**Project Based Learning Report**

On

**Implementation of Best First Search Algorithm in Python**

Submitted in the partial fulfillment of the requirements.

For the Project based learning in (**ARTIFICIAL INTELLIGENCE AND DATA MINING**)

In

Electronics & Communication Engineering

By

**2014111036 Aashi Chouksey (05)**

**2014111052 Mahek Gurjar (15)**

**2014111048 Siya Gupta (68)**

Under the guidance of Course In-charge

Dr. Tanuja S. Dhope

Department of Electronics & Communication Engineering

Bharati Vidyapeeth

(Deemed to be University)

College of Engineering,

Pune – 4110043

**Academic Year: 2023-24**

Bharati Vidyapeeth (Deemed to be University)

College of Engineering Pune-411046

**DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING**

## CERTIFICATE

**This is to be Certified that the Project Based Learning report entitled, “Implementation of Best First Search Algorithm in Python” is done by.**

**2014111036 Aashi Chouksey**

**2014111052 Mahek Gurjar**

**2014111048 Siya Gupta**

**In partial fulfillment of the requirements for the award of credits for Project Based Learning (PBL) in Integrated circuits and amplifier design of Bachelor of Technology Semester VII, Electronics & Communication Engineering.**

### Date:

**Dr. Tanuja S. Dhope Dr. A. A. Shinde**

**Course In-charge Head of Dept. ECE**

**INDEX**

|  |  |  |
| --- | --- | --- |
| **SERIAL NO.** | **CONTENTS** | **PAGE NO.** |
| 1 | Problem Statement | 4-4 |
| 2 | Best First Search Algorithm | 5-10 |
| 3 | Software Used | 11-11 |
| 5 | Result and Analysis | 12-17 |
| 6 | Conclusion | 18-18 |
| 7 | Appendix | 19-19 |

**PROBLEM STATEMENT**

**Implementation of Best First Search Algorithm in Python**

**CHAPTER -1**

**Best-First Search Algorithm**

The best [First Search algorithm in artificial intelligence](https://www.mygreatlearning.com/academy/learn-for-free/courses/best-first-search-in-artificial-intelligence?gl_blog_id=10837) is used for finding the shortest path from a given starting node to a goal node in a graph. The algorithm works by expanding the nodes of the graph in order to increase the distance from the starting node until the goal node is reached.

## **What is** **Best First Search?**

If we consider searching as a form of traversal in a graph, an uninformed search algorithm would blindly traverse to the next node in a given manner without considering the cost associated with that step. An informed search, like BFS, on the other hand, would use an evaluation function to decide which among the various available nodes is the most promising (or ‘BEST’) before traversing to that node.

BFS uses the concept of a Priority queue and heuristic search. To search the graph space, the BFS method uses two lists for tracking the traversal. An ‘Open’ list that keeps track of the current ‘immediate’ nodes available for traversal and a ‘CLOSED’ list that keeps track of the nodes already traversed.

## **Best First Search Algorithm**

1. Create 2 empty lists: OPEN and CLOSED
2. Start from the initial node (say N) and put it in the ‘ordered’ OPEN list
3. Repeat the next steps until the GOAL node is reached
   1. If the OPEN list is empty, then EXIT the loop returning ‘False’
   2. Select the first/top node (say N) in the OPEN list and move it to the CLOSED list. Also, capture the information of the parent node
   3. If N is a GOAL node, then move the node to the Closed list and exit the loop returning ‘True’. The solution can be found by backtracking the path
   4. If N is not the GOAL node, expand node N to generate the ‘immediate’ next nodes linked to node N and add all those to the OPEN list
   5. Reorder the nodes in the OPEN list in ascending order according to an evaluation function f(n)

This algorithm will traverse the shortest path first in the queue. The time complexity of the algorithm is given by O(n\*logn).

## **Variants of Best First Search**

The two variants of BFS are **Greedy Best First Search** and **A\* Best First Search**. Greedy BFS makes use of the Heuristic function and search and allows us to take advantage of both algorithms.

There are many ways to identify the ‘BEST’ node for traversal and accordingly there are various flavors of BFS algorithm with different heuristic evaluation functions f(n). We will cover the two most popular versions of the algorithm in this blog, namely Greedy Best First Search and [A\* Best First Search](https://www.mygreatlearning.com/blog/a-search-algorithm-in-artificial-intelligence/).

Let us say we want to drive from city S to city E in the shortest possible road distance, and we want to do it in the fastest way, by exploring the least number of cities along the way, i.e., the least number of steps.

Whenever we arrive at an intermediate city, we get to know the air/flight distance from that city to our goal city E. This distance is an approximation of how close we are to the goal from a given node and is denoted by the

heuristic function h(n). This heuristic value is mentioned within each node. However, note that this is not always equal to the actual road distance, as the road may have many curves while moving up a hill, and more.Also, when we travel from one node to the other, we get to know the actual road distance between the current city and the immediate next city on the way which is mentioned over the paths in the given figure. The sum of the distance from the start city to each of these immediate next cities is denoted by the function g(n).At any point, the decision on which city to go to next is governed by our evaluation function. The city which gives the least value for this evaluation function will be explored first.

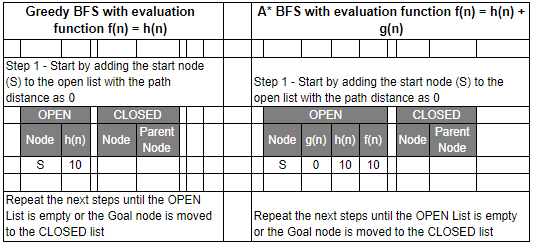
The only difference between Greedy BFS and A\* BFS is in the evaluation function. For Greedy BFS the evaluation function is f(n) = h(n) while for A\* the evaluation function is f(n) = g(n) + h(n).

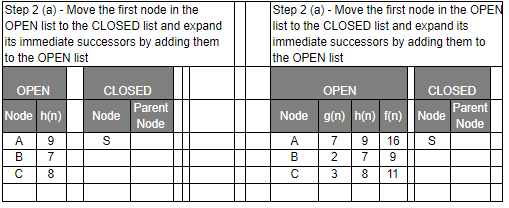
Essentially, since A\* is the more optimal of the two approaches as it also takes into consideration the total distance travelled so far i.e., g(n).

## **Best First Search Algorithm Example**

## Let us have a look at the graph below and try to implement both Greedy BFS and A\* algorithms step by step using the two lists, OPEN and CLOSED.

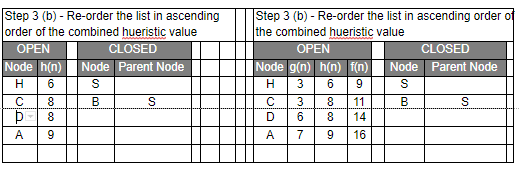
|  |  |
| --- | --- |
| g(n) | Path Distance |
| h(n) | Estimate to Goal |
| f(n) | Combined Hueristics i.e., g(n) + h(n) |

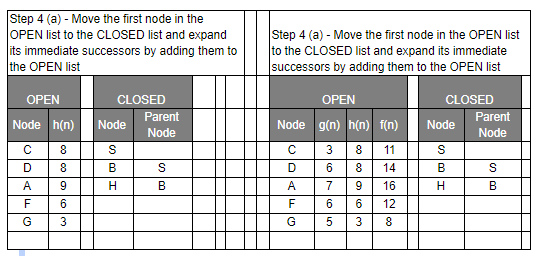


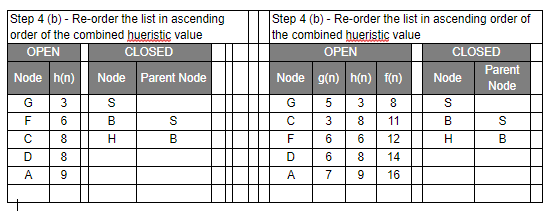


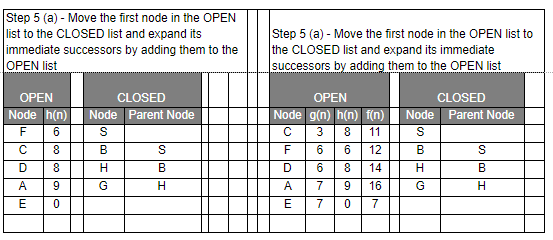
A close-up of a chart

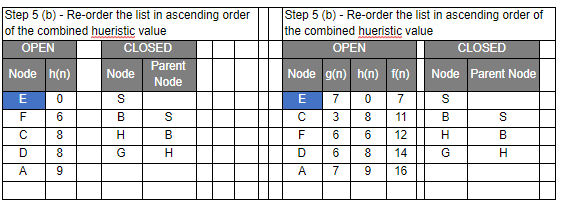
Description automatically generated

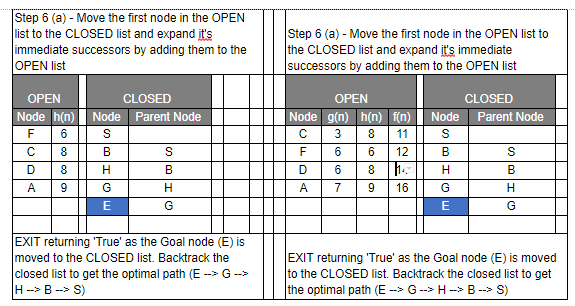












Even though you would find that both Greedy BFS and A\* algorithms find the path equally efficiently, a number of steps, you may notice that the A\* algorithm is able to produce is a more optimal path than Greedy BFS. So, in summary, both Greedy BFS and A\* are the Best first searches but Greedy BFS is neither complete nor optimal whereas A\* is both complete and optimal. However, A\* uses more memory than Greedy BFS, but it guarantees that the path found is optimal.

### **Advantages and Disadvantages of Best First Search**

**Advantages**:  
1. Can switch between BFS and DFS, thus gaining the advantages of both.  
2. More efficient when compared to DFS.

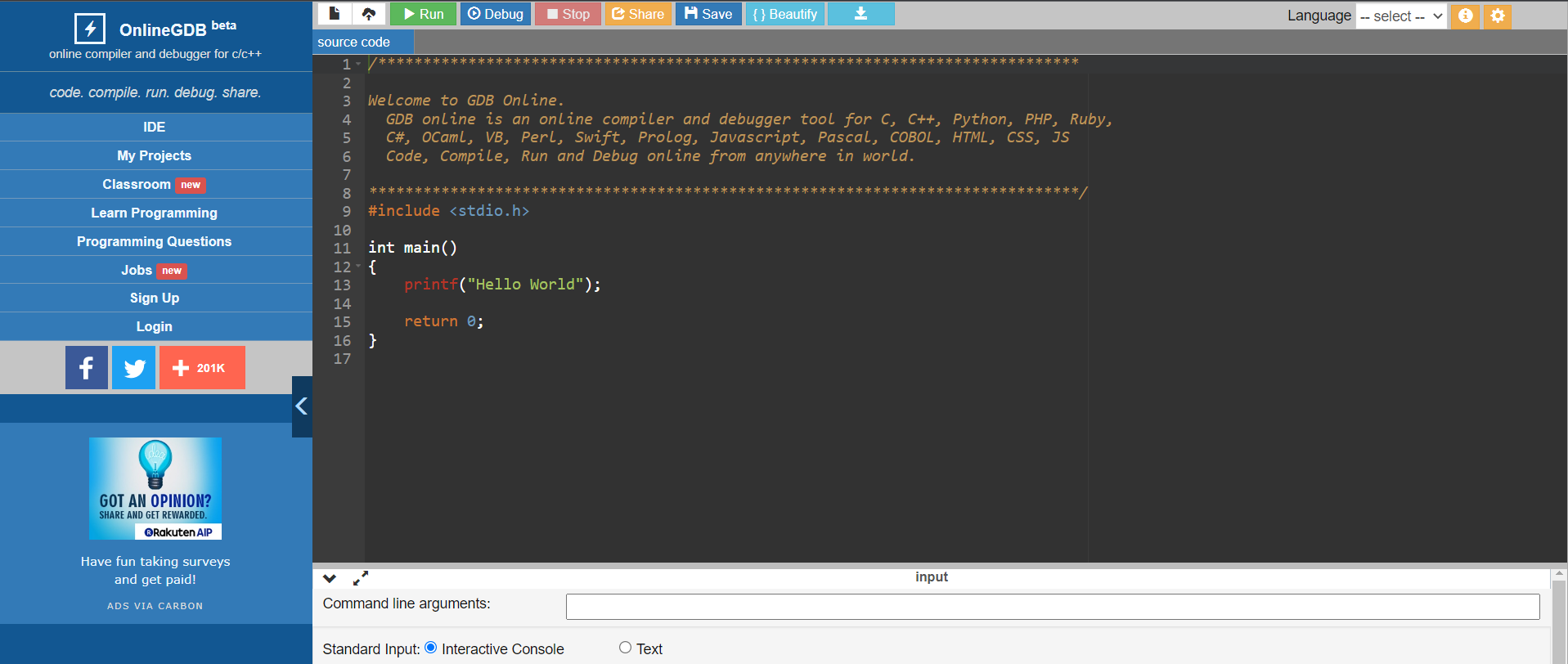
**Disadvantages:**  
1. Chances of getting stuck in a loop are higher.

**CHAPTER – 2**

**Software Used**

GDB

GDB stands for GNU Project Debugger and is a powerful debugging tool for C (along with other languages like C++). It helps you to poke around inside your C programs while they are executing and also allows you to see what exactly happens when your program crashes. GDB operates on executable files which are binary files produced by the compilation process.



**CHAPTER: -3**

**Result and Analysis:** -

We implemented Best First Search Algorithm and were able to successfully implement.

# A Node class for GBFS Pathfinding

class Node:

def \_\_init\_\_ (self, v, weight):

self.v=v

self.weight=weight

# pathNode class will help to store

# the path from src to dest.

class pathNode:

def \_\_init\_\_(self, node, parent):

self.node=node

self.parent=parent

# Function to add edge in the graph.

def addEdge(u, v, weight):

# Add edge u -> v with weight weight.

adj[u].append(Node(v, weight))

# Declaring the adjacency list

adj = []

# Greedy best first search algorithm function

def GBFS(h, V, src, dest):

"""

This function returns a list of

integers that denote the shortest

path found using the GBFS algorithm.

If no path exists from src to dest, we will return an empty list.

"""

# Initializing openList and closeList.

openList = []

closeList = []

# Inserting src in openList.

openList.append(pathNode(src, None))

# Iterating while the openList

# is not empty.

while (openList):

currentNode = openList[0]

currentIndex = 0

# Finding the node with the least 'h' value

for i in range(len(openList)):

if(h[openList[i].node] < h[currentNode.node]):

currentNode = openList[i]

currentIndex = i

# Removing the currentNode from

# the openList and adding it in

# the closeList.

openList.pop(currentIndex)

closeList.append(currentNode)

# If we have reached the destination node.

if(currentNode.node == dest):

# Initializing the 'path' list.

path = []

cur = currentNode

# Adding all the nodes in the

# path list through which we have

# reached to dest.

while(cur != None):

path.append(cur.node)

cur = cur.parent

# Reversing the path, because

# currently it denotes path

# from dest to src.

path.reverse()

return path

# Iterating over adjacents of 'currentNode'

# and adding them to openList if

# they are neither in openList or closeList.

for node in adj[currentNode.node]:

for x in openList:

if(x.node == node.v):

continue

for x in closeList:

if(x.node == node.v):

continue

openList.append(pathNode(node.v, currentNode))

return []

# Driver Code

""" Making the following graph

src = 0

/ | \

/ | \

1 2 3

/\ | /\

/ \ | / \

4 5 6 7 8

/

/

dest = 9

"""

# The total number of vertices.

V = 10

## Initializing the adjacency list

for i in range(V):

adj.append([])

addEdge(0, 1, 2)

addEdge(0, 2, 1)

addEdge(0, 3, 10)

addEdge(1, 4, 3)

addEdge(1, 5, 2)

addEdge(2, 6, 9)

addEdge(3, 7, 5)

addEdge(3, 8, 2)

addEdge(7, 9, 5)

# Defining the heuristic values for each node.

h = [20, 22, 21, 10, 25, 24, 30, 5, 12, 0]

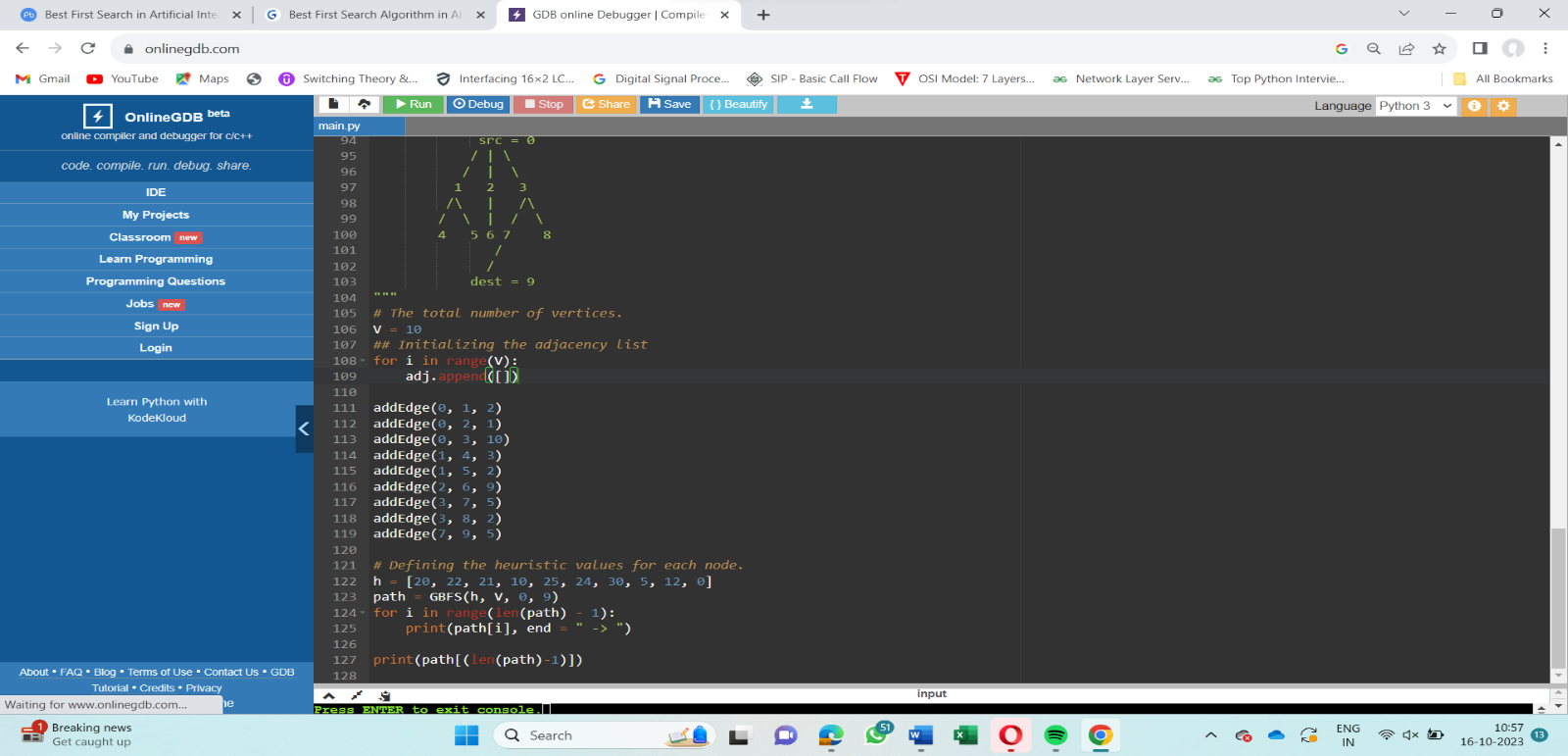
path = GBFS(h, V, 0, 9)

for i in range(len(path) - 1):

print(path[i], end = " -> ")

print(path[(len(path)-1)])

**OUTPUT**

****

**A screenshot of a computer

Description automatically generated**

**CHAPTER: -4**

**Conclusion & Outcome:-**

The project “ Implementation of the Best First Search Algorithm in Python” has helped us to understand the concept of Best First Search Algorithm, its advantages and applications .

The Best First Search algorithm plays a crucial role in various domains of artificial intelligence, including gaming, puzzle-solving, route optimization, action planning, robotics, and more. By incorporating heuristic evaluation and a priority queue, BFS efficiently finds the shortest path from an initial state to a goal node in a graph. While it has advantages such as efficiency and, it also has limitations, such as the potential for loop-related issues.

Hence, Course Outcome 2(Design and Analyse Search Techniques and Game Playing Technique) is achieved .

**CHAPTER-5**

Appendix

https://github.com/igsiya/AIDM/commit/95c49dca82be244ba3d28d67804964b51e8a2209