Kecerdasan Buatan JTI POLINEMA

Uninformed Search

Team Teaching Kecerdasan Buatan 2020

Capaian Pembelajaran:

Dengan mempelajari materi strategi pencarian, mahasiswa dapat :

- Memahami konsep strategi pencarian.
- Memahami manfaat strategi pencarian.
- Membangun solusi dari permasalahan yang ada melalui penerapan strategi pencarian.

Problem solving agents

- Merupakan salah satu agen yang berbasis tujuan (goal based)
- Memutuskan apa yang harus dilakukan dengan mencari urutan tindakan yang mengarah pada keadaaan (states) yang diinginkan.

Aksi problem solving agent

- Formulasi Tujuan
- Perumusan masalah
- Pencarian
- Penemuan Solusi
- Eksekusi

Searching

- Searching merupakan mekanisme pemecahan masalah yang memiliki urutan langkah-langkah yang dibutuhkan untuk memperoleh solusi pencarian suatu kondisi dengan mengidentifikasikan proses try and error secara sistematis pada eksplorasi setiap alternatif jalur yang ada.

Searching

- Searching berarti usaha untuk:
  - Menemukan solusi
  - Menemukan jawaban dari pertanyaan (Search Engine)
  - Menemukan jalur yang benar (Pathfinding)
  - Menemukan Langkah terbaik untuk bergerak (Permainan Catur)
  - dst

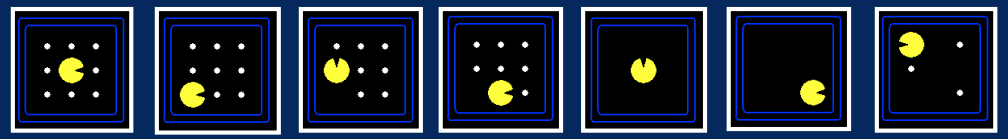
Agen Penyelesaian Problem

- Percept
  - persepsi yang ada
- Seq
  - urutan tindakan
- State
  - deskripsi dari keadaan lingkungan sekitar
- Goal
  - tujuan yang dicapai
- Problem
  - perumusan masalah

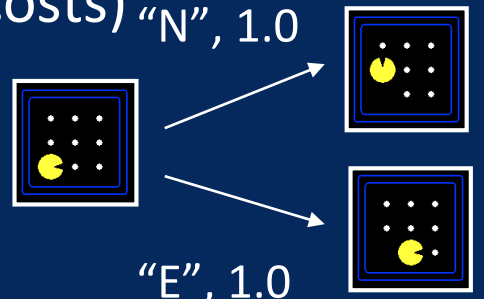
```
function SIMPLE-PROBLEM-SOLVING-AGENT( percept) returns an action
  static: seq, an action sequence, initially empty
          state, some description of the current world state
          goal, a goal, initially null
          problem, a problem formulation
  state ← UPDATE-STATE(state, percept)
  if seq is empty then do
      goal ← FORMULATE-GOAL(state)
      problem ← FORMULATE-PROBLEM(state, goal)
      seq ← SEARCH( problem)
  action ← FIRST(seq)
  seq ← REST(seq)
  return action
```

Ilustrasi permasalahan search

- Permasalahan Search terdiri dari :
  - State Space



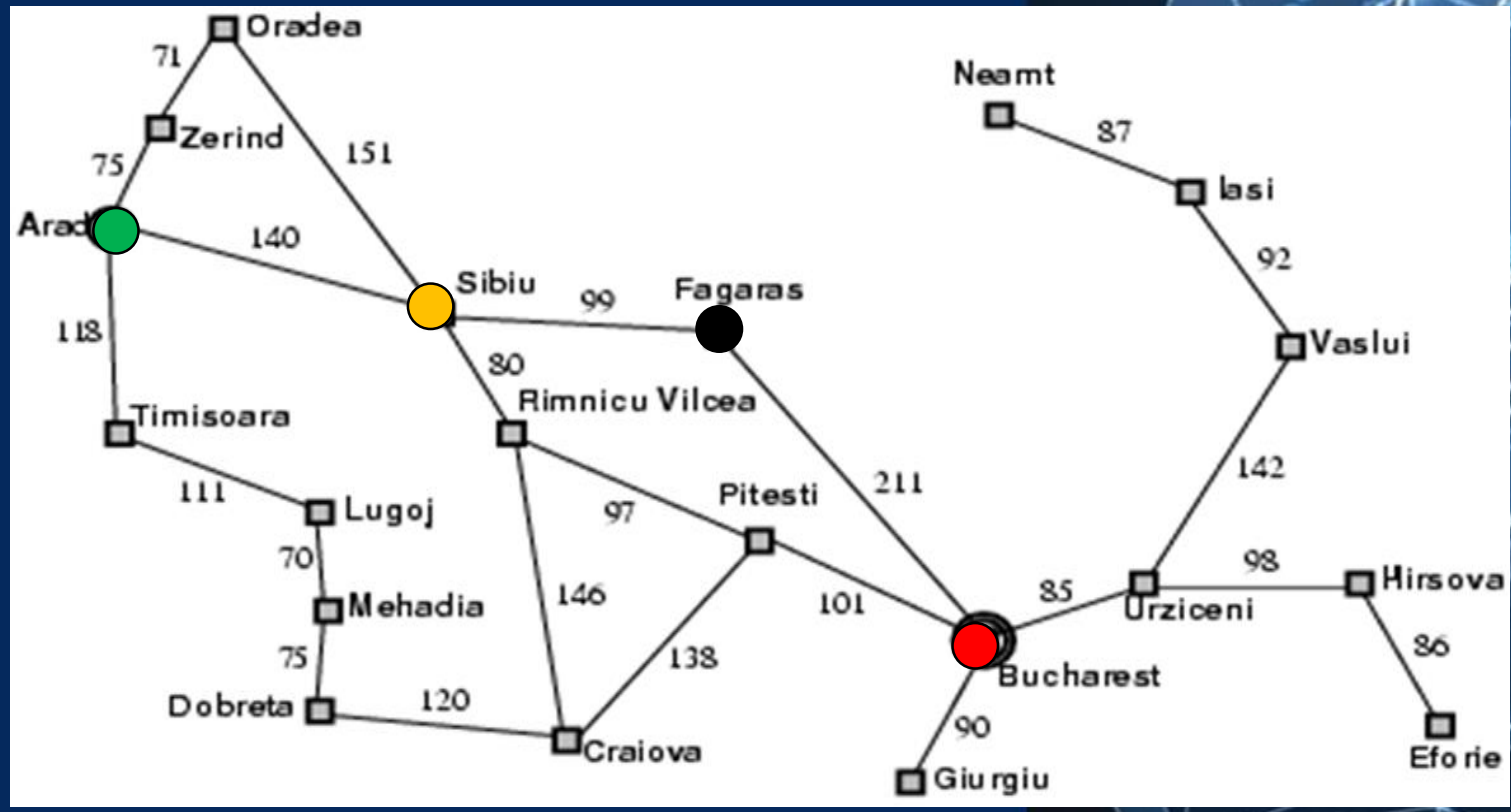
  - A successor function (with actions, costs)



  - A start state and a goal test
- Solusinya berupa urutan aksi yang mulai dari state mulai sampai dengan state selesai

Agen Penyelesaian Problem

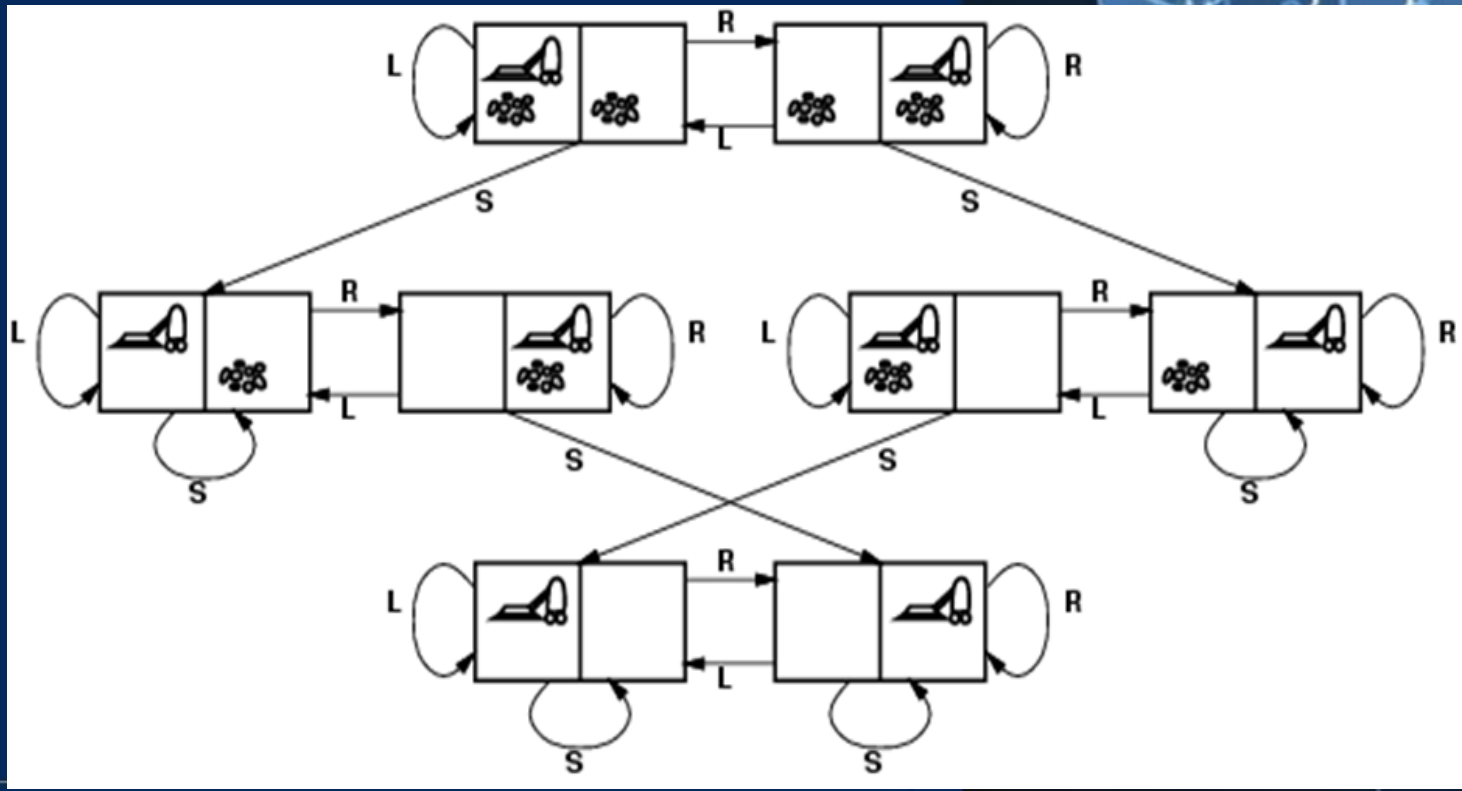
- Contoh : Romania
  - Berlibur ke Rumania, saat ini berada di **Arad**
  - Penerbangan (keberangkatan) dilakukan besok dari **Bucharest**
  - Merumuskan tujuan (Formulate goal)
    - Berada di Bucharest
  - Merumuskan masalah (Formulate problem) :
    - States : berbagai kota sebagai alternatif tempat yang akan dilalui
    - Actions : drive antara kota
  - Cari solusi (Find solution) :
    - Urutan kota yang dilalui untuk mencapai tujuan. Misalnya ; **Arad**, **Sibiu**, **Fagaras**, **Bucharest**



Contoh Problem : Vacuum Cleaner World

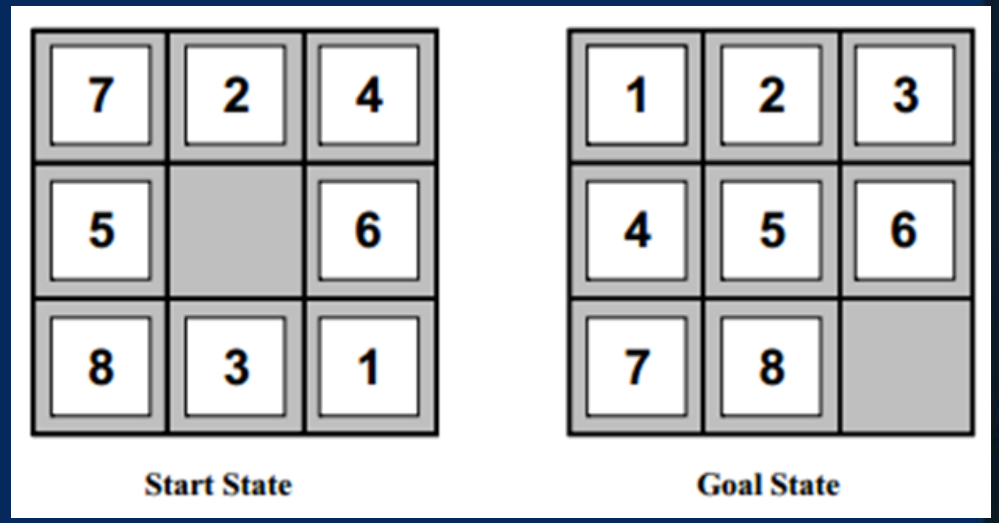
Definisi state space sbb:

- State:
  - lokasi agent
  - status debu.
- Possible action:
  - DoKeKiri(L)
  - DoKeKanan(R)
  - DoSedot(S).
- Goal test:
  - semua ruangan sudah bebas debu.
- Path cost:
  - asumsi step cost sama untuk semua action
  - mis: Path cost = 1 per action.



Contoh Problem: 8-Puzzle

- State:
  - lokasi 8 buah angka dalam matriks 3x3
- Possible action (move, blank) :
  - Left
  - Right
  - Up
  - down
- Goal test:
  - apakah konfigurasi angka sudah seperti goal state.
- Path cost:
  - asumsi, 1 step cost = Path cost = jumlah langkah1 per move.
  - dalam path.



Contoh Problem : 8-Queens Problem

Letakkan 8 bidak menteri (queen!) sedemikian sehingga tidak ada yang saling “makan” (menteri bisa makan dalam satu baris, kolom, diagonal).

- **State**: Papan catur dengan n bidak menteri, $0 \leq n \leq 8$.
- **Initial state**: Papan catur yang kosong.
- **Possible action**: Letakkan sebuah bidak menteri di posisi kosong.
- **Goal test**: 8 bidak menteri di papan, tidak ada yang saling makan.
- **Note** : Formulasi yang lebih baik akan melarang menempatkan queen dalam setiap persegi yang sudah diserang.

Algoritma Pencarian Dasar

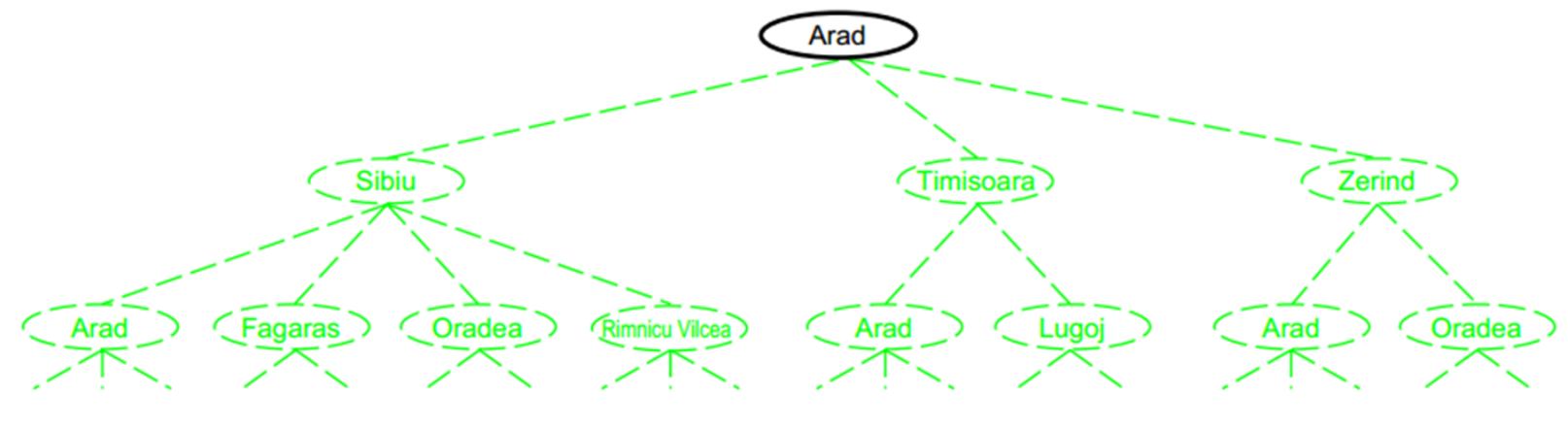
Tree search algorithms :

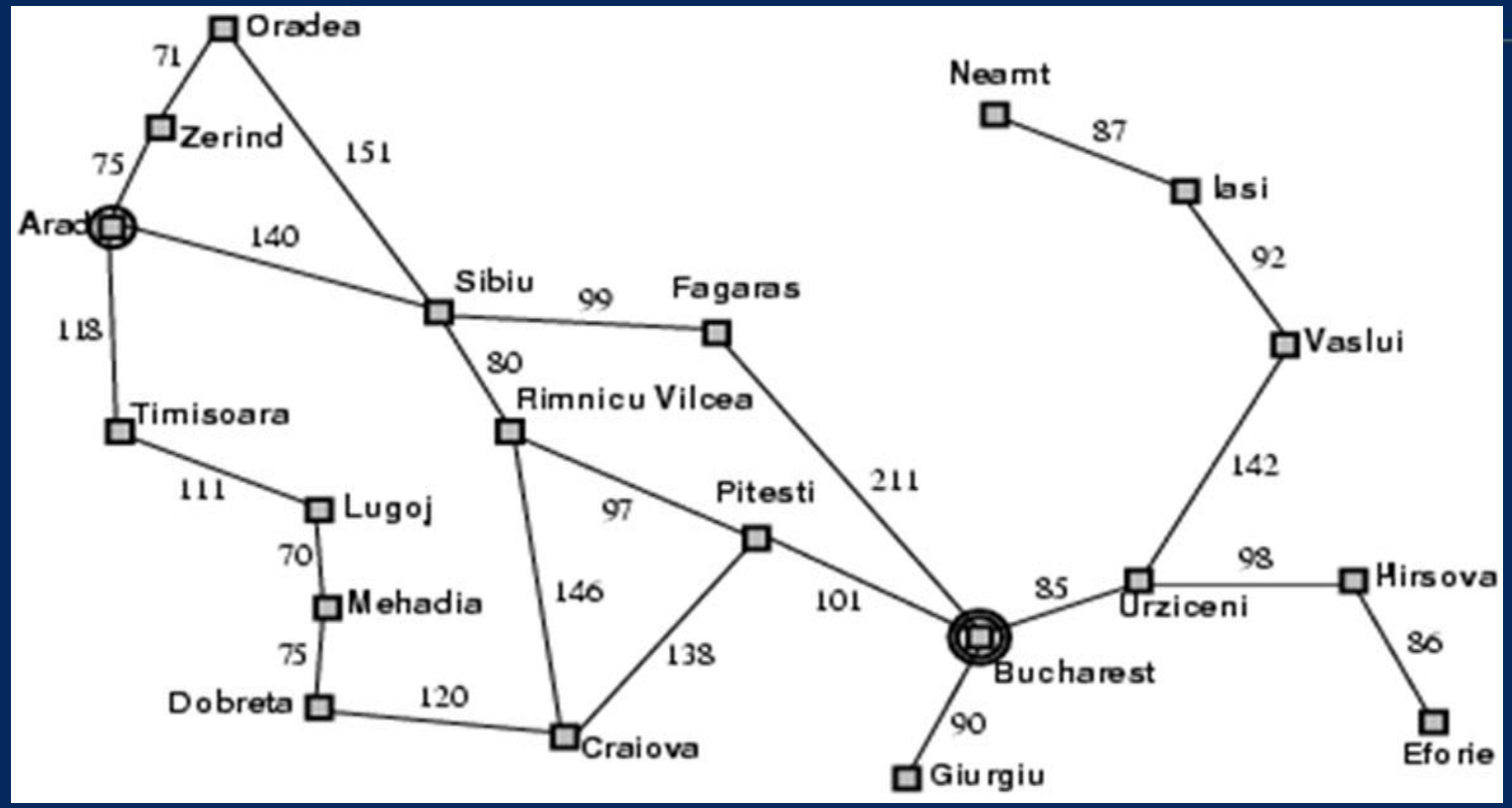
- Setelah merumuskan masalah → cari solusinya menggunakan sebuah search algorithm
- Search tree merepresentasikan state space.
- Search tree terdiri dari kumpulan node: struktur data yang merepresentasikan suatu state pada suatu path, dan memiliki parent, children, depth, dan path cost.
- Root node merepresentasikan initial state.
- Penerapan successor function terhadap (state yang diwakili) node menghasilkan children baru → ini disebut node expansion.
- Kumpulan semua node yang belum di-expand disebut fringe (pinggir) sebuah search tree.

Algoritma Pencarian Dasar

Tree search algorithms (Basic idea) :

- Mulai dari root node (Arad) sebagai current node.
- Lakukan node expansion terhadapnya.
- Pilih salah satu node yang di-expand sebagai current node yang baru.
- Ulangi langkah sebelumnya.

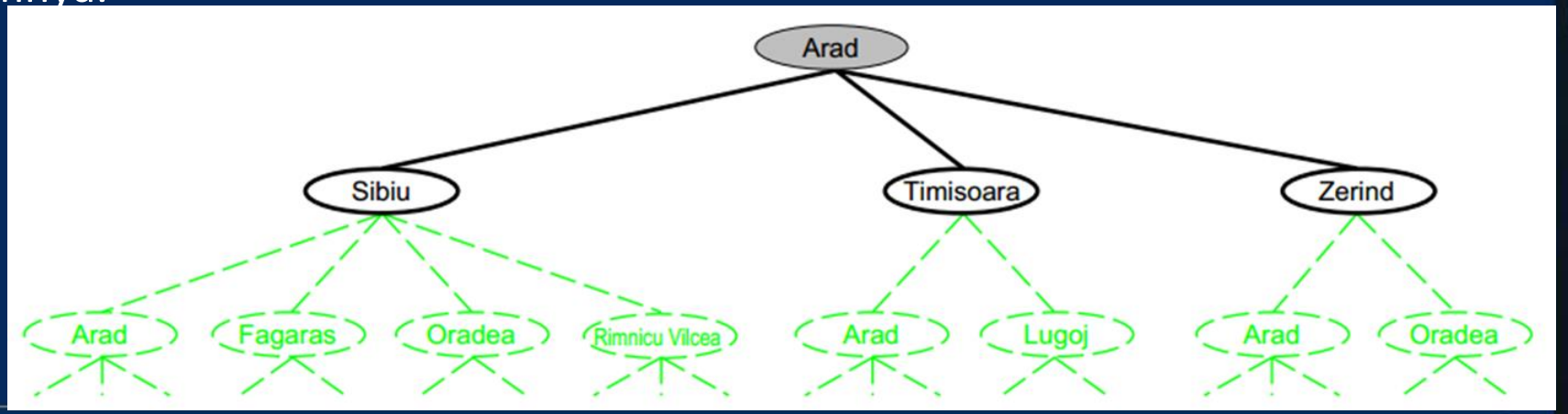



Oradea
71
Neamt
87
Zerind
151
75
Iasi
Arad
140
92
Sibiu
99
Fagaras
118
Vaslui
80
Rimnicu Vilcea
Timisoara
142
111
Pitesti
211
Lugoj
97
98
70
Hirsova
Mehadia
146
101
85
Urziceni
75
86
138
Bucharest
Dobreta
120
Craiova
90
Eforie
Giurgiu

Algoritma Pencarian Dasar

Tree search algorithms (Basic idea) :

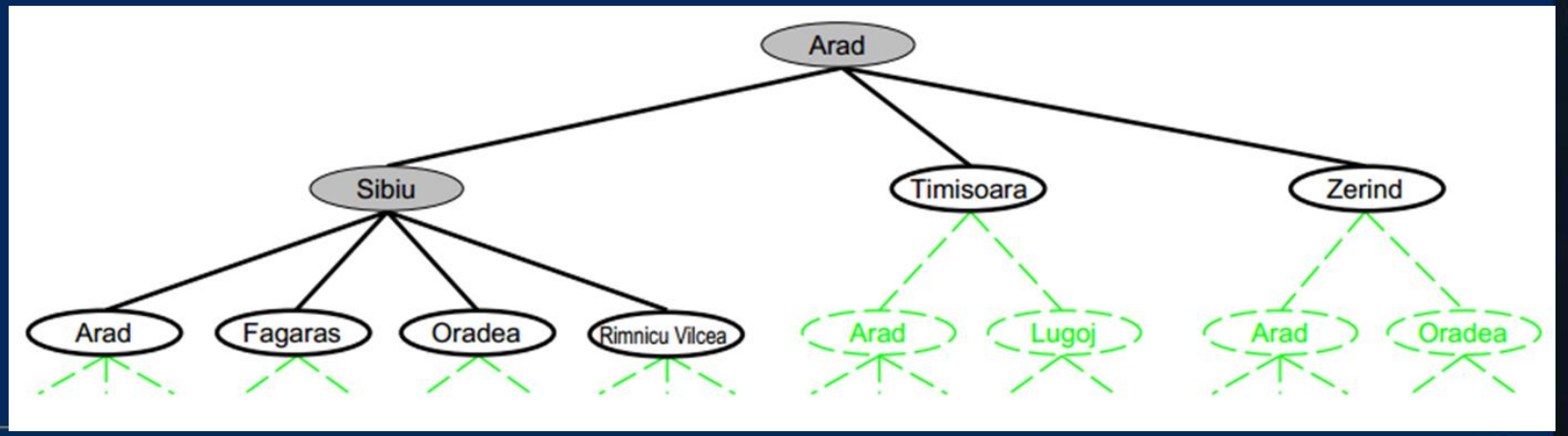
- Mulai dari root node (Arad) sebagai current node.
- Lakukan node expansion terhadapnya.
- Pilih salah satu node yang di-expand sebagai current node yang baru. Ulangi langkah sebelumnya.



Algoritma Pencarian Dasar

Tree search algorithms (Basic idea) :

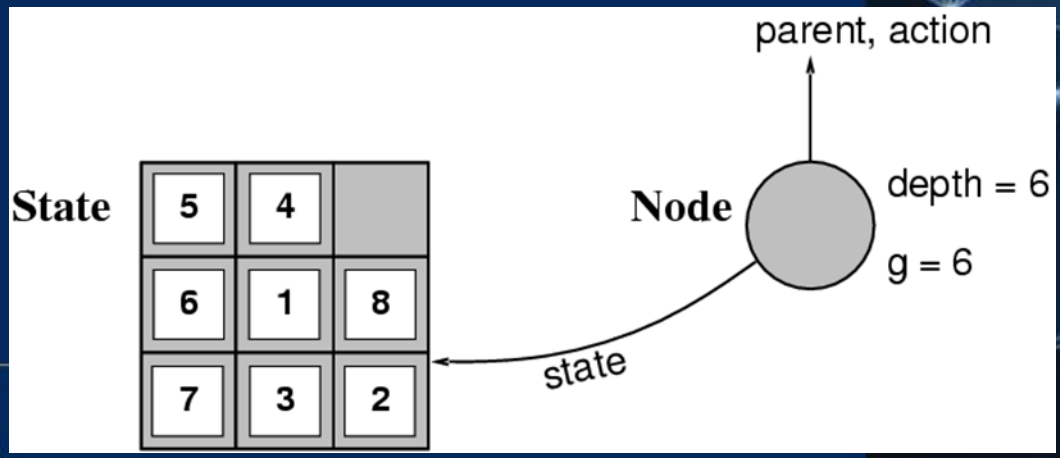
- Mulai dari root node (Arad) sebagai current node.
- Lakukan node expansion terhadapnya.
- Pilih salah satu node yang di-expand sebagai current node yang baru. Ulangi langkah sebelumnya.

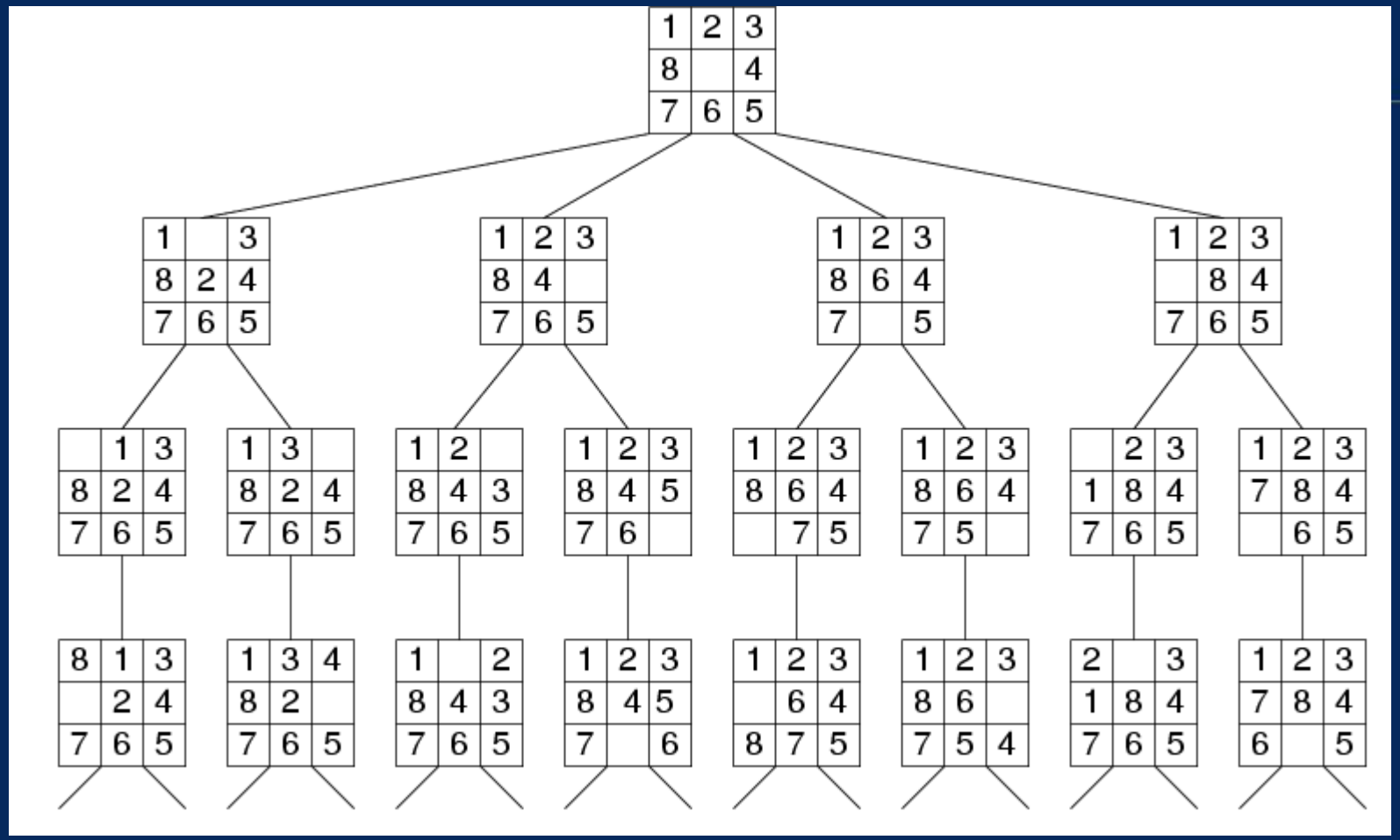


Algoritma Pencarian Dasar

Implementation: states vs. nodes

- Sebuah state merepresentasikan abstraksi keadaan nyata dari masalah.
- Sebuah node adalah struktur data yang menjadi bagian dari search tree.
- State tidak memiliki parent, children, depth, path cost!
- Node = state pada path tertentu. Dua node berbeda bisa mewakili state yang sama!




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

Metode Searching

Metode searching dibedakan menjadi 2 jenis :

- Uninformed Search :
  - Pencarian buta/tanpa informasi
- Informed Search
  - Pencarian dengan algoritma pengetahuan yang spesifik dari suatu permasalahan

Strategi Pencarian Uninformed

- Terdapat berbagai jenis strategi untuk melakukan search.
- Semua strategi ini berbeda dalam satu hal: urutan dari node expansion.
- Search strategy di-evaluasi berdasarkan:
  - completeness: apakah solusi (jika ada) pasti ditemukan?
  - time complexity: jumlah node yang di-expand.
  - space complexity: jumlah maksimum node di dalam memory.
  - optimality: apakah solusi dengan minimum cost pasti ditemukan?
- Time & space complexity diukur berdasarkan
  - b - branching factor dari search tree
  - d - depth (kedalaman) dari solusi optimal
  - m - kedalaman maksimum dari search tree

Strategi Pencarian Uninformed

- Uninformed strategy hanya menggunakan informasi dari definisi masalah.
- Bisa diterapkan secara generik terhadap semua jenis masalah yang bisa direpresentasikan dalam sebuah state space.
- Ada beberapa jenis :
  - **Breadth-first search**
  - Uniform-cost search
  - **Depth-first search**
  - Depth-limited search
  - Iterative-deepening search

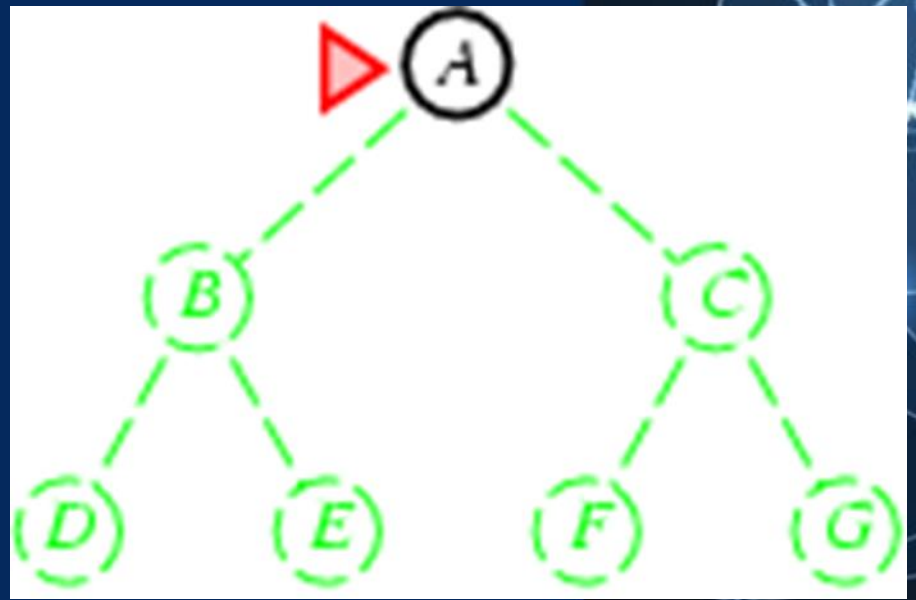
Breadth-First Search (BFS)

- BFS merupakan metode searching berdasarkan algoritma graph
- Metode BFS adalah metode sederhana diawali dari node root yang diperluas kemudian sucecessors node root akan di perluas kemudian, kemudian dilanjutkan dengan successor hasil node yang diperluas dan seterusnya
- Pencarian dilakukan pada semua simpul dalam setiap level secara berurutan **dari** **kiri ke kanan**.
- Jika pada satu level belum ditemukan solusi, maka pencarian dilanjutkan pada level berikutnya dst… sampai solusi terpenuhi
- Membutuhkan memori besar, sulit diimplementasikan pada dunia nyata.

Breadth-First Search

- Lakukan node expansion terhadap node di-fringe yang paling dekat ke root
- Implementasi: fringe adalah sebuah queue, data struktur FIFO (First In First Out)
- Hasil node expansion (successor function) ditaruh di belakang

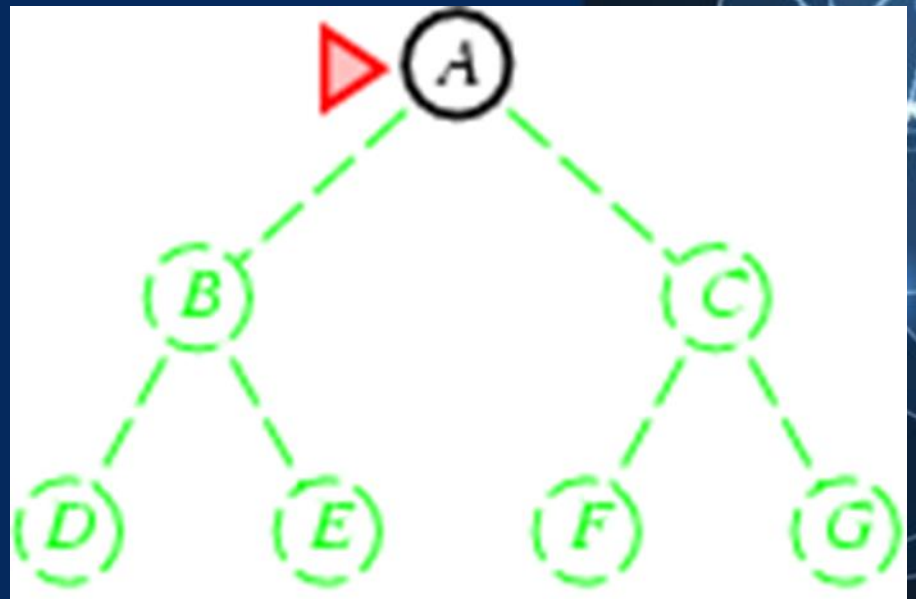
BFS traversal queue:



Breadth-First Search

- Lakukan node expansion terhadap node di-fringe yang paling dekat ke root
- Implementasi: fringe adalah sebuah queue, data struktur FIFO (First In First Out)
- Hasil node expansion (successor function) ditaruh di belakang

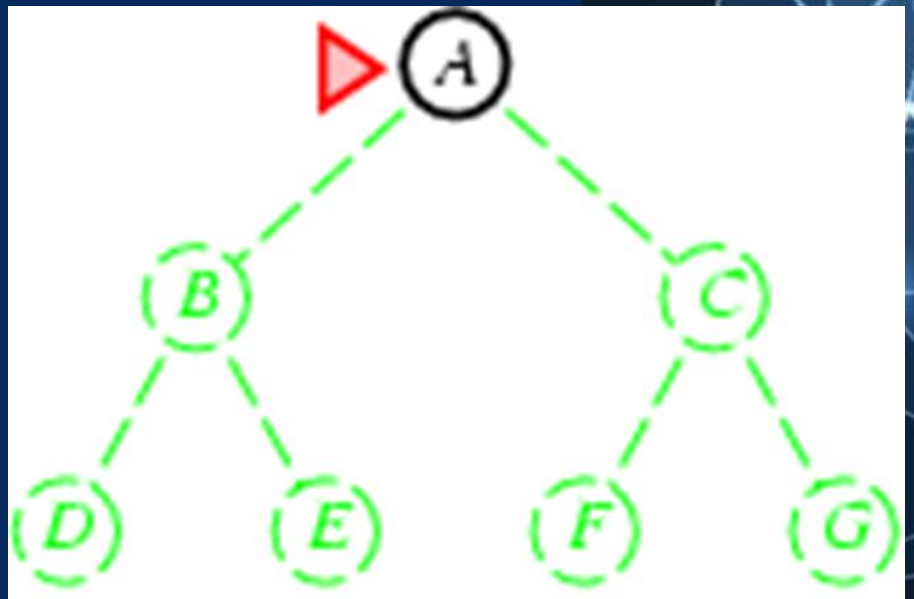
BFS traversal queue:



Breadth-First Search

- Lakukan node expansion terhadap node di-fringe yang paling dekat ke root
- Implementasi: fringe adalah sebuah queue, data struktur FIFO (First In First Out)
- Hasil node expansion (successor function) ditaruh di belakang

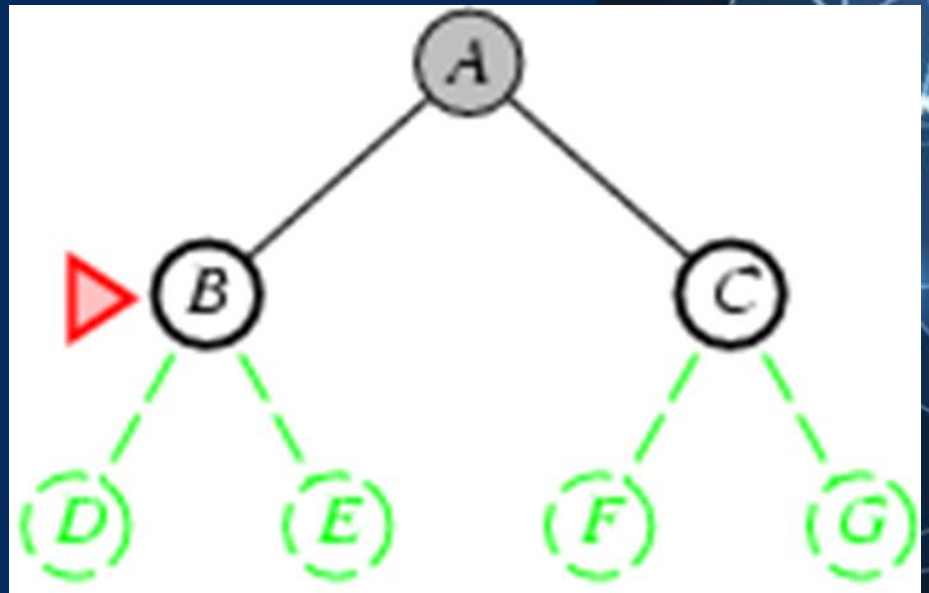
BFS traversal queue:



Breadth-First Search

- Lakukan node expansion terhadap node di-fringe yang paling dekat ke root
- Implementasi: fringe adalah sebuah queue, data struktur FIFO (First In First Out)
- Hasil node expansion (successor function) ditaruh di belakang

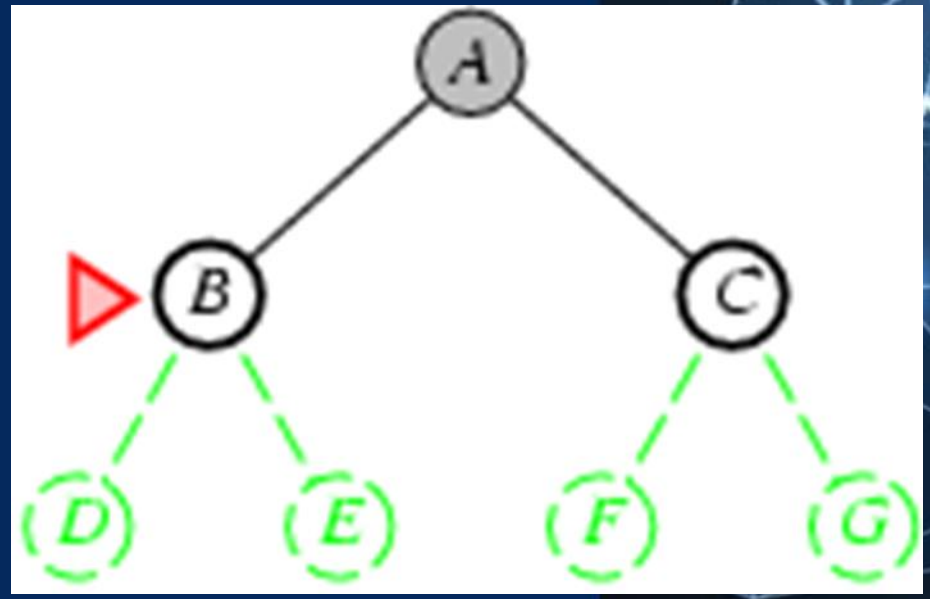
BFS traversal queue:



Breadth-First Search

- Lakukan node expansion terhadap node di-fringe yang paling dekat ke root
- Implementasi: fringe adalah sebuah queue, data struktur FIFO (First In First Out)
- Hasil node expansion (successor function) ditaruh di belakang

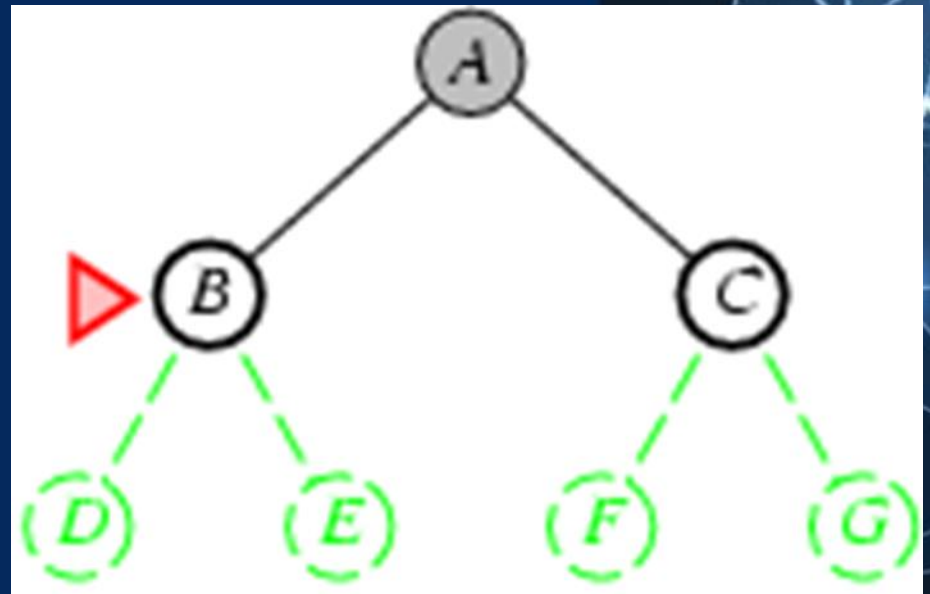
BFS traversal queue:



Breadth-First Search

- Lakukan node expansion terhadap node di-fringe yang paling dekat ke root
- Implementasi: fringe adalah sebuah queue, data struktur FIFO (First In First Out)
- Hasil node expansion (successor function) ditaruh di belakang

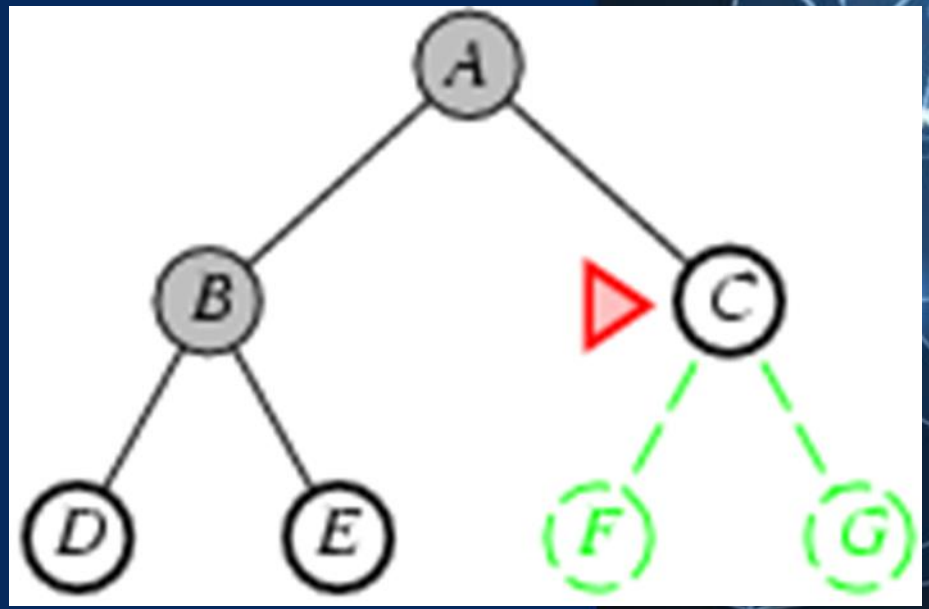
BFS traversal queue:



Breadth-First Search

- Lakukan node expansion terhadap node di-fringe yang paling dekat ke root
- Implementasi: fringe adalah sebuah queue, data struktur FIFO (First In First Out)
- Hasil node expansion (successor function) ditaruh di belakang

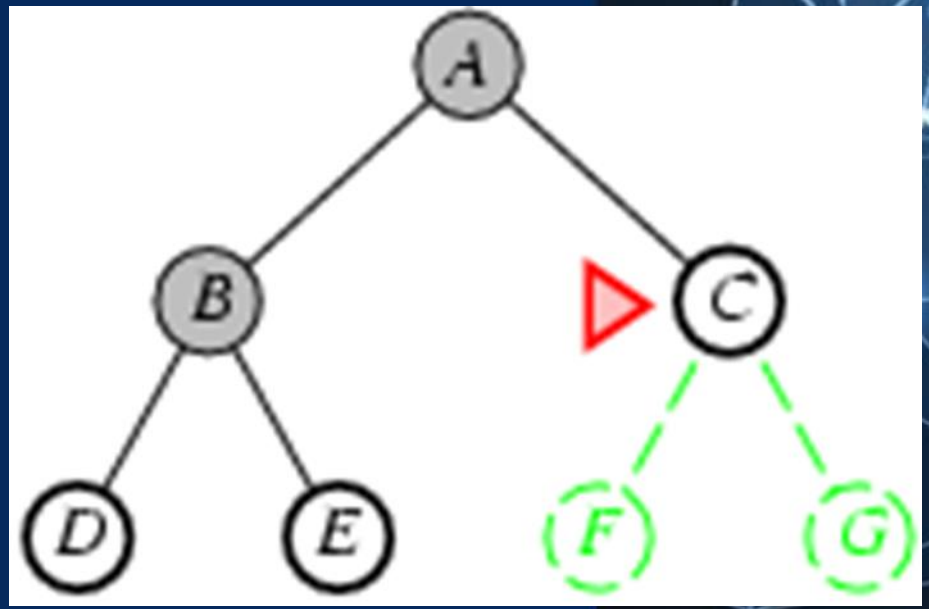
BFS traversal queue:



Breadth-First Search

- Lakukan node expansion terhadap node di-fringe yang paling dekat ke root
- Implementasi: fringe adalah sebuah queue, data struktur FIFO (First In First Out)
- Hasil node expansion (successor function) ditaruh di belakang

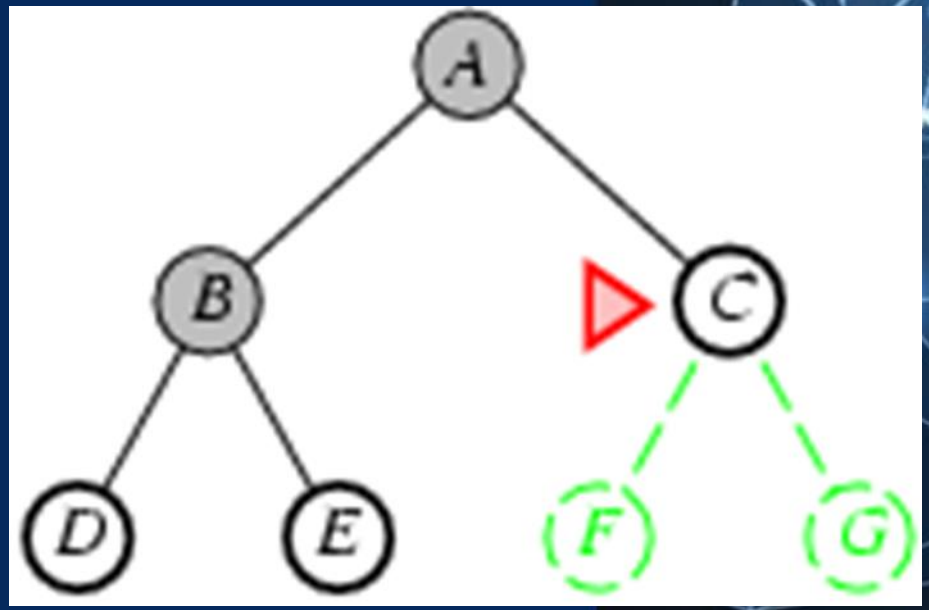
BFS traversal queue:



Breadth-First Search

- Lakukan node expansion terhadap node di-fringe yang paling dekat ke root
- Implementasi: fringe adalah sebuah queue, data struktur FIFO (First In First Out)
- Hasil node expansion (successor function) ditaruh di belakang

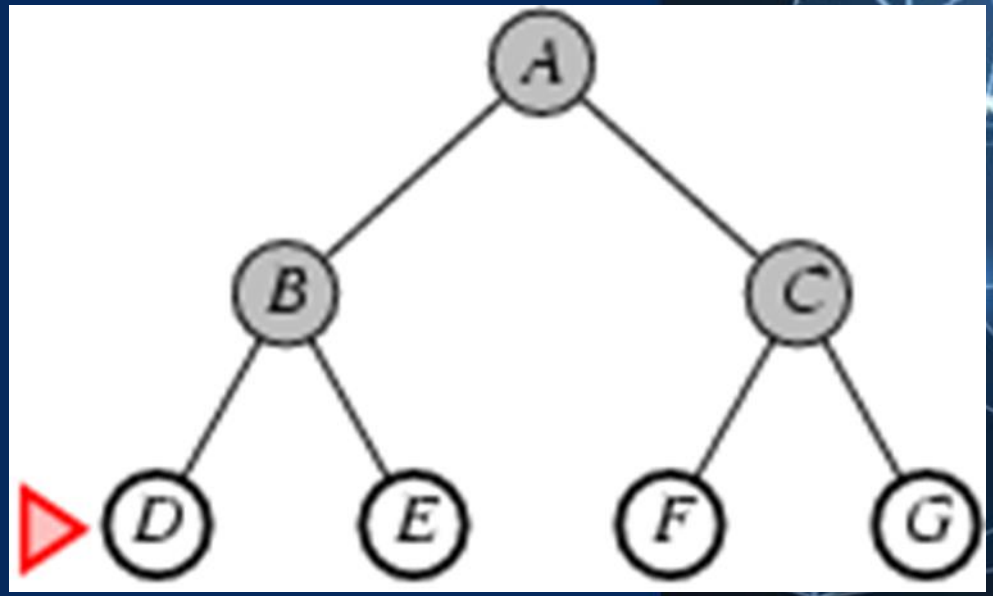
BFS traversal queue:



Breadth-First Search

- Lakukan node expansion terhadap node di-fringe yang paling dekat ke root
- Implementasi: fringe adalah sebuah queue, data struktur FIFO (First In First Out)
- Hasil node expansion (successor function) ditaruh di belakang

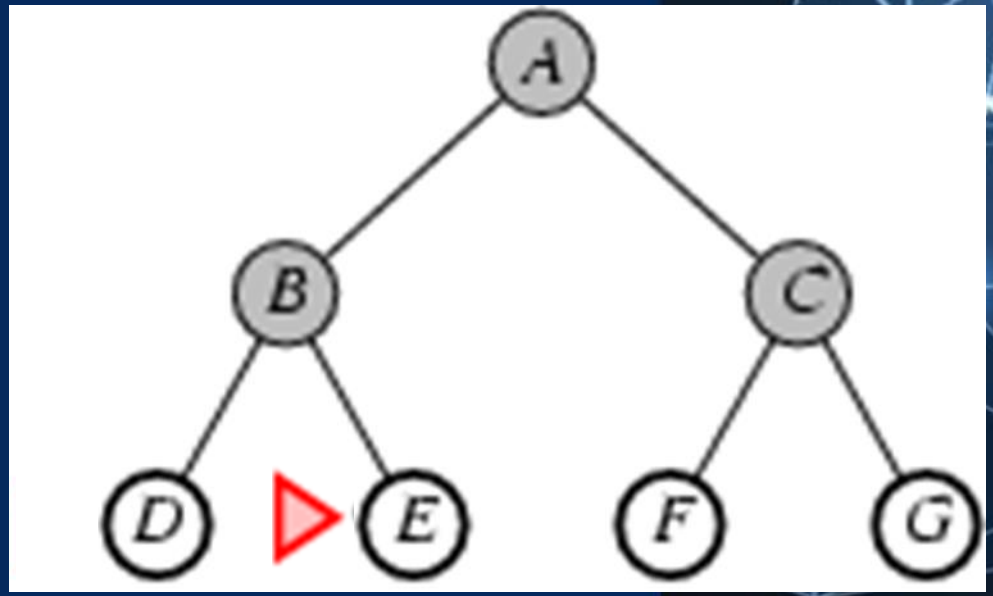
BFS traversal queue:



Breadth-First Search

- Lakukan node expansion terhadap node di-fringe yang paling dekat ke root
- Implementasi: fringe adalah sebuah queue, data struktur FIFO (First In First Out)
- Hasil node expansion (successor function) ditaruh di belakang

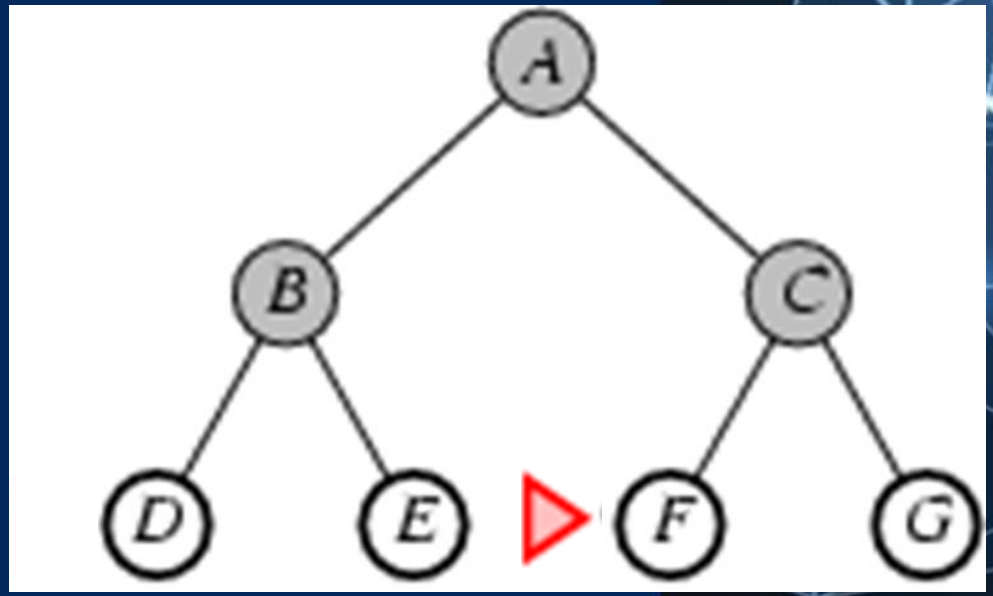
BFS traversal queue:



Breadth-First Search

- Lakukan node expansion terhadap node di-fringe yang paling dekat ke root
- Implementasi: fringe adalah sebuah queue, data struktur FIFO (First In First Out)
- Hasil node expansion (successor function) ditaruh di belakang

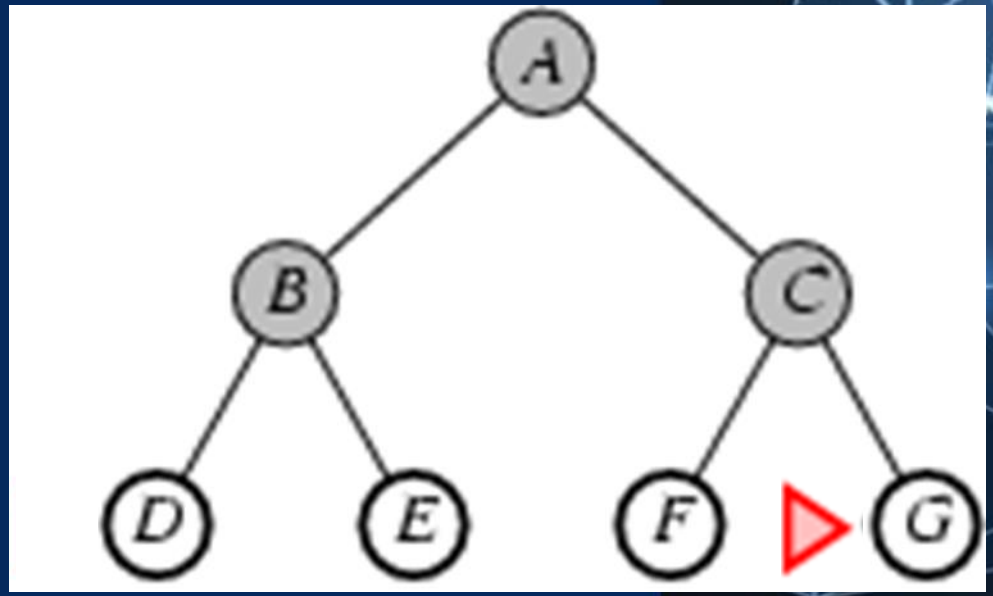
BFS traversal queue:



Breadth-First Search

- Lakukan node expansion terhadap node di-fringe yang paling dekat ke root
- Implementasi: fringe adalah sebuah queue, data struktur FIFO (First In First Out)
- Hasil node expansion (successor function) ditaruh di belakang

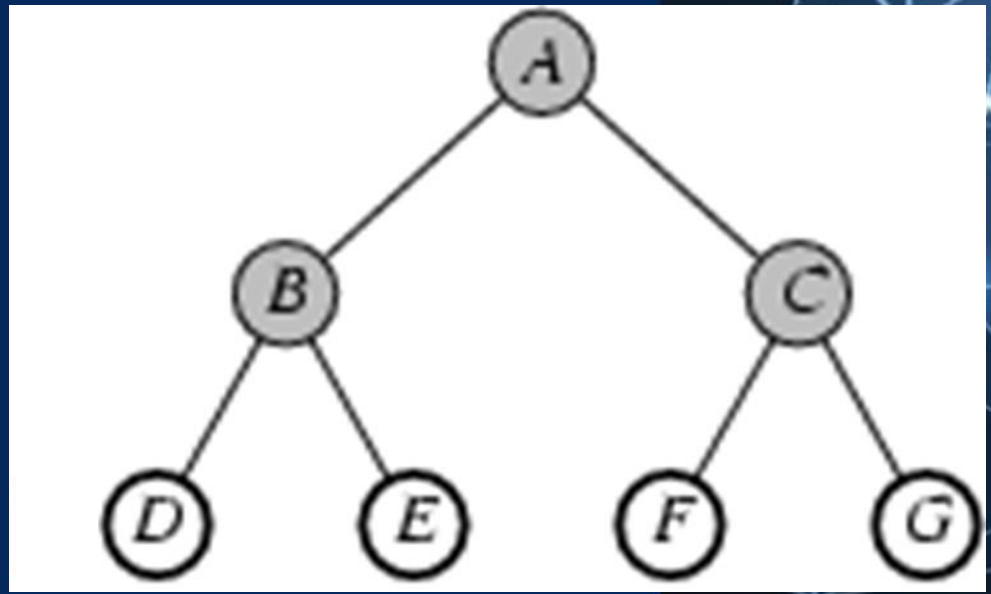
BFS traversal queue:



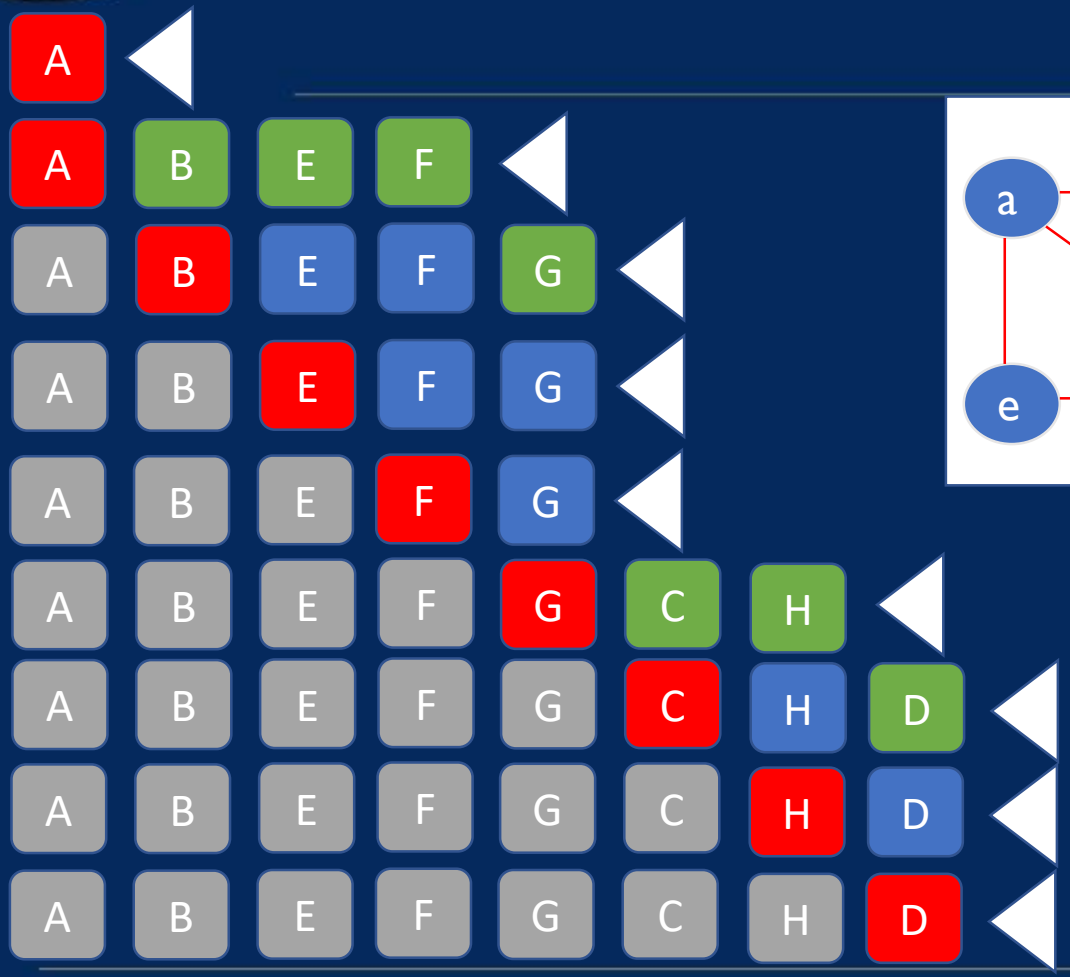
Breadth-First Search

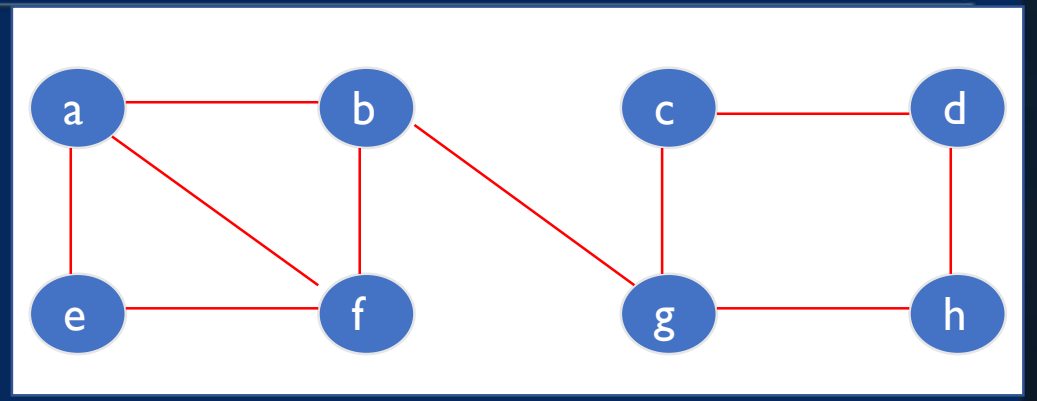
- Lakukan node expansion terhadap node di-fringe yang paling dekat ke root
- Implementasi: fringe adalah sebuah queue, data struktur FIFO (First In First Out)
- Hasil node expansion (successor function) ditaruh di belakang

BFS traversal queue:



Contoh Lain





Successor of

Breadth-First Search

- Properties of breadth-first search :
  - Complete? Ya, jika b terbatas
  - Time complexity? $b + b2 + b3 + \ldots + bd + b(bd - 1) = O(bd + 1)$ → eksponensial dalam d.
  - Space complexity? O(bd +1), karena semua node yang di-generate harus disimpan.
  - Optimal? Ya, jika semua step cost sama, tapi pada umumnya tidak optimal.
  - Masalah utama breadth-first search adalah space :
    - Mis: 1 node memakan 1000 byte, dan b = 10
    - Jika d = 6, ada 107 node ≈ 10 gigabyte.
    - Jika d = 12, ada 1013 node ≈ 10 petabyte!

Depth-First Search (DFS)

- DFS akan memperluas node yang paling dalam terlebih dahulu
- Pencarian dilakukan pada suatu simpul dalam setiap level dari yang paling kiri.
- Jika pada level yang terdalam solusi belum ditemukan, maka pencarian dilanjutkan pada simpul sebelah kanan dan simpul yang kiri dapat dihaps dari memori. Demikian seterusnya sampai ditemukan solusi.
- Kelebihan Pemakaian memori lebih sedikit, jika solusi yang dicari berada pada level yang dalm paling kiri, mka DFS akan menemukannya dengan cepat.

Depth-First Search (DFS)

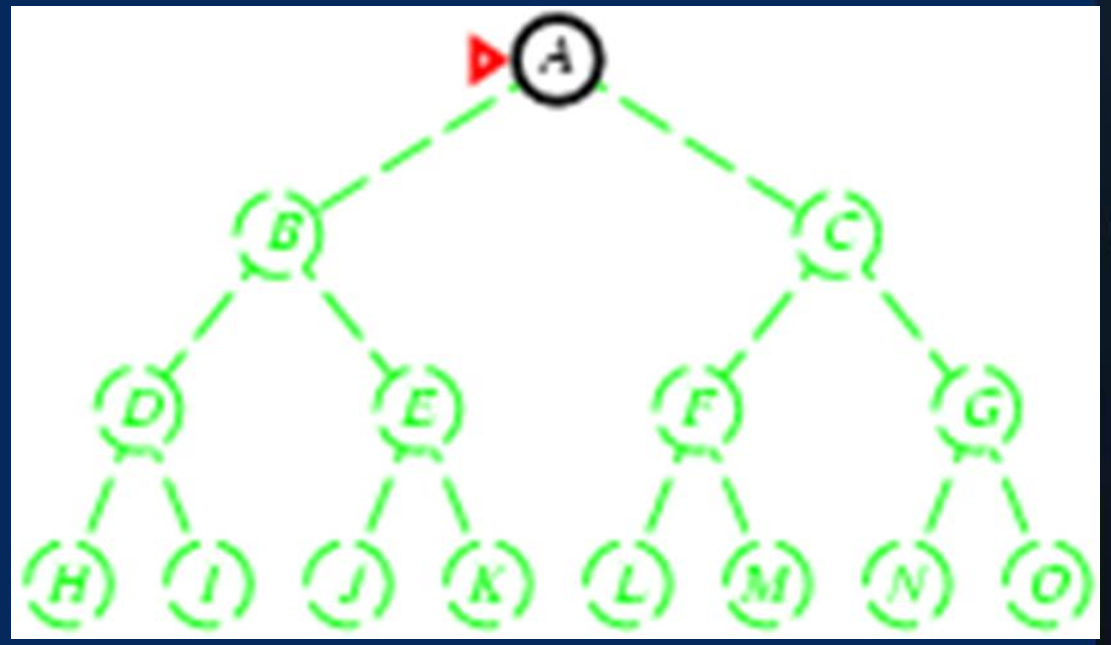
- Lakukan node expansion terhadap node di fringe yang paling jauh dari root.
- Implementasi: fringe adalah sebuah stack, data struktur LIFO (Last In First Out)
- Hasil node expansion ditaruh di depan
- Depth-first search sangat cocok diimplementasikan secara rekursif.

Depth-First Search

DFS traversal stack

Urutan yang ditelusuri:

A

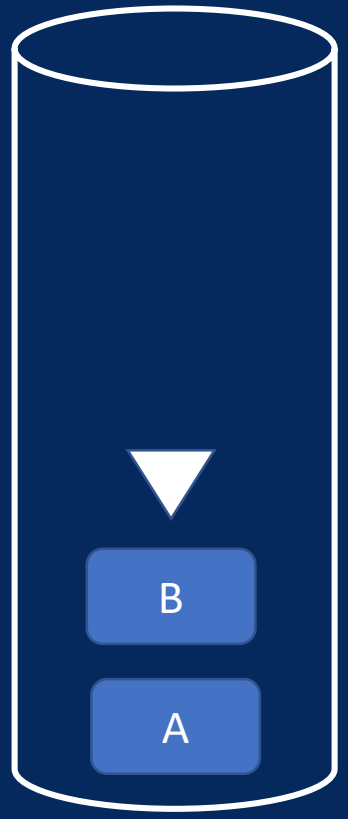


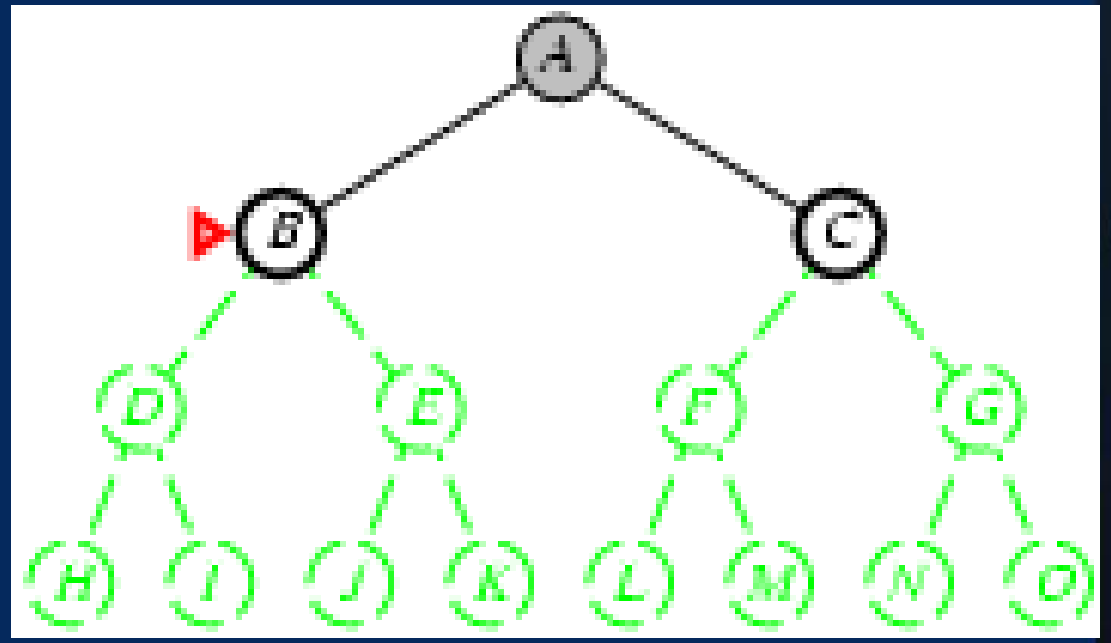
Depth-First Search

DFS traversal stack

Urutan yang ditelusuri:

A B



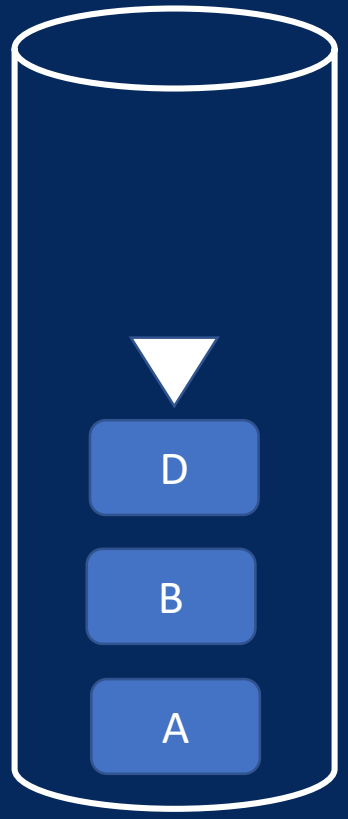


Depth-First Search

DFS traversal stack

Urutan yang ditelusuri:

A B D

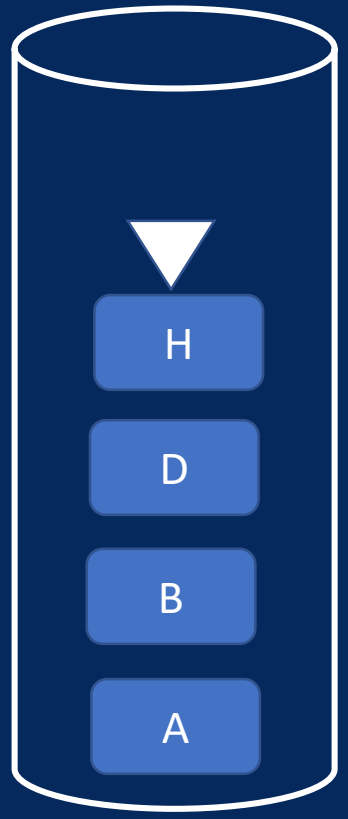


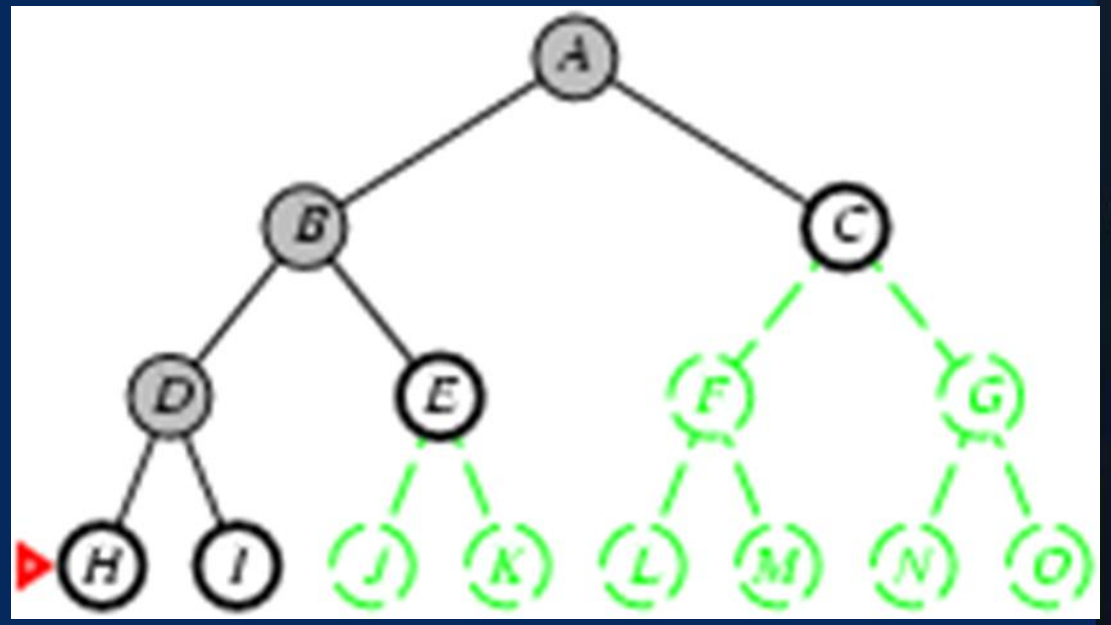
Depth-First Search

DFS traversal stack

Urutan yang ditelusuri:

A B D H



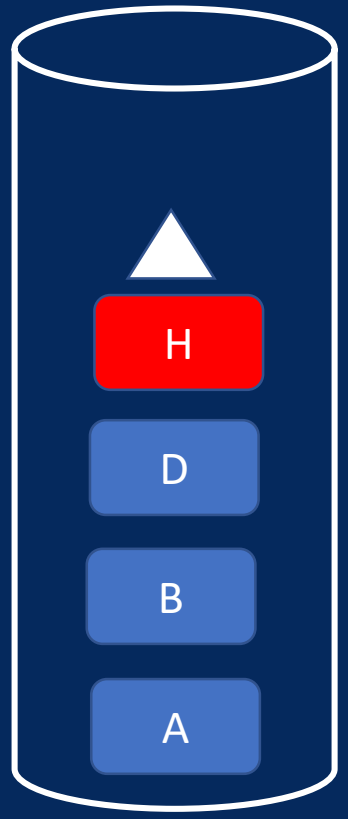


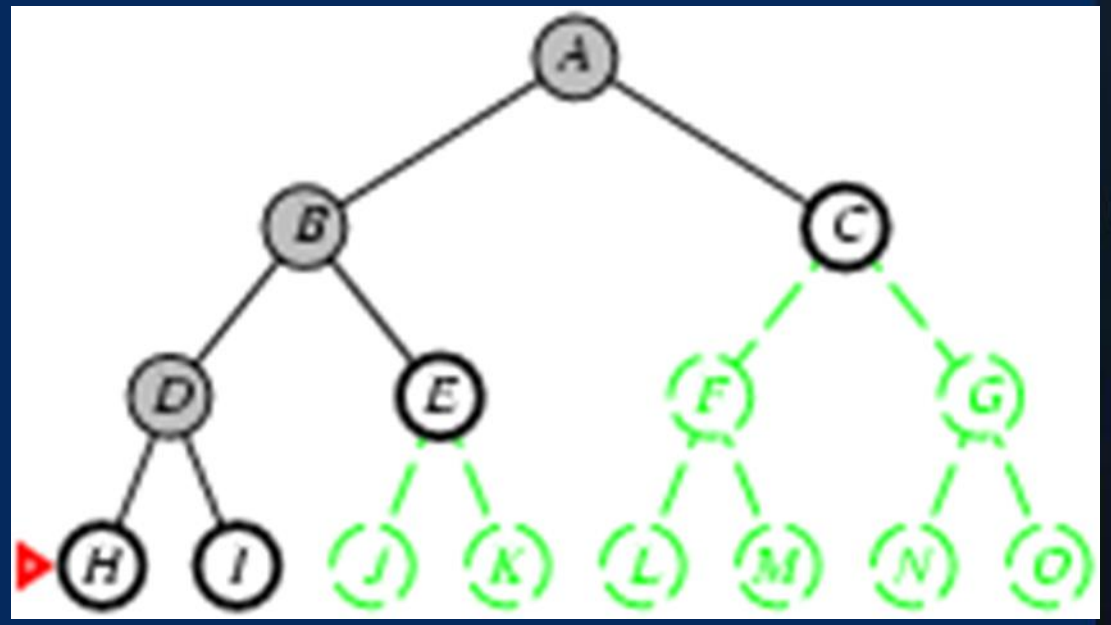
Depth-First Search

DFS traversal stack

Urutan yang ditelusuri:

A B D H



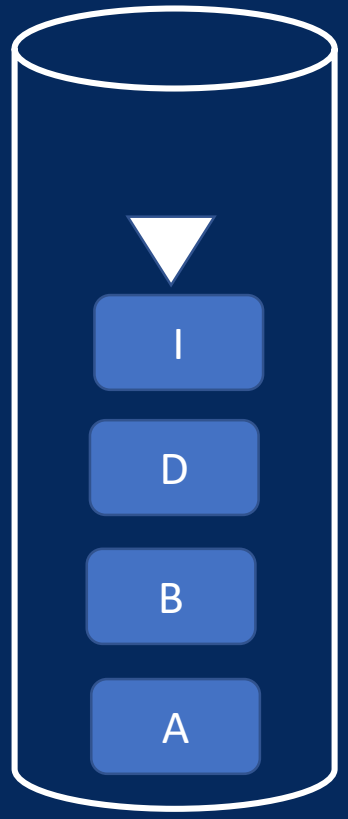


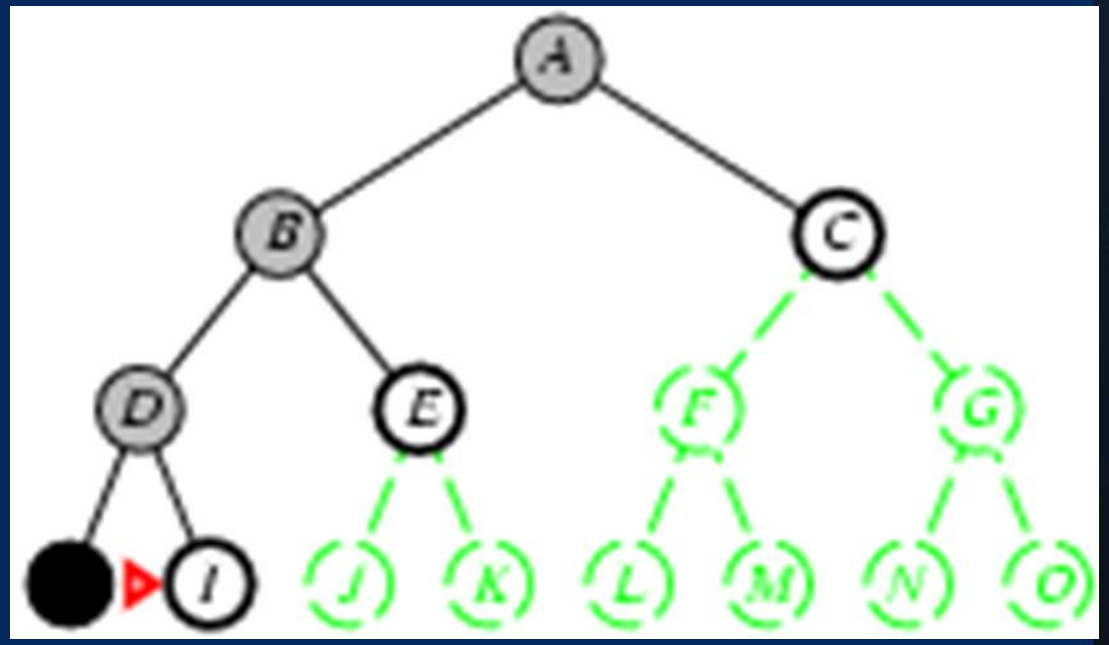
Depth-First Search

DFS traversal stack

Urutan yang ditelusuri:

A B D H I



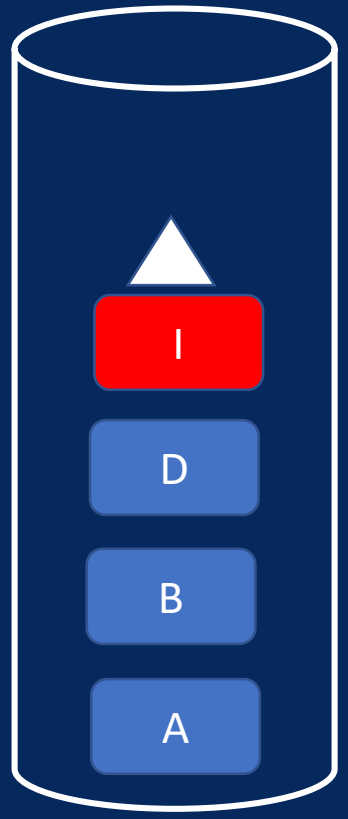


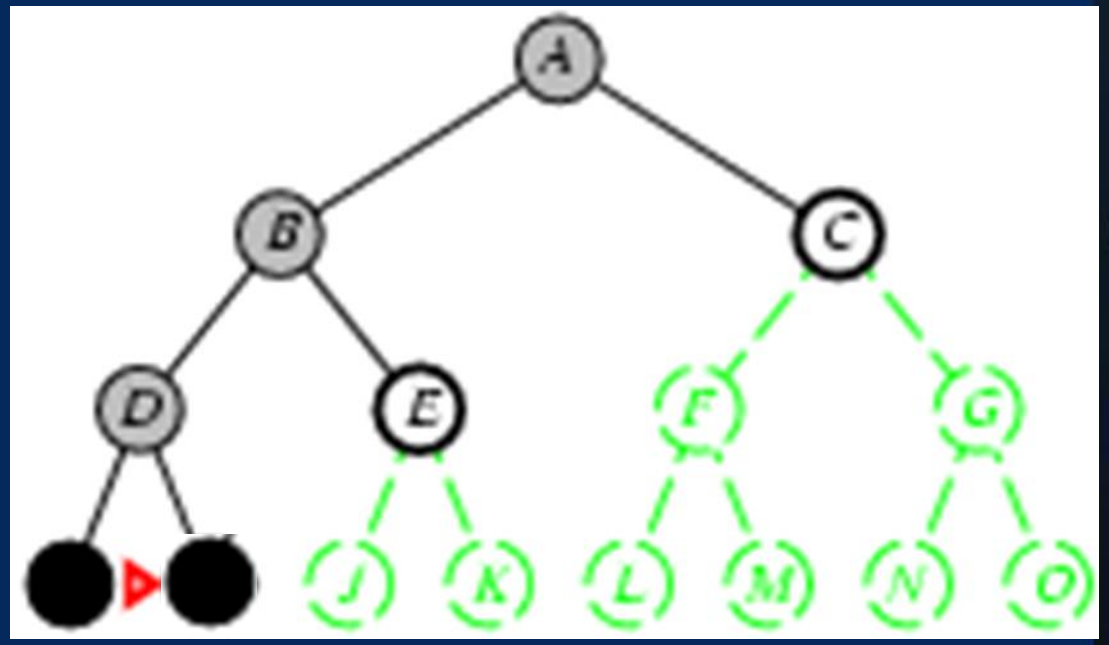
Depth-First Search

DFS traversal stack

Urutan yang ditelusuri:

A B D H I



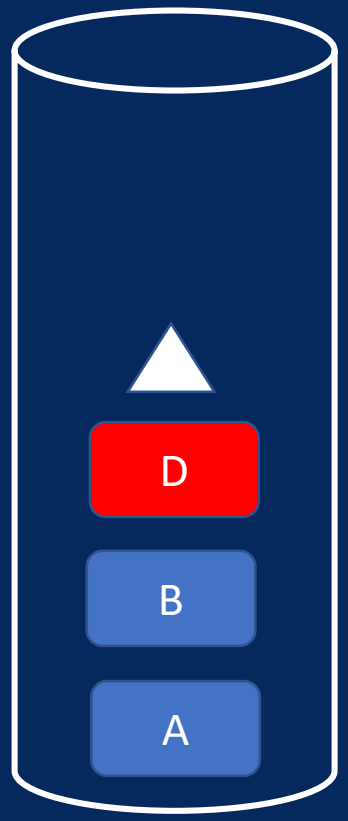


Depth-First Search

DFS traversal stack

Urutan yang ditelusuri:

A B D H I

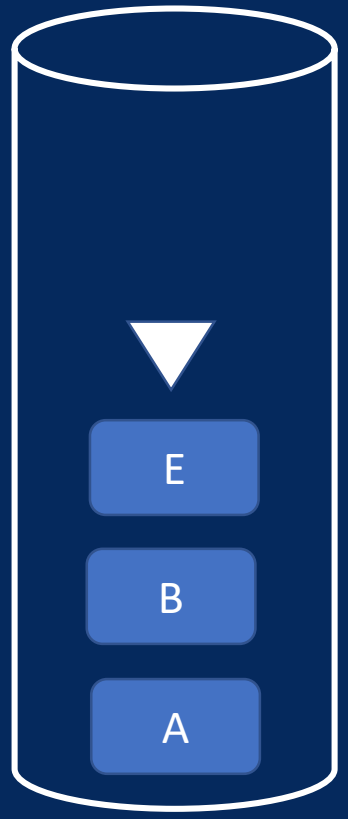


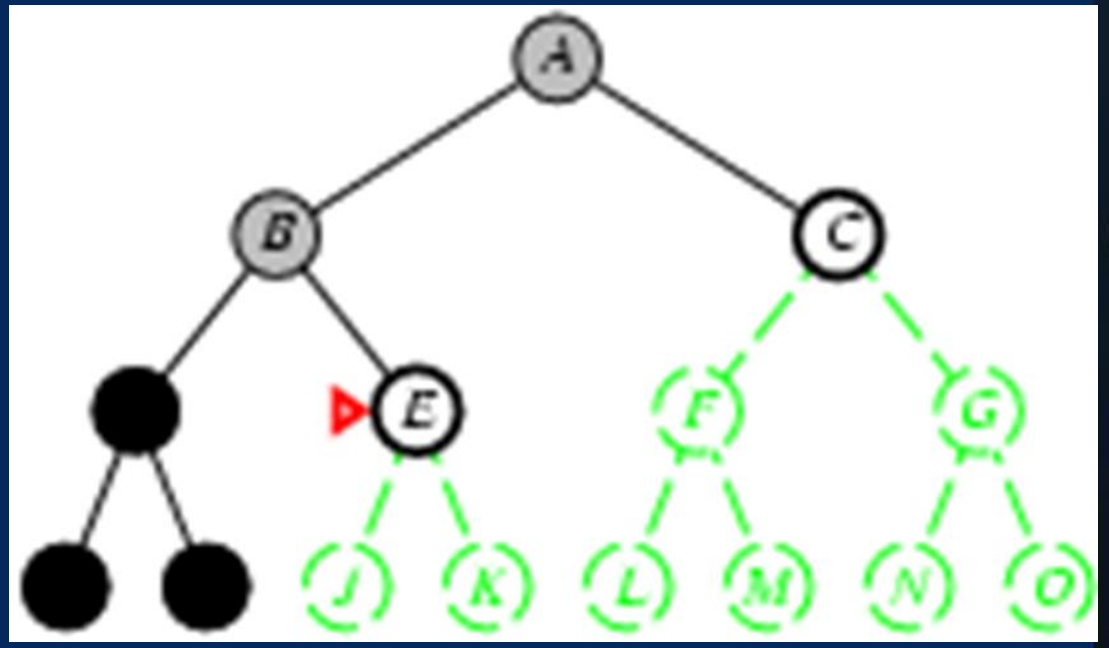
Depth-First Search

DFS traversal stack

Urutan yang ditelusuri:

A B D H I E



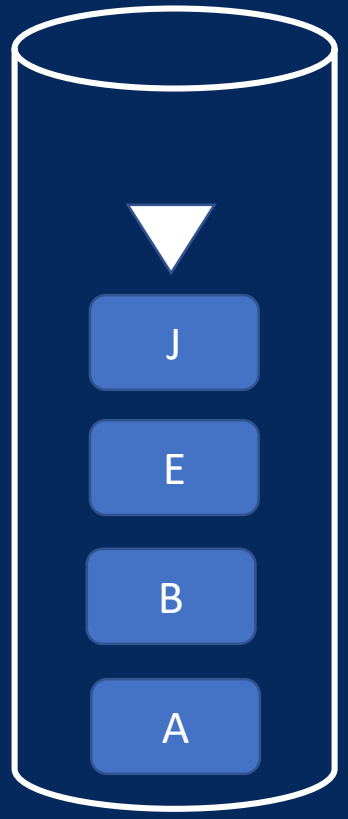


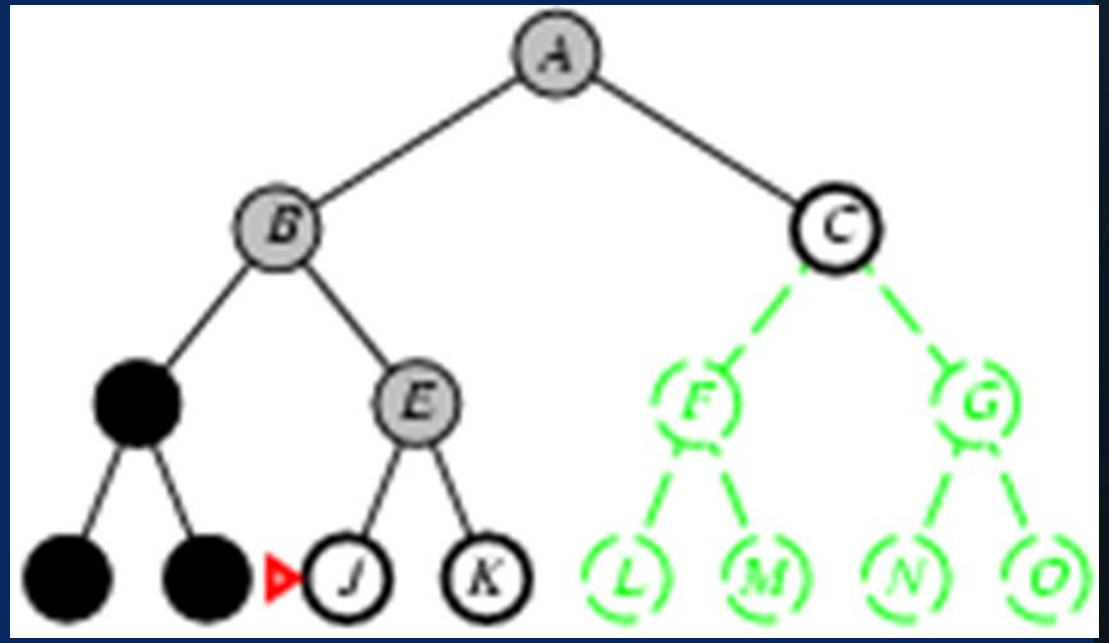
Depth-First Search

DFS traversal stack

Urutan yang ditelusuri:

A B D H I E J



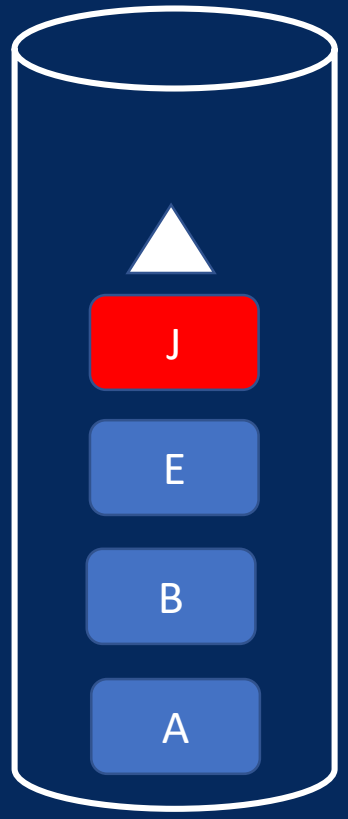


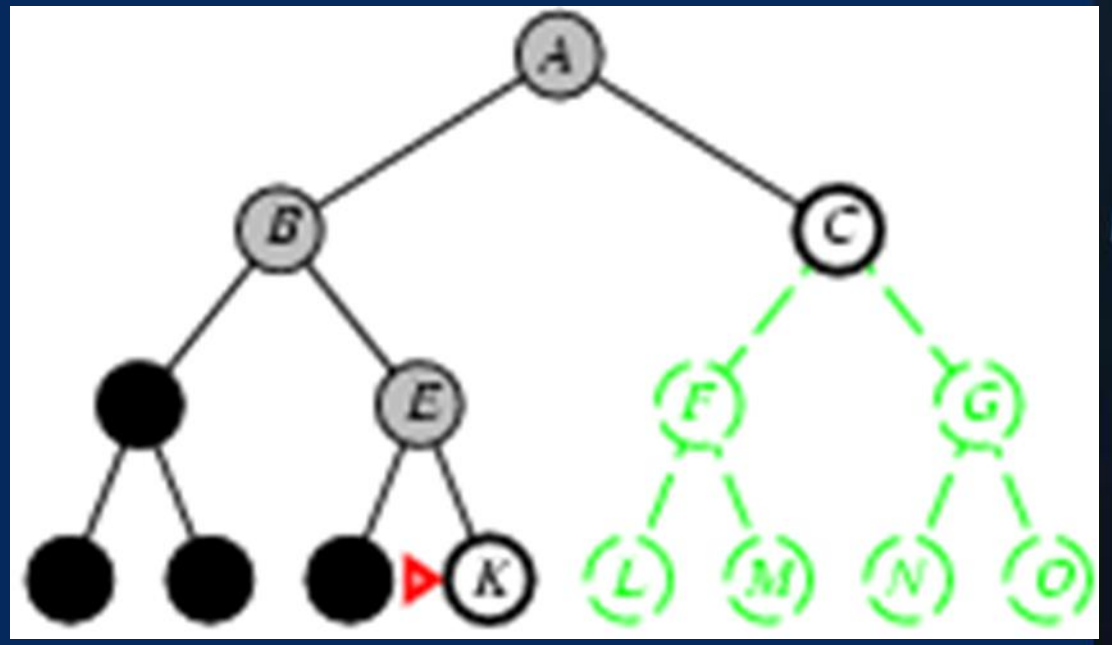
Depth-First Search

DFS traversal stack

Urutan yang ditelusuri:

A B D H I E J



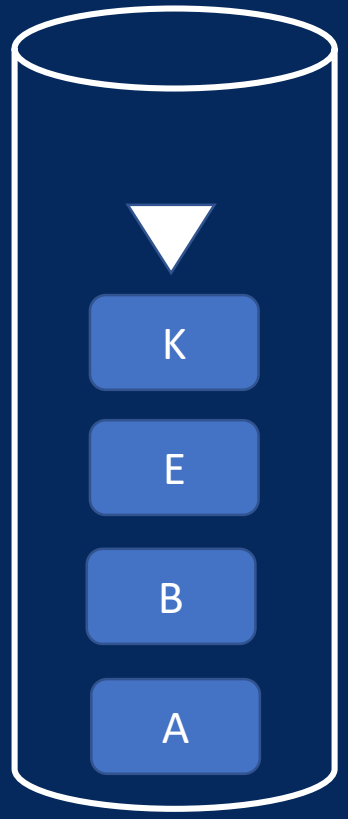


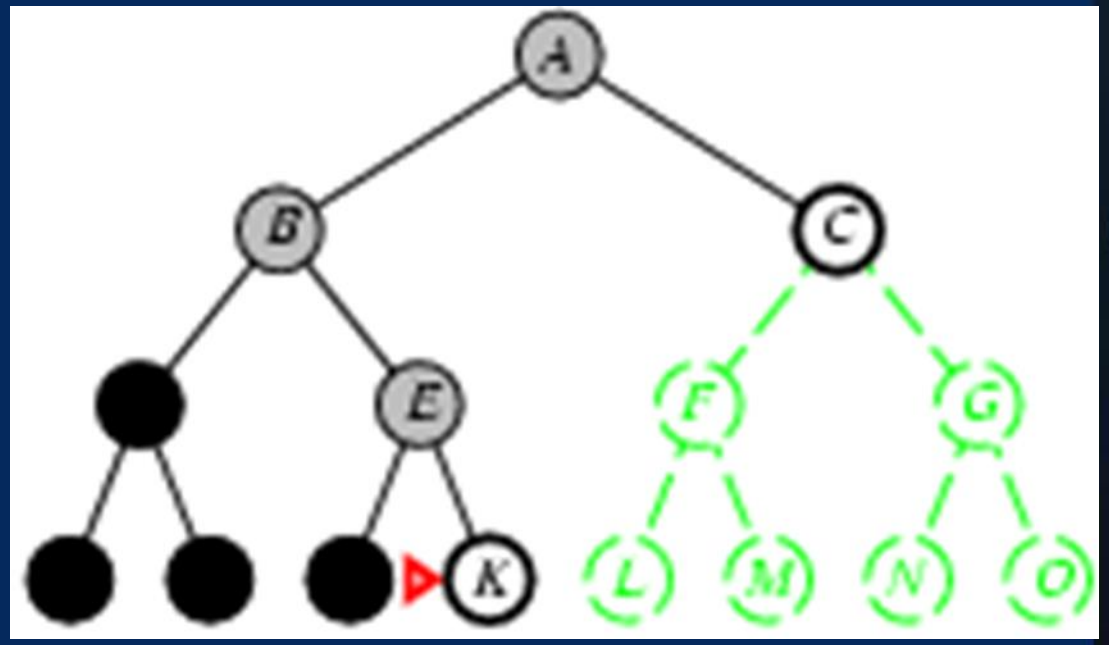
Depth-First Search

DFS traversal stack

Urutan yang ditelusuri:

A B D H I E J K



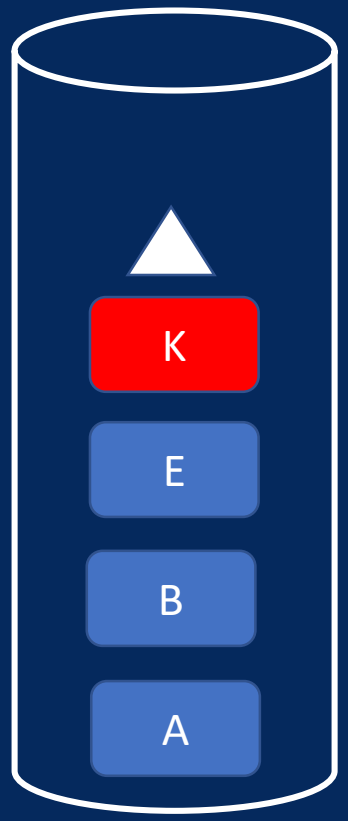


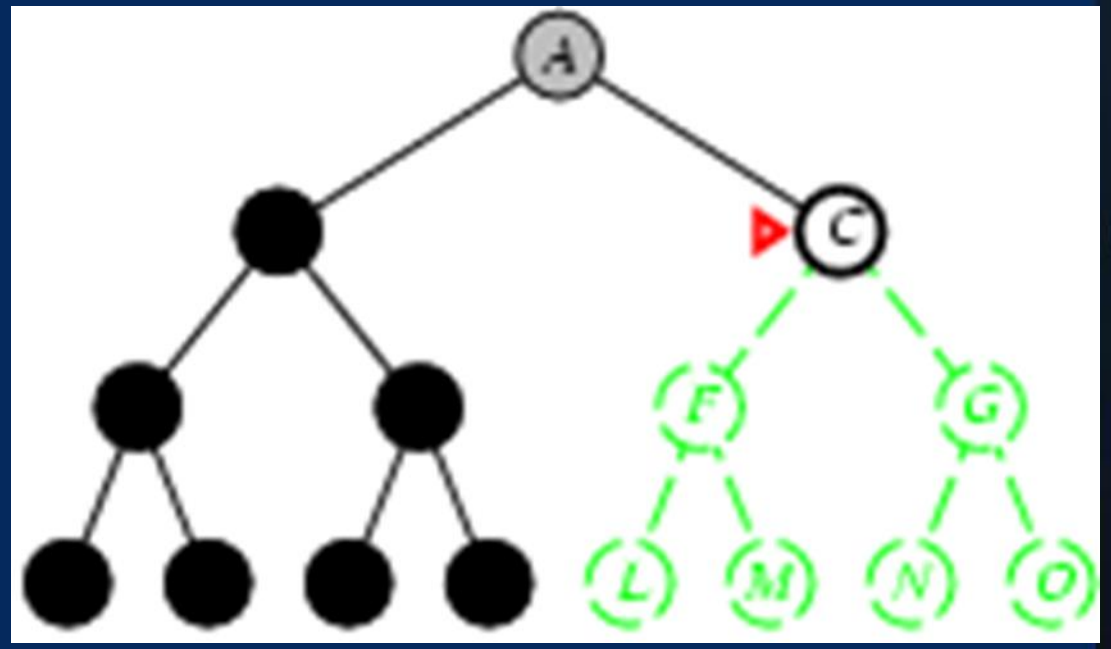
Depth-First Search

DFS traversal stack

Urutan yang ditelusuri:

A B D H I E J K



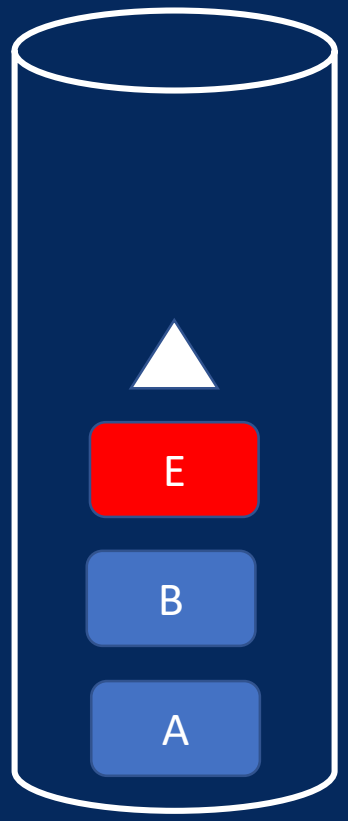


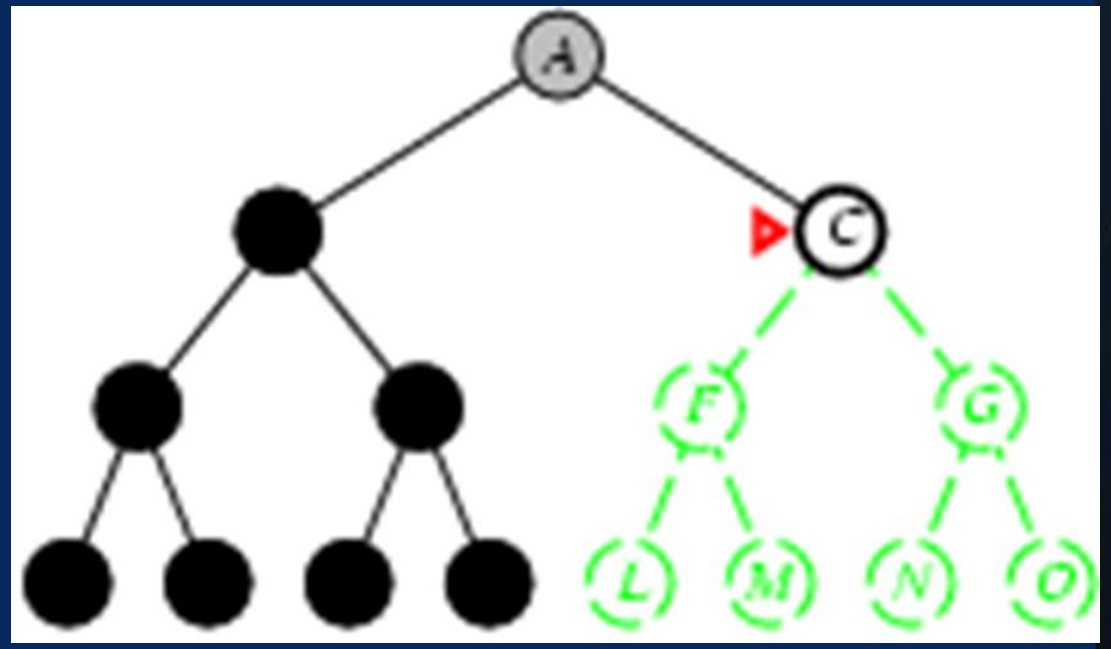
Depth-First Search

DFS traversal stack

Urutan yang ditelusuri:

A B D H I E J K



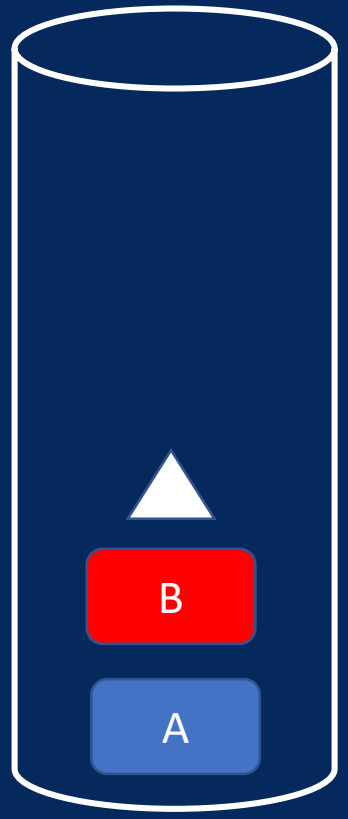


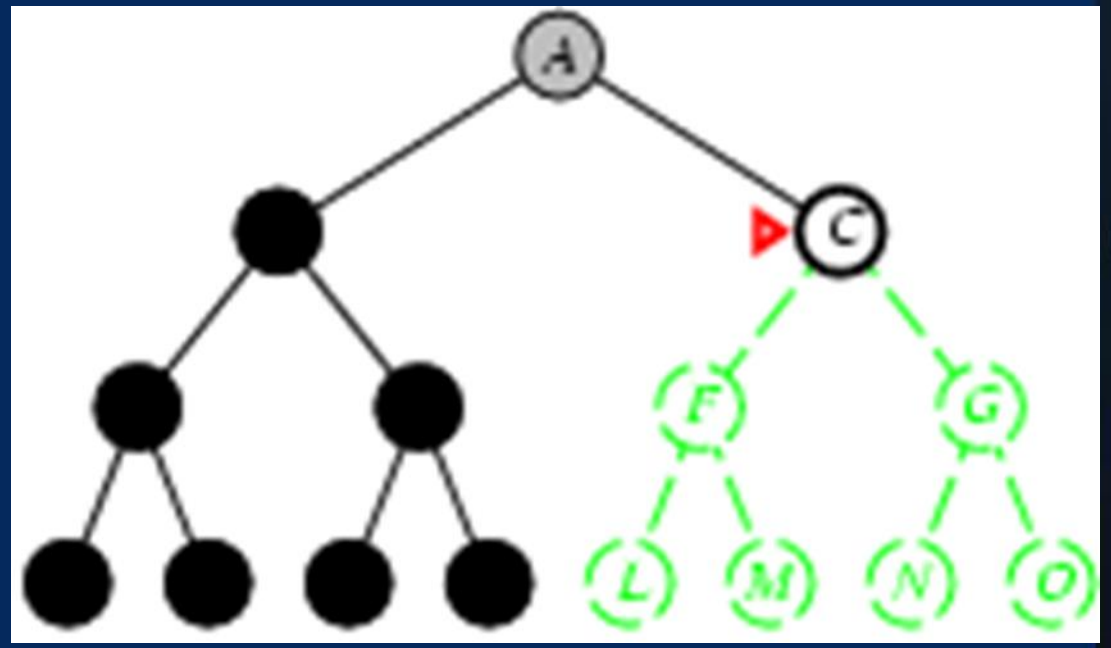
Depth-First Search

DFS traversal stack

Urutan yang ditelusuri:

A B D H I E J K



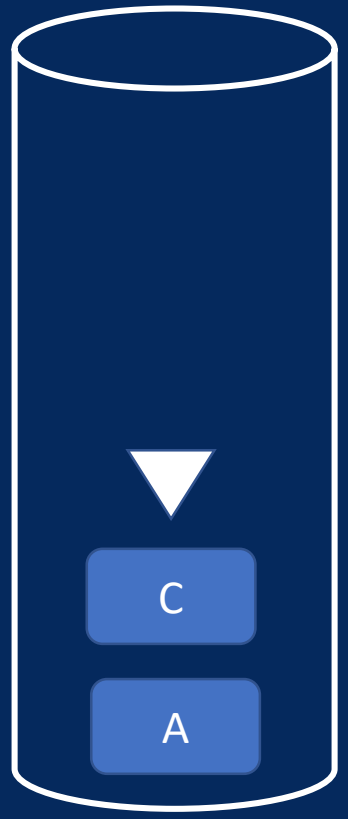


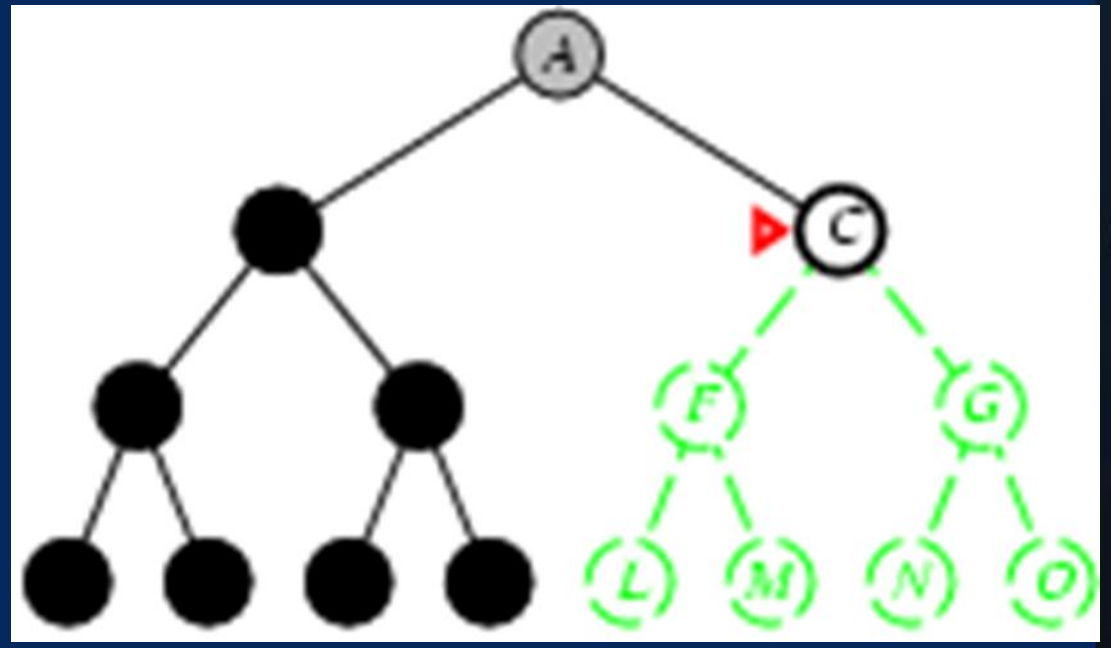
Depth-First Search

DFS traversal stack

Urutan yang ditelusuri:

A B D H I E J K C



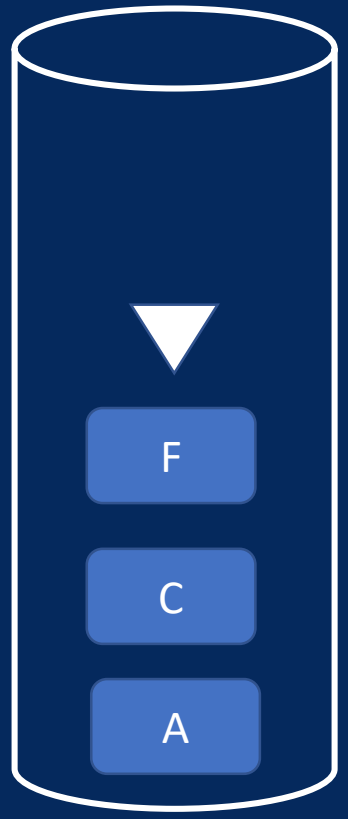


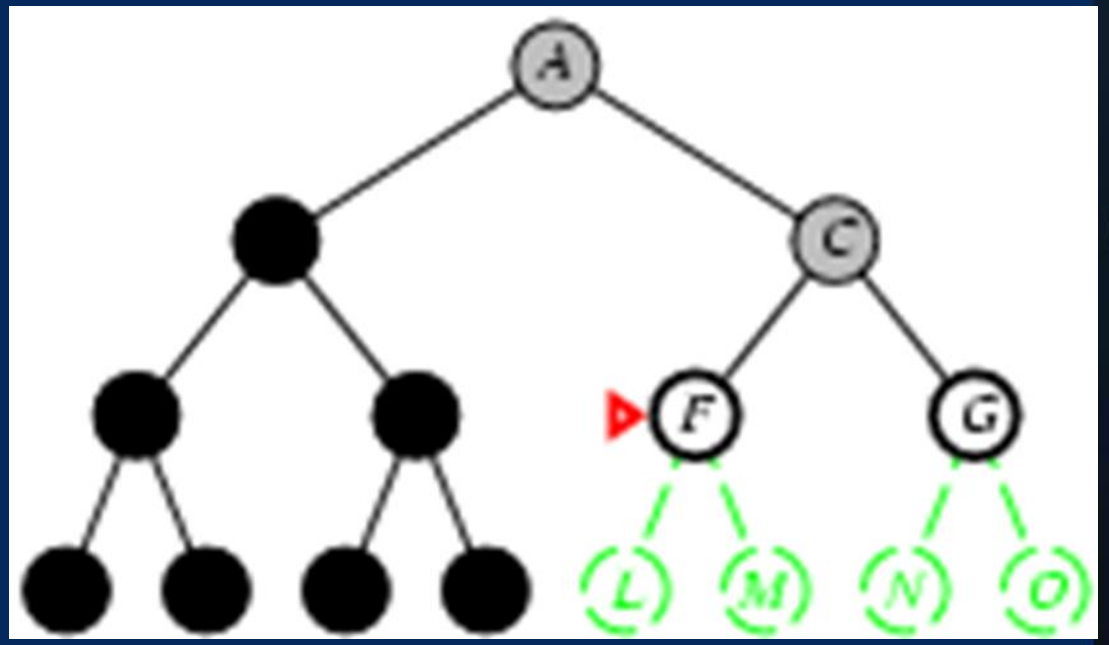
Depth-First Search

DFS traversal stack

Urutan yang ditelusuri:

A B D H I E J K C F



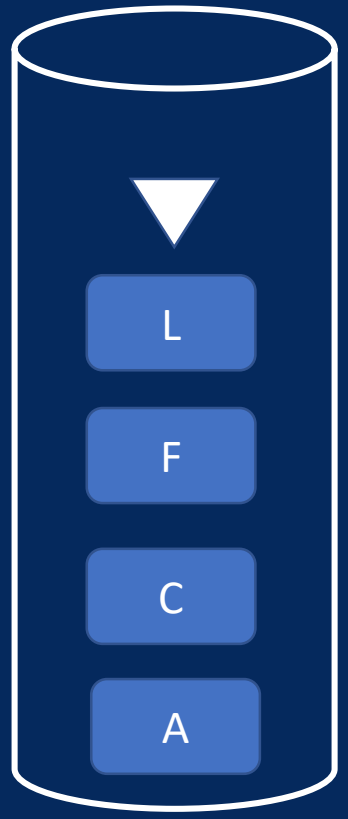


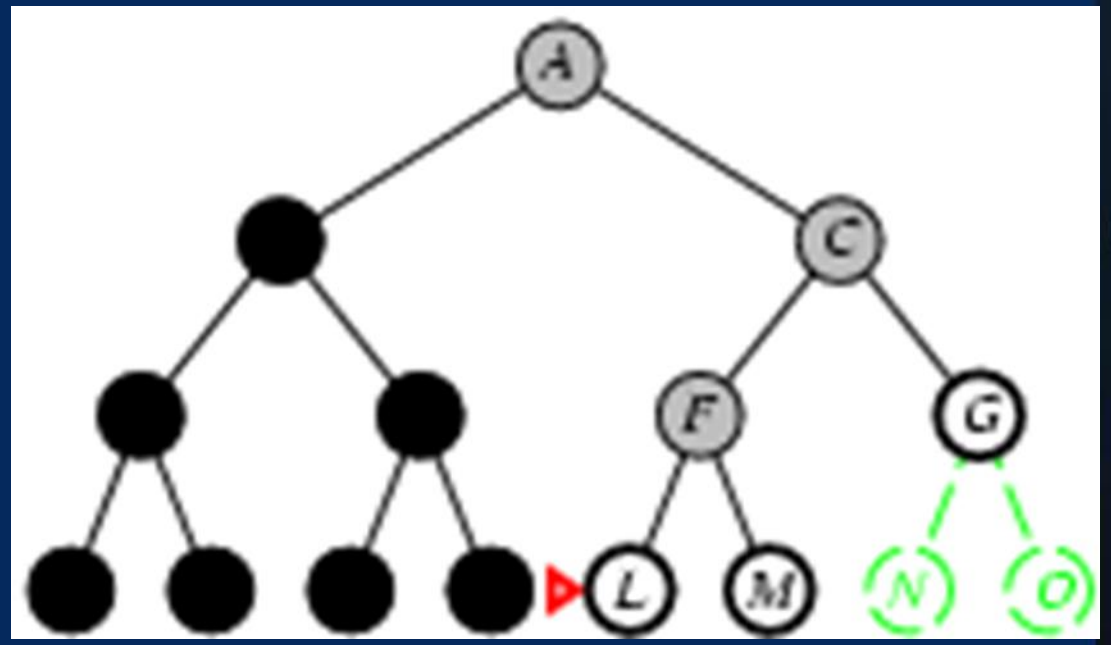
Depth-First Search

DFS traversal stack

Urutan yang ditelusuri:

A B D H I E J K C F L



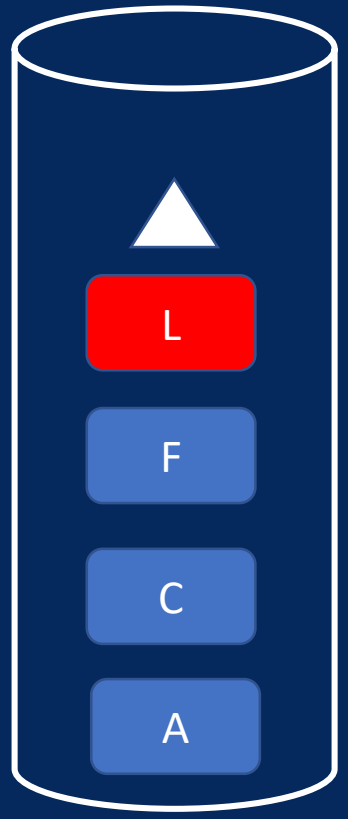


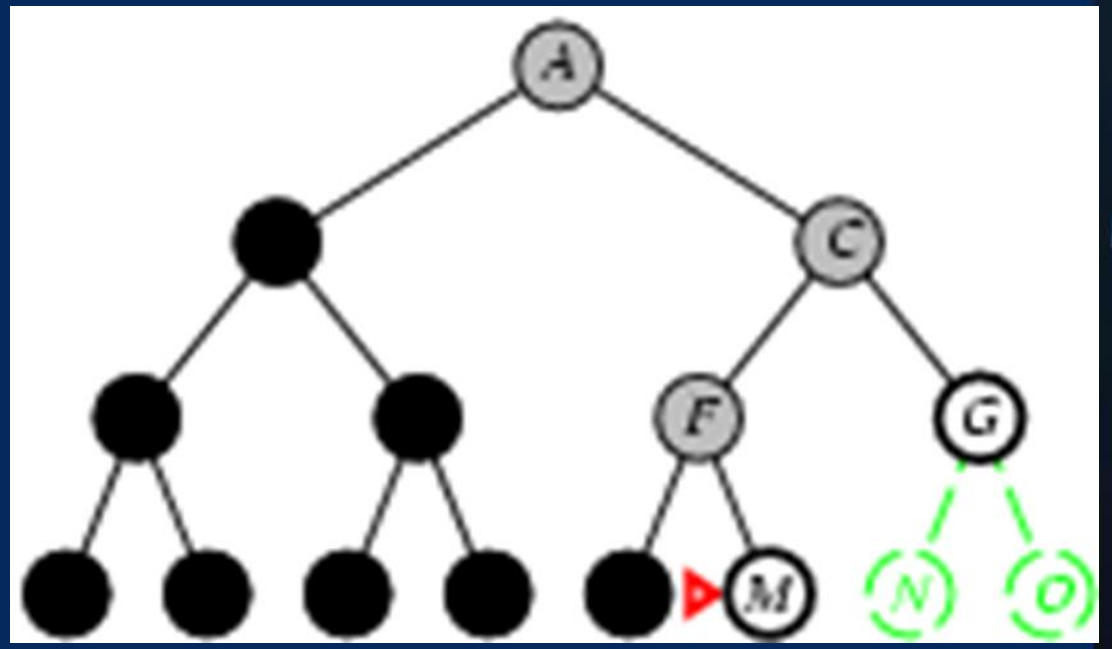
Depth-First Search

DFS traversal stack

Urutan yang ditelusuri:

A B D H I E J K C F L



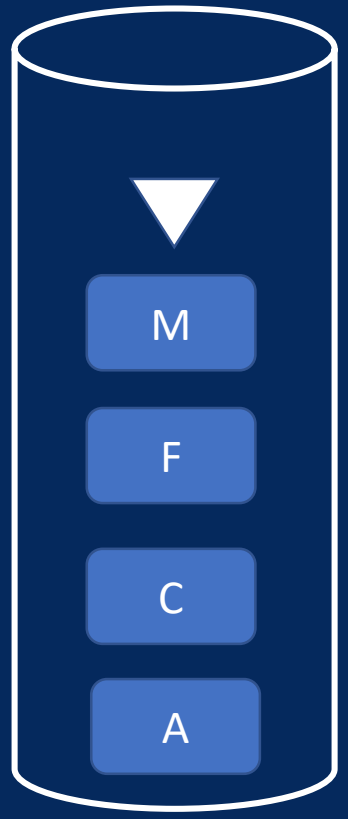


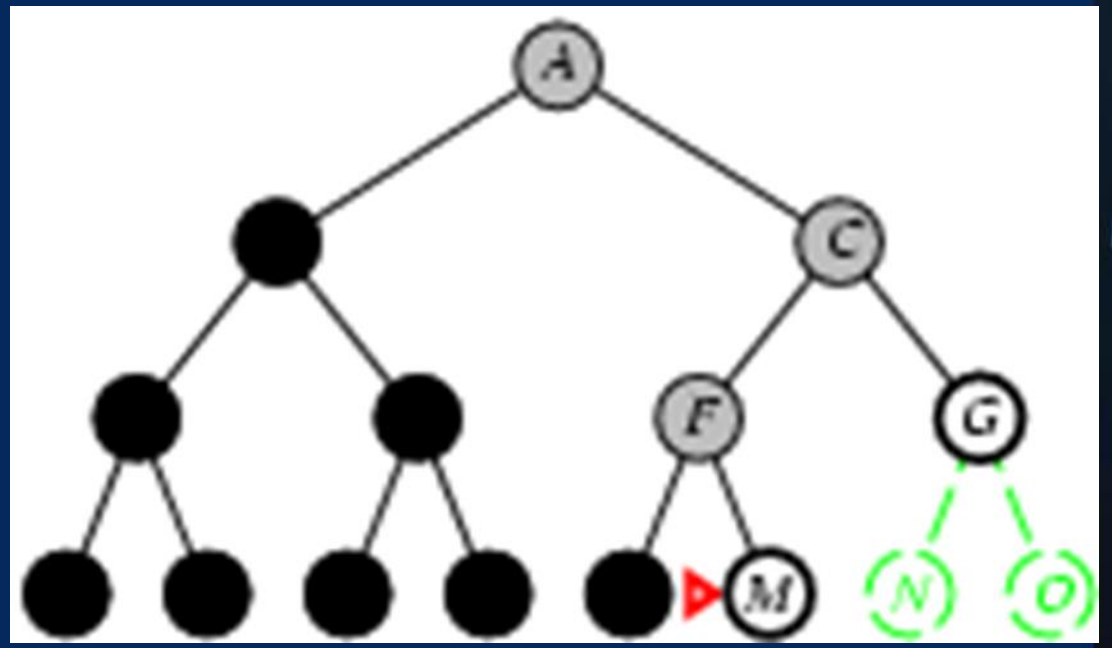
Depth-First Search

DFS traversal stack

Urutan yang ditelusuri:

A B D H I E J K C F L M



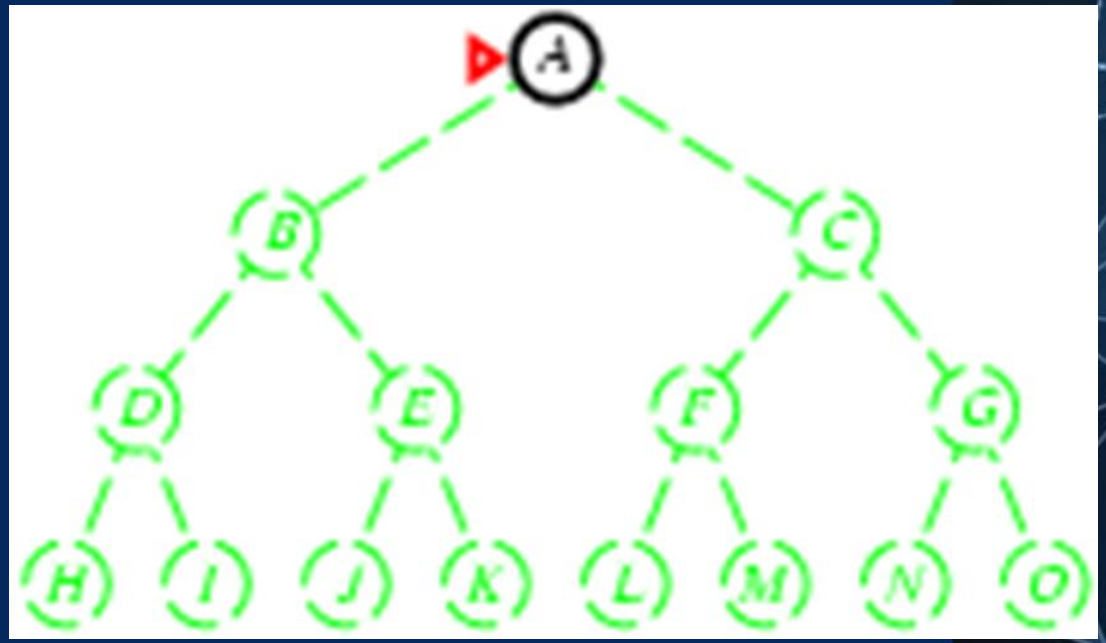


Silahkan lanjutkan…

Depth-First Search

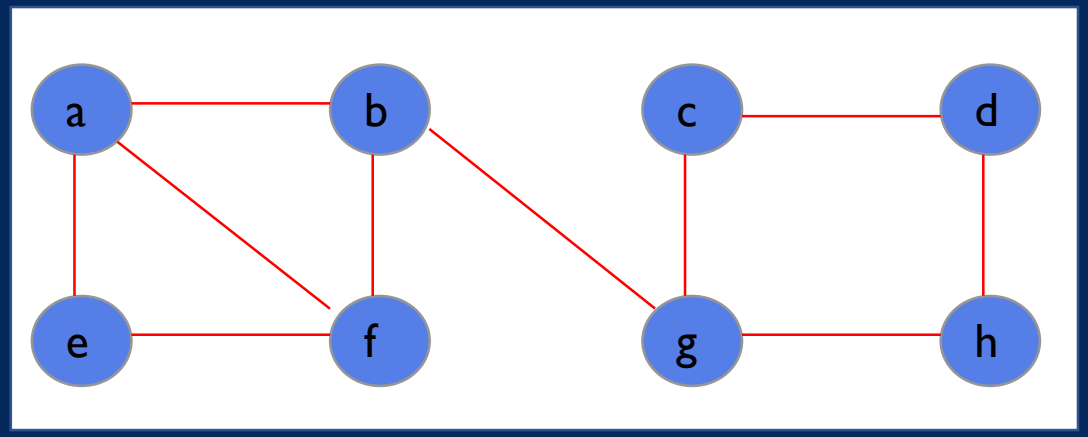
Urutan yang ditelusuri:

- A B D H I E J K C F L M G N O



Depth-First Search

- Contoh Lainnya:



Urutan yang ditelusuri:

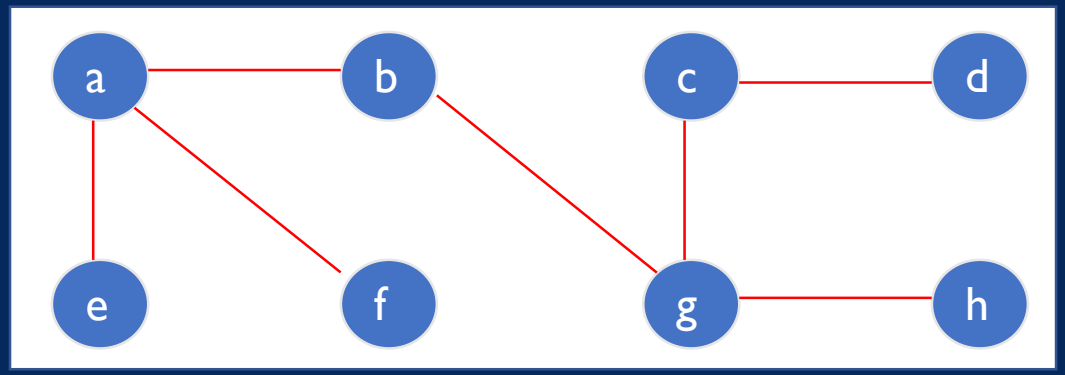
- a b f e g c d h

Depth-First Search

- Properties of depth-first search
- Complete? Tidak, bisa gagal jika m tak terbatas, atau state space dengan loop .
- Time complexity? O(bm) → jika m »d , parah!
- Space complexity? O(bm) → linear space!
- Optimal? Tidak.
- Depth-first search mengatasi masalah space :
  - Mis: 1 node memakan 1000 byte, dan b = 10
  - Jika d = 12, space yang dibutuhkan hanya 118 kilobyte . . .
  - bandingkan dengan 10 petabyte!

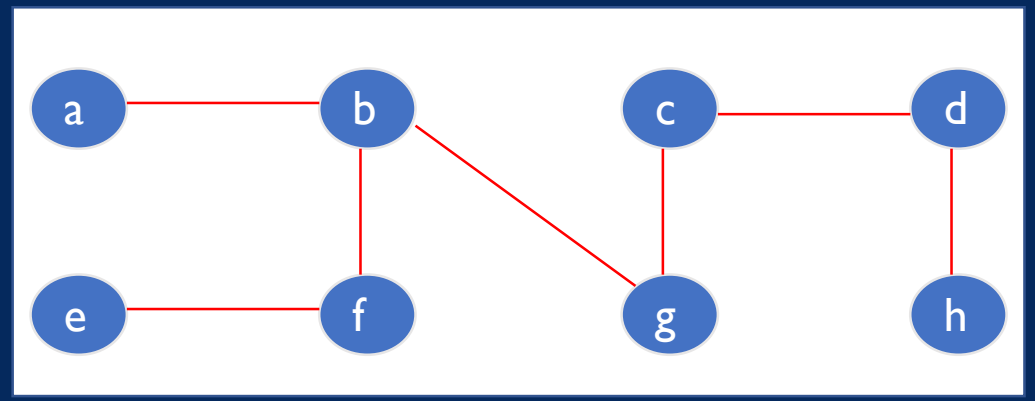
Latihan

- Bagaimana hasil BFS traverse pada graph berikut?



Latihan

- Bagaimana hasil DFS traverse pada graph berikut?



Informed Search

Kecerdasan Buatan JTI POLINEMA

Metode Pencarian Heuristik

- Pencarian buta tidak selalu dapat diterapkan dengan baik, hal ini disebabkan waktu aksesnya yang cukup lama serta besarnya memori yang dibutuhkan.
- Kelemahan ini sebenarnya dapat diatasi jika ada informasi tambahan (fungsi heuristik) dari domain yang bersangkutan.

Metode Pencarian Heuristik

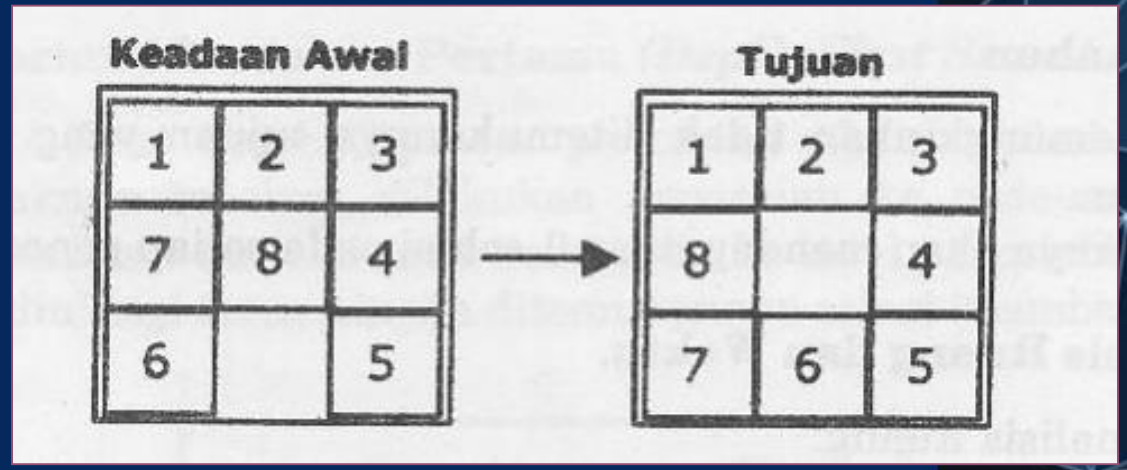
- Heuristik adalah sebuah teknik yang mengembangkan efisiensi dalam proses pencarian, namun dengan kemungkinan mengorbankan kelengkapan (*completeness*).
- Untuk dapat menerapkan heuristik tersebut dengan baik dalam suatu domain tertentu,

  diperlukan suatu Fungsi Heuristik.
- Fungsi heuristik digunakan untuk menghitung *path cost* suatu node tertentu menuju ke node tujuan.

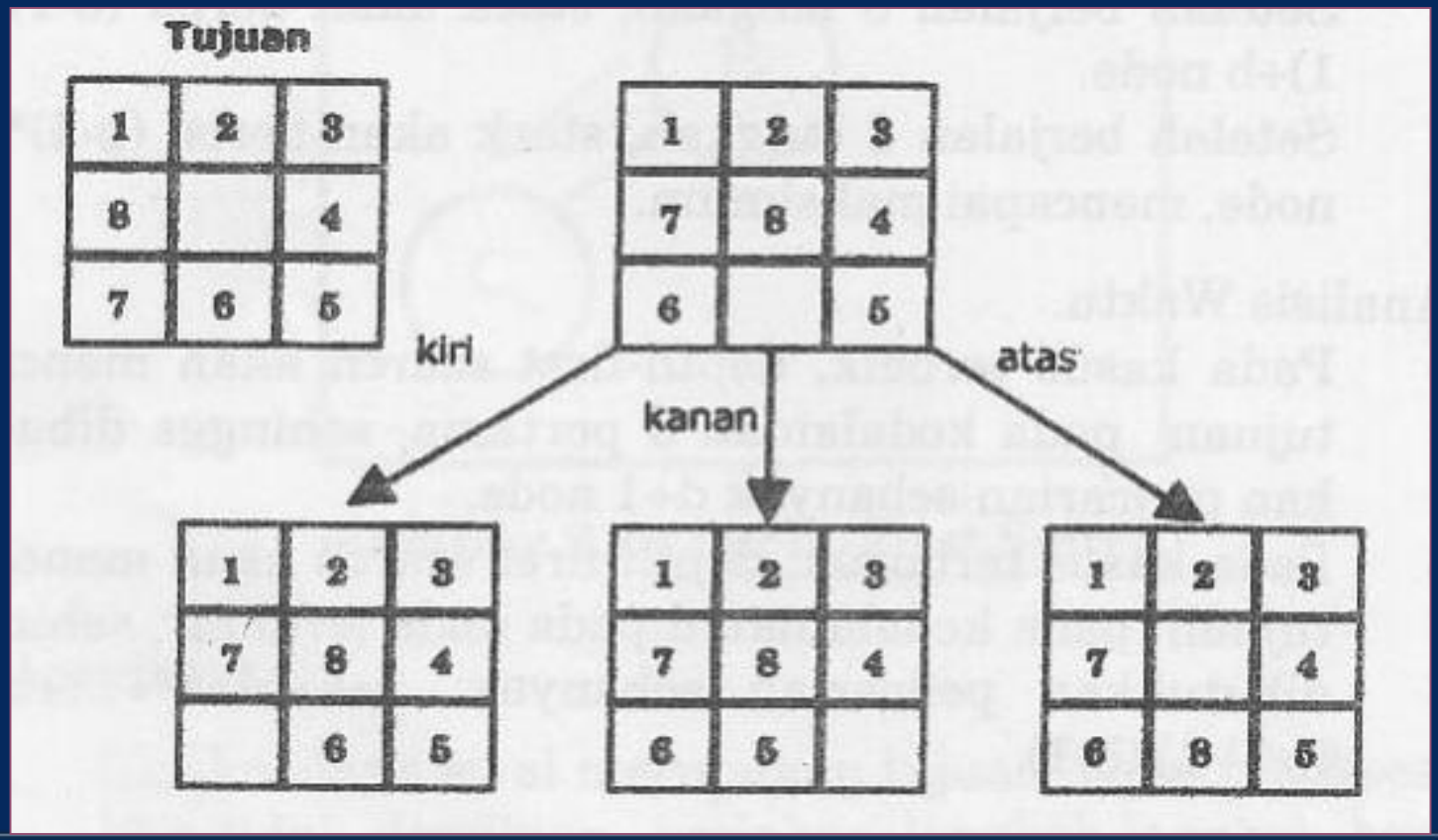
Fungsi Heuristik

Kasus 8-puzzle

- Ada 4 operator yang dapat digunakan untuk menggerakkan dari satu keadaan (state) ke keadaan yang baru.
  - Geser ubin kosong ke kiri
  - Geser ubin kosong ke kanan
  - Geser ubin kosong ke atas
  - Geser ubin kosong ke bawah

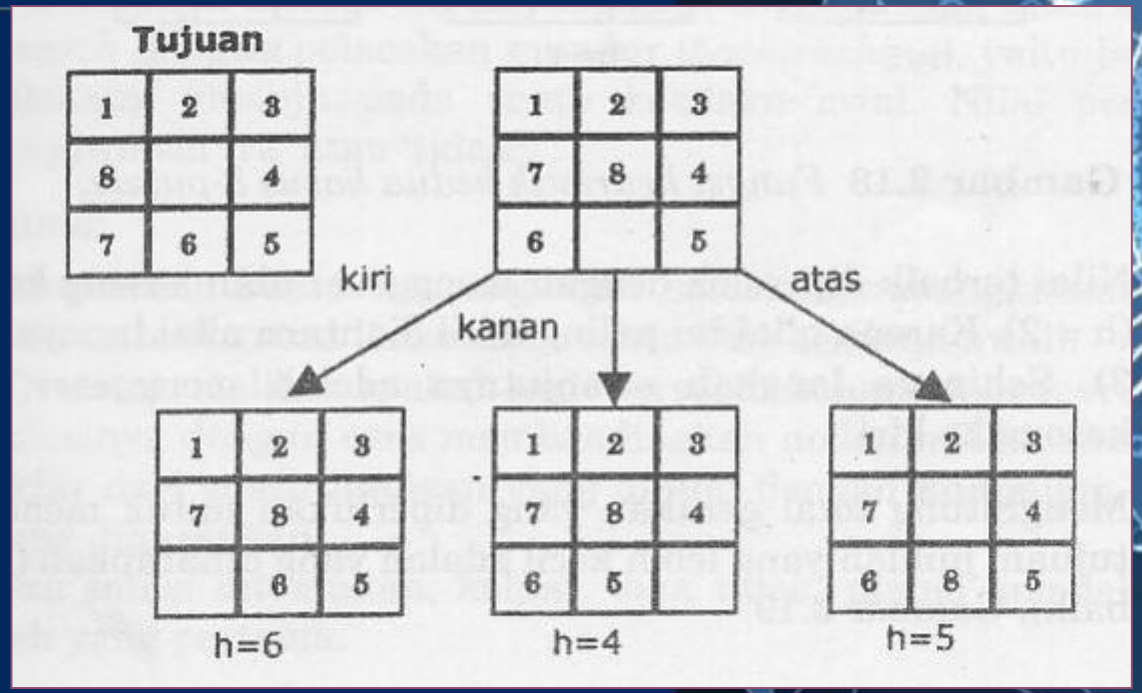


Kasus 8-puzzle



Kasus 8-puzzle

- Informasi yang diberikan dapat berupa jumlah ubin yang menempati posisi yang benar. Jumlah yang lebih tinggi adalah yang diharapkan.
- Sehingga langkah selanjutnya yang harus dilakukan adalah menggeser ubin kosong ke kiri.



Metode simple hill climbing

Algoritma:

1. Initial State
2. Goal State
3. Evaluasi keadaan awal, jika tujuan berhenti, jika tidak lanjut dengan keadaan sekarang sebagai keadaan awal
4. Kerjakan langkah berikut sampai solusi ditemukan atau tidak ada lagi operator baru sebagai keadaan sekarang :

Metode simple hill climbing

1. Cari operator yang belum pernah digunakan. Gunakan operator untuk keadaan yang baru.
2. Evaluasi keadaan sekarang:
   a. Jika keadaan tujuan , keluar.
   b. Jika bukan tujuan, namun nilainya lebih baik dari sekarang, maka jadikan keadaan tersebut sebagai keadaaan sekarang
   c. Jika keadaan baru tidak lebih baik daripada keadaan sekarang, maka llanjutkan iterasi.

Metode simple hill climbing

- Ruang keadaan berisi semua kemungkinan lintasan yang mungkin. Operator digunakan untuk menukar posisi kota-kota yang bersebelahan. Fungsi heuristik yang digunakan adalah panjang lintasan yang terjadi.
- Operator yang akan digunakan adalah menukar urutan posisi 2 kota dalam 1 lintasan. Bila ada n kota, dan ingin mencari kombinasi lintasan dengan menukar posisi urutan 2 kota, maka akan didapat sebanyak :

$$\frac{n!}{2!(n-2)!} - \frac{4!}{2!(4-2)!} - 6 \text{ kombinasi}$$

Metode simple hill climbing

Keenam kombinasi ini akan dipakai semuanya sebagai operator, yaitu :

Tukar 1,2 = menukar urutan posisi kota ke – 1 dengan kota ke – 2

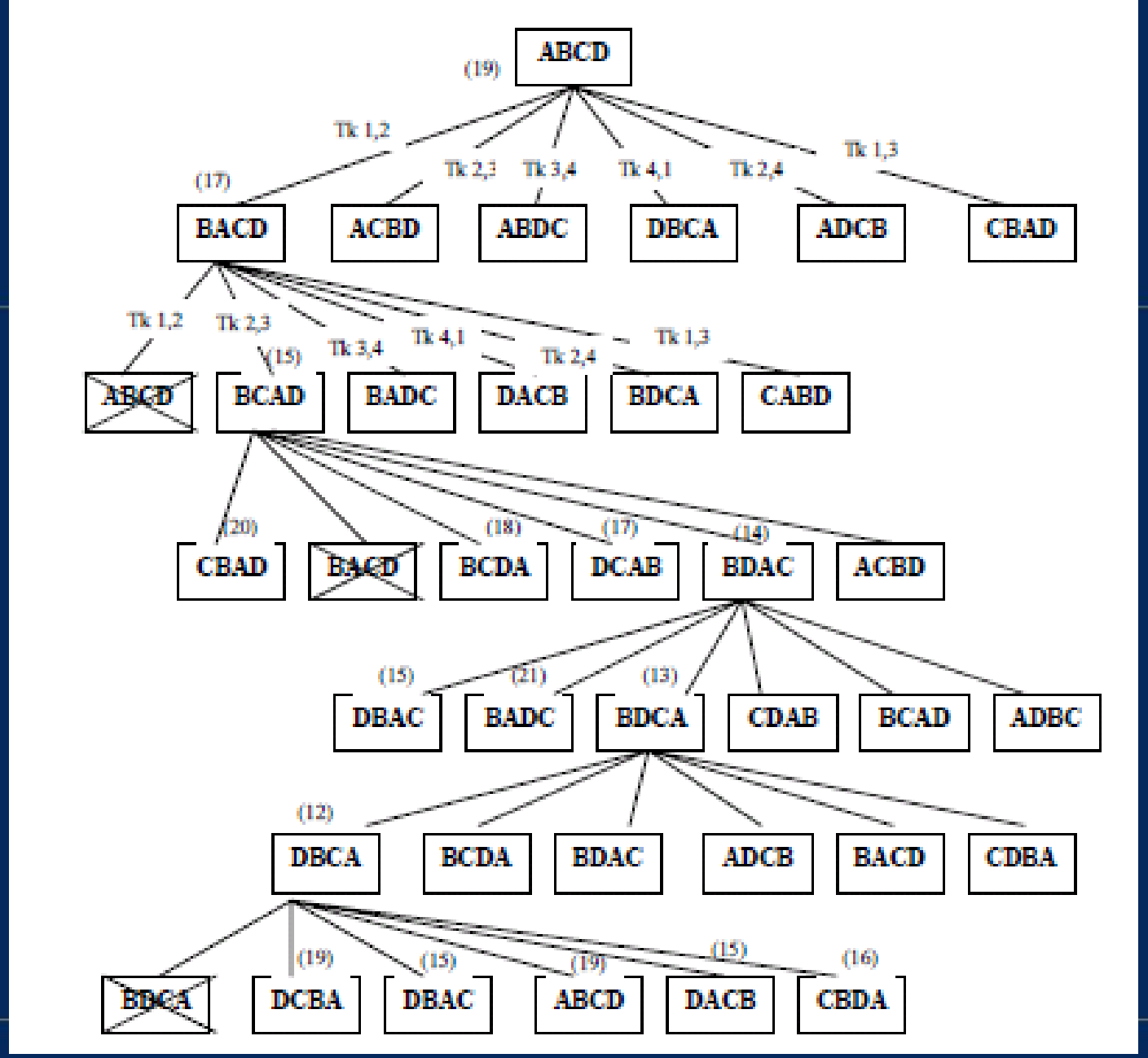
Tukar 2,3 = menukar urutan posisi kota ke – 2 dengan kota ke – 3

Tukar 3,4 = menukar urutan posisi kota ke – 3 dengan kota ke – 4

Tukar 4,1 = menukar urutan posisi kota ke – 4 dengan kota ke – 1

Tukar 2,4 = menukar urutan posisi kota ke – 2 dengan kota ke – 4

Tukar 1,3 = menukar urutan posisi kota ke – 1 dengan kota ke – 3


(19) ABCD
Tk 1,2
Tk 2,3
Tk 3,4
Tk 4,1
Tk 2,4
Tk 1,3
(17) BACD
ACBD
ABDC
DBCA
ADCB
CBAD
Tk 1,2
Tk 2,3
Tk 3,4
Tk 4,1
Tk 2,4
Tk 1,3
ABCD
(15) BCAD
BADC
DACB
BDCA
CABD
(20) CBAD
BACD
(18) BCDA
(17) DCAB
(14) BDAC
ACBD
(15) DBAC
(21) BADC
(13) BDCA
CDAB
BCAD
ADBC
(12) DBCA
BCDA
BDAC
ADCB
BACD
CDBA
BDCA
(19) DCBA
(15) DBAC
(19) ABCD
(15) DACB
(16) CBDA

- Keadaan awal, lintasan ABCD (=19).
- Level pertama, hill climbing mengunjungi BACD (=17), BACD (=17) < ABCD (=19), sehingga
  - BACD menjadi pilihan selanjutnya dengan operator Tukar 1,2
- Level kedua, mengunjungi ABCD, karena operator Tukar 1,2 sudah dipakai BACD, maka pilih node lain yaitu
  - BCAD (=15), BCAD (=15) < BACD (=17)
- Level ketiga, mengunjungi CBAD (=20), CBAD (=20) > BCAD (=15), maka pilih node lain yaitu
    - BCDA (=18), pilih node lain yaitu DCAB (=17), pilih node lain yaitu BDAC (=14), BDAC (=14) < BCAD (=15)
- Level keempat, mengunjungi DBAC (=15), DBAC(=15) > BDAC (=14), maka pilih node lain yaitu
  - BADC (=21), pilih node lain yaitu BDCA (=13), BDCA (=13) < BDAC (=14)
- Level kelima, mengunjungi DBCA (=12), DBCA (=12) < BDCA (=13)
- Level keenam, mengunjungi BDCA, karena operator Tukar 1,2 sudah dipakai DBCA, maka pilih node
  - lain yaitu DCBA, pilih DBAC, pilih ABCD, pilih DACB, pilih CBDA
- Karena sudah tidak ada node yang memiliki nilai heuristik yang lebih kecil dibanding nilai heuristik DBCA, maka **node DBCA** (=12) adalah **lintasan terpendek (SOLUSI)**

Terima Kasih

Kecerdasan Buatan JTI POLINEMA