**Supplementary Material for Session 3**

1. **Major components of Prolog code for GBF and A\* search:**

% Including data files

:-use\_module(inputGraph).

-----------

% Declaration of dynamic data

:-dynamic(t\_node/2).

:-dynamic(pq/1).

:-dynamic(pp/1).

----------

*Changed:* …:-dynamic(t\_node/4).

Additional: :-dynamic(t\_n\_indx/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).

--------

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

% 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.

---------

*Changed:* … H=node(N,\_,\_,\_) … H=node(N,I,\_,\_), update\_with(N,I)

% 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).

---------

*Changed:* … (N,I)…

% 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(\_).

---------

*Changed:* … (N,I)…

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].

---------

*Changed:*

insrt\_to\_pq(X,I1,I,FNV):- pq(Lst), insert12pq(node(X,I1,I,FNV),Lst,Lst1),

retract(pq(\_)), assert(pq(Lst1)).

… El=node(\_,\_,\_,V1), H=node(\_,\_,\_,V2)…

% 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).

---------

*Changed:* … (N,I)… t\_node(N,I,nil,\_) … 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).

---------

1. **Utilities for Prolog code**

% 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):-write('Done 1').

do(2):- write('Done 2').

do(3):- write('Done 3').

do(4):- write('Done 4').

do(5):- abort.

% Display 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.

---------

*Changed:* … 'astars\_db.pl' …

%Clear the database

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

---------

*Changed:* … t\_node(\_,\_,\_,\_)

*Added:* retractall(t\_n\_indx(\_))

1. **Utilities for Python code**

**# Writing to and reading from a file in Python**

f1=open(fn, "w")

print("\n")

for i in range(ln):

name=str(input("Enter the name:"))

dept=str(input("Enter the department:"))

cgpa=str(input("Enter the cgpa:"))

std=name+"\t"+dept+"\t"+cgpa

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

print("\n")

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

**# Including files**

import s3Module1 as m1

m1.display\_file\_lines(fn,ln)

n=m1.num\_of\_lines(fn)

print("Number of lines in {} is {}.".format(fn,n))

m1.display\_file(fn)

**#Content of** s3Module1

**#Finding number of lines in a file fn**

**def num\_of\_lines(fn):**

**c=0**

**for l in open(fn):**

**c=c+1**

**return c**

**"""**

**Displaying all lines of student file fn each containing name,**

**department and cgpa separated by tabs**

**"""**

**def display\_file(fn):**

**f1=open(fn, "r")**

**for l in f1:**

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

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

**f1.close**

**"""**

**Displaying ln lines of student file fn each containing name,**

**department and cgpa separated by tabs**

**"""**

**def display\_file\_lines(fn,ln):**

**f1=open(fn, "r")**

**for i in range(ln):**

**l=f1.readline()**

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

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

**f1.close**