DATA 609: Homework 7

Modeling Using Graph Theory

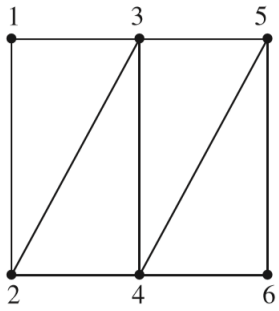
Aaron Grzasko

October 7, 2017

```
# load libraries
library(dplyr)
library(tidyr)
library(knitr)
library(kableExtra)
library(igraph)
```

Page 304, Question 2

The bridges and land masses of a certain city can be modeled with the graph G below:



Graph G

(a): Is G Eulerian? Why or why not?

Graph G is not an Eulerian circuit.

A Eulerian circuit is a sequence of adjacent vertices that meet the following conditions:

- The sequence begins and ends on the same vertex.
- The path traversal includes every vertex and edge.
- Each edge is visited exactly once.

There are two necessary criteria used to determine if a graph is Eulerian:

- Connectivity: There must be a path between every pair of vertices. In other words, there are no unreachable vertices. The graph in question satisfies this condition.
- All vertices have even degrees: Each vertex in an Eulerian graph have an even number of edges incident with it. Graph G fails this condition, as vertices 2 and 5 have three incident edges.

(b): Suppose we relax the requirement of the walk so that the walker need not start and end at the same land mass but still must traverse every bridge exactly once. Is this type of walk possible in a city modeled by graph G? If so, how? If not, why not?

Yes, this is possible in graph G.

An undirected graph that traverses each edge exactly once, but is not restricted otherwise is referred to as an Eulerian trail. This type of graph satisfies the following criteria:

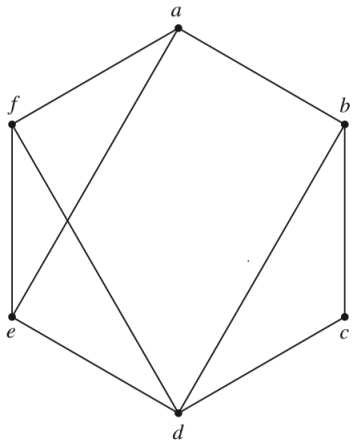
- The graph is connected. Graph G satisfies this condition.
- Zero or two vertices have an odd degree. Graph G also satisfies this criterion, as there are two vertices with odd degree.

Therefore, graph G represents an Eulerian trail.

Here is one of multiple possible solutions to this problem: 1-3-2-4-3-5-4-6-5

Page 307, Question 1

Consider the graph below:



Graph G

(a): Write down the set of edges E(G)

$$E(G) = \{ab, ae, af, bc, bd, cd, de, df, ef\}$$

(b): Which edges are incident with vertex b?

$$ab, bc, bd$$

(c): Which vertices are adjacent to vertex c?

Vertices b and d

(d): Compute deg(a).

$$\deg(a) = 3$$

In other words, there are three edges incident with vertex a: ab,ae, and af.

(e): Compute $|E(G)|$

The number of edges in graph G, i.e. $|E(G)|$, corresponds to the number elements in set $E(G)$:

$$|E(G)| = 9$$

Page 320, Question 10

A basketball coach needs to find a starting lineup for her team. There are five positions that must be filled: point guard (1), shooting guard (2), swing (3), power forward (4), and center (5). Given the the following table, create a graph model and use it to find a feasible starting lineup.

```
# data
people <- c('Alice','Bonnie', 'Courtney','Deb','Ellen','Fay','Gladys','Hermione')
positions <- c('1,2','1','1,2','3,4,5', '2', '1', '3,4', '2,3')
mydf <- data.frame(people = people, positions = positions)

kable(mydf)
```

people	positions
Alice	1,2
Bonnie	1
Courtney	1,2
Deb	3,4,5
Ellen	2
Fay	1
Gladys	3,4
Hermione	2,3

Let's build a graph model:

```
# create relationships of undirected graph, matrix form
A <- matrix(c(1,1,0,0,0,1,0,0,0,0,1,1,0,0,0,0,0,1,1,1,0,1,0,0,0,1,0,0
              ,0,0,0,0,1,1,0,0,1,1,0,0),nrow=8, byrow=TRUE)
row.names(A) <- c('A','B','C','D','E','F','G','H')
colnames(A) <- 1:5
A
```

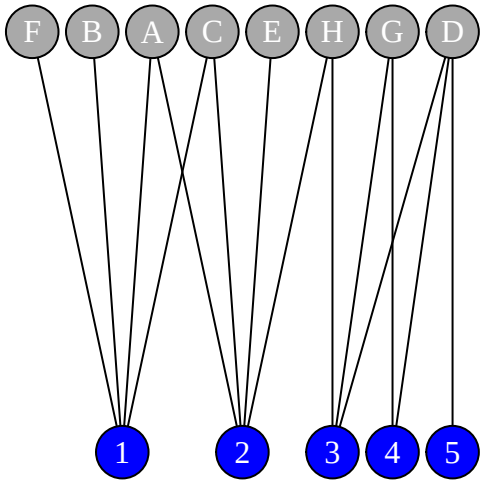
```
  1 2 3 4 5
A 1 1 0 0 0
B 1 0 0 0 0
C 1 1 0 0 0
D 0 0 1 1 1
E 0 1 0 0 0
F 1 0 0 0 0
G 0 0 1 1 0
H 0 1 1 0 0
```

```
# make graph, examine properties
g <- graph_from_incidence_matrix(A)
g
```

```
IGRAPH 19c7b2a UN-B 13 14 --
+ attr: type (v/l), name (v/c)
+ edges from 19c7b2a (vertex names):
 [1] A--1 A--2 B--1 C--1 C--2 D--3 D--4 D--5 E--2 F--1 G--3 G--4 H--2 H--3
```

```
# plot graph
V(g)$color <- V(g)$type
V(g)$color=gsub("FALSE","darkgray",V(g)$color)
V(g)$color=gsub("TRUE","blue",V(g)$color)
```

```
plot(g, edge.color="black", layout = layout_as_bipartite,
     vertex.size=25, vertex.label.color="white")
```



We can build a feasible lineup by visual examination of the graph:

Let's start with position 5 (Center) where there is only one viable option. We will then progress to the other positions:

- Position 5 (C): This must be filled by Deb because vertex D is the only vertex adjacent to vertex 5.
- Position 4 (PF): The only option is Gladys because Deb fills the center position.
- Position 3 (S): This position must be filled by Hermione because Deb and Gladys have been assigned.
- Position 2 (SG): This position can be filled by Alice, Courtney, or Ellen. Hermione has already been assigned.
- Position 1 (PG): Alice and Courtney are viable options, assuming neither of them are already assigned to position 2. Fay and Bonnie are available for this position regardless of the other position assignments.

Here is one possible solution:

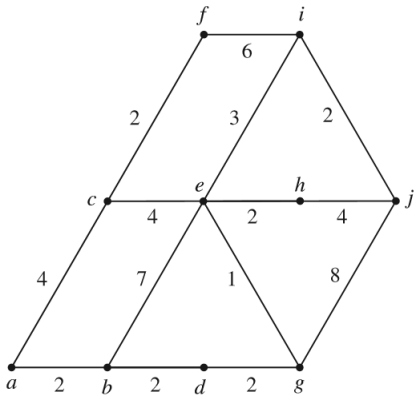
1 – F, 2 – A, 3 – H, 4 – G, 5 – D

What changes if the coach decides she can't play Hermione in position 3?

There is no feasible solution in this situation. While Deb and Gladys are both capable of playing the swing position, Deb must play center–she's the only player available for this position–and Hermione must play must play power forward given Deb's assignment. We are then left with no available players to play swing.

Page 331, Question 1

Find a shortest path from node a to node j with edge weights shown in the graph below.



Graph G

First, let's build a function to implement Dijkstra's Shortest-Path Algorithm in R.

```
# manual implementation of shortest path algorithm to calculate distance and path
dijkstra.sp <- function(mat, start, end) {

  # initialize distance labels
  L.V <- rep(Inf,nrow(mat)) # assume all distances infinite initially
  L.V[start] <- 0          # set dist of initial vertex at 0
  perm <- start            # make distance to initial vertex permanent
  path <- numeric(nrow(mat))
  while (TRUE) {
    for (p in perm) {         # loop through permanent nodes
      for (n in 1:ncol(mat)) {    # loop through potential adjacent nodes to p
        if(mat[p,n] != Inf & mat[p,n] != 0 & !(n %in% perm)) {  # if node is adjacent
          tempval <- L.V[p] + mat[p,n]      # tempdist = dist(start, p) + dist(p,n)
          if (tempval < L.V[n]) {
            L.V[n] <- tempval
            path[n] <- c(p)
          }
          L.V[n] <- min(tempval,L.V[n])   # if tempdist < current label, update L.V
        }

      }
    }
    temp_flag <- !(1:length(L.V) %in% perm)   # determine if nodes in L.V are temp

    # replace perm node values with infinity in L.V
    L.V_temp <- ifelse(L.V * temp_flag == 0, Inf, L.V * temp_flag)
    min_pos <- match(min(L.V_temp),L.V_temp)  # find position of minimum temp label
    perm <- c(perm, min_pos)                  # update perm vector with with min_pos

    if (min_pos == end)  break                # stop if reach end node

  }
```

```
    # recreate full path progression, in backwards order
    loc <- end
    full_path <- end
    while (loc != start) {
      loc <- path[loc]
      full_path <- c(full_path,loc)

    }
    full_path <- rev(full_path)   # full path, from start to finish

    # return distance and path as list
    answer <- list(L.V[end], full_path)
    names(answer) <- c("distance","path")
    answer
}
```

Now let's solve for the shortest path from node a to j in the graph above.

```
# initially assume infinite distance between two non-adjacent nodes
# assume distance between node and itself is 0

# edge distances
r1 <- c(0,2,4,Inf,Inf,Inf,Inf,Inf,Inf,Inf) # a
r2 <- c(2,0,Inf,2,7,Inf,Inf,Inf,Inf,Inf) # b
r3 <- c(4,Inf,0,Inf,4,2,Inf,Inf,Inf,Inf) # c
r4 <- c(Inf,2,Inf,0,Inf,Inf,2,Inf,Inf,Inf) # d
r5 <- c(Inf,7,4,Inf,0,Inf,1,2,3,Inf) # e
r6 <- c(Inf,Inf,2,Inf,Inf,0,Inf,Inf,6,Inf) # f
r7 <- c(Inf,Inf,Inf,2,1,Inf,0,Inf,Inf,8) # g
r8 <- c(Inf,Inf,Inf,Inf,2,Inf,Inf,0,Inf,4) # h
r9 <- c(Inf,Inf,Inf,Inf,3,6,Inf,Inf,0,2) # i
r10 <- c(Inf,Inf,Inf,Inf,Inf,Inf,8,4,2,0) # j

# store in matrix form

A <- rbind(r1,r2,r3,r4,r5,r6,r7,r8,r9,r10)
row.names(A) <- letters[1:10]
colnames(A) <- letters[1:10]

# beginning and end nodes in letter format
beg_lt <- 'a'
end_lt <- 'j'

# convert node letter to number
beg_num <- match(beg_lt,letters)
end_num <- match(end_lt, letters)

# solve for shortest distance between specified nodes
dijkstra.sp(A,beg_num,end_num)$distance
```

```
[1] 12
```

```
# show path
letters[dijkstra.sp(A,beg_num,end_num)$path]
```

```
[1] "a" "b" "d" "g" "e" "i" "j"
```

Finally, let's check our work using functions from the igraph package:

```
# create graph, using previously created matrix A
g <- graph_from_adjacency_matrix(A, mode = 'undirected', weighted = TRUE)

# find a shortest path
shortest_paths(g, 'a','j')$vpath
```

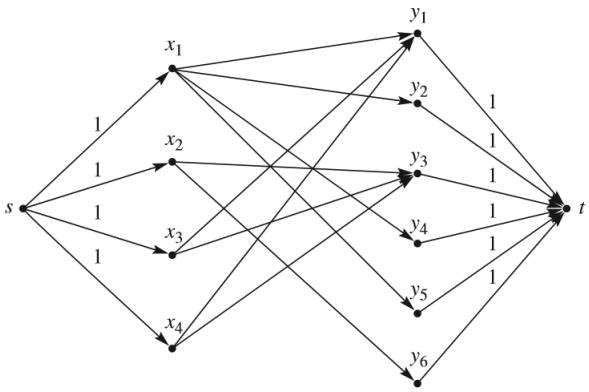
```
[[1]]
+ 7/10 vertices, named, from 19f0db3:
[1] a b d g e i j
```

```
# find distance of shortest path
distances(g,'a','j', algorithm = "dijkstra")
```

```
   j
a 12
```

Page 330, Question 3

Use our maximum-flow algorithm to find the maximum flow from s to t in the graph



Graph G

Based on the diagram, we know that maximum flow can be no larger than

$$min(|X|, |Y|) = min(4, 6) = 4$$

We can now specify a path with flow of 4:

Initial flow, $f_c \leftarrow 0$

1. Directed path: $s - x_2 - y_6 - t$; $f_c = 1$; using x_2y_6
2. Directed path: $s - x_1 - y_2 - t$; $f_c = 2$; using x_1y_2
3. Directed path: $s - x_3 - y_1 - t$; $f_c = 3$; using x_3y_1
4. Directed path: $s - x_4 - y_3 - t$; $f_c = 4$; using x_4y_3

We can also verify our work using `max_bipartite_match()` in the igraph package.
See below: we confirm that the maximum matching size is 4, but the routine actually found a different set of x and y combinations; so new we have found two unique matching combinations.

```r
# set up graph
r1 <- c(1,1,0,1,1,0)
r2 <- c(0,0,1,0,0,1)
r3 <- c(1,0,1,0,0,0)
r4 <- c(1,0,1,0,0,0)
A <- rbind(r1,r2,r3,r4)
row.names(A) <- c("x1","x2","x3","x4")
colnames(A) <- c("y1","y2","y3","y4","y5","y6")
g <- graph_from_incidence_matrix(A, directed = TRUE)

max_bipartite_match(g)
```

```
$matching_size
[1] 4

$matching_weight
[1] 4

$matching
  x1   x2   x3   x4   y1   y2   y3   y4   y5   y6
"y5" "y6" "y1" "y3" "x3"   NA "x4"   NA "x1" "x2"
```

References

- Wikipedia entry on Eulerian Paths: https://en.wikipedia.org/wiki/Eulerian_path
- Epp, S.S. (2004). *Discrete Mathematics with Applications (3rd ed.) (pp 667 - 676).* Boston, MA: Brooks/Cole.

- Inspiration for manual shortest path calculation: https://uqkdhanj.wordpress.com/2015/02/10/dijkstras-shortest-pathway-algorithm/

- Bipartite graphs in igraph package: https://rpubs.com/lgadar/load-bipartite-graph