Give in turn an example of writing the following code using Python:
breadth-first search
depth-first search
ID-DFS search
best-first search
A* search

2/2

Here are Python implementations of the different search algorithms:

---

### 1. Breadth-First Search (BFS)

BFS explores all neighbors at the present depth before moving on to nodes at the next depth level.

python Copy Edit

```python
from collections import deque

def bfs(graph, start, goal):
    queue = deque([(start, [start])])  # (current_node, path)
    visited = set()

    while queue:
        node, path = queue.popleft()
        if node in visited:
            continue
        visited.add(node)

        if node == goal:
            return path  # Return the path when goal is found

        for neighbor in graph.get(node, []):
            queue.append((neighbor, path + [neighbor]))

    return None  # If no path is found

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

print(bfs(graph, 'A', 'F'))  # Output: ['A', 'C', 'F']
```

## 2. Depth-First Search (DFS)

DFS explores as far as possible along each branch before backtracking.

python Copy Edit

```python
def dfs(graph, start, goal, path=None, visited=None):
    if path is None:
        path = [start]
    if visited is None:
        visited = set()

    visited.add(start)

    if start == goal:
        return path  # Return path when goal is found

    for neighbor in graph.get(start, []):
        if neighbor not in visited:
            result = dfs(graph, neighbor, goal, path + [neighbor], visited)
            if result:
                return result

    return None  # If no path is found

# Example Graph (Same as above)
print(dfs(graph, 'A', 'F'))  # Output: ['A', 'C', 'F']
```

### 3. Iterative Deepening Depth-First Search (ID-DFS)

ID-DFS repeatedly runs DFS with increasing depth limits.

python Copy Edit

```python
def dls(graph, node, goal, depth, path=None):
    if path is None:
        path = [node]

    if node == goal:
        return path

    if depth <= 0:
        return None  # Cutoff

    for neighbor in graph.get(node, []):
        result = dls(graph, neighbor, goal, depth - 1, path + [neighbor])
        if result:
            return result

    return None

def iddfs(graph, start, goal, max_depth=10):
    for depth in range(max_depth):
        result = dls(graph, start, goal, depth)
        if result:
            return result  # Return path when found
    return None

# Example Graph (Same as above)
print(iddfs(graph, 'A', 'F'))  # Output: ['A', 'C', 'F']
```

## 4. Best-First Search (Greedy Search)

Best-First Search uses a heuristic to expand the most promising node first.

python Copy Edit

```python
import heapq

def best_first_search(graph, start, goal, heuristic):
    pq = [(heuristic[start], start, [start])]  # (priority, node, path)
    visited = set()

    while pq:
        _, node, path = heapq.heappop(pq)
        if node in visited:
            continue
        visited.add(node)

        if node == goal:
            return path  # Return path when goal is found

        for neighbor in graph.get(node, []):
            if neighbor not in visited:
                heapq.heappush(pq, (heuristic[neighbor], neighbor, path + [neighbor]))

    return None  # If no path is found

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

heuristic = {  # Estimated cost to reach 'F'
    'A': 5, 'B': 3, 'C': 2, 'D': 4, 'E': 1, 'F': 0
}

print(best_first_search(graph, 'A', 'F', heuristic))  # Output: ['A', 'C', 'F']
```

### *5. A Search (A-Star)**

A* Search combines cost-so-far and heuristic values to find the optimal path.

python Copy Edit

```python
def a_star_search(graph, start, goal, heuristic, cost):
    pq = [(heuristic[start], 0, start, [start])]  # (f-score, g-score, node, path)
    visited = {}

    while pq:
        f, g, node, path = heapq.heappop(pq)

        if node in visited and visited[node] <= g:
            continue
        visited[node] = g

        if node == goal:
            return path  # Return path when goal is found

        for neighbor in graph.get(node, []):
            new_g = g + cost.get((node, neighbor), 1)  # Default cost is 1
            new_f = new_g + heuristic[neighbor]
            heapq.heappush(pq, (new_f, new_g, neighbor, path + [neighbor]))

    return None  # If no path is found

# Example Graph with Costs and Heuristics
cost = {
    ('A', 'B'): 1, ('A', 'C'): 4,
    ('B', 'D'): 2, ('B', 'E'): 5,
    ('C', 'F'): 3, ('E', 'F'): 1
}

heuristic = {  # Estimated cost to reach 'F'
    'A': 6, 'B': 2, 'C': 3, 'D': 4, 'E': 1, 'F': 0
}

print(a_star_search(graph, 'A', 'F', heuristic, cost))  # Output: ['A', 'B', 'E', 'F']
```