Ryan Vollmer - CS 325 - HW 3

1) a) (A,E), (E,B), (B,C), (C,D), (C,G), (G,F), (D,H)

b) A, E, B, C, D, F, G, H

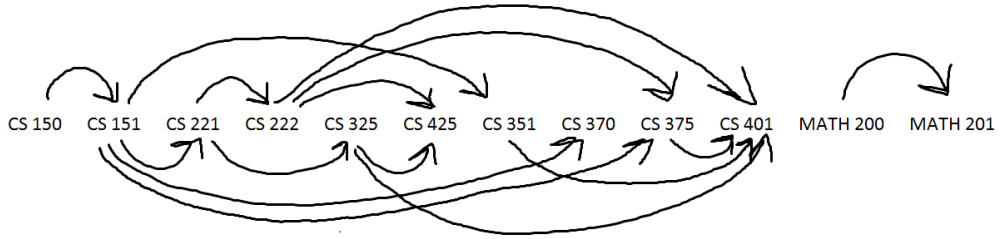
	A	B	C	D	E	F	G	H	chosen
0	0	inf	inf	inf	inf	inf	inf	inf	A
1	0	5	inf	inf	4	inf	inf	inf	E
2	0	5	inf	inf	4	15	inf	inf	B
3	0	5	11	inf	4	15	15	inf	C
4	0	5	11	12	4	15	15	38	D
5	0	5	11	12	4	15	15	24	F
6	0	5	11	12	4	15	15	24	G
7	0	5	11	12	4	15	15	24	H

2) By definition of a simple path on a DAG that includes every vertex, there must be every vertex connected to only a single other vertex. There will be only a single (the last one) vertex with no adjacent vertex, and only a single vertex with no parent (the first one). This also means the number of edges must be 1 less than the number of vertex. If it is not, then there must be more than a single edge going to a given vertex, making it no longer a simple path.

```
# graph G with G.V vertex, G.E edges
SIMPLE_HAMILTONIAN(G)
  count = 0;
  # number of edges must be 1 less than number of vertex (constant)
  if G.E.length != G.V.length - 1
    return false
  # else go through all vertex (V runtime)
  for v in G.V
      # and make sure there is 0 or 1 adjacent (constant)
      if v.adj.length == 0
          count = count + 1
          if count > 1
              return false
      else if v.adj.length > 1
          return false
  return true
```

All actions will take constant time other than looping through G.V, runtime will therefore be bounded above by O(V) (which is O(V+E)), and below by omega(1), as if it doesn't have the right number of edges/vertex it will run in constant time.

3) a)



b) CS 150, CS 151, CS 221, CS 222, CS 325, CS 351, CS 370, CS 375, CS 401, CS 425, MATH 200, MATH 201

c) CS 150, CS 151, CS 221, CS 222, CS 325, CS 351, CS 370, CS 375, CS 401, CS 425, MATH 200, MATH 201

d) CS 150, CS 151, CS 221, CS 222, CS 375, CS 401 => length of 5 (6 courses)

Started at CS 150, determined how many possible outcomes following each path would bring. After CS 325 it became impossible for longer length so stopped, as length of 5 already found, and only length of 4 remained (as know that last 2 are not connected).

4) a) start by coloring the first vertex a random color. We will then color all adjacent vertex the opposite color, if there are any vertex that are already the same color it is not possible. We will continue through all adjacent vertex until there are no more.

```
# graph G with G.V vertex, G.E edges
COLOR_GRAPH(G)
  S = empty set
  Q = empty Queue

  G.V[0].color = random 'red' or 'blue'  # set to red or blue randomly
  add G.V[0] to S
  Q.enqueue(G.V[0])

  while Q not empty
    cur = Q.dequeue()
    for v in Adj[cur]
      if v not in S
        if v.color == cur.color
          return "Not two colorable."
        if cur.color == 'red'
          v.color = 'blue'
        else
          v.color = 'red'
        add v to S
        Q.enqueue(v)
```

b) This is almost identical to BFS, in that the only modifications are constant time. It will therefore run in O(V+E).

5) a) I would use Dijkstra's shortest path algorithm

	A	B	C	D	E	F	G	H	chosen	parent
0	inf	inf	inf	inf	inf	inf	0	inf	G	-
1	inf	inf	9	7	2	8	0	3	E	G
2	inf	11	9	5	2	8	0	3	H	G
3	inf	6	9	5	2	8	0	3	D	E
4	inf	6	8	5	2	8	0	3	B	H
5	inf	6	8	5	2	8	0	3	F	G
6	12	6	8	5	2	8	0	3	C	D
7	12	6	8	5	2	8	0	3	A	C

(G,E), (G,H), (G,F), (H,B), (E,D), (C,A), (D,C)

b) Find the furthest point away from some vertex, repeat the same for that vertex, find the midpoint vertex of those two. Will modify Dijkstras to do this.

```
# G=graph, s=source
OPTIMAL_LOCATION(G, s)
  Q = []
  # get largest vertex, assumes a return
  # from Dijkstra of list of vertex
  v1 = Dijkstra(G, s)
  v2 = Dijkstra(G, max(v1))
  # iterate through parents to build list
  cur = max(v2)
  while cur != max(v1)
    add cur to Q
    cur = cur.p   # get next parent
  add max(v1) to Q
  # mid point is optimal, return that vertex
  return Q[floor(Q.length / 2)]
```

Run time will be cost of Dijkstras twice of O(ElgV) with assumption of binary heap used, they are independent of each other and the rest is constant other than the while loop, which is O(V). Therefore the total run time is O(ElgV).

c) E, because it clearly must be one of the points located in the center due to the relatively close relations between distances. If you have E as the midpoint the furthest points away are:

B=8, F=8, A=10

If you move that point to G then A=12 which makes G less optimal.
If you then choose D then B=11, which is also not optimal.
You could repeat this process for any other point, but the same results will appear.