```
callastar(Start, Start, [0, 0, Start]) :-
        nl,
        format('The Start State and Goal State are the same').

callastar(Start, Goal, [Cost, Cost, Goal, Start]) :-
        edge(Start, Goal, Cost).

callastar(Start, Goal, Path) :-
        distanceSL(Start, Goal, SLD),
        astar([[0,SLD,Start]], Goal, [], Path).

astar([[PCost, PHCost, Goal |SubPath]|QTail], Goal, _Visted, [PCost, PHCost, Goal |SubPath]) :-

       CASE ONE CODE HERE:  Goal state has been reached.

astar([[PCost, PHCost, CurrentV | SubPath]|QTail], Goal, Visited, Result) :-
           not(member(CurrentV, Visited)),
           findall(V, edge(CurrentV, V, _Cost1), Edgelist), …

          CASE TWO CODE HERE
              1)  Remove vertices from Edgelist that are in Visited list
              2)  Expand the set of configurations using (filtered) Edgelist
              3)  Append(Expandedqueue, Oldqueue, Newqueue)
              4)  Find the minimum configuration and move to the head of Newqueue
              5)  Add CurrentV to Visited list
              6)  Recursively call astar

astar([[PCost, PHCost, CurrentV | SubPath]|QTail], Goal, Visited, Result) :-
           member(CurrentV, Visited), …

          CASE THREE CODE HERE
              1)  Qtail is the Newqueue.
              2)  Find the minimum configuration in QTail and move it to the head of the
                  Newqueue
              3)  Recursively call astar
```

```
NOTE:  Why don't I need to remove more states from the configuration?
       Because every vertex can only be "expanded" once, when it is reached.
       I also don't need to "sort" the queue (so it is not really a queue).
       I just need to find the minimum configuration to expand next.
```

You can assume edges are stored in both directions.
And every vertex has an X,Y coordinate.

```
edge(va, vb, 40).
edge(vb, va, 40).
edge(va, vh, 40).
edge(vh, va, 40).

coordinate(va, 90, 160).
coordinate(vb, 70, 140).
coordinate(vc, 40, 80).
coordinate(vd, 70, 40).
```

CODE for computing Straight Line Distance. Note I round to two decimal places.
(There is also rounding in my code when I add distances).
Rounding isn't necessary (and can be removed) but this made it easier
to read the configuration when I was debugging.

```
distancecalc(Xa, Ya, Xb, Yb, Result) :-
       Result is round(100*sqrt((Xa - Xb)*(Xa - Xb) + (Ya - Yb)*(Ya - Yb)))/100.

distanceSL(V1, V2, SLD) :-
        coordinate(V1, X1, Y1),
        coordinate(V2, X2, Y2),
        distancecalc(X1, Y1, X2, Y2, SLD).
```