Questions:

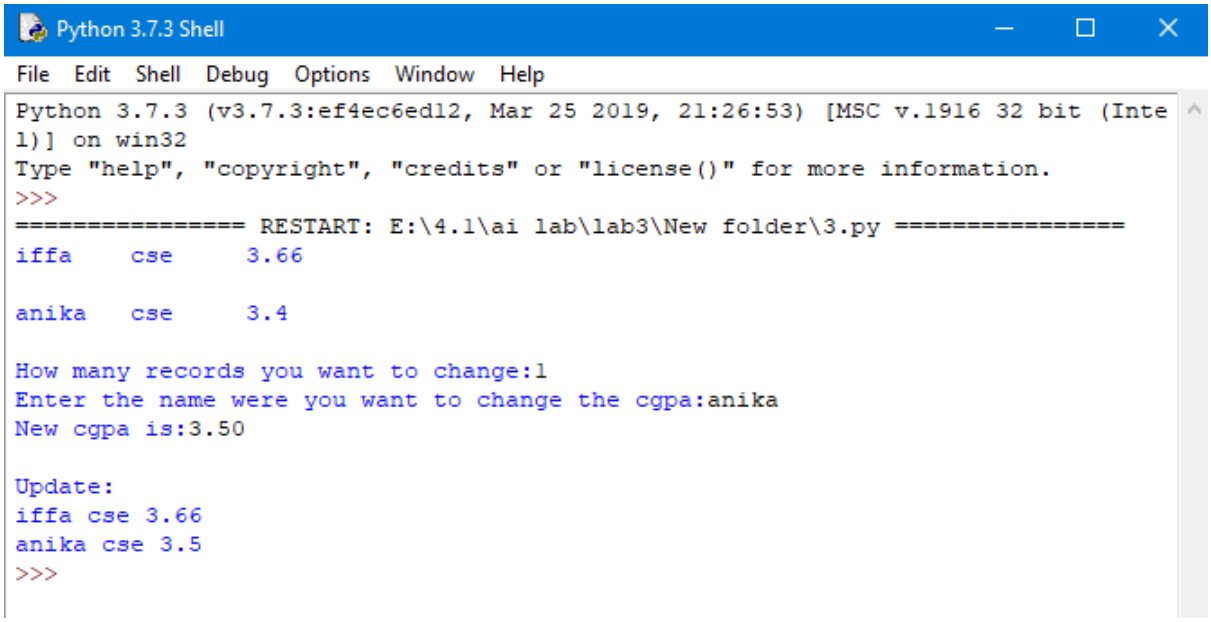
1) **Write a Python program that reads the file created as demonstrated into a dictionary taking 'name' as the key and a list consisting of 'dept' and 'cgpa' as the value for each line. Make changes in some 'cgpa' and then write back the whole file.**
2) **Implement in generic ways (as multi-modular 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

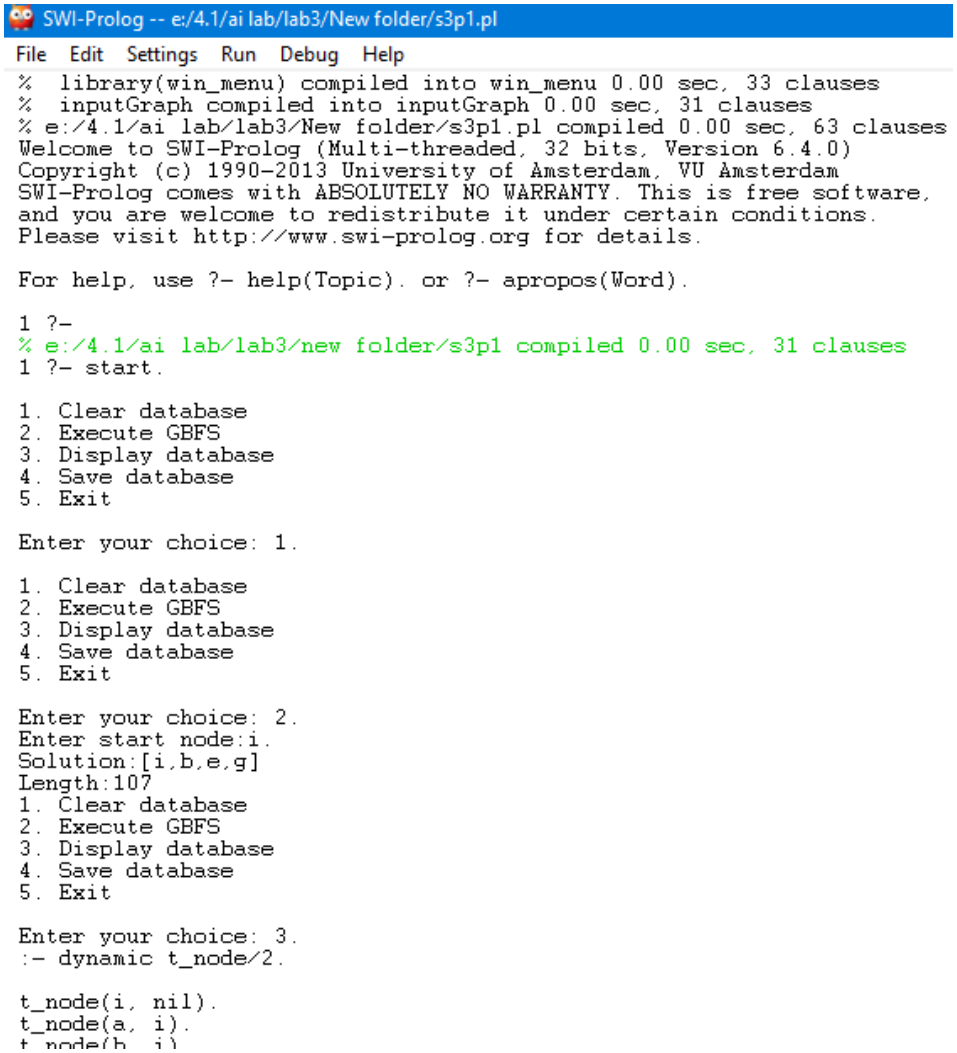
The demonstrated Prolog code is below:

```
% 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:



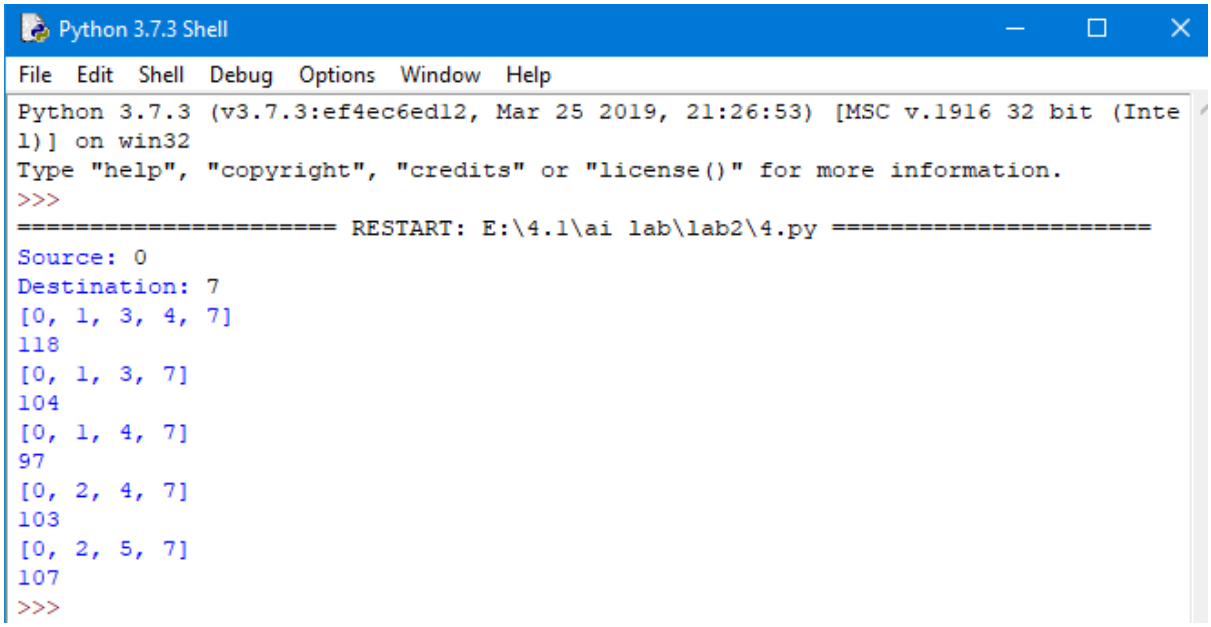
```
Enter your choice: 3.
:- dynamic t_node/2.

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]).


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

Enter your choice: 4.

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

Enter your choice: 5.
% Execution Aborted
2 ?-
```

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:



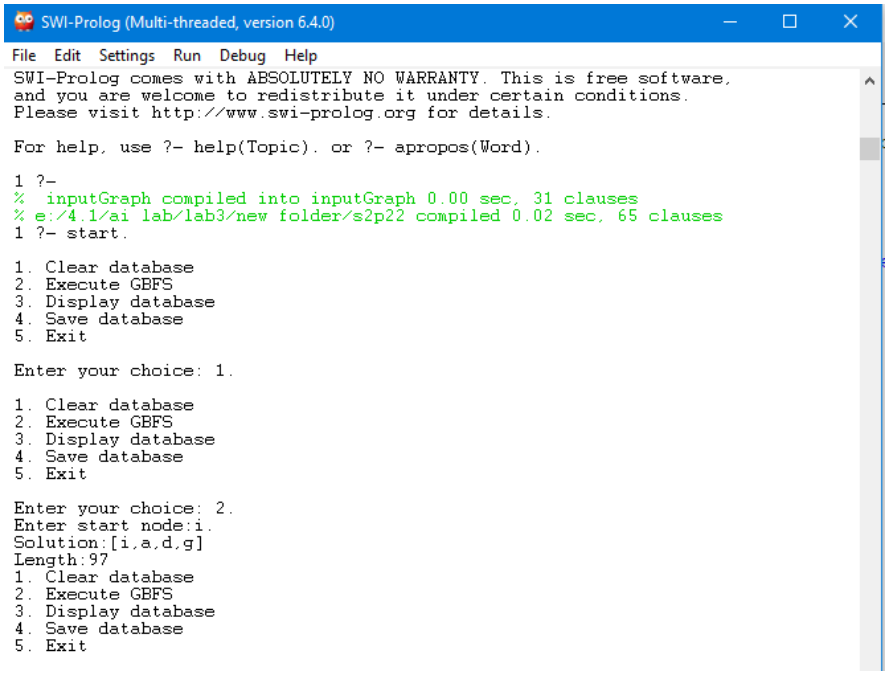
Solution to the question no 2 (A*)

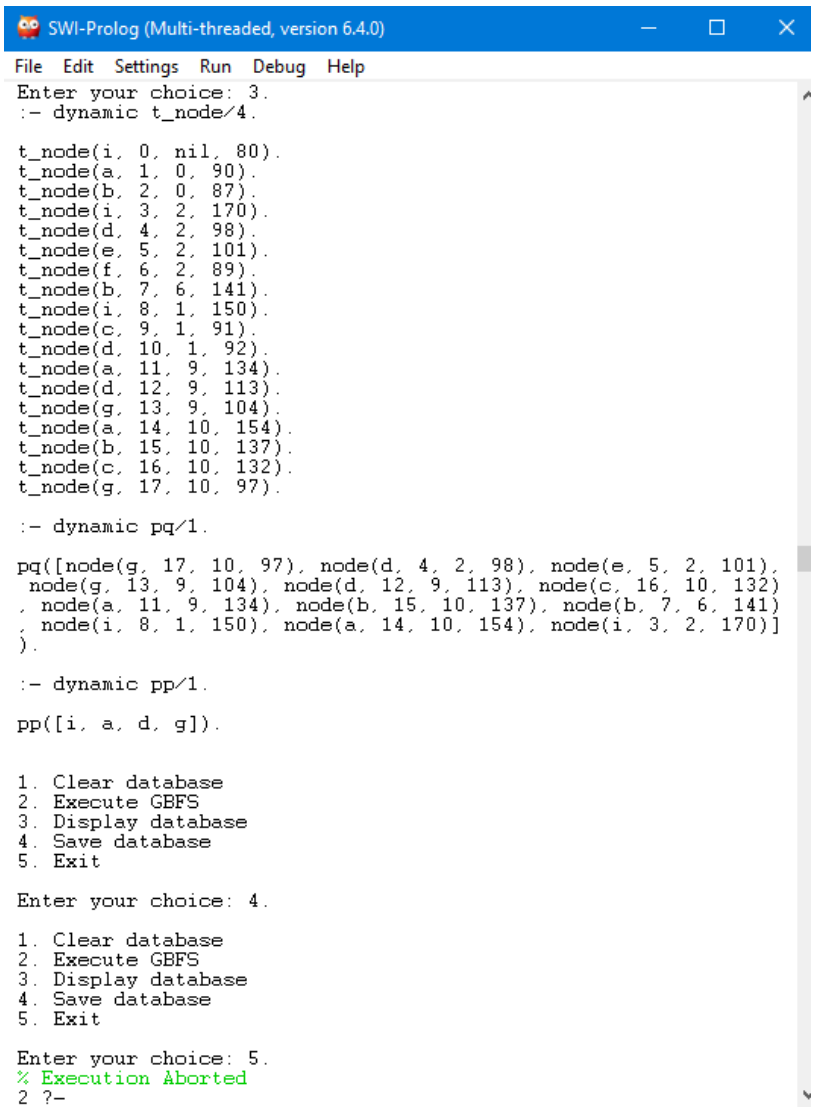
```
% Including data files
:-use_module(inputGraph).
```

```
% 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:


SWI-Prolog (Multi-threaded, version 6.4.0)
File Edit Settings Run Debug Help
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.
For help, use ?- help(Topic). or ?- apropos(Word).
1 ?-
% inputGraph compiled into inputGraph 0.00 sec, 31 clauses
% e:/4.1/ai lab/lab3/new folder/s2p22 compiled 0.02 sec, 65 clauses
1 ?- start.
1. Clear database
2. Execute GBFS
3. Display database
4. Save database
5. Exit
Enter your choice: 1.
1. Clear database
2. Execute GBFS
3. Display database
4. Save database
5. Exit
Enter your choice: 2.
Enter start node:i.
Solution:[i,a,d,g]
Length:97
1. Clear database
2. Execute GBFS
3. Display database
4. Save database
5. Exit


SWI-Prolog (Multi-threaded, version 6.4.0)
File Edit Settings Run Debug Help
Enter your choice: 3.
:- dynamic t_node/4.
t_node(i, 0, nil, 80).
t_node(a, 1, 0, 90).
t_node(b, 2, 0, 87).
t_node(i, 3, 2, 170).
t_node(d, 4, 2, 98).
t_node(e, 5, 2, 101).
t_node(f, 6, 2, 89).
t_node(b, 7, 6, 141).
t_node(i, 8, 1, 150).
t_node(c, 9, 1, 91).
t_node(d, 10, 1, 92).
t_node(a, 11, 9, 134).
t_node(d, 12, 9, 113).
t_node(g, 13, 9, 104).
t_node(a, 14, 10, 154).
t_node(b, 15, 10, 137).
t_node(c, 16, 10, 132).
t_node(g, 17, 10, 97).
:- dynamic pq/1.
pq([node(g, 17, 10, 97), node(d, 4, 2, 98), node(e, 5, 2, 101),
node(g, 13, 9, 104), node(d, 12, 9, 113), node(c, 16, 10, 132)
, node(a, 11, 9, 134), node(b, 15, 10, 137), node(b, 7, 6, 141)
, node(i, 8, 1, 150), node(a, 14, 10, 154), node(i, 3, 2, 170)]
).
:- dynamic pp/1.
pp([i, a, d, g]).
1. Clear database
2. Execute GBFS
3. Display database
4. Save database
5. Exit
Enter your choice: 4.
1. Clear database
2. Execute GBFS
3. Display database
4. Save database
5. Exit
Enter your choice: 5.
% Execution Aborted
2 ?-