

Source code:

```
print("Water Jug Problem")

# Initial values for the jugs
x = int(input("Enter the initial amount in Jug1 (x): "))
y = int(input("Enter the initial amount in Jug2 (y): "))

while True:
    r = int(input("Enter the rule number (1-10): "))
    if r == 1:
        if x < 4: # Fill Jug1 to its capacity (4 liters)
            x = 4

    elif r == 2:
        if y < 3: # Fill Jug2 to its capacity (3 liters)
            y = 3

    elif r == 5:
        if x > 0: # Empty Jug1
            x = 0

    elif r == 6:
        if y > 0: # Empty Jug2
            y = 0

    elif r == 7:
        if x + y >= 4 and y > 0:
            # Pour water from Jug2 to Jug1 until Jug1 is full or Jug2 is empty
            x, y = 4, y - (4 - x)

    elif r == 8:
        if x + y >= 3 and x > 0:
            # Pour water from Jug1 to Jug2 until Jug2 is full or Jug1 is empty
            x, y = x - (3 - y), 3

    elif r == 9:
        if x + y <= 4 and y > 0: # Pour all water from Jug2 into Jug1
            x, y = x + y, 0

    elif r == 10:
        if x + y <= 3 and x > 0: # Pour all water from Jug1 into Jug2
            x, y = 0, x + y

    # Display the current state of both jugs
    print("x =", x)
    print("y =", y)

    # Check if the goal state (2 liters in Jug1) is reached
```

```
if x == 2 and y==0:  
    print("The result is a Goal State")  
    break
```

Output:

```
PS C:\Users\sabik\OneDrive\Desktop\PyChapter1> python -u "c:\Users\sabik\OneDrive\Desktop\PyChapter1\AI.py\waterjug.py"  
Water Jug Problem  
Enter the initial amount in Jug1 (x): 0  
Enter the initial amount in Jug2 (y): 0  
Enter the rule number (1-10): 1  
x = 4  
y = 0  
Enter the rule number (1-10): 8  
x = 1  
y = 3  
Enter the rule number (1-10): 6  
x = 1  
y = 0  
Enter the rule number (1-10): 10  
x = 0  
y = 1  
Enter the rule number (1-10): 1  
x = 4  
y = 1  
Enter the rule number (1-10): 8  
x = 2  
y = 3  
Enter the rule number (1-10): 6  
x = 2  
y = 0  
The result is a Goal State
```

Compiler: VSC

Language: Python

Source code:

```
print("\n\t{:^60}".format("Madan Bhandari Memorial College"))
print("\t{:^60}".format("Hey there!!"))
print(" - Simple Question Answer - ")
print("-----")
print("You may ask any one of these questions:")
print("1. Hi")
print("2. How are you?")
print("3. I want to know the procedure for admission")
print("4. What are the necessary documents?")
print("5. Where is the college located?")
print("6. How can I contact the college?")
print("7. Good Bye")
while True:
    question = input("\nEnter one question from above list: ").strip().lower()
    if question == 'hi':
        print("Hello!")
    elif question == 'how are you?':
        print("I am fine, thank you!")
    elif question == 'i want to know the procedure for admission':
        print("First, you need to fill up the admission form, which can be collected from the front desk at the college. Submit it with all the necessary documents.\nThen, the selection will be done through an interview process.")
    elif question == 'what are the necessary documents?':
        print("The necessary documents are:\n1. High School Certificates\n2. Citizenship\n3. Marksheet")
    elif question == 'where is the college located?':
        print("The college is located at Binayak Nagar, New Baneshwor, near Bhatbhatini Super Market.")
    elif question == 'how can i contact the college?':
```

```

print("You can contact the college by:\n1. Emailing us at mbmc@gmail.com\n2.
Calling us at 1234567\n3. Visiting the front desk during working hours.")

elif question == 'good bye':

print("Good Bye! Have a great day!")

    break

else:

print("Error! Please ask a valid question from the list.")

```

Output:

```

PS C:\Users\sabik\OneDrive\Desktop\PyChapter1> python -u "c:\Users\sabik\OneDrive\Desktop\PyChapter1\AI.py\chatbot.py"

          Madan Bhandari Memorial College
          Hey there!!

- Simple Question Answer -
-----
You may ask any one of these questions:
1. Hi
2. How are you?
3. I want to know the procedure for admission
4. What are the necessary documents?
5. Where is the college located?
6. How can I contact the college?
7. Good Bye

Enter one question from above list: Hi
Hello!

Enter one question from above list: How are you?
I am fine, thank you!

Enter one question from above list: I want to know
the procedure for admission
First, you need to fill up the admission form, which can be collected from the front desk at the college. Submit it with all the necessary documents.
Then, the selection will be done through an interview process.

Enter one question from above list: What are the necessary documents?
The necessary documents are:
1. High School certificates
2. Citizenship
3. Marksheet

Enter one question from above list: How can I contact the college?
You can contact the college by:
1. Emailing us at mbmc@gmail.com
2. Calling us at 1234567
3. Visiting the front desk during working hours.

Enter one question from above list: Good Bye
Good Bye! Have a great day!

```

Compiler: VSC

Language: Python

Source code:

```
from collections import deque

# Define the graph
graph = {
    'A': ['B', 'D', 'E'],
    'B': ['C'],
    'C': [],
    'D': [],
    'E': ['F'],
    'F': ['G', 'H'],
    'G': [],
    'H': [],
}

def bfs_tree(graph, start):
    visited = []
    queue = deque([start])
    parent = {start: None}

    while queue:
        node = queue.popleft()
        if node not in visited:
            visited.append(node)
            print(node, end=" ")

            for neighbour in graph[node]:
                if neighbour not in visited:
                    queue.append(neighbour)
                    parent[neighbour] = node

    return parent

def print_tree(parent, root):
    print("\nTree structure:")
    for node in parent:
        if parent[node] is not None:
            print(f"{parent[node]} -> {node}")

def main():
    root_node = input("Enter the root node: ")
    goal_node = input("Enter the goal node: ")

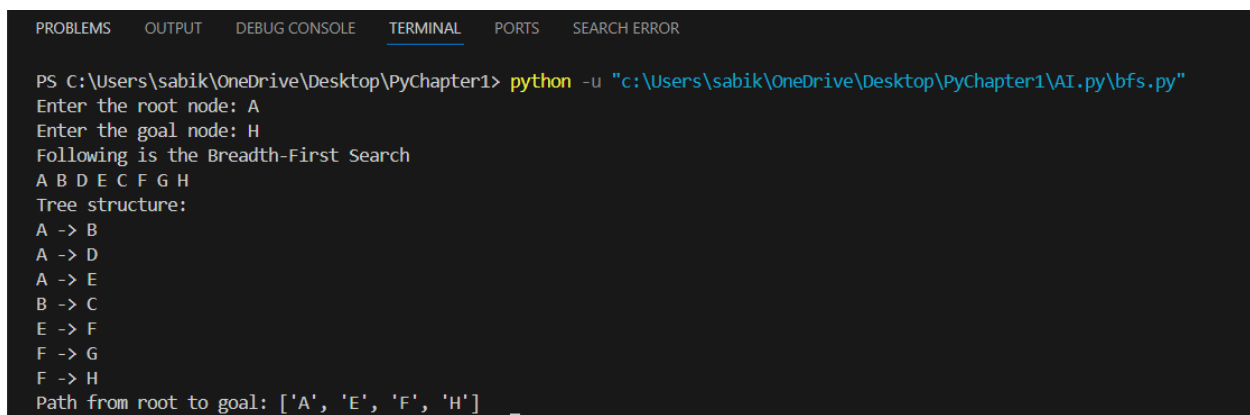
    print("Following is the Breadth-First Search")
    parent = bfs_tree(graph, root_node)

    print_tree(parent, root_node)

    # Additional logic to find the path from root to goal
```

```
path = []
current_node = goal_node
while current_node is not None:
    path.append(current_node)
    current_node = parent[current_node]
path.reverse()
print("Path from root to goal:", path)
if __name__ == "__main__":main()
```

Output:



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SEARCH ERROR

PS C:\Users\sabik\OneDrive\Desktop\PyChapter1> python -u "c:\Users\sabik\OneDrive\Desktop\PyChapter1\AI.py\bfs.py"
Enter the root node: A
Enter the goal node: H
Following is the Breadth-First Search
A B D E C F G H
Tree structure:
A -> B
A -> D
A -> E
B -> C
E -> F
F -> G
F -> H
Path from root to goal: ['A', 'E', 'F', 'H']
```

Compiler: VSC

Language: Python

Source code:

```
from queue import PriorityQueue
v = 14
graph = [[] for _ in range(v)]
# Function for implementing Best First Search
def best_first_search(actual_src, target, n):
    visited = [False] * n
    pq = PriorityQueue()
    pq.put((0, actual_src))
    visited[actual_src] = True

    while not pq.empty():
        u = pq.get()[1]
        # Displaying the path having lowest cost
        print(u, end=" ")
        if u == target:
            break
        for v, c in graph[u]:
            if not visited[v]:
                visited[v] = True
                pq.put((c, v))
    print()

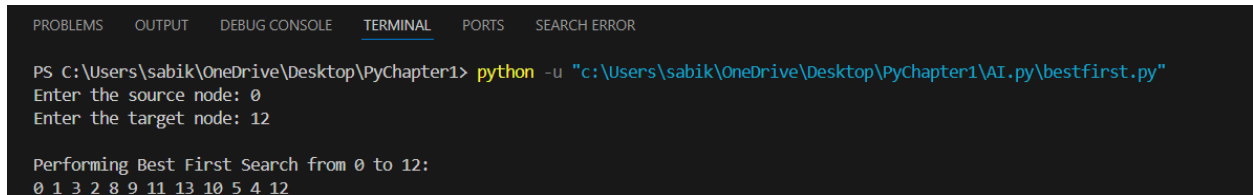
# Function for adding edges to the graph
def addedge(x, y, cost):
    graph[x].append((y, cost))
    graph[y].append((x, cost))

# Adding edges to the graph
addege(0, 1, 3)
addege(0, 2, 6)
addege(0, 3, 5)
addege(1, 4, 9)
addege(1, 5, 8)
addege(2, 6, 12)
addege(2, 7, 14)
addege(3, 8, 7)
addege(8, 9, 5)
addege(8, 10, 6)
addege(9, 11, 1)
addege(9, 12, 10)
addege(9, 13, 2)

# Asking user for source and target nodes
source = int(input("Enter the source node: ").strip())
target = int(input("Enter the target node: ").strip())
# Ensure the source and target are within the valid range
if 0 <= source < v and 0 <= target < v:
    print(f"\nPerforming Best First Search from {source} to {target}:")
    best_first_search(source, target, v)
else:
```

```
print("Invalid node values. Please enter values between 0 and", v-1)
```

Output:



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SEARCH ERROR

PS C:\Users\sabik\OneDrive\Desktop\PyChapter1> python -u "c:\Users\sabik\OneDrive\Desktop\PyChapter1\AI.py\bestfirst.py"
Enter the source node: 0
Enter the target node: 12

Performing Best First Search from 0 to 12:
0 1 3 2 8 9 11 13 10 5 4 12
```

Compiler: VSC

Language: Python

Source code:

```
from collections import deque
# Define the graph
graph = {
    'A': ['B', 'D', 'E'],
    'B': ['C'],
    'C': [],
    'D': [],
    'E': ['F'],
    'F': ['G', 'H'],
    'G': [],
    'H': [],
}

def dfs_tree(graph, start):
    visited = []
    stack = [start]
    parent = {start: None}

    while stack:
        node = stack.pop()
        if node not in visited:
            visited.append(node)
            print(node, end=" ")

            for neighbour in graph[node]:
                if neighbour not in visited:
                    stack.append(neighbour)
                    parent[neighbour] = node

    return parent

def print_tree(parent, root):
    print("\nTree structure:")
    for node in parent:
        if parent[node] is not None:
            print(f"{parent[node]} -> {node}")

def main():
    root_node = input("Enter the root node: ")
    goal_node = input("Enter the goal node: ")

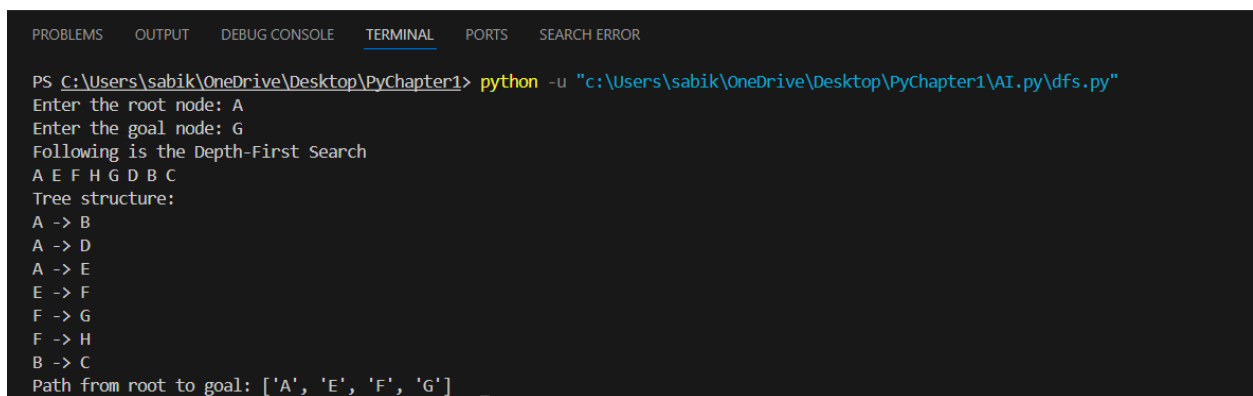
    print("Following is the Depth-First Search")
    parent = dfs_tree(graph, root_node)

    print_tree(parent, root_node)
    # Additional logic to find the path from root to goal
    path = []
    current_node = goal_node
```

```
while current_node is not None:
    path.append(current_node)
    current_node = parent[current_node]
path.reverse()
print("Path from root to goal:", path)

if __name__ == "__main__":
    main()
```

Output:



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SEARCH ERROR

PS C:\Users\sabik\OneDrive\Desktop\PyChapter1> python -u "c:\Users\sabik\OneDrive\Desktop\PyChapter1\AI.py\dfs.py"
Enter the root node: A
Enter the goal node: G
Following is the Depth-First Search
A E F H G D B C
Tree structure:
A -> B
A -> D
A -> E
E -> F
F -> G
F -> H
B -> C
Path from root to goal: ['A', 'E', 'F', 'G']
```

Compiler: VSC

Language: Python

Source code:

```
from queue import PriorityQueue

v = 14
graph = [[] for _ in range(v)]

# Function for implementing Best First Search
def best_first_search(actual_src, target, n):
    visited = [False] * n
    pq = PriorityQueue()
    pq.put((0, actual_src))
    visited[actual_src] = True

    while not pq.empty():
        u = pq.get()[1]
        # Displaying the path having lowest cost
        print(u, end=" ")
        if u == target:
            break
        for v, c in graph[u]:
            if not visited[v]:
                visited[v] = True
                pq.put((c, v))
    print()

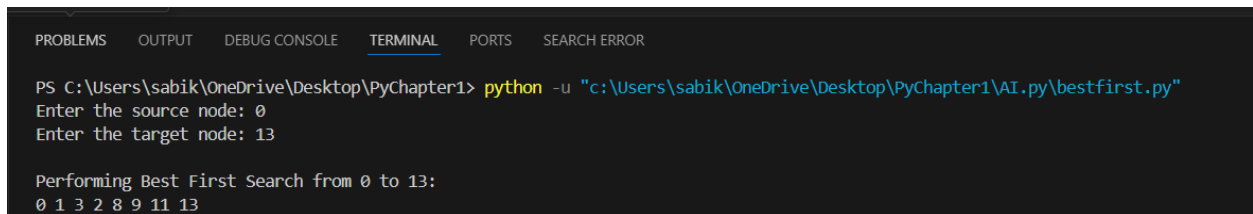
# Function for adding edges to the graph
def addedge(x, y, cost):
    graph[x].append((y, cost))
    graph[y].append((x, cost))

# Adding edges to the graph
addege(0, 1, 3)
addege(0, 2, 6)
addege(0, 3, 5)
addege(1, 4, 9)
addege(1, 5, 8)
addege(2, 6, 12)
addege(2, 7, 14)
addege(3, 8, 7)
addege(8, 9, 5)
addege(8, 10, 6)
addege(9, 11, 1)
addege(9, 12, 10)
addege(9, 13, 2)

# Asking user for source and target nodes
source = int(input("Enter the source node: ").strip())
target = int(input("Enter the target node: ").strip())
```

```
# Ensure the source and target are within the valid range
if 0 <= source < v and 0 <= target < v:
    print(f"\nPerforming Best First Search from {source} to {target}:")
    best_first_search(source, target, v)
else:
    print("Invalid node values. Please enter values between 0 and", v-1)
```

Output:



The screenshot shows a terminal window with a dark background. At the top, there are tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL' (which is active), 'PORTS', and 'SEARCH ERROR'. The terminal content shows the command prompt 'PS C:\Users\sabik\OneDrive\Desktop\PyChapter1>' followed by the command 'python -u "c:\Users\sabik\OneDrive\Desktop\PyChapter1\AI.py\bestfirst.py"'. Below this, the program prompts 'Enter the source node: 0' and 'Enter the target node: 13'. The output of the program is 'Performing Best First Search from 0 to 13:' followed by the sequence of nodes '0 1 3 2 8 9 11 13'.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SEARCH ERROR

PS C:\Users\sabik\OneDrive\Desktop\PyChapter1> python -u "c:\Users\sabik\OneDrive\Desktop\PyChapter1\AI.py\bestfirst.py"
Enter the source node: 0
Enter the target node: 13

Performing Best First Search from 0 to 13:
0 1 3 2 8 9 11 13
```

Compiler: VSC

Language: Python

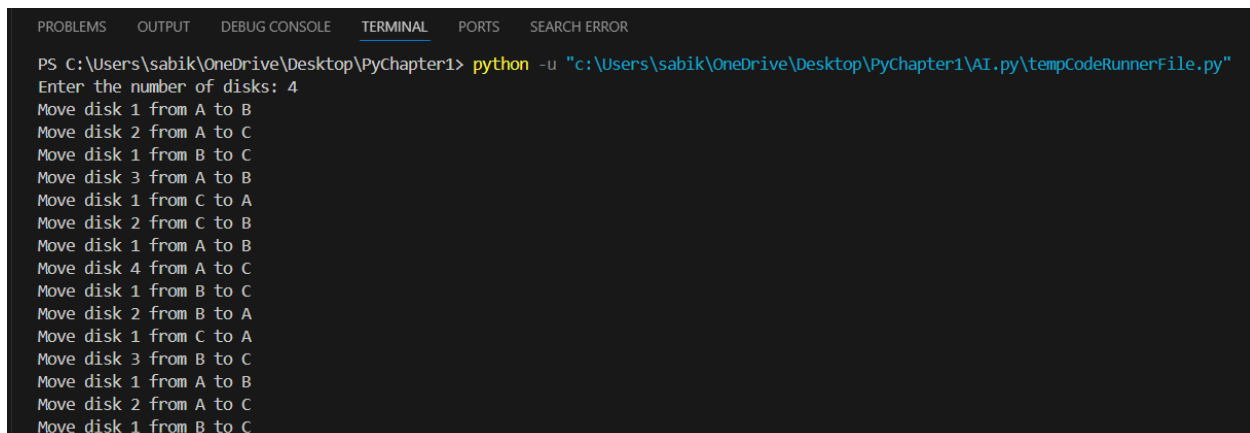
Source code:

```
def tower_of_hanoi(n, source, auxiliary, destination):
    if n == 1:
        print(f"Move disk 1 from {source} to {destination}")
        return
    tower_of_hanoi(n-1, source, destination, auxiliary)
    print(f"Move disk {n} from {source} to {destination}")
    tower_of_hanoi(n-1, auxiliary, source, destination)

# Ask the user for the number of disks
num_disks = int(input("Enter the number of disks: "))

# Initial call to the function with source 'A', auxiliary 'B', and destination 'C'
tower_of_hanoi(num_disks, 'A', 'B', 'C')
```

Output:



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SEARCH ERROR
PS C:\Users\sabik\OneDrive\Desktop\PyChapter1> python -u "c:\Users\sabik\OneDrive\Desktop\PyChapter1\AI.py\tempCodeRunnerFile.py"
Enter the number of disks: 4
Move disk 1 from A to B
Move disk 2 from A to C
Move disk 1 from B to C
Move disk 3 from A to B
Move disk 1 from C to A
Move disk 2 from C to B
Move disk 1 from A to B
Move disk 4 from A to C
Move disk 1 from B to C
Move disk 2 from B to A
Move disk 1 from C to A
Move disk 3 from B to C
Move disk 1 from A to B
Move disk 2 from A to C
Move disk 1 from B to C
```

Compiler: VSC

Language: Python

Source code:

```
from collections import deque
from queue import PriorityQueue

# Define the graph with edge costs
graph = {
    'A': {'B': 1, 'D': 3, 'E': 2},
    'B': {'C': 1},
    'C': {},
    'D': {},
    'E': {'F': 5},
    'F': {'G': 2, 'H': 3},
    'G': {},
    'H': {}
}

# Heuristic function for A* (assumed for this example)
def heuristic(node, goal):
    h_values = {
        'A': 7,
        'B': 6,
        'C': 2,
        'D': 6,
        'E': 3,
        'F': 1,
        'G': 0,
        'H': 0
    }
    return h_values[node]

# A* Search Algorithm Implementation
def a_star_search(graph, start, goal, h):
    open_list = PriorityQueue()
    open_list.put((0, start))
    came_from = {start: None}
    g_score = {node: float('inf') for node in graph}
    g_score[start] = 0
    f_score = {node: float('inf') for node in graph}
    f_score[start] = h(start, goal)

    while not open_list.empty():
        current = open_list.get()[1]

        if current == goal:
            return reconstruct_path(came_from, current)

        for neighbor, cost in graph[current].items():
            tentative_g_score = g_score[current] + cost
```

```

        if tentative_g_score < g_score[neighbor]:
            came_from[neighbor] = current
            g_score[neighbor] = tentative_g_score
            f_score[neighbor] = g_score[neighbor] + h(neighbor, goal)
            open_list.put((f_score[neighbor], neighbor))

    return None

# Helper function to reconstruct the path
def reconstruct_path(came_from, current):
    total_path = [current]
    while current in came_from and came_from[current] is not None:
        current = came_from[current]
        total_path.append(current)
    total_path.reverse()
    return total_path

def print_tree(parent, root):
    print("\nTree structure:")
    for node in parent:
        if parent[node] is not None:
            print(f"{parent[node]} -> {node}")

def main():
    root_node = input("Enter the root node: ")
    goal_node = input("Enter the goal node: ")

    print("Following is the A* Search")
    path = a_star_search(graph, root_node, goal_node, heuristic)

    if path:
        print("Path from root to goal:", path)
    else:
        print("No path found from root to goal.")

if __name__ == "__main__":
    main()

```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SEARCH ERROR

PS C:\Users\sabik\OneDrive\Desktop\PyChapter1> python -u "c:\Users\sabik\OneDrive\Desktop\PyChapter1\AI.py\Asearch.py"
Enter the root node: A
Enter the goal node: G
Following is the A* Search
Path from root to goal: ['A', 'E', 'F', 'G']
```

Compiler: VSC

Language: Python