IS609 HW7

J. Hamski

March 16, 2016

1

a. deg(G) = 9, which is odd. G is not Eulerian - you cannot traverse each edge exactly once and arrive at the vertex you originated from. In order to traverse each edge from a vertex and arrive back at it, you must go 'out and back' which implies 2 edges. Therefore, over the whole graph in order to go 'out and back' in any order there must be an even number of edges.

b. Yes, via the following route (one of several): 5-3 3-1 1-2 2-3 3-4 4-5 5-6 6-4 4-2

The necissary condition to have a walk that traverses all edges once but doesn't necissarily wind up in the same location is: - have no more than one vertex with only one edge - have no adjacent vertices with odd numbers of connections - evens can be adjacent to evens or odds, odds must be adjacent to evens.

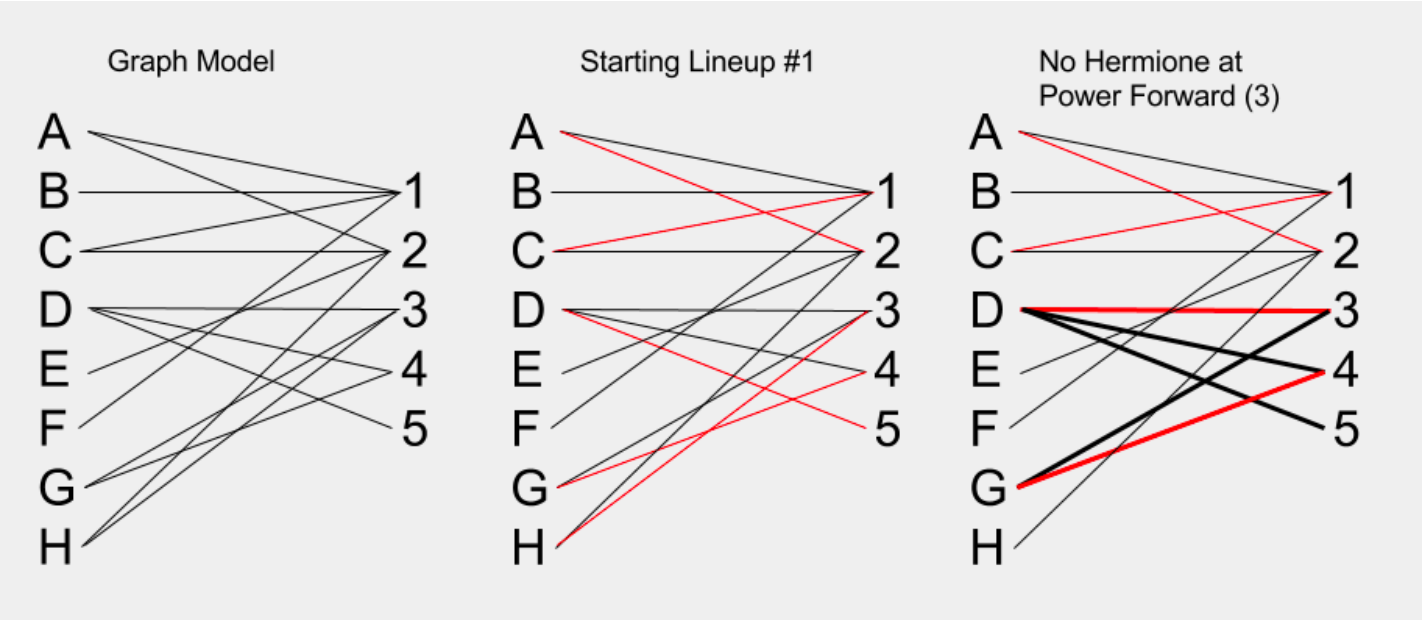
…there may be more, but these are what I found.

2

a. E(G) = {ab, ae, af, bd, bc, cd, df, de, ef}
b. {ab, bd, bc}
c. {b, d}
d. deg(a) = 3
e. |E(G)| = 9

I noticed that once you solved (a), you no longer need to refer to the drawn graph. All information is encapulated in the set describing the edges! It even becomes apparent how you could solve them algorithmically by iterating over the set E(G).

3



The team has a shallow bench at power forward (3), small forward (4), and center (5). So, if Hermione doesn't cover power forward, Deb and Gladys are the only ones left to cover the three 'big' positions of 3, 4, and 5, hence they cannot field a complete team.

4

First, create a matrix representation of the graph.

```
a <- c(0, 2, 4, 0, 0, 0, 0, 0, 0, 0)
b <- c(2, 0, 0, 2, 7, 0, 0, 0, 0, 0)
c <- c(4, 0, 0, 0, 4, 2, 0, 0, 0, 0)
d <- c(0, 2, 0, 0, 0, 0, 2, 0, 0, 0)
e <- c(0, 7, 4, 0, 0, 0, 1, 2, 3, 0)
f <- c(0, 0, 2, 0, 0, 0, 0, 0, 6, 0)
g <- c(0, 0, 0, 2, 1, 0, 0, 0, 0, 8)
h <- c(0, 0, 0, 0, 2, 0, 0, 0, 0, 8)
i <- c(0, 0, 0, 0, 3, 6, 0, 0, 0, 2)
j <- c(0, 0, 0, 0, 0, 0, 8, 4, 2, 0)

graph <- as.matrix(cbind(a, b, c, d, e, f, g, h, i, j))
row.names(graph) <- c("a", "b", "c", "d", "e", "f", "g", "h", "i", "j")
graph
```

```
##   a b c d e f g h i j
## a 0 2 4 0 0 0 0 0 0 0
## b 2 0 0 2 7 0 0 0 0 0
## c 4 0 0 0 4 2 0 0 0 0
## d 0 2 0 0 0 0 2 0 0 0
## e 0 7 4 0 0 0 1 2 3 0
## f 0 0 2 0 0 0 0 0 6 0
## g 0 0 0 2 1 0 0 0 0 8
## h 0 0 0 0 2 0 0 0 0 4
## i 0 0 0 0 3 6 0 0 0 2
## j 0 0 0 0 0 0 8 8 2 0
```

Using a package...

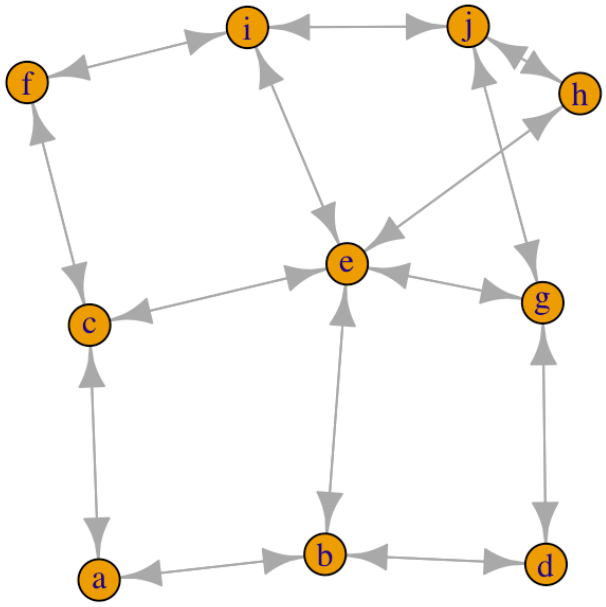
```
library(igraph)
```

```
##
## Attaching package: 'igraph'
```

```
## The following objects are masked from 'package:stats':
##
##     decompose, spectrum
```

```
## The following object is masked from 'package:base':
##
##     union
```

```
adjacency <- graph.adjacency(graph, weighted=TRUE)
plot(adjacency)
```




```
shortest.paths(adjacency, algorithm = "dijkstra")
```

```
##    a  b c d e f g h  i  j
## a  0  2 4 4 7 6 6 9 10 12
## b  2  0 6 2 5 8 4 7  8 10
## c  4  6 0 7 4 2 5 6  7  9
## d  4  2 7 0 3 9 2 5  6  8
## e  7  5 4 3 0 6 1 2  3  5
## f  6  8 2 9 6 0 7 8  6  8
## g  6  4 5 2 1 7 0 3  4  6
## h  9  7 6 5 2 8 3 0  5  4
## i 10  8 7 6 3 6 4 5  0  2
## j 12 10 9 8 5 8 6 4  2  0
```

The above shows the shortest path from a (0) to j (10) has a cost of 12.

This shortest path is:

```
get.shortest.paths(adjacency, from = 1, to = 10)$vpath
```

```
## [[1]]
## + 7/10 vertices, named:
## [1] a b d g e i j
```

```
get.paths <- function(node.array){
  paths <- which( node.array>0, arr.ind=TRUE)
  return(paths)}

get.costs <- function(node.array){
  costs <- node.array[node.array > 0]
  return(costs)}
```

```
path.matrix <- cbind(get.paths(graph), get.costs(graph))
colnames(path.matrix) <- c("row", "col", "cost")
path.matrix
```

```
##   row col cost
## b   2   1    2
## c   3   1    4
## a   1   2    2
## d   4   2    2
## e   5   2    7
## a   1   3    4
## e   5   3    4
## f   6   3    2
## b   2   4    2
## g   7   4    2
## b   2   5    7
## c   3   5    4
## g   7   5    1
## h   8   5    2
## i   9   5    3
## c   3   6    2
## i   9   6    6
## d   4   7    2
## e   5   7    1
## j  10   7    8
## e   5   8    2
## j  10   8    8
## e   5   9    3
## f   6   9    6
## j  10   9    2
## g   7  10    8
## h   8  10    4
## i   9  10    2
```

Now, I can traverse from col to row, with the cost associated with each path.

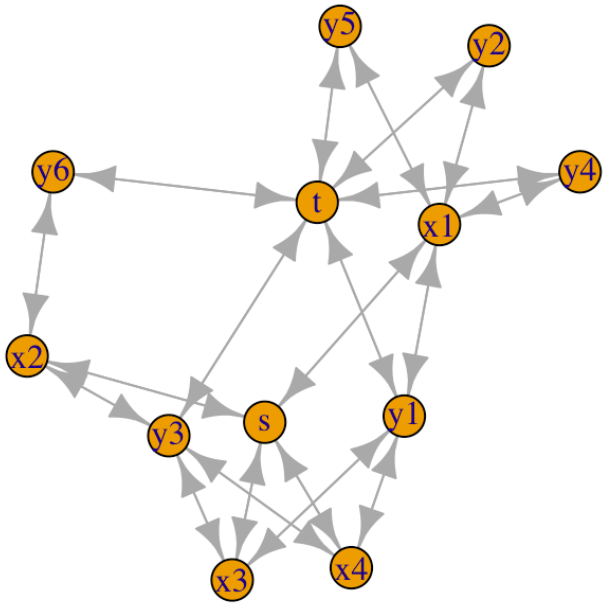
5

```
s <- c(0,1,1,1,1,0,0,0,0,0,0,0)
x1 <-c(1,0,0,0,0,1,1,0,1,1,0,0)
x2 <-c(1,0,0,0,0,0,0,1,0,0,1,0)
x3 <-c(1,0,0,0,0,1,0,1,0,0,0,0)
x4 <-c(1,0,0,0,0,1,0,1,0,0,0,0)
y1 <-c(0,1,0,1,1,0,0,0,0,0,0,1)
y2 <-c(0,1,0,0,0,0,0,0,0,0,0,1)
y3 <-c(0,0,1,1,1,0,0,0,0,0,0,1)
y4 <-c(0,1,0,0,0,0,0,0,0,0,0,1)
y5 <-c(0,1,0,0,0,0,0,0,0,0,0,1)
y6 <-c(0,0,1,0,0,0,0,0,0,0,0,1)
t <- c(0,0,0,0,0,1,1,1,1,1,1,0)

max.graph <- as.matrix(cbind(s, x1,x2,x3,x4,y1,y2,y3,y4,y5,y6,t))
max.graph
```

```
##        s x1 x2 x3 x4 y1 y2 y3 y4 y5 y6 t
##  [1,] 0  1  1  1  1  0  0  0  0  0  0 0
##  [2,] 1  0  0  0  0  1  1  0  1  1  0 0
##  [3,] 1  0  0  0  0  0  0  1  0  0  1 0
##  [4,] 1  0  0  0  0  1  0  1  0  0  0 0
##  [5,] 1  0  0  0  0  1  0  1  0  0  0 0
##  [6,] 0  1  0  1  1  0  0  0  0  0  0 1
##  [7,] 0  1  0  0  0  0  0  0  0  0  0 1
##  [8,] 0  0  1  1  1  0  0  0  0  0  0 1
##  [9,] 0  1  0  0  0  0  0  0  0  0  0 1
## [10,] 0  1  0  0  0  0  0  0  0  0  0 1
## [11,] 0  0  1  0  0  0  0  0  0  0  0 1
## [12,] 0  0  0  0  0  1  1  1  1  1  1 0
```

```
max.adjacency <- graph.adjacency(max.graph, weighted=TRUE)
plot(max.adjacency)
```



Maximum flow

```
graph.maxflow(max.adjacency, source = 1, target = 12)$value
```

```
## [1] 4
```

6

$$Maximize z = \sum_j x_{sj}$$

$$subject \rightarrow:$$

$$\sum_i x_{ij} = \sum_k x_{jk} \ \forall j \in V(G) - s, t$$

$$x_{ij} \leq u_{ij} \ \forall ij \in A(G)$$

$$x_{ij} \geq 0 \ \forall ij \in A(G)$$

$$x_{ab} \leq 2$$

$$x_{ac} \leq 6$$

$$x_{bc} \leq 2$$

$$x_{bd} \leq 4$$

$$x_{cd} \leq 1$$

$$x_{sa} \leq 4$$

$$x_{sb} \leq 5$$

$$x_{tc} \leq 7$$

$$x_{td} \leq 3$$