**Questions:**

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

**Solution to the question no 1**

The demonstrated Python code:

dict={}

f1=open('stdfile.py', "r")

for l in f1:

name, dept, cgpa =l.split("\t")

dict[name] = [dept,float(cgpa)]

print(name+'\t'+dept+'\t'+str(cgpa))

f1.close

num=int(input("How many records you want to change:"))

for i in range(num):

name = input("Enter the name were you want to change the cgpa:")

cgpa = float(input("New cgpa is:"))

dict[name][1]= cgpa

f1=open('stdfile.py', "w")

for name in dict:

dept=dict[name][0]

cgpa=dict[name][1]

std=name+"\t"+dept+"\t"+str(cgpa)

print(std, end="\n", file=f1)

f1.close

print("\nUpdate:")

f1=open('stdfile.py', "r")

for l in f1:

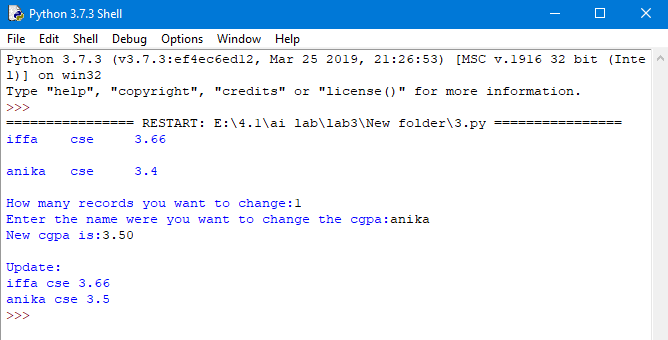
name, dept, cgpa =l.split("\t")

dict[name] = [dept,float(cgpa)]

print(name, dept, float(cgpa), end="\n")

f1.close

A sample of input and output is as below:



**Solution to the question no 2**

The demonstrated Prolog code is below:

% 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(\_).

% 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).

% List dynamic data

list\_records:-listing(t\_node), listing(pq), listing(pp).

% Save file with modified records in place of old ones.

save\_records:-tell('gbfs\_db.pl'), listing(t\_node), listing(pq), listing(pp),told.

%Clear the database

clr\_db:-retractall(t\_node(\_,\_)), retractall(pp(\_)), retractall(pq(\_)).

% 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):-clr\_db.

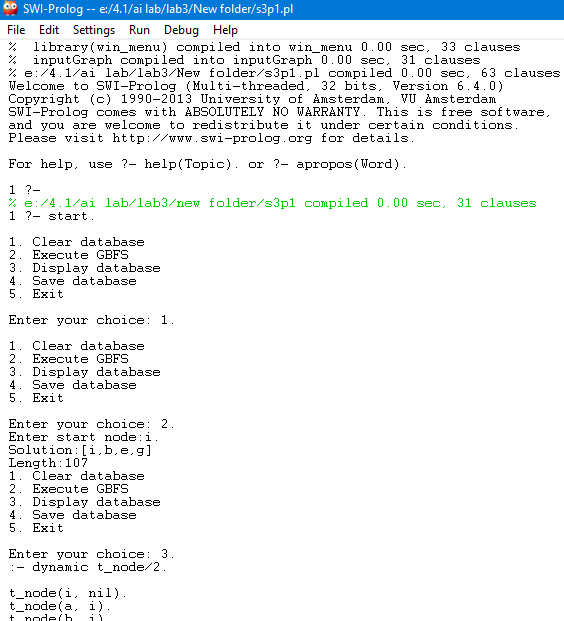
do(2):-search.

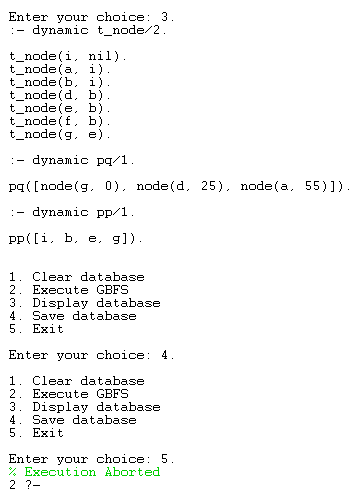
do(3):-list\_records.

do(4):-save\_records.

do(5):-abort.

A sample of input and output is as below:





The demonstrated Python code is below:

from collections import defaultdict

def dfs(source,dest,visited,path):

visited[source]= True

path.append(source)

if source == dest:

total =0

print(path)

l = len(path)

for i in range(l-1):

total += graph[path[i]][path[i+1]]

print(total)

else:

for i in graph[source]:

if visited[i] == False:

dfs(i,dest,visited,path)

path.pop()

visited[source]=False

graph = defaultdict(dict)

graph[0][1]=35

graph[0][2]=45

graph[1][3]=22

graph[1][4]=32

graph[2][4]=28

graph[2][5]=36

graph[2][6]=27

graph[4][7]=30

graph[3][4]=31

graph[3][7]=47

graph[4][7]=30

graph[5][7]=26

source = int(input("Source: "))

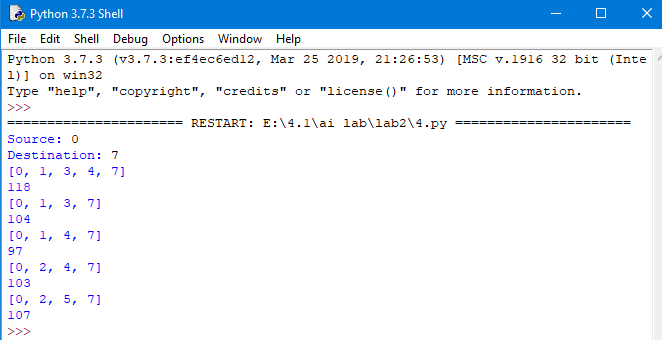
dest = int(input("Destination: "))

visited = [False]\*8

path=[]

dfs(source,dest, visited, path)

A sample of input and output is as below:



**Solution to the question no 2 (A\*)**

% Including data files

:-use\_module(inputGraph).

% Declaration of dynamic data

:-dynamic(t\_node/4).

:-dynamic(t\_indx/1).

:-dynamic(pq/1).

:-dynamic(pp/1).

% Search begins

search:-write('Enter start node:'),read(S),h\_fn(S,HV),

assert(t\_node(S,0,nil,HV)),assert(pq([node(S,0,'nil',HV)])),assert(t\_n\_indx(1)),

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,I,\_,\_), update\_with(N,I),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,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).

% List dynamic data

list\_records:-listing(t\_node), listing(pq), listing(pp).

% Save file with modified records in place of old ones.

save\_records:-tell('astars\_db.pl'), listing(t\_node), listing(pq), listing(pp),told.

%Clear the database

clr\_db:-retractall(t\_node(\_,\_,\_,\_)),retractall(t\_n\_indx(\_)), retractall(pp(\_)), retractall(pq(\_)).

% 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):-clr\_db.

do(2):-search.

do(3):-list\_records.

do(4):-save\_records.

do(5):-abort.

A sample of input and output is as below:

