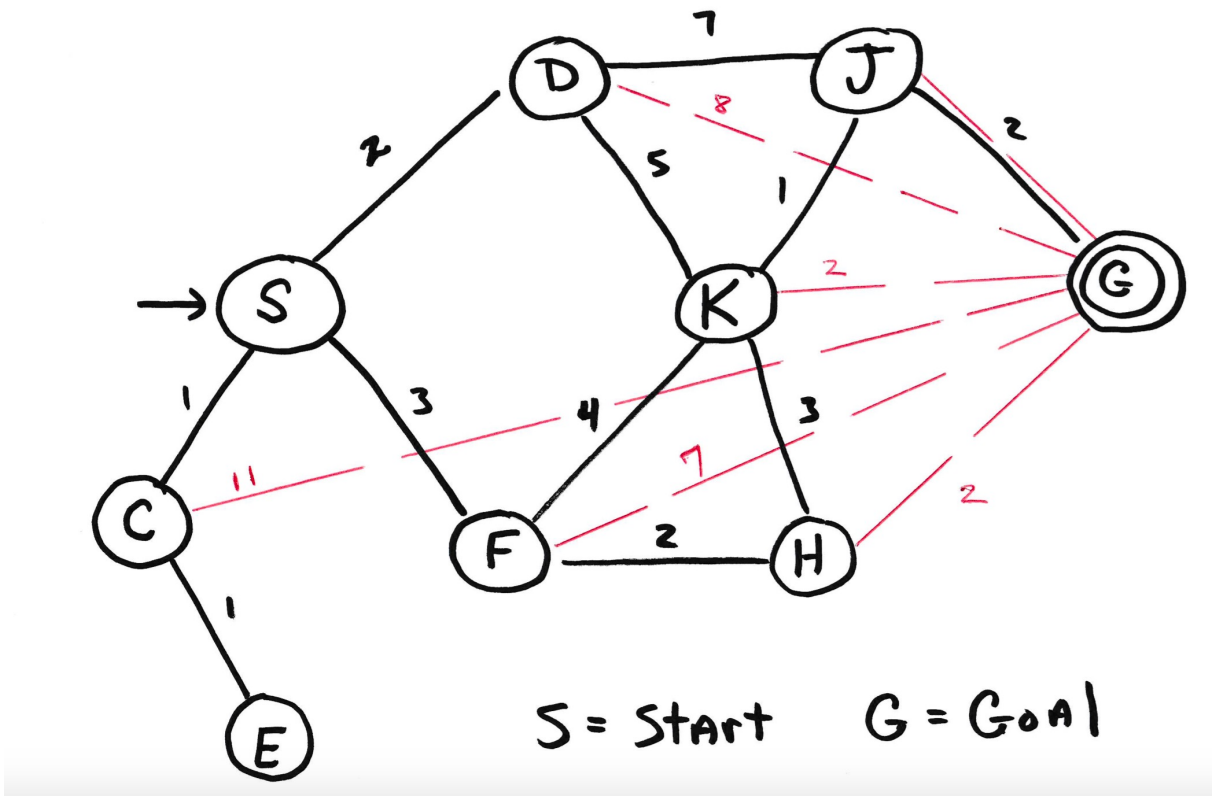
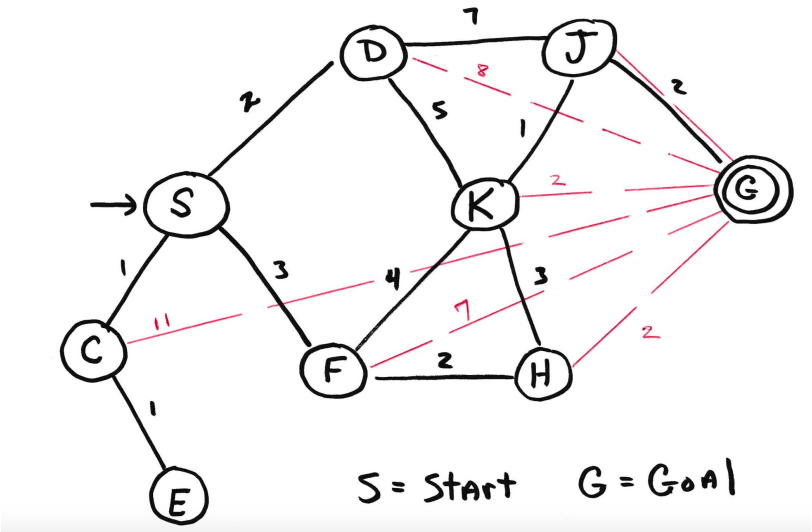
SHORTEST PATH (SP)



The Heuristic
in RED
is the SLD

Think of the SLD Heuristic as a spring
that is pulling us toward the goal.

SHORTEST PATH (SP) WITH A*



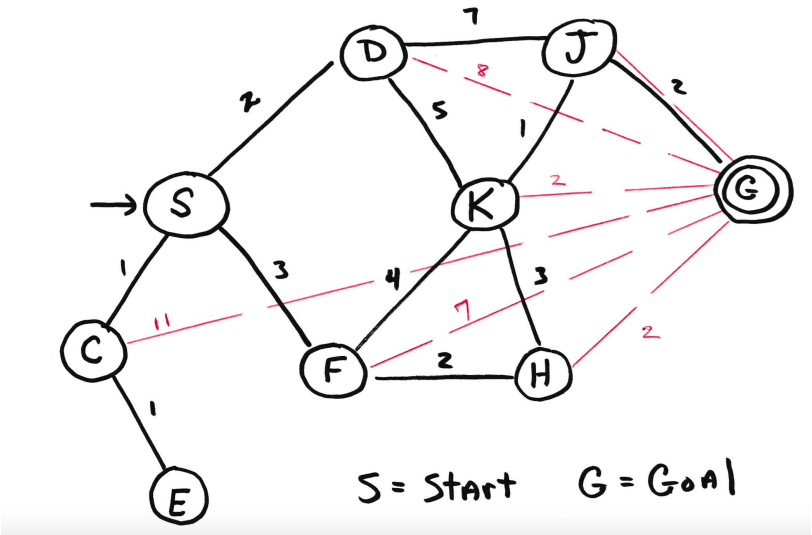
Sort by
(Vertex, Partial+H(V), From)

Queue (S)
Queue ((D, 2+8, S), (F, 3+7, S), (C, 1+11, S))

VISITED
(S, 0)
(D, 2, S)

SLD is a great ADMISSIBLE Heuristic,
But D might or might not be on the shortest path!
However as search progresses H(V) approaches zero.
So A* has to converge correctly! See WHY?

SHORTEST PATH (SP) WITH A*

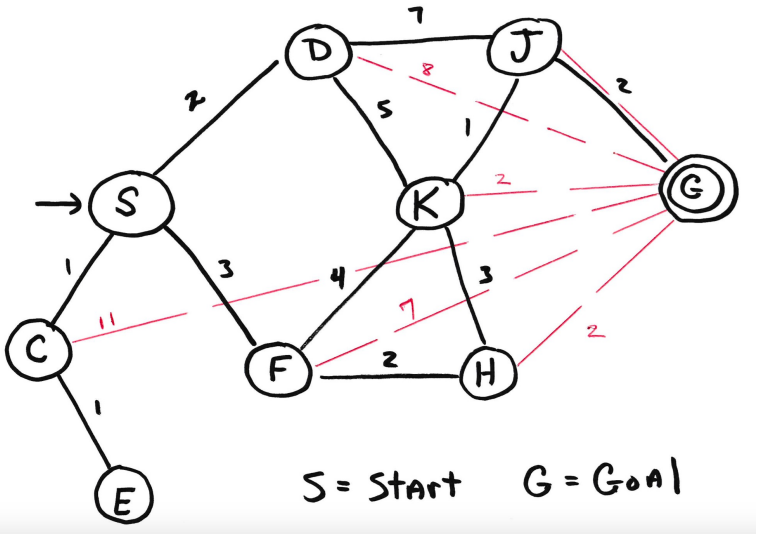


(Vertex, Partial+H(V), From)

Sort by
Partial+H(V)

Queue (S)
Queue ((D, 2+8, S), (F, 3+7, S), (C, 1+11, S))
Queue ((K, 7+2, D), (F, 3+7, S), (J, 9+2, D), (C, 1+11, S))

SHORTEST PATH (SP) WITH A*

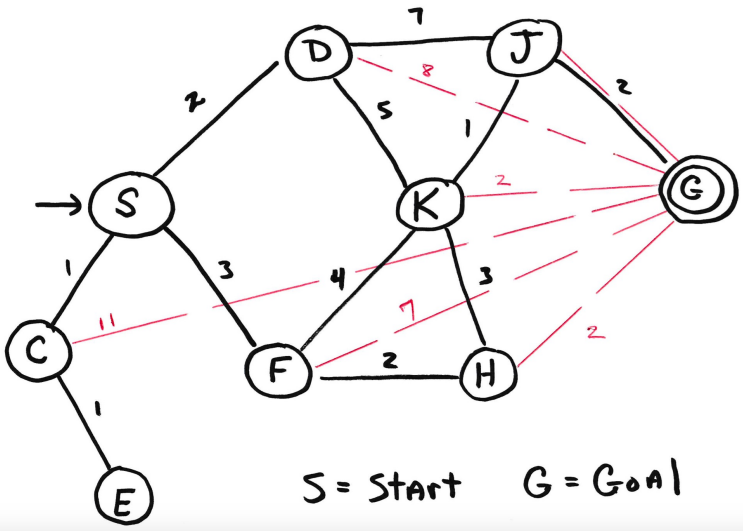


(Vertex, Partial+H(V), From)

Queue (S)
Queue ((D, 2+8, S), (F, 3+7, S), (C, 1+ 11, S))
Queue ((K, 7+2, D), ((F, 3+7, S), (J, 9+2, D), (C, 1+11, S))
Queue ((F, 3+7, S), (J, 8+2, K), (C, 1+11, S), (H, 10+2, K))

Note that we now have a better Partial for J.

SHORTEST PATH (SP) WITH A*

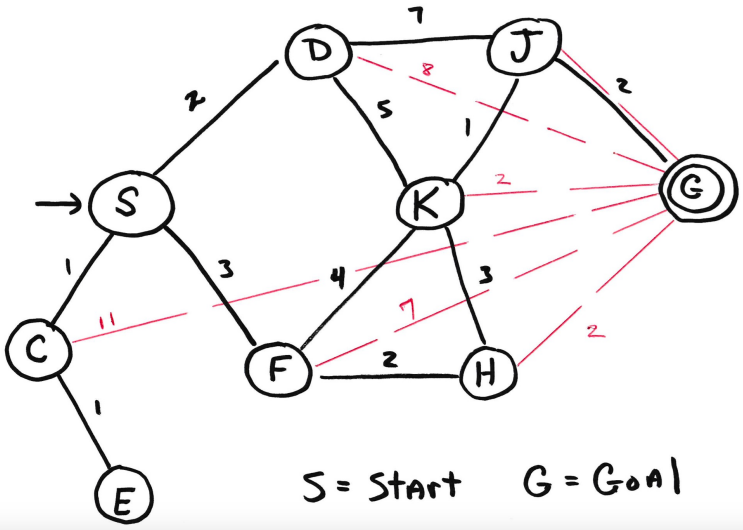


(Vertex, Partial+H(V), From)

Queue (S)
Queue ((D, 2+8, S), (F, 3+7, S), (C, 1+ 11, S))
Queue ((K, 7+2, D), ((F, 3+7, S), (J, 9+2, D), (C, 1+11, S))
Queue ((F, 3+7, S), (J, 8+2, K), (C, 1+11, S), (H, 10+2, K))

F will expand K and H. But we have visited K!
And H is in the Queue.

SHORTEST PATH (SP) WITH A*



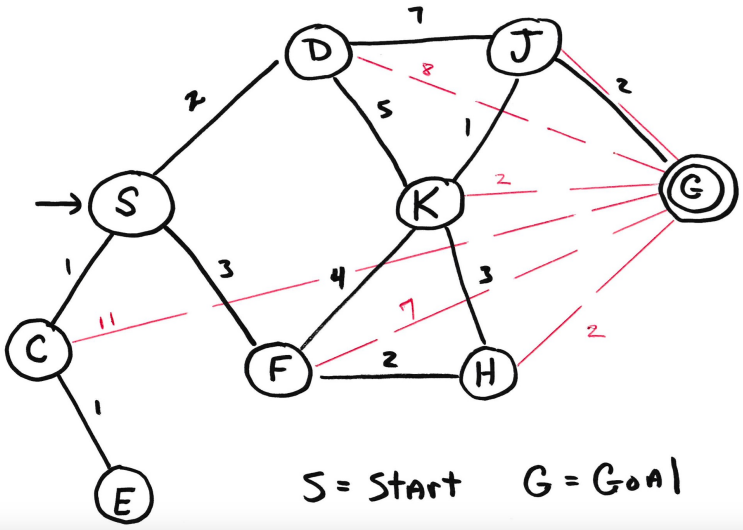
(Vertex, Partial+H(V), From)

Queue (S)
Queue ((D, 2+8, S), (F, 3+7, S), (C, 1+ 11, S))
Queue ((K, 7+2, D), ((F, 3+7, S), (J, 9+2, D), (C, 1+11, S))
Queue ((F, 3+7, S), (J, 8+2, K), (C, 1+11, S), (H, 10+2, K))

MY CODE: Only expands each vertex once, but does not remove duplicates from the Queue. Why not?

Queue ((F, 3+7, S), (J, 8+2, K), (J, 9+2, D) (C, 1+11, S), (H, 10+2, K))

SHORTEST PATH (SP) WITH A*



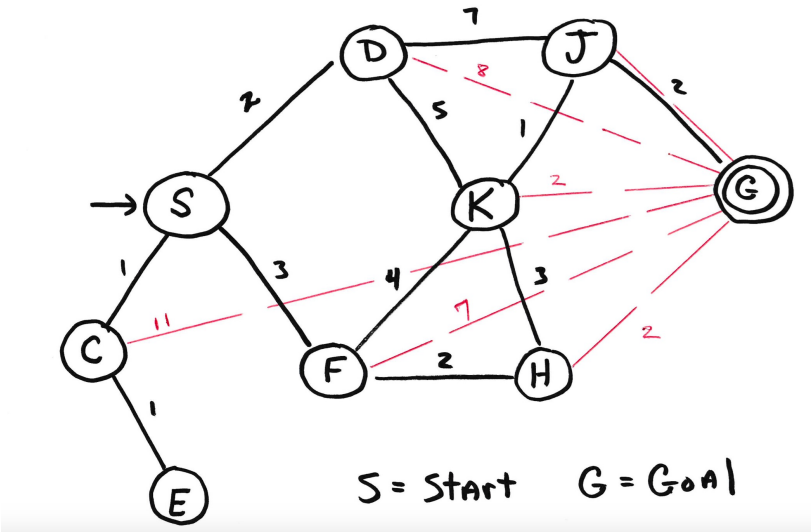
(Vertex, Partial+H(V), From)

Queue ((K, 7+2, D), ((F, 3+7, S), (J, 9+2, D), (C, 1+11, S))
Queue ((F, 3+7, S), (J, 8+2, K), (J, 9+2, D) (C, 1+11, S), (H, 10+2, K))

MY CODE: Only expands each vertex once, but does not remove duplicates from the Queue. Why not?

The shortest path so far is always selected.
Updating the Queue has a cost.
E.G. When you visit J, you MUST select the shortest partial path.

SHORTEST PATH (SP) WITH A*



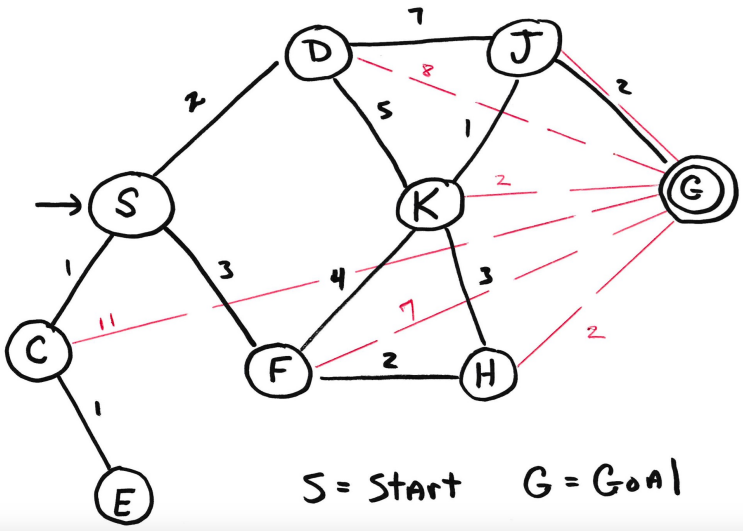
(Vertex, Partial+H(V), From)

Queue (S)
Queue ((D, 2+8, S), (F, 3+7, S), (C, 1+ 11, S))
Queue ((K, 7+2, D), ((F, 3+7, S), (J, 9+2, D), (C, 1+11, S))
Queue ((F, 3+7, S), (J, 8+2, K), (C, 1+11, S), (H, 10+2, K))

MY CODE: Stores the path:

Instead of storing (J, 8+2, K) my code stores [8, 10, J, K, F, S]

SHORTEST PATH (SP) WITH A*



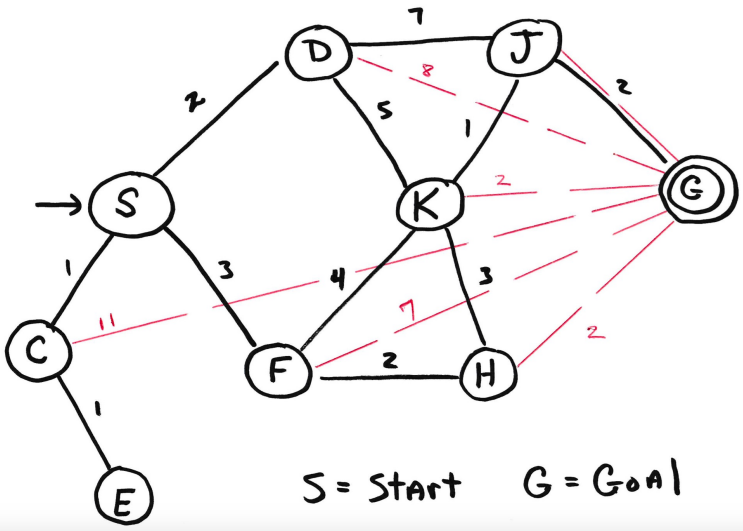
(Vertex, Partial+H(V), From)

Queue ((K, 7+2, D), ((F, 3+7, S), (J, 9+2, D), (C, 1+11, S))
Queue ((F, 3+7, S), (J, 8+2, K), (C, 1+11, S), (H, 10+2, K))

MY CODE: Only expands each vertex once, but does not remove duplicates from the Queue. Why not?

When is J visited? When you **expand** D which connects to J?
Or when you actually **expand** vertex J?

SHORTEST PATH (SP) WITH A*

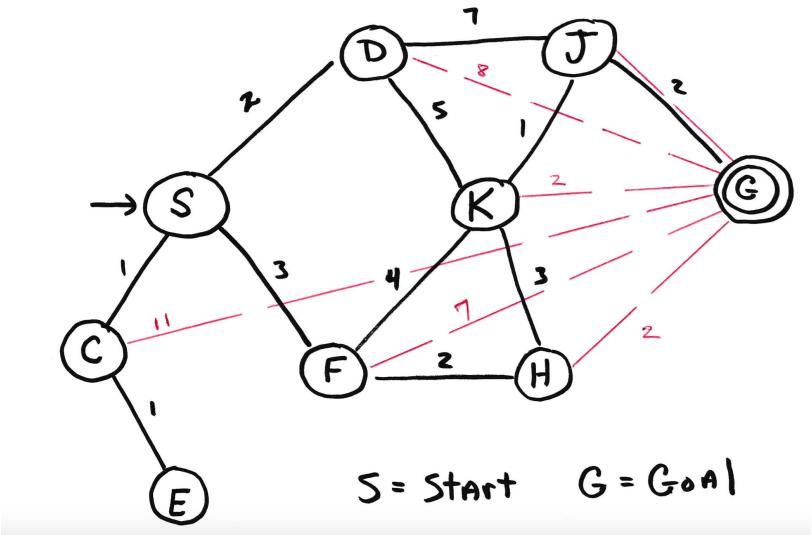


(Vertex, Partial+H(V), From)

Queue ((K, 7+2, D), ((F, 3+7, S), (J, 9+2, D), (C, 1+11, S))
Queue ((F, 3+7, S), (J, 8+2, K), (C, 1+11, S), (H, 10+2, K))

MY CODE: The "list of next edges" and the "visited" list are both simple lists. Remove anything from the "list of next edges" that has already been "visited." A vertex can be in the "Queue" more than once, but will only be visited once.

SHORTEST PATH (SP) WITH A*



(Vertex, Partial+H(V), From)

Example. Visited (and expanded) list: [F, K, D, S]

The neighbors of F are [S, K, H]
but you have already expanded K and S.
So remove S and K from the "list of neighbors" of F.
Add [6, 8, H, F, S] to the Queue of Configurations.

This way, you avoid searching the queue of configurations.

SHORTEST PATH (SP) WITH A*

Here is a Configuration Queue.
What was the start state?

[[40,130.55,vb,va],[40,93.85,vh,va],[90,121.62,vi,va],[150,280,vc,va]]

The Queue (Configuration Vector) is not sorted.
What is the next configuration to be expanded?

SHORTEST PATH (SP) WITH A*

```
coordinate(va, 90, 160).
coordinate(vb, 70, 140).
coordinate(vc, 40, 80).
coordinate(vd, 70, 40).
coordinate(ve, 130, 60).
coordinate(vf, 100, 100).
coordinate(vg, 110, 130).
coordinate(vh, 110, 150).
coordinate(vi, 170, 160).
coordinate(vj, 160, 130).
coordinate(vk, 180, 70).
coordinate(vl, 150, 50).
coordinate(vm, 190, 30).
coordinate(vn, 210, 70).
coordinate(vq, 190, 110).
coordinate(vp, 200, 130).
```

You will be given
Vertices and coordinates.

A sample is found in
Agraph.pl

SHORTEST PATH (SP) WITH A*

Here is a Configuration Queue.
What was the start state?

[[40,130.55,vb,va],[40,93.85,vh,va],[90,121.62,vi,va],[150,280,vc,va]]

I modified my code to only compute SLD to 2 decimal places.

```
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).
```

SHORTEST PATH (SP) WITH A*

Here is another Configuration Queue.
What was the start state?

[[60,110,vg,vh,va],[160,200,vp,vh,va],[120,151.62,vi,vh,va],[40,130.55,vb,va],[90,121.62,vi,va],[150,280,vc,va]]

The Queue (Configuration Vector) is not sorted.
What is the next configuration to be expanded?

SHORTEST PATH (SP) WITH A*

Here is another Configuration Queue.
What was the start state?

[[60,110,vg,vh,va],[160,200,vp,vh,va],[120,151.62,vi,vh,va],[40,130.55,vb,va],[90,121.62,vi,va],[150,280,vc,va]]

The Queue (Configuration Vector) is not sorted.
What is the next configuration to be expanded?

Just search the Queue (Configuration Vector)
For the entry with the minimal PHCost

PHCost is PartialCost + SLD.

SHORTEST PATH (SP) WITH A*

```
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, ExpandedList, Result) :-
            member(CurrentV, ExpandedList), …

          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
```

SHORTEST PATH (SP) WITH A*

```
astar([[PCost, PHCost, CurrentV | SubPath]|QTail], Goal, Visited, Result) :-
            not(member(CurrentV, ExpandedList)),
            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
```

SHORTEST PATH (SP) WITH A*

```
astar([[PCost, PHCost, CurrentV | SubPath]|QTail], Goal, ExpandedList, Result) :-
        not(member(CurrentV, ExpandedList)),


 astar([[PCost, PHCost, CurrentV | SubPath]|QTail], Goal, Visited, Result) :-
            not(member(CurrentV, ExpandedList)),
            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
```

SHORTEST PATH (SP) WITH A*

```
astar([[PCost, PHCost, CurrentV | SubPath]|QTail], Goal, ExpandedList, Result) :-
        not(member(CurrentV, ExpandedList)),
           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
```

SHORTEST PATH (SP) WITH A*

Common bugs (where the logic is almost correct)

```
myrule(A, [Head|Tail], SomePath) :-
    something[A, Head].
    myrule(A, Tail, Somepath).
```

SHORTEST PATH (SP) WITH A*

find(V, edge(va, V, _Cost), Edgelist).

```
edge(va, vb, 40).
edge(va, vh, 40).
edge(va, vi, 90).
edge(va, vc, 150).
edge(vb, va, 40).
edge(vb, vg, 50).
edge(vb, vc, 80).
edge(vb, vd, 110).
edge(vc, vb, 80).
edge(vc, vf, 80).
edge(vc, va, 150).
edge(vc, vd, 60).
edge(vd, vc, 60).
edge(vd, vb, 110).
```

Returns Edgelist = [vb, vh, vi, vc]

Visited = [vc, vb]. /* these have been visited and expanded */

Filtered Edgelist = [vi, vc]