

Artificial Intelligence

Instructor: Kietikul Jearanaitanakij

Department of Computer Engineering

King Mongkut's Institute of Technology Ladkrabang

Lecture 3

Uninformed Search

- Uninformed Search Strategies
- Breadth-First Search
- Uniform-Cost Search
- Depth-First Search
- Depth-Limited Search
- Iterative Deepening Search
- Bidirectional Search

Search Strategies

There are two groups of search strategies:

1. **Uninformed Search (Blind Search)** → មិនមានព័ត៌មានបន្ថែម

The strategies have no additional information about states. All they can do is generate successors and distinguish a goal state from a non-goal state. Search strategies are distinguished by the order in which nodes are expanded.

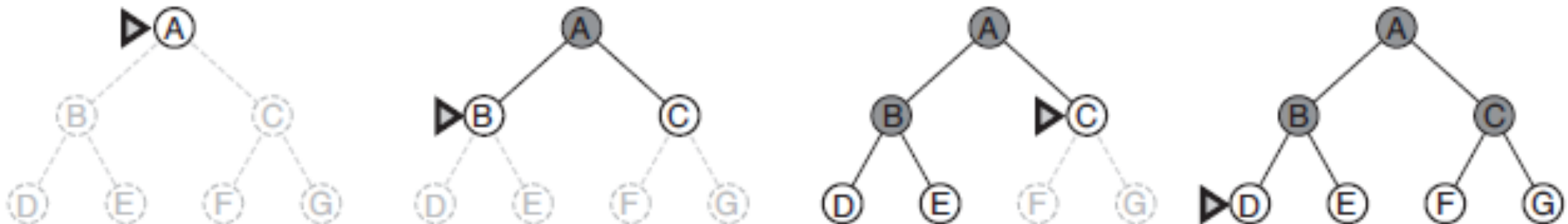
2. **Informed Search (Heuristic Search)** → មានព័ត៌មានបន្ថែម

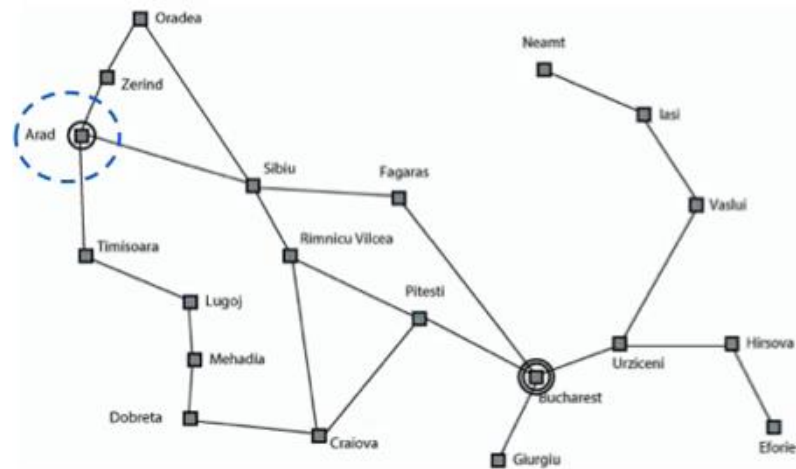
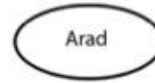
The strategies have useful information for searching. They know whether one non-goal state is “more promising” than another.

Uninformed Search Strategies

1. Breadth-first search

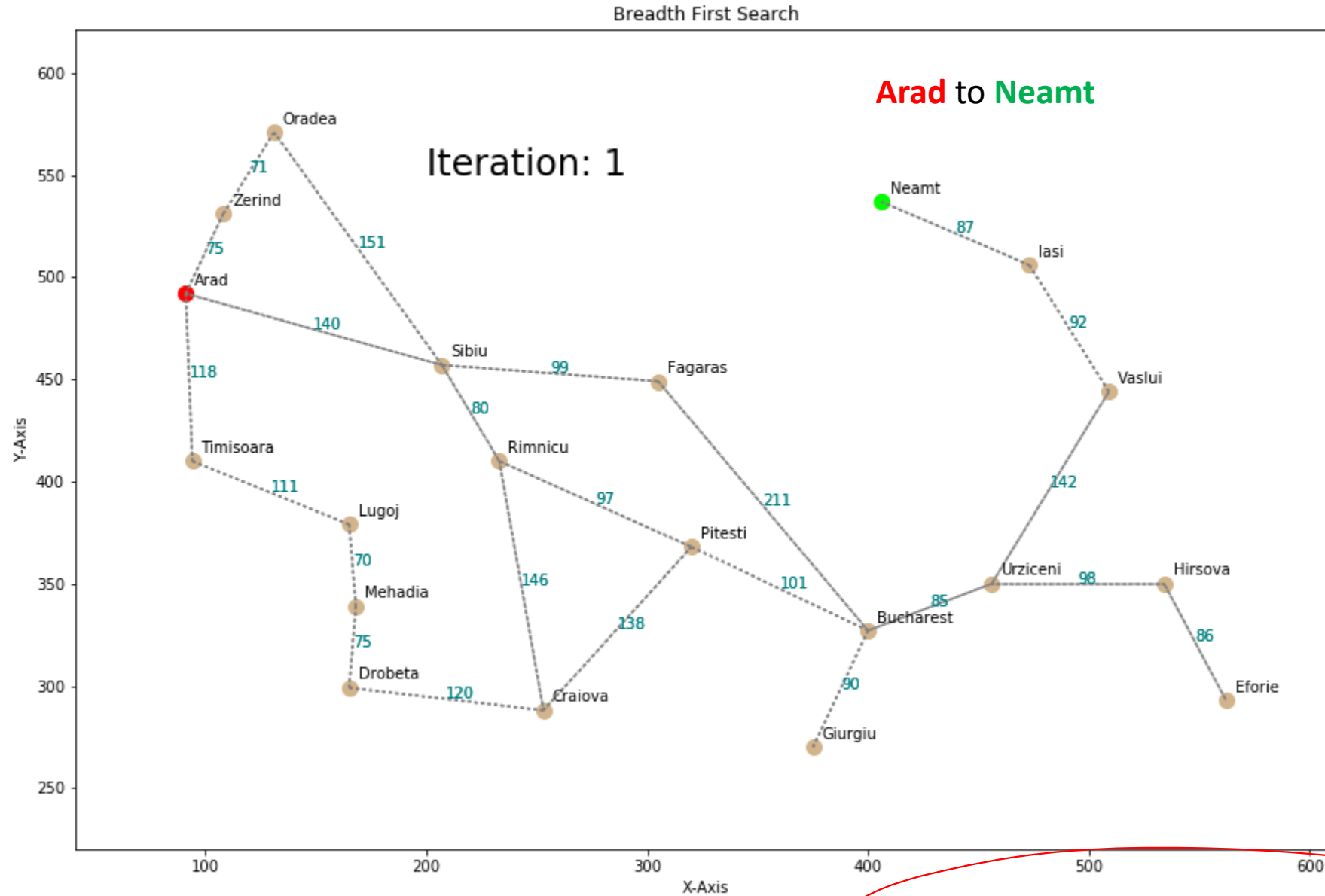
Breadth-first search is a simple strategy in which the root node is expanded first, then all the successors of the root node are expanded next, then their successors, and so on. In general, all the nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded.





Breadth-First Search
©Alan Blair, 2013-17

BFS: <https://youtu.be/aJnDZscuoj8>



Uninformed Search Strategies

function BREADTH-FIRST-SEARCH(*problem*) **returns** a solution, or failure

node \leftarrow a node with STATE = *problem*.INITIAL-STATE, PATH-COST = 0

if *problem*.GOAL-TEST(*node*.STATE) **then return** SOLUTION(*node*)

frontier \leftarrow a FIFO queue with *node* as the only element

explored \leftarrow an empty set

loop do

if EMPTY?(*frontier*) **then return** failure

node \leftarrow POP(*frontier*) /* chooses the shallowest node in *frontier* */

 add *node*.STATE to *explored*

for each *action* **in** *problem*.ACTIONS(*node*.STATE) **do**

child \leftarrow CHILD-NODE(*problem*, *node*, *action*)

if *child*.STATE is not in *explored* or *frontier* **then**

if *problem*.GOAL-TEST(*child*.STATE) **then return** SOLUTION(*child*)

frontier \leftarrow INSERT(*child*, *frontier*)

Uninformed Search Strategies

Analysis of Breadth-First Search

Completeness : Yes

Optimality : Yes (in terms of steps) →

Time complexity : In the worst case, it is the time spends to generate the last node. Then the total number of nodes generated is $b + b^2 + b^3 + \dots + b^d = \underline{O(b^d)}$.

Space complexity : There will be $O(b^{d-1})$ nodes in the explored set and $O(b^d)$ nodes in the frontier. So the space complexity is $O(b^d)$, i.e., it is dominated by the size of the frontier.

b is branching factor, d is depth of the solution

path cost → optimal
cost = cost + cost of node
เลือกที่สั้นที่สุด (เลือก)
ขนาดที่เล็ก = depth เล็ก
เลือกที่สั้นที่สุด (เลือก)

Uninformed Search Strategies

ការប្រើប្រាស់ប្រភេទ bfs → ពិបាកណាស់ណាស់

Time and memory requirements for breadth-first search (scary, isn't it?)

អនុលោម

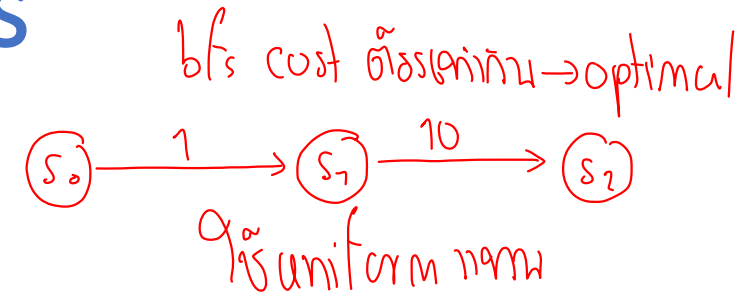
Depth	Nodes	Time	Memory
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	10^6	1.1 seconds	1 gigabyte
8	10^8	2 minutes	103 gigabytes
10	10^{10}	3 hours	10 terabytes
12	10^{12}	13 days	1 petabyte
14	10^{14}	3.5 years	99 petabytes
16	10^{16}	350 years	10 exabytes

ប្រែប្រួល

How does time and space functions grow?

Uninformed Search Strategies

2. Uniform-cost search → คิดถึงต้นทุนก่อนเลือก bfs

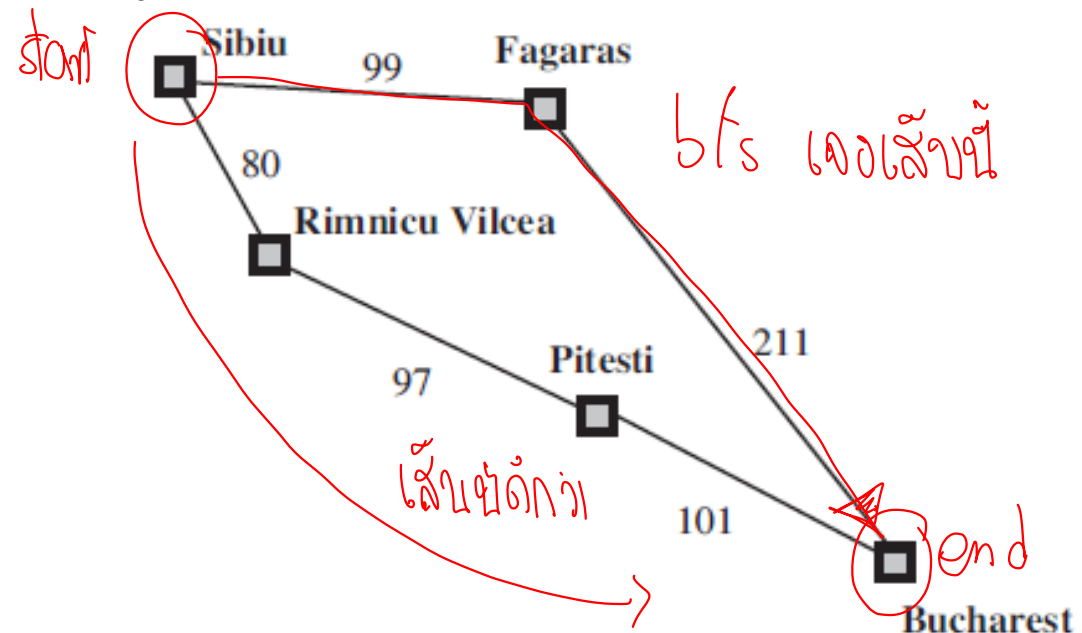


When all step costs are equal, breadth-first search is optimal because it always expands the shallowest unexpanded node. Instead of expanding the shallowest node, uniform-cost search expands the node n with the lowest path cost $g(n)$.

Example:

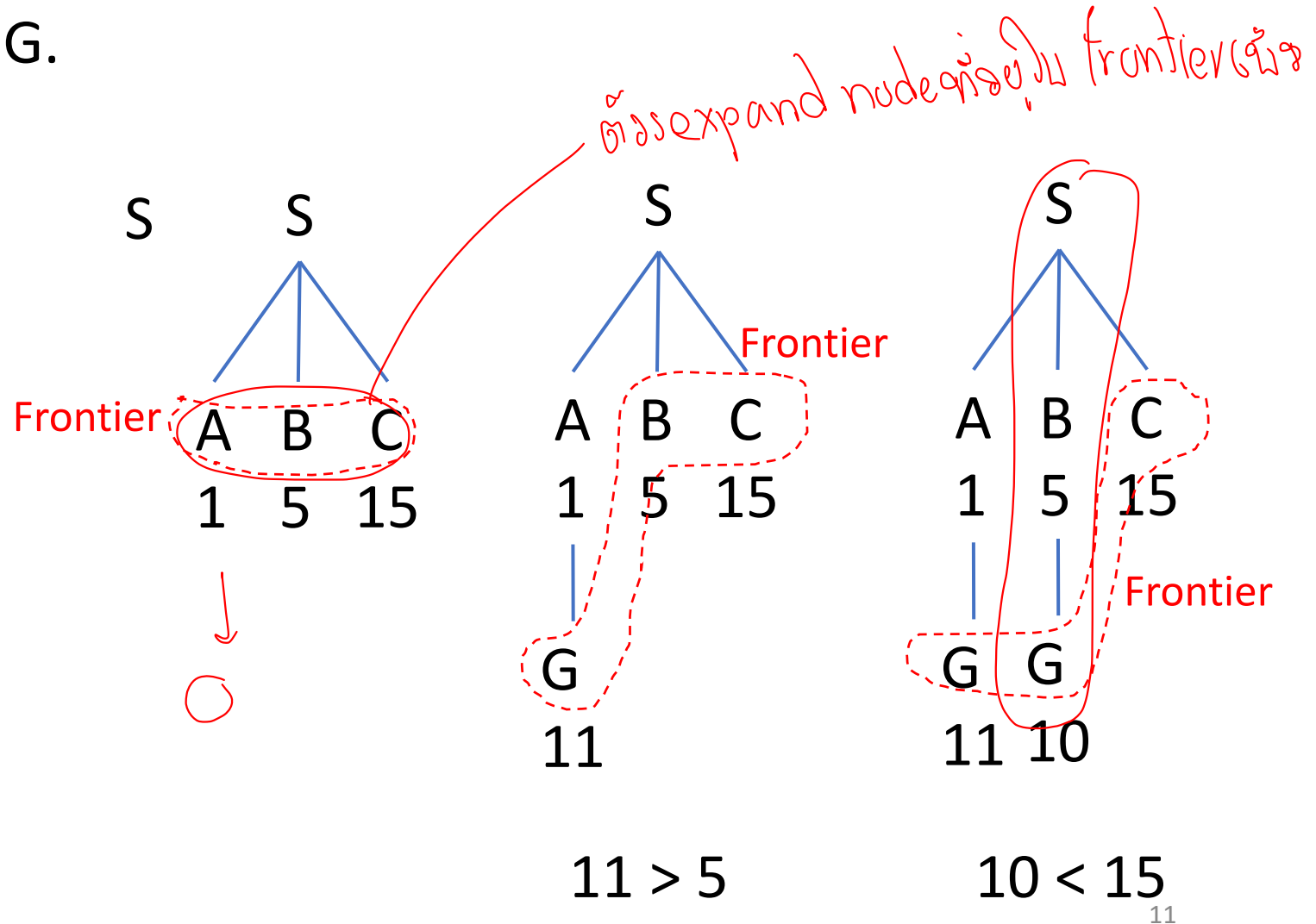
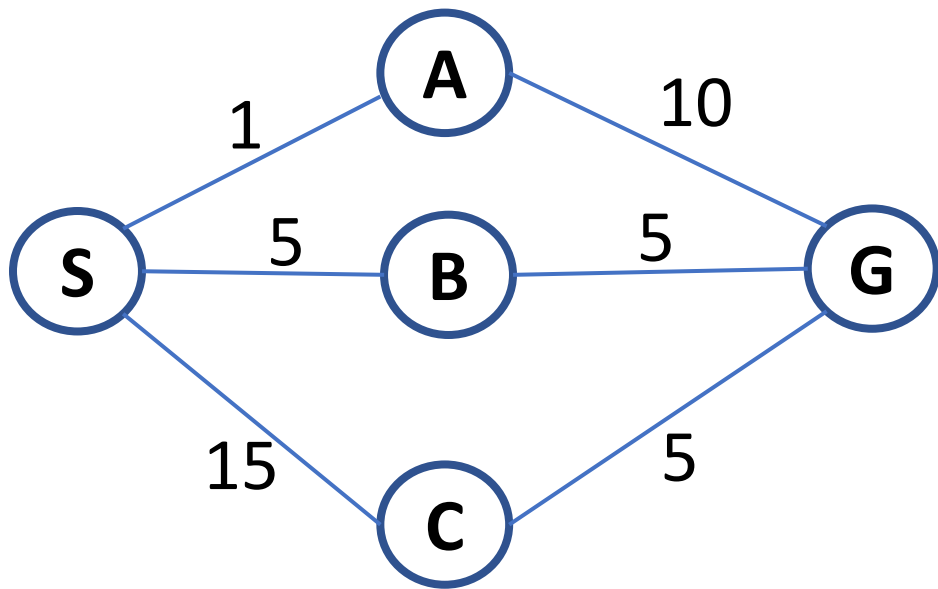
Path_Cost(S-→F-→B) = 310

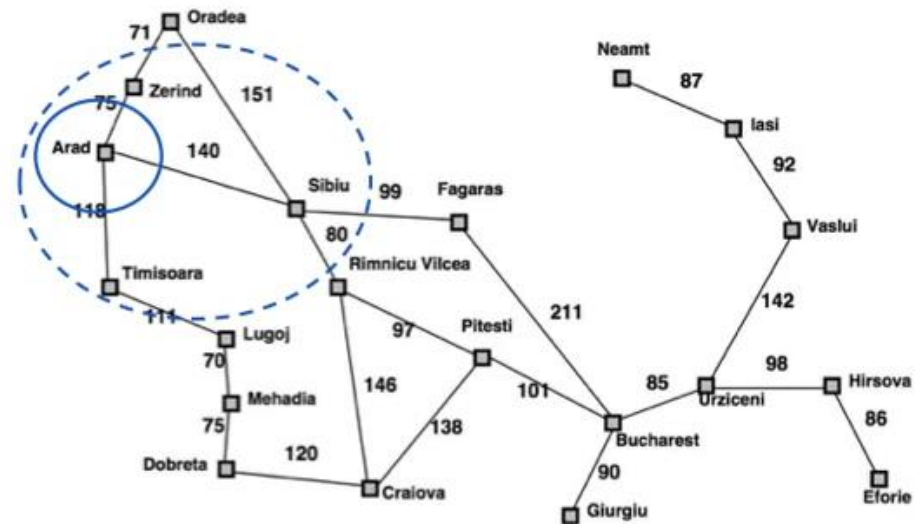
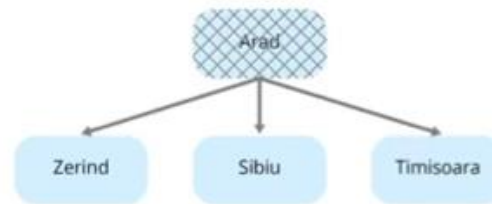
Path_Cost(S-→R-→P-→B) = 278



Uninformed Search Strategies

Example: Start from S to goal G.





Uniform-Cost Search
©Alan Blair, 2013-17

Uniform-cost search: <https://youtu.be/-FY7t2kqWX4>

Uninformed Search Strategies

function UNIFORM-COST-SEARCH(*problem*) **returns** a solution, or failure

node \leftarrow a node with STATE = *problem*.INITIAL-STATE, PATH-COST = 0

frontier \leftarrow a priority queue ordered by PATH-COST, with *node* as the only element

explored \leftarrow an empty set

loop do

if EMPTY?(*frontier*) **then return** failure

node \leftarrow POP(*frontier*) /* chooses the lowest-cost node in *frontier* */

if *problem*.GOAL-TEST(*node*.STATE) **then return** SOLUTION(*node*)

 add *node*.STATE to *explored*

for each *action* **in** *problem*.ACTIONS(*node*.STATE) **do**

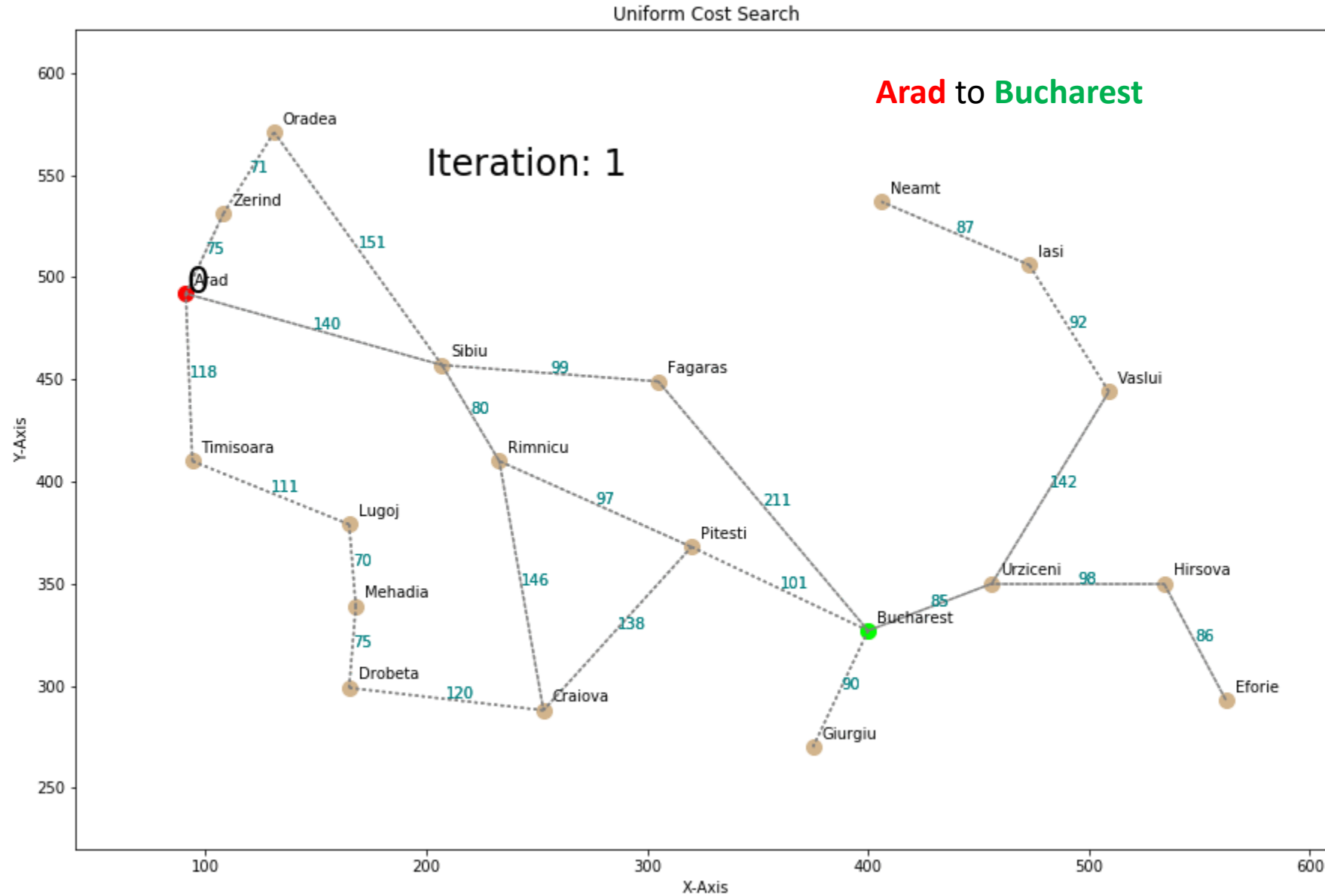
child \leftarrow CHILD-NODE(*problem*, *node*, *action*)

if *child*.STATE is not in *explored* or *frontier* **then**

frontier \leftarrow INSERT(*child*, *frontier*)

else if *child*.STATE is in *frontier* with higher PATH-COST **then**

 replace that *frontier* node with *child*



Uninformed Search Strategies

Analysis of Uniform-cost Search

Completeness : Yes → ถ้ามีขอบเขตของค่าต้นทุน, inherit from bfs ธรรมดา

Optimality : Yes → เลือก cost น้อยสุด

Time complexity : $O(b^{1+\lceil C^*/\epsilon \rceil})$

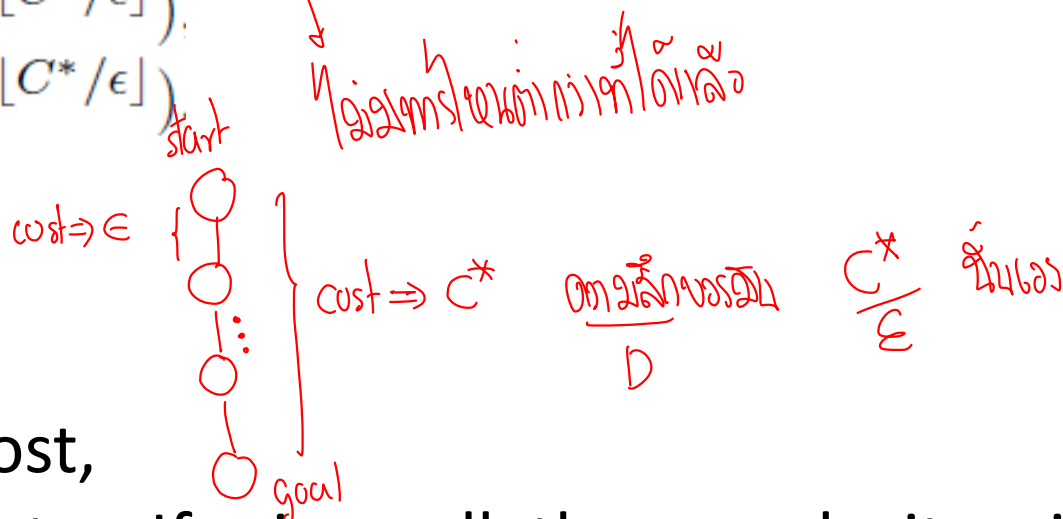
Space complexity : $O(b^{1+\lceil C^*/\epsilon \rceil})$

Where:

b is branching factor,

C^* is the optimal path cost,

Every action costs at least ϵ . If ϵ is small, the complexity will be high.

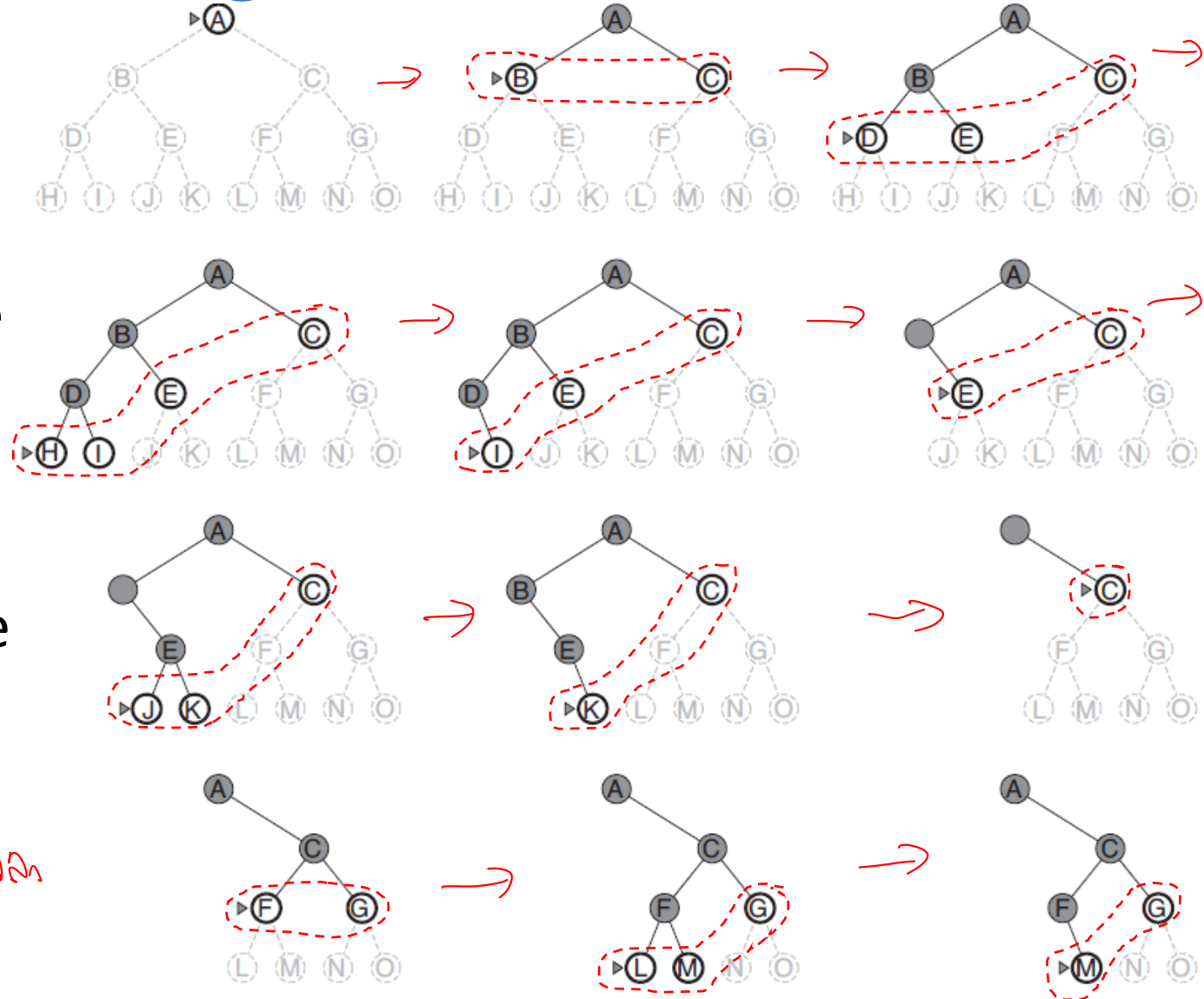


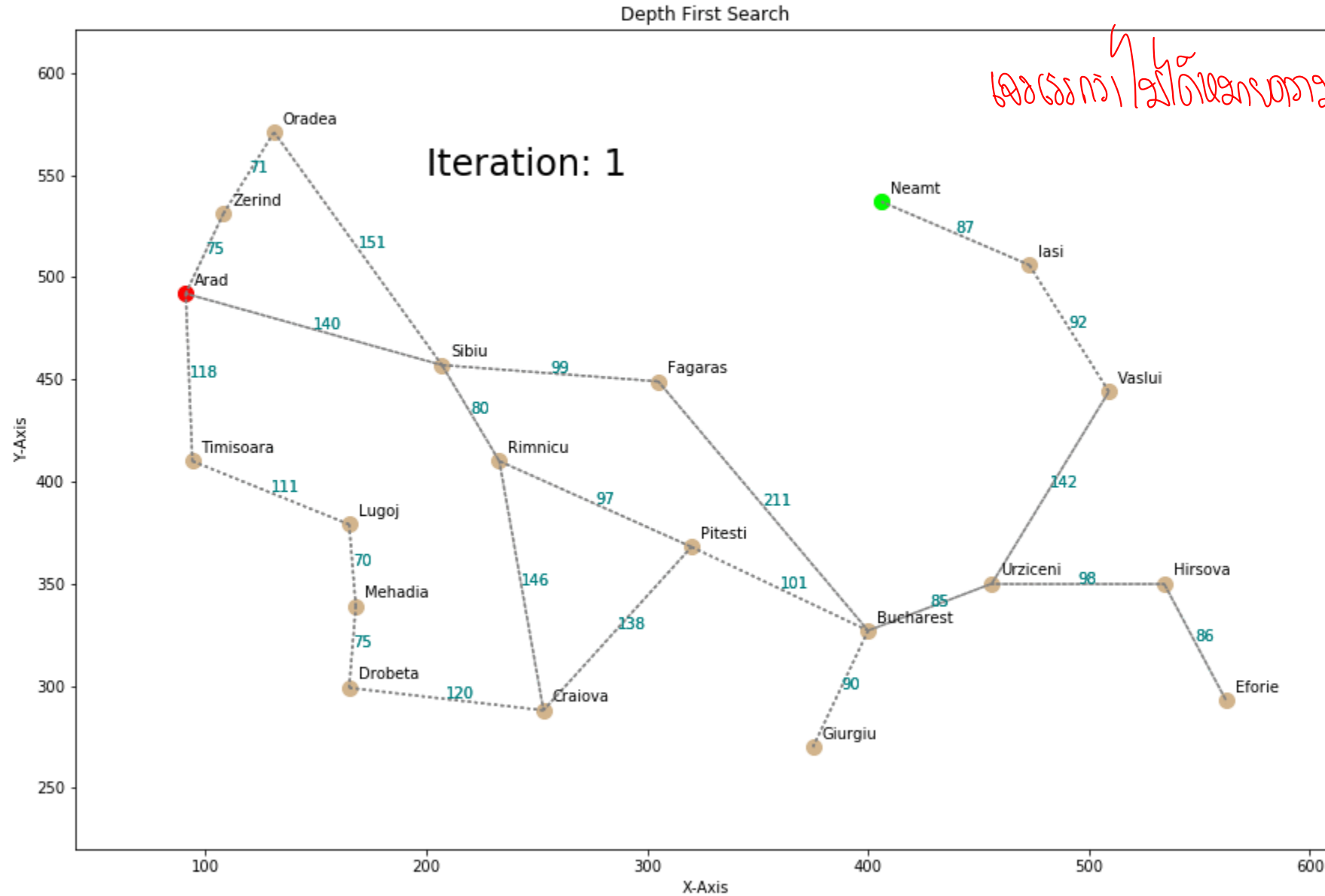
Uninformed Search Strategies

3. Depth-First Search *မှတ်တမ်းချက်*

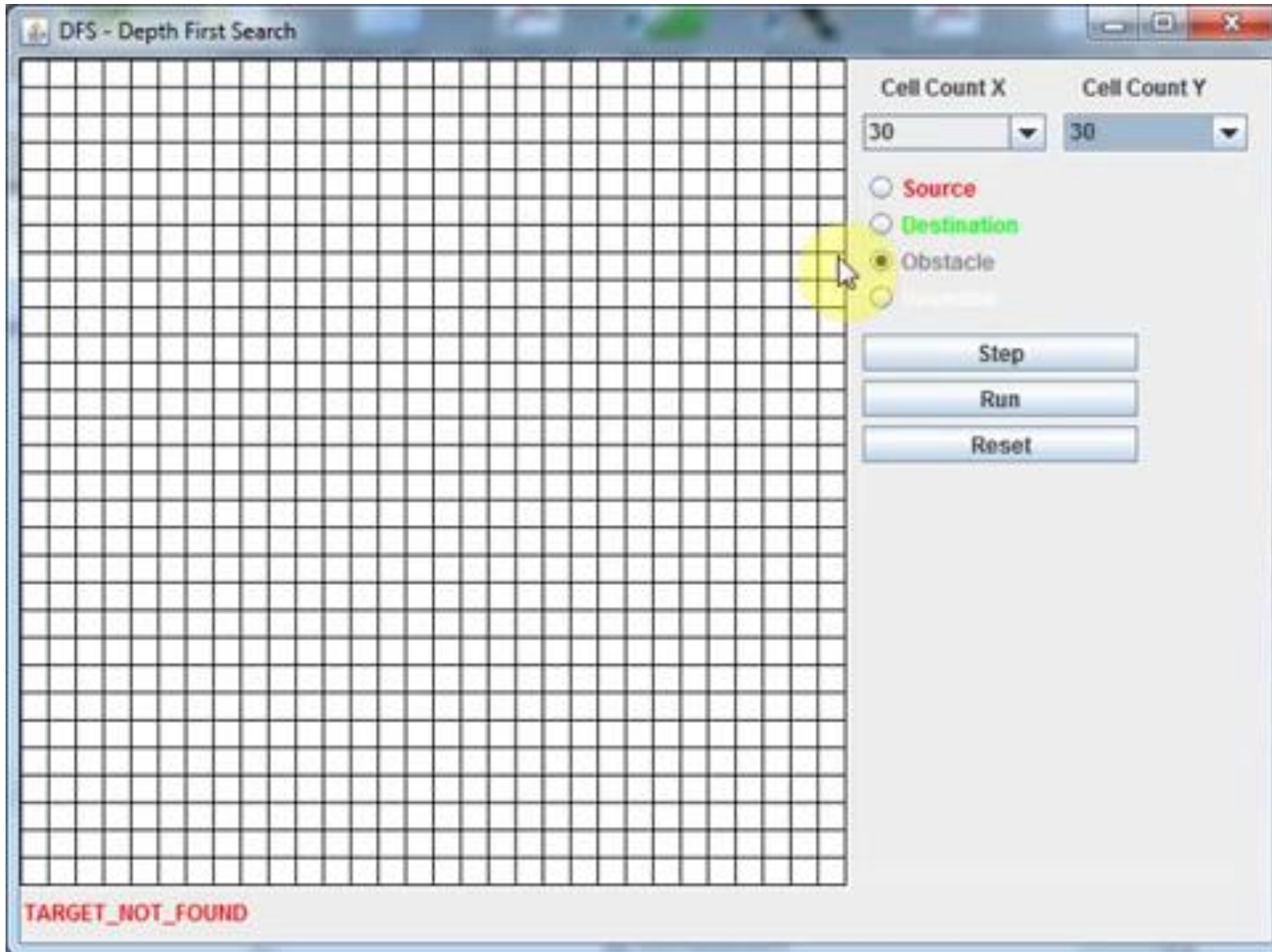
DFS always expands the deepest node in the current frontier of the search tree.

While Breadth-first-search uses a FIFO queue, depth-first search use a LIFO queue (stack).





ลองแรกๆ / จะได้แนวทางว่าทำอย่างไร



అవినీతి

Uninformed Search Strategies

Analysis of Depth-First Search

Completeness : No (if redundant states are allowed)

Optimality : No → 4 ข้อได้ผล แต่ 9 ข้อที่เสีย

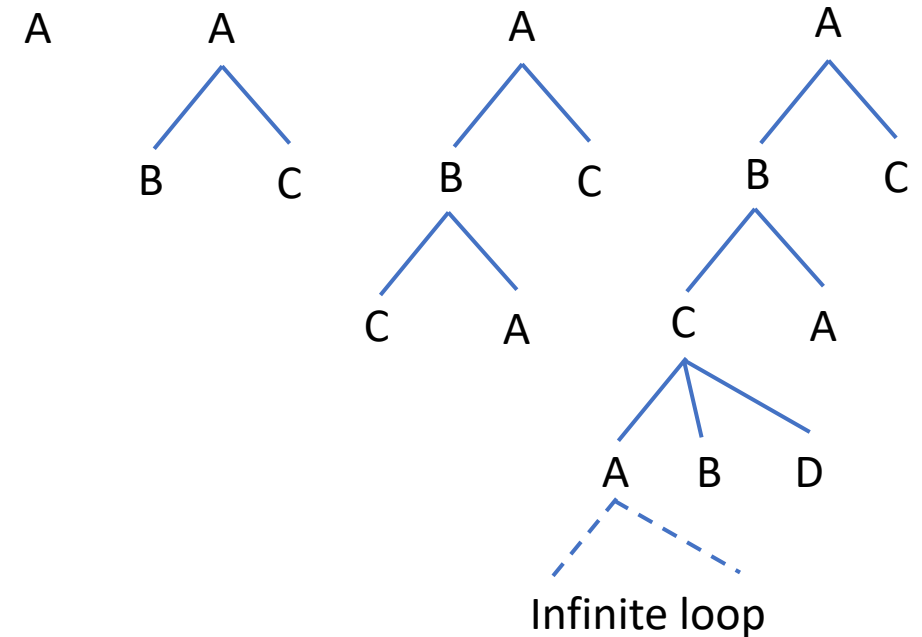
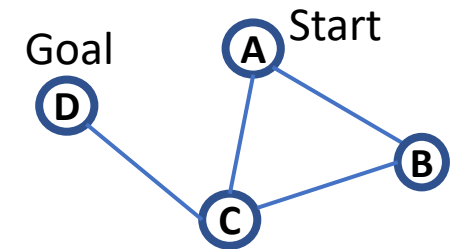
Time complexity : $O(b^d)$

Space complexity : $O(bd)$ => **Good**

Where:

b is branching factor,

d is the max. depth of solution.

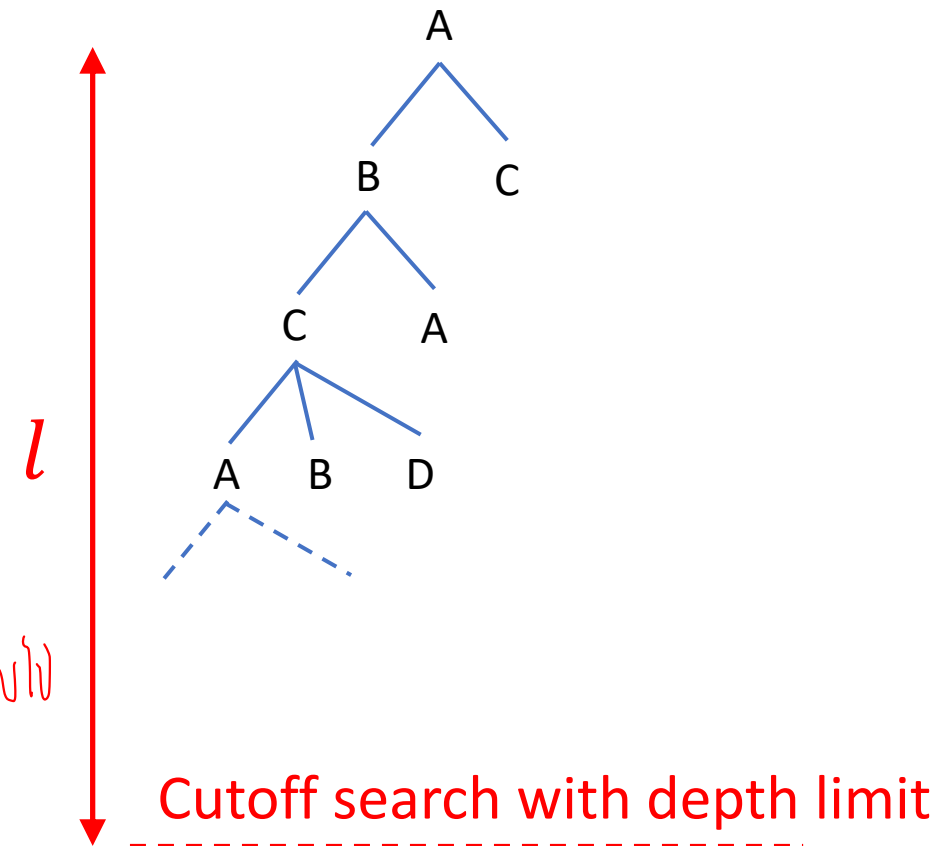


4 ข้อได้ผล แต่ 9 ข้อที่เสีย

Uninformed Search Strategies

4. Depth-Limited Search *มสั้งล้น dfs*

Incompleteness of DFS can be alleviated by a predetermined depth limit. The depth limit solves the infinite-path problem. DLS usually uses when we know the maximum solution depth of the problem.



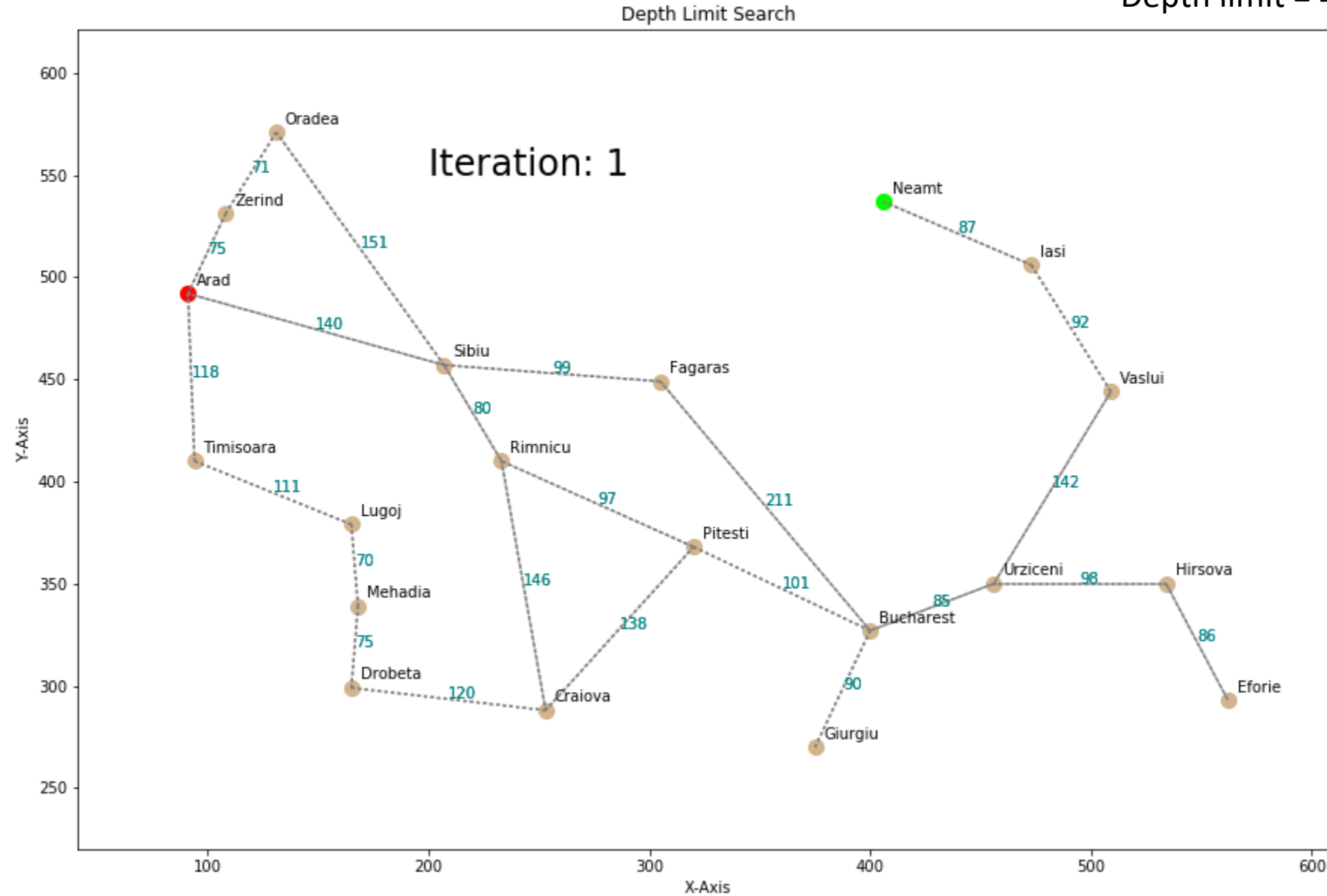
ล้น cut off → backtrack กลับไป



ถ้าไม่เจอคำตอบก็ตัดทิ้ง

→ กลับไปหาลำดับอื่น

Depth limit = 4



Uninformed Search Strategies

Analysis of Depth-limited Search

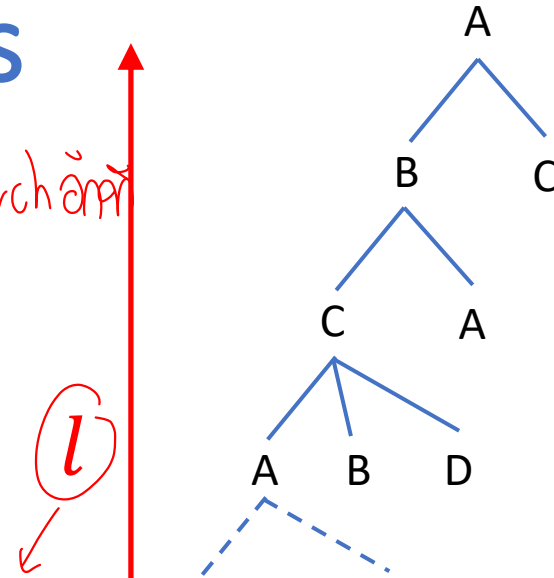
Completeness : Yes (if $l \geq \text{maximum depth of the solution}$)
Optimality : No
Time complexity : $O(b^l)$
Space complexity : $O(bl)$

60202182

Uninformed Search Strategies

5. Iterative Deepening Search (Iterative deepening depth-first)

IDS iterates through the depth-limited search by gradually increasing the limit—first 0, then 1, then 2, and so on—until a goal is found.



vary
↓
ค่าของ α

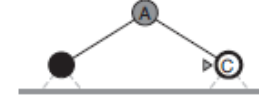
Cutoff search with depth limit

IDS combines the benefits of DFS and BFS. Like DFS, its memory requirements are modest: $O(bd)$. Like BFS, it is complete when the branching factor is finite and optimal when the path cost is a nondecreasing function of the depth of the node.

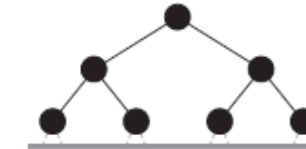
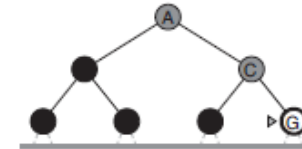
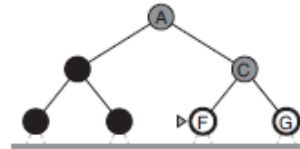
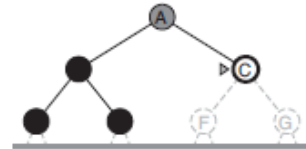
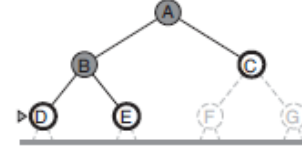
Limit = 0



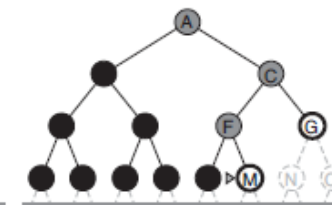
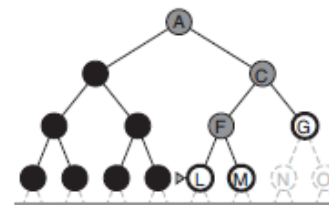
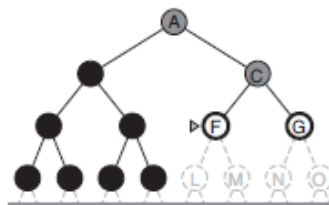
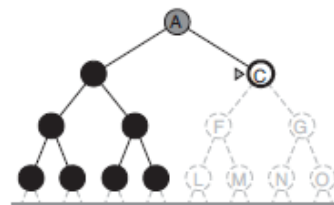
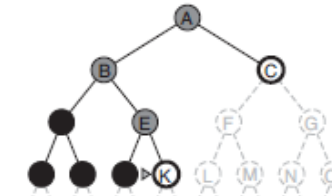
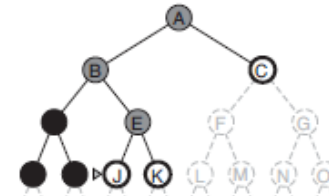
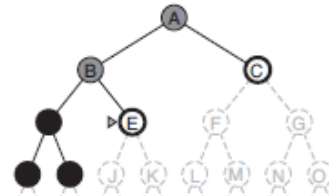
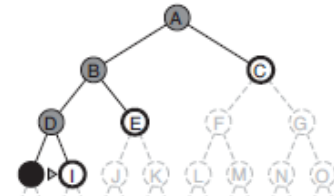
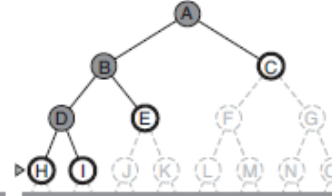
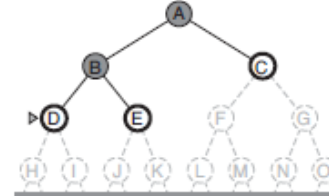
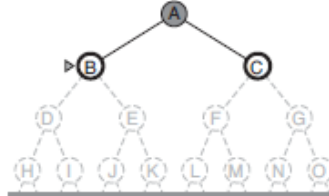
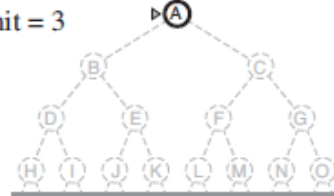
Limit = 1



Limit = 2



Limit = 3



กราฟเชิงภาพ

bfs

↓
ทำในใจก่อน
แล้วค่อย

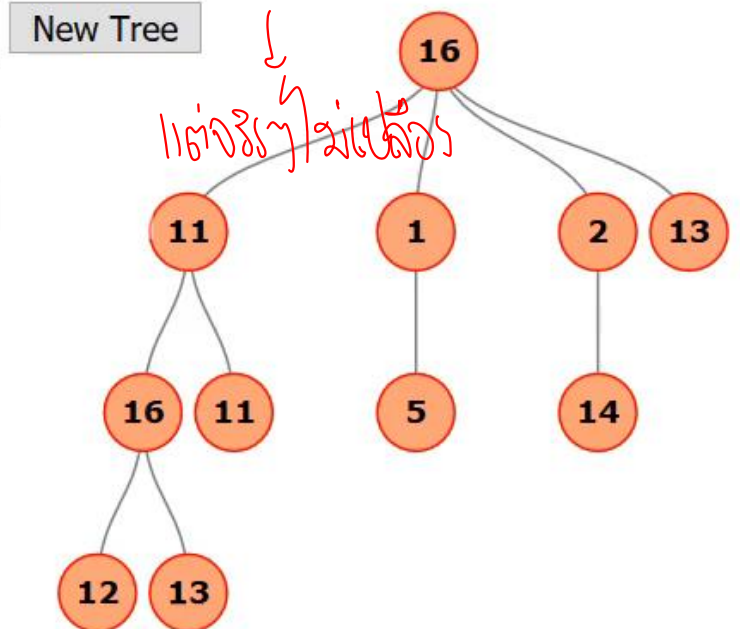
Algorithm Wiki

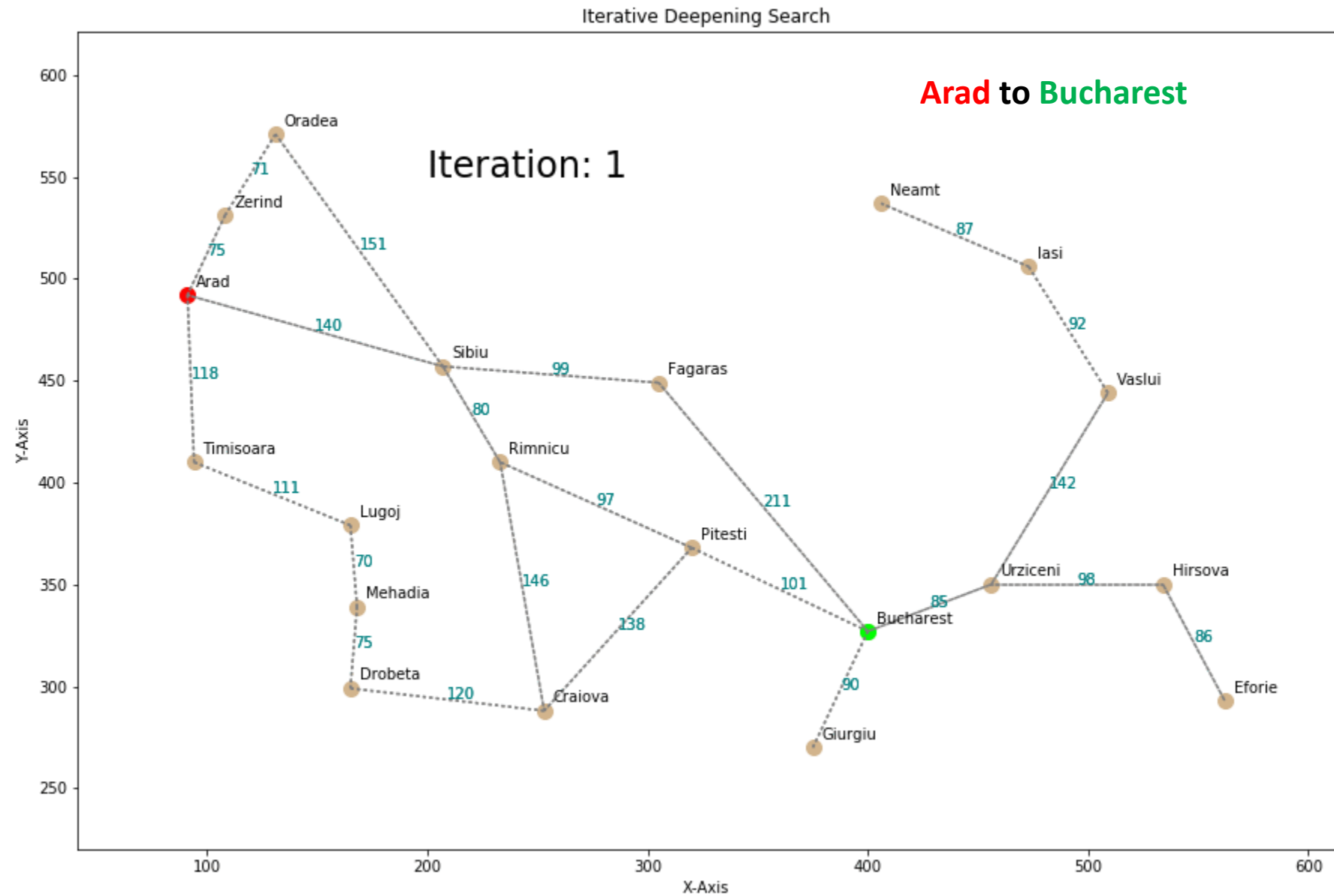
Recent Changes Media Ma

Trace: • Iterative deepening depth-first search • **Iterative deepening depth-first search**


```
1 function iterativeDeepeningDepthFirstSearch(node) {
2   // Repeatedly depth-first search up-to a maximum depth of 6.
3   for (var maxDepth = 1; maxDepth < 6; maxDepth++) {
4     depthFirstSearch(node, maxDepth);
5   }
6 }
7
8 function depthFirstSearch(node, maxDepth) {
9   // If reached the maximum depth, stop recursing.
10  if (maxDepth <= 0) {
11    return;
12  }
13
14  // Recurse for all children of node.
15  for (var i=0, c=node.children.length; i<c; i++) {
16    depthFirstSearch(node.children[i], maxDepth-1);
17  }
18 }
```

ดูแบบนี้อาจจะเปลี่ยนนะ





Uninformed Search Strategies

- Iterative deepening search may seem wasteful because states are generated multiple times.


↓ វាមិនមែនជាការបាត់បង់ទេ? ចំណុចសំខាន់ៗ
- It turns out this is not too costly.
- The reason is that in a search tree with the same (or nearly the same) branching factor at each level, most of the nodes are in the bottom level, so it does not matter much that the upper levels are generated multiple times.

Uninformed Search Strategies

In an iterative deepening search, the nodes on the bottom level (depth d) are generated once, those on the next-to-bottom level are generated twice, and so on, up to the children of the root, which are generated d times. So **the total number of nodes generated** in the worst case is $\frac{d+1}{2} \times \text{nodes at depth } d$.

$$N(\text{IDS}) = (d)b + (d-1)b^2 + \dots + (1)b^d,$$

อธิบดีไปทำเรื่อง

↓
ทำง.ไปจวรอบต้งว.ล

which give a time complexity $O(b^d)$.

However, there is some extra cost for generating the upper levels multiple times, but it is not large. For example, if $b = 10$ and $d = 5$, the numbers are

$$N(\text{IDS}) = 50 + 400 + 3,000 + 20,000 + 100,000 = 123,450$$

$$N(\text{BFS}) = 10 + 100 + 1,000 + 10,000 + 100,000 = 111,110.$$

• 1. 11. 11. 11

IDS is preferred when the search space is large and the solution depth is unknown.

Uninformed Search Strategies

Analysis of Iterative Deepening Search

Completeness : Yes → *ခွဲစိတ်*
Optimality : Yes → *ခွဲစိတ်*
Time complexity : $O(b^d)$ → *ခွဲစိတ်*
Space complexity : $O(bd)$ → *ခွဲစိတ်*

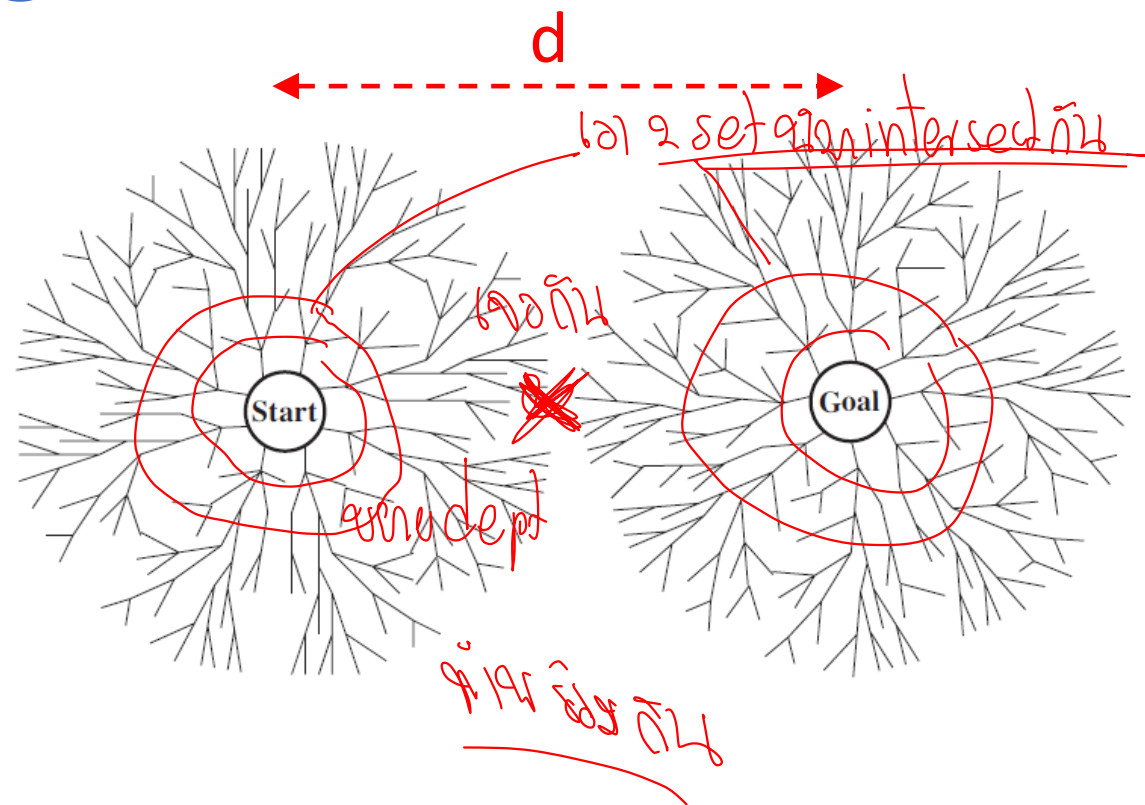
วันพฤหัสบดีที่ ๑๕ ธันวาคม ๒๕๖๓

5. Bidirectional Search \rightarrow $\hat{S}_{now}^1, \hat{S}_{now}^2, S_{now}, g$

Idea : Run two simultaneous searches—one forward from the initial state and the other backward from the goal—hoping that the two searches meet in the middle.

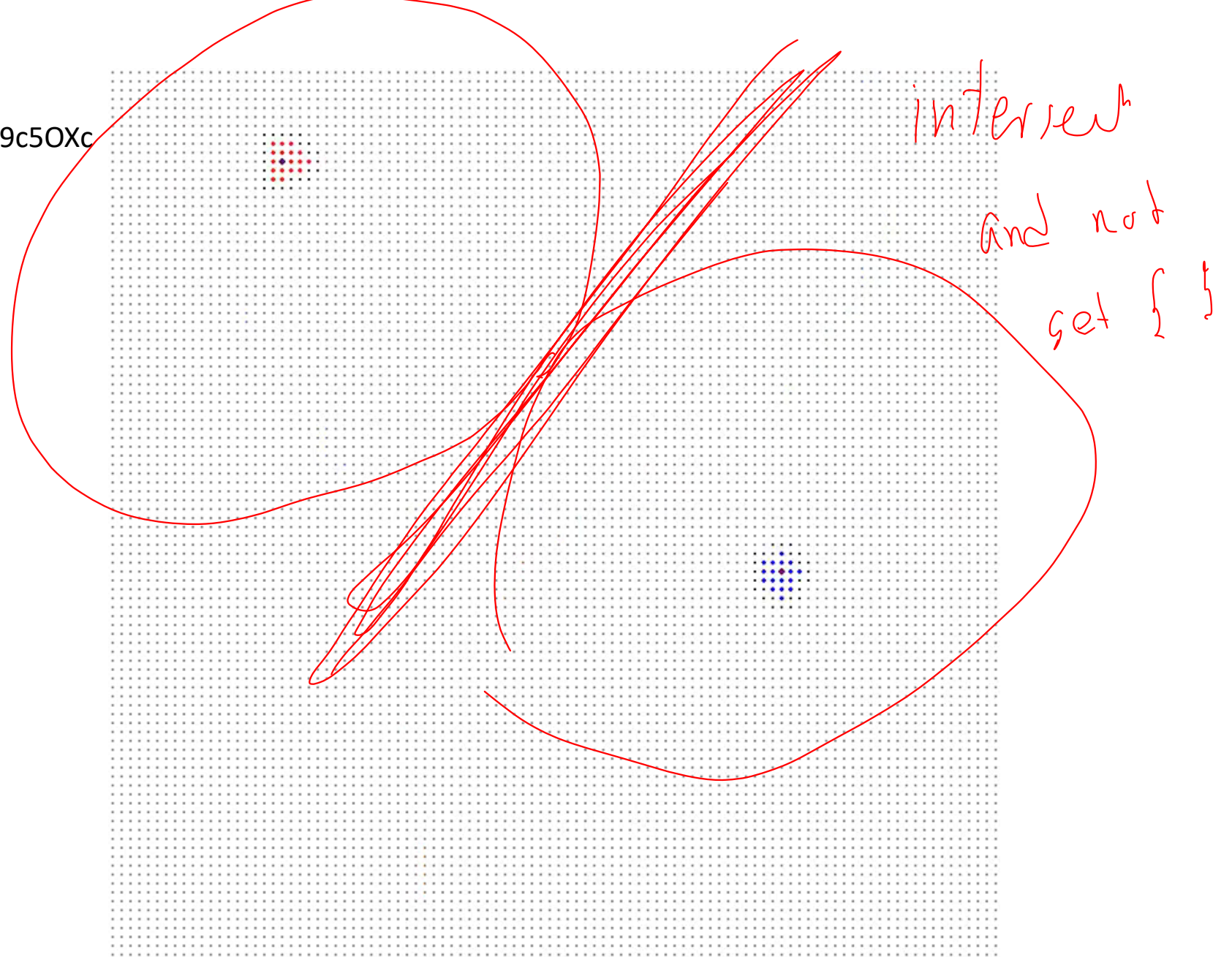
The motivation is that $b^{d/2} + b^{d/2}$ is much less than b^d .

For each search on both sides, we can use either breadth-first search or iterative deepening search to reduce the space complexity.



Bidirectional Search Simulation

<https://www.youtube.com/watch?v=YtPT99c5OXc>



Uninformed Search Strategies

Analysis of Iterative Deepening Search

Completeness : Yes

Optimality : Yes

(if the cost for each action is equal & monotonically increasing)

Time complexity : $O(b^{d/2})$

$$b^{d/2} < b^d$$

Space complexity : $O(b^{d/2})$ BFS, $O(bd/2)$ IDS

$$O(bd) \approx \frac{O(bd)}{2} \rightarrow \text{constant}$$

Uninformed Search Strategies

Comparison of uninformed search strategies

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes ^a	Yes ^{a,b}	No	No	Yes ^a	Yes ^{a,d}
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes ^c	Yes	No	No	Yes ^c	Yes ^{c,d}

Figure 3.21 Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; ℓ is the depth limit. Superscript caveats are as follows: ^a complete if b is finite; ^b complete if step costs $\geq \epsilon$ for positive ϵ ; ^c optimal if step costs are all identical; ^d if both directions use breadth-first search.