



**SUBMITTED TO:** MR. MUBASHIR IQBAL

**SUBMITTED BY:** AHMED SALEEM RANA

**REG ID:** 24-CYS-023

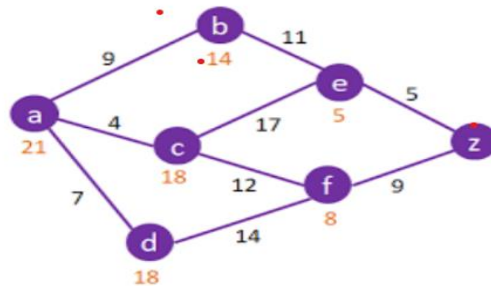
**SECTION:** B

**HITEC UNIVERSITY TAXILA, CANTT**

## TASK 01

### PART A:

- a) Apply the Best First Search and Beam Search algorithm on the graph below. Root Node is (a) and goal node is (z).



### BFS - CODE:

```

1  D# ----- Graph definition -----
2  graph = {
3      'a': ['b', 'c', 'd'],
4      'b': ['a', 'e'],
5      'c': ['a', 'e', 'f'],
6      'd': ['a', 'f'],
7      'e': ['b', 'c', 'z'],
8      'f': ['c', 'd', 'z'],
9      'z': []
10 }
11
12 # ----- Heuristic values from the diagram -----
13 h = {
14     'a': 21,
15     'b': 14,
16     'c': 18,
17     'd': 18,
18     'e': 5,
19     'f': 8,
20     'z': 0
21 }
22
23 # ----- Best First Search Algorithm -----
24 def best_first_search(start, goal, graph, h):
25     from heapq import heappush, heappop
26
27     open_list = []
28     heappush(open_list, (h[start], start))
29     visited = set()
30
31     parent = {start: None}

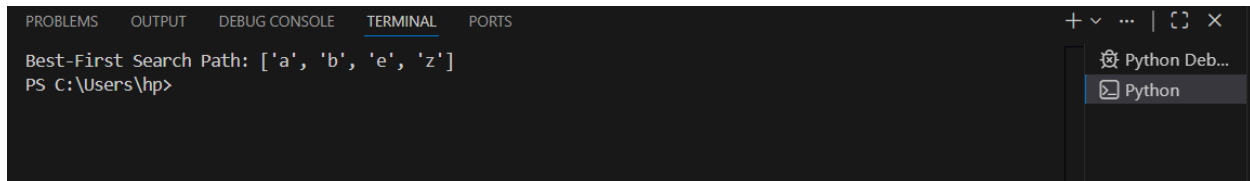
```

```

32
33     while open_list:
34         _, current = heappop(open_list)
35
36         if current == goal:
37             # reconstruct path
38             path = []
39             while current:
40                 path.append(current)
41                 current = parent[current]
42             return list(reversed(path))
43
44         visited.add(current)
45
46         for neighbor in graph[current]:
47             if neighbor not in visited:
48                 if neighbor not in parent: # prevent overwriting parents
49                     parent[neighbor] = current
50                 heappush(open_list, (h[neighbor], neighbor))
51
52     return None
53
54 # ----- Run Best First Search -----
55 path = best_first_search('a', 'z', graph, h)
56 print("Best-First Search Path:", path)

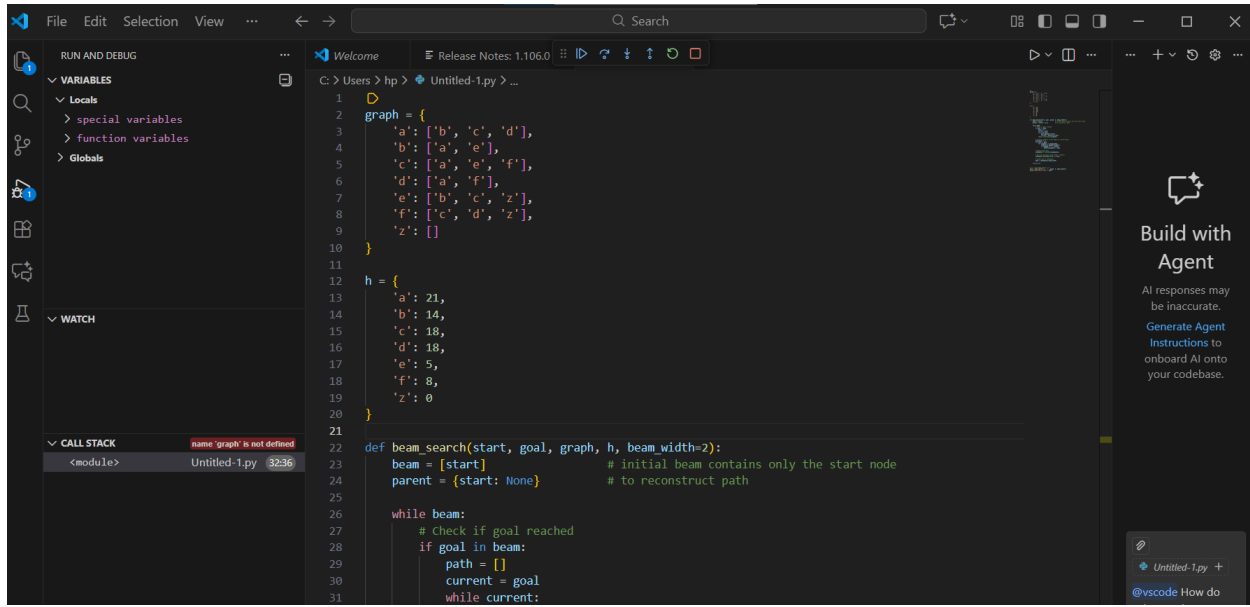
```

## OUTPUT:

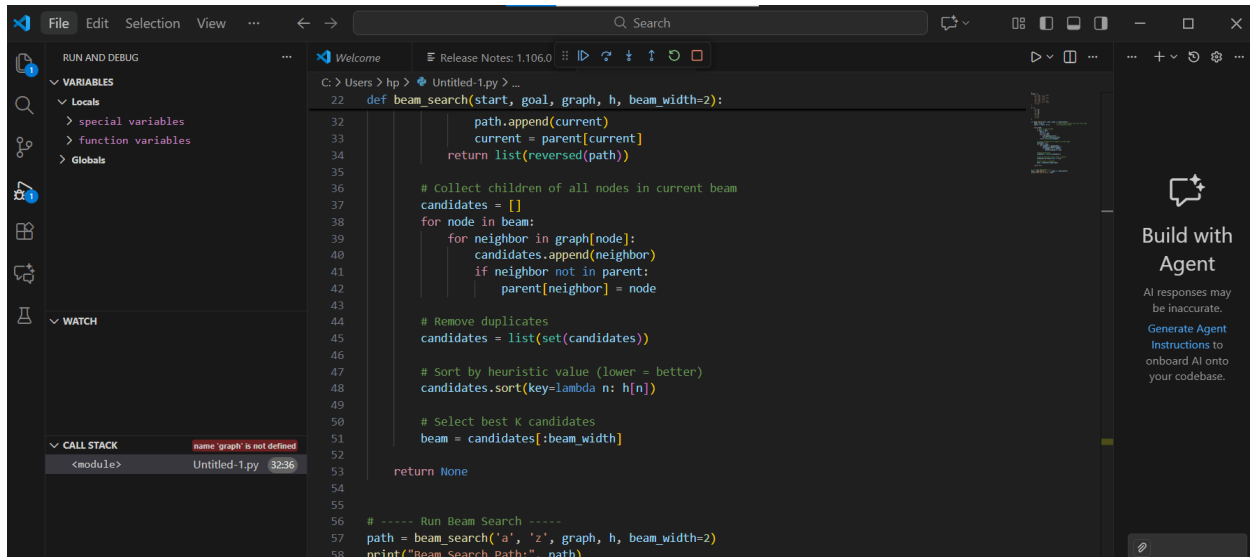


```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Best-First Search Path: ['a', 'b', 'e', 'z']
PS C:\Users\hp>
```

## BEAM SEARCH ALGORITHM CODE:



```
File Edit Selection View ... Search
Welcome Release Notes: 1.106.0
C:\Users\hp> Untitled-1.py > ...
1 graph = {
2     'a': ['b', 'c', 'd'],
3     'b': ['a', 'e'],
4     'c': ['a', 'e', 'f'],
5     'd': ['a', 'f'],
6     'e': ['b', 'c', 'z'],
7     'f': ['c', 'd', 'z'],
8     'z': []
9 }
10
11
12 h = {
13     'a': 21,
14     'b': 14,
15     'c': 18,
16     'd': 18,
17     'e': 5,
18     'f': 8,
19     'z': 0
20 }
21
22 def beam_search(start, goal, graph, h, beam_width=2):
23     beam = [start] # initial beam contains only the start node
24     parent = {start: None} # to reconstruct path
25
26     while beam:
27         # Check if goal reached
28         if goal in beam:
29             path = []
30             current = goal
31             while current:
```



```
22 def beam_search(start, goal, graph, h, beam_width=2):
23     beam = [start] # initial beam contains only the start node
24     parent = {start: None} # to reconstruct path
25
26     while beam:
27         # Check if goal reached
28         if goal in beam:
29             path = []
30             current = goal
31             while current:
32                 path.append(current)
33                 current = parent[current]
34             return list(reversed(path))
35
36         # Collect children of all nodes in current beam
37         candidates = []
38         for node in beam:
39             for neighbor in graph[node]:
40                 candidates.append(neighbor)
41                 if neighbor not in parent:
42                     parent[neighbor] = node
43
44         # Remove duplicates
45         candidates = list(set(candidates))
46
47         # Sort by heuristic value (lower = better)
48         candidates.sort(key=lambda n: h[n])
49
50         # Select best K candidates
51         beam = candidates[:beam_width]
52
53     return None
54
55
56 # ----- Run Beam Search -----
57 path = beam_search('a', 'z', graph, h, beam_width=2)
58 print("Beam Search Path:", path)
```

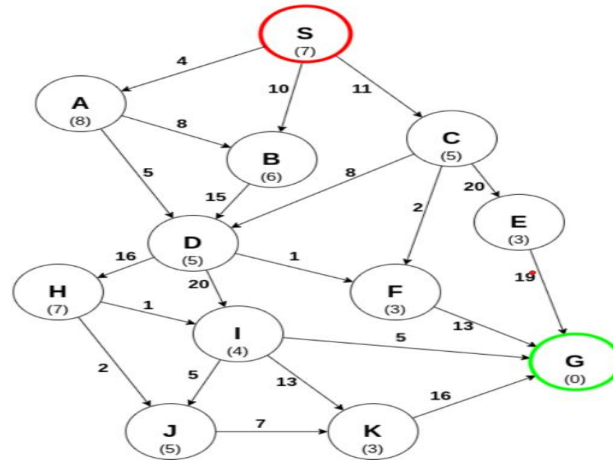
## OUTPUT:



```
Python Deb...
Python
Beam Search Path: ['a', 'b', 'e', 'z']
PS C:\Users\hp>
```

## PART B

- b) Apply Best First Search, Beam Search algorithm and A\* on the graph below.



## BFS – CODE:

```

1  D
2  graph = {
3      'S': ['A', 'B', 'C'],
4      'A': ['D'],
5      'B': ['D'],
6      'C': ['B', 'E', 'F'],
7      'D': ['H', 'I', 'F'],
8      'E': ['G'],
9      'F': ['G', 'K'],
10     'H': ['I', 'J'],
11     'I': ['J', 'G'],
12     'J': ['K'],
13     'K': ['G'],
14     'G': []
15 }
16 h = {
17     'S': 7,
18     'A': 8,
19     'B': 6,
20     'C': 5,
21     'D': 5,
22     'E': 3,
23     'F': 3,
24     'G': 0,
25     'H': 7,
26     'I': 4,
27     'J': 5,
28     'K': 3
29 }
30
31 def best_first_search(start, goal, graph, h):
32     open_list = []
33     closed_list = []
34     open_list.append(start)
35     while open_list:
36         current = min(open_list, key=lambda node: h[node])
37         closed_list.append(current)
38         for neighbor in graph[current]:
39             if neighbor not in closed_list:
40                 open_list.append(neighbor)
41     return goal

```

```

File Edit Selection View ... Search
C:\Users\hp> Untitled-1.py ...
31 def best_first_search(start, goal, graph, h):
32     heappush(open_list, (h[start], start))
33     visited = set()
34     parent = {start: None}
35
36     while open_list:
37         _, current = heappop(open_list)
38
39         if current == goal:
40             # Reconstruct path
41             path = []
42             while current:
43                 path.append(current)
44                 current = parent[current]
45             return list(reversed(path))
46
47         visited.add(current)
48
49         for neighbor in graph[current]:
50             if neighbor not in visited:
51                 if neighbor not in parent:
52                     parent[neighbor] = current
53                     heappush(open_list, (h[neighbor], neighbor))
54
55     return None
56
57 # ----- Run the search -----
58 path = best_first_search('S', 'G', graph, h)
59 print("Best-First Search Path:", path)
60

```

**Build with Agent**  
AI responses may be inaccurate.  
Generate Agent Instructions to onboard AI onto your codebase.

## OUTPUT:

```

PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL PORTS
Best-First Search Path: ['S', 'C', 'E', 'G']
PS C:\Users\hp>

```

**Python Deb...**  
**Python**

## DEEP SEARCH ALGORITHM CODE:

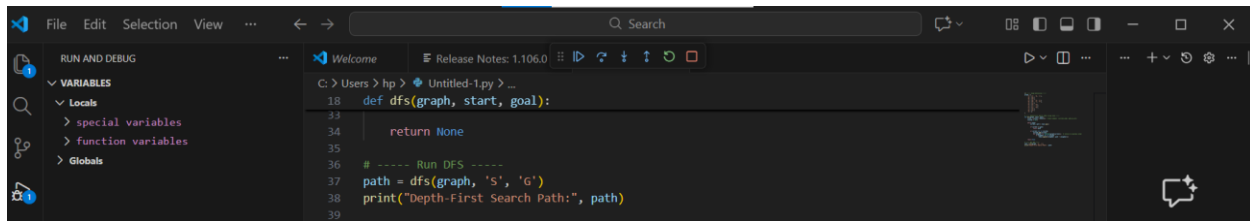
```

File Edit Selection View ... Search
C:\Users\hp> Untitled-1.py ...
1  D# ----- Graph definition -----
2  graph = {
3      'S': ['A', 'B', 'C'],
4      'A': ['D'],
5      'B': ['D'],
6      'C': ['B', 'E', 'F'],
7      'D': ['H', 'I', 'F'],
8      'E': ['G'],
9      'F': ['G', 'K'],
10     'H': ['I', 'J'],
11     'I': ['J', 'G'],
12     'J': ['K'],
13     'K': ['G'],
14     'G': []
15 }
16
17 # ----- Depth-First Search (DFS) using stack -----
18 def dfs(graph, start, goal):
19     stack = [(start, [start])] # Each element: (current_node, path_so_far)
20     visited = set()
21
22     while stack:
23         current, path = stack.pop()
24
25         if current == goal:
26             return path
27
28         if current not in visited:
29             visited.add(current)
30             for neighbor in reversed(graph[current]): # reverse to maintain order
31                 if neighbor not in visited:
32                     stack.append((neighbor, path + [neighbor]))
33

```

**Build with Agent**  
AI responses may be inaccurate.  
Generate Agent Instructions to onboard AI onto your codebase.

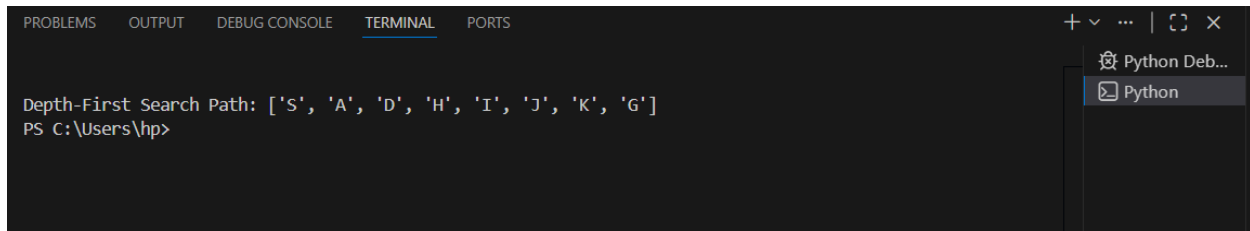
Untitled-1.py +  
@vscode How do I change the theme to light



The screenshot shows a code editor with a dark theme. On the left, there is a 'RUN AND DEBUG' sidebar with a 'VARIABLES' section. The main editor area displays a Python script for a Depth-First Search (DFS) function. The code is as follows:

```
18 def dfs(graph, start, goal):
19     # Base case: if start is goal, return the path
20     if start == goal:
21         return [start]
22     # Recursion: explore neighbors
23     for neighbor in graph[start]:
24         # Recursive call to find path from neighbor to goal
25         path = dfs(graph, neighbor, goal)
26         # If a path is found, prepend the current node and return
27         if path:
28             return [start] + path
29     # If no path is found, return None
30     return None
31
32 # Example usage
33 graph = {
34     'S': ['A', 'D', 'H'],
35     'A': ['D', 'I'],
36     'D': ['H', 'I', 'J'],
37     'H': ['I', 'K'],
38     'I': ['K', 'G'],
39     'J': ['K'],
40     'K': ['G']
41 }
42 start = 'S'
43 goal = 'G'
44 path = dfs(graph, start, goal)
45 print("Depth-First Search Path:", path)
```

## OUTPUT:



The screenshot shows a terminal window with a dark theme. The terminal output is as follows:

```
Depth-First Search Path: ['S', 'A', 'D', 'H', 'I', 'J', 'K', 'G']
PS C:\Users\hp>
```

## CONCLUSION:

We have applied several search algorithms on two different graphs to identify paths from the start to the goal. In the first graph, **Best-First Search (DFS)** algorithm is used which relies on heuristics to prioritize nodes closer to the goal, while **Beam Search** algorithm restricted the number of nodes explored at each level to find a path efficiently. In the second graph, we implemented **Depth-First Search (DFS)**, which traversed the graph deeply before backtracking, successfully reaching the goal node 'S' to 'G'.