**BTVN:**

**Tìm hiểu các loại search và ghi ra các tiêu chí về Complete, Space, Time, Optimal.**

***Hạn: 6/11/2021***

----------------------------------------------- ========== -------------------------------------------------

**- Các loại search:**

**+ Uninformed Search (Blind Search / Tìm kiếm mù):**

* **Tìm kiếm theo chiều sâu (Depth First Search).**
* **Tìm kiếm theo chiều rộng (Breadth First Search).**
* **Áp dụng thuật toán Dijkstra (Uniform Cost Search)**.
* **Best First Search.**
* **Iterative deepening search.**
* **Bidirrectional search.**

## **+ Informed Search (Heuristic Search/Heuristic Function / Tìm kiếm dựa kinh nghiệm):**

* **Greedy Best First Search.**
* **A\* search.**
* **Bidirectional A\* search.**
* **IDA\*.**
* **RBFS (recursive best-first search).**
* **SMA\* (simplified memory-bounded A\*).**

**- Các tiêu chí về Complete, Space, Time, Optimal**

* **Breadth First Search:**

**-** Complete:

* This is a systematic search strategy that is therefore complete even on infinite state spaces.
* It is complete in either case.

- Space:

* The memory requirements are a bigger problem for breadth-first search than the execution time.

- Time:

* We could implement Breadth-First Search as a call to BEST-FIRST-SEARCH where the evaluation function f(n) is the depth of the node—that is, the number of actions it takes to reach the node.
* We can do an **early goal test**, checking whether a node is a solution as soon as it is generated, rather than the **late goal test** that best-first search uses, waiting until a node is popped off the queue.
* Breadth-first search always finds a solution with a minimal number of actions.
* It is generating nodes at depth *d*, it has already generated all the nodes at depth *d - 1*, so if one of them were a solution, it would have been found.

- Optimal:

* Breadth-first search is cost-optimal.
* **Uniform Cost Search: (Dijkstra’s algorithm)**

**-** Complete:

* Uniform-cost search is complete.

- Space:

* Uniform-cost search can explore large trees of actions with low costs before exploring paths involving a high-cost and perhaps useful action.

- Time:

* The first solution it finds will have a cost that is at least as low as the cost of any other node in the frontier.
* Uniform-cost search considers all paths systematically in order of increasing cost, never getting caught going down a single infinite path.
* It finds a lower cost, so it replaces the previous path in *reached* and is added to the *frontier*.
* It turns out this node now has the lowest cost, so it is considered next, found to be a goal, and returned.

- Optimal:

* Uniform-cost search is cost-optimal.
* When all action costs are equal, and uniform-cost search is similar to breadth-first search.
* **Depth First Search:**

**-** Complete:

* For finite state spaces that are trees it is efficient and complete.
* For acyclic state spaces it may end up expanding the same state many times via different paths, but will (eventually) systematically explore the entire space.
* In cyclic state spaces it can get stuck in an infinite loop; therefore some implementations of depth-first search check each new node for cycles. Finally, in infinite state spaces, depth- first search is not systematic: it can get stuck going down an infinite path, even if there are no cycles. Thus, depth-first search is incomplete.

- Space:

* Depth-first search has much smaller needs for memory.
* Depth-first search don’t keep a *reached* table at all, and the frontier is very small.
* A variant of depth-first search called backtracking search uses even less memory.

- Time:

* Always expands the *deepest* node in the frontier first.
* It could be implemented as a call to BEST-FIRST-SEARCH where the evaluation function *f* is the negative of the depth.
* Search proceeds immediately to the deepest level of the search tree, where the nodes have no successors.
* The search then "backs up" to the next deepest node that still has unexpanded successors.

- Optimal:

* Depth-first search is not cost-optimal.
* It returns the first solution it finds, even if it is not cheapest.