AI 2002
Artificial Intelligence

Dr. Hashim Yasin

Uniform-Cost Search

Uniform-Cost Search

- The **uniform-cost search** expands the node n with the *lowest path cost* $g(n)$.

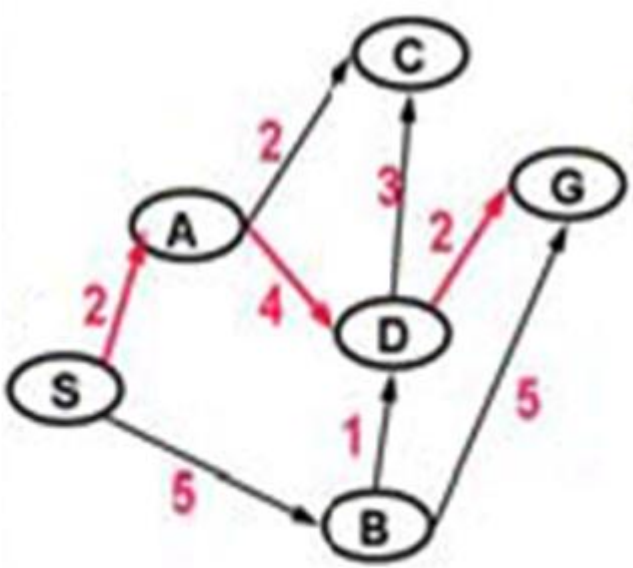
Breadth First Search vs Uniform Cost Search:

- Uniform cost search has two significant differences from breadth-first search.
  1. The *goal test is applied to a node when it is* ***selected for expansion*** rather than when it is first generated.
  2. The ***goal test is added in case if a better path is found*** *to a node currently on the frontier*

Uniform-Cost Search

- ❑ Pick best (by **path length**) element of Q
- ❑ Add path extensions to Q

	Q
1	(0 S)
2	**(2 A S) (5 B S)**
3	**(4 C A S) (6 D A S)** (5 B S)
4	(6 D A S) (5 B S)

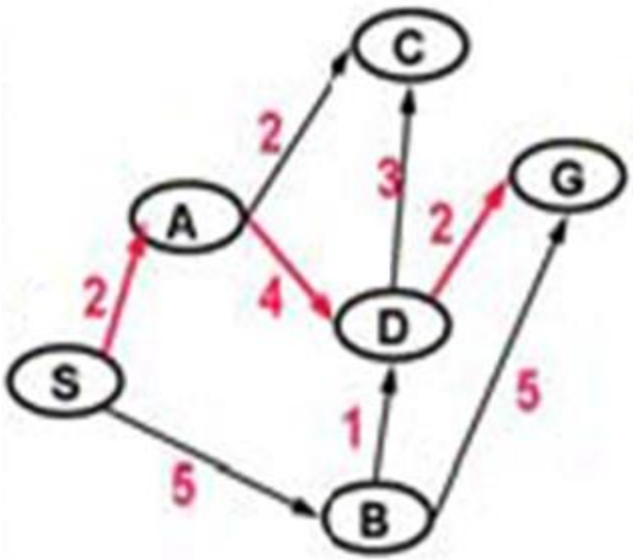


- ❑ **Blue Color represents added paths**
- ❑ Underline paths are chosen for extension.

Uniform-Cost Search

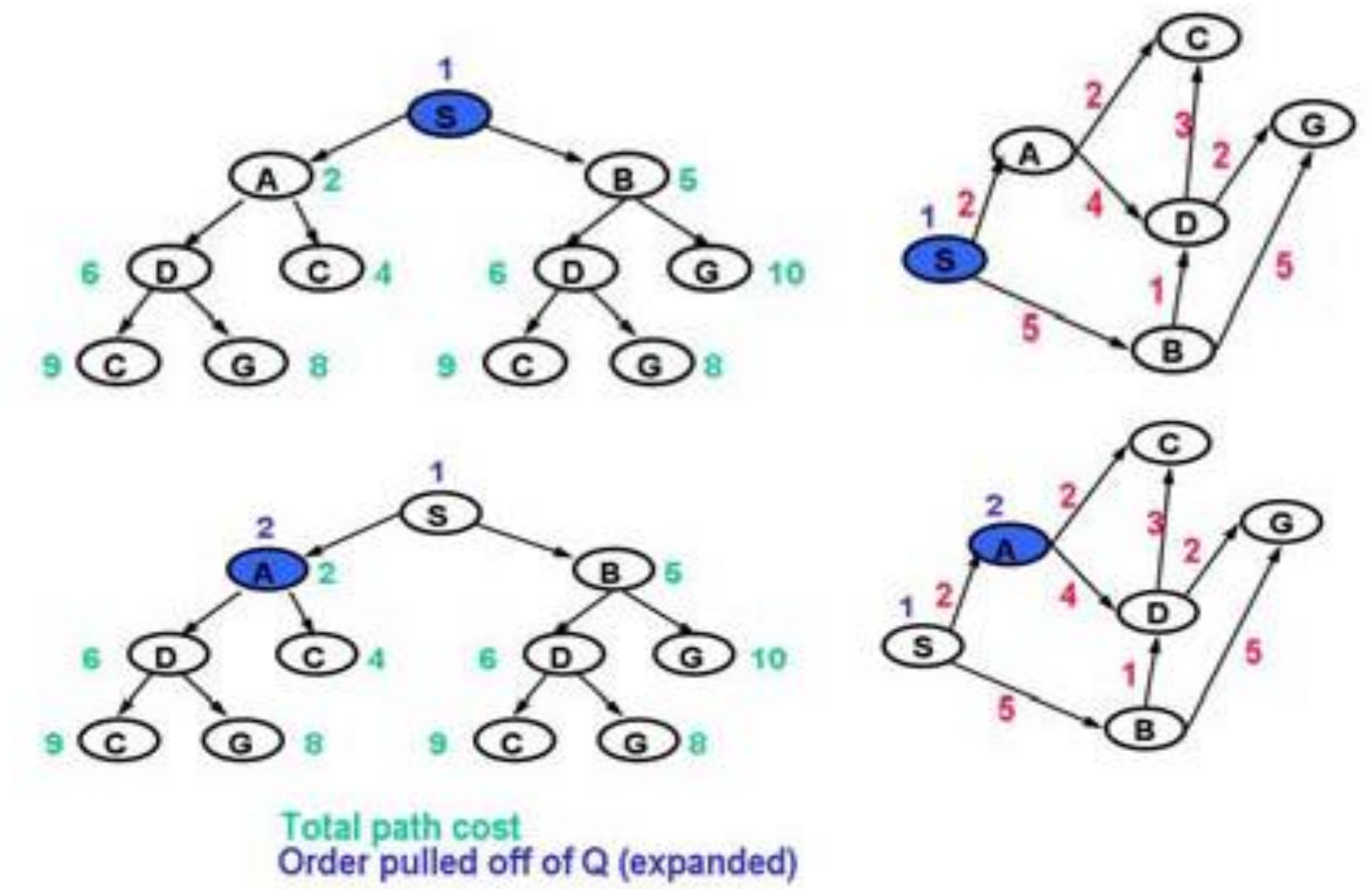
- Pick best (by **path length**) element of Q
- Add path extensions to Q

	Q
1	(0 S)
2	**(2 A S) (5 B S)**
3	**(4 C A S) (6 D A S)** (5 B S)
4	(6 D A S) (5 B S)
5	**(6 D B S) (10 G B S)** (6 D A S)
6	**(8 G D B S) (9 C D B S)** (10 G B S) (6 D A S)
7	**(8 G D A S) (9 C D A S)** (8 G D B S) (9 C D B S) (10 G B S)

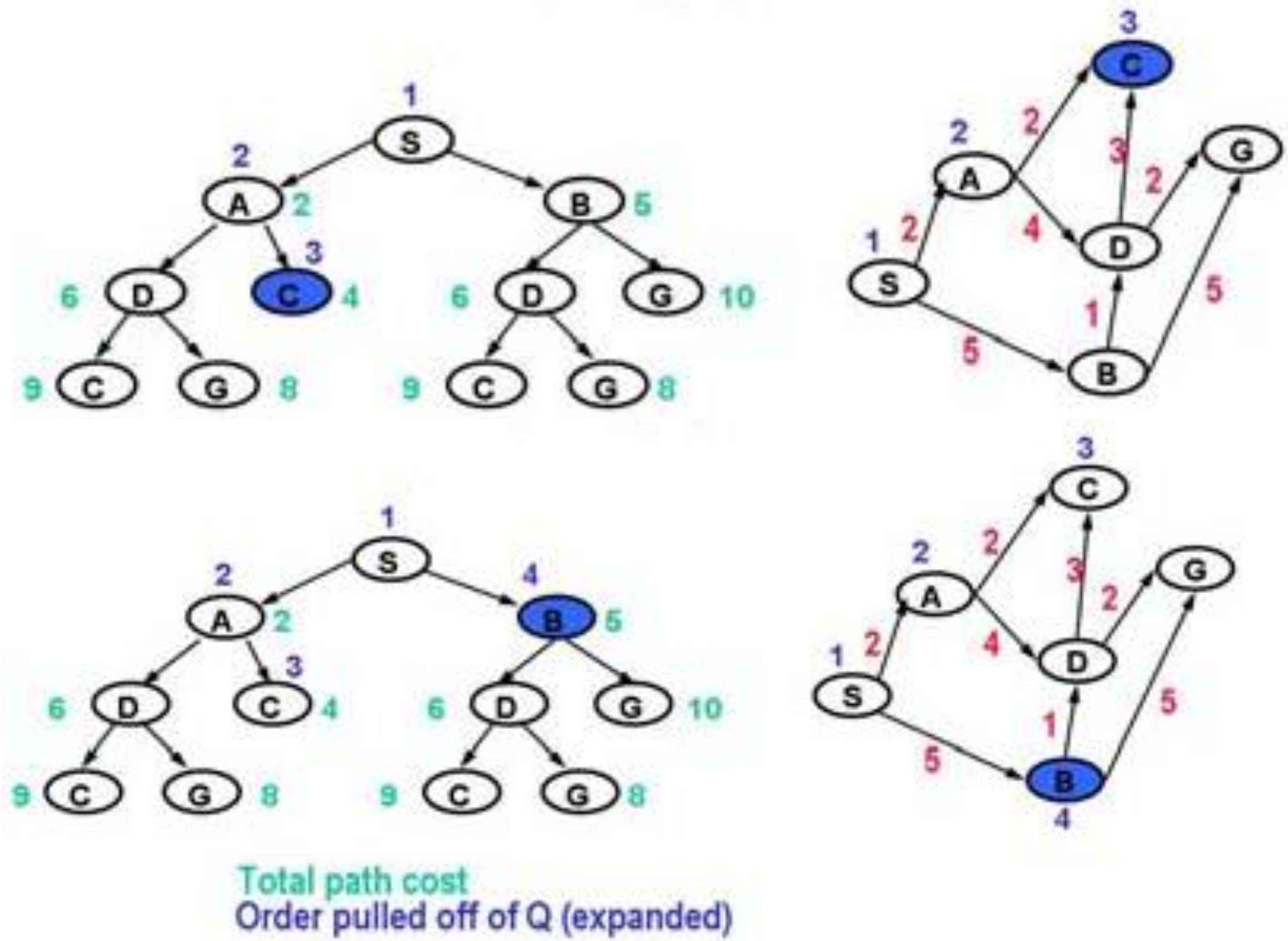


- **Blue Color represents added paths**
- Underline paths are chosen for extension.

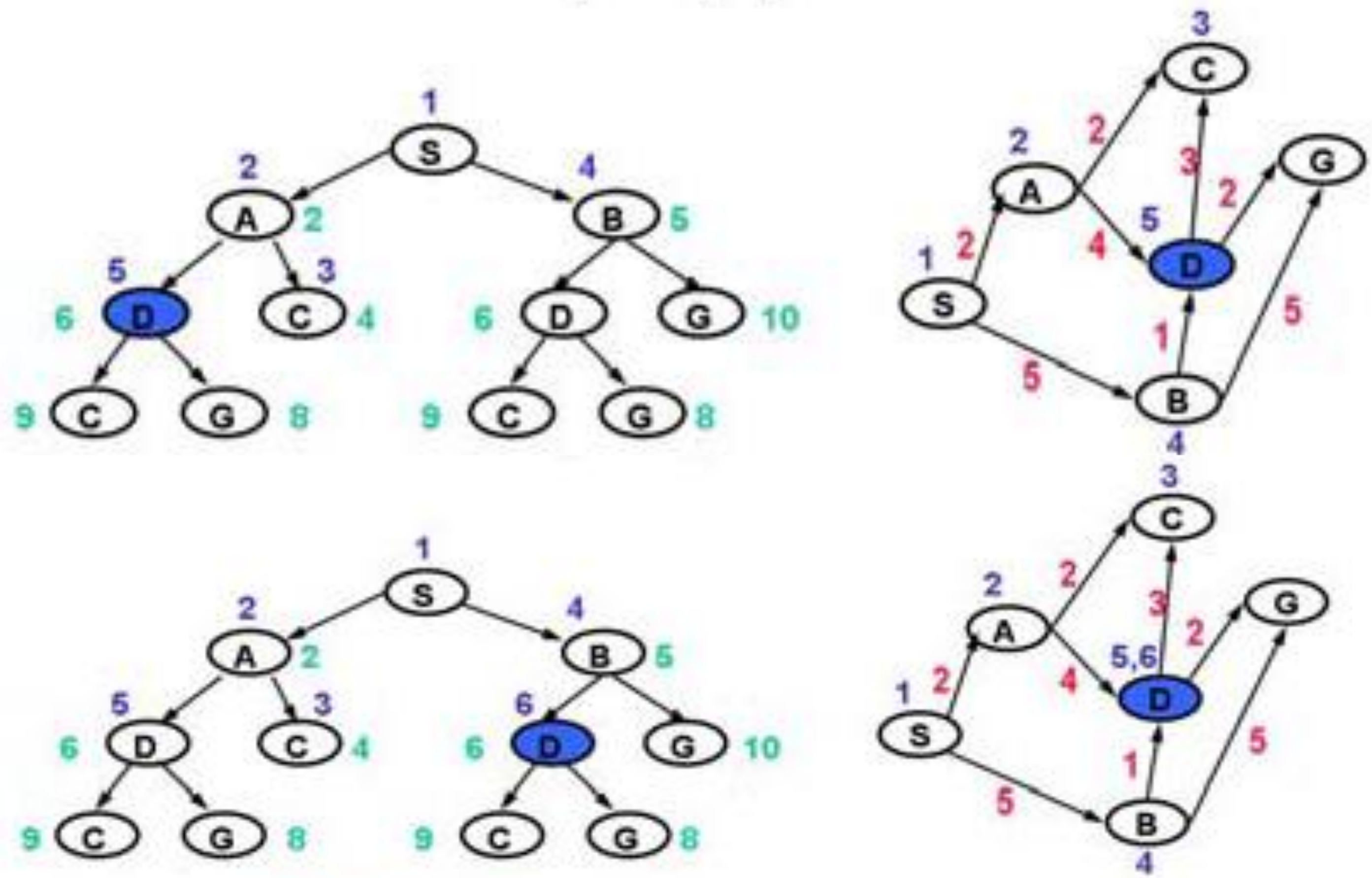
Uniform-Cost Search



Uniform-Cost Search

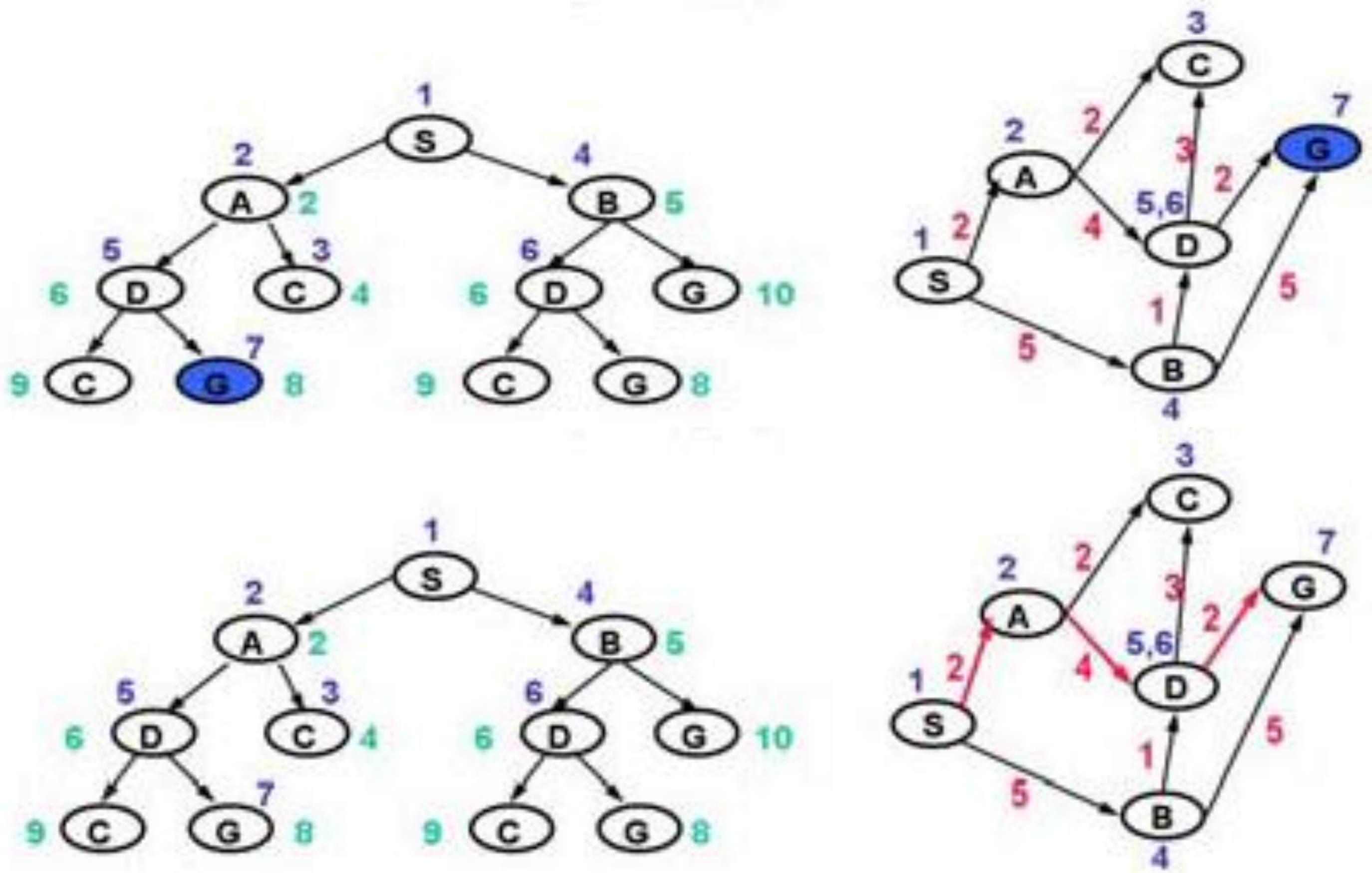


Uniform-Cost Search



Total path cost
Order pulled off of Q (expanded)

Uniform-Cost Search



Total path cost
Order pulled off of Q (expanded)

Uniform-Cost Search

- Almost Equivalent to breadth-first
  - if step costs are all equal.
  - except that the Breadth-first search stops as soon as it generates a goal, whereas uniform-cost search examines all the nodes at the goal's depth to see if one has a lower cost;
  - Thus uniform-cost search **does strictly more work** by expanding nodes at depth d unnecessarily.
- It is implemented by the **queue ordered by path cost** rather than depths,
- Its complexity cannot easily be characterized in terms of $\boldsymbol{b}$ and $\boldsymbol{d}$.

Uniform-Cost Search

Completeness

- Uniform-Cost Search is considered **complete**
  - when the cost of every step is greater than or equal to some small positive constant "ε".
- It will get **stuck in an infinite loop** if there is a path with an infinite sequence of zero-cost actions

Optimality

- This condition is also sufficient to ensure ***optimality.***
  - *It means that the cost of a path always increases as we go along the path*.

Uniform-Cost Search

- *Let C^* be the cost of the optimal solution,* and assume that *every action costs at least* $\boldsymbol{\varepsilon}$**.** Then the algorithm's **worst-case time and space** is

$$O\left(b^{1+\lfloor C^*/\varepsilon \rfloor}\right)$$

Which can be much greater than b^d

- When all step costs are equal, of course,

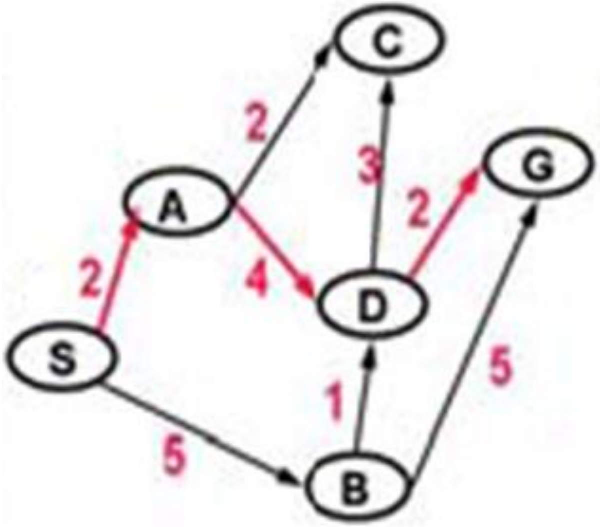
$$O\left(b^{1+\lfloor C^*/\varepsilon \rfloor}\right) = O\left(b^{d+1}\right)$$

Uniform-Cost Search

(with strict Expanded List)

- ❑ Pick best (by **path length**) element of Q
- ❑ Add path extensions to Q

	Q	Expanded
1	<u>(0 S)</u>	S
2	**<u>(2 A S)</u> (5 B S)**	S, A
3	**<u>(4 C A S)</u> (6 D A S)** (5 B S)	S, A, **C**



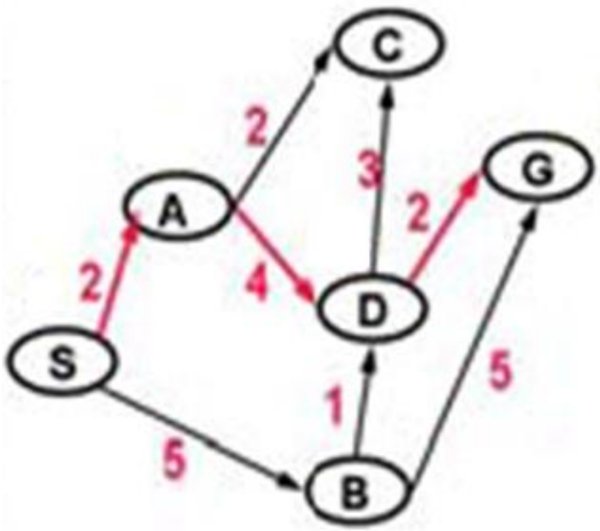
- ❑ **Blue Color represents added paths**
- ❑ <u>Underline</u> paths are selected for extension.

Uniform-Cost Search

(with strict Expanded List)

- Pick best (by **path length**) element of Q
- Add path extensions to Q

	Q	Expanded
1	(0 S)	S
2	(2 A S) (5 B S)	S, A
3	(4 C A S) (6 D A S) (5 B S)	S, A, C
4	(6 D A S) (5 B S)	S, A, C, B
5	(~~6 D B S~~) (10 G B S) (6 D A S)	S, A, C, B, D



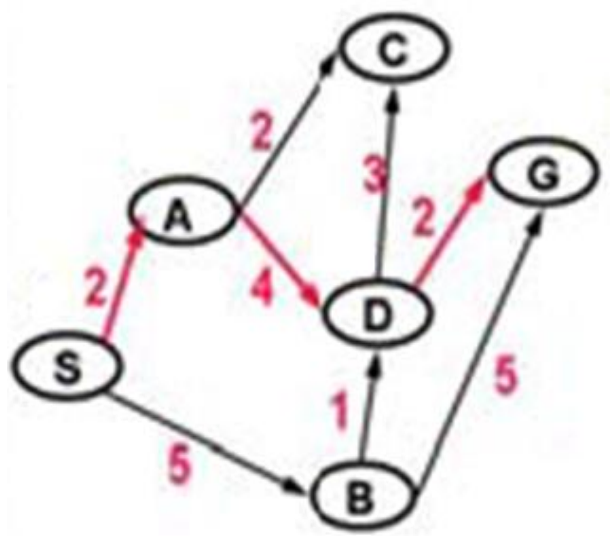
- **Blue Color represents added paths**
- Underline paths are selected for extension.

Uniform-Cost Search

(with strict Expanded List)

- Pick best (by **path length**) element of Q
- Add path extensions to Q

	Q	Expanded
1	(0 S)	S
2	(2 A S) (5 B S)	S, A
3	(4 C A S) (6 D A S) (5 B S)	S, A, C
4	(6 D A S) (5 B S)	S, A, C, B
5	(~~6 D B S~~) (10 G B S) (6 D A S)	S, A, C, B, D
6	(8 G D A S) (~~9 C D A S~~) (~~10 G B S~~)	S, A, C, B, D, **G**



- **Blue Color represents added paths**
- Underline paths are selected for extension.

Uniform-Cost Search

- Uniform-cost search is **concerned only with expanding shortest paths**.
- It pays **no particular attention to the goal** (since it has no any way of knowing where it is).

*Uniform-cost search is an algorithm for finding the **shortest paths** to all states in a graph rather than being focused in reaching a particular goal.*

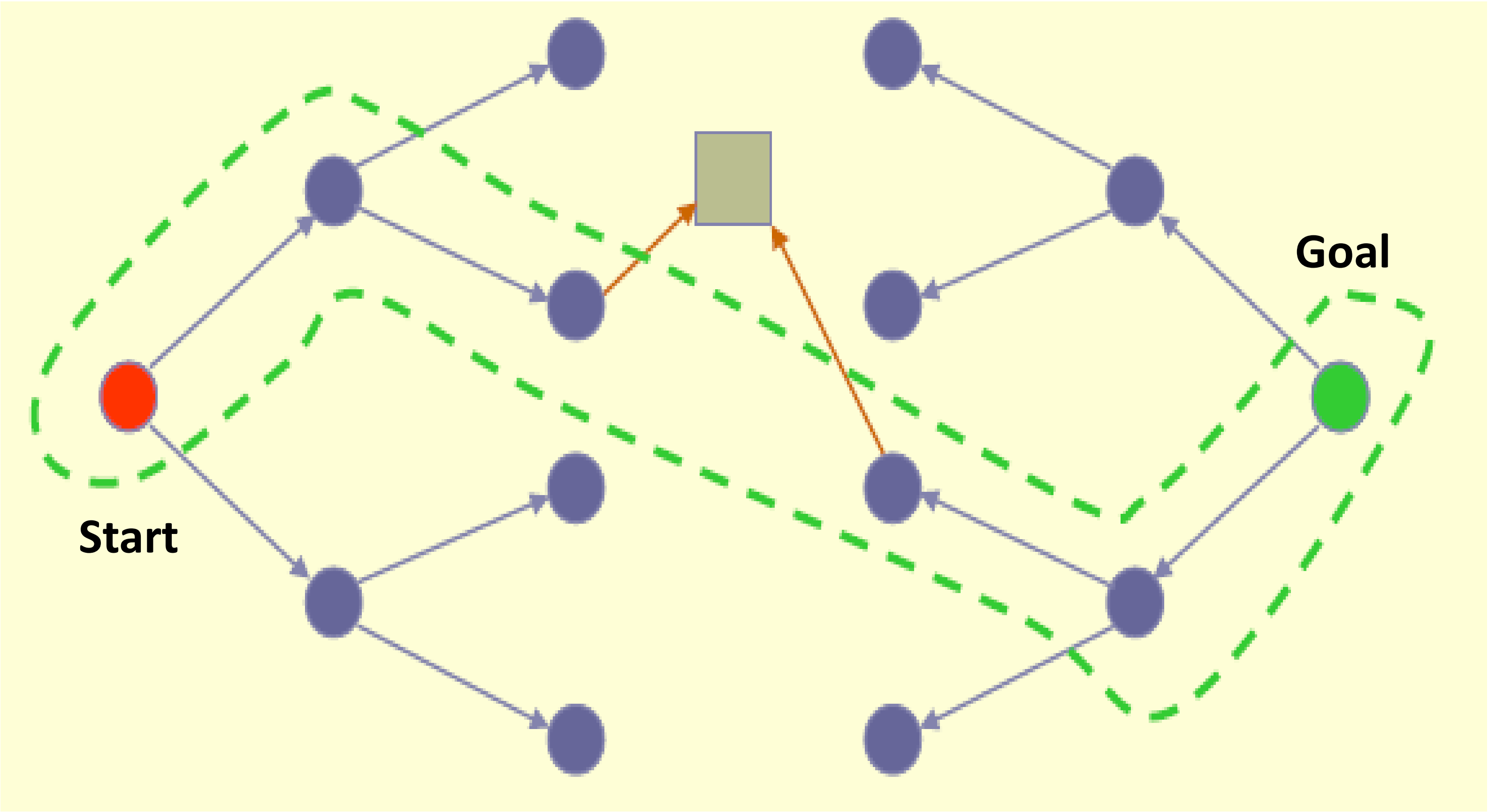
Bidirectional Search

Bidirectional Search

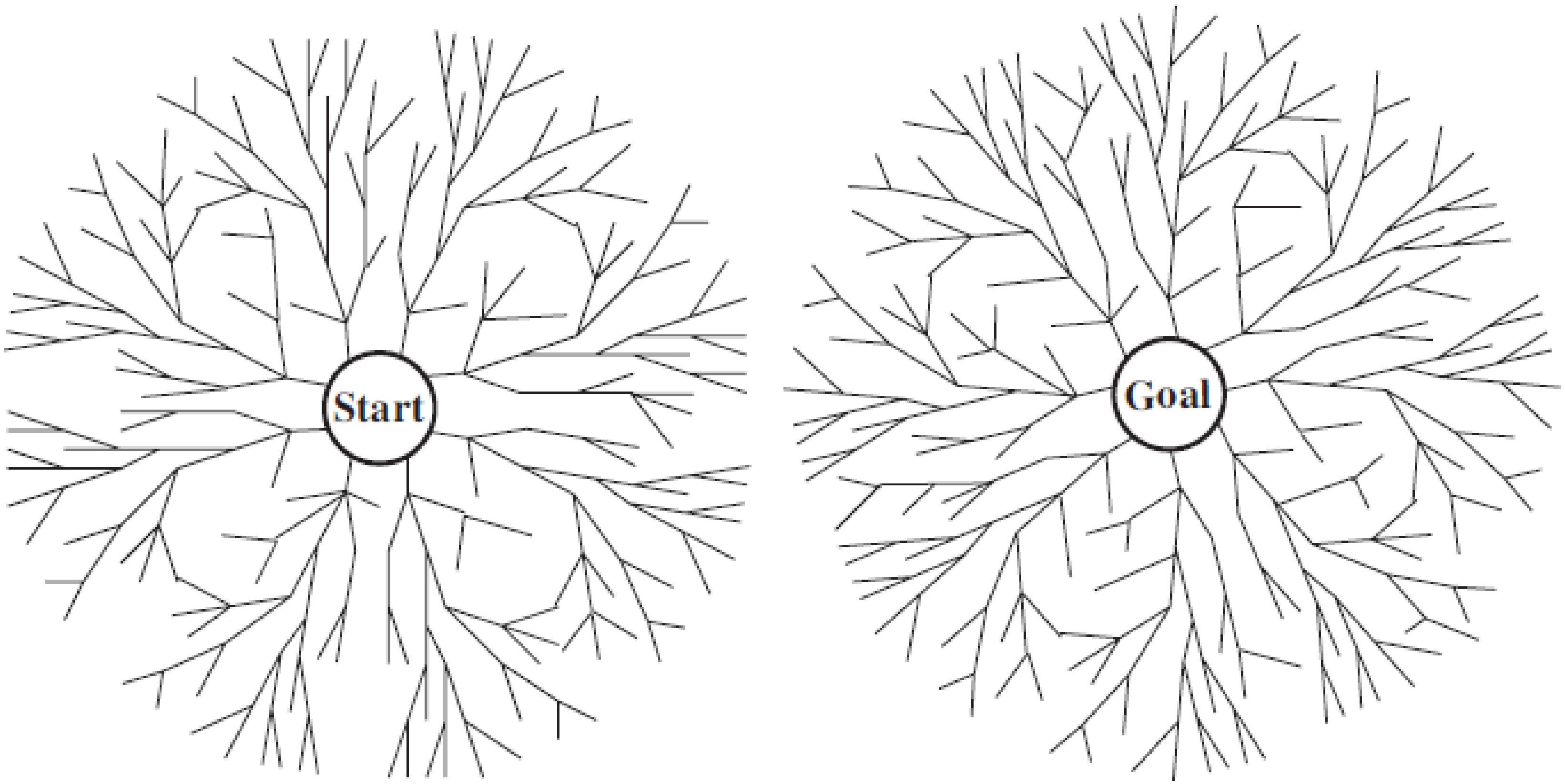
- Run two simultaneous searches
  - forward from the initial state
  - backward from the goal
  - stop when the two searches meet

- However, computing backward is difficult
  - A huge amount of goal states
  - At the goal state, which actions are used to compute it?
  - Can the actions be reversible to compute its predecessors?

Bidirectional Search



Bidirectional Search



A schematic view of a bidirectional search that is about to succeed when a branch from the start node **meets** a branch from the goal node.

Bidirectional Search

- Time and space complexity

$$O(b^{d/2}) \ll O(b^d)$$

For example,

- If a problem has solution depth $d = 6$, and each direction runs breadth-first search one node at a time, then in the worst case the two searches meet when they have generated all of the nodes at depth 3.
- For $b = 10$, this means a total of 2,220 node generations, compared with 1,111,110 for a standard breadth-first search.

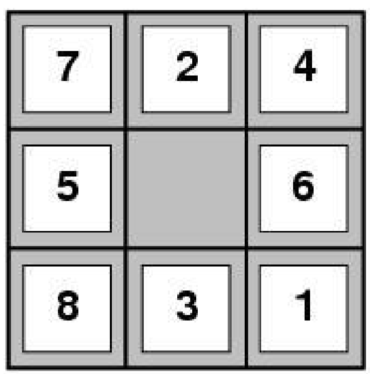
$$(10 + 100 + 1000 + 10{,}000 + 100{,}000 + 1{,}000{,}000) = 1{,}111{,}110$$

$$2 \times (10 + 100 + 1000) = 2220$$

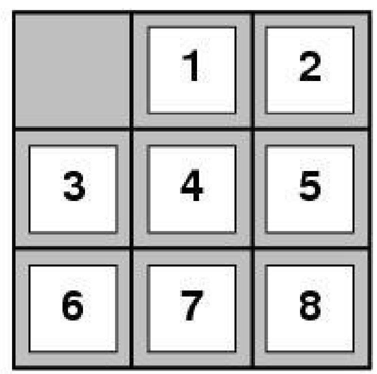
Bidirectional Search

"Backward from the goal." How?

- **For example**, the **8-puzzle problem** or for **finding a route in Romania**, there is just one goal state, so the backward search is very much like the forward search,



Start State



Goal State

Evaluation of tree-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]

- ***b***: branching factor
- ***d***: depth of the shallowest solution;
- ***m***: maximum depth of the search tree;
- ***l***: is the depth limit.

a) complete if ***b*** is finite
b) complete if step costs $\geq$ ε for positive ε ;
c) optimal if step costs are all identical;
d) if both directions use breadth-first search

Avoiding Repeated States

For all search strategies:

- There is possibility of **expanding states that have already been encountered and expanded before**, on some other path
  - It may cause the path to be infinite
  - loop forever
- Algorithms that forget their history are doomed to repeat it.

Avoiding Repeated States

Three ways to deal with this possibility:

- **Do not return to the state** it just came from
  - Refuse generation of any successor same as its **parent state**
- Do not create **paths with cycles**
  - Refuse generation of any successor same as its **ancestor states**
- Do not generate any ***generated state***
  - Not only its ancestor states, but also all other expanded states have to be checked against

Reading Material

- **Artificial Intelligence,** A Modern Approach

  Stuart J. Russell and Peter Norvig
  - **Chapter 3.**