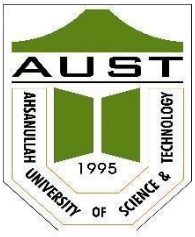
AHSANULLAH UNIVERSITY OF SCIENCE AND TECHNOLOGY
DHAKA-1208, BANGLADESH.



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:

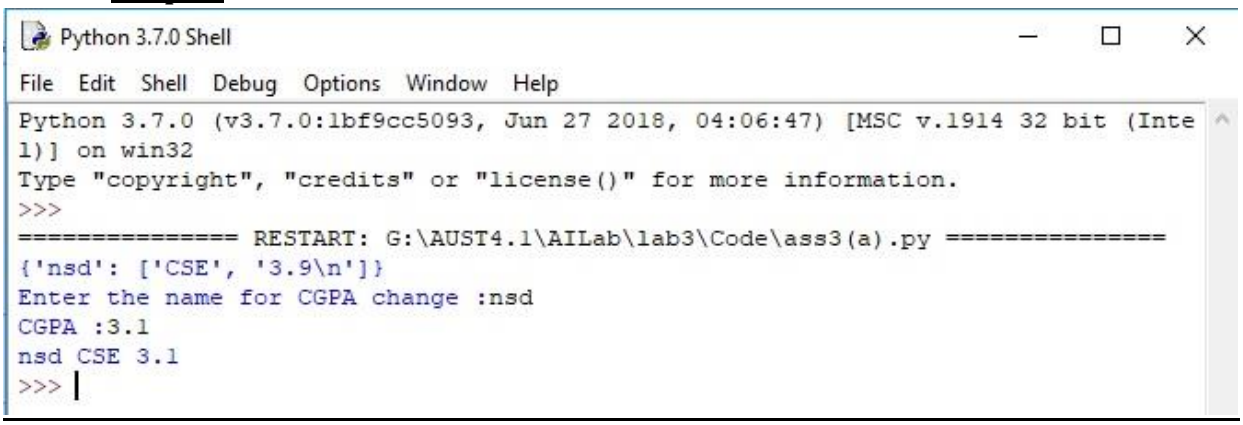
```
fn="te.txt"
Mydict={}
f1=open(fn, "r")
for l in f1:
        name, dept, cgpa =l.split("\t")
        Mydict[name]=[dept,cgpa]
f1.close
print(Mydict)
name=str(input("Enter the name for CGPA change :"))
dept, cgpa = Mydict[name]

cgpa=str(input("CGPA :"))
Mydict[name]=[dept, cgpa]

f1=open(fn, "w")
for x,y in Mydict.items():
        std=str(x)+"\t"+str(y[0])+"\t"+str(y[1])
        print(std, end="\n", file=f1)
f1.close

f1=open(fn, "r")
for l in f1:
        name, dept, cgpa =l.split("\t")
        print(name, dept, float(cgpa), end="\n")
      f1.close
```

Output:



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

```
:-module(inputGraph, [ neighbor/3, h_fn/2 ]).

% Description of the weighted graph

neighbor(i,a,35). neighbor(a,i,35). neighbor(i,b,45). neighbor(b,i,45).
neighbor(a,c,22). neighbor(c,a,22). neighbor(a,d,32). neighbor(d,a,32).
neighbor(b,d,28). neighbor(d,b,28). neighbor(b,e,36). neighbor(e,b,36).
neighbor(b,f,27). neighbor(f,b,27). neighbor(c,d,31). neighbor(d,c,31).
neighbor(c,g,47). neighbor(g,c,47). neighbor(d,g,30). neighbor(g,d,30).
neighbor(e,g,26). neighbor(g,e,26).

% Description of heuristic function values
% Straight line distance from a node to the goal node, g

h_fn(i,80). h_fn(a,55). h_fn(b,42). h_fn(c,34).
h_fn(d,25). h_fn(e,20). h_fn(f,17). h_fn(g,0).
```

Best-first Search:

```
% Including data files
:-use_module(inputGraph).

% Declaration of dynamic data
:-dynamic(t_node/2).
:-dynamic(pq/1).
:-dynamic(pp/1).

% Search begins
search:-write('Enter start node:'),read(S),h_fn(S,HV),
    assert(t_node(S, 'nil')),assert(pq([node(S,HV)])),
    assert(pp([])),generate,find_path_length(L), display_result(L).

% Generating the solution
generate:-pq([H|_]),H=node(N,_),N=g, add_to_pp(g),!.
generate:-pq([H|_]),H=node(N,_),update_with(N), generate.

% Adding a node to possible path
add_to_pp(N):-pp(Lst), append(Lst,[N],Lst1), retract(pp(_)),
    assert(pp(Lst1)).

% Updating data according to selected node.
update_with(N):-update_pq_tr(N), update_pp(N).

% Updating Priority Queue and Tree
update_pq_tr(N):-pq(Lst), delete_1st_element(Lst,Lst1), retract(pq(_)),
    assert(pq(Lst1)), add_children(N).
```

```
delete_1st_element(Lst,Lst1):-Lst = [_|Lst1].
add_children(N):- neighbor(N,X,_), not(t_node(X,_)),insrt_to_pq(X),
    assert(t_node(X,N)),fail.
add_children(_).
add_children(N,I):- neighbor(N,X,D), t_n_indx(I1), t_node(_,I,_,V),
    h_fn(N,V1), h_fn(X,V2), FNV is V+D-V1+V2,
    insrt_to_pq(X,I1,I,FNV), assert(t_node(X,I1,I,FNV)),
    incr_indx, fail.
add_children(_,_).
incr_indx:-  t_n_indx(X), Y is X+1, retract(t_n_indx(X)), assert(t_n_indx(Y))
   .

% Inserting node to Priority Queue
insrt_to_pq(X):- pq(Lst), h_fn(X,V), insert12pq(node(X,V),Lst,Lst1),
    retract(pq(_)), assert(pq(Lst1)).

insert12pq(El,[], [El]):-!.
insert12pq(El, L1, L2):-L1=[H|_], El=node(_,V1), H=node(_,V2),
                            not(V1 > V2), L2 = [El|L1], !.
insert12pq(El, L1, L2):-L1=[H|T], insert12pq(El, T, Lx), L2 = [H|Lx].

% Updating Possible Path
update_pp(N):- retract(pp(_)), assert(pp([])), renew_pp(N).
renew_pp(N):-t_node(N,nil), pp(X), append([N],X,X1),
    retract(pp(_)), assert(pp(X1)), !.
renew_pp(N):- pp(X), append([N],X,X1), retract(pp(_)), assert(pp(X1)),
    t_node(N,N1), renew_pp(N1).

% Finding 'shortest' path length
find_path_length(L):-pp(Lst),path_sum(Lst,L).
path_sum(Lst,0):- Lst=[g|_],!.
path_sum(Lst,L):-
Lst=[N|T],T=[N1|_], neighbor(N,N1,D), path_sum(T,L1),L is L1+D.

% Displaying 'shortest' path and its length
display_result(L):- pp(Lst), write('Solution:'), write(Lst),nl,
    write('Length:'), write(L).

% Arrange a menu of actions
start:- repeat,
    write('\n1. Clear database'),
    write('\n2. Execute GBFS'),
    write('\n3. Display database'),
    write('\n4. Save database'),
    write('\n5. Exit'),
    write('\n\nEnter your choice: '),
    read(N), N >0, N < 6,
    do(N), N=5,!.

do(1):-retractall(t_node(_,_)), retractall(pp(_)), retractall(pq(_)).
do(2):- search.
do(3):- listing(t_node), listing(pq), listing(pp).
do(4):- tell('gbfs_db.pl'), listing(t_node), listing(pq), listing(pp),told.
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.
```

```
% Adding a node to possible path
add_to_pp(N):-pp(Lst), append(Lst,[N],Lst1), retract(pp(_)),
    assert(pp(Lst1)).


% Updating data according to selected node.
update_with(N,I):-update_pq_tr(N,I), update_pp(N,I).

% Updating Priority Queue and Tree
update_pq_tr(N,I):-pq(Lst), delete_1st_element(Lst,Lst1), retract(pq(_)),
    assert(pq(Lst1)), add_children(N,I).
delete_1st_element(Lst,Lst1):-Lst = [_|Lst1].
add_children(N,I):- neighbor(N,X,D), t_n_indx(I1), t_node(_,I,_,V),
    h_fn(N,V1), h_fn(X,V2), FNV is V+D-V1+V2,
    insrt_to_pq(X,I1,I,FNV), assert(t_node(X,I1,I,FNV)),
    incr_indx, fail.
   add_children(_,_).
   incr_indx:-  t_n_indx(X), Y is X+1, retract(t_n_indx(X)), assert(t_n_indx(
Y)).

% Inserting node to Priority Queue
insrt_to_pq(X,I1,I,FNV):- pq(Lst), insert12pq(node(X,I1,I,FNV),Lst,Lst1),
    retract(pq(_)), assert(pq(Lst1)).

insert12pq(El,[], [El]):-!.
insert12pq(El, L1, L2):-L1=[H|_], El=node(_,_,_,V1), H=node(_,_,_,V2),
                            not(V1 > V2), L2 = [El|L1], !.
insert12pq(El, L1, L2):-L1=[H|T], insert12pq(El, T, Lx), L2 = [H|Lx].

% Updating Possible Path
update_pp(N,I):- retract(pp(_)), assert(pp([])), renew_pp(N,I).
renew_pp(N,I):-t_node(N,I,nil,_), pp(X), append([N],X,X1),
    retract(pp(_)), assert(pp(X1)), !.
renew_pp(N,I):- pp(X), append([N],X,X1), retract(pp(_)), assert(pp(X1)),
    t_node(N,I,I1,_),t_node(N1,I1,_,_), renew_pp(N1,I1).

% Finding 'shortest' path length
find_path_length(L):-pp(Lst),path_sum(Lst,L).
path_sum(Lst,0):- Lst=[g|_],!.
path_sum(Lst,L):-
Lst=[N|T],T=[N1|_], neighbor(N,N1,D), path_sum(T,L1),L is L1+D.

% Displaying 'shortest' path and its length
display_result(L):- pp(Lst), write('Solution:'), write(Lst),nl,
    write('Length:'), write(L).

% Arrange a menu of actions
start:- repeat,
    write('\n1. Clear database'),
    write('\n2. Execute GBFS'),
    write('\n3. Display database'),
    write('\n4. Save database'),
    write('\n5. Exit'),
    write('\n\nEnter your choice: '),
```

```
    read(N), N >0, N < 6,
    do(N), N=5,!.

do(1):-retractall(t_node(_,_)), retractall(pp(_)), retractall(pq(_)).
do(2):- search.
do(3):- listing(t_node), listing(pq), listing(pp).
do(4):- tell('astars_db.pl' ), listing(t_node), listing(pq), listing(pp),told
   .
do(5):- abort.
```