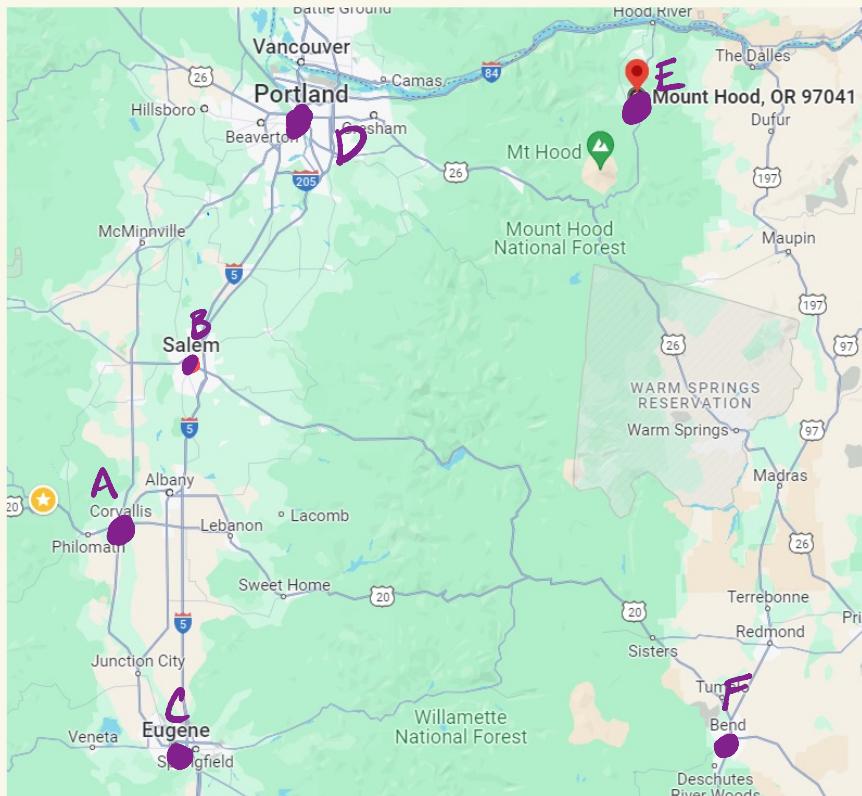


ADD NODES



```
GRAPHNODE A {"CORVALLIS"};
```

```
GRAPHNODE B {"SALEM"};
```

```
GRAPHNODE C {"EUGENE"};
```

```
GRAPHNODE D {"PORTLAND"};
```

```
GRAPHNODE E {"MT HOOD"};
```

```
GRAPHNODE F {"BEND"};
```



```
GRAPH ADD-NODE(A);
```

```
GRAPH ADD-NODE(F);
```

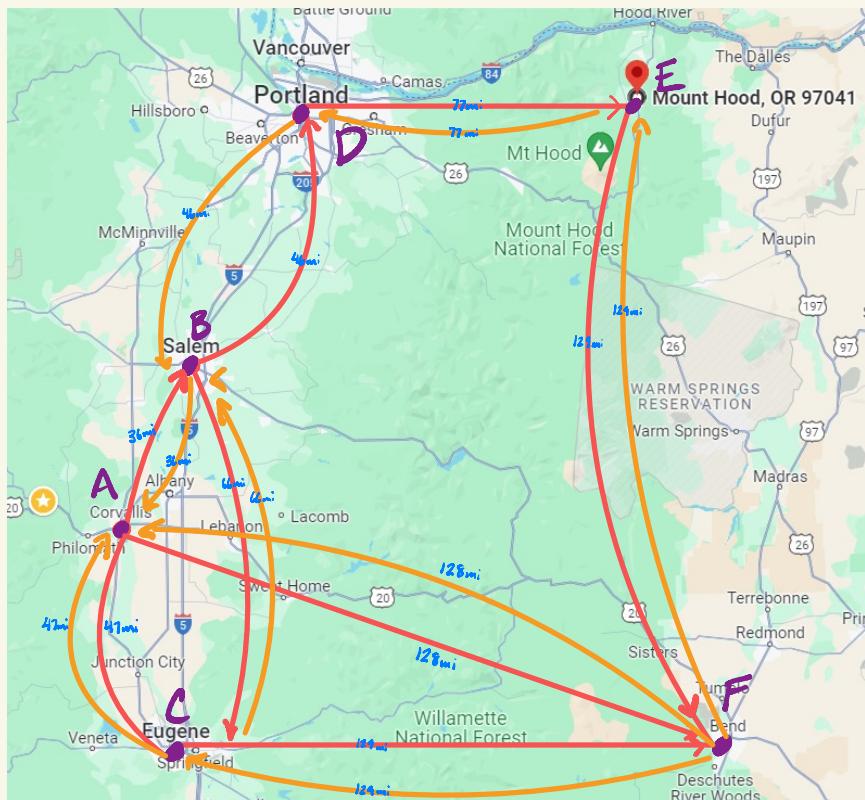


```
GRAPH.h  
ADD-NODE (GRAPHNODE *NEW-NODE){  
    Nodes.pushBack(NEW-NODE);  
}
```



MAIN TERMINAL
A. NAME: "CORVALLIS"
F. NAME: "BEND"

ADD EDGES & WEIGHTS



MAIN.CPP

EDGE AF {128, &A, &F};

EDGE FA {128, &F, &A};



Graph.h

GRAPH_ADD_EDGE(&AF);

ADD_EDGE(EDGE *NEW_EDGE) {

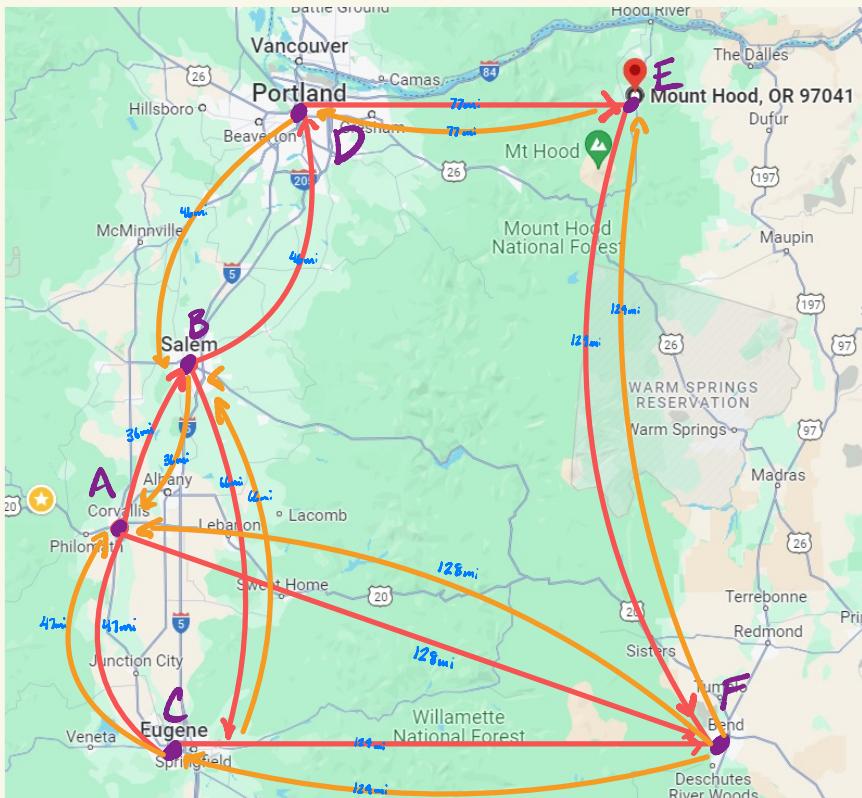
 NEW_EDGE->SOURCE->NEIGHBOR.push_back(VAL_EDGE);
}

MAIN TERMINAL

AF (WEIGHT, SRC->NAME, DST->NAME): (128, CORVALLIS, BEND)

FA (WEIGHT, SRC->NAME, DST->NAME): (128, BEND, CORVALLIS)

SHORTEST PATH ALGORITHM (DIJKSTRA's)



SHORTEST PATHS FROM CORVALLIS [mi]

Unvisited					
1	B	C	D	E	F
2		C	D	E	F
3		C		E	F
4		C			F
5		C			
6					

Shortest Path					
A	B	C	D	E	F
0	36	47	∞	∞	128
0	36	47	(46+36)	∞	128
0	36	47	82	(77+46+36)	128
0	36	47	82	159	128
0	36	47	82	159	128
0	36	47	82	159	128

Visited					
A	B	D	E	F	C

TEST PROGRAM
AGAINST THESE VALUES

SHORTEST PATHS FROM EUGENE [mi]

Unvisited					
1	A	B	D	E	F
2	B	D	E	F	
3		D	E	F	
4		E	F		
5			F		
6					

Shortest Path					
A	B	C	D	E	F
47	66	0	∞	∞	129
47	66	0	∞	∞	129
47	66	0	(46+66)	∞	129
47	66	0	112	(77+46+66)	129
47	66	0	112	189	129
47	66	0	112	189	129

Visited					
C	A	B	D	E	F

FINDING SHORTEST PATH:

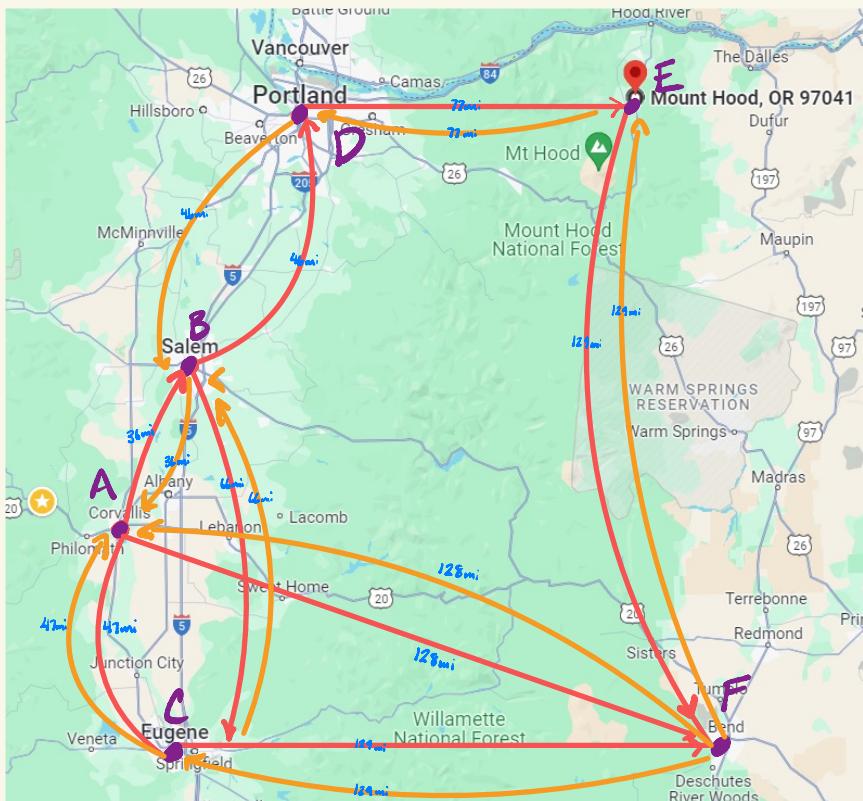
- SET DISTANCES FOR ALL NODES TO INFINITY
- SET DISTANCE TO THE START NODE TO ZERO (DISTANCE TO ITSELF)
- STORE UNVISITED NODES (CITIES) & WEIGHTS (DISTANCES) IN A QUEUE (Q)
- SET THE SOURCE NODE (DISTANCE = 0) ON TOP OF THE Q
- WHILE Q IS NOT EMPTY:
 - GET CURRENT NODE (SMALLEST DISTANCE) FROM TOP OF Q
 - POP() CURRENT NODE OFF Q
 - IF CURRENT NODE \rightarrow DISTANCE $>$ DISTANCE \rightarrow CURRENT NODE:
 - * SKIP THIS NODE, ALREADY HAVE A KNOWN SHORTER PATH
 - FOR ALL NEIGHBORING NODES OF CURRENT NODE (CONNECTED BY EDGE):
 - * DETERMINE NEW DISTANCE TO ALL NEIGHBORING NODES
 - IF NEW DISTANCE $<$ KNOWN DISTANCE:
 - * UPDATE SHORTEST DISTANCE TO NEIGHBOR
 - * PUSH() NEW DISTANCE & NEIGHBOR OUT TO THE Q
 - UPDATE NEW CURRENT NODE
- STORE EACH NODE & SHORTEST DISTANCE IN A VECTOR
- RETURN RESULT VECTOR

- CONSIDER HASH TABLE TO STORE DISTANCES?

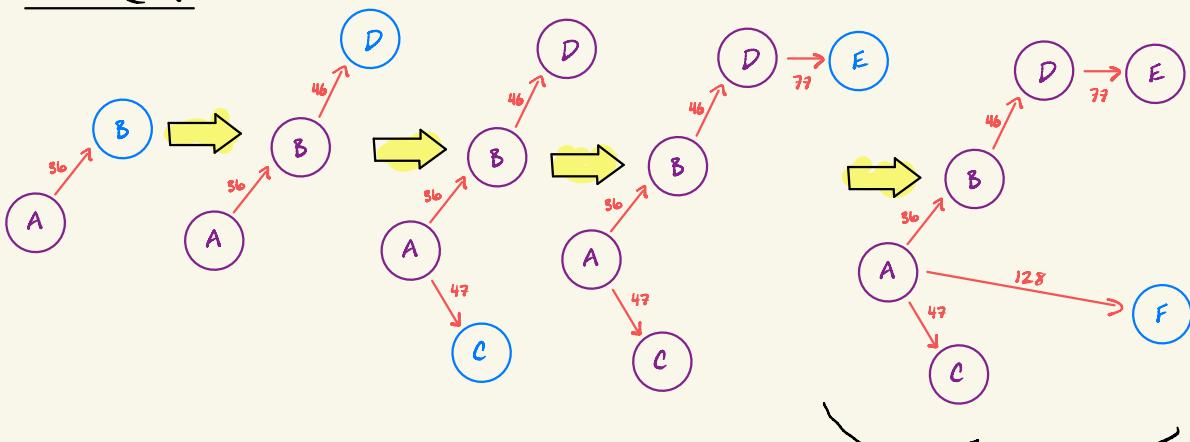
- KEY AS GRAPH NODE* & VALUE IS SHORTEST DISTANCE
- * STILL HAVEN'T TURNED IN ASSIGNMENT #7...

- NEED TO ORGANIZE Q, SHORTEST DISTANCE ALWAYS ON TOP

(Pim's) MST Algorithm - Test #1

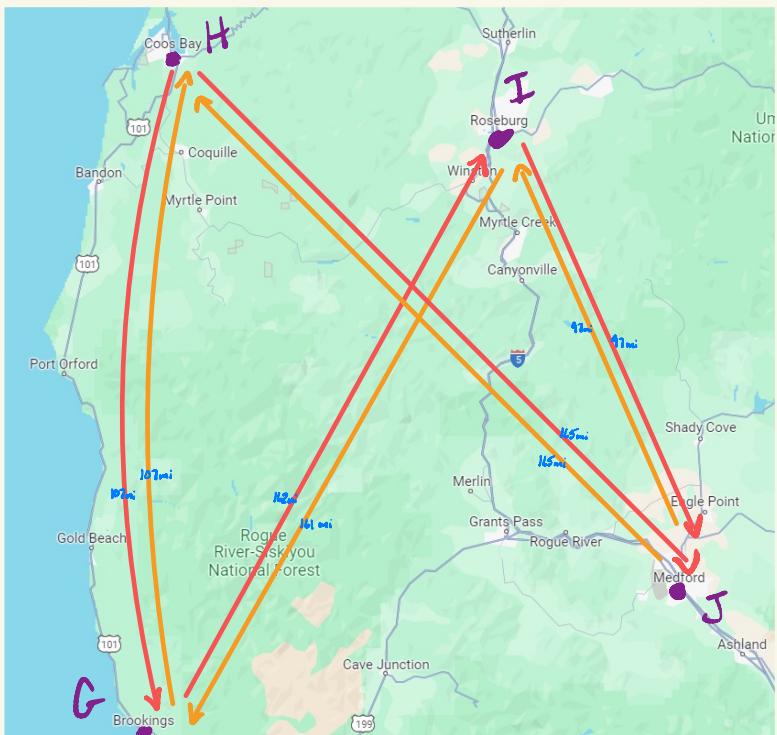


START @ 'A'

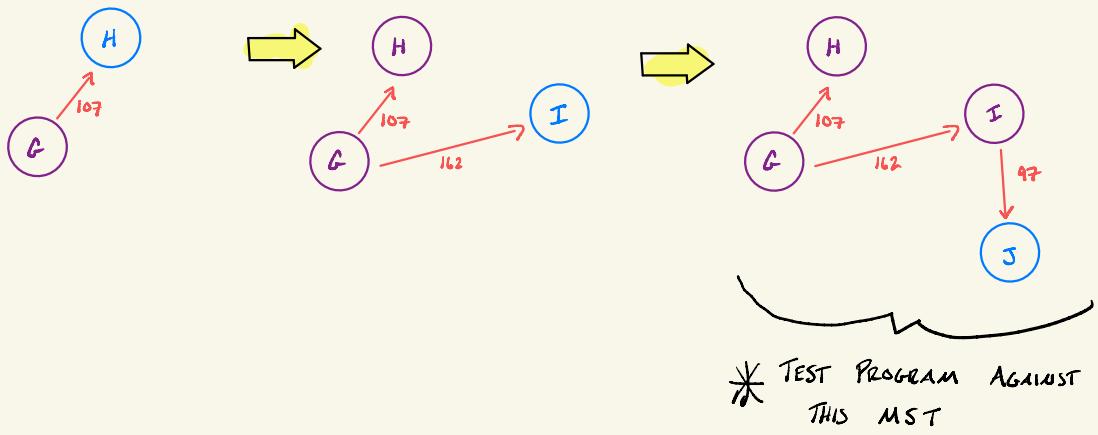


* TEST PROGRAM AGAINST
THIS MST

(Pm's) MST ALGORITHM - TEST #2



START @ 'G'



FINDING MST:

- VECTOR TO STORE EDGES OF MST
- Q TO STORE EDGE LENGTHS & EDGE* (ORDERED)
- HASH TABLE TO TRACK IF NODES ARE IN VECTOR? (KEY \rightarrow GraphNode*, VALUE \rightarrow T/F)?
 - * STILL NEED ASSIGNMENT $\neq \dots$
- ADD CURRENT NODE'S EDGES TO Q
- WHILE Q IS NOT EMPTY & VECTOR CONTAINS $<(NODES.size() - 1)$ EDGES:
 - FIND EDGE WITH SHORTEST DISTANCE IN Q
 - POP THAT EDGE FROM Q
 - IF DESTINATION NODE OF THIS IS ALREADY IN VECTOR:
 - * SKIP THIS EDGE, CANNOT CREATE A CYCLE
 - ADD EDGE TO THE VECTOR
 - ADD EDGE TO THE Q
 - RETURN VECTOR
- NEED ANOTHER FUNCTION TO ITERATE OVER NEIGHBORS, CHECK IF EDGES ARE IN VECTOR (T/F), AND ADD MISSING (F) EDGES TO Q
- NEED A WAY TO SORT Q (SHORTEST \rightarrow LONGEST EDGES)