Experiment No :-1

Practical Aim :- Implement Depth first search algorithm and Breadth first search algorithm use an Undirected graph and develop a recursive algorithm for searching all the vertices of a graph or tree data structure.

Aim :- To implement Depth first search (DFS) and Breadth first search (BFS) algorithm using an Undirected graph and develop a recursive algorithm to traverse all vertices of a graph or tree data structures.

Objectives

1. To understand and implement graph traversal technique.
2. To differentiate Both DFS & BFS approaches.
3. To write Recursive and Interative solution for traversal
4. To explore all vertical in a connected Undirected graph.

Page No.
Date

Software Requirment

- Anaconda with python 3.7
- Python 3.6 Or above.
- Text Editor
- Cups.
- Experta Python package.

Hardware Requirement :-
PIV 2GB RAM, 500GB HDD
Processor :- Intel is Or above.
RAM - 4GB minimum
Hard disk :- 500MB free Space
Input / Output device - Keyboard
Monitor.

Theory :-
Graph Basics
- A graph is a set of vertices and edge.
- A graph can be Undirected Or directed.
- Graph and Commonly represented Using.

1. Adjacency Matrix
2. Adjacency List.

BFS (Breadth first search)
- Explores Vertices level by level
- Uses a queue (FIFO)
- Starts from a source node and explores its neighbours first.
- Useful for finding the shortest path ~~is~~ in unweighted graph.

DFS - (Depth first search)
- Explores vertices depth wise before Backtracking
- Uses Recursion Or a Stack
- Traverses a path fully Before moving to next path
- Useful in topological sorting, cycle detection and maze solving.

Algorithm.
DFS ~~Algorithm~~ Algorithm (Recursive)
DFS(G, v visited)
mark 'v' as visited.
for all neighbours of v in G
if u is not visited
DFS (G, u, visited).

AIL

Page No. 1

Date

Dr.D.Y.PATIL EDUCATIONAL FEDERATION Varale Campus

Lab experiment No:-2

Practical Title:-

Implementation of A* search algorithm for solving the 8-puzzle game

AIM:-

To implement the A* search algorithm for solving the 8-puzzle game and demonstrate its efficiency in finding the optimal solution path.

* objective

① To understand the working principle of the A* search algorithm.

② To apply A* to a game search problem (8-puzzle)

③ To compare A* to a game search algorithm ~~pro~~ of possible moves using

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

④ To produce the shortest sequence of mouse from the initial state of the goal state.

⑤ To produce the shortest sequence of mouse from the initial state of the goal state.

* Software requirement

① Operating system window

② Programming language : python

③ IDE : ~~python~~ pycharm / VS code

* Hardware requirement

① Processor : latest or above

② RAM 4GB minimum

③ input devices : keyboard

④ Output devices : Monitor

* Formula

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

g(n):- Cost from start node to the current

h(n) :- estimate cost from current node to the goal.

f(n) :- estimate total cost from start to goal via node.

For the 8-puzzle problem, the game consists of 3×3 board containing file no. 18 and one blank space the goal is to reach the target arrangement by sliding files.

Common heuristic.

① Misplaced :- Count of title not in the Position.

② Manhattan distance:- Sum of the distance of file from there correct position.

* Algorithm

① Start with an initial board state.

② Initialize priority queue with start node

③ While queue is not empty,

1] Remove the node with lowest f(n)

2] If the node is the goal state return the path.

3] Generate all valid neighbour

4] For each neighbour

- Calculate g(n) & h(n)
- path into the priority queue if not already visited.

5] If no solution is found return .

Conclusion :-

The A* algorithm successfully finds the shortest path for solving the 8 puzzle game ~~conclu~~ combing the 8 puzzle game the actual cost and heuristic estimates. It is efficient for problem with well defined heavily on the accuring of the heuristic function.

AIL

Page No. 1
Date

Dr.D.Y.PATIL EDUCATIONAL FEDERATION Varale Campus

Lab experiment : 03

Title :-

Solving the N-Queue problems using constraint satisfaction with back tracking and Branch & Bounce.

AIM :-

To implement and compare the back tracking and branch & Bound approches for solving the N-Queens problem as a constraint satisfaction problem (CSP)

Objectives

① To understand the concept of constraints satisfaction problem in Artificial ~~inteli~~ intelligence.

② To solve the N-Quee problems using the back tracting method.

Page No.
Date

③ To optimize the solutio and using the branch and bound approach.

④ To compare performanc in term of search spa and execution time.

⑤ To demonstrate the efficietive of connecting in reducing uneessary computetion.

* Software requirement

- Operating System

Windows / linux / program banguage : python 3X

I-DE : Pycharm / Vs code / Jupiter Notebook.

Libraries: No external lib ries required.

* Hardware Requirement

① Processor :- i5 10th core 18 or higher.

② RAM :- 4GB Or more.

③ Storage :- 500 MB free space.

④ Input Devices :- keyboard.

⑤ Output devices :- Monitor.

* Theory

The N-queens problem is a classic constraint satisfaction problem in which we place N queens on NXN chess board so that

No two queens share the same row

No two queens share the same column.

Page No.
Date

Back Tracking approach.

A depth first search meth
Places queen one by one
different colums.
If conflict is found back
track to the previous tre
another position.

Branch and Bound approach.

Improves over back trackin
by more
Mount uins theme arrays.
- Columns[]: marks occupied
- left diagonal[]: marks ouipe
left diagonal.
- Right diagonal[]:- marks occuined
diagonals.

*Algorithm

① Start from left most column
② Place a queen in a row
when it not attached.
③ If safe place the queen
& move to the next
column.

④ If placing leads to a conflicts, remove the queen and try next row

⑤ Repeat until all queens one placed or no valid placement exists.

* Branch and Bound.

1] Initialize column[], left diagonal[] right diagonal[] one false.

2] Straight from column 0.

3] For each row in the column 0 and right diagonal.

4] Back track by marking them false if no further placement possible.

Page No.
Date

Conclusion

Both Backtracking & Branch Bound can solve the N problem but branch bound generally faster because it prunes ~~infeed~~ solutions e alier redi· the search space this ~~almost~~ demons the aduantages of combinin with optimization Strategies AI

Timely Submission	Perfor-mance	Understanding

~~Sr~~ Regularity	Sign

Page No.
Date

Dr.D.Y.PATIL EDUCATIONAL FEDERATION Varale Campus

Aim :- Implement Greedy search algorithm for any of the following application.

1. Selection Sort
2. Minimum Spanning tree
3. Single Scheduling problem
4. Job - Source Shortest path problem.
5. prim's minimal Spanning tree algorithm.
6. ~~pri~~ Kruskal's minimum Spanning tree

We implement :- Dijkstra's Algorithm

Objective

- To Study & implement the Greedy search strategy.
- To understand how local Optimal Choice can lead to a globally Optimal solution

- Software Requirement

programming Language / python / C++ / Java

IDE - pycharm / Vs code / Turbo C++ / Eclipse

Compiler : GCC for C++ python 3.x

IDK for java

Operating System : Window / Linux

Page No.
Date

Hardware Requirement: Intel is
Or above
- RAM 4 GB minimum
- Hardwork: 250 MB free space
- I/O Device Keyboard, Monitor
, Mouse.

Theory:-

The Greedy search algorithm
works by making the locally
Optimal choice of each step with t
hope of finding a global optima
Greedy algorithms build up solutio
piece by piece, always choosing
the next piece that offers the most
immediate benefit.
They are simple fast & well work wel
for optimization problem

Examples:- Dijkstra's Algorithm
It finds the shortest path from
a single source node to all other
nodes in a weighted path.

At each steps it picks the nodes
with the minimum distance.

Page No.
Date

Algorithm

Steps:

1. Initialize distance of all vertices as infinity & distance of source as a.
2. Mark all vertices unvisited
3. Select the unvisited vertex with the smallest distance
4. Update the distance value of its neighbouring vertices.
5. Mark the current vertex as visited
6. Repeat steps 3-5 until all vertices are visited.
7. Display the shortest path distance from the source.

Conclusion :- Greedy algorithm are useful for solving Optimization problem efficiently Implementing Dijkstra's algorithm using the greedy approach helps us understand & how local decision can lead to global Optimization.

AIL

Page No. 1
Date

Dr.D.Y.PATIL EDUCATIONAL FEDERATION Varale Campus

NAAC

Experiment NO:05

* Problem Title:
Develop an elementary chatbot Powr to mor interaction

* Objective
- To design by implement the Simple chatbot per customer interaction.
- To understand the working of rule based retrival - based chatbots
- To demonstrate natural language interaction between a user a the system.

* software Requirement
Programming language
IDE - Pycharm luscode / Jupiter
Library : NLTK, BEOr simple conditional statement.

OS: windows/linux

Hardwore Requirement:
Program: Intel i3 orabove
RAM : 4GB minimum
Hord disk: 200mB free space.
I/IO device : Keyboard monitor.

Page No.
Date

Theory
- A chatbot is an AI. boa
program that simulates hum
conversion
- It can interact with user
through test
- Two main types
Rule- based & chatbot
Predefined processing a ml to
understood user queries

In this experiment are develop
a rule based chatbot matches
uses queries with - predefined
responses.
- If the I/P do not match
it provides a default I/don't
understand type response.
- If the I/P responses

Tws approch is simple fi
suitable for dementary appli

Algorithm :-

1. Start the program
2. Define a set of predefine
question By Their response

3. Accept I/P from the user.
4. Compare user I/P with stored keyboards using string matching
5. If a match is form, display the corrospending response
6. If no match is found display a default message.
7. Continue the conversion until the user types "exist"
8. End the program

Conclusion:
An elementry chatbot can similar simple human-like convenion using rule-based responses This experiment gives a foundation for building advanced chatbot using NLP & ML

Timely submitted	Perfor-mance	under-standing	
Viva	regularity	Total	sign