Great! Here's a clear explanation and Python code for two Informed Search Algorithms:

🧠 Informed Search Algorithms

Informed (or heuristic) search algorithms use heuristic functions to estimate the cost from a node to the goal, helping the algorithm search more efficiently than uninformed methods.

🔍 1. Best-First Search (Greedy Best-First Search)

✅ Idea:

- Always chooses the node that appears to be closest to the goal based only on the heuristic h(n).

✅ Python Code:

```
import heapq


def best_first_search(graph, start, goal, heuristic):
    visited = set()
    queue = [(heuristic[start], [start])]

    while queue:
        _, path = heapq.heappop(queue)
        node = path[-1]

        if node == goal:
            return path

        if node not in visited:
            visited.add(node)
            for neighbor in graph.get(node, []):
                new_path = list(path)
                new_path.append(neighbor)
                heapq.heappush(queue, (heuristic[neighbor], new_path))
```

```
    return None

# Example graph
graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': ['F'],
    'F': []
}

# Heuristic values for each node
heuristic = {
    'A': 5,
    'B': 4,
    'C': 2,
    'D': 6,
    'E': 3,
    'F': 0  # Goal
}

print("Best-First Search Path:", best_first_search(graph, 'A', 'F', heuristic))
```

⭐ **2. *A Search Algorithm****

✅ **Idea:**

- **Uses both actual cost g(n) from start and estimated cost h(n) to goal:**

f(n) = g(n) + h(n)

✅ **Python Code:**

```
def a_star_search(graph, start, goal, heuristic):
    queue = [(heuristic[start], 0, [start])]  # (f(n), g(n), path)
```

```
    visited = set()

    while queue:
        f, g, path = heapq.heappop(queue)
        node = path[-1]

        if node == goal:
            return path

        if node not in visited:
            visited.add(node)
            for neighbor in graph.get(node, []):
                if neighbor not in visited:
                    new_g = g + 1  # Assuming all edges cost 1
                    new_f = new_g + heuristic[neighbor]
                    new_path = list(path)
                    new_path.append(neighbor)
                    heapq.heappush(queue, (new_f, new_g, new_path))

    return None

print("A* Search Path:", a_star_search(graph, 'A', 'F', heuristic))
```

✅ **Example Output:**

```
Best-First Search Path: ['A', 'C', 'F']
A* Search Path: ['A', 'C', 'F']
```

🔁 **Summary Table**

Algorithm	Uses Heuristic	Uses Path Cost	Strategy
Best-First Search	✅ h(n)	❌	Greedy (closest to goal)

Algorithm	Uses Heuristic	Uses Path Cost	Strategy
A* Search	✅ h(n)	✅ g(n)	Optimal + Efficient