AI:

Prolog

1. Create a new Prolog program for finding a path from one node to other, by following edges between nodes.

```
edge(1, 2).
edge(1, 3).
edge(2, 3).
edge(2, 4).
edge(3, 4).
edge(4, 5).

path(Start, End):- edge(Start, End),Write(Start),Write(End).
path(Start, End):- write(Start), edge(Start,temp),path(temp, End).
```

Ans: path(1,5).

TSP:

```
/* perm(A, B) : B  is a permutation of Generator of B's */
perm([],[]).
perm([A|S], [A|T]) : - perm(S,T).
perm([A|S], [A|T]):- perm(S,T1),
exchange(A, B, S, T)



%%--------------------------------TSP-----------------------------%%
/*  exchange A for in set S to obtain set T */

exchange(A, B, [B|T], [A|T]).
exchange(A, B ,[C|S], [C|T]):-
exchange(A, B, S ,T).

cities(P):- setof(C,city(C),P).
walk([C|W]):- cities([C|P]), perm(P,W).

ccost([A|R], V) :- ccost([A|R], V,A).

ccost([A|R], V, F) :-cost(A,F,V), !.
ccost([A,B|R],V, F) :- cost(A, B, V1) ,
                       ccost([B|R],V2,F) ,
                       V is V1+V2.

itinerary(W, V) :-walk(W), ccost(W,V).
```

```
solve(X):- setof(V-W),itinerary(W, V, B)), best(B,X).
best([K-P|R], X):- best(R, L-Q),better(K-P, L-Q,X),!.
best([X],X).

better(K-P, L-_,K-P):- K<L,!.


/* data */

city(bombay).
city(delhi).

c(bombay,delhi, 1456).

cost(A, B, V) :- c(A, B, V);c(B, A, V).


/* Alternative Source reference : stackoverflow */

road(delhi,bombay, 1456).
road(delhi,chennai, 2158).
road(delhi,chennai, 2158).


get_road(Start, End, Visited, Result) :- get_road(Start, End, [Start], 0, Visited, Result).

get_road(Start, End, Waypoints, DistanceAcc, Visited,TotalDistance):-
road(Start, End,distance),
reverse([End|Waypoints], Visited),
TotalDistance is   DistanceAcc + Distance.


get_road(Start, End, Waypoints, DistanceAcc, Visited,TotalDistance):-

road(Start, Waypoint,distance),
\+member(Waypoint,Waypoints),
NewDistanceAcc is  DistanceAcc +  Distance,
get_road(Waypoint, End, [Waypoint|Waypoints], NewDistanceAcc, Visited,TotalDistance).

Usage :
?- get_road(bombay, delhi,Visited, Distance).
```