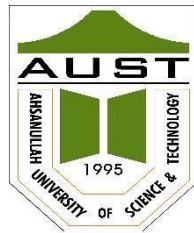


AHSANULLAH UNIVERSITY OF SCIENCE AND TECHNOLOGY
DHAKA-1208, BANGLADESH.



Department of Computer Science and Engineering
Spring 2019

Program: Bachelor of Science in Computer Science and Engineering

Course No: CSE 4108

Course Title: Artificial Intelligence Lab

Assignment No: 03

Date of Submission: 02/03/2019

Submitted to Dr. S.M. Abdullah Al-Mamun
Professor, Department of CSE, AUST.

Md. Siam Ansary
Adjunct Faculty, Department of CSE, AUST.

Submitted By

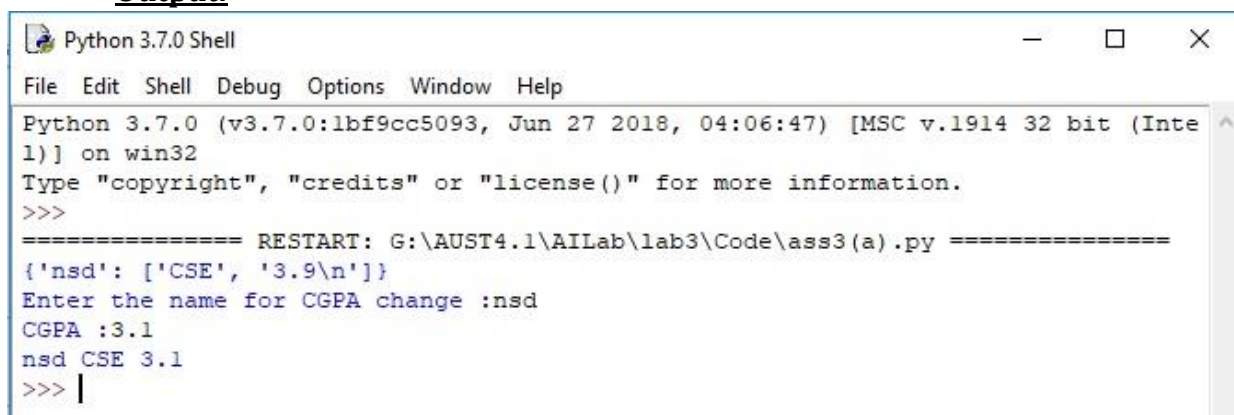
Name : Robiul Hasan Nowshad
Student ID : 16.01.04.061
Lab Group : B1

1. Write a Python program that reads the file created as demonstrated into a dictionary taking 'name' as the key and a list consisting of 'dept' and 'cgpa' as the value for each line. Make changes in some 'cgpa' and then write back the whole file.

Python program:

```
1. fn="te.txt"
2. Mydict={}
3. f1=open(fn, "r")
4. for l in f1:
5.     name, dept, cgpa =l.split("\t")
6.     Mydict[name]=[dept,cgpa]
7. f1.close
8. print(Mydict)
9. name=str(input("Enter the name for CGPA change :"))
10. dept, cgpa = Mydict[name]
11.
12. cgpa=str(input("CGPA :"))
13. Mydict[name]=[dept, cgpa]
14.
15. f1=open(fn, "w")
16. for x,y in Mydict.items():
17.     std=str(x)+"\t"+str(y[0])+"\t"+str(y[1])
18.     print(std, end="\n", file=f1)
19. f1.close
20.
21. f1=open(fn, "r")
22. for l in f1:
23.     name, dept, cgpa =l.split("\t")
24.     print(name, dept, float(cgpa), end="\n")
25.     f1.close
```

Output:



```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: G:\AUST4.1\AILab\lab3\Code\ass3(a).py =====
{'nsd': ['CSE', '3.9\n']}
Enter the name for CGPA change :nsd
CGPA :3.1
nsd CSE 3.1
>>> |
```

2. Implement in generic ways (as multi-modular and interactive systems) the Greedy Best-First and A* search algorithms in Prolog and in Python.

InputGraph.pl

```
1. :-module(inputGraph, [ neighbor/3, h_fn/2 ]).
2.
3. % Description of the weighted graph
4.
5. neighbor(i,a,35). neighbor(a,i,35). neighbor(i,b,45). neighbor(b,i,45).
6. neighbor(a,c,22). neighbor(c,a,22). neighbor(a,d,32). neighbor(d,a,32).
7. neighbor(b,d,28). neighbor(d,b,28). neighbor(b,e,36). neighbor(e,b,36).
8. neighbor(b,f,27). neighbor(f,b,27). neighbor(c,d,31). neighbor(d,c,31).
9. neighbor(c,g,47). neighbor(g,c,47). neighbor(d,g,30). neighbor(g,d,30).
10. neighbor(e,g,26). neighbor(g,e,26).
11.
12. % Description of heuristic function values
13. % Straight line distance from a node to the goal node, g
14.
15. h_fn(i,80). h_fn(a,55). h_fn(b,42). h_fn(c,34).
16. h_fn(d,25). h_fn(e,20). h_fn(f,17). h_fn(g,0).
```

Best-first Search:

```
1. % Including data files
2. :-use_module(inputGraph).
3.
4. % Declaration of dynamic data
5. :-dynamic(t_node/2).
6. :-dynamic(pq/1).
7. :-dynamic(pp/1).
8.
9. % Search begins
10. search:-write('Enter start node:'),read(S),h_fn(S,HV),
11.     assert(t_node(S, 'nil')),assert(pq([node(S,HV)])),
12.     assert(pp([])),generate,find_path_length(L), display_result(L).
13.
14. % Generating the solution
15. generate:-pq([H|_]),H=node(N,_),N=g, add_to_pp(g),!.
16. generate:-pq([H|_]),H=node(N,_),update_with(N), generate.
17.
18. % Adding a node to possible path
19. add_to_pp(N):-pp(Lst), append(Lst,[N],Lst1), retract(pp(_)),
20.     assert(pp(Lst1)).
21.
22. % Updating data according to selected node.
23. update_with(N):-update_pq_tr(N), update_pp(N).
24.
25. % Updating Priority Queue and Tree
26. update_pq_tr(N):-pq(Lst), delete_1st_element(Lst,Lst1), retract(pq(_)),
27.     assert(pq(Lst1)), add_children(N).
```

```

28.delete_1st_element(Lst,Lst1):-Lst = [_|Lst1].
29.add_children(N):- neighbor(N,X,_), not(t_node(X,_)),insrt_to_pq(X),
30.    assert(t_node(X,N)),fail.
31.add_children(_).
32.add_children(N,I):- neighbor(N,X,D), t_n_inde(I1), t_node(_,I,_,V),
33.    h_fn(N,V1), h_fn(X,V2), FNV is V+D-V1+V2,
34.    insrt_to_pq(X,I1,I,FNV), assert(t_node(X,I1,I,FNV)),
35.    incr_inde, fail.
36.add_children(_,_).
37.incr_inde:- t_n_inde(X), Y is X+1, retract(t_n_inde(X)), assert(t_n_inde(Y))
38.
39.% Inserting node to Priority Queue
40.insrt_to_pq(X):- pq(Lst), h_fn(X,V), insert12pq(node(X,V),Lst,Lst1),
41.    retract(pq(_)), assert(pq(Lst1)).
42.
43.insert12pq(E1,[], [E1]):-!.
44.insert12pq(E1, L1, L2):-L1=[H|_], E1=node(_,V1), H=node(_,V2),
45.    not(V1 > V2), L2 = [E1|L1], !.
46.insert12pq(E1, L1, L2):-L1=[H|T], insert12pq(E1, T, Lx), L2 = [H|Lx].
47.
48.% Updating Possible Path
49.update_pp(N):- retract(pp(_)), assert(pp([])), renew_pp(N).
50.renew_pp(N):-t_node(N,nil), pp(X), append([N],X,X1),
51.    retract(pp(_)), assert(pp(X1)), !.
52.renew_pp(N):- pp(X), append([N],X,X1), retract(pp(_)), assert(pp(X1)),
53.    t_node(N,N1), renew_pp(N1).
54.
55.% Finding 'shortest' path length
56.find_path_length(L):-pp(Lst),path_sum(Lst,L).
57.path_sum(Lst,0):- Lst=[g|_],!.
58.path_sum(Lst,L):-
59.    Lst=[N|T],T=[N1|_], neighbor(N,N1,D), path_sum(T,L1),L is L1+D.
60.% Displaying 'shortest' path and its length
61.display_result(L):- pp(Lst), write('Solution:'), write(Lst),nl,
62.    write('Length:'), write(L).
63.
64.% Arrange a menu of actions
65.start:- repeat,
66.    write('\n1. Clear database'),
67.    write('\n2. Execute GBFS'),
68.    write('\n3. Display database'),
69.    write('\n4. Save database'),
70.    write('\n5. Exit'),
71.    write('\n\nEnter your choice: '),
72.    read(N), N > 0, N < 6,
73.    do(N), N=5,!.
74.
75.do(1):-retractall(t_node(_, _)), retractall(pp(_)), retractall(pq(_)).
76.do(2):- search.
77.do(3):- listing(t_node), listing(pq), listing(pp).
78.do(4):- tell('gbfs_db.pl'), listing(t_node), listing(pq), listing(pp),told.
79.do(5):- abort.

```

Output:

```
1 ?- start.

1. Clear database
2. Execute GBFS
3. Display database
4. Save database
5. Exit

Enter your choice: 2.
Enter start node:i.
Solution:[i,b,e,g]
Length:107
1. Clear database
2. Execute GBFS
3. Display database
4. Save database
5. Exit

Enter your choice: 3.
:- dynamic t_node/2.

t_node(i, nil).
t_node(a, i).
t_node(b, i).
t_node(d, b).
t_node(e, b).
t_node(f, b).
t_node(g, e).

:- dynamic pq/1.

pq([node(g, 0), node(d, 25), node(a, 55)]).

:- dynamic pp/1.

pp([i, b, e, g]).
```

A* Search:

```
1. % Including data files
2. :-use_module(inputGraph).
3.
4. % Declaration of dynamic data
5. :-dynamic(t_node/4).
6. :-dynamic(pq/1).
7. :-dynamic(pp/1).
8. :-dynamic(t_n_indx/1).
9.
10.% Search begins
11.search:-write('Enter start node:'),read(S),h_fn(S,HV),
12.    assert(t_node(S,0,nil,HV)),assert(pq([node(S,0,'nil',HV)])),
13.    assert(t_n_indx(1)),generate,find_path_length(L), display_result(L).
14.
15.% Generating the solution
16.generate:-pq([H|_]),H=node(N,_,_,_),N=g, add_to_pp(g),!.
17.generate:-pq([H|_]),H=node(N,I,_,_), update_with(N,I), generate.
18.
```

```

19.% Adding a node to possible path
20.add_to_pp(N):-pp(Lst), append(Lst,[N],Lst1), retract(pp(_)),
21.    assert(pp(Lst1)).
22.
23.
24.% Updating data according to selected node.
25.update_with(N,I):-update_pq_tr(N,I), update_pp(N,I).
26.
27.% Updating Priority Queue and Tree
28.update_pq_tr(N,I):-pq(Lst), delete_1st_element(Lst,Lst1), retract(pq(_)),
29.    assert(pq(Lst1)), add_children(N,I).
30.delete_1st_element(Lst,Lst1):-Lst = [_|Lst1].
31.add_children(N,I):- neighbor(N,X,D), t_n_indx(I1), t_node(_,I,_,V),
32.    h_fn(N,V1), h_fn(X,V2), FNV is V+D-V1+V2,
33.    insrt_to_pq(X,I1,I,FNV), assert(t_node(X,I1,I,FNV)),
34.    incr_indx, fail.
35.    add_children(_,_).
36.    incr_indx:- t_n_indx(X), Y is X+1, retract(t_n_indx(X)), assert(t_n_indx(
    Y)).
37.
38.% Inserting node to Priority Queue
39.insrt_to_pq(X,I1,I,FNV):- pq(Lst), insert12pq(node(X,I1,I,FNV),Lst,Lst1),
40.    retract(pq(_)), assert(pq(Lst1)).
41.
42.insert12pq(E1,[], [E1]):-!.
43.insert12pq(E1, L1, L2):-L1=[H|_], E1=node(_,_,_,V1), H=node(_,_,_,V2),
44.    not(V1 > V2), L2 = [E1|L1], !.
45.insert12pq(E1, L1, L2):-L1=[H|T], insert12pq(E1, T, Lx), L2 = [H|Lx].
46.
47.% Updating Possible Path
48.update_pp(N,I):- retract(pp(_)), assert(pp([])), renew_pp(N,I).
49.renew_pp(N,I):-t_node(N,I,nil,_), pp(X), append([N],X,X1),
50.    retract(pp(_)), assert(pp(X1)), !.
51.renew_pp(N,I):- pp(X), append([N],X,X1), retract(pp(_)), assert(pp(X1)),
52.    t_node(N,I,I1,_),t_node(N1,I1,_,_), renew_pp(N1,I1).
53.
54.% Finding 'shortest' path length
55.find_path_length(L):-pp(Lst),path_sum(Lst,L).
56.path_sum(Lst,0):- Lst=[g|_],!.
57.path_sum(Lst,L):-
    Lst=[N|T],T=[N1|_], neighbor(N,N1,D), path_sum(T,L1),L is L1+D.
58.
59.% Displaying 'shortest' path and its length
60.display_result(L):- pp(Lst), write('Solution:'), write(Lst),nl,
61.    write('Length:'), write(L).
62.
63.% Arrange a menu of actions
64.start:- repeat,
65.    write('\n1. Clear database'),
66.    write('\n2. Execute GBFS'),
67.    write('\n3. Display database'),
68.    write('\n4. Save database'),
69.    write('\n5. Exit'),
70.    write('\n\nEnter your choice: '),

```

```
71.    read(N), N > 0, N < 6,
72.    do(N), N=5,!.
73.
74.do(1):-retractall(t_node(_,_)), retractall(pp(_)), retractall(pq(_)).
75.do(2):- search.
76.do(3):- listing(t_node), listing(pq), listing(pp).
77.do(4):- tell('astars_db.pl' ), listing(t_node), listing(pq), listing(pp),told
.
78.do(5):- abort.
```