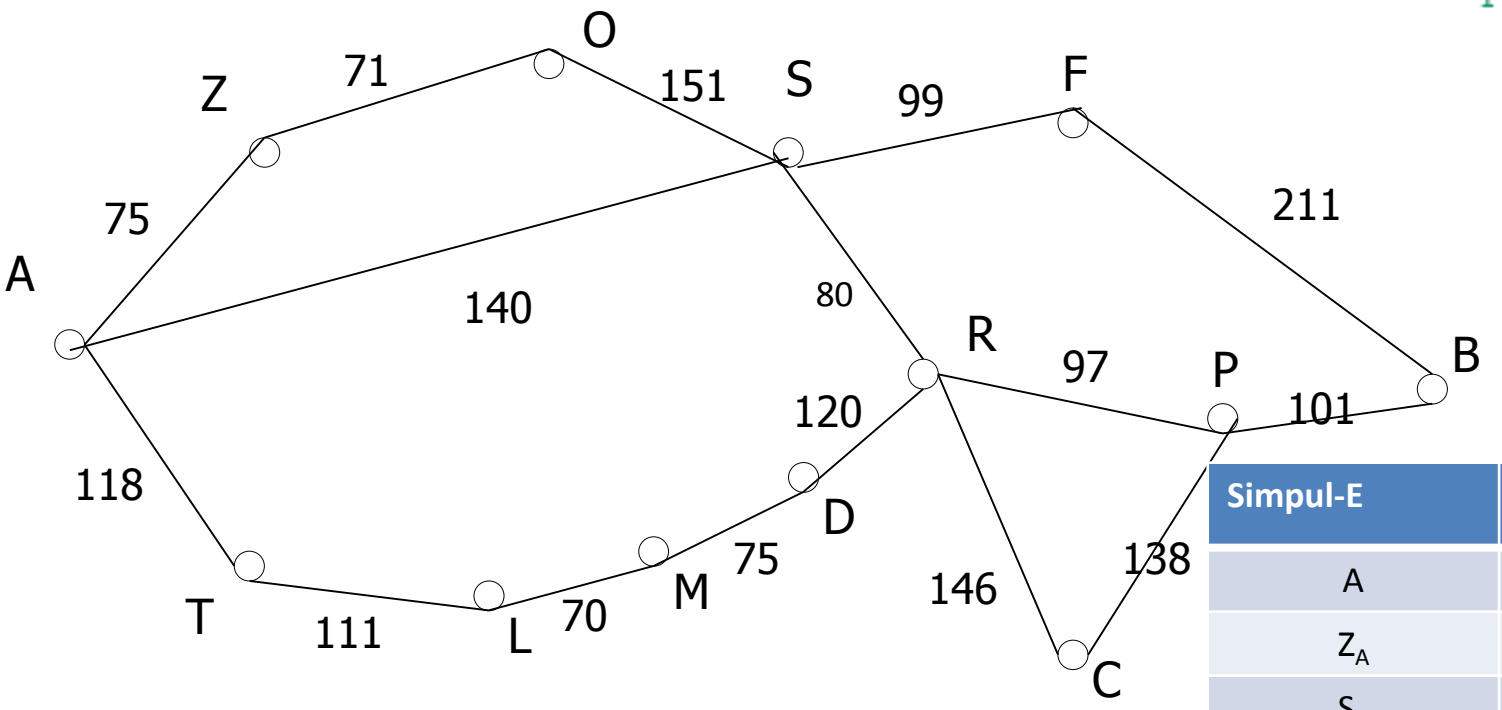


S: set of cities
i.s: A (Arad)
g.s: B (Bucharest)
Goal test: s = B ?
Path cost: time ~ distance

Breadth-First Search (BFS)

Treat agenda as a queue (FIFO)



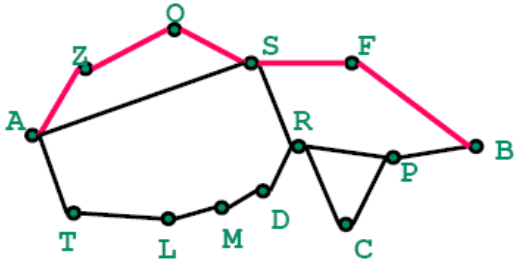


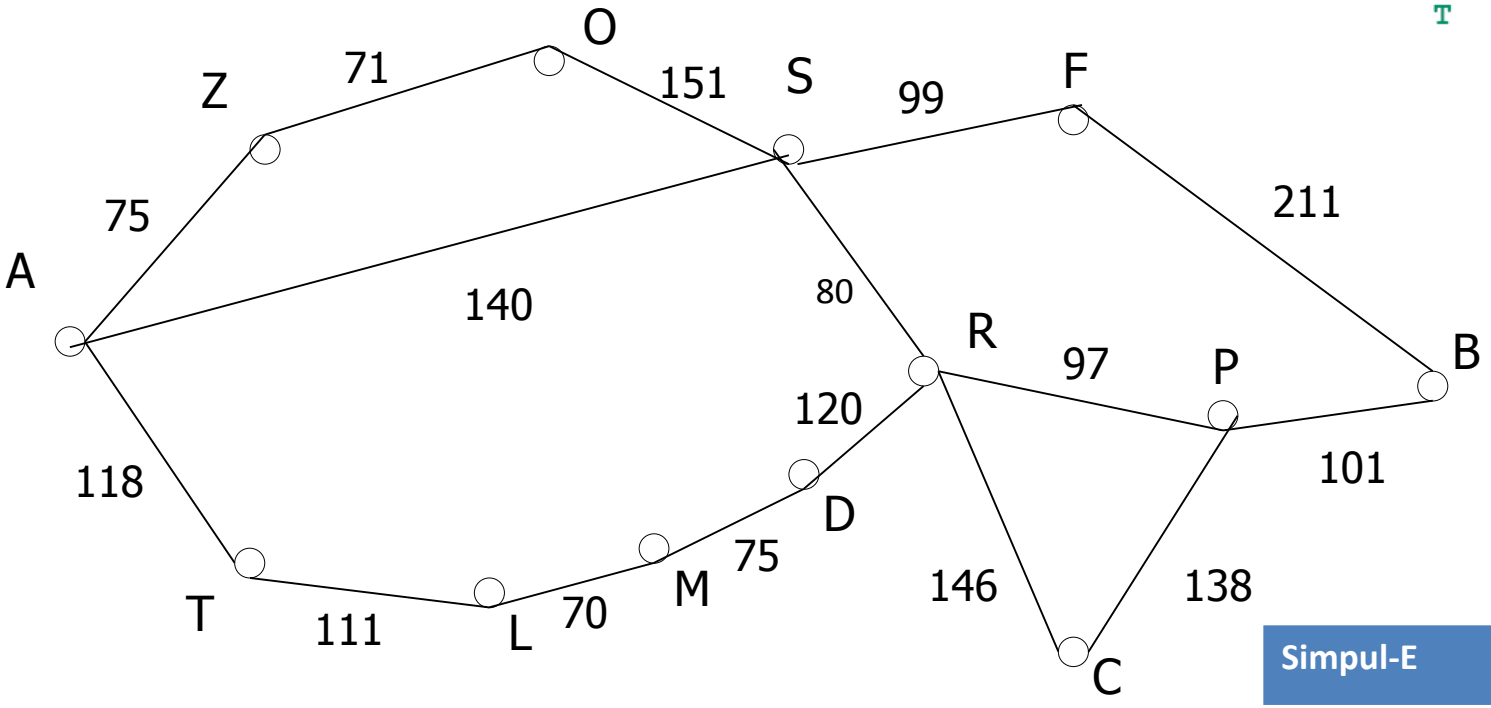
Path: A→ S → F → B,
Path-cost = 450

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)



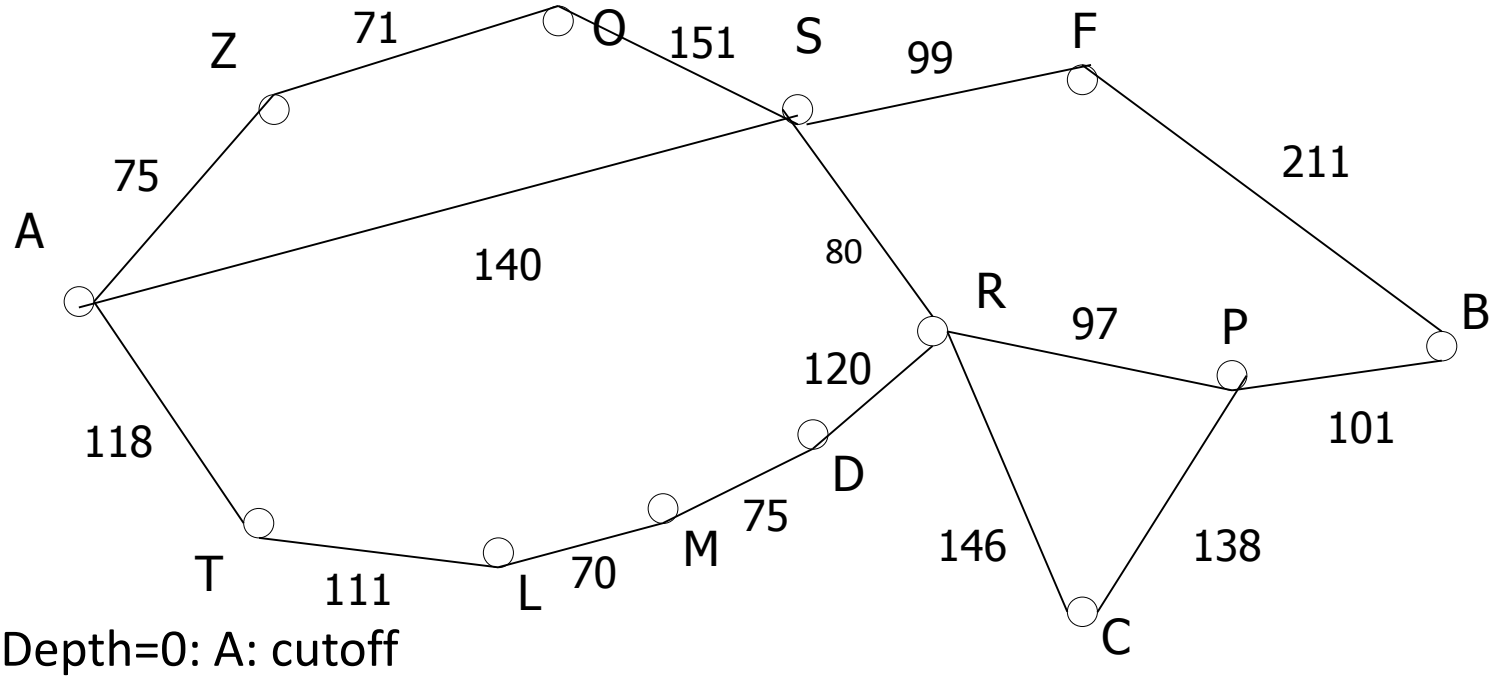


Path: A→ Z → O → S → F → B

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

IDS



Depth=0: A: cutoff

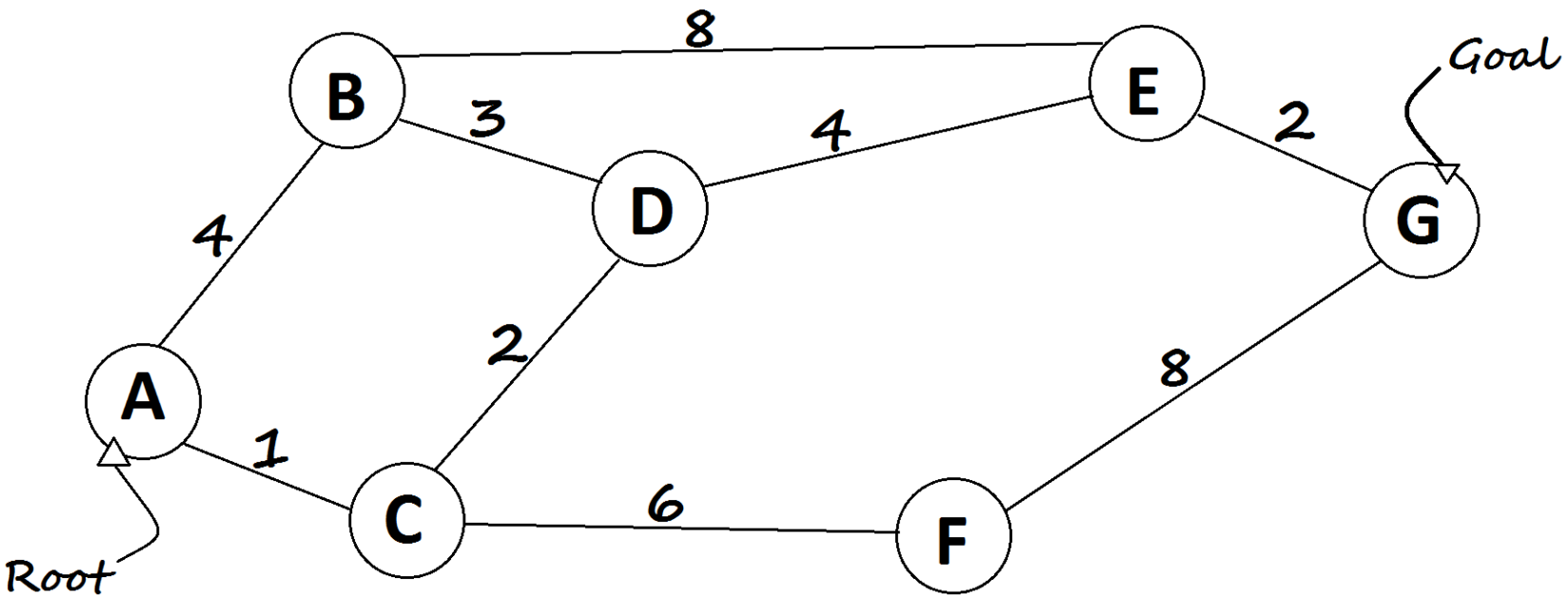
Depth=1: A → Z_A,S_A,T_A → Z_A: cutoff, S_A: cutoff, T_A: cutoff

Depth=2: A → Z_A,S_A,T_A → O_{AZ}, S_A,T_A → O_{AZ} : cutoff → F_{AS}, R_{AS},T_A → F_{AS} : cutoff → R_{AS} : cutoff → L_{AT} → L_{AT} : cutoff

Depth=3: A → Z_A,S_A,T_A → O_{AZ}, S_A,T_A → S_{AZO},S_A,T_A → S_{AZO}: cutoff → F_{AS}, R_{AS},T_A → B_{ASF}, R_{AS},T_A → B_{ASF}

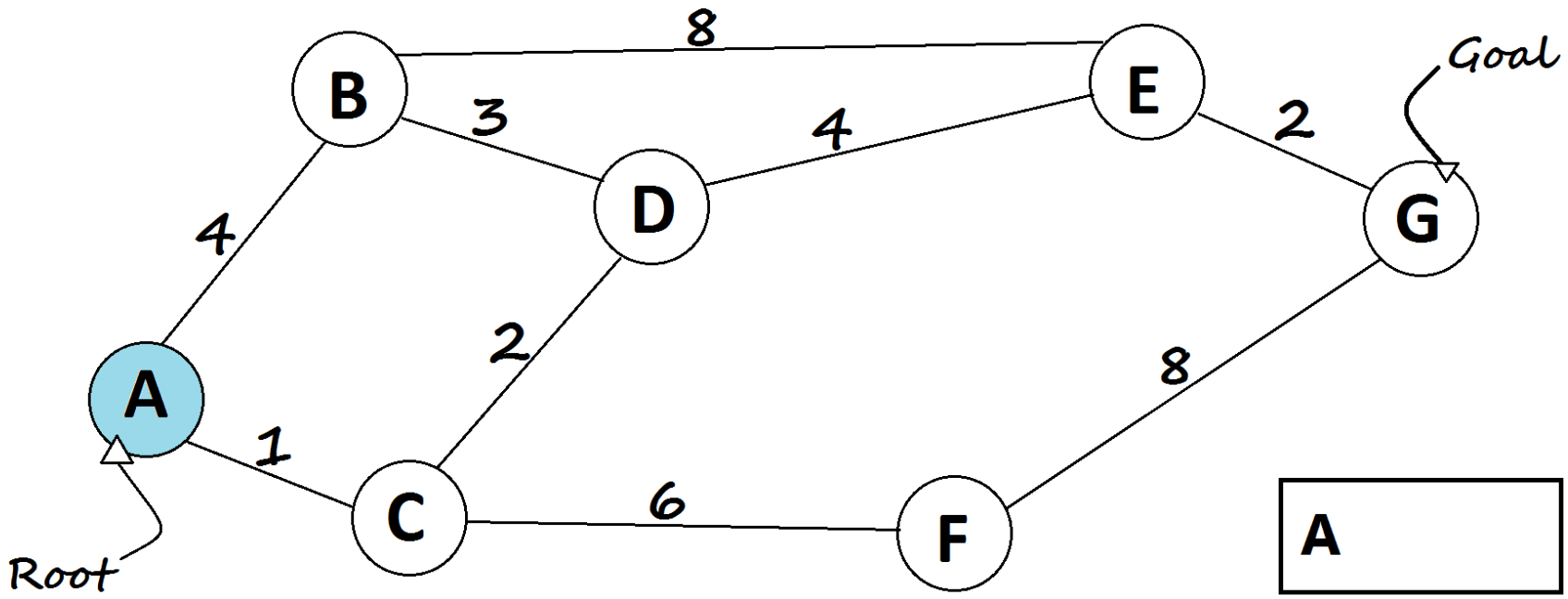
Stop: B=goal, path: A→ S → F → B, path-cost = 450

Contoh *path finding* lainnya:



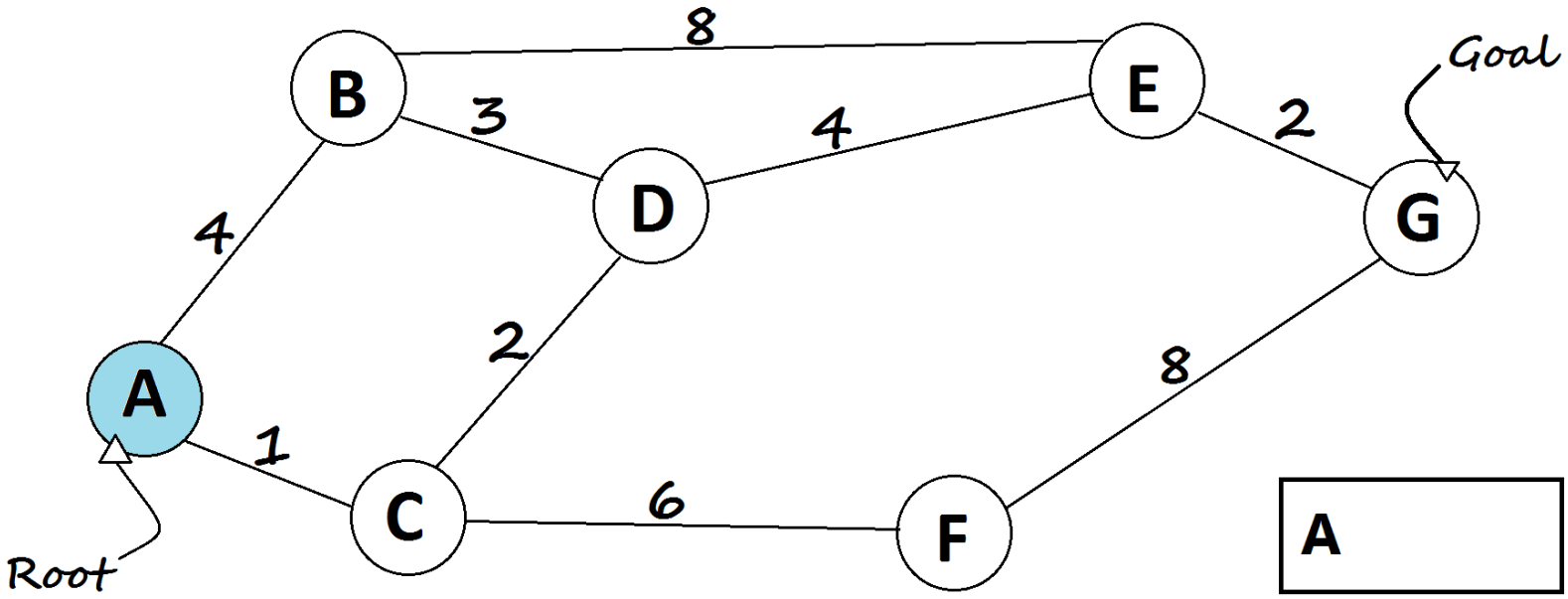
Sumber:https: //medium.com/omarelgabrys-blog/path-finding-algorithms-f65a8902eb40

BFS:



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

DFS:



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

SELAMAT BELAJAR