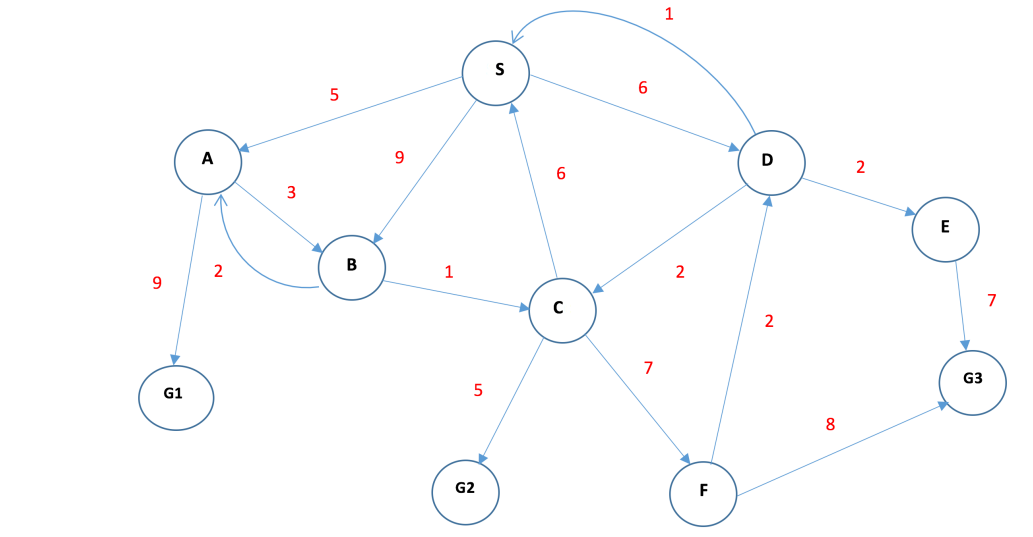
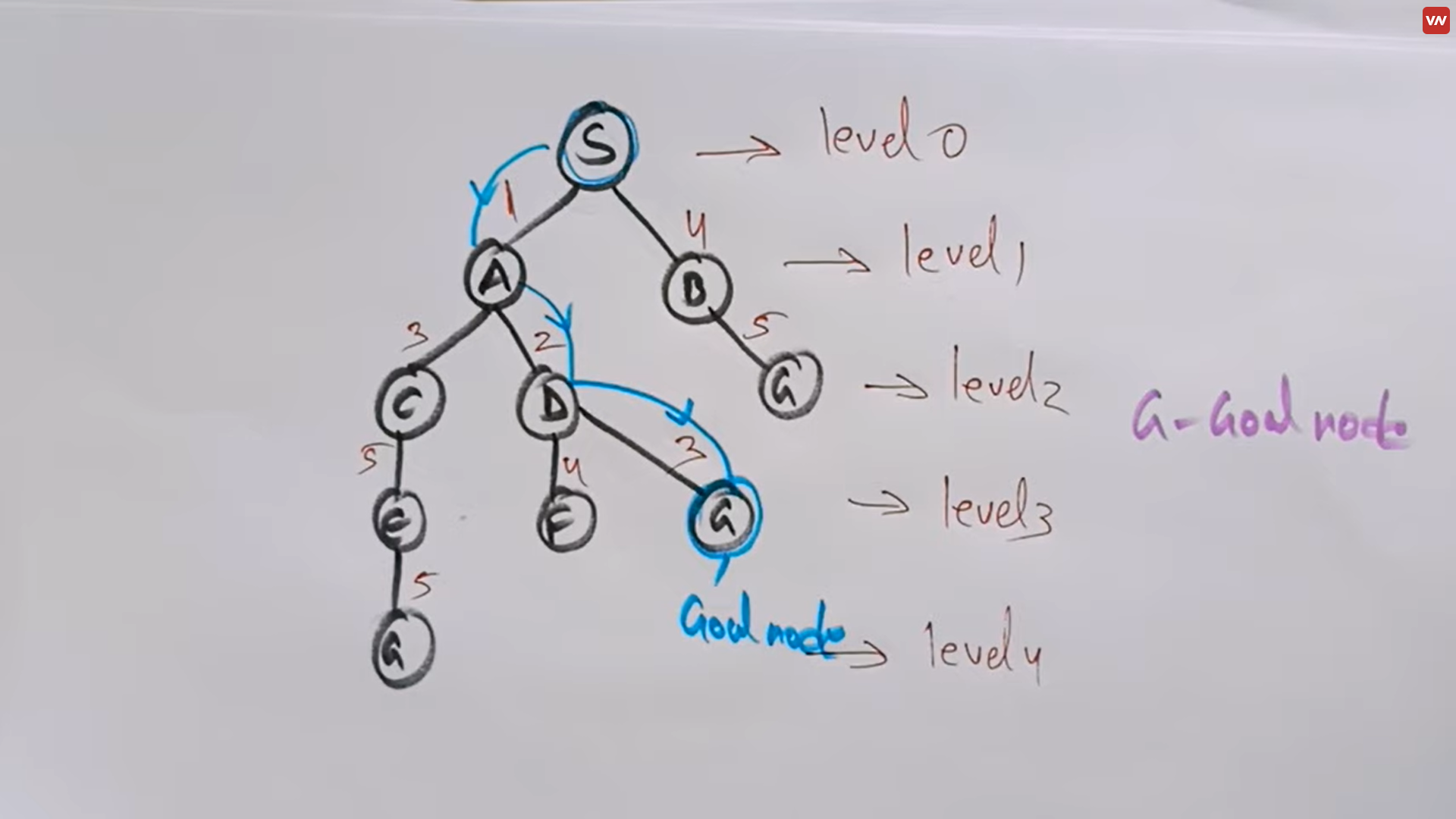
**TUTORIAL 4**

1) What is the reason why “Uniform cost search” is called so? Explain with an example toy graph

* It is used to traverse a weighted tree or graph.
* It comes into play when a different cost is available for each edge.
* The goal of UCS is to find a path to the goal node which has the lowest cumulative cost.
* It expands nodes according to their path costs from the root.
* It can be used to solve any graph or tree where the optimal cost is in demand.
* A UCS algorithm is implemented using priority queue.
* It gives priority to the lowest cumulative cost.



2) What is “Dept Limited Search”? How can it be considered as an extension to the “Depth First Search”?

imposes a maximum depth limit on the search. In DFS, the algorithm explores as far as possible along each branch before backtracking. However, this can lead to an infinite loop in graphs with cycles or very deep paths.

To overcome this issue, Depth-Limited Search sets a limit on the depth of the search tree. It restricts the depth of exploration to a certain level before backtracking. By doing so, it avoids getting trapped in infinite paths or deep branches.

3) How does “Iterative Deepening DFS” work? Someone says it is a combination of both BFS and DFS” Provide justifications to this statement

Iterative Deepening Depth-First Search (IDDFS) is a search algorithm that combines the benefits of both Breadth-First Search (BFS) and Depth-First Search (DFS) algorithms. It repeatedly performs depth-limited searches, gradually increasing the depth limit with each iteration until a solution is found.

In IDDFS, the search starts with a depth limit of 0 and gradually increases the depth limit with each iteration until the goal is found or the search space is exhausted. At each depth limit, IDDFS performs a complete DFS within that limit, similar to regular DFS. If the goal is not found, IDDFS increases the depth limit and performs DFS again.

By gradually increasing the depth limit, IDDFS explores the search space in a depth-first manner, similar to DFS. However, by performing multiple iterations with increasing depth limits, IDDFS effectively covers the search space layer by layer, similar to BFS.

The justification for IDDFS being a combination of BFS and DFS lies in its ability to achieve the completeness of BFS and the memory efficiency of DFS. Since IDDFS performs a complete DFS within each depth limit, it guarantees completeness by ensuring that all nodes within the current depth limit are explored. By using a depth-first approach, it maintains the memory efficiency of DFS as it only needs to store the current path and backtracks when necessary.

4) How does “informed search” differ from “uninformed search”? Why can we consider “informed search” is more efficient compared to the “uninformed search”?

Uninformed Search: Uninformed search algorithms do not have any additional information about the problem domain beyond the problem definition itself. They explore the search space systematically, considering all possible paths without any prior knowledge. Examples of uninformed search algorithms include Breadth-First Search (BFS) and Depth-First Search (DFS).

Informed Search: Informed search algorithms, on the other hand, utilize additional knowledge or heuristics about the problem domain to guide the search process. They make use of problem-specific information to estimate the desirability of different states or paths in the search space. Informed search algorithms prioritize the exploration of more promising paths likely to lead to the goal. Examples of informed search algorithms include Best-First Search, A\* Search, and Greedy Search.

Efficiency of Informed Search over Uninformed Search: Informed search is generally considered more efficient compared to uninformed search for several reasons:

1. Exploiting Problem-Specific Information: Informed search algorithms make use of domain-specific knowledge or heuristics to guide the search towards the goal state. This additional information helps in making informed decisions about which paths to explore, leading to more efficient search processes.
2. Reduced Search Space: By utilizing heuristics, informed search algorithms can often prune or avoid exploring paths that are unlikely to lead to the goal. This helps in reducing the search space, thereby saving computational resources and time.
3. Faster Convergence: Informed search algorithms tend to converge faster towards the goal state compared to uninformed search algorithms. By prioritizing more promising paths based on heuristics, they can focus on potential solutions, avoiding unnecessary exploration of unpromising paths.
4. Optimality: Informed search algorithms can also find optimal solutions (if they exist) more efficiently compared to uninformed search algorithms. By using problem-specific information, they can make better-informed choices, leading to optimal or near-optimal solutions.

5) What do you mean by a heuristic cost function? Give example heuristic cost functions that can be used for 2D space.

A heuristic cost function, also known as a heuristic function or an evaluation function, is used in informed search algorithms to estimate the cost or desirability of reaching a goal state from a given state in a search problem. The heuristic cost function provides a heuristic value, often an estimated distance or cost, for each state, which guides the search algorithm in making informed decisions.

1. Manhattan Distance: Also known as the L1 distance or city block distance, this heuristic calculates the sum of the absolute differences between the x and y coordinates of the current state and the goal state. It assumes movement can only occur horizontally or vertically, not diagonally. The Manhattan distance provides a good estimate in grid-based environments.

2. Chess game or to find the minimum distance to a location in a map.

6) What are the two types of informed search algorithms that we discussed in the class? List down few similarities and differences between those informed search algorithms.

1) Best first search

2) A\* Search

**DIFFERENCES**

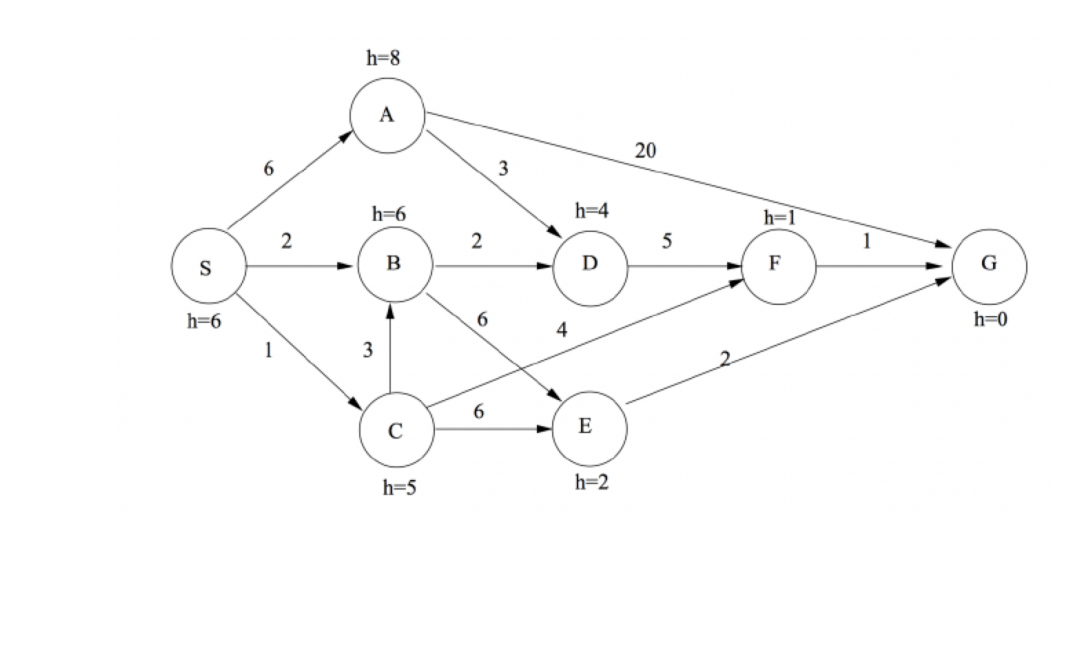
|  |  |
| --- | --- |
| BEST FIRST SEARCH | A\* SEARCH |
| Best-First Search solely relies on the heuristic estimate (h) and does not consider the actual path cost. It does not guarantee optimality and can be influenced solely by the heuristic value. | A\* Search considers both the actual cost of the path (g) and the heuristic estimate (h) to evaluate the total estimated cost of the path |
| Best-First Search does not provide an optimality guarantee. It can find a solution quickly based on the heuristic evaluation, but it may not be optimal, as it disregards the path cost. | A\* Search guarantees finding an optimal solution if the heuristic is admissible (never overestimates the cost) and consistent (satisfies the triangle inequality). It considers both the path cost and the heuristic estimate to make informed decisions. |
| Best-First Search generally requires less memory as it only needs to store the heuristic evaluations and does not consider the path cost. | A\* Search typically uses more memory as it needs to store information about the path cost (g) and maintain a priority queue to prioritize the states based on the total estimated cost (f). |

**SIMILARITIES**

Both algorithms utilize a priority queue.

Both algorithms employ a heuristic function to estimate the desirability or cost of reaching the goal from a given state.

7) Execute Greedy Search and A\* search for the following graph individually. Output the traversal order for each algorithm and make sure mention your steps clearly.



**BEST FIRST SEARCH**

1. OPEN LIST – S

CLOSED - \_\_\_

1. OPEN LIST – A,B,C

CLOSED – S

1. OPEN LIST – A,B

CLOSED – S,C

1. OPEN LIST – A,B,E,F

CLOSED – S,C

1. OPEN LIST – A,B,E,G

CLOSED – S,C,F

1. OPEN LIST – A,B,E

CLOSED – S,C,F,G

**SO THE TRAVERSAL ORDER IS S – C – F – G**

**A\* SEARCH**

|  |  |
| --- | --- |
| STATE | H(n) |
| A | 8 |
| B | 6 |
| C | 5 |
| D | 4 |
| E | 2 |
| F | 1 |
| S | 6 |
| G | 0 |

1. S – A = 12, S – B = 8, S – C = 7

SO S – C has the lowest so S – C

1. S – C – B = 10

S – C – E = 9

S – C – F = 5 so this path is considered

1. S – C – F – G = 0

SO THE PATH IS S – C – F – G