UNINFORMED SEARCHING

Sukmawati NE

Searching

- Teknik pencarian
  - teknik penyelesaian masalah yang mempresentasikan masalah ke dalam ruang keadaan (*state*) dan secara sistematis melakukan pembangkitan dan pengujian *state-state* dari *initial state* sampai ditemukan suatu *goal state*.
- Contoh :
  - Digunakan dalam pencarian rute optimum untuk memandu seseorang di perjalanan, misal setiap taksi dilengkapi dengan GPS (*Global Positioning System*)

Metode Pencarian

- Pencarian Buta (Blind Search/Uninformed search) :
  - hanya menggunakan informasi berdasarkan problem yang didefinisikan
- Pencarian Terbimbing (Informed/Heuristic Search)
  - Mengeksploitasi informasi-informasi yang dapat mendukung bahwa satu node “lebih menjanjikan” dari pada node lain

Metode Pencarian

- Pencarian Buta (Blind Search/Uninformed search)
  - Pencarian melebar (Breadth-First Search)
  - Pencarian mendalam pertama (Depth-First search)
  - Pencarian mendalam terbatas (Depth Limited search)
  - Iterative Deepening Depth – First Search
  - Bidirectional Search
- Pencarian Terbimbing (Informed/heuristic Search)
  - Generate and Test
  - Pendakian Bukit (Hill Climbing)
  - Pencarian Terbaik Pertama (Best First Search)
  - Tabu Search
  - Simulated Anealing
  - Cheapest Insertion Heuristic

Comparing Uninformed Search Strategies

- Completeness
  - Will a solution always be found if one exists?
  - Apakah solusinya (jika ada) pasti ditemukan?
- Time
  - How long does it take to find the solution?
  - Often represented as the number of nodes searched
  - Merepresentasikan jumlah node yang diexpand
- Space
  - How much memory is needed to perform the search?
  - Often represented as the maximum number of nodes stored at once
  - Jumlah maksimum node di dalam memory
- Optimal
  - Will the optimal (least cost) solution be found?
  - Apakah solusi dengan minimum cost pasti ditemukan

Comparing Uninformed Search Strategies

- Time and space complexity are measured in
  - b – maximum branching factor of the search tree
  - m – maximum depth of the state space (kedalaman maksimum dari ruang keadaan atau pohon pencarian, ada kemungkinan infinite)
  - d – depth of the least cost solution (kedalaman dari solusi optimal)

Breadth-First Search

- Pada metode Breadth-First Search, semua node pada level n akan dikunjungi terlebih dahulu sebelum mengunjungi node-node pada level n+1
- Pencarian dimulai dari node akar terus ke level ke-1 dari kiri ke kanan, kemudian berpindah ke level berikutnya demikian pula dari kiri ke kanan hingga ditemukannya goal

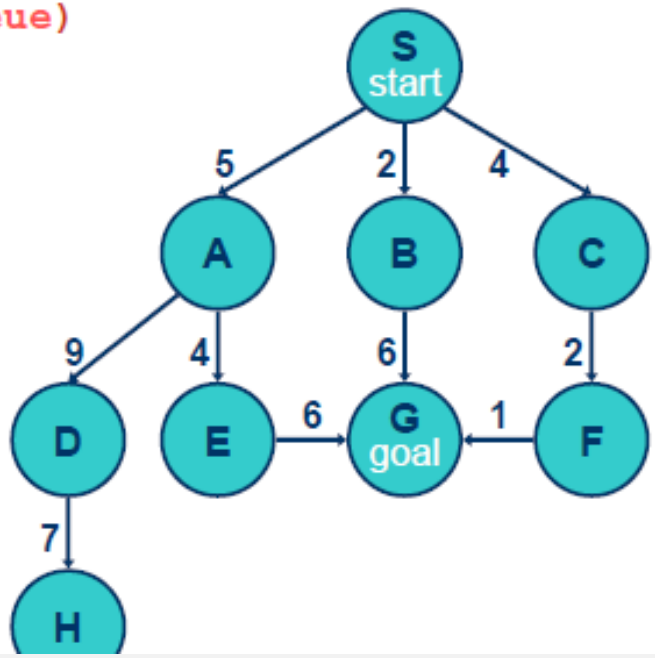
Search

- **Breadth-First Search**

`generalSearch(problem, queue)`

# of nodes tested: 0, expanded: 0

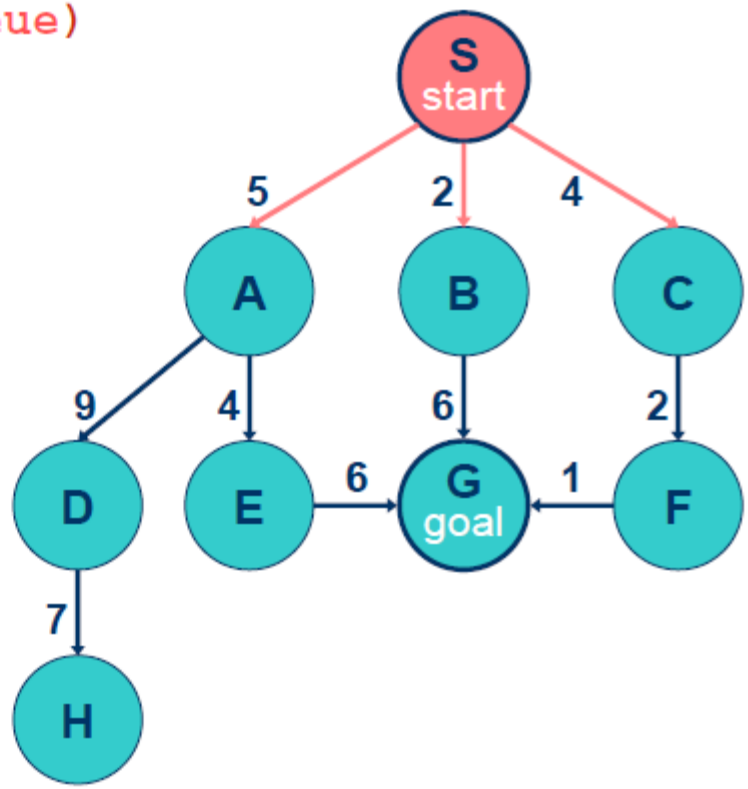
expnd. node	nodes list
	{S}



`generalSearch(problem, queue)`

# of nodes tested: 1, expanded: 1

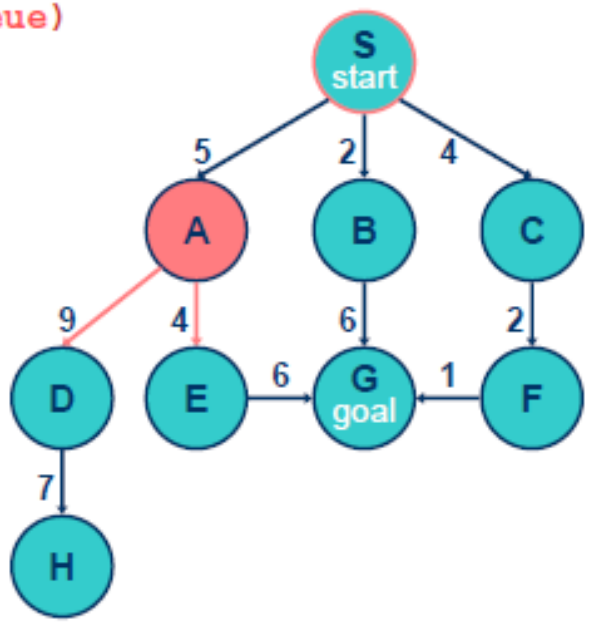
expnd. node	nodes list
	{S}
S not goal	{A,B,C}



`generalSearch(problem, queue)`

# of nodes tested: 2, expanded: 2

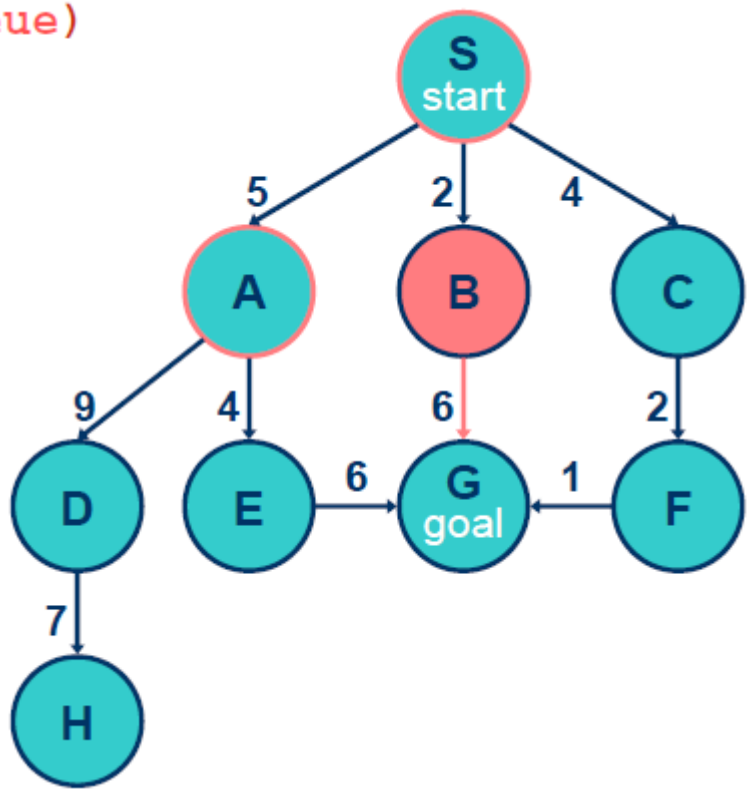
expnd. node	nodes list
	{S}
S	{A,B,C}
A not goal	{B,C,D,E}



`generalSearch(problem, queue)`

# of nodes tested: 3, expanded: 3

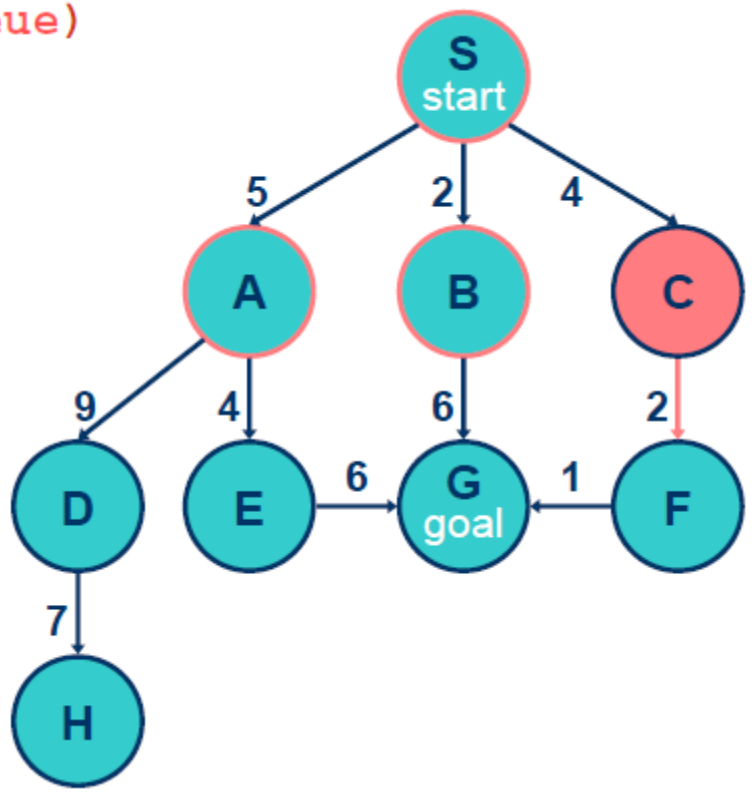
expnd. node	nodes list
	{S}
S	{A,B,C}
A	{B,C,D,E}
B not goal	{C,D,E,G}



`generalSearch(problem, queue)`

# of nodes tested: 4, expanded: 4

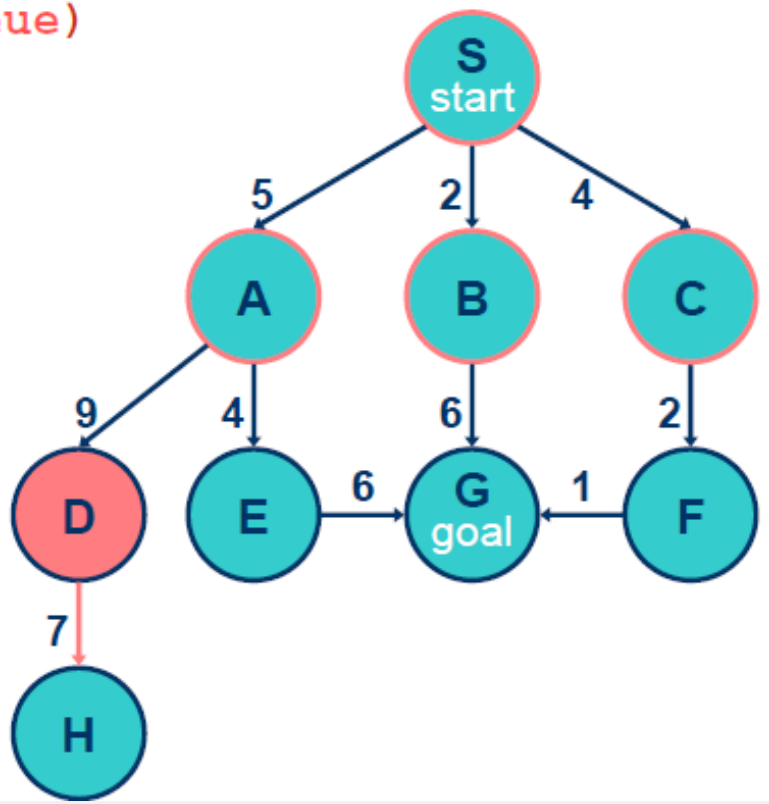
expnd. node	nodes list
	{S}
S	{A,B,C}
A	{B,C,D,E}
B	{C,D,E,G}
C not goal	{D,E,G,F}



`generalSearch(problem, queue)`

# of nodes tested: 5, expanded: 5

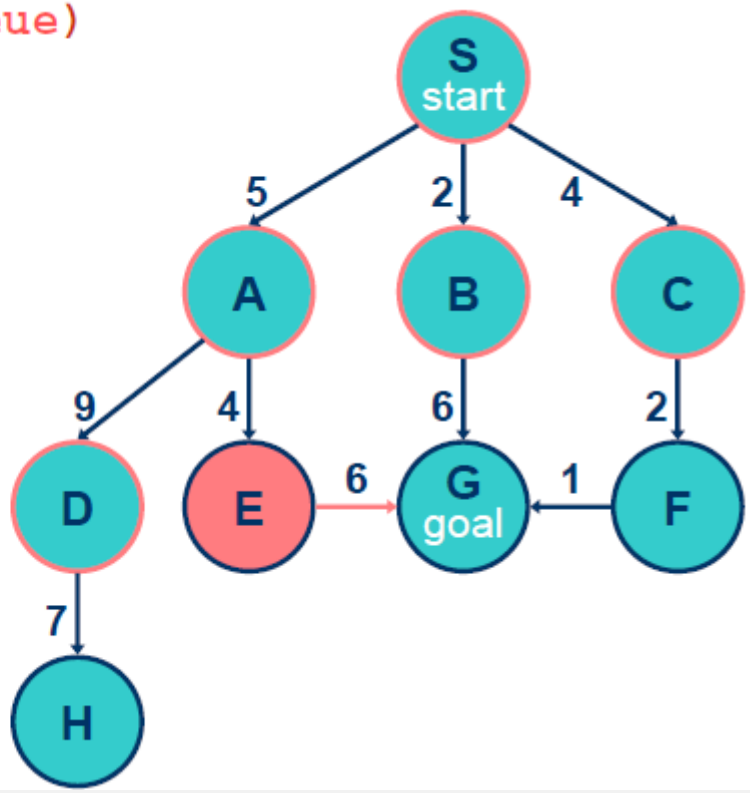
expnd. node	nodes list
	{S}
S	{A,B,C}
A	{B,C,D,E}
B	{C,D,E,G}
C	{D,E,G,F}
D not goal	{E,G,F,H}



`generalSearch(problem, queue)`

# of nodes tested: 6, expanded: 6

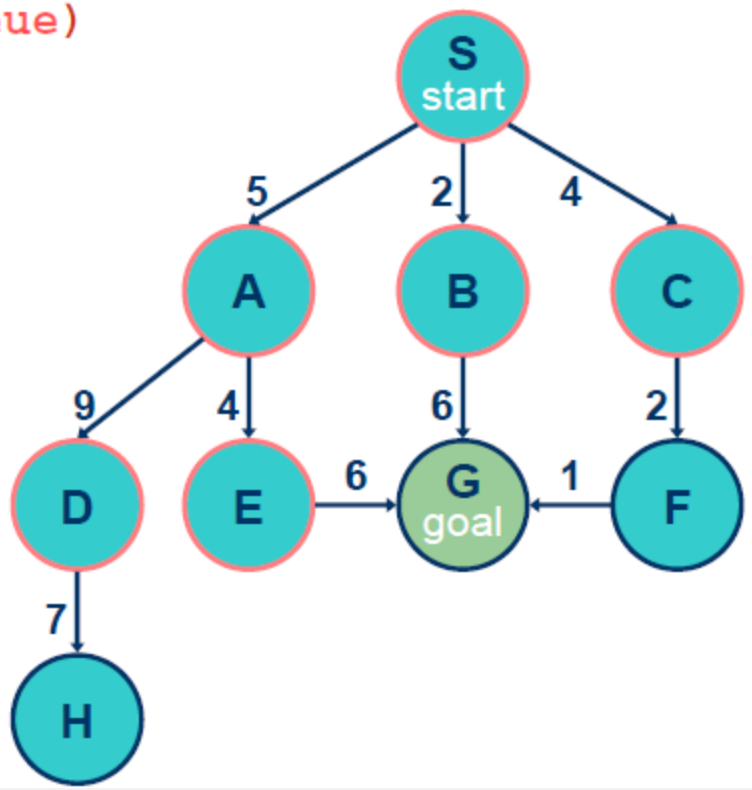
expnd. node	nodes list
	{S}
S	{A,B,C}
A	{B,C,D,E}
B	{C,D,E,G}
C	{D,E,G,F}
D	{E,G,F,H}
E not goal	{G,F,H,G}



`generalSearch(problem, queue)`

# of nodes tested: 7, expanded: 6

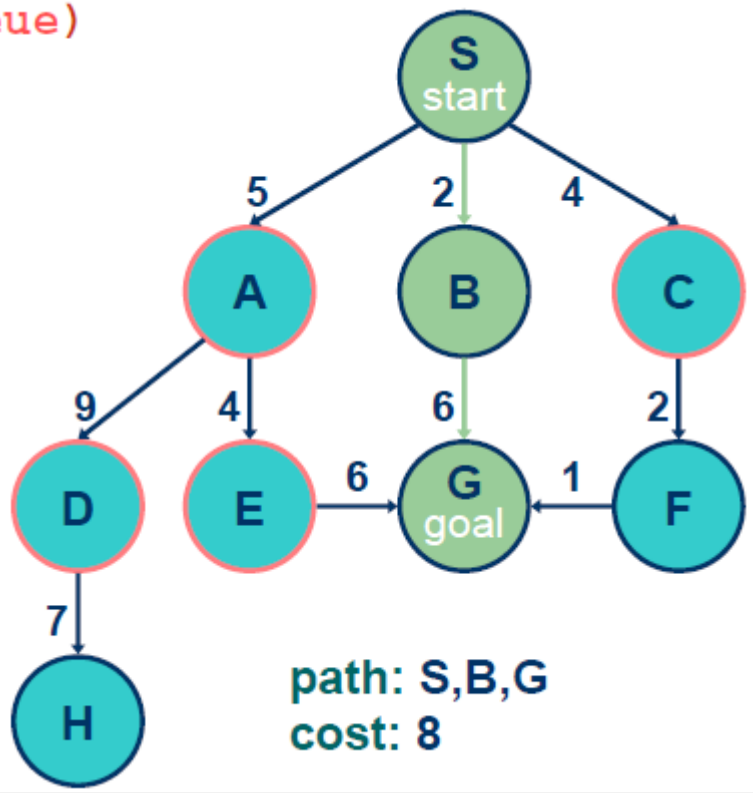
expnd. node	nodes list
	{S}
S	{A,B,C}
A	{B,C,D,E}
B	{C,D,E,G}
C	{D,E,G,F}
D	{E,G,F,H}
E	{G,F,H,G}
G goal	{F,H,G} no expand



`generalSearch(problem, queue)`

# of nodes tested: 7, expanded: 6

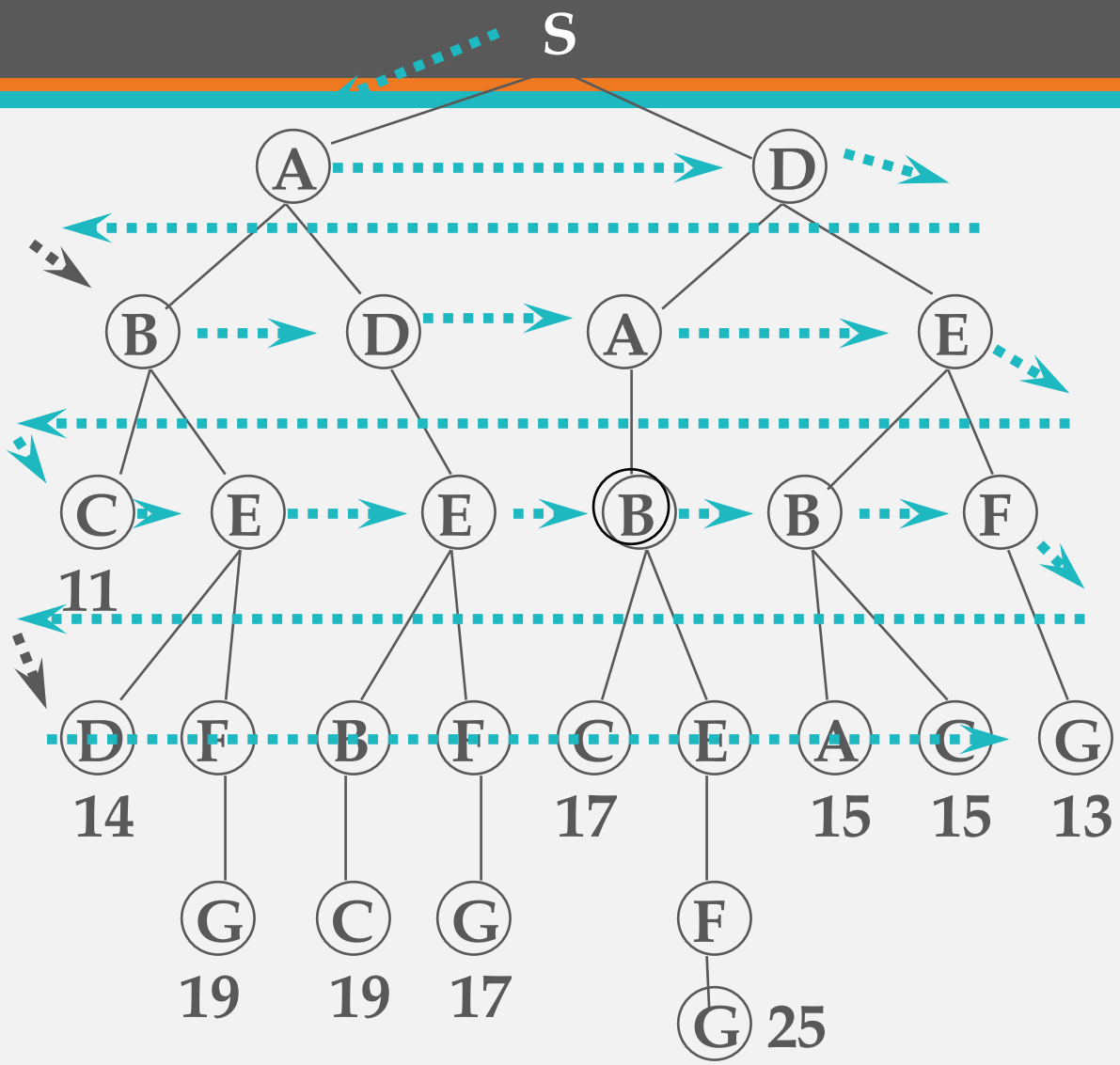
expnd. node	nodes list
	{S}
S	{A,B,C}
A	{B,C,D,E}
B	{C,D,E,G}
C	{D,E,G,F}
D	{E,G,F,H}
E	{G,F,H,G}
G	{F,H,G}



Breadth-First Search

```
function Breath-First-Search(problem) returns solution
   nodes := Make-Queue(Make-Node(Initial-State(problem))
loop do
  if nodes is empty then return failure
   node := Remove-Front (nodes)
  if Goal-Test[problem] applied to State(node) succeeds
      then return node
   new-nodes := Expand (node, Operators[problem]))
   nodes := Insert-At-End-of-Queue(new-nodes)
end
```

Another Breath-first search



Evaluation BFS

- Complete
  - Yes if b (max branching factor) is finite
- Time
  - $1 + b + b^2 + \ldots + b^d + b(b^d-1) = O(b^{d+1})$
  - exponential in d
- Space
  - $O(b^{d+1})$
  - Keeps every node in memory
- Optimal
  - Yes (if cost is 1 per step); not optimal in general

Latihan

- Buatlah search tree untuk masalah penuangan air pada teko A (4 liter) dan teko B (3 liter) untuk menghasilkan 2 liter pada teko A
- Dengan BFS, carilah solusinya, menghasilkan berapa cost nya?

Depth First Search

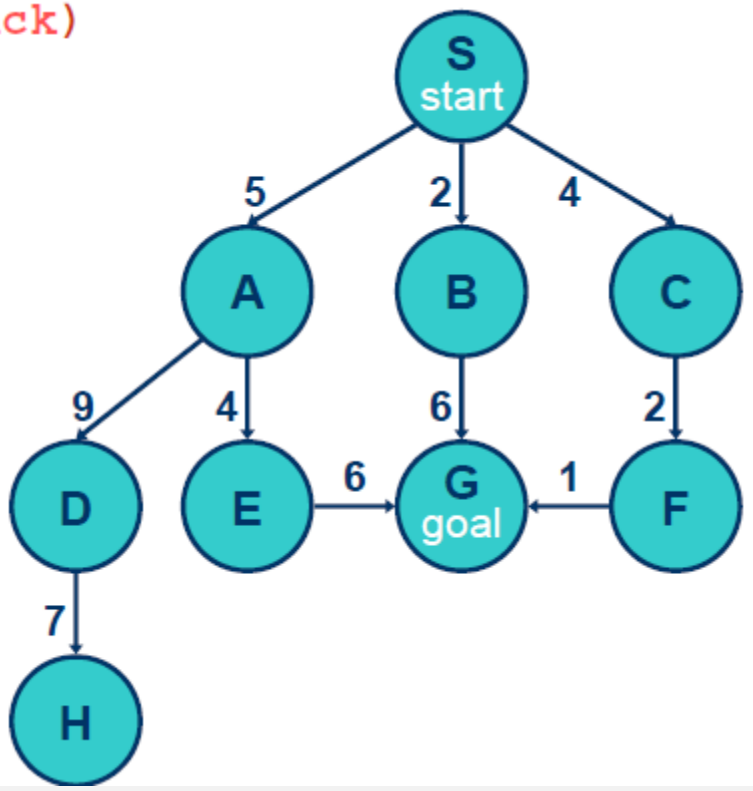
- Pada Depth-First Search, proses pencarian akan dilakukan pada semua anaknya sebelum dilakukan pencarian ke node-node yang selevel.
- Pencarian dimulai dari node akar ke level yang lebih tinggi. Proses ini diulangi terus hingga ditemukannya solusi

Depth-first

`generalSearch(problem, stack)`

# of nodes tested: 0, expanded: 0

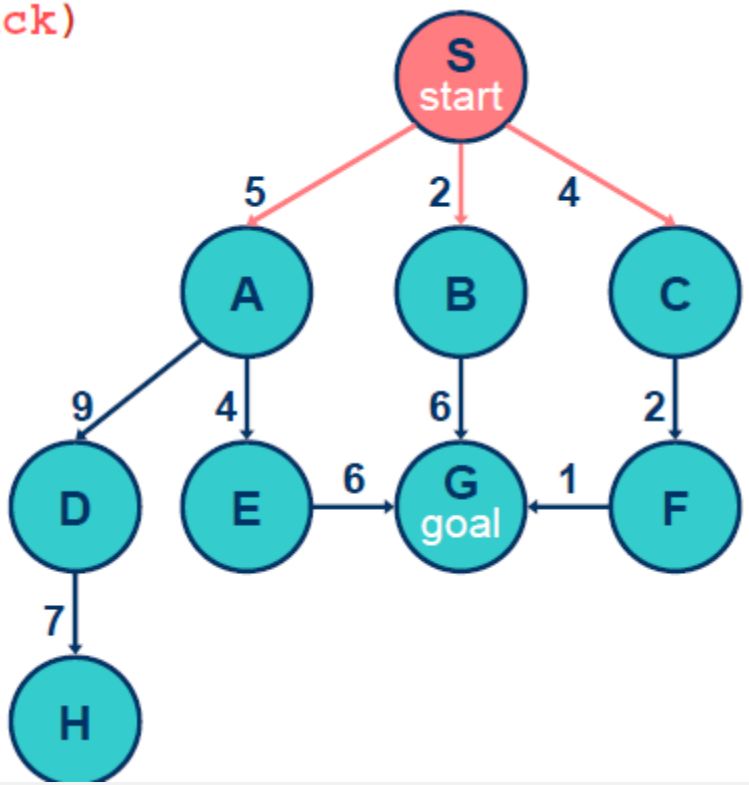
expnd. node	nodes list
	{S}



`generalSearch(problem, stack)`

# of nodes tested: 1, expanded: 1

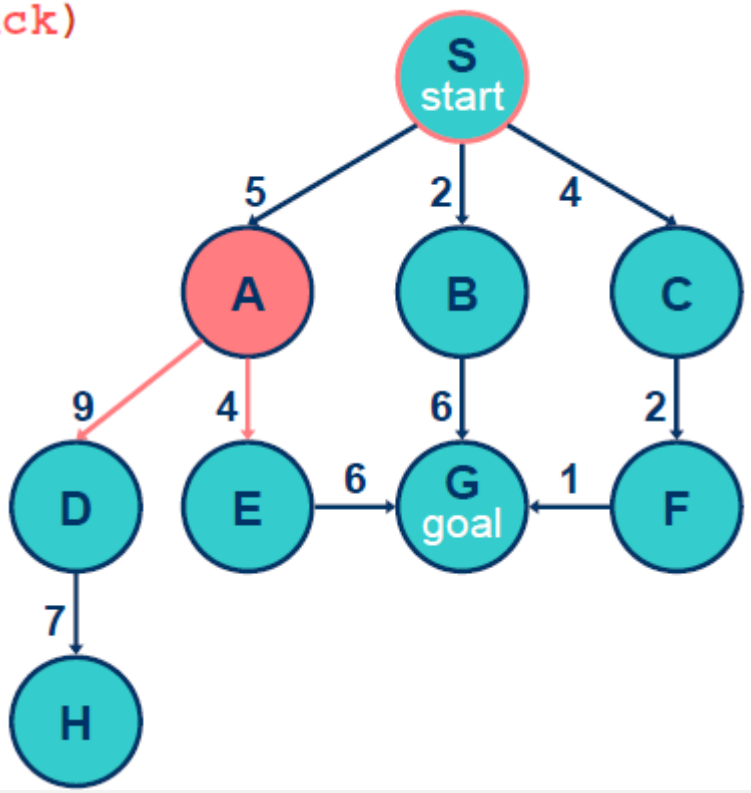
expnd. node	nodes list
	{S}
S not goal	{A,B,C}



`generalSearch(problem, stack)`

# of nodes tested: 2, expanded: 2

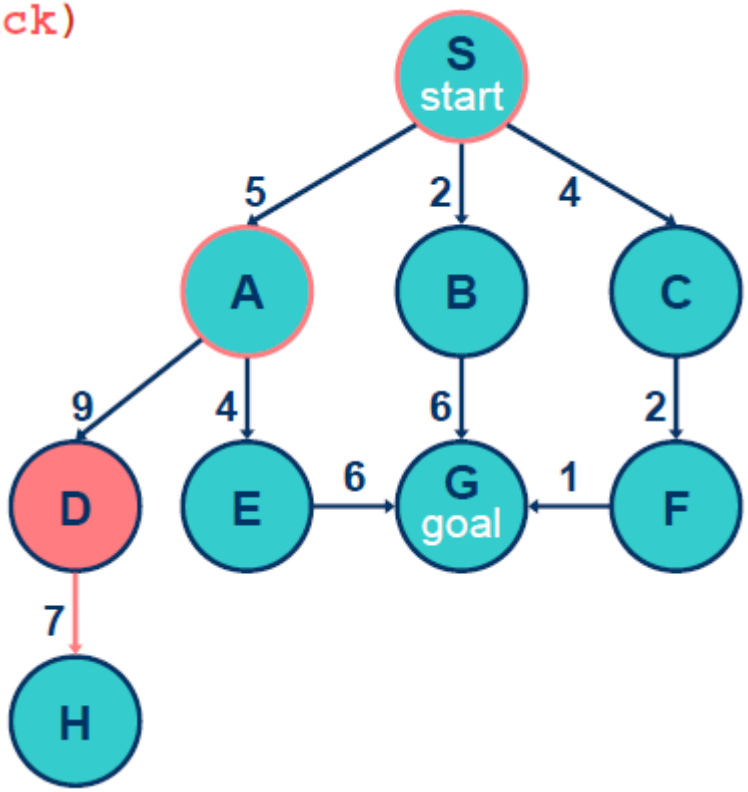
expnd. node	nodes list
	{S}
S	{A,B,C}
A not goal	{D,E,B,C}



`generalSearch(problem, stack)`

# of nodes tested: 3, expanded: 3

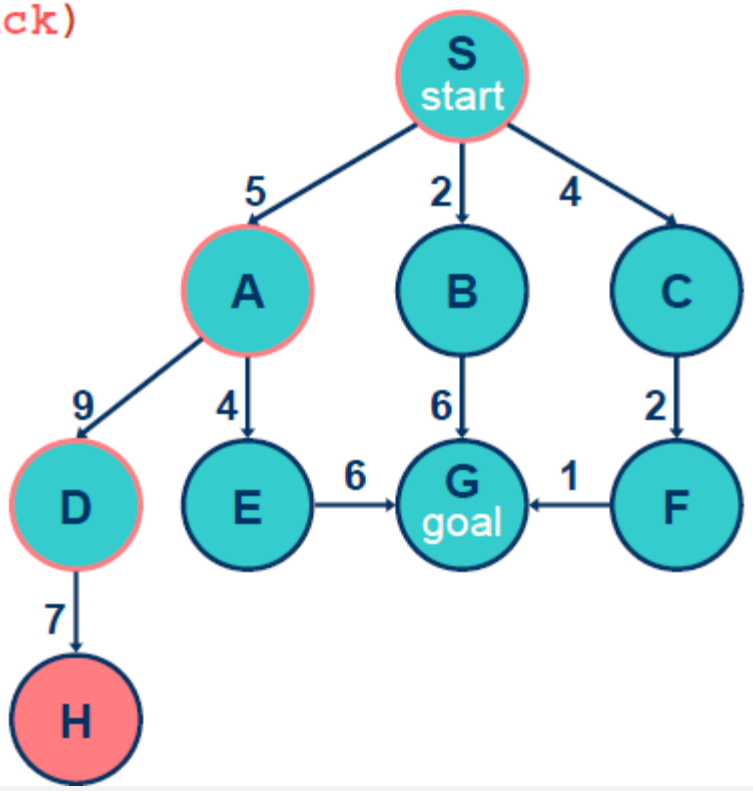
expnd. node	nodes list
	{S}
S	{A,B,C}
A	{D,E,B,C}
D not goal	{H,E,B,C}



`generalSearch(problem, stack)`

# of nodes tested: 4, expanded: 4

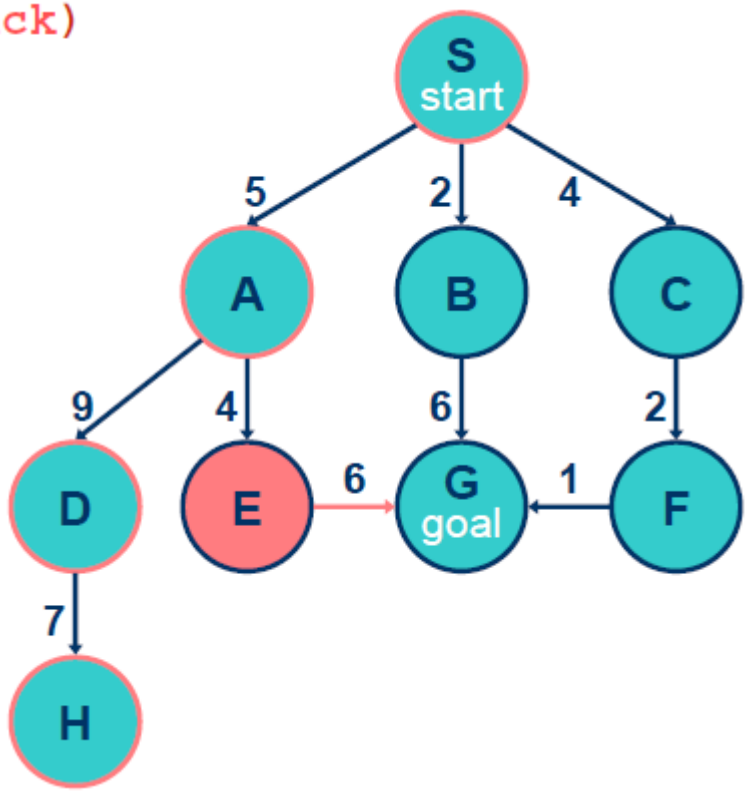
expnd. node	nodes list
	{S}
S	{A,B,C}
A	{D,E,B,C}
D	{H,E,B,C}
H not goal	{E,B,C}



`generalSearch(problem, stack)`

# of nodes tested: 5, expanded: 5

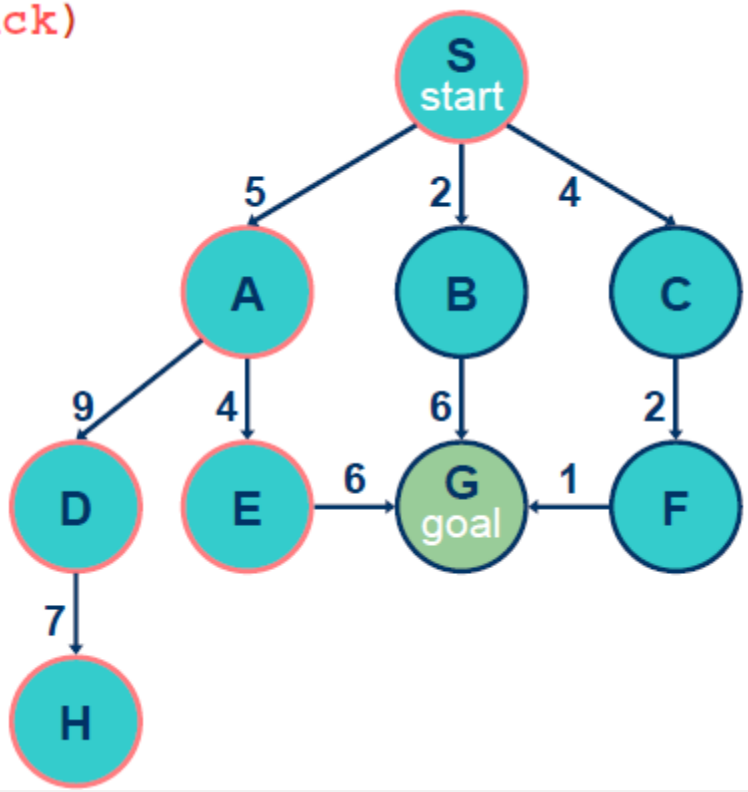
expnd. node	nodes list
	{S}
S	{A,B,C}
A	{D,E,B,C}
D	{H,E,B,C}
H	{E,B,C}
E not goal	{G,B,C}



`generalSearch(problem, stack)`

# of nodes tested: 6, expanded: 5

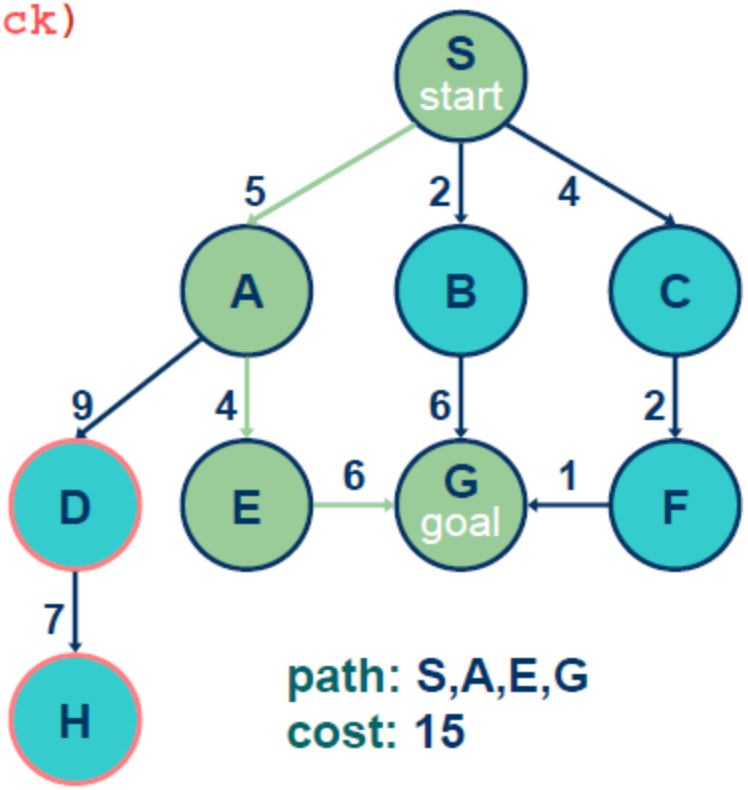
expnd. node	nodes list
	{S}
S	{A,B,C}
A	{D,E,B,C}
D	{H,E,B,C}
H	{E,B,C}
E	{G,B,C}
G goal	{B,C} no expand



`generalSearch(problem, stack)`

# of nodes tested: 6, expanded: 5

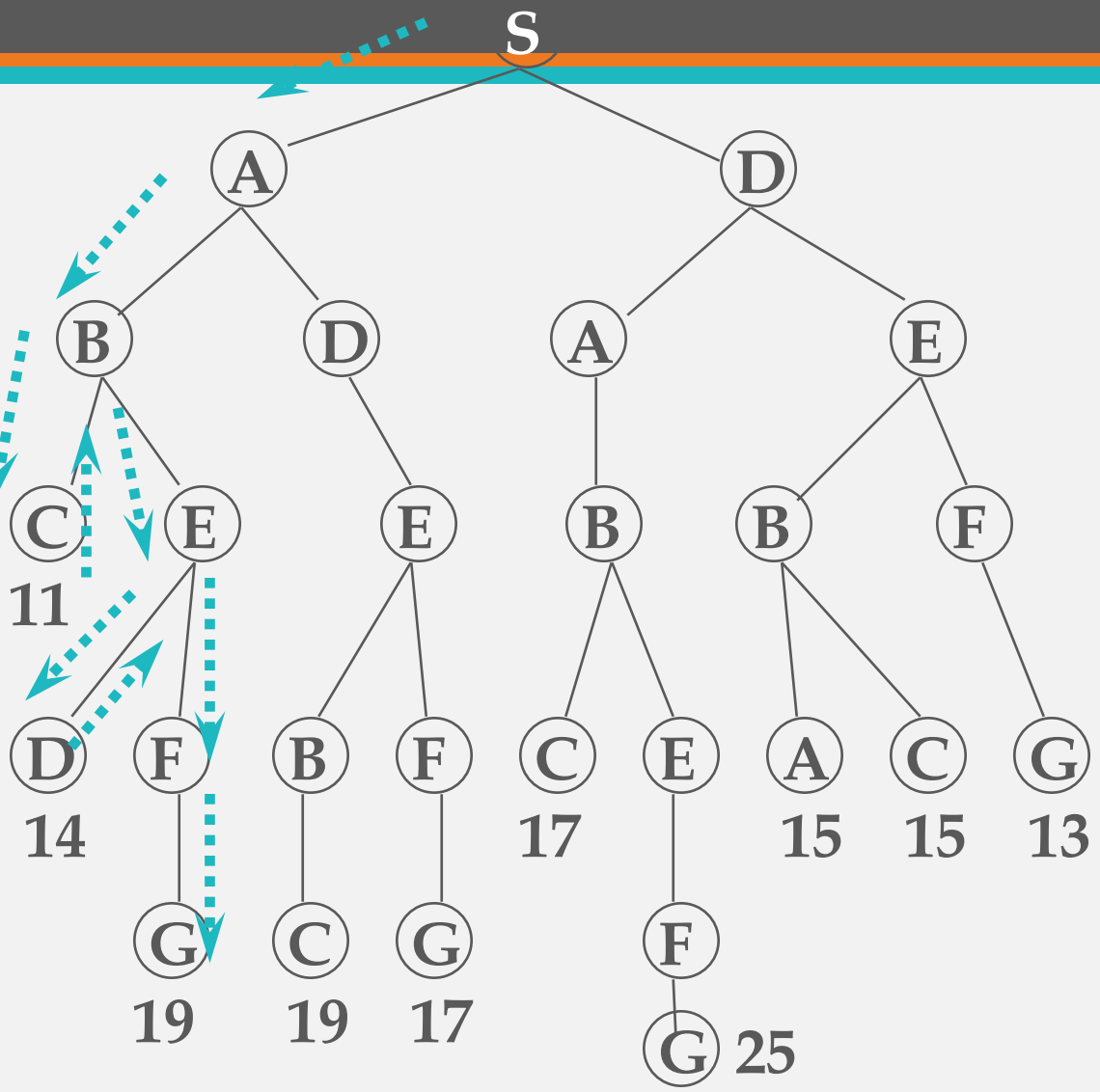
expnd. node	nodes list
	{S}
S	{A,B,C}
A	{D,E,B,C}
D	{H,E,B,C}
H	{E,B,C}
E	{G,B,C}
G	{B,C}



Depth first search

```
function Depth-First-Search(problem) returns solution
   nodes := Make-Queue(Make-Node(Initial-State(problem))
loop do
  if nodes is empty then return failure
   node := Remove-Front (nodes)
  if Goal-Test[problem] applied to State(node) succeeds
      then return node
   new-nodes := Expand (node, Operarors[problem]))
   nodes := Insert-At-Front-of-Queue(new-nodes)
end
```

Depth-first search



Evaluation DFS

- Complete
  - No: fails in infinite-depth spaces, spaces with loops
    - Modify to avoid repeated spaces along path
  - Yes: in finite spaces
- Time
  - $O(b^m)$
  - Not great if m is much larger than d
  - But if the solutions are dense, this may be faster than breadth-first search
- Space
  - O(bm)…linear space
- Optimal
  - No

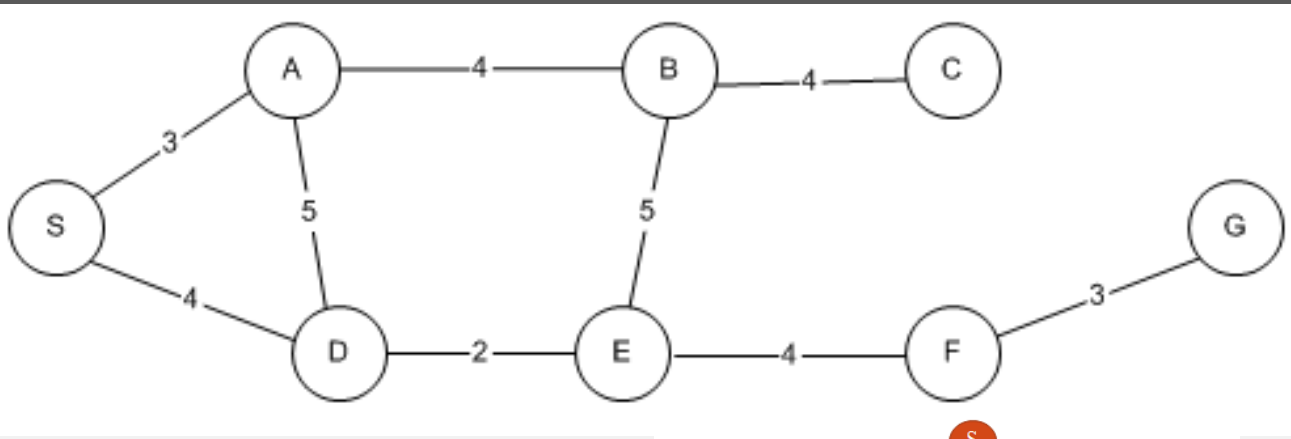
Latihan

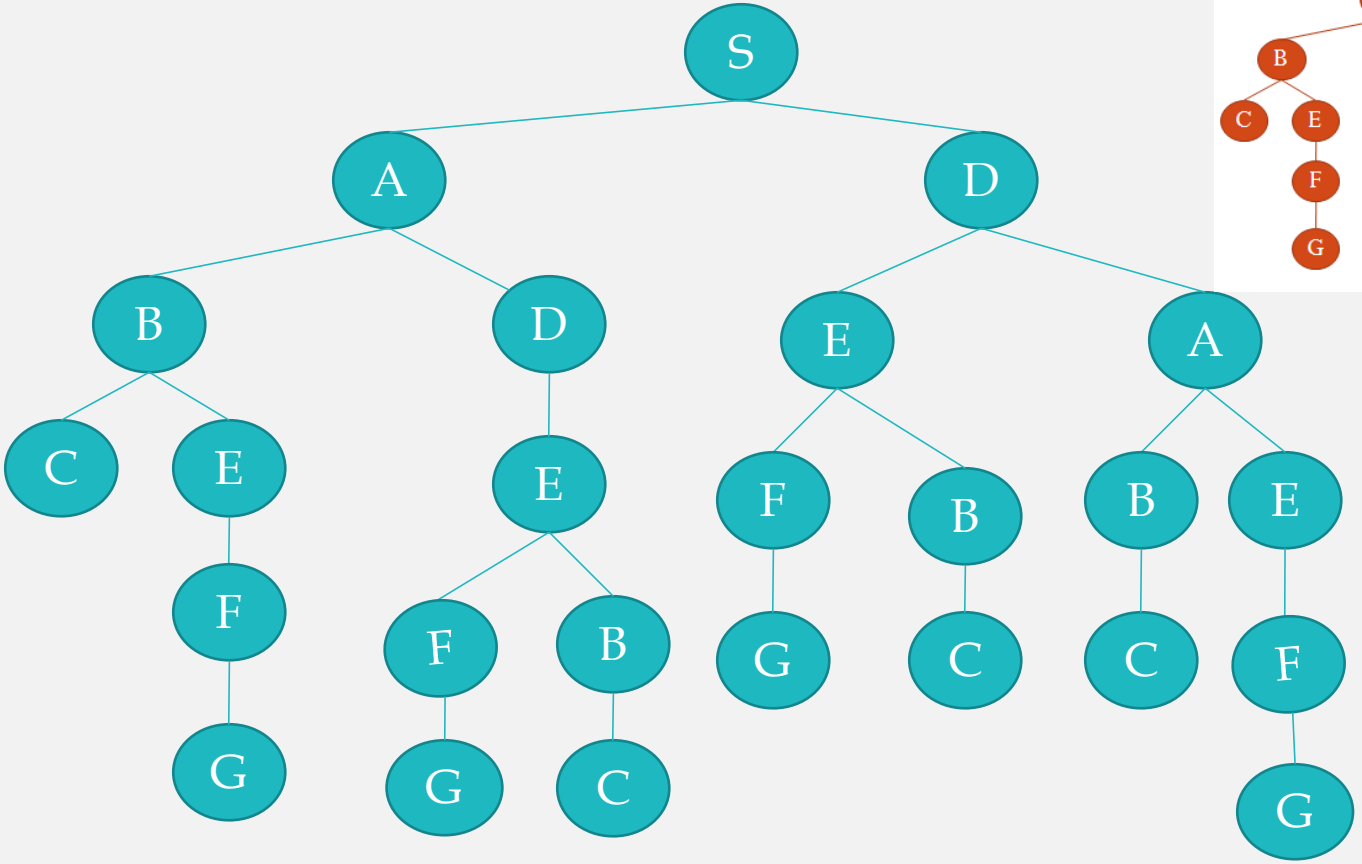
- Dengan menggunakan search tree yang sudah dibuat di latihan sebelumnya, cari solusi dengan DFS, ada berapa langkah untuk mencapai solusi?

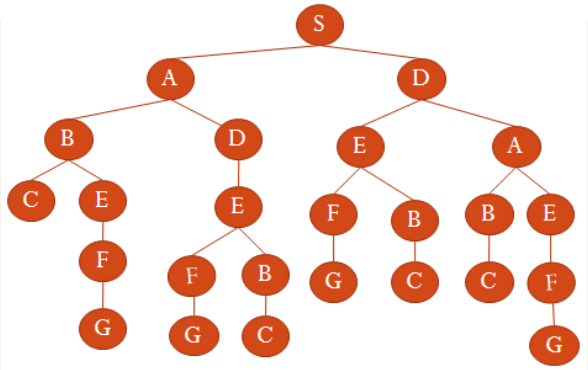
Depth Limited Search

- Sama dengan pada Depth First Search
- Tetapi kedalaman dari pohon dibatasi
- Jika batas kedalaman sudah tercapai akan dilanjutkan ke cabang berikutnya
- Misal kedalaman maksimal : 3

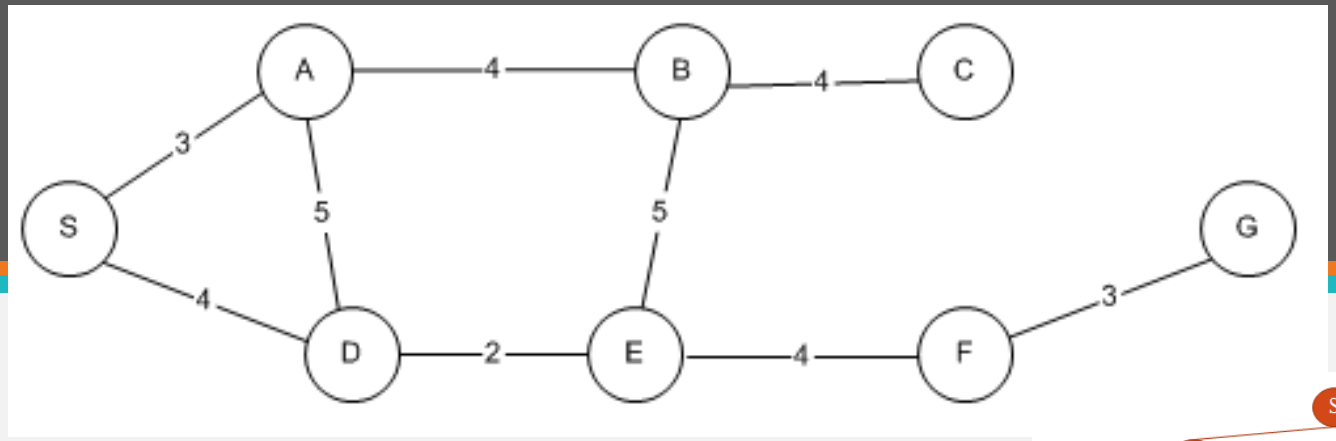
Contoh kasus

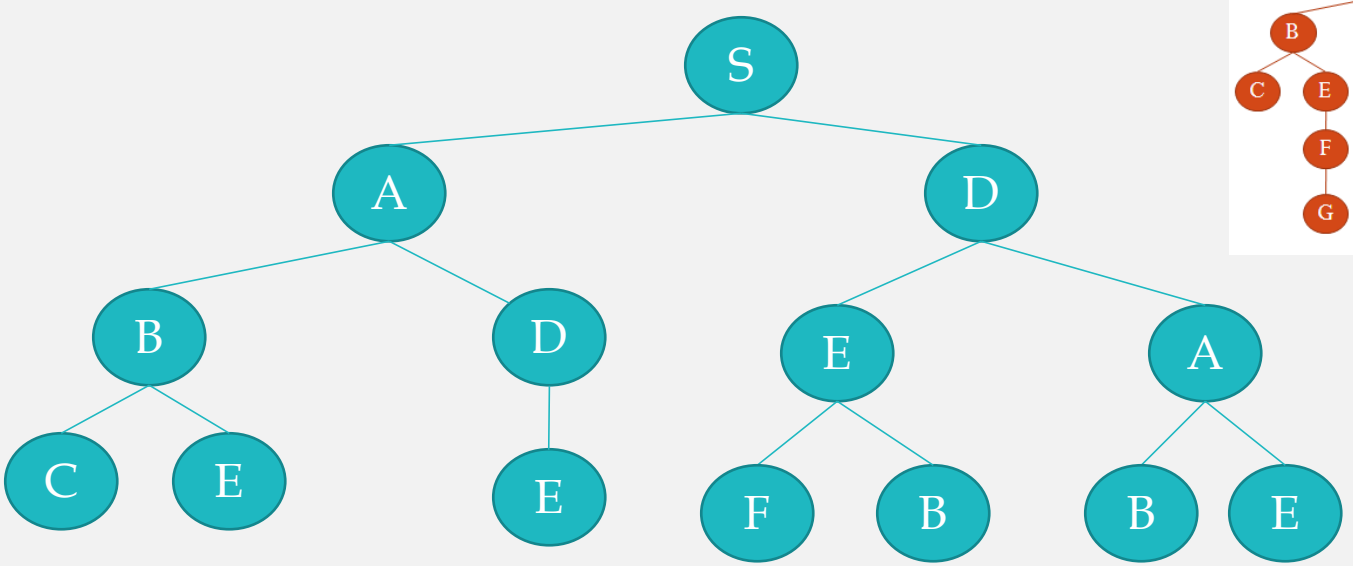


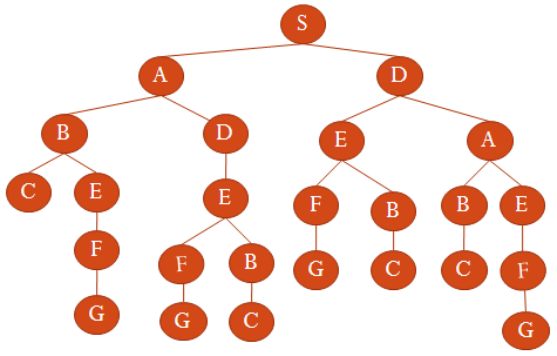




DLS

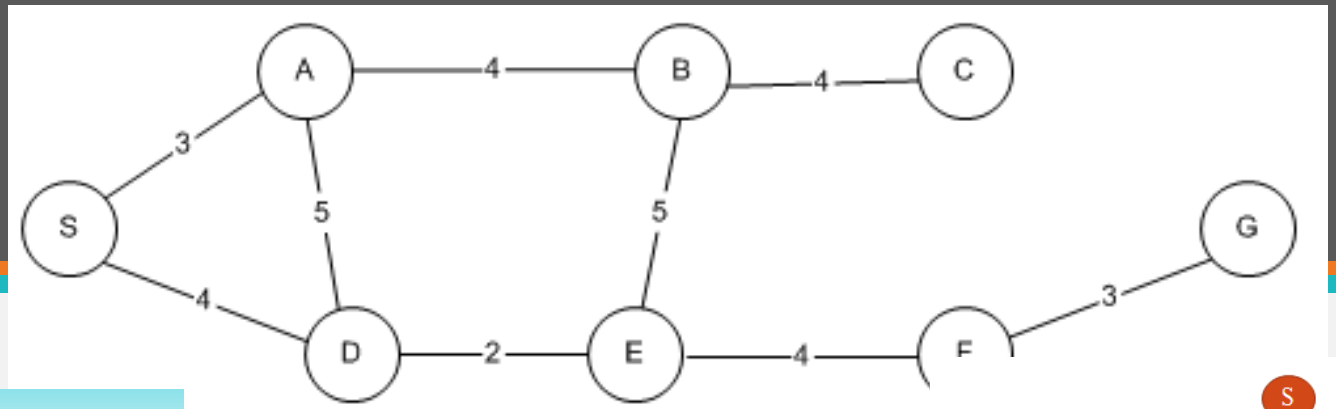

A
B
C
S
D
E
F
G
4
4
3
5
5
4
2
4
3


S
A
D
B
D
E
A
C
E
E
F
B
B
E


S
A
D
B
D
E
A
C
E
E
F
B
B
E
F
G
F
B
G
C
C
F
G

Iterative Deepening Search

- Secara iterative akan menggunakan Depth Limited Search dari kedalaman 0 sampai kedalaman n
- Merupakan penggabungan antara Breath First Search dan Depth First Search

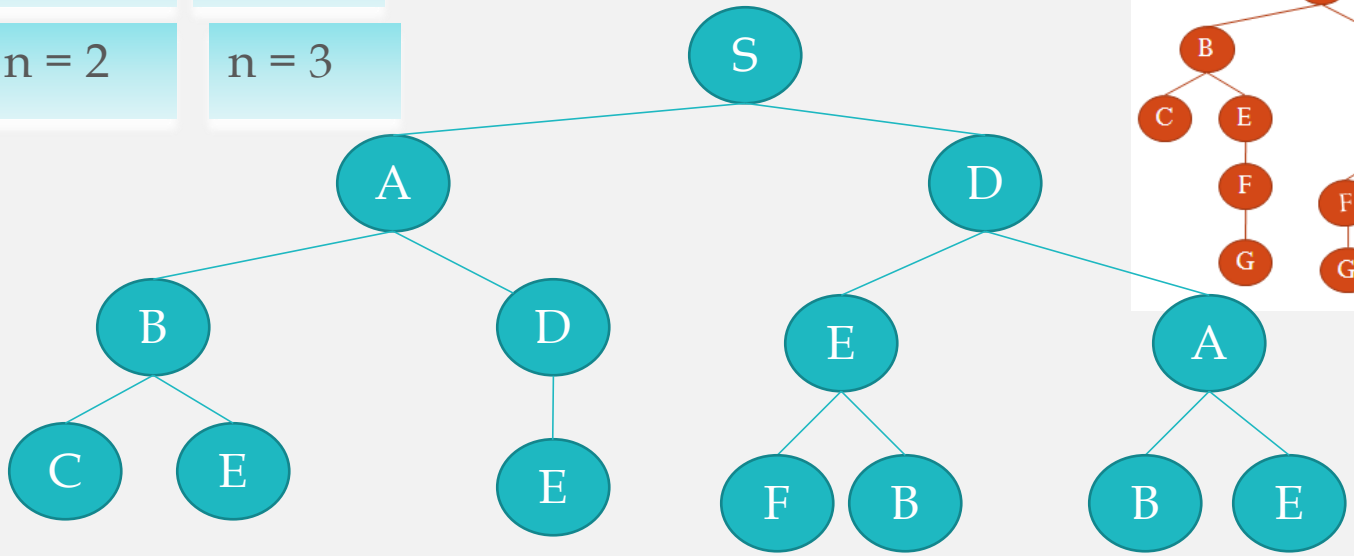

A
B
C
S
D
E
G
4
4
3
5
5
4
2
4
3

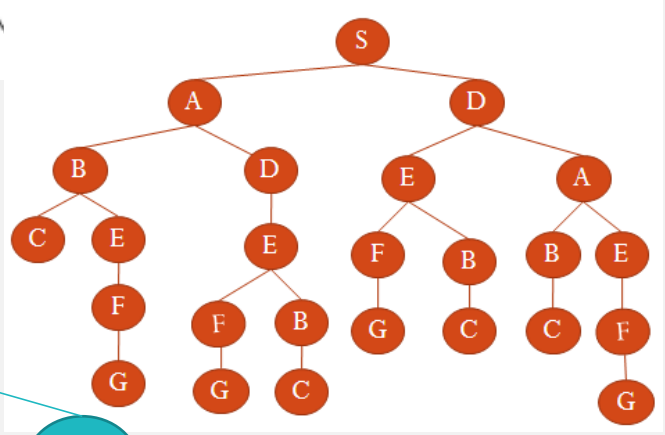
n = 0

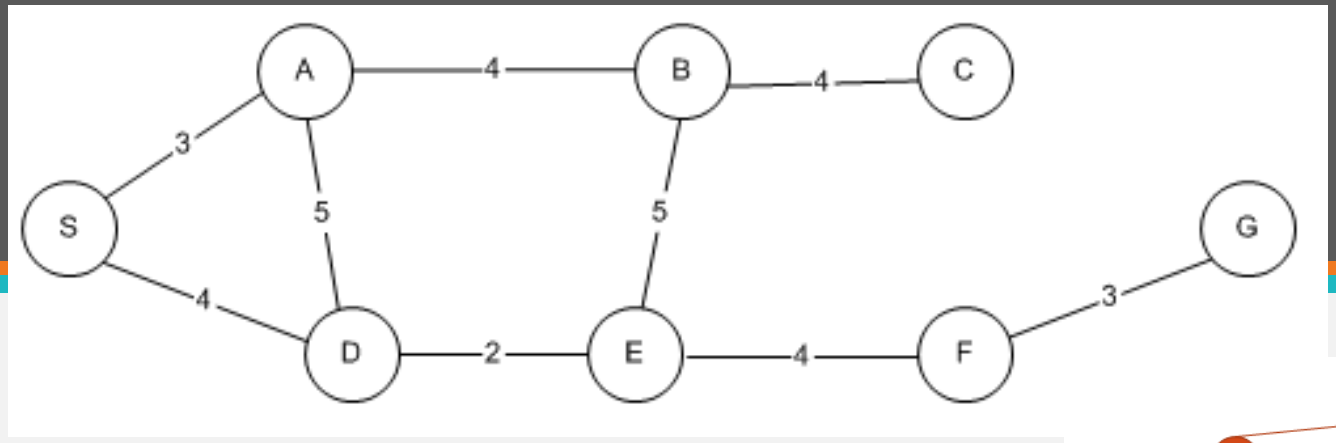
n = 1

n = 2

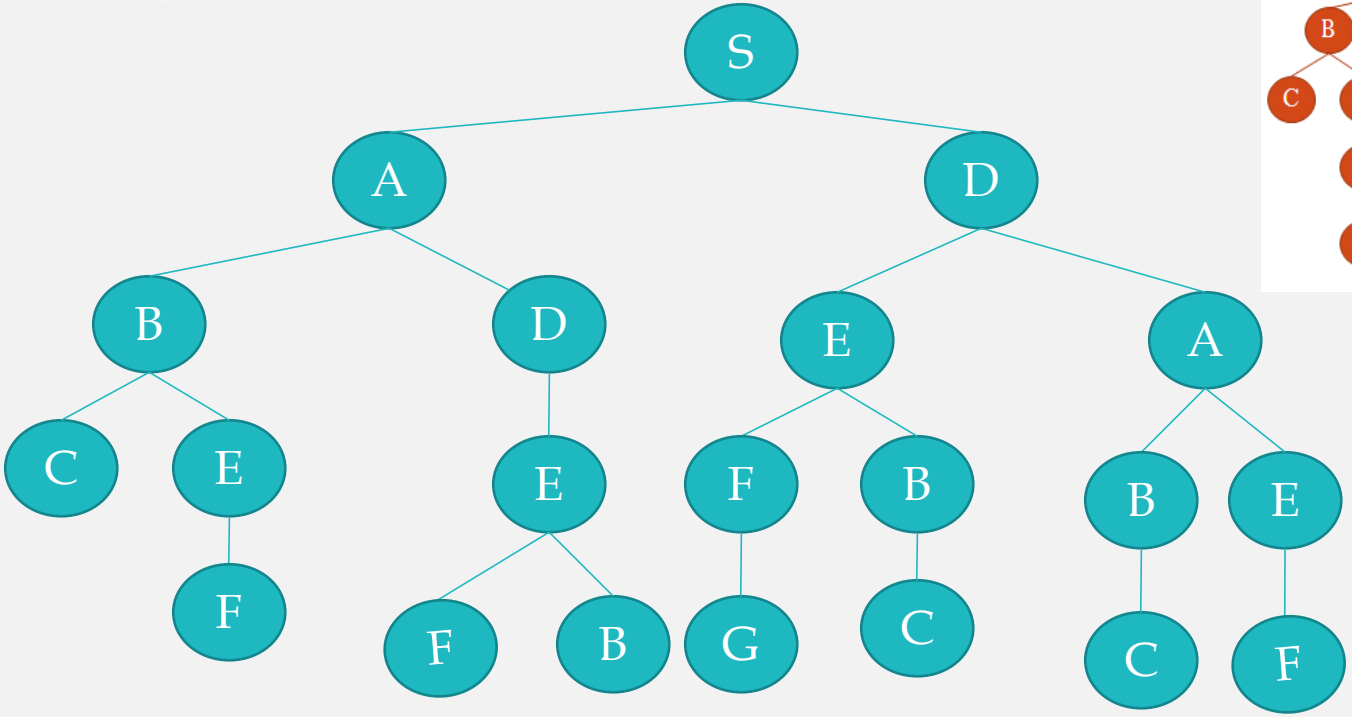
n = 3

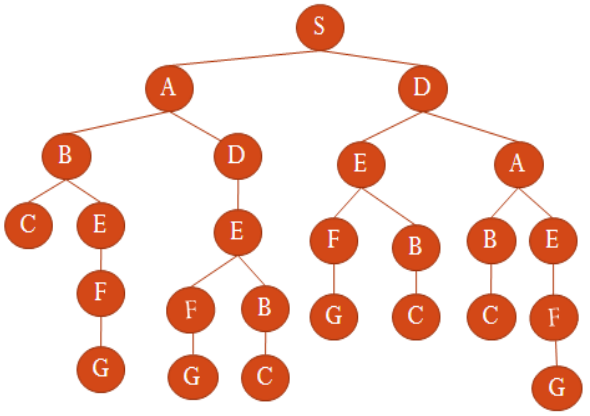

S
A
D
B
D
E
A
C
E
E
F
B
B
E

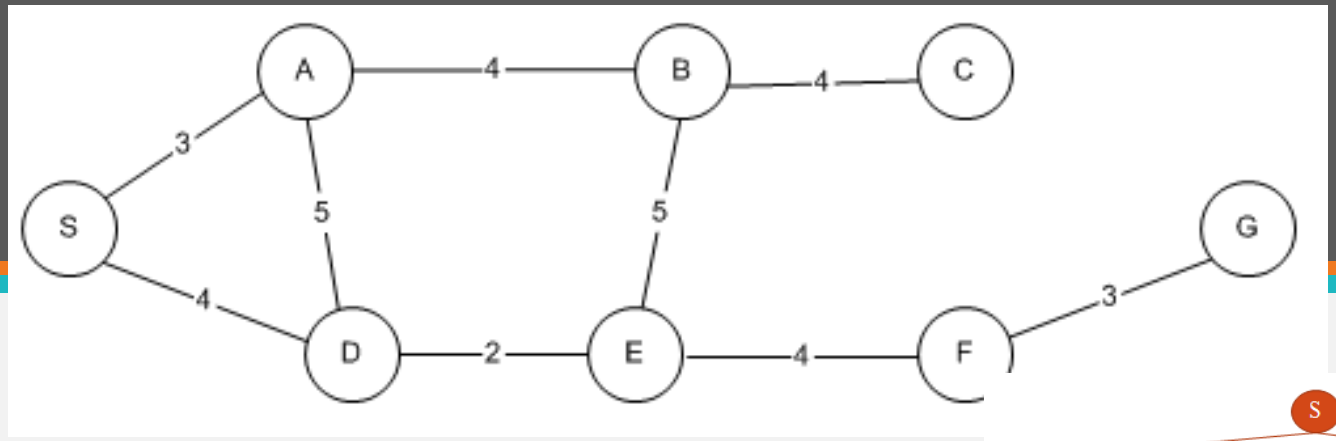

S
A
D
B
D
E
A
C
E
F
E
F
B
B
E
F
G
F
B
G
C
C
F
G
G
C


A
B
C
S
D
E
F
G
4
4
3
5
5
4
2
4
3

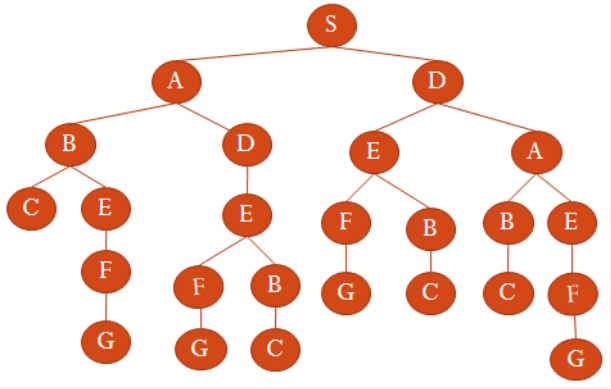
n = 4

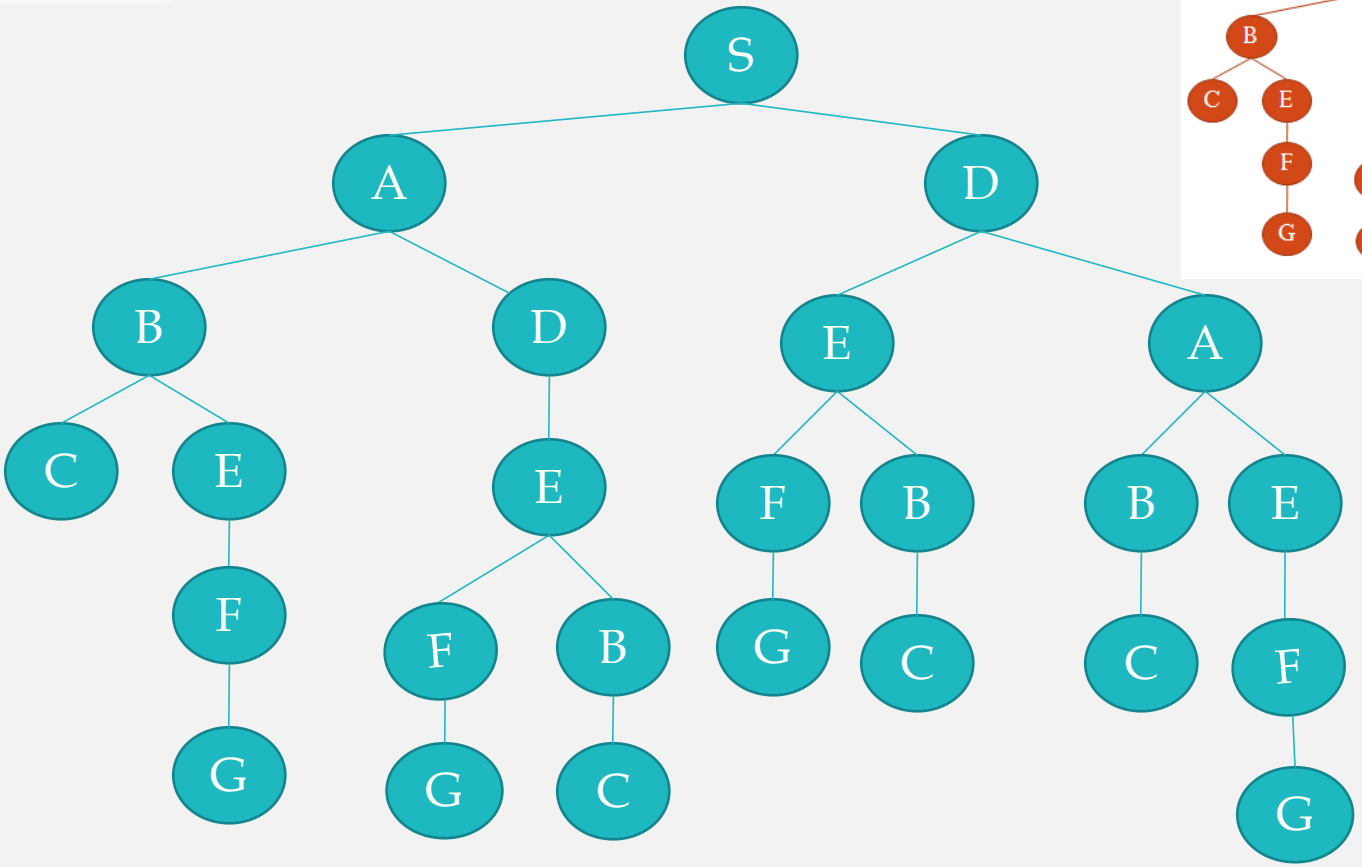

S
A
D
B
D
E
A
C
E
E
F
B
B
E
F
F
B
G
C
C
F


S
A
D
B
D
E
A
C
E
E
F
B
B
E
F
G
F
B
C
C
F
G


A
B
C
S
D
E
F
G
4
4
3
5
5
4
2
4
3

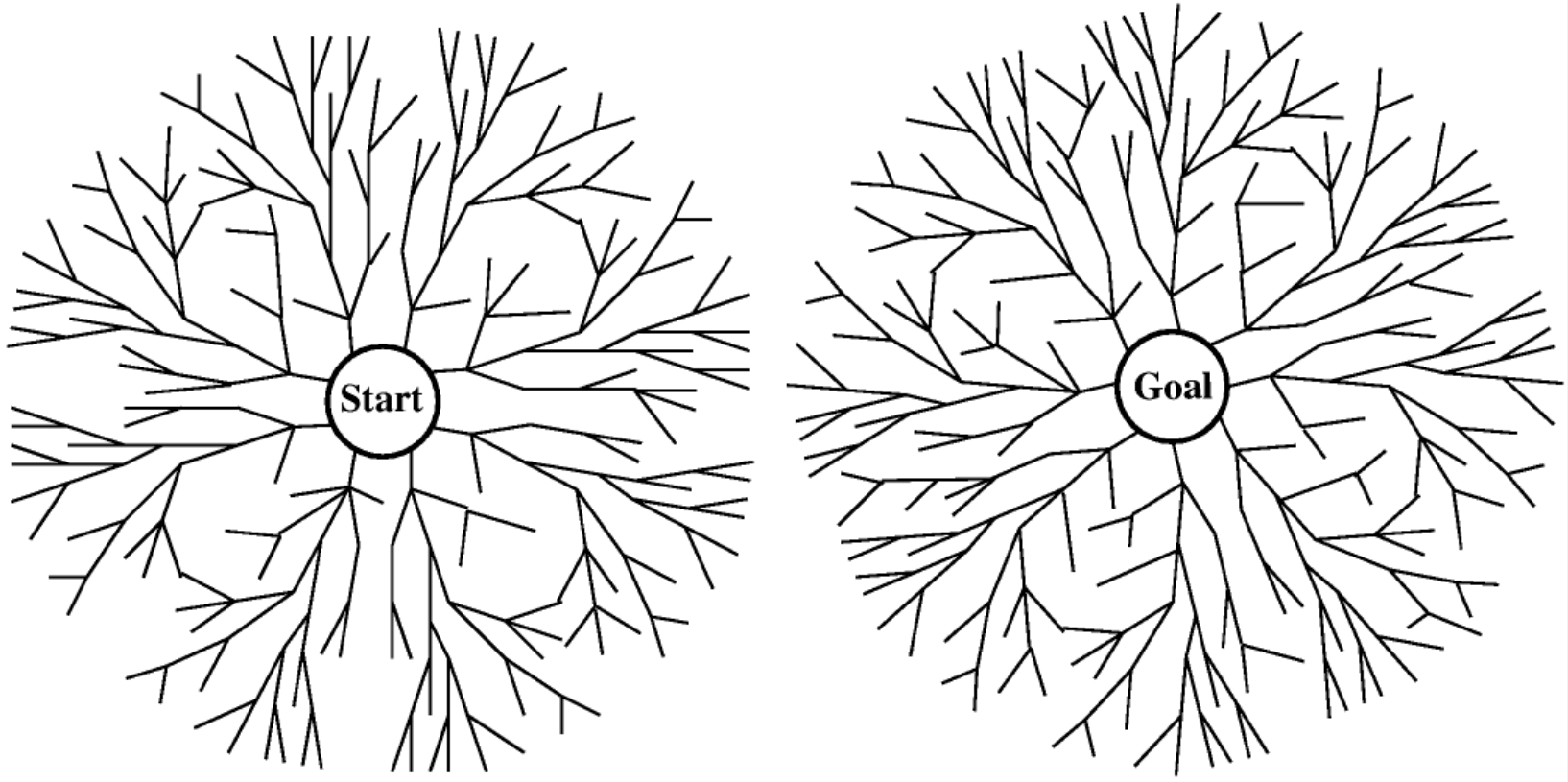
n = 5


S
A
D
B
D
E
A
C
E
E
F
B
B
E
F
B
F
B
G
C
C
F
G
G
C
G

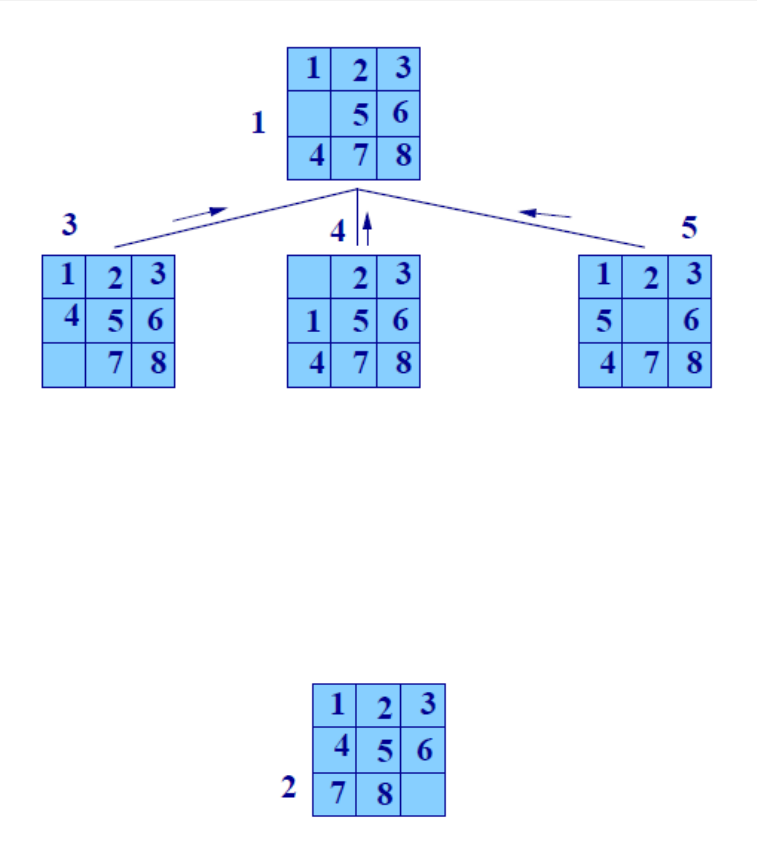

S
A
D
B
D
E
A
C
E
E
F
B
B
E
F
B
F
B
G
C
C
F
G
G
C
G

Bidirectional Search

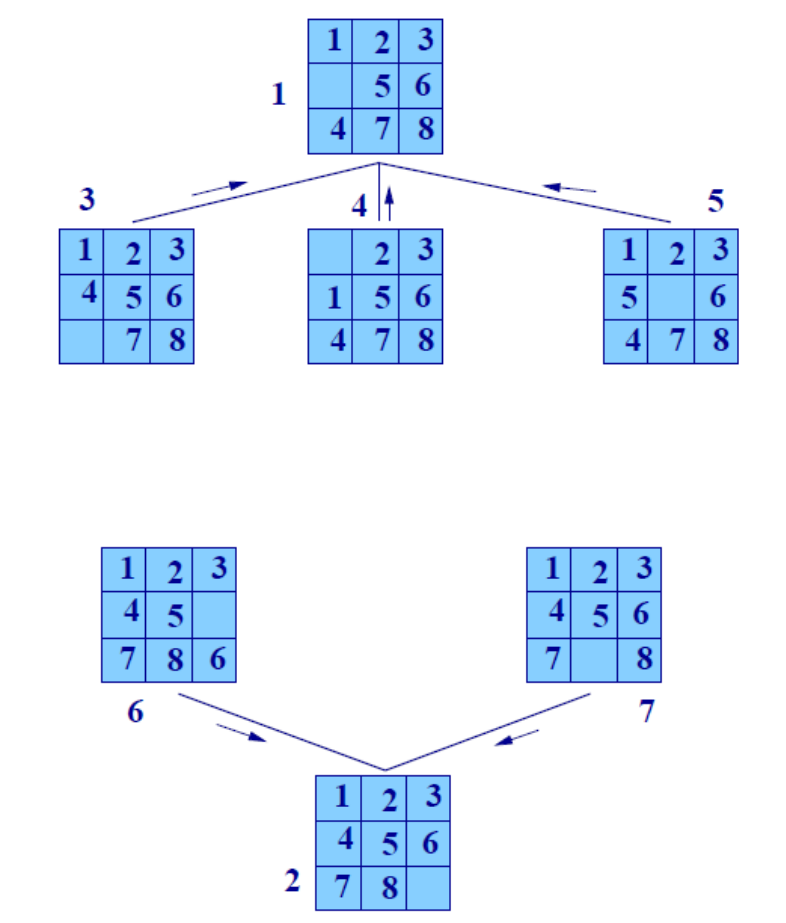
- To go both ways, Top Down and Bottom Up.



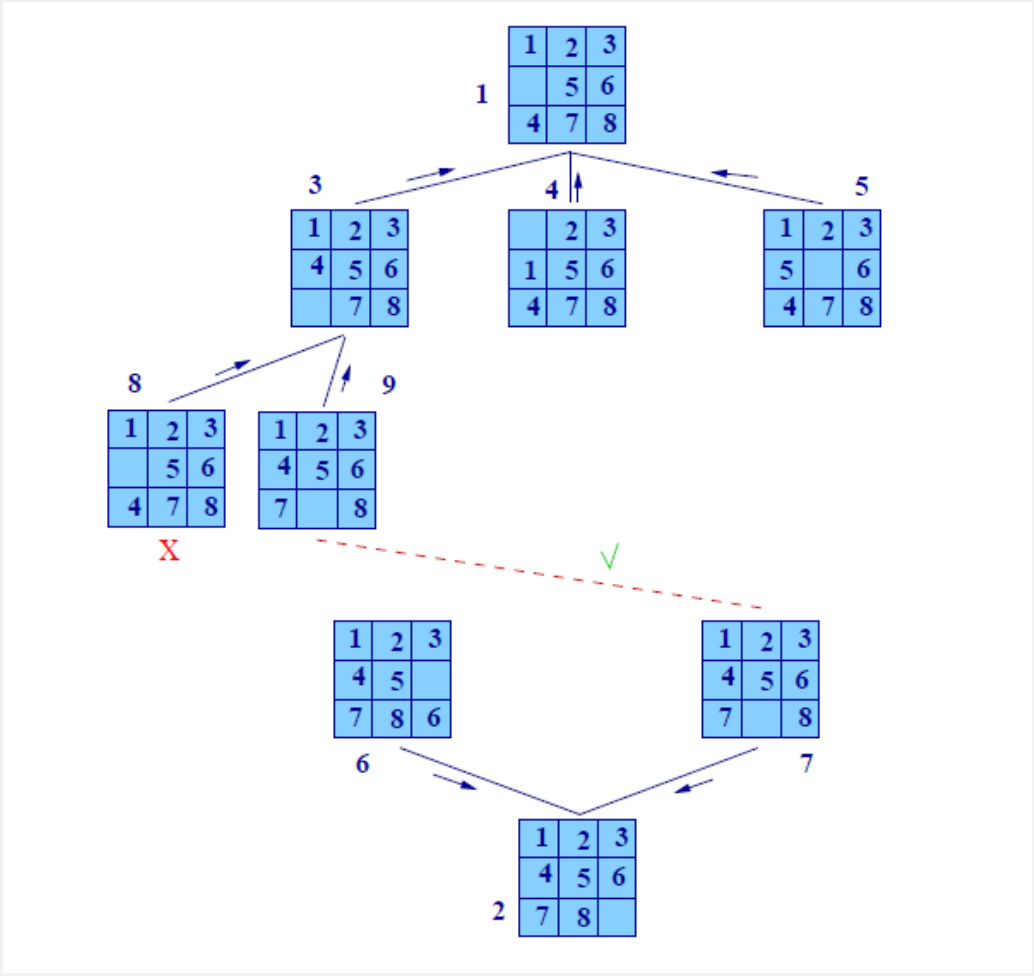
Contoh 8-puzzle



Contoh : 8-puzzle



Contoh : 8-puzzle



Hindari Repeated States

- Requires comparing state descriptions
- Breadth-first strategy:
  - Keep track of all generated states
  - If the state of a new node already exists, then discard the node

Hindari Repeated States

- Depth-first strategy:
  - Solution 1:
    - Keep track of all states associated with nodes in current tree
    - If the state of a new node already exists, then discard the node

  → Avoids loops

  - Solution 2:
    - Keep track of all states generated so far
    - If the state of a new node has already been generated, then discard the node

KUIS

- Kuis dulu ya....