

# Breadth/Depth First Search (BFS/DFS)

Bahan Kuliah IF2211 Strategi Algoritmik

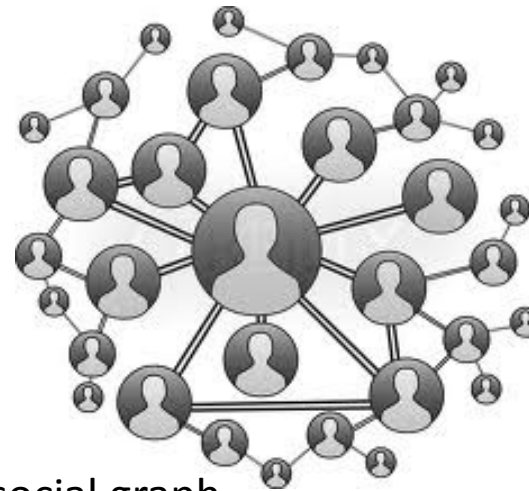
Oleh: Rinaldi Munir

Update: Masayu Leylia Khodra

22 September 2013

# Traversals Graf

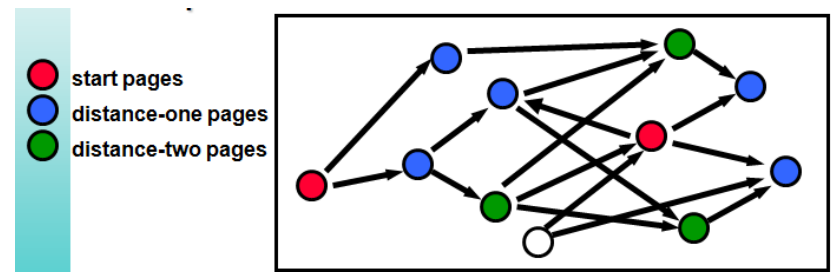
- Algoritma traversal graf: mengunjungi simpul dengan cara yang sistematis
  - Pencarian melebar (breadth first search/BFS)
  - Pencarian mendalam (depth first search/DFS)
  - Asumsi: graf terhubung
- Graf: representasi persoalan →  
Traversals graf: pencarian solusi



social graph

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

Web page network

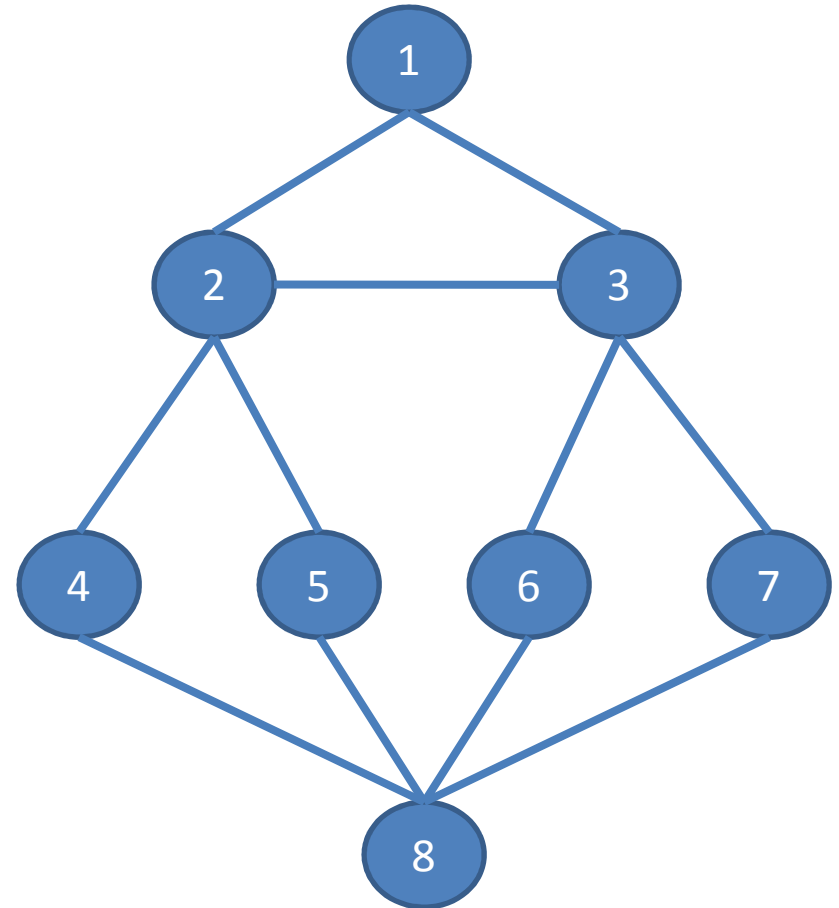


# Algoritma Pencarian

- 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\*

# 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  $n \times n$ ,  
 $a_{ij} = 1$ , 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

```

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:**

```

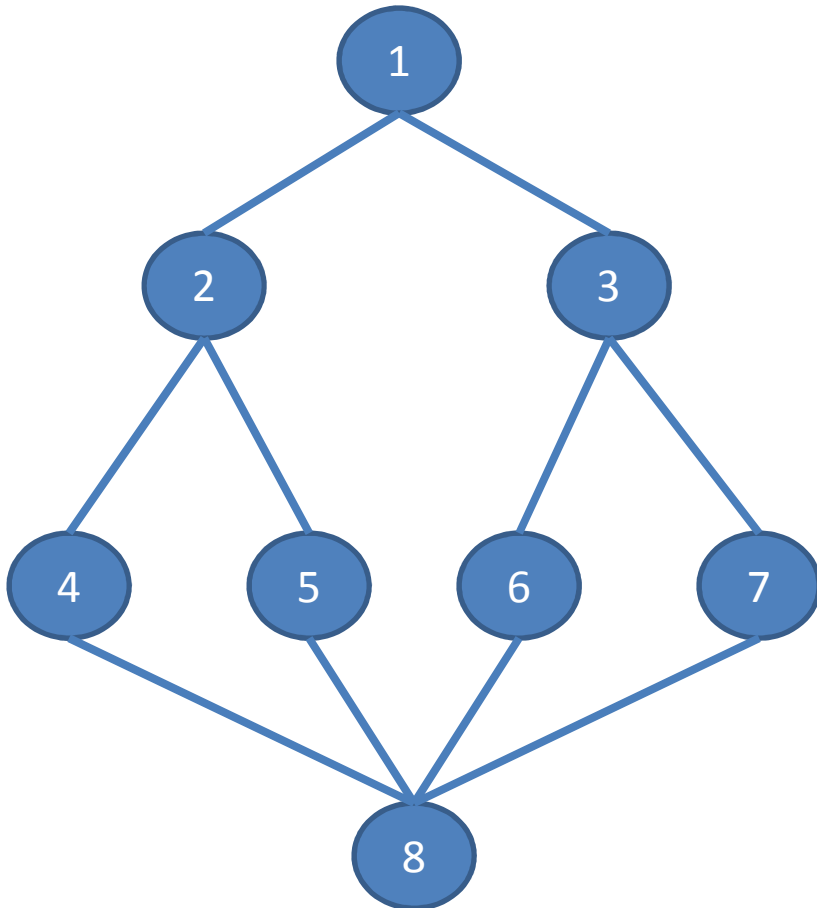
  BuatAntrian(q)      { buat antrian kosong }

  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) }

```

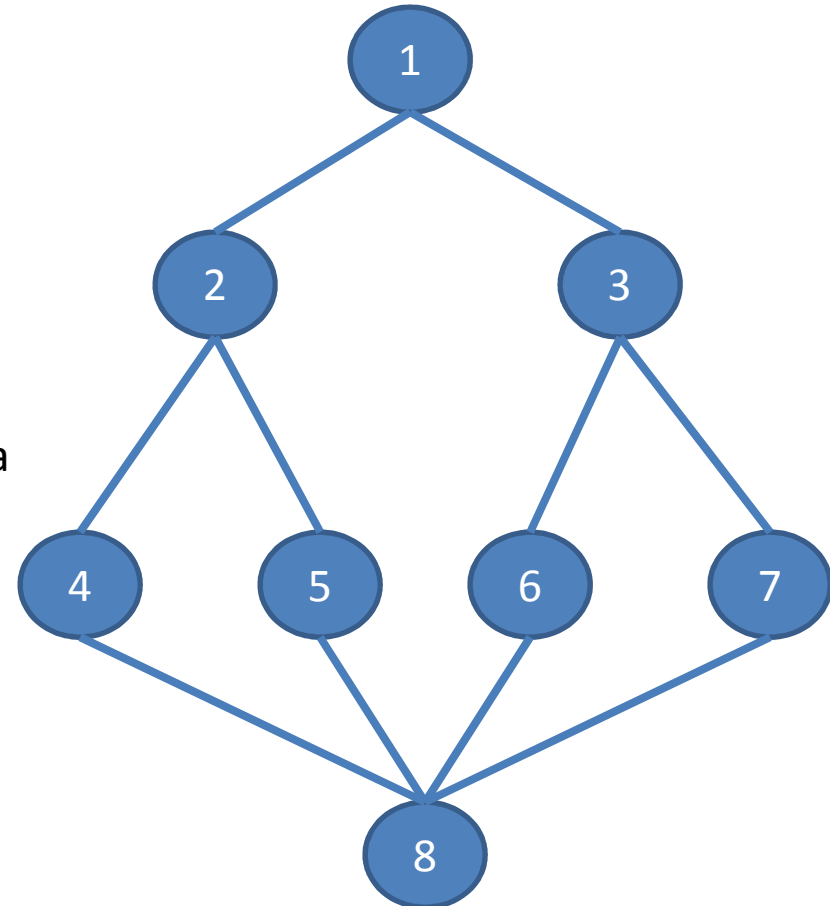
# BFS: Ilustrasi



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

# 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 simpul yang 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
```

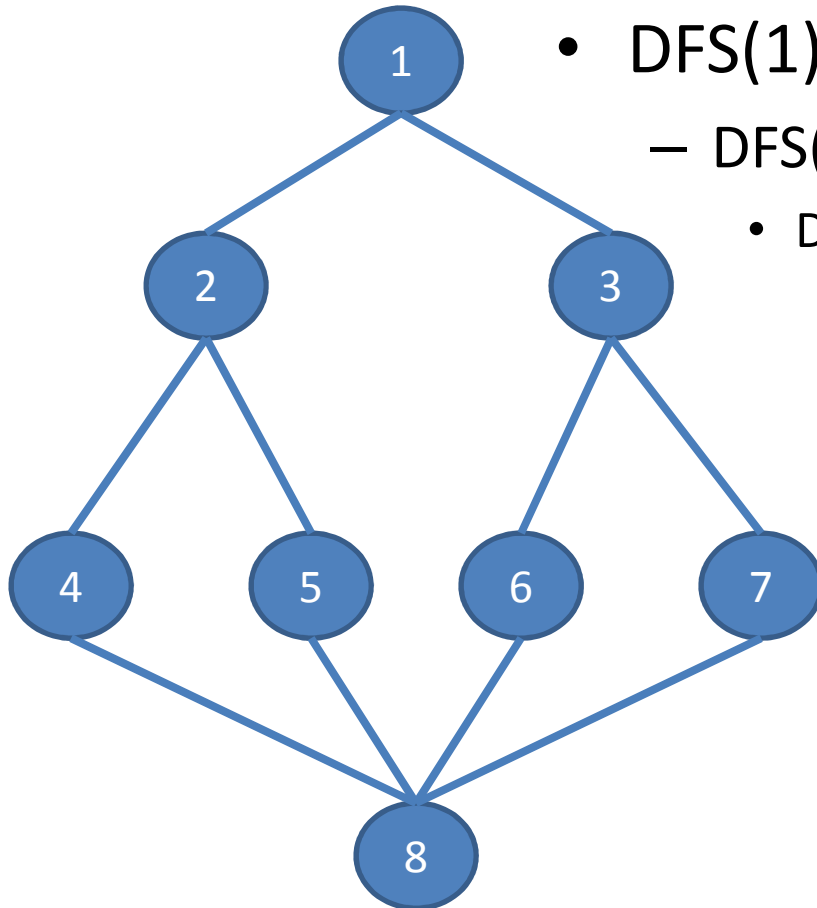
```
        DFS(w)
```

```
      endif
```

```
    endif
```

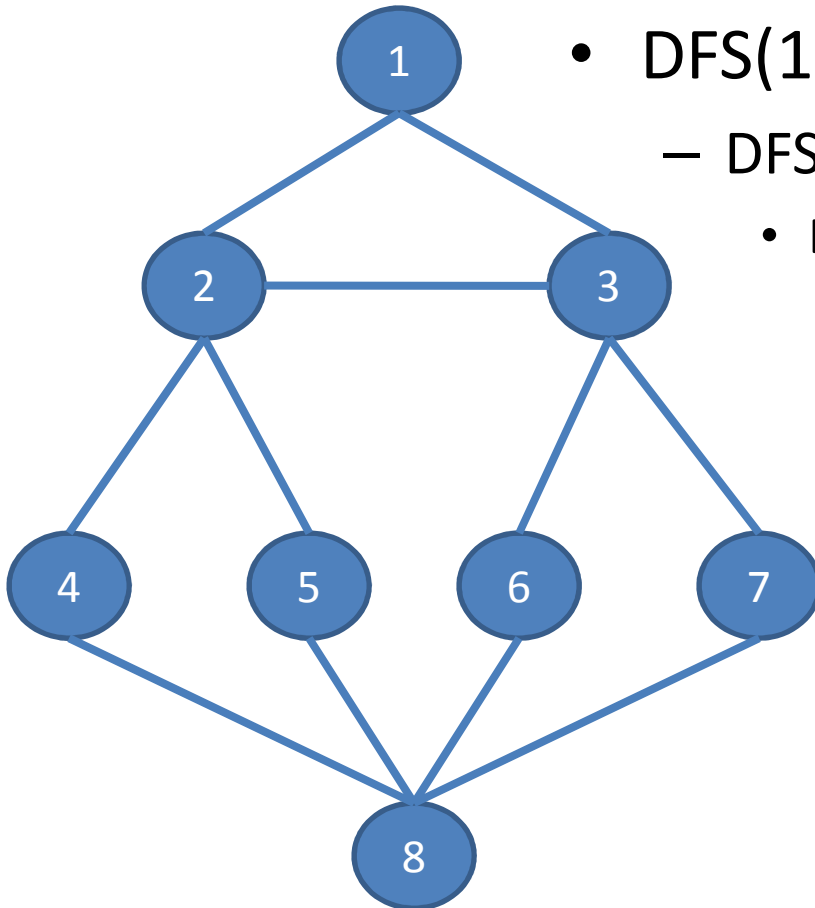
```
  endfor
```

# DFS: Ilustrasi 1



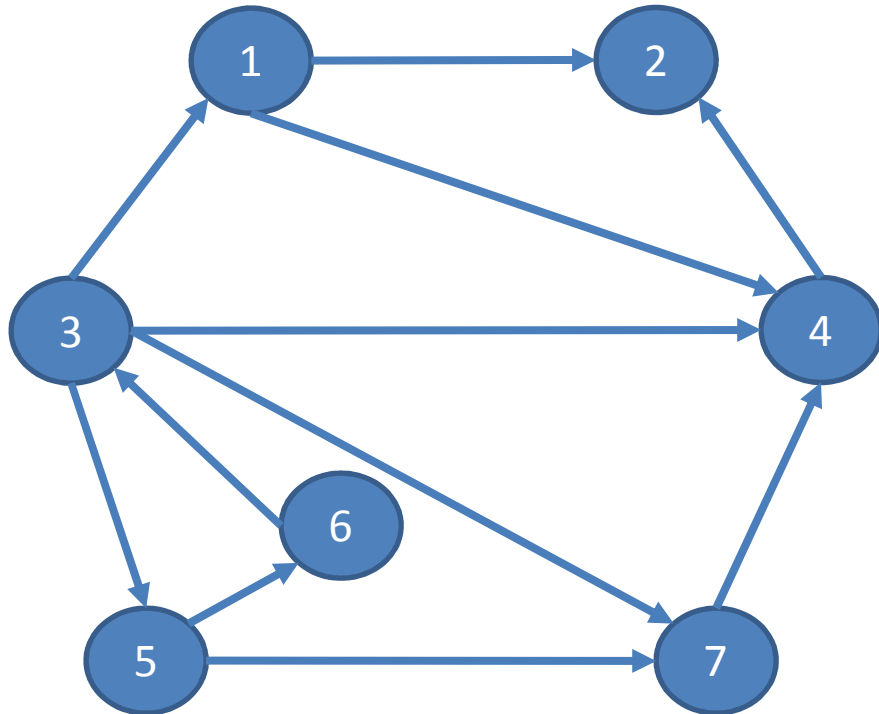
- DFS(1):  $v=1$ ; dikunjungi[1]=true; DFS(2)
  - DFS(2):  $v=2$ ; dikunjungi[2]=true; DFS(4)
    - DFS(4):  $v=4$ ; dikunjungi[4]=true; DFS(8)
      - DFS(8):  $v=8$ ; dikunjungi[8]=true; DFS(5)
        - » DFS(5):  $v=5$ ; dikunjungi[5]=true
        - » DFS(6):  $v=6$ ; dikunjungi[6]=true; DFS(3)
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

# 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

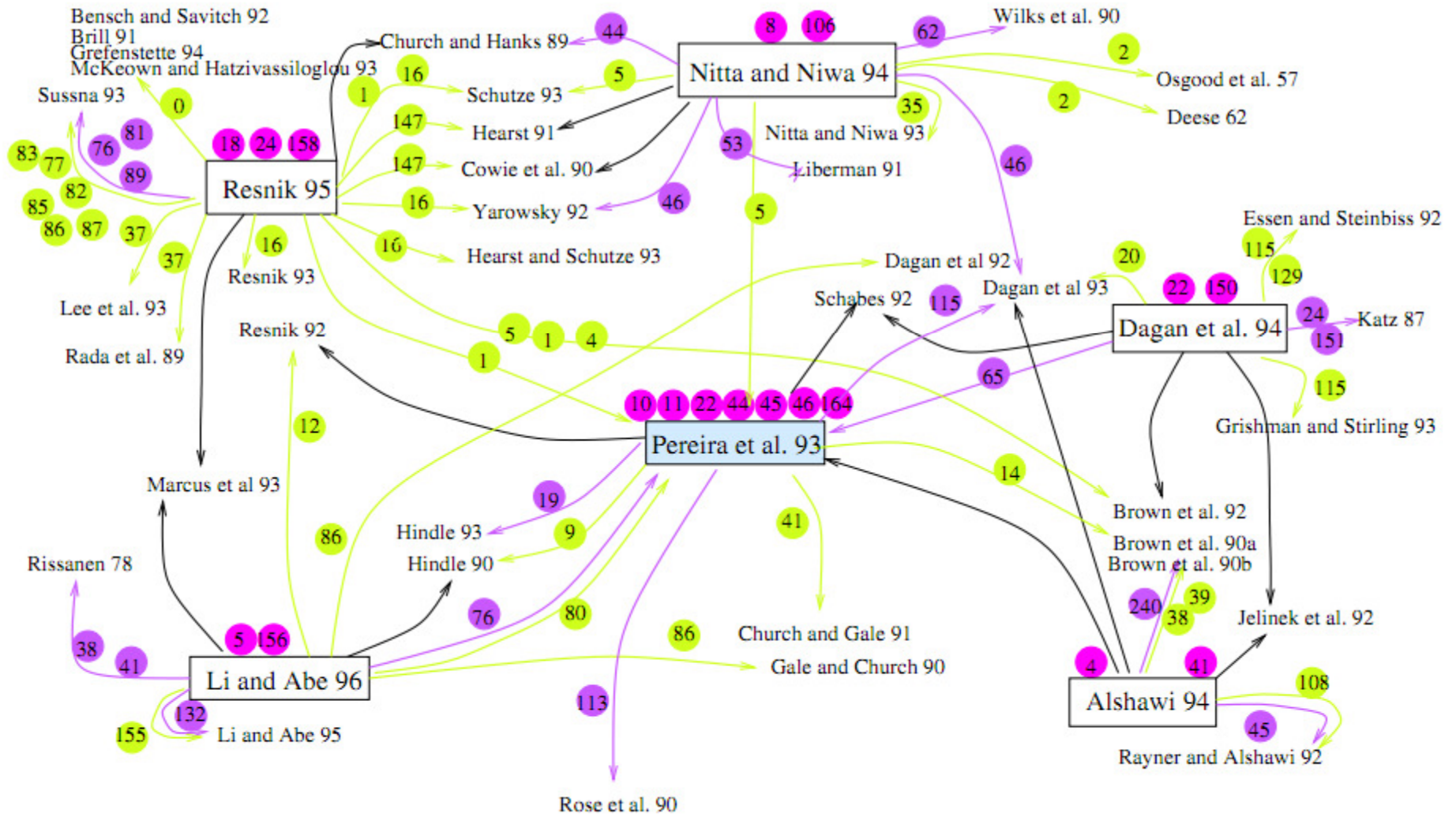
# Contoh (hal 113)



- 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

# Penerapan BFS dan DFS: Citation Map

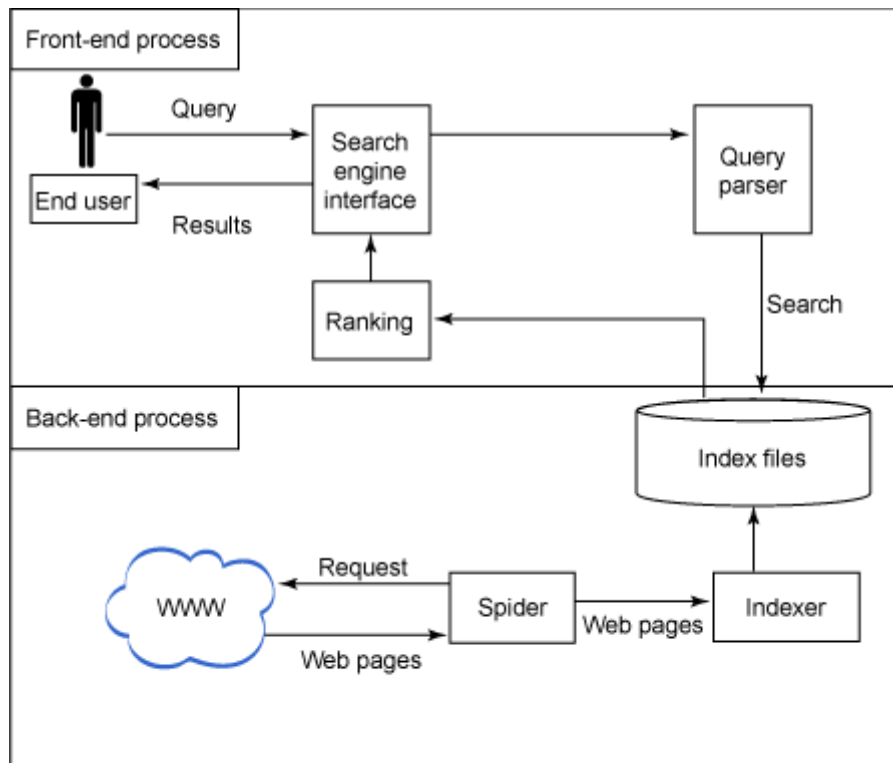


Sumber: Teufel (1999), Argumentative Zoning

RN-MLK/IF2211/2013

# Penerapan BFS dan DFS: Web Spider

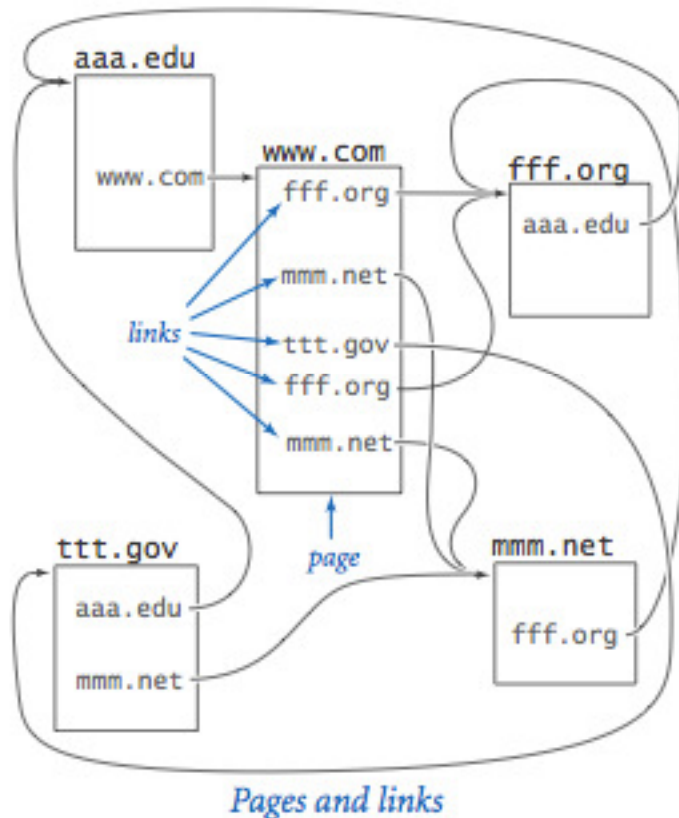
## Arsitektur umum mesin pencari



<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/>

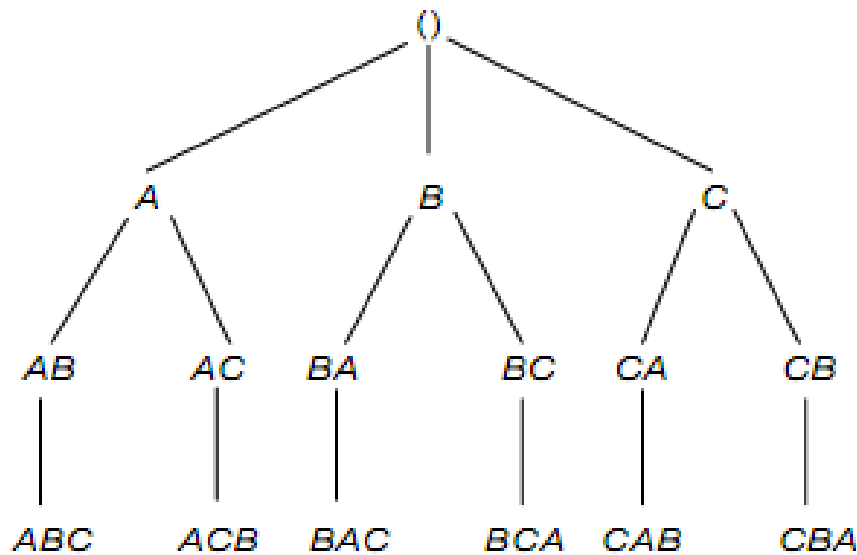
- Halaman web dimodelkan sebagai graf berarah
  - Simpul menyatakan halaman web (web page)
  - Sisi menyatakan link ke halaman web
- Bagaimana teknik menjelajahi web? Secara DFS atau BFS
- Dimulai dari web page awal, lalu setiap link ditelusuri secara DFS sampai setiap web page tidak mengandung link.

# Pencarian Solusi dengan BFS/DFS

- Persoalan optimasi:  $n$  kandidat solusi
- Pencarian solusi  $\approx$  pembentukan pohon dinamis
  - Akar: *initial state*
  - Simpul: *problem state* (layak membentuk solusi)
  - Daun: *solution/goal state*
  - Cabang: operator/langkah dalam persoalan
  - Ruang status (*state space*): himpunan semua simpul
  - Ruang solusi: himpunan status solusi
  - Pohon ruang status (*state space tree*)
- 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*).

# Permainan 8-Puzzle

2	1	6
4		8
7	5	3

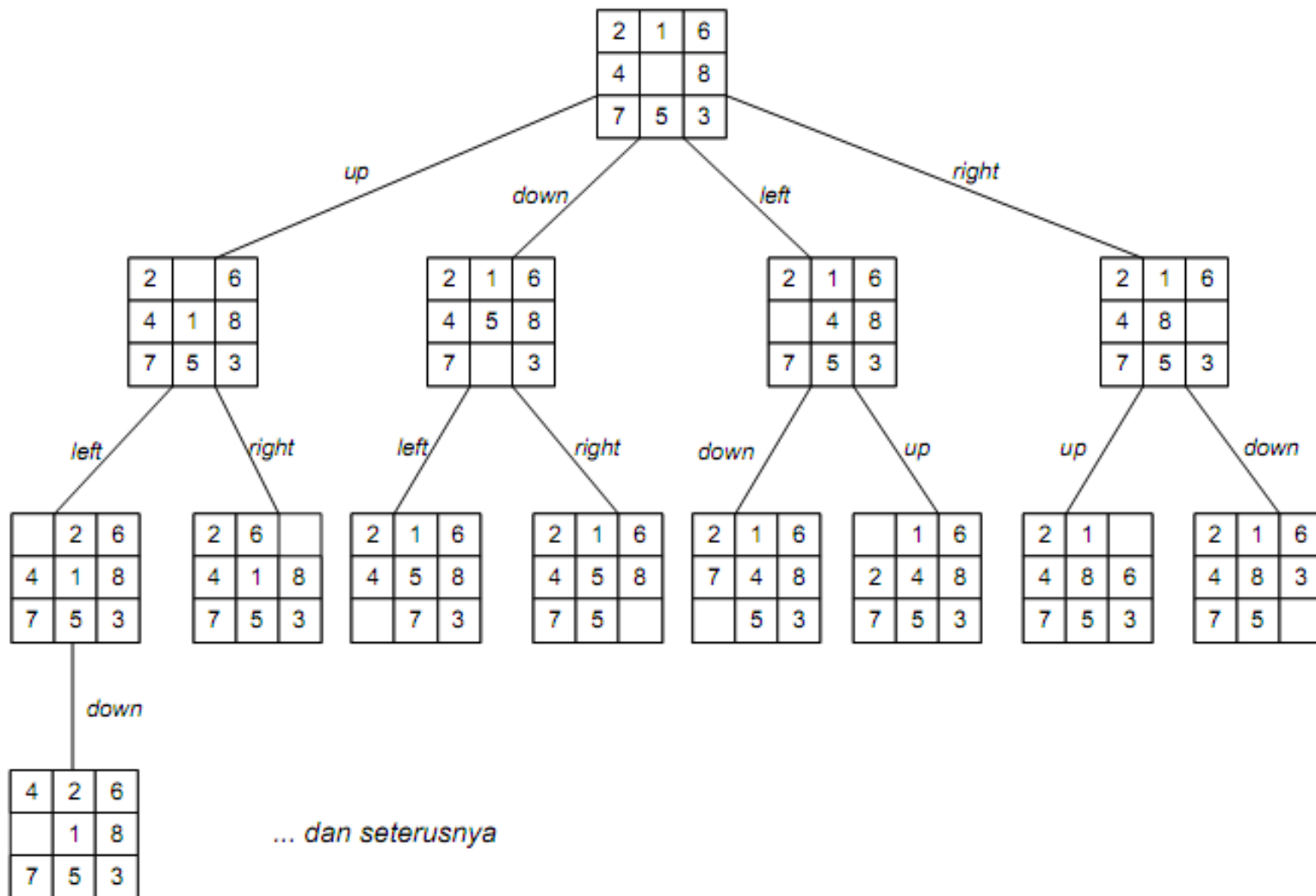
(a) Susunan awal  
(*initial state*)

1	2	3
8		4
7	6	5

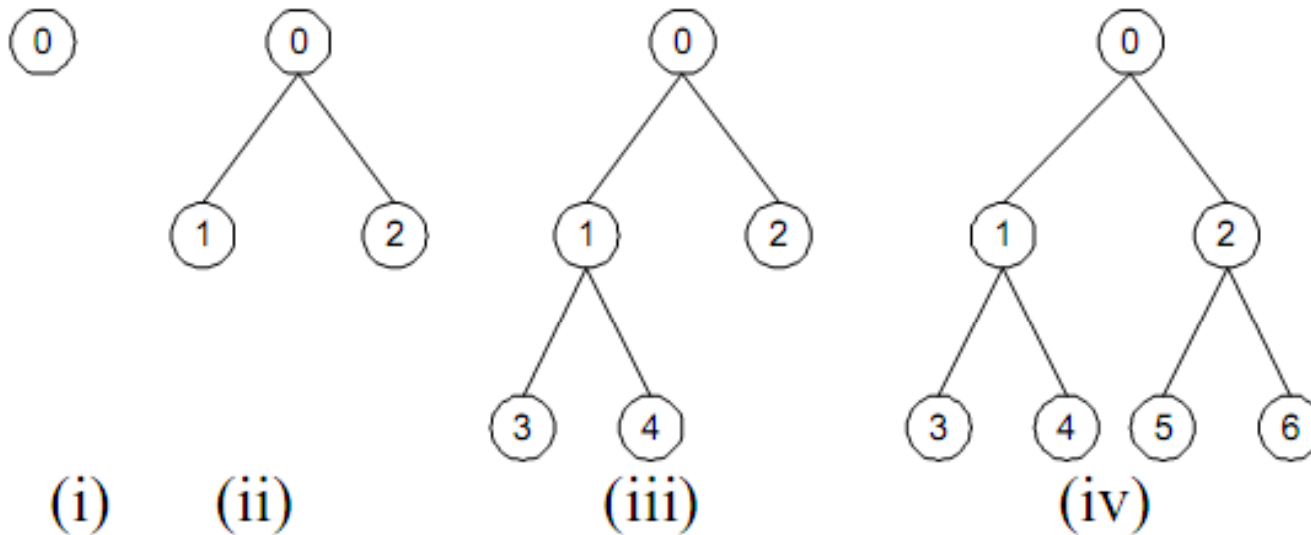
(b) Susunan akhir  
(*goal state*)

- State berdasarkan ubin kosong (*blank*)

# 8-Puzzle: Pohon Ruang Status

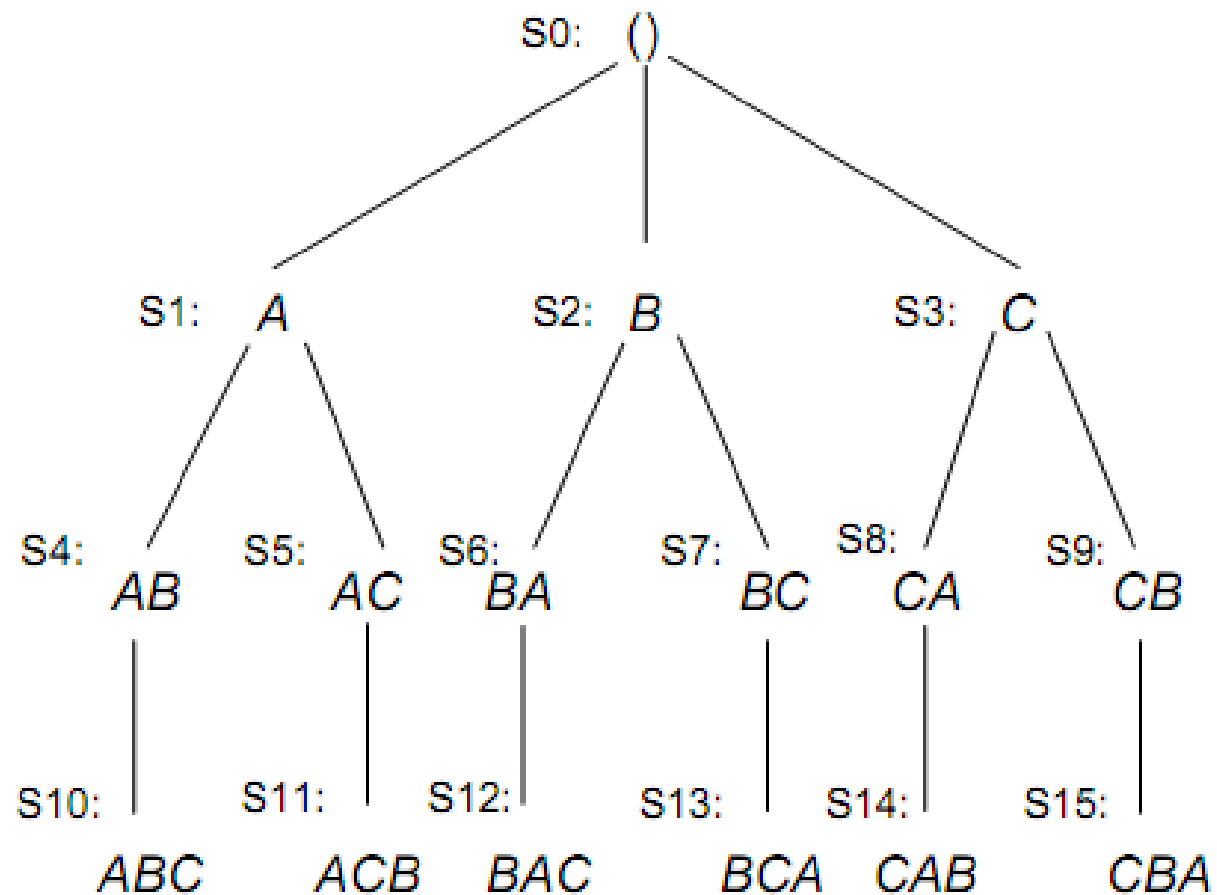


# BFS untuk Pembentukan Pohon Ruang Status

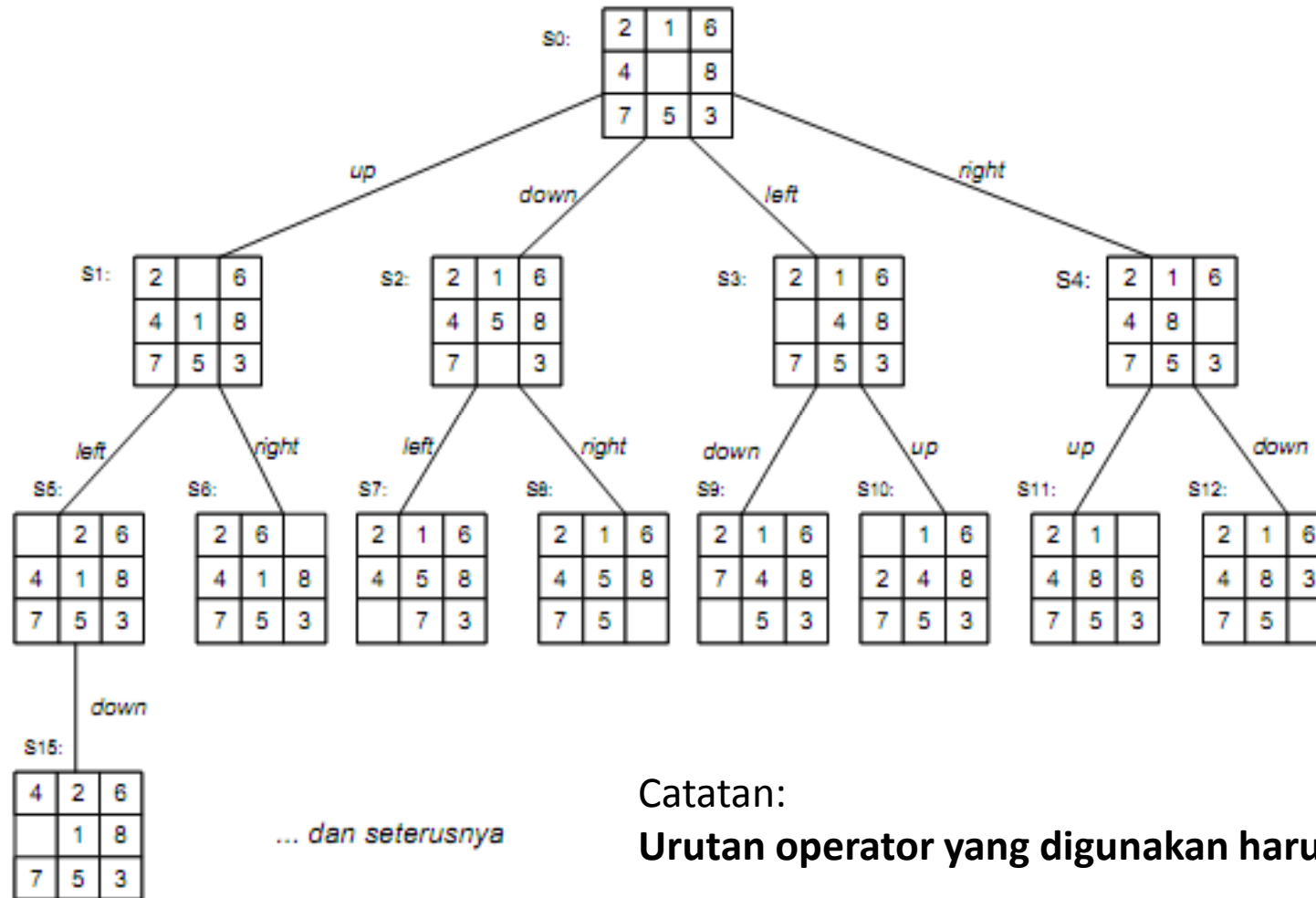


- Inisialisasi dengan status awal sebagai akar, lalu tambahkan simpul anaknya, dst.
- Semua simpul pada level  $d$  dibangkitkan terlebih dahulu sebelum simpul-simpul pada level  $d+1$

# BFS untuk Permutasi



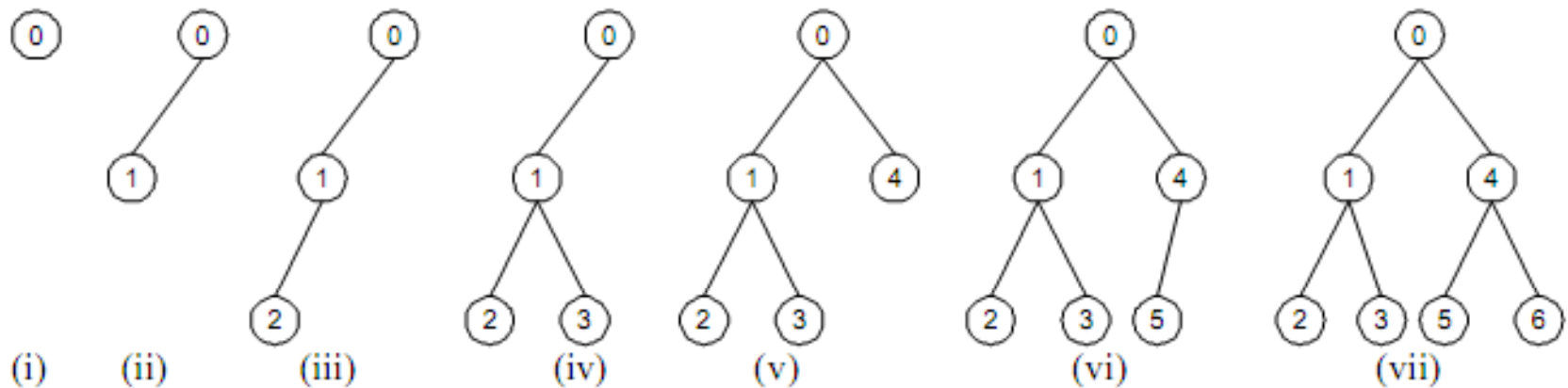
# BFS untuk 8-Puzzle



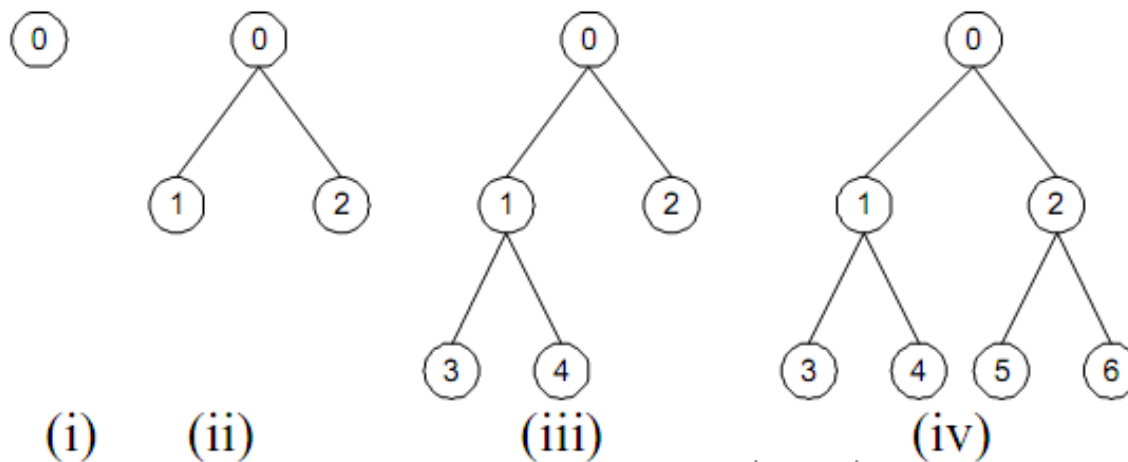
Catatan:  
**Urutan operator yang digunakan harus konsisten.**

# DFS untuk Pembentukan Pohon Ruang Status

DFS:

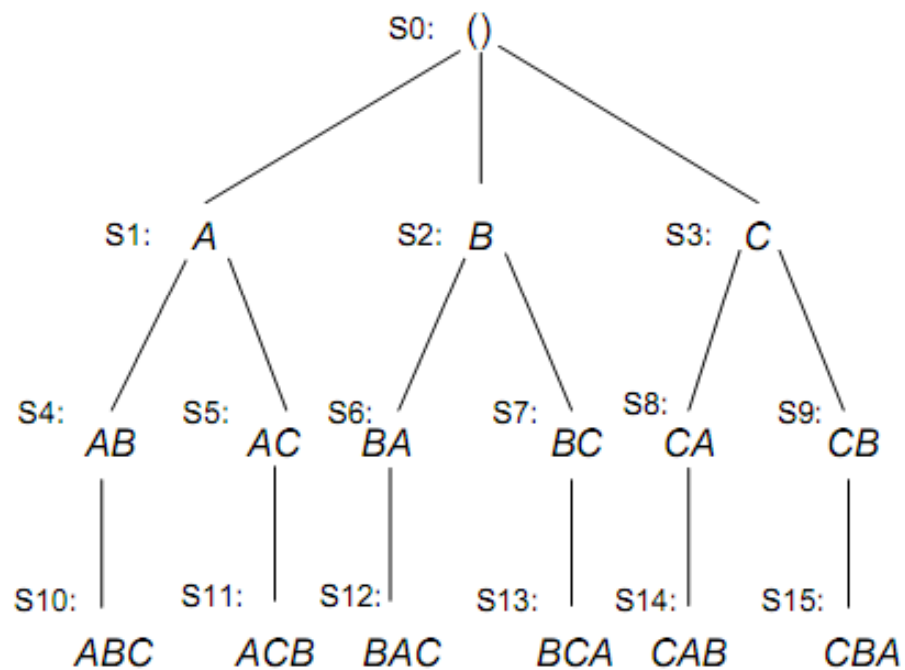


BFS:

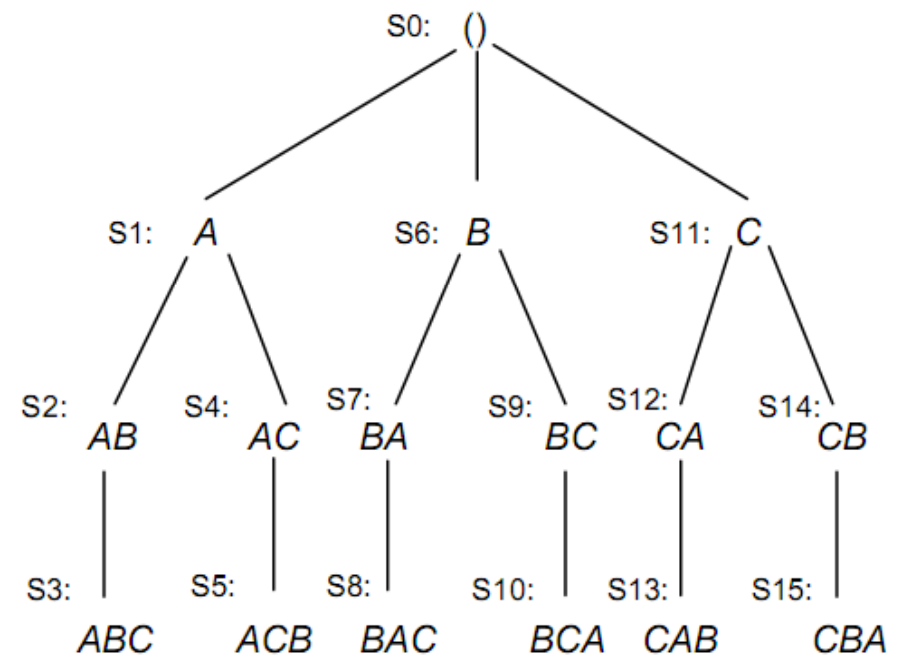


# Permutasi A,B,C

**BFS:**

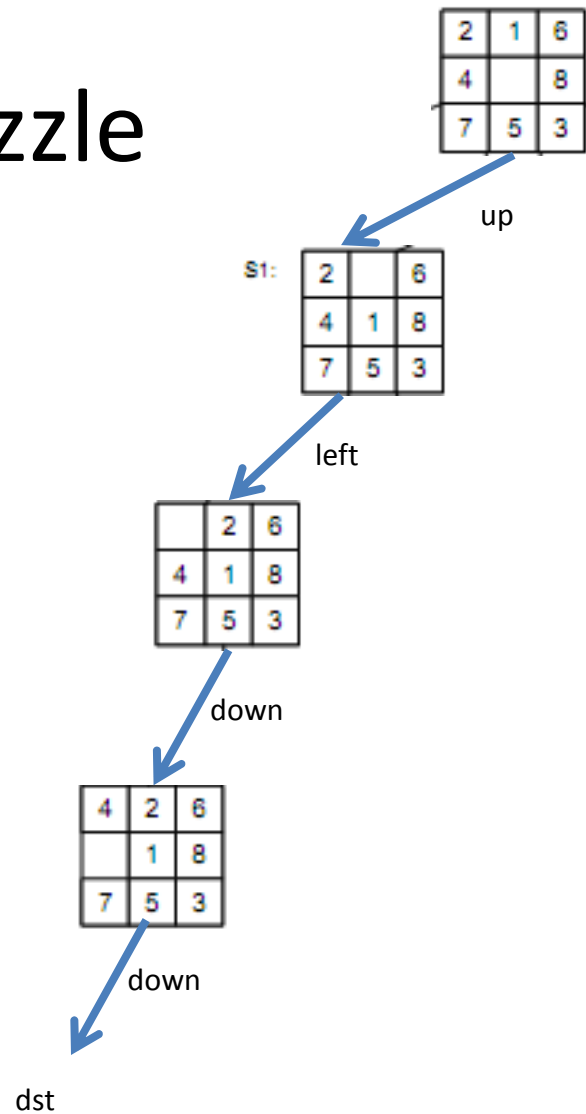


**DFS:**





# DFS untuk 8-Puzzle



DFS dapat menambahkan batasan kedalaman pohon ruang status yang dibentuk

# DFS: Contoh Lain

**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.

DOWN pindahkan bidak satu posisi ke bawah

LEFT pindahkan bidak satu posisi ke kiri

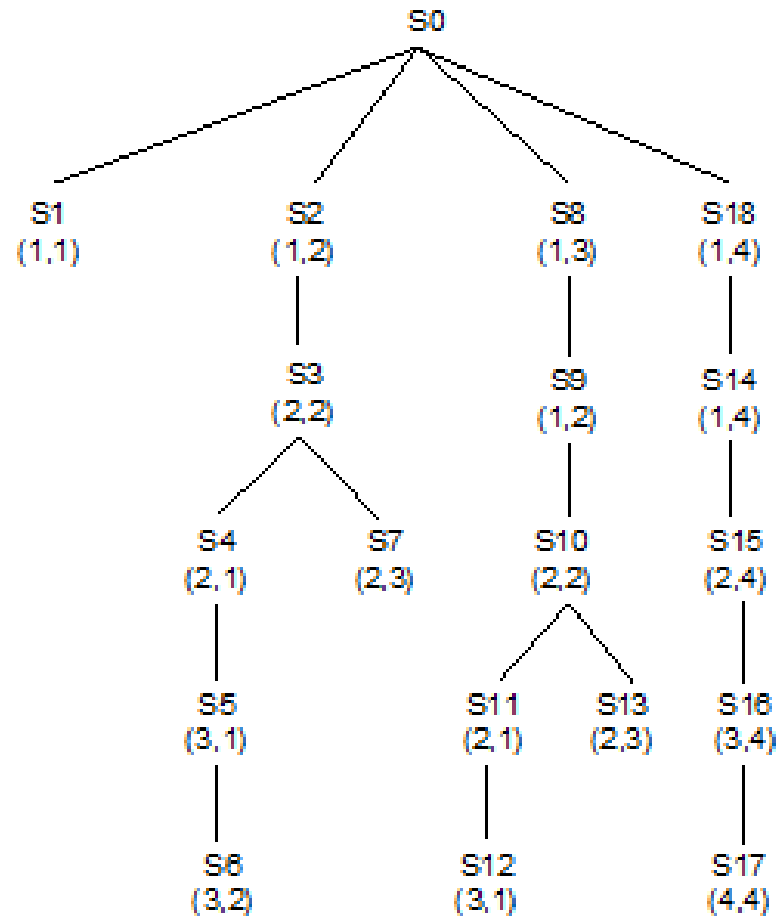
RIGHT pindahkan bidak satu posisi ke kanan

Batas kedalaman maksimum pohon ruang status diandaikan 5.

	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

# Pohon Ruang Status



# PR

- Misalkan anda mempunyai dua buah ember, masing-masing bervolume 5 liter dan 3 liter. Anda diminta mendapatkan air (dari sebuah danau) sebanyak 4 liter di dalam salah satu ember dengan menggunakan bantuan hanya kedua ember tersebut (tidak ada peralatan lain yang tersedia, hanya kedua ember itu saja yang ada!). Anda boleh memindahkan air dari satu ember ke ember lain, membuang seluruh air dari ember, dan sebagainya. Gambarkan pencarian solusi persoalan ini dengan membangun pohon secara dinamis dengan algoritma *DFS* atau *BFS*. Anda harus menjelaskan apa yang menjadi *state* persoalan.

# 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

# Iterative Deepening Search (IDS)

- IDS: perform a sequence of DFS searches with increasing depth-cutoff until goal is found
- Assumption: most of the nodes are in the bottom level so it does not matter much that upper levels are generated multiple times.

Depth  $\leftarrow$  0

Iterate

    result  $\leftarrow$  DLS (problem, depth)

stop: result  $\neq$  cutoff

    depth  $\leftarrow$  depth+1

$\rightarrow$  result



# Route/Path Planning

Materi Kuliah IF2211 – Strategi Algoritma  
Teknik Informatika - ITB

# Referensi

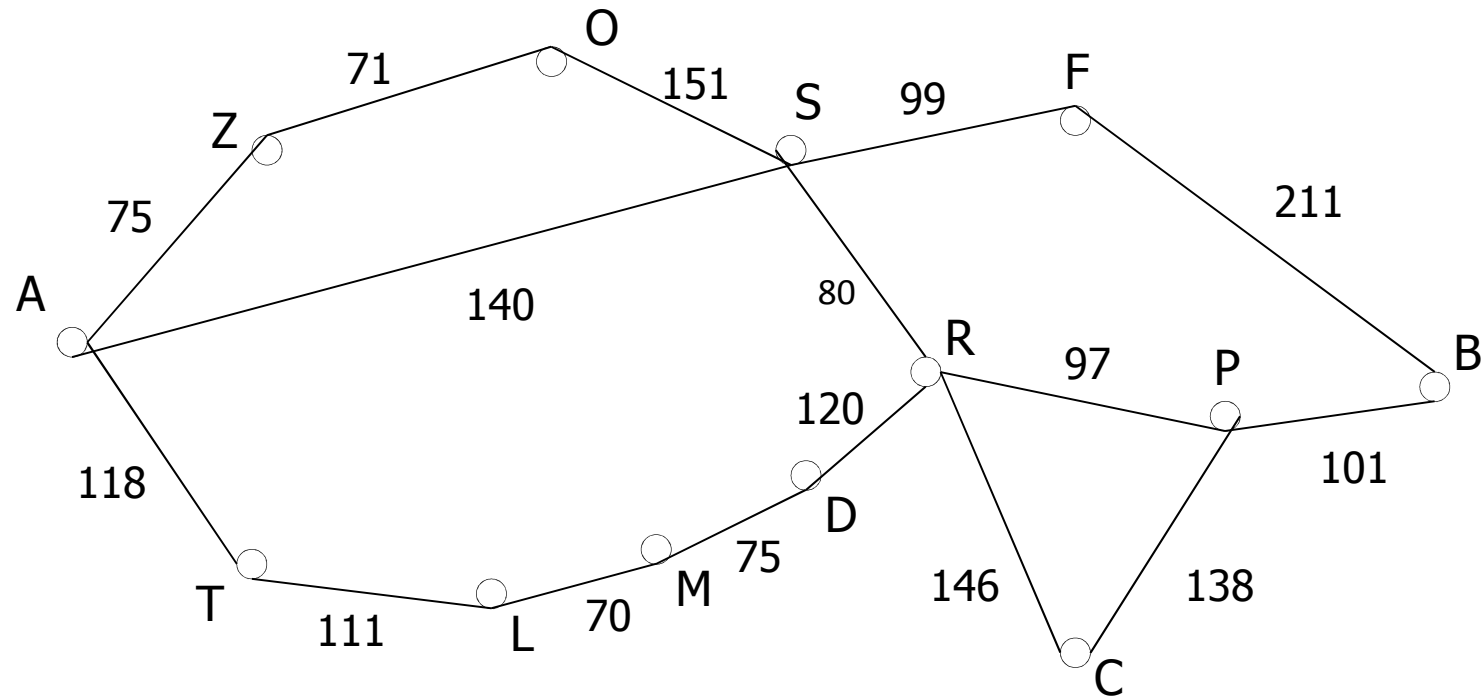
- Materi kuliah IF3054 Inteligensi Buatan Teknik Informatika ITB, Course Website:  
<http://kuliah.itb.ac.id> → STEI → Teknik Informatika → IF3054
- Stuart J Russell & Peter Norvig, Artificial Intelligence: A Modern Approach, 3rd Edition, Prentice-Hall International, Inc, 2010, Textbook  
Site: <http://aima.cs.berkeley.edu/> (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)

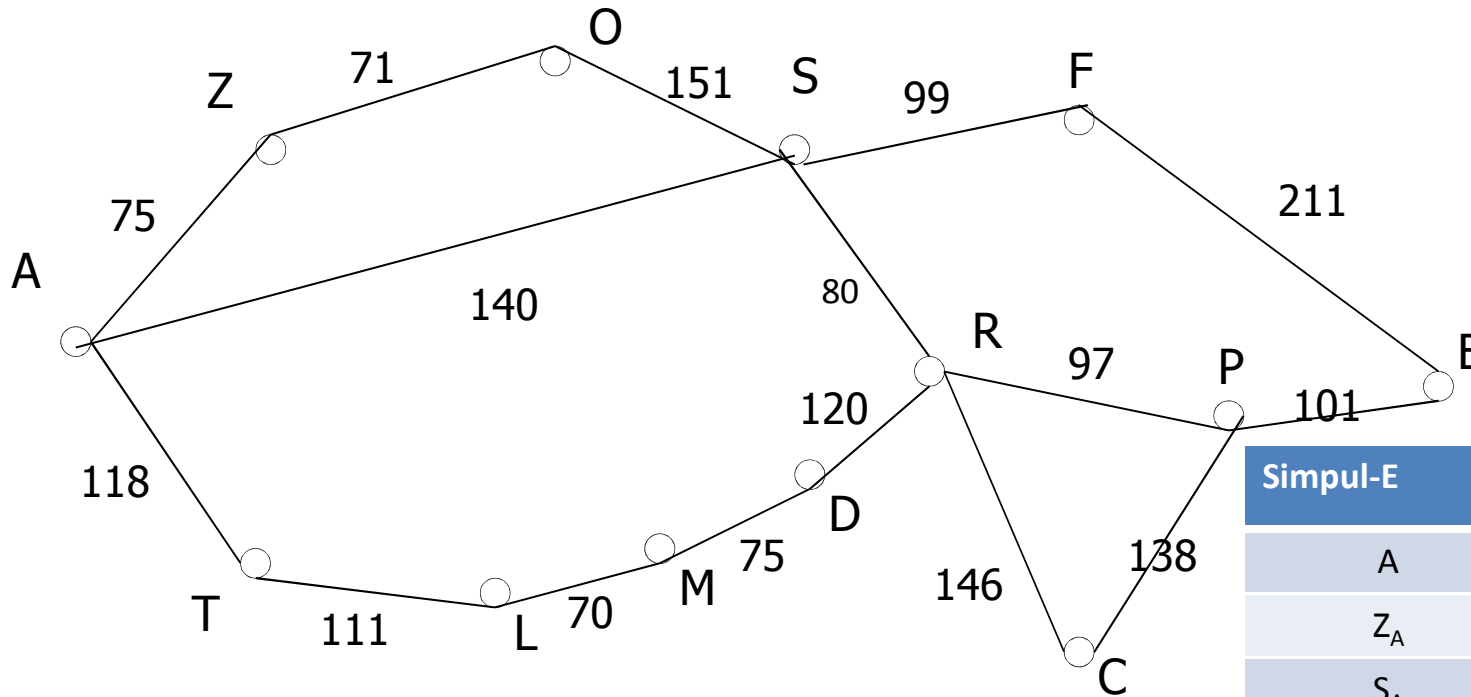
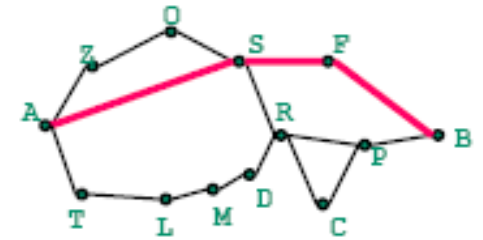
Goal test:  $s = B$  ?

Path cost: time  $\sim$  distance

# Uninformed Search

# Breadth-First Search (BFS)

Treat agenda as a queue (FIFO)

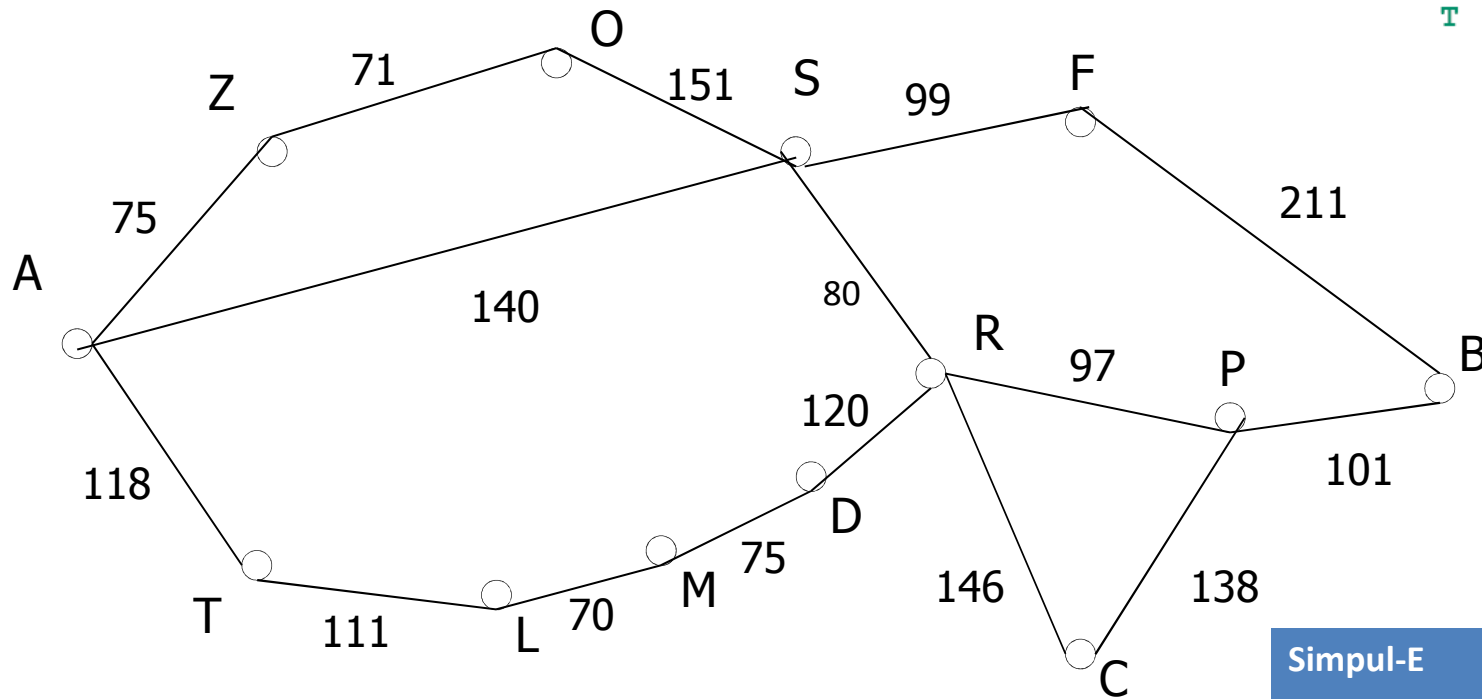
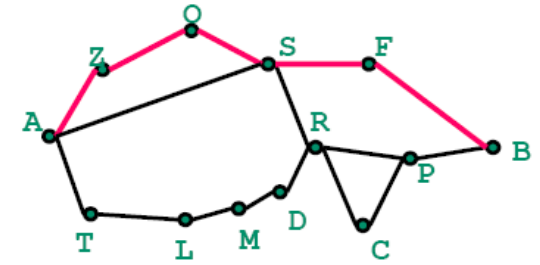


**Path: A → S → F → B,**  
**Path-cost = 450**

Simplul-E	Simplul 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)

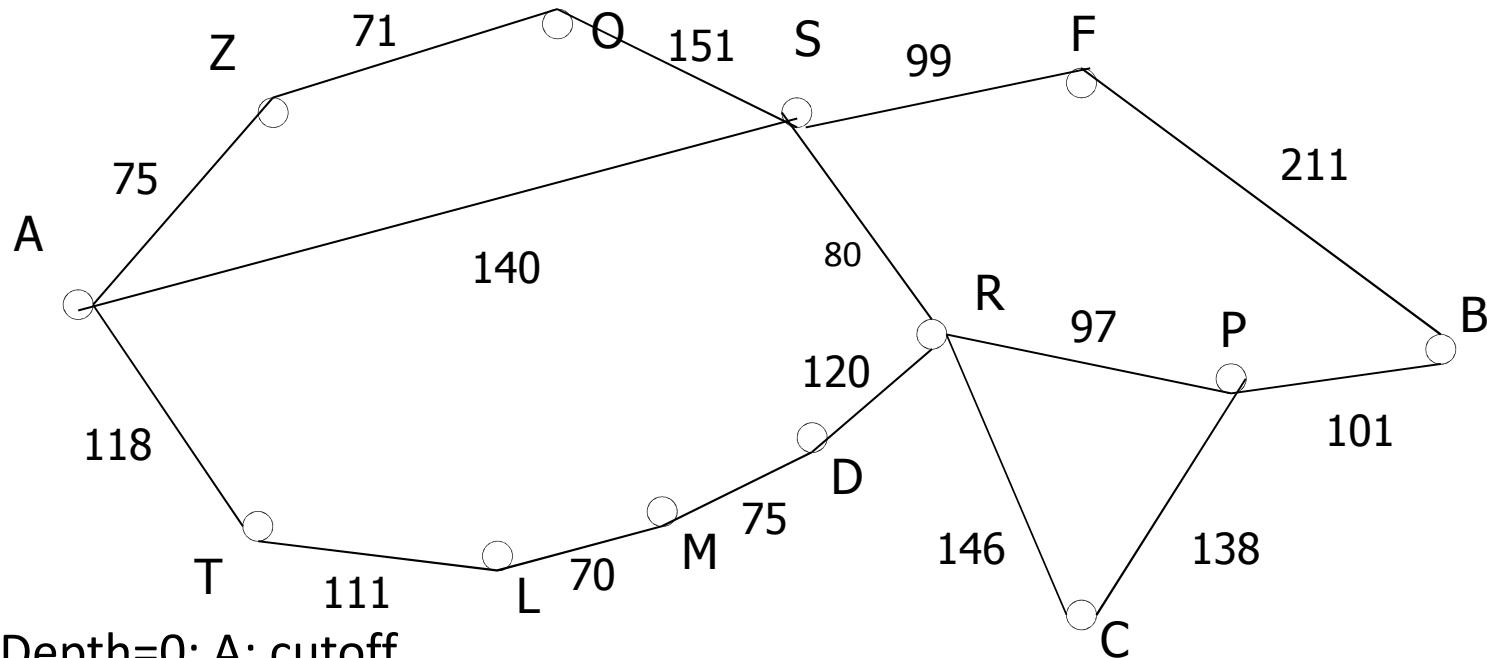
Treat agenda as a stack (LIFO)



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

Simplu-E	Simplu 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  $\rightarrow$   $Z_A, S_A, T_A \rightarrow Z_A$ : cutoff,  $S_A$ : cutoff,  $T_A$ : cutoff

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

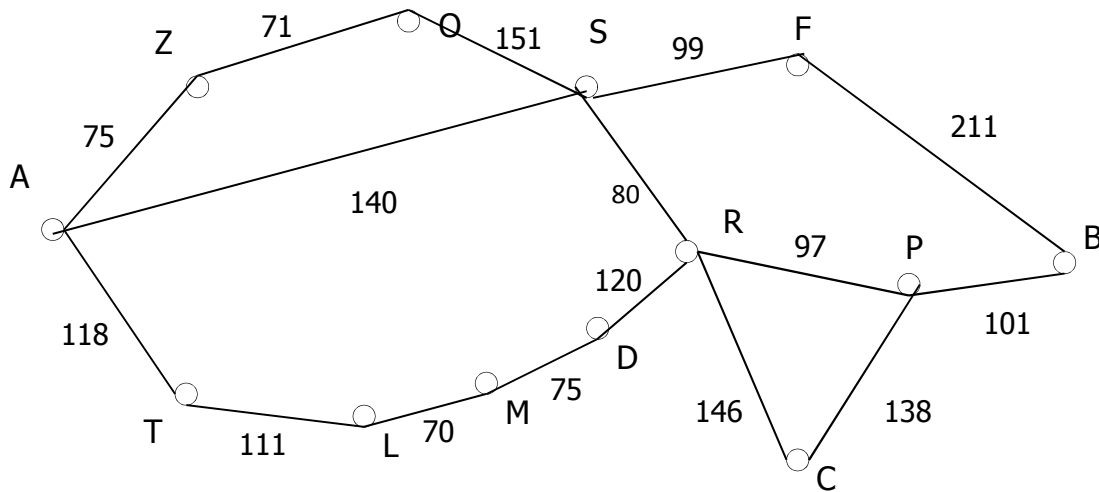
Depth=3: A  $\rightarrow$   $Z_A, S_A, T_A \rightarrow O_{AZ}, S_A, T_A \rightarrow S_{AZO}, S_A, T_A \rightarrow S_{AZO}$ : cutoff  $\rightarrow F_{AS}, R_{AS}, T_A$   
 $\rightarrow B_{ASF}, R_{AS}, T_A \rightarrow B_{ASF}$

**Stop: B=goal, path: A  $\rightarrow$  S  $\rightarrow$  F  $\rightarrow$  B, path-cost = 450**



# Uniform Cost Search (UCS)

- BFS & IDS find path with fewest steps
- If steps  $\neq$  cost, this is not relevant (to optimal solution)
- How can we find the shortest path (measured by sum of distances along path)?



**Path: A  $\rightarrow$  S  $\rightarrow$  R  $\rightarrow$  P  $\rightarrow$  B**  
**Path-cost = 418**

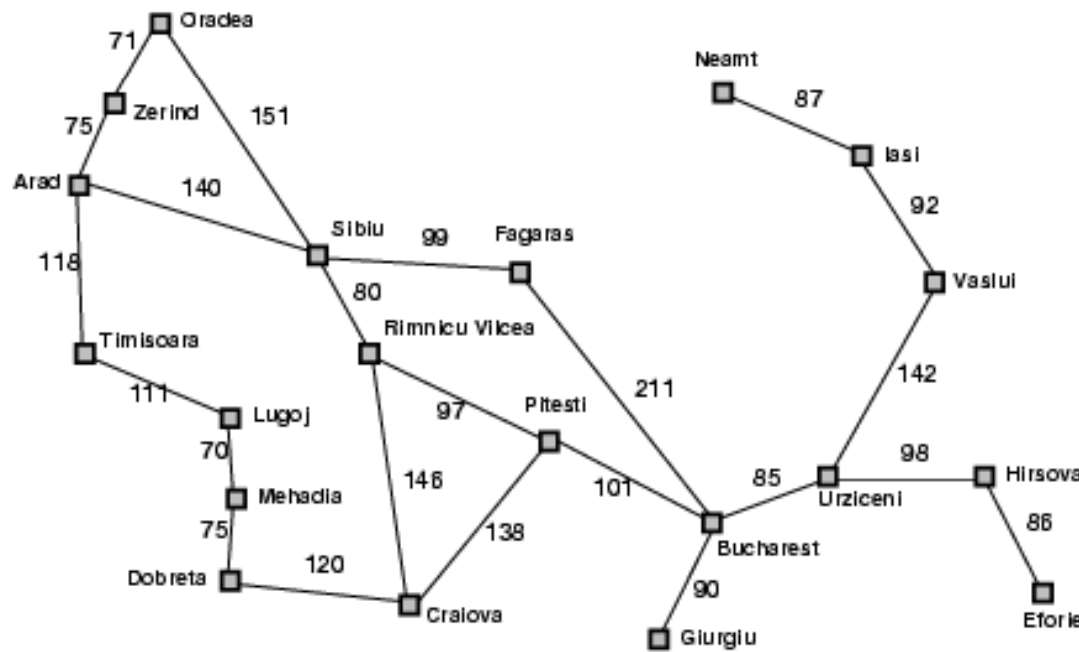
Simplu-E	Simplu Hidup
A	Z <sub>A-75</sub> , T <sub>A-118</sub> , S <sub>A-140</sub>
Z <sub>A-75</sub>	T <sub>A-118</sub> , S <sub>A-140</sub> , O <sub>AZ-146</sub>
T <sub>A-118</sub>	S <sub>A-140</sub> , O <sub>AZ-146</sub> , L <sub>AT-229</sub>
S <sub>A-140</sub>	O <sub>AZ-146</sub> , R <sub>AS-220</sub> , L <sub>AT-229</sub> , F <sub>AS-239</sub> , O <sub>AS-291</sub>
O <sub>AZ-146</sub>	R <sub>AS-220</sub> , L <sub>AT-229</sub> , F <sub>AS-239</sub> , O <sub>AS-291</sub>
R <sub>AS-220</sub>	L <sub>AT-229</sub> , F <sub>AS-239</sub> , O <sub>AS-291</sub> , P <sub>ASR-317</sub> , D <sub>ASR-340</sub> , C <sub>ASR-366</sub>
L <sub>AT-229</sub>	F <sub>AS-239</sub> , O <sub>AS-291</sub> , M <sub>ATL-299</sub> , P <sub>ASR-317</sub> , D <sub>ASR-340</sub> , C <sub>ASR-366</sub>
F <sub>AS-239</sub>	O <sub>AS-291</sub> , M <sub>ATL-299</sub> , P <sub>ASR-317</sub> , D <sub>ASR-340</sub> , C <sub>ASR-366</sub> , B <sub>ASF-450</sub>
O <sub>AS-291</sub>	M <sub>ATL-299</sub> , P <sub>ASR-317</sub> , D <sub>ASR-340</sub> , C <sub>ASR-366</sub> , B <sub>ASF-450</sub>
M <sub>ATL-299</sub>	P <sub>ASR-317</sub> , D <sub>ASR-340</sub> , D <sub>ATLM-364</sub> , C <sub>ASR-366</sub> , B <sub>ASRP-418</sub> , C <sub>ASRP-455</sub> , B <sub>ASF-450</sub>
P <sub>ASR-317</sub>	D <sub>ASR-340</sub> , D <sub>ATLM-364</sub> , C <sub>ASR-366</sub> , B <sub>ASRP-418</sub> , C <sub>ASRP-455</sub> , B <sub>ASF-450</sub>
D <sub>ASR-340</sub>	D <sub>ATLM-364</sub> , C <sub>ASR-366</sub> , B <sub>ASRP-418</sub> , C <sub>ASRP-455</sub> , B <sub>ASF-450</sub>
D <sub>ATLM-364</sub>	C <sub>ASR-366</sub> , B <sub>ASRP-418</sub> , C <sub>ASRP-455</sub> , B <sub>ASF-450</sub>
C <sub>ASR-366</sub>	B <sub>ASRP-418</sub> , C <sub>ASRP-455</sub> , B <sub>ASF-450</sub>

# Informed Search

# Best-first search

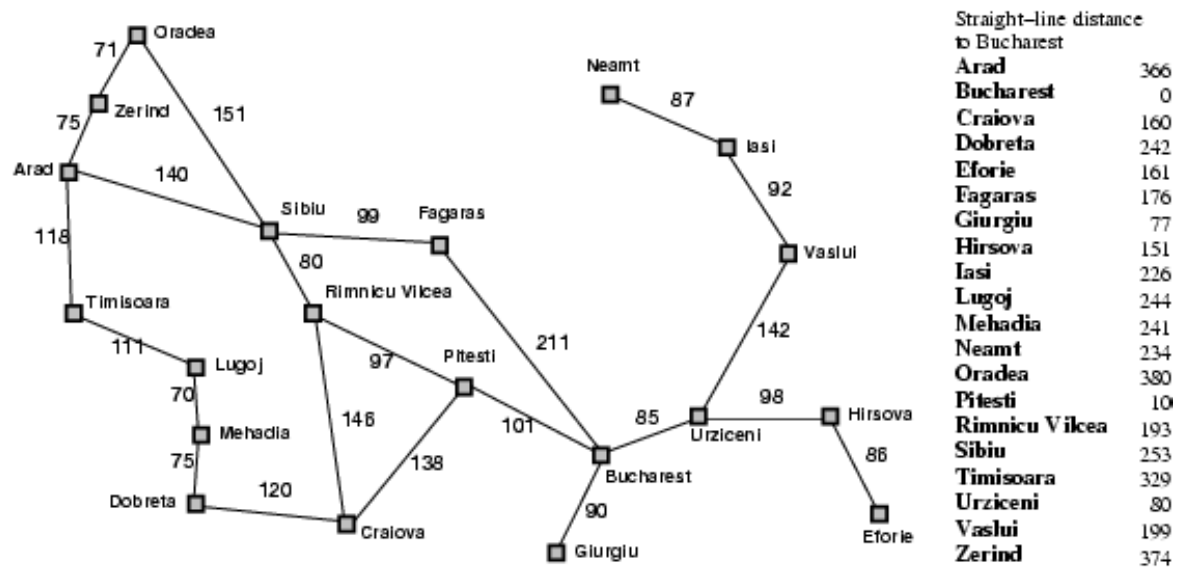
$g(n)$  = cost so far to reach  $n$   
 $h(n)$  = estimated cost from  $n$  to goal  
 $f(n)$  = estimated total cost of path through  $n$  to goal

- Idea: use an **evaluation function**  $f(n)$  for each node
  - greedy best-first search:  $f(n) = h(n)$
  - A\* search:  $f(n) = g(n) + h(n)$
  - e.g.,  $h_{SLD}(n)$  = straight-line distance from  $n$  to Bucharest
- Romania with step costs in km

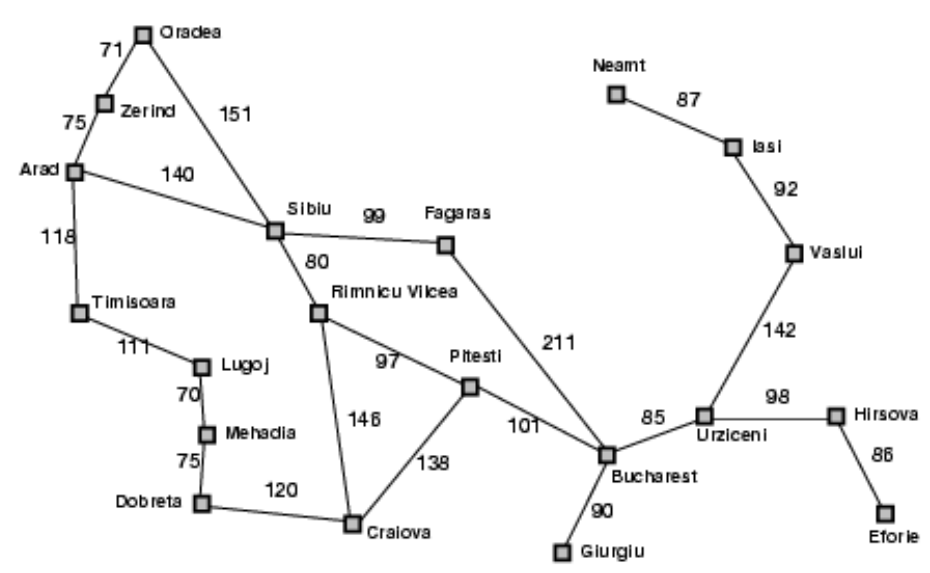
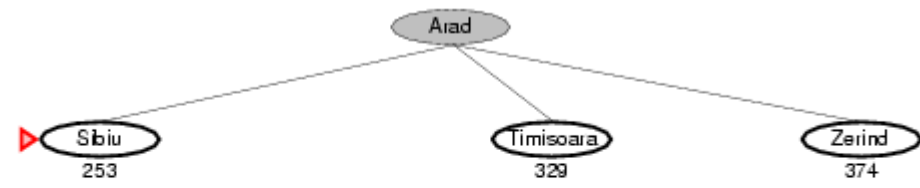


Straight-line distance to Bucharest	
<b>Arad</b>	366
<b>Bucharest</b>	0
<b>Craiova</b>	160
<b>Dobreta</b>	242
<b>Eforie</b>	161
<b>Fagaras</b>	176
<b>Giurgiu</b>	77
<b>Hirsova</b>	151
<b>Iasi</b>	226
<b>Lugoj</b>	244
<b>Mehadia</b>	241
<b>Neamt</b>	234
<b>Oradea</b>	380
<b>Pitesti</b>	10
<b>Rimnicu Vilcea</b>	193
<b>Sibiu</b>	253
<b>Timisoara</b>	329
<b>Urziceni</b>	80
<b>Vaslui</b>	199
<b>Zerind</b>	374

# Greedy best-first search example



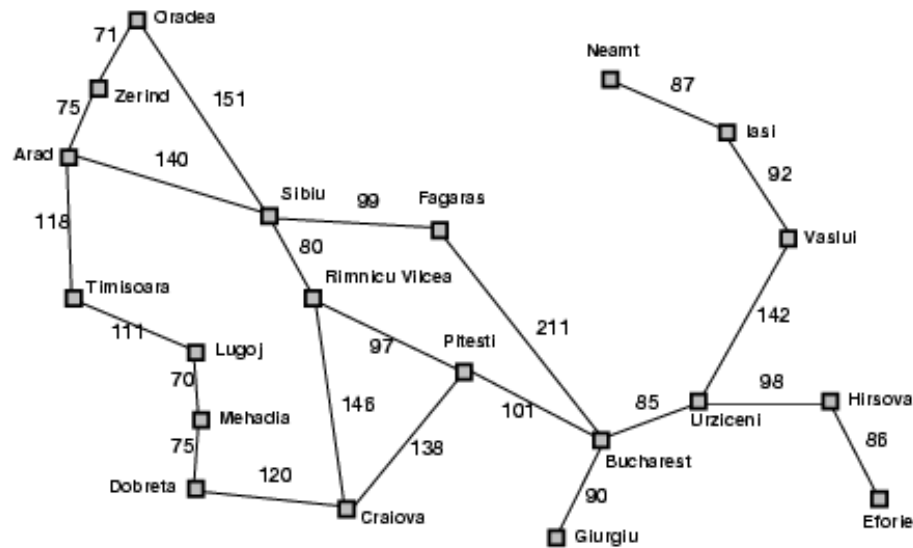
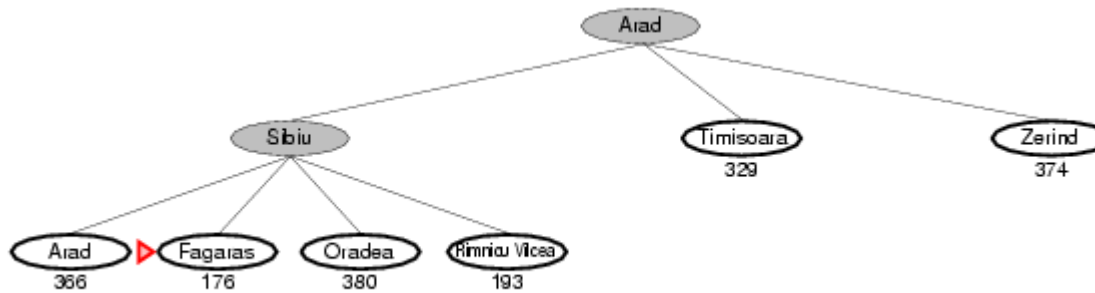
# Greedy best-first search example



Straight-line distance to Bucharest

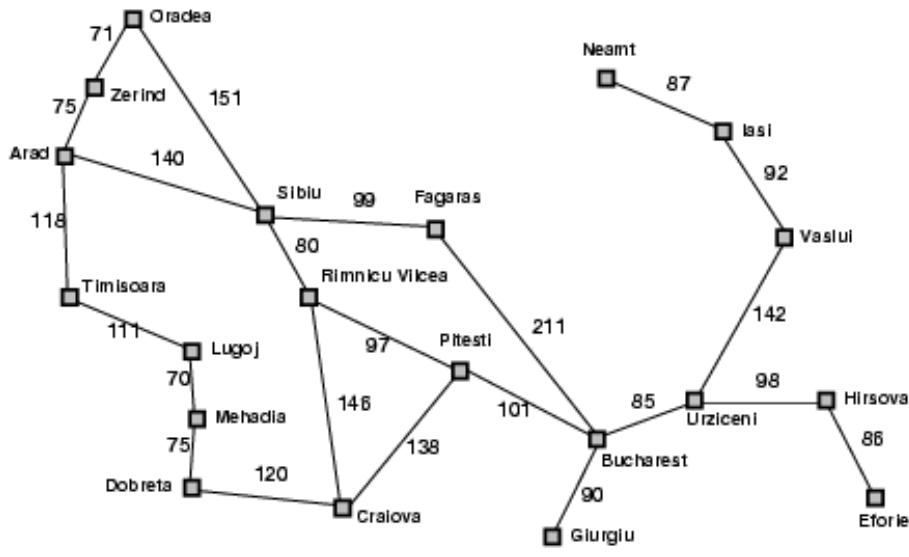
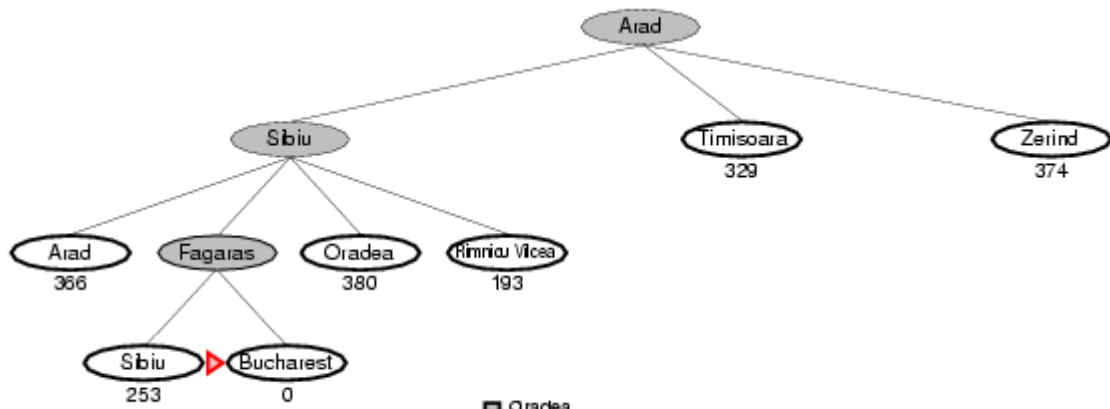
<b>Arad</b>	366
<b>Bucharest</b>	0
<b>Craiova</b>	160
<b>Dobreta</b>	242
<b>Eforie</b>	161
<b>Fagaras</b>	176
<b>Giurgiu</b>	77
<b>Hirsova</b>	151
<b>Iasi</b>	226
<b>Lugoj</b>	244
<b>Mehadia</b>	241
<b>Neamt</b>	234
<b>Oradea</b>	380
<b>Pitesti</b>	10
<b>Rimnicu Vilcea</b>	193
<b>Sibiu</b>	253
<b>Timisoara</b>	329
<b>Urziceni</b>	80
<b>Vaslui</b>	199
<b>Zerind</b>	374

# Greedy best-first search example



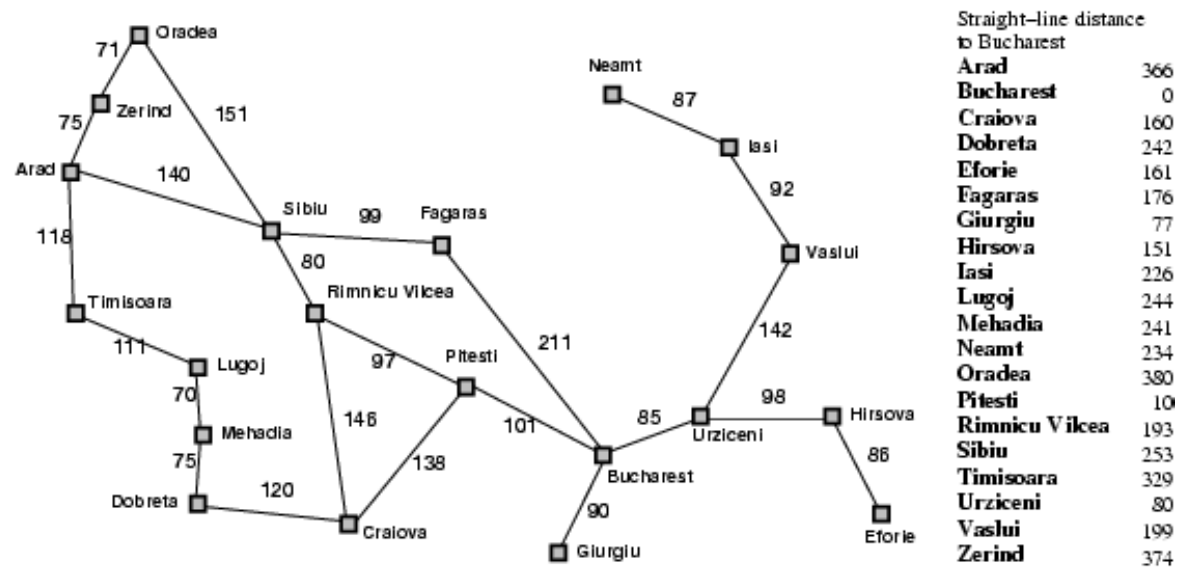
Straight-line distance to Bucharest	
<b>Arad</b>	366
<b>Bucharest</b>	0
<b>Craiova</b>	160
<b>Dobreta</b>	242
<b>Eforie</b>	161
<b>Fagaras</b>	176
<b>Giurgiu</b>	77
<b>Hirsova</b>	151
<b>Iasi</b>	226
<b>Lugoj</b>	244
<b>Mehadia</b>	241
<b>Neamt</b>	234
<b>Oradea</b>	380
<b>Pitesti</b>	10
<b>Rimnicu Vilcea</b>	193
<b>Sibiu</b>	253
<b>Timisoara</b>	329
<b>Urziceni</b>	80
<b>Vaslui</b>	199
<b>Zerind</b>	374

# Greedy best-first search example



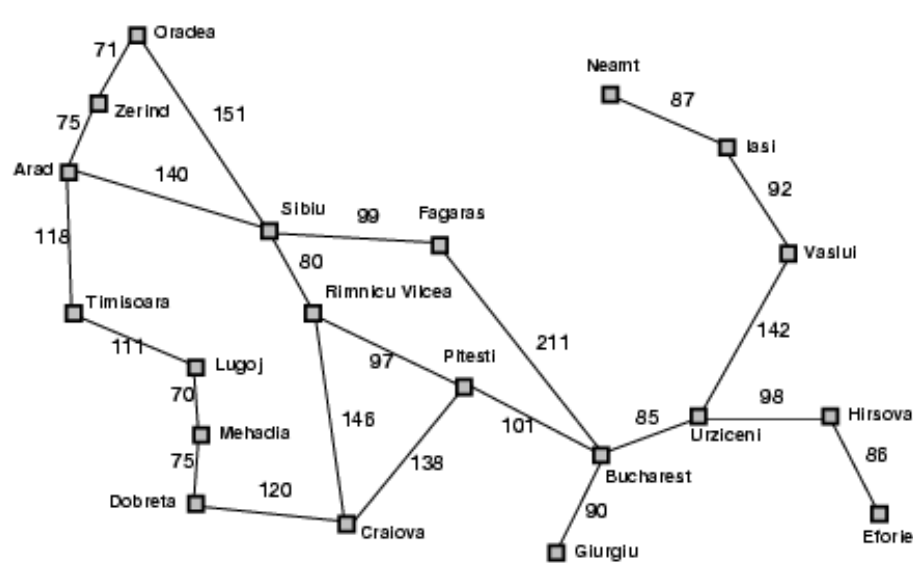
Straight-line distance to Bucharest	
<b>Arad</b>	366
<b>Bucharest</b>	0
<b>Craiova</b>	160
<b>Dobreta</b>	242
<b>Eforie</b>	161
<b>Fagaras</b>	176
<b>Giurgiu</b>	77
<b>Hirsova</b>	151
<b>Iasi</b>	226
<b>Lugoj</b>	244
<b>Mehadia</b>	241
<b>Neamt</b>	234
<b>Oradea</b>	380
<b>Pitesti</b>	10
<b>Rimnicu Vilcea</b>	193
<b>Sibiu</b>	253
<b>Timisoara</b>	329
<b>Urziceni</b>	80
<b>Vaslui</b>	199
<b>Zerind</b>	374

# A\* search example





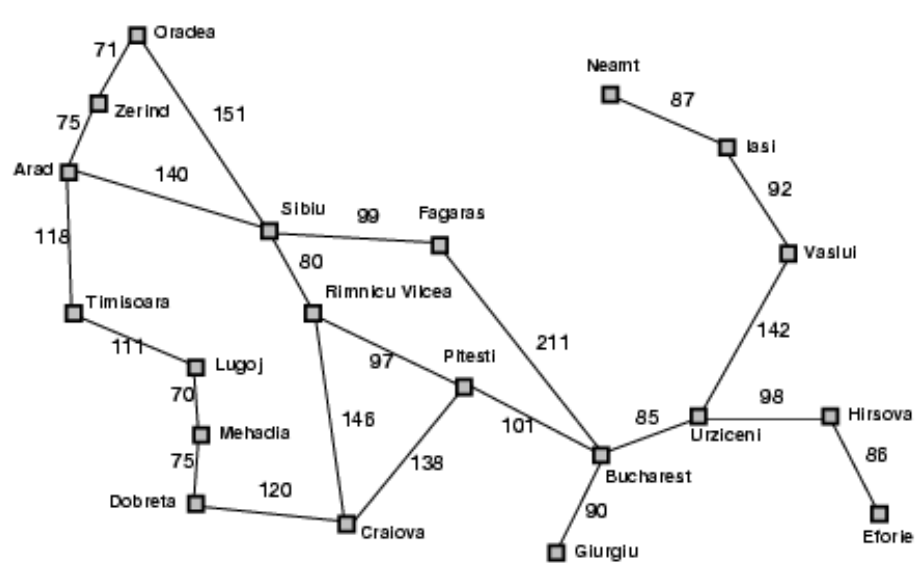
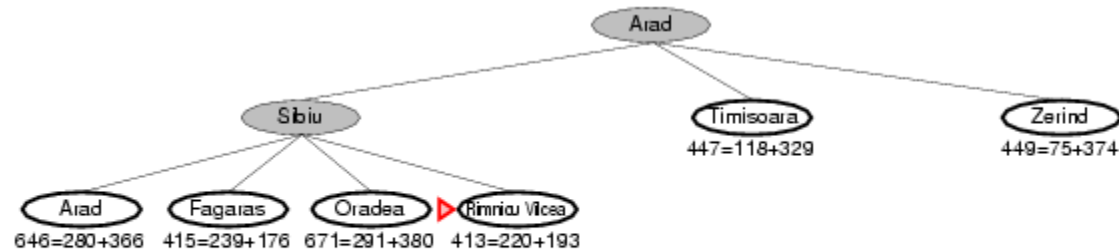
# A\* search example



Straight-line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

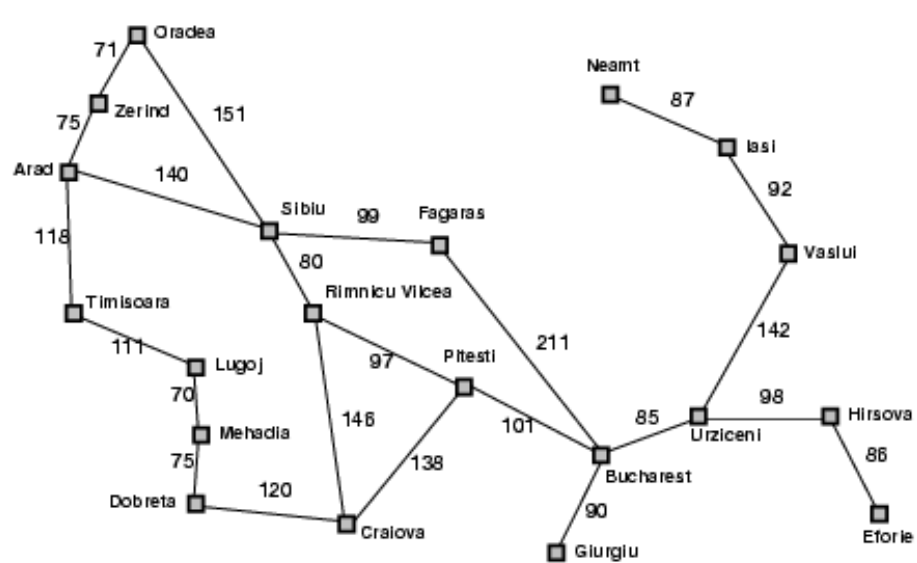
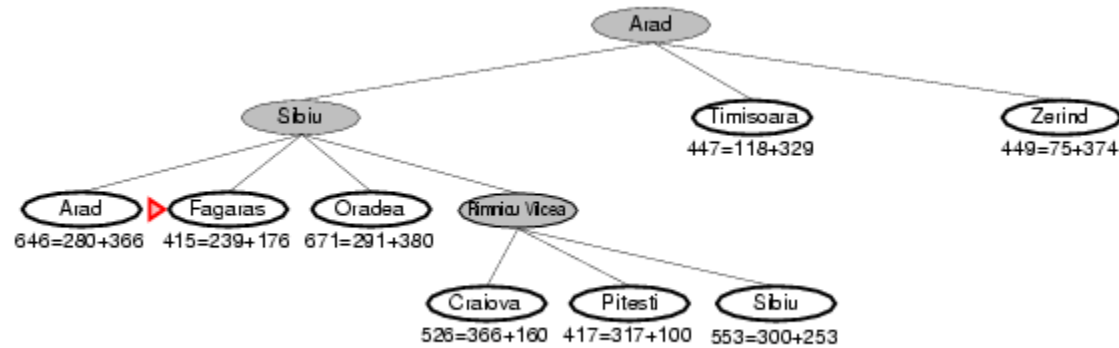
# A\* search example



Straight-line distance to Bucharest

<b>Arad</b>	366
<b>Bucharest</b>	0
<b>Craiova</b>	160
<b>Dobreta</b>	242
<b>Eforie</b>	161
<b>Fagaras</b>	176
<b>Giurgiu</b>	77
<b>Hirsova</b>	151
<b>Iasi</b>	226
<b>Lugoj</b>	244
<b>Mehadia</b>	241
<b>Neamt</b>	234
<b>Oradea</b>	380
<b>Pitesti</b>	10
<b>Rimnicu Vilcea</b>	193
<b>Sibiu</b>	253
<b>Timisoara</b>	329
<b>Urziceni</b>	80
<b>Vaslui</b>	199
<b>Zerind</b>	374

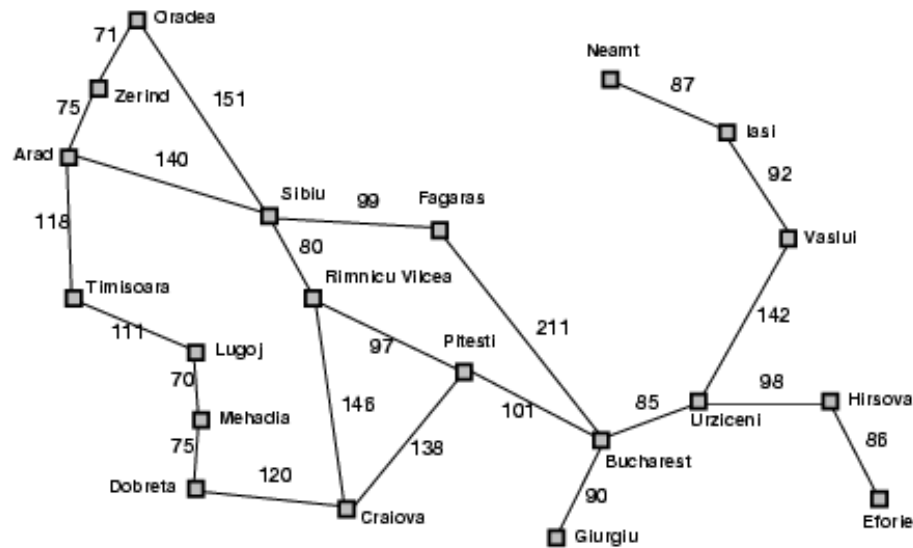
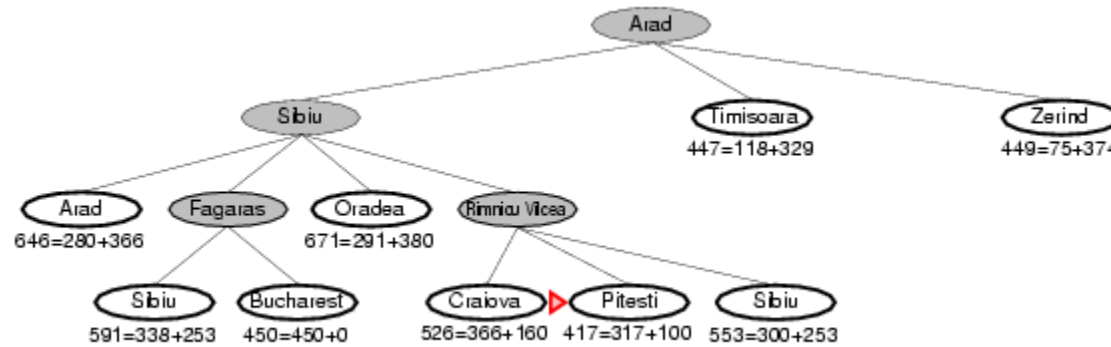
# A\* search example



Straight-line distance to Bucharest

<b>Arad</b>	366
<b>Bucharest</b>	0
<b>Craiova</b>	160
<b>Dobreta</b>	242
<b>Eforie</b>	161
<b>Fagaras</b>	176
<b>Giurgiu</b>	77
<b>Hirsova</b>	151
<b>Iasi</b>	226
<b>Lugoj</b>	244
<b>Mehadia</b>	241
<b>Neamt</b>	234
<b>Oradea</b>	380
<b>Pitesti</b>	10
<b>Rimnicu Vilcea</b>	193
<b>Sibiu</b>	253
<b>Timisoara</b>	329
<b>Urziceni</b>	80
<b>Vaslui</b>	199
<b>Zerind</b>	374

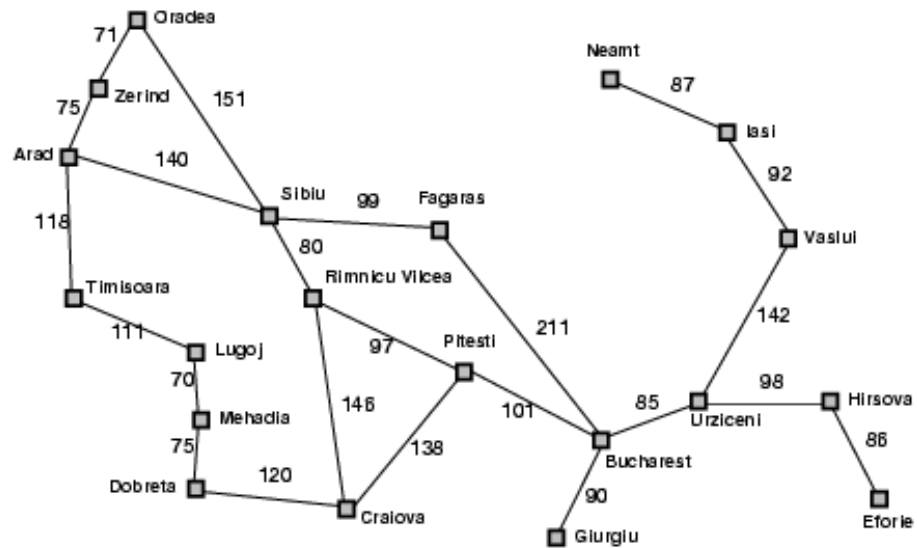
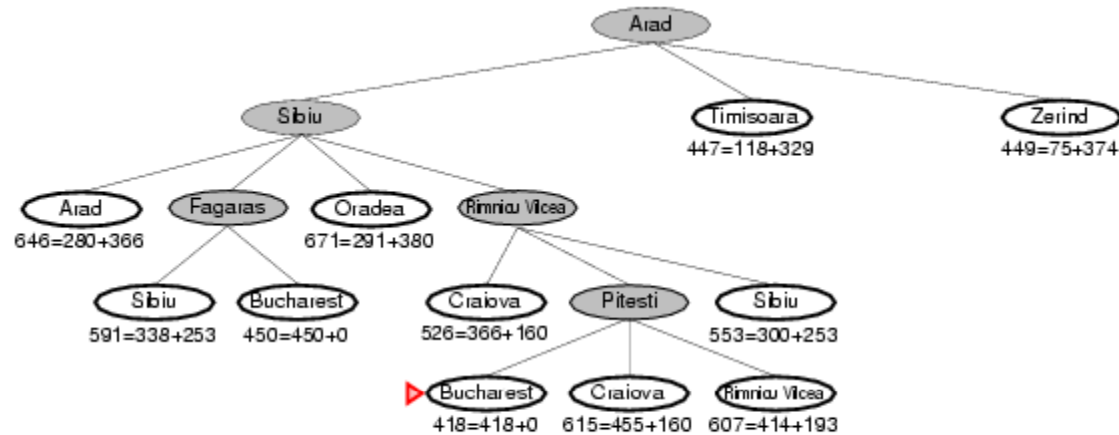
# A\* search example



Straight-line distance to Bucharest

<b>Arad</b>	366
<b>Bucharest</b>	0
<b>Craiova</b>	160
<b>Dobreta</b>	242
<b>Eforie</b>	161
<b>Fagaras</b>	176
<b>Giurgiu</b>	77
<b>Hirsova</b>	151
<b>Iasi</b>	226
<b>Lugoj</b>	244
<b>Mehadia</b>	241
<b>Neamt</b>	234
<b>Oradea</b>	380
<b>Pitesti</b>	10
<b>Rimnicu Vilcea</b>	193
<b>Sibiu</b>	253
<b>Timisoara</b>	329
<b>Urziceni</b>	80
<b>Vaslui</b>	199
<b>Zerind</b>	374

# A\* search example



Straight-line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

**THANK YOU**