Searching
(Algoritma Pencarian)

Kecerdasan Komputational

Algoritma Pencarian (Searching)

- Merupakan algoritma untuk mencari kemungkinan penyelesaian
- Sering dijumpai oleh peneliti di bidang AI
- Dalam melakukan pencarian, salah satu cara yang banyak digunakan untuk menggambarkan masalah adalah dengan mencantumkan atau menggambarkan semua kemungkinan keadaan yang ada.
- Memecahkan masalah berarti bergerak atau berpindah dari satu ruang (node) dari titik awal sampai titik yang dituju (ditentukan), oleh Karena itu kita memerlukan satu set operator untuk bergerak dari satu node ke node lainnya.

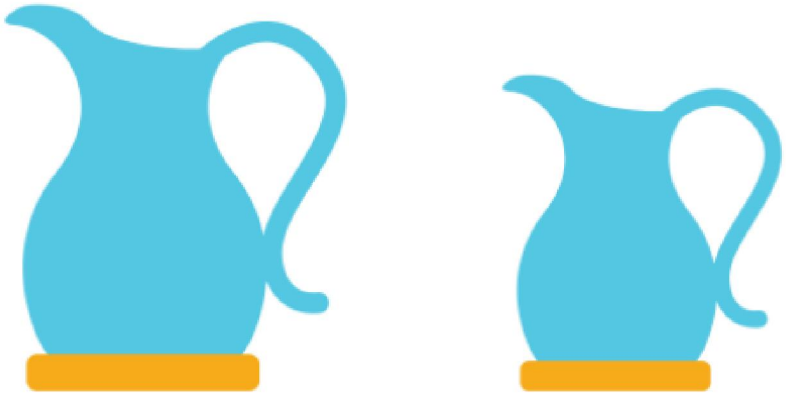
Mendefinisikan permasalahan

- Mendefinisikan suatu state space (ruang keadaan)
- Menetapkan satu atau lebih state awal
- Menetapkan satu atau lebih state tujuan
- Menetapkan rules (kumpulan aturan)
  - Generalisasi
  - Asumsi

ATRIBUT PENCARIAN:

- **Optimalisasi:** apakah algoritma dapat menemukan cost path terendah untuk mencapai goal?
- **Kelengkapan**: apakah algoritma akan menemukan jalur menuju goal/tujuan jika memang ada?
  - digunakan untuk mendefinisikan “return failure otherwise”
  - jika tidak ada solusi, dan algoritma tidak dapat mendeteksinya (mendeteksi state berulang-ulang(loop)), maka akan menjadi infinite search dan tidak akan ada hasil yang dikembalikan.
- **Kompleksitas Waktu**: waktu yang dibutuhkan untuk melakukan pencarian (contoh: jumlah nodes yang digeneralisasikan selama proses pencarian).
- **Kompleksitas waktu**: memori yang dibutuhkan.

Water jug problem: 4-gallon and 3-gallon



- no marker on the bottle
- pump to fill the water into the jug
- How can you get exactly 2 gallons of water into the 4-gallons jug?

A state space search

(x,y) : order pair

x : water in 4-gallons → x = 0,1,2,3,4

y : water in 3-gallons → y = 0,1,2,3

start state : (0,0)

goal state : (2,n) where n = any value

Rules :

1. Fill the 4 gallon-jug (4,-)
2. Fill the 3 gallon-jug (-,3)
3. Empty the 4 gallon-jug (0,-)
4. Empty the 3 gallon-jug (-,0)

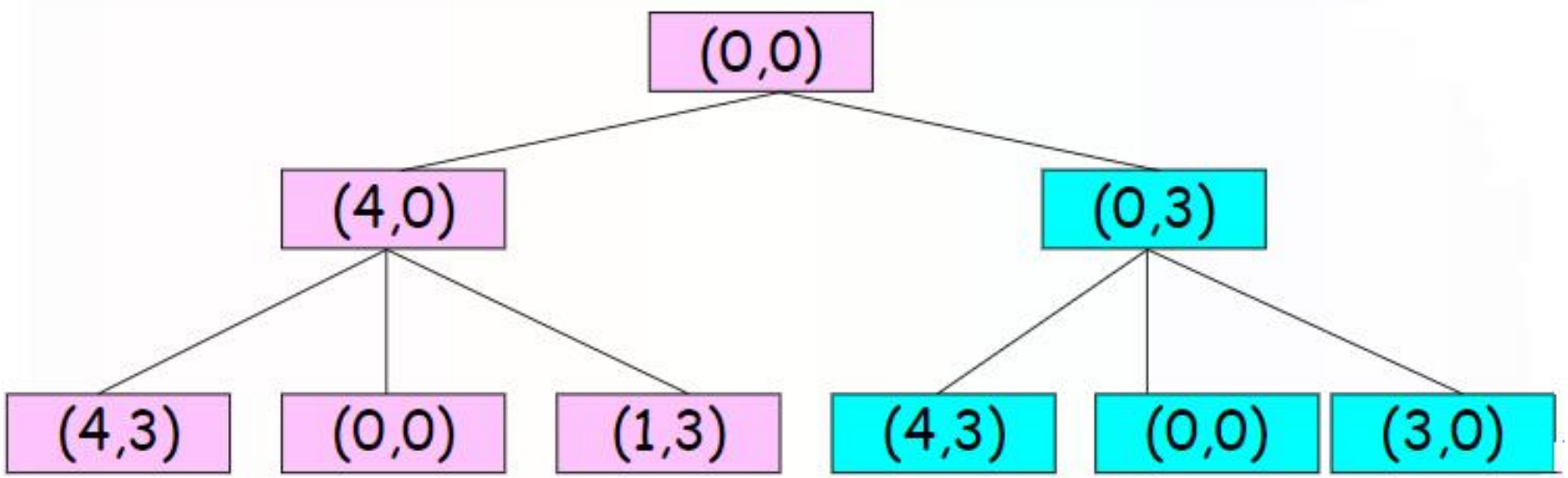
Water jug rules

Aturan Ke-	Kondisi Awal	Kondisi Tujuan	Aturan
1	(x,y) dng $x < 4$	(4,y)	Isi teko A sampai penuh
2	(x,y) dng $y < 3$	(x,3)	Isi teko B sampai penuh
3	(x,y) dng $x > 0$	(x-d,y)	Tuangkan sebagian air keluar dari teko A
4	(x,y) dng $y > 0$	(x,y-d)	Tuangkan sebagian air keluar dari teko B
5	(x,y) dng $x > 0$	(0,y)	Kosongkan teko A dengan membuang airnya ke tanah
6	(x,y) dng $y > 0$	(x,0)	Kosongkan teko B dengan membuang airnya ke tanah
7	(x,y) dimana $x+y \geq 4$ dan $y>0$	(4,y-(4-x)	Tuangkan air dari teko B ke teko A sampai teko A penuh
8	(x,y) dimana $x+y \geq 3$ dan $x>0$	(x-(3-y),3)	Tuangkan air dari teko A ke teko B sampai teko B penuh
9	(x,y) dimana $x+y \leq 4$ dan $y>0$	(x+y,0)	Tuangkan seluruh air dari teko B ke teko A
10	(x,y) dimana $x+y \leq 3$ dan $x>0$	(0,x+y)	Tuangkan seluruh air dari teko A ke teko B
11	(0,2)	(2,0)	Tuangkan 2 galon air dari teko B ke teko A
12	(2,y)	(0,y)	Kosongkan 2 galon air di teko A dengan membuang airnya ke tanah

A water jug solution

Solution : path / plan

Search Tree



Water jug problem.

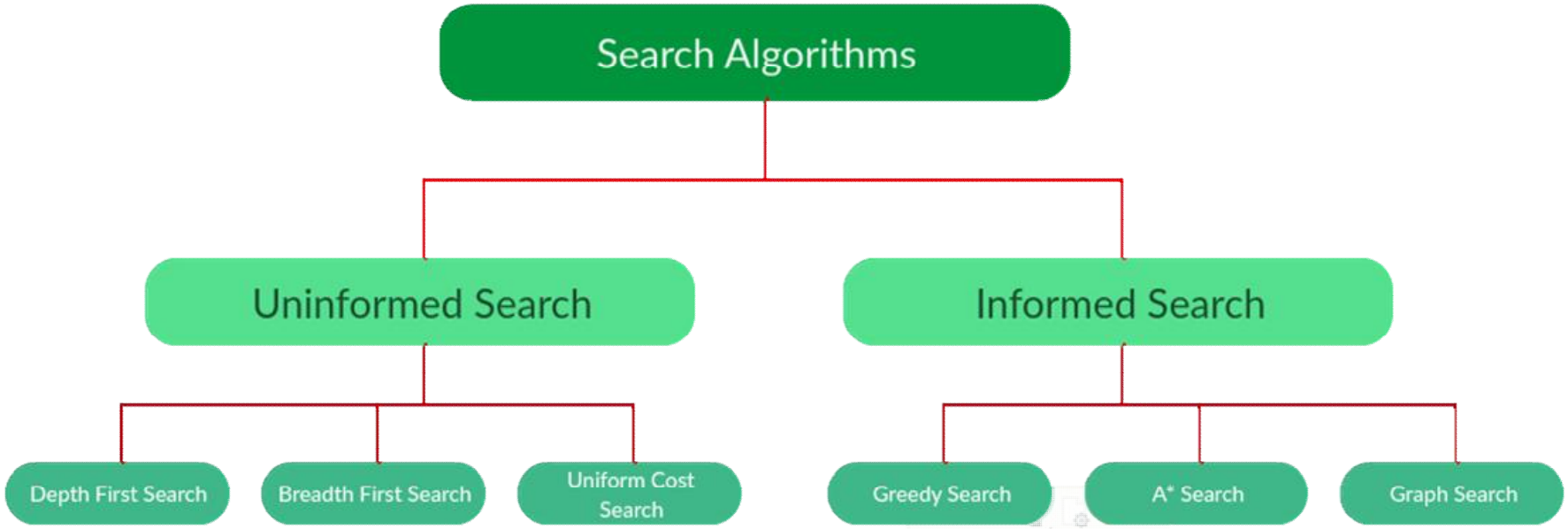
Kriteria mengukur perfomansi metode pencarian

- ***Completeness*** : apakah metode tersebut menjamin penemuan solusi jika solusinya memang ada?
- ***Time complexity*** : berapa lama waktu yang diperlukan? [semakin cepat, semakin baik]
- ***Space complexity*** : berapa banyak memori yang diperlukan
- ***Optimality*** : apakah metode tersebut menjamin menemukan solusi yang terbaik jika terdapat beberapa solusi berbeda?

Metode searching

- Terdapat dua metode untuk melakukan pencarian yaitu dengan
  - BLIND SEARCH (Uninformed Searching)
  - HEURISTIC SEARCH (Informed Searching)

Types of search algorithms



BLIND SEARCHING

(Informed Searching)

Uninformed Search Algorithms

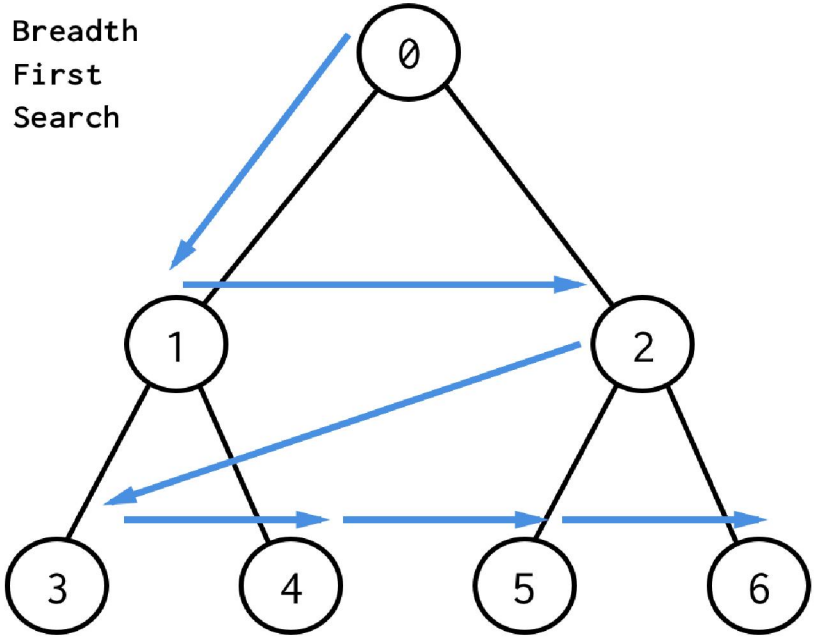
- The search algorithms in this section have **no additional information** on the goal node other than the one provided in the problem definition.
- The plans to reach the goal state from the start state differ **only by the order and/or length of actions**.
- Uninformed search is also called **Blind search**.
- The following uninformed search algorithms are discussed in this section.
  - Depth First Search
  - Breath First Search
  - Uniform Cost Search

Blind searching

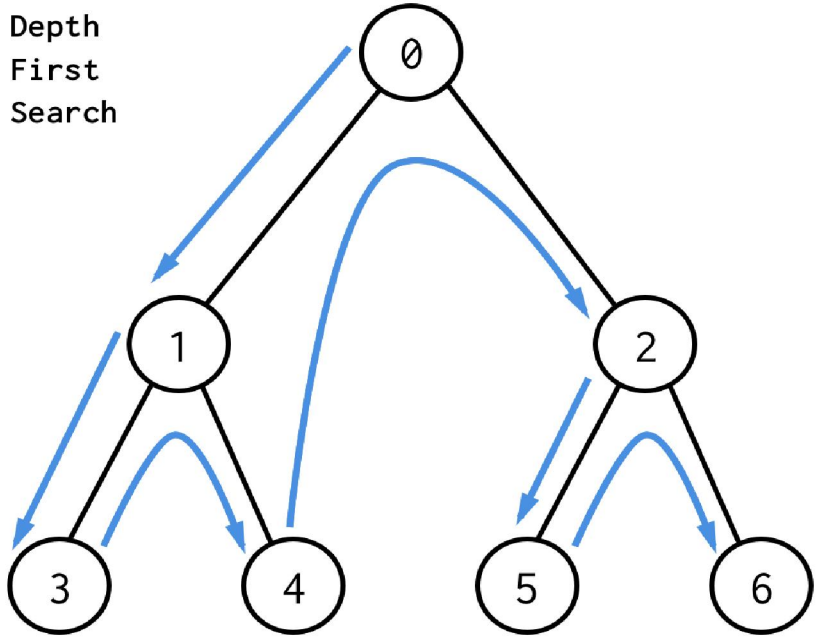
- **Blind Search** atau **Uninformed Search** secara umum mengartikan bahwa saat proses pencarian kita **tidak memiliki clue/hint** apakah hasil yang ditemukan lebih baik daripada yang lainnya, sehingga kita tidak mengetahui apakah hasil dari eksplorasi tersebut bermanfaat secara maksimal atau tidak.
- Dengan kata lain, dikatakan "**blind**" karena **tidak ada informasi awal yang digunakan dalam proses pencarian**.
- Search Space (ruang pencarian) dieksplorasi tanpa memanfaatkan apapun informasi yang menyangkut pada masalah maka dari itu digunakan istilah blind search atau naive search, dan karena metode ini masih sangat umum maka hasil yang didapat secara instrinsik **kurang efisien**.

Common blind search algorithms

1. Breadth First Search



2. Depth First Search



1. BREADTH-FIRST SEARCH

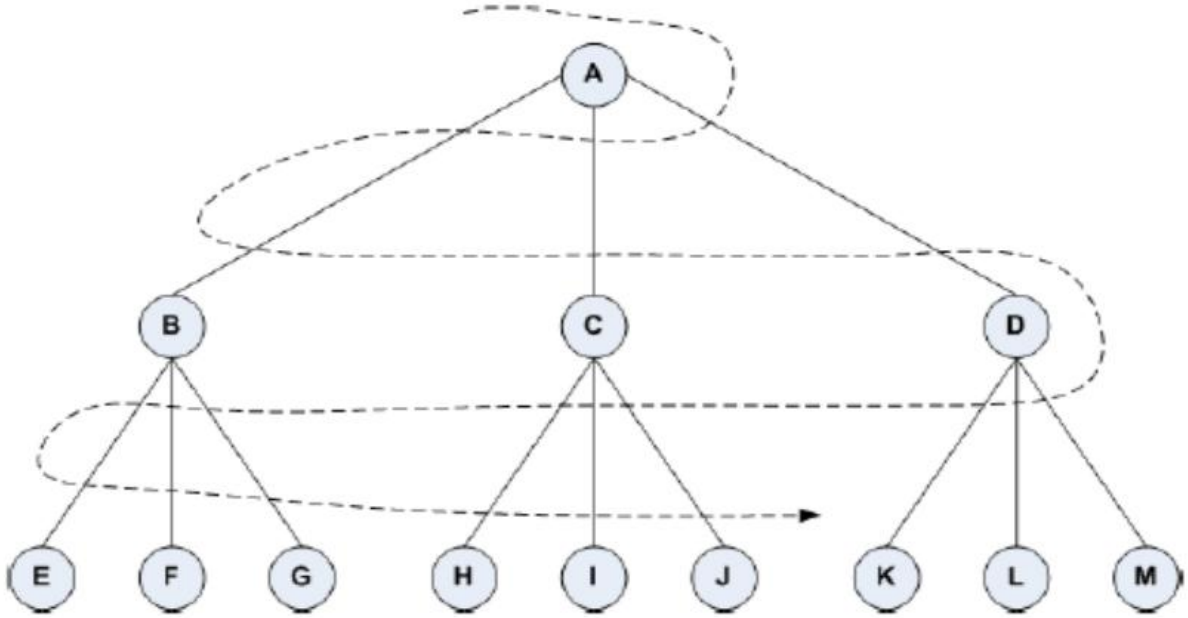
- Breadth First Search (BFS) juga memiliki alur algoritma yang paling sederhana dibandingkan dengan metode blind yang lain.
- Kesederhanaan algoritma menjadi alasan mengapa BFS selalu dipelajari lebih dulu ketika membahas masalah pencarian.
- Breadth-First Search (BFS) dimulai dari akar node (**root**) lalu mengeksplor ke seluruh cabang (branch) dalam level yang sama.
- Melakukan proses **pencarian** pada semua node yang berada **pada level atau hirarki yang sama terlebih** dahulu sebelum **melanjutkan** proses pencarian pada **node di level berikutnya**.
- BFS akan mencari satu per satu node secara **melebar dari kiri ke kanan** secara berurutan berdasarkan tingkat level nodenya.
- Jika pada satu level **belum ditemukan solusi yang diinginkan**, maka pencarian **dilanjutkan** hingga level berikutnya.
- Demikian seterusnya hingga ditemukan solusi.
- Maka, dengan cara seperti ini, BFS menjamin ditemukannya solusi apabila solusinya memang ada.

BREADTH-FIRST SEARCH

- Sifat:
  - BFS dikatakan komplit/selesai jika terdapat solusi dan BFS menemukannya.
  - BFS dikatakan optimal jika solusi yang didapat dapat dipastikan menjadi jalur terpendek (shortest path).
  - Algoritma dapat diimplementasikan dengan First In First Out (FIFO) Queue.

BREADTH-FIRST SEARCH

- Eksplorasi node dimulai root (A) lalu bergerak ke kanan untuk mencari node pada level yang sama, jika sudah tidak ada, maka akan ke level selanjutnya dimulai dari kiri-kanan sampai menemukan goal (tujuan).

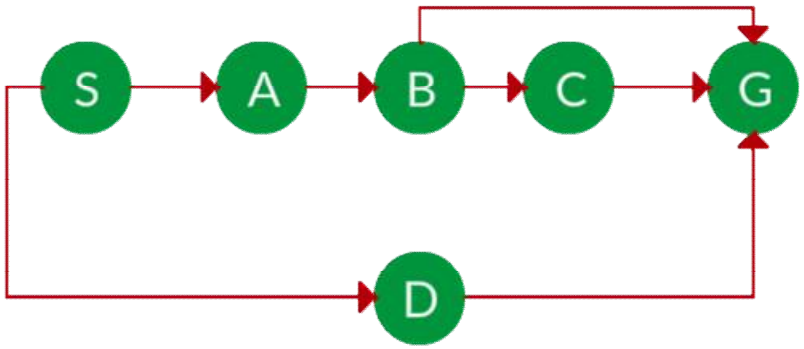


ALGORITMA:

```
List open, closed, successors={};
Node root_node, current_node;
insert-last(root_node,open)
while not-empty(open);
        current_node=remove-first(open);
        insert-last(current_node,closed);
        if (goal(current_node))
                return current_node;
        else
                successors=successorsOf(current_node);
        for(x in successors)
                if(not-in(x,closed)) insert-last(x,open);
        endIf
endWhile
```

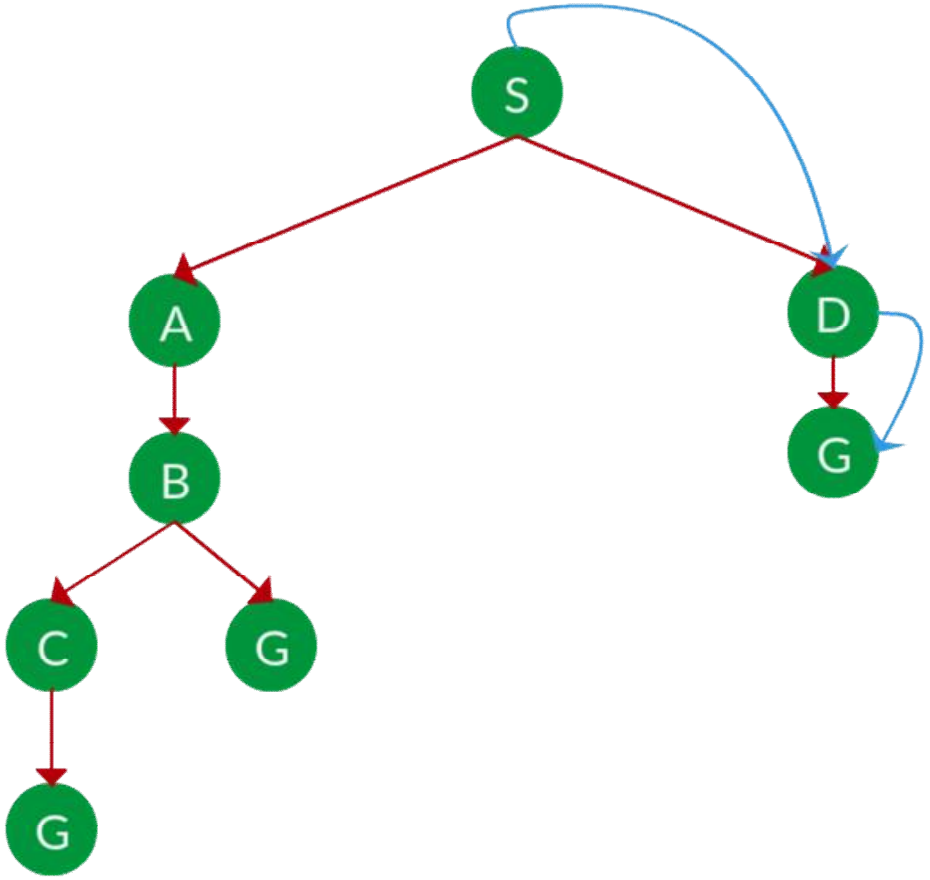
Example 1. Breadth First Search

Problem: Which solution would BFS find to move from node S to node G if run on the graph below?



Route : S -> A -> D -> B -> G
Result Path : S -> D -> G

Solution: The equivalent search tree for the above graph is as follows. As BFS traverses the tree "shallowest node first", it would always pick the shallower branch until it reaches the solution (or it runs out of nodes, and goes to the next branch). The traversal is shown in blue arrows.



Breadth First Search

- Let s = the depth of the shallowest solution.
- n^i = number of nodes in level i.
- **Time complexity:** Equivalent to the number of nodes traversed in BFS until the shallowest solution.

$$T(n) = 1 + n^2 + n^3 + \ldots + n^s = O(n^s)$$

- **Space complexity:** Equivalent to how large can the fringe get.

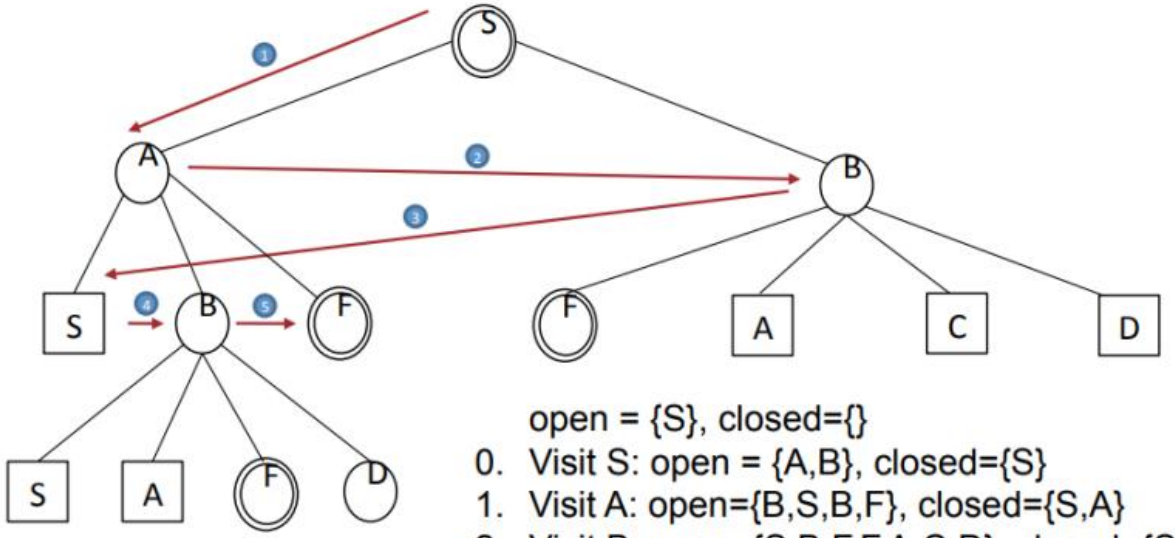
$$S(n) = O(n^s)$$

- **Completeness:** BFS is complete, meaning for a given search tree, BFS will come up with a solution if it exists.
- **Optimality:** BFS is optimal as long as the costs of all edges are equal.

Example 2: Breadth First Search

- Initial State: S
- State Goal: F

Keterangan:
Terdapat 3 buah node F namun, BFS akan mengembalikan node F yang pertama kali ditemukan yaitu dari parent node A. Karena ini merupakan BLIND SEARCH yang berarti pencarian tidak mendapatkan clue/petunjuk sehingga hasil pencarian belum tentu mendapatkan hasil yang paling efisien.

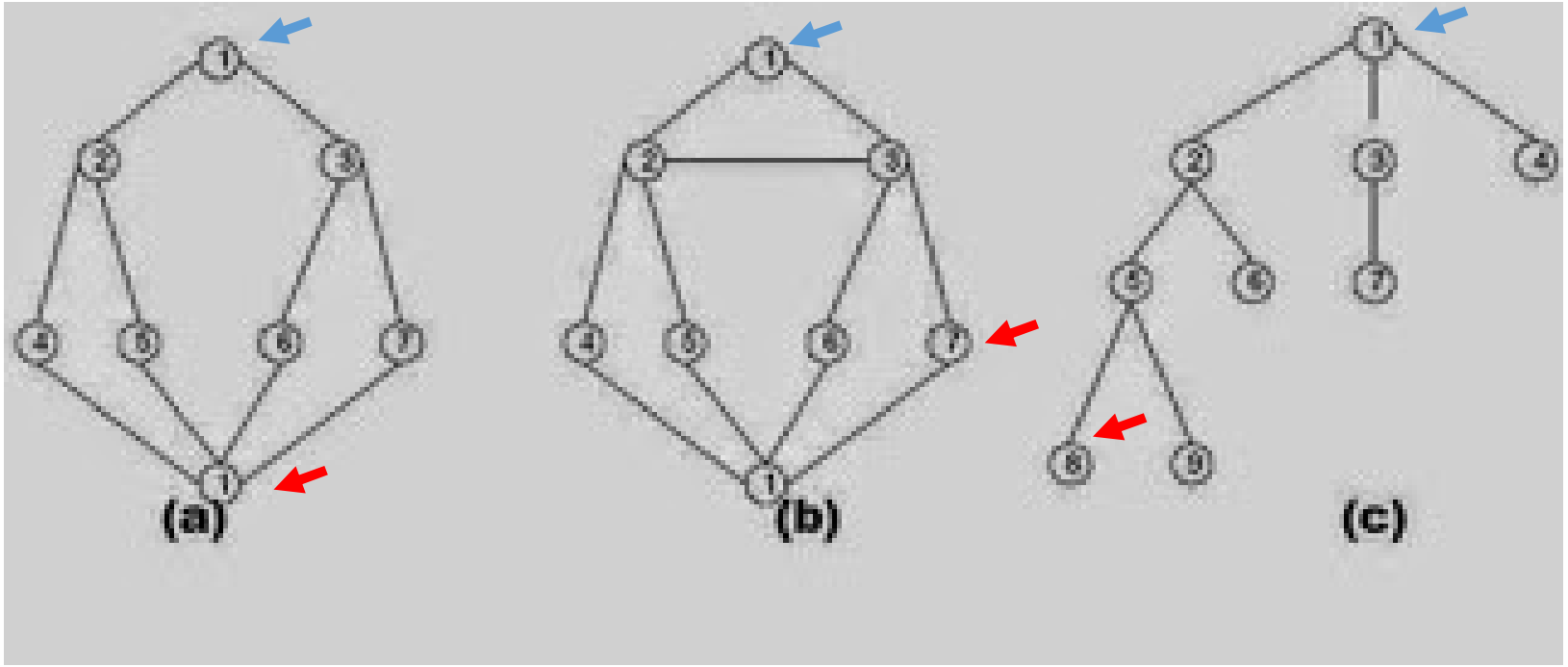


open = {S}, closed={}
0. Visit S: open = {A,B}, closed={S}
1. Visit A: open={B,S,B,F}, closed={S,A}
2. Visit B: open={S,B,F,F,A,C,D}, closed={S,A,B}
3. Visit S: open={B,F,F,A,C,D}, closed={S,A,B,S}
4. Visit B: open={F,F,A,C,D,S,A,C,D}, closed= {S,A,B,S,B}
5. Visit F: Goal Found!

Rute : S -> A -> B -> S -> B -> F
HASIL: S -> A -> F

Example 3: Breadth First Search

- Gambar a : Tentukan jalur terpendek dari simpul 1 hingga kembali ke simpul 1 lagi.
- Gambar b : Tentukan rute dari node 1 hingga node 7.
- Gambar c : Tentukan lintasan terpendek dari kota 1 ke kota 8.



Rute yang dilewati adalah :
Gbr a : 1 - 2 - 3 - 4 - 5 - 6 - 7 – 1
Result Path?

Gbr b : 1 - 2 - 3 - 4 - 5 - 6 – 7
Result Path?

Gbr c : 1 - 2 - 3 - 4 - 5 - 6 - 7 – 8
Result Path?

Breadth First Search

Keuntungan

- Tidak akan menemui jalan buntu
- Menjamin ditemukannya solusi (jika solusinya memang ada) dan solusi yang ditemukan pasti yang paling baik
- Jika ada satu solusi maka bread-first search akan menemukannya

Kelemahan

- Membutuhkan memori yang cukup banyak
- Membutuhkan waktu yang cukup lama

2. Depth First Search

- Depth-First Search melakukan eksplorasi **dimulai root node** (akar) lalu ke cabang pertama (dimulai dari paling kiri) **sampai ke kedalaman maksimum**, setelah itu baru dilakukan eksplorasi ke cabang lainnya, dan terus dilakukan sampai menemukan state goal.
- Jika telah mencapai node yang terdalam namun tidak juga ditemukan solusi, maka akan mundur sampai menemukan cabang yang belum dieksplorasi.
- Tree dilakukan pencarian dengan **top-to-bottom, left-to-right**.

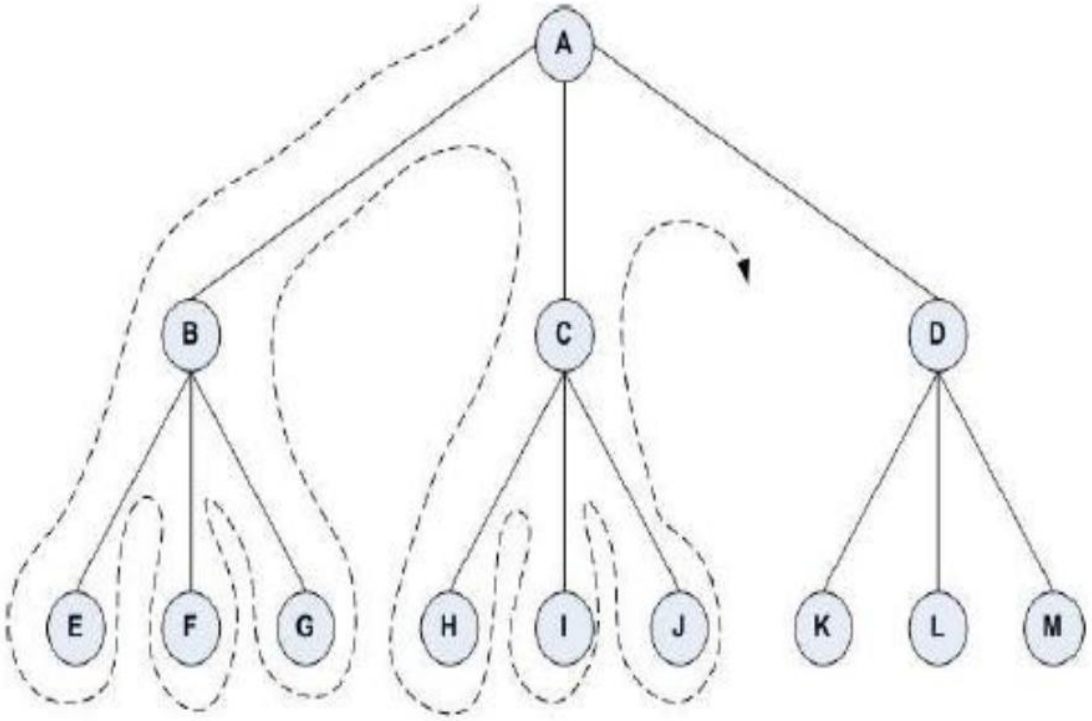
Depth First Search dikatakan tidak komplit:

- Jika siklus yang disajikan dalam graf lalu DFS melakukan eksplorasi terhadap siklus berulang (tidak ada batas).
- Jika tidak terdapat siklus, maka algoritma komplit.
- Efek siklus dapat dibatasi dengan memaksakan kedalaman pencarian yang maksimal (namun algoritma tetap tidak komplit).

- Depth-First Search dikatakan **tidak optimal jika** solusi pertama yang ditemukan bukan merupakan solusi dengan shortest path (jalur terpendek) menuju goal yang telah ditentukan.
- Algoritma dapat diimplementasikan dengan menggunakan Last In First Out (LIFO) stack atau menggunakan Rekursif (recursion).

Cara Kerja Algoritma Depth First Search

Eksplorasi dimulai dari **root node** (**S**) lalu ke cabang pertamanya (**B**), setelah itu ke cabang (child) pertama node B yaitu **E**, karena node E tidak memiliki child maka ke cabang kedua node B yaitu **F**, dan dilakukan dengan cara seperti itu sampai menemukan state goal yang telah ditapkan.

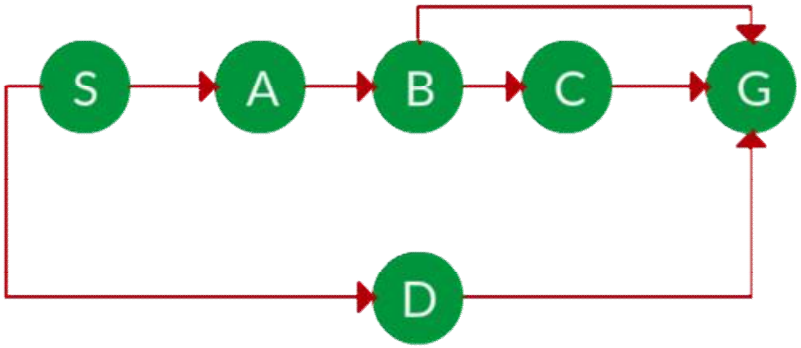


Algoritma Depth First Search

```
List open, closed, successors={};
Node root_node, current_node;
insert-first(root_node,open)
while not-empty(open);
        current_node= remove-first(open);
        insert-first (current_node,closed);
        if (goal(current_node))
                return current_node;
        else
                successors=successorsOf(current_node);
        for(x in successors)
                if(not-in(x,closed))
                        insert-first(x,open);
                endIf
endWhile
```

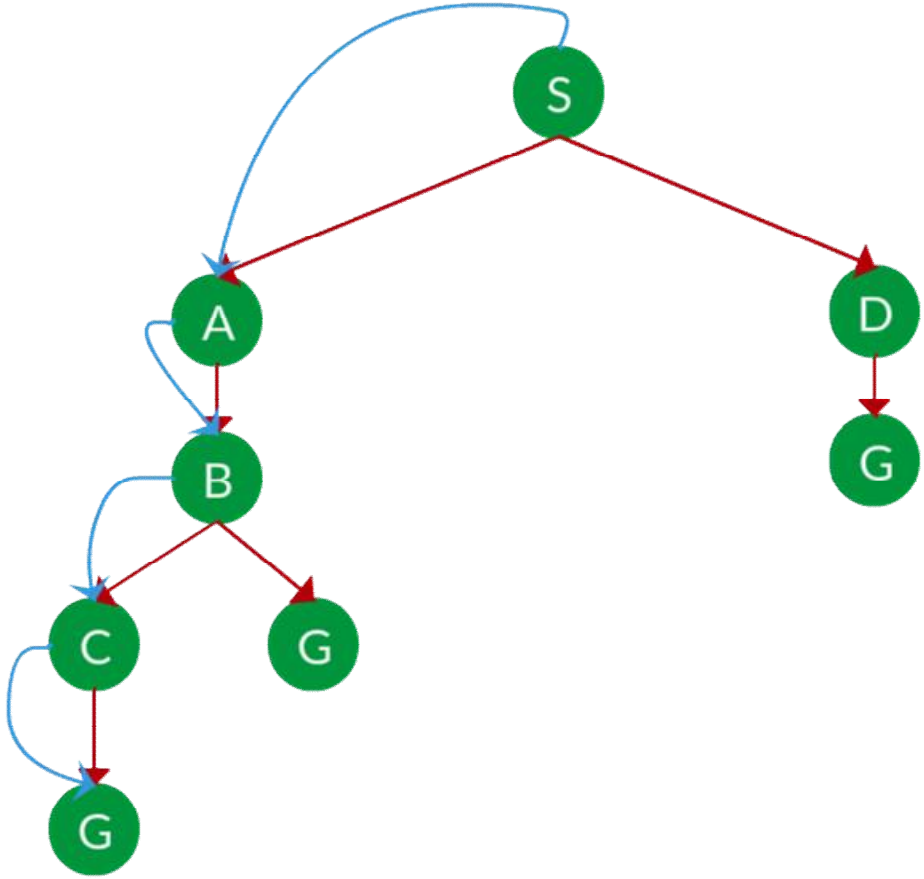
Exmple 1:
Depth First Search

Problem: Which solution would DFS find to move from node S to node G if run on the graph below?



Route : **S -> A -> B -> C -> G**
Result Path: **S -> A -> B -> C -> G**

Solution: The equivalent search tree for the above graph is as follows. As DFS traverses the tree “deepest node first”, it would always pick the deeper branch until it reaches the solution (or it runs out of nodes, and goes to the next branch). The traversal is shown in blue arrows.



Depth First Search

- Let d = the depth of the search tree = number of levels of the search tree.
- n^i = number of nodes in level i.
- **Time complexity:** Equivalent to the number of nodes traversed in DFS.

$$T(n) = 1 + n^2 + n^3 + \ldots + n^d = O(n^d)$$

- **Space complexity:** Equivalent to how large can the fringe get.

$$S(n) = O(n \times d)$$

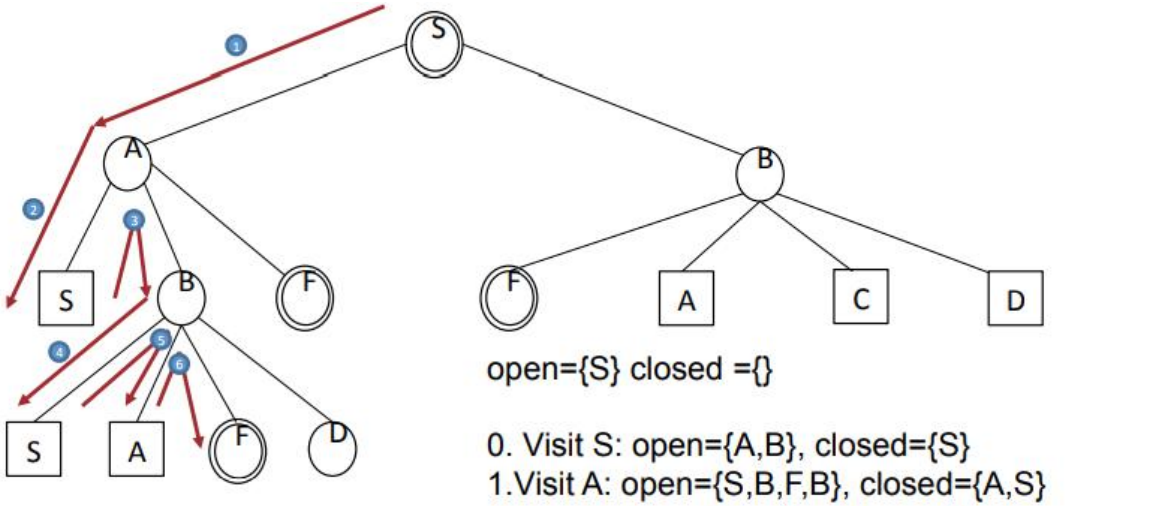
- **Completeness:** DFS is complete if the search tree is finite, meaning for a given finite search tree, DFS will come up with a solution if it exists.
- **Optimality:** DFS is not optimal, meaning the number of steps in reaching the solution, or the cost spent in reaching it is high.

Example 2: Depth First Search

- **INITIAL STATE : S**
- **STATE GOAL : F**

- Dimulai dari root node (S) selanjutnya berpindah ke cabang (child) pertamanya (paling kiri) yaitu node A, lalu berpindah ke cabang pertama yang dimiliki node yaitu node S. Karena node S tidak memilki child maka berpindah ke child selanjtnya dari node A yaitu B, lalu berpindah ke child pertama cabang B yaitu S, karena node S tidak memiliki child maka berpindah ke child kedua miilik node B yaitu A, dan karena node A juga tidak memiliki child maka berpindah ke node ketiga cabang milik B yaitu F, karena kita telah menemukan state goal yaitu node F, maka search berhenti dan mengembalikkan hasil S-A-B-F.

- **RUTE : S -> A -> S -> B -> S -> A -> F**
- **HASIL : S -> A -> B -> F**

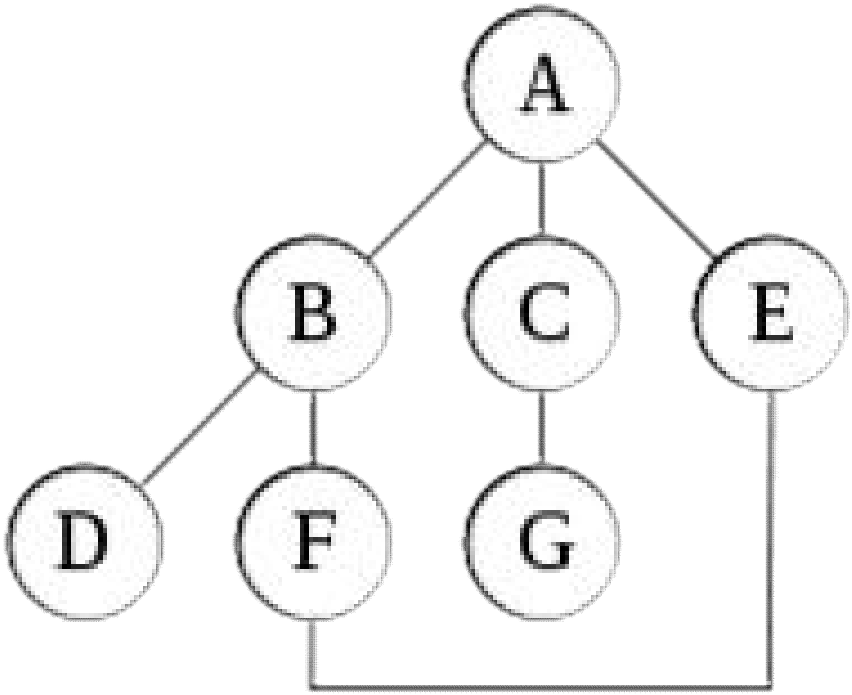


open={S} closed ={}

0. Visit S: open={A,B}, closed={S}
1.Visit A: open={S,B,F,B}, closed={A,S}
2.Visit S: open={B,F,B}, closed={S,A,S}
3.Visit B: open={S,A,F,D,F,B}, closed={B,S,A,S}
4.Visit S: open={A,F,D,F,B}, closed={S,B,S,A,S}
5.Visit A: open={F,D,F,B}, closed={A,S,B,S,A,S}
6.Visit F: GOAL Reached!

Example 3: Depth First Search (with loop)

- Pencarian mendalam-pertama mulai dari A, dengan asumsi bahwa tepi kiri dalam grafik ditunjukkan dipilih sebelum tepi kanan, dan dengan asumsi pencarian sebelumnya-ingat node dikunjungi dan tidak akan mengulangi mereka (karena ini adalah grafik kecil), akan mengunjungi node dalam urutan sebagai berikut: A, B, D, F, E, C, G.
- Melakukan pencarian yang sama tanpa mengingat hasil sebelumnya mengunjungi node dalam mengunjungi node dalam urutan A,B, D, F, E, A, B, D, F, E, dll selamanya, terperangkap dalam A, B, D, F , E siklus dan tidak pernah mencapai C atau G.
- Iteratif memperdalam mencegah **loop** ini dan akan mencapai node berikut pada kedalaman berikut, dengan asumsi itu hasil dari kiri-ke-kanan seperti di atas:



Keuntungan

- Memori yang relatif kecil
- Secara kebetulan, akan menemukan solusi tanpa harus menguji lebih banyak lagi

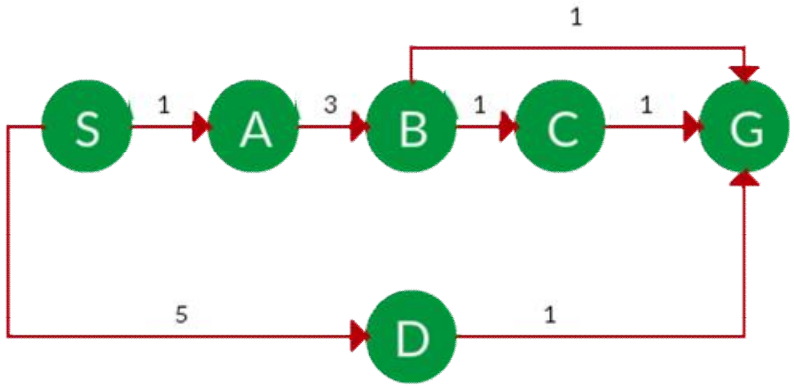
Uniform Cost Search

- UCS is **different from BFS and DFS** because here **the costs come into play**.
- In other words, traversing via different edges might not have the same cost.
- The **goal is to find a path where the cumulative sum of costs is least**.

- Cost of a node is defined as:
  - *cost(node) = cumulative cost of all nodes from root*
  - *cost(root) = 0*

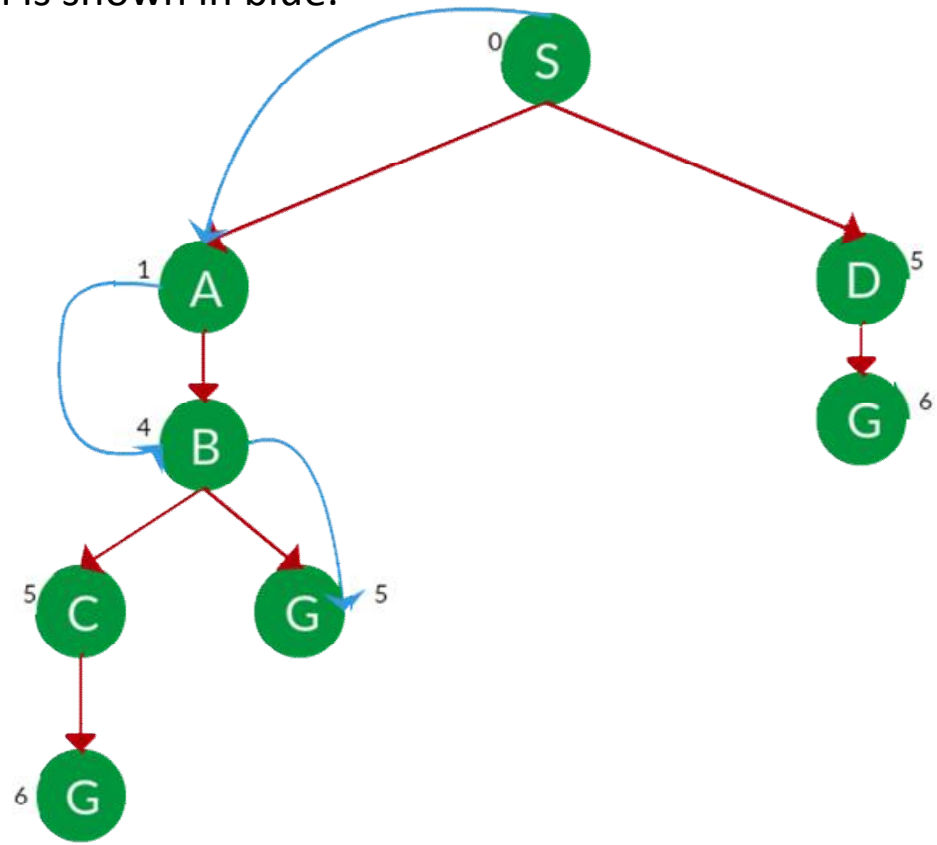
Uniform Cost Search

Problem: Which solution would UCS find to move from node S to node G if run on the graph below?



Path: S -> A -> B -> G
Cost: 5

Solution: The equivalent search tree for the above graph is as follows. **Cost of each node is the cumulative cost of reaching that node from the root**. Based on UCS strategy, **the path with least cumulative cost is chosen**. Note that due to the many options in the fringe, the algorithm explores most of them so long as their cost is low, and discards them when a lower cost path is found; these discarded traversals are not shown below. The actual traversal is shown in blue.



Uniform Cost Search

- Let C = cost of solution.
- ε = arcs cost.
- Then C/ε = effective depth
- **Time complexity:** $T(n) = O(n^{C/\varepsilon})$
- **Space complexity:** $S(n) = O(n^{C/\varepsilon})$
- **Advantages:**
  - UCS is complete.
  - UCS is optimal.
- **Disadvantages:**
  - Explores options in every “direction”.
  - No information on goal location.

Referensi

- https://users.cs.cf.ac.uk/Dave.Marshall/AI2/node22.html#figdep_tree
- https://www.geeksforgeeks.org/search-algorithms-in-ai/
- https://www.javatpoint.com/ai-uninformed-search-algorithms
- https://syifamss.wordpress.com/2017/12/08/metode-pencarian-buta-blind-search-metode-pencarian-heuristik/
- http://alfanfikri27.blogspot.com/2017/12/metode-blind-search-heuristik.html
- https://fitrahadiarief.wordpress.com/2017/12/09/metode-pencarian-buta-blind-search-method-dan-metode-pencarian-heuristik/