Experiment - 05

Aim: Implement greedy BFS and A * search ~~techniq~~ technique in any programming language.

Theory:

* Informed search techniques:

It refers to search algorithms which help in navigating large databases with certain available information about the end goal in search and most widely used in large databases where uninformed search algorithms can't accurately create precise result.

There are different types of informed search strategies:-

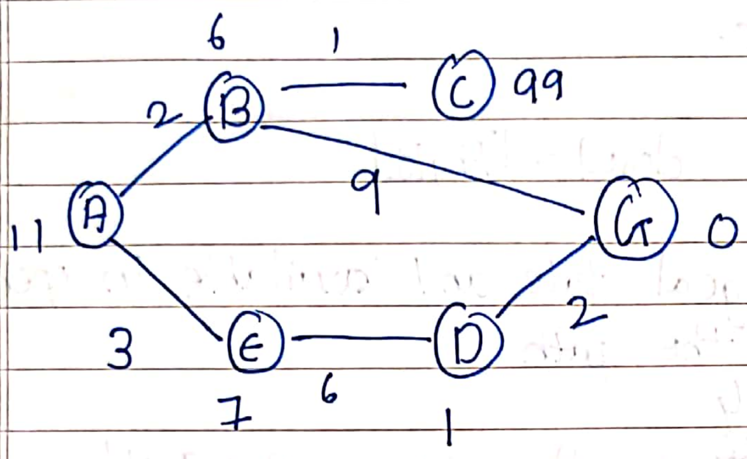
1] Pure Heuristic search.
2] Best first or greedy search.
3] A* tree search
4] A* graph search.

* Compare Best first search and A* search algorithm

Best first search	A* search
- This algorithm visits the next state based on heuristics function f(n) = h with lowest heuristic value	- This algorithm visits the next state based on heuristics fcn) = h+g where 'h' is cost of travel from particular state & 'g' is cost from start to goal state.

- It doesn't consider the cost of the path to that particular state. All it takes about is that which next state from the current state has lower heuristic	- It doesnt choose the next state only with the lowest heuristic value ~~bet~~ but it selects the one that gives the lowest value when considering its heuristic cost of getting to the state.
Time complexity = $O(b^d)$ Space complexity = $O(b^d)$	- Time complexity: $O(bd)$ Space complexity: $O(b^d)$

Sourabh Besa D12C 08

a] 

Find optimal path from A–G using
a] BFS
b] Greedy BFS
c] A* search

Ans: a] Best first Search
initially open = [ ] , closed [ ]
A → open [A] , closed [ ]
A hugs E & B as neighbours so,
open = [B, E] closed = [A]

node (n)	A	B	C	D	E	G
h(n)	11	6	99	1	7	0
g(n)	0	2	3	9	3	11

heuristic function for bfs ⇒ f(n) = g(n)

Since,
B is least cost path so we traverse to B and add it to closed set.

i.e. open = [E] closed = [A, B]

e. since G is goal state and available in open ~~list~~
∴ we consider the path
$A \rightarrow B \rightarrow G$
and cost of a path is = 2 + 9 = 11

b) Greedy Best first Search
f(n) = h(n) → heuristic function for ~~greedy~~ greedy Best first ~~t~~ search.

- Add initial state A to open list.
open = [A] closed = []
- Add neighbors of A i.e B & E to open list
open = [B, E] closed = [A]
f(B) = h(B) = 6
f(E) = h(E) = 7
∴ f(B) < f(E) so, ~~o~~ true traverse B.

open = [C, G, E] closed = [A, B]

f(C) = 99
f(G) = 0
f(E) = 7
∵ f(G) is minimum so we choose this path
∴ The final optimal path is A - B - G cost = 2+9 = 11

c] A* search algorithm.

heuristic function

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

for open = [A] closed = []

A,

Neighbours of A − B, E

open = [B, E], closed = [A]

$f(B) = g(B) + h(B) = 7+6 = 15$

$f(E) = g(E) + h(E) = 7+7 = 14$

∵ f(B) > f(E) ∴ We add to closed list & traverse it & add its neighbour to open list.

open = [D, G, B]

closed = [A, ~~B~~ E]

$f(D) = 1+1 = 2$ $f(G) = 11+0$ $f(B) = 1$

∵ f(D) is minimum we add D to closed

open = [G, B] closed = [A, E, D]

$f(G) - 11$ $f(B) = 15$

∵ f(G) is minimum so we add G to closed list & we have reached goal state.

∴ ~~Fin~~ Final Path

A → E → D → G

~~cot~~ cost = 3+6+1 = 10 //

FOR EDUCATIONAL USE

Experiment 05

1. Best First Search

Code:

```
from queue import PriorityQueue
v = 14
graph = [[] for i in range(v)]

def bfs(source, target, n):
    visited = [False] * n
    pq = PriorityQueue()
    pq.put((0, source))
    while pq.empty() == False:
        u = pq.get()[1]
        print(u, end=" ")
        visited[u]=True
        if u == target:
            break

        for v, c in graph[u]:
            if visited[v] == False:
                visited[v] = True
                pq.put((c, v))
    print()


def addedge(x, y, cost):
    graph[x].append((y, cost))
    graph[y].append((x, cost))

addedge(0, 1, 3)
addedge(0, 2, 6)
addedge(0, 3, 5)
addedge(1, 4, 9)
addedge(1, 5, 8)
addedge(2, 6, 12)
addedge(2, 7, 14)
addedge(3, 8, 7)
addedge(8, 9, 5)
addedge(8, 10, 6)
addedge(9, 11, 1)
addedge(9, 12, 10)
addedge(9, 13, 2)

source = 0
```

```
target = 9
print(f"Source: {source} , Destination: {target}")
print("The Path is as follows")
bfs(source, target, v)
```

OUTPUT:

```
Source: 0 , Destination: 9
The Path is as follows
0 1 3 2 8 9
```

2. A* Search

Code:

```
from collections import deque

class Graph:
    def___init__(self, adjacency_list):
        self.adjacency_list = adjacency_list

    def get_neighbors(self, v):
        return self.adjacency_list[v]

    def h(self, n):
        H = {
            'A': 1,
            'B': 4,
            'C': 1,
            'D': 3,
            'E': 2
        }

        return H[n]

    def a_star_algorithm(self, start_node, stop_node):
        open_list = set([start_node])
        closed_list = set([])

        g = {}

        g[start_node] = 0
        parents = {}
        parents[start_node] = start_node
```

```
        while len(open_list) > 0:
            n = None
            for v in open_list:
                if n == None or g[v] + self.h(v) < g[n] + self.h(n):
                    n = v;

            if n == None:
                print('Path does not exist!')
                return None

            if n == stop_node:
                reconst_path = []

                while parents[n] != n:
                    reconst_path.append(n)
                    n = parents[n]

                reconst_path.append(start_node)

                reconst_path.reverse()

                print('Path found: {}'.format(reconst_path))
                return reconst_path

            for (m, weight) in self.get_neighbors(n):
                if m not in open_list and m not in closed_list:
                    open_list.add(m)
                    parents[m] = n
                    g[m] = g[n] + weight

                else:
                    if g[m] > g[n] + weight:
                        g[m] = g[n] + weight
                        parents[m] = n

                        if m in closed_list:
                            closed_list.remove(m)
                            open_list.add(m)
            open_list.remove(n)
            closed_list.add(n)

        print('Path does not exist!')
        return None

if___name__=="__main__":
        adjacency_list = {
        'A': [('C', 10), ('B', 7)],
        'B': [('D', 5), ('C', 2)],
```

```
'C': [('E', 12)],
'D': [('E', 13)]
}
graph1 = Graph(adjacency_list)
graph1.a_star_algorithm('A', 'E')
```

OUTPUT:

```
Path found: ['A', 'B', 'C', 'E']
```

Conclusion:

In this experiment we learned about different informed search strategies and have compared them with each other.We also have sucessfully implemented best first search and A* search in python getting desired results from the program.