Problem 1:

Function findFullyInwardVertex(A[v,v])

Set v to number of vertices in the graph

         Set i to 0, j to 0

         While i < v and j < v

         If A[i][j] is equal to 1

                 Increment i by 1

             Else

                 Increment j by 1

         End While

        If i > v

         Return “no solution”

       For j = 0, j < v, j++

If A[i][j] is equal to 1

                Return “no solution”

            If A[j][i] is equal to 0 and j is not equal to i

                Return “no solution”

        End For

      Return i

End Function

Problem 2:

CheckBipartite(G, s)

Create a new queue called q

Set the color of s to be Red

Add s to the queue q

While q is not empty

v = q.Pop()

For all neighbors w of v in graph G

If w does not have a color (unmarked)

If v.color = Red

Set w.color = Blue

Else

Set w.color = Red

End Else If

Add w to the queue q

Set Parent(w) = v

Else If v.color = w.color

Return false

End Else If

Return true

End Function

Problem 3:

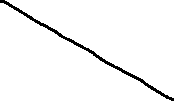
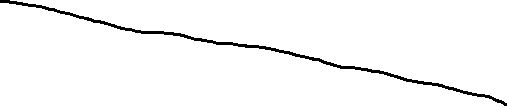
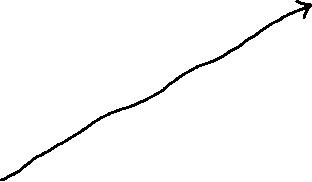
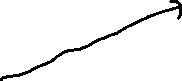
Part 1:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| V (**Explored**  **Vertex**) | S | X | Z | T | Y |
| **S** |  |  |  |  |  |
| **Z** |  |  |  |  |  |
| **T** |  |  |  |  |  |
| **X** |  |  |  |  |  |
| **Y** |  |  |  |  |  |

Part 2;

Let assume that the edge from y to t is now -4

The actual shortest dist(s,t) would be 4 but the shortest s-t distance returned by Dijkstra would be 6 since the path from s🡪z🡪y🡪t would not be considered because y🡪t has a negative weight.



Problem 4:

Part 1:

The running time of Dijkstra in big-O notation would be O(|)

For dense graphs Dijkstra is better. |E| > / log(|V|)

For parse graphs min-heap is better |E| < / log(|V|)

For E log(V) = graph the are the same

Part 2:

Don’t know how to do this

Problem 5:

Key property: d(x) + w(x,y) >\_(y)

Claim: For all vertices v in G, we will always have d(y) >\_ dist(s,y)

Proof by contradiction:

* Say the claim was false: will show contradiction, so claim must be true
* Consider first time that Dijkstra sets some d(y) < dist(s,y)
* D(y) must have been set during Explore(x), where y is in OUT(x) 🡪 d(y) = d(x) + w(x,y)

Initially when you start the claim is true d(s) = 0 =dist(s,s)

As we Explore(x): d(y) = d(x) + w(x,y)

Key argument is that y was the first bad vertex which this means y is not bad and d(x) >\_ dist(s,x) so d(y) >\_ dist(s,x) + w(x,y) >\_ dist(s,y) which will always be true.