Coloring the Map of Paris

Raphaella Xie

10/21/2022

1. Part I: Introduction

1a. Problem Description

I want to color the map of the 20 arrondissements (administrative districts) in Paris with the fewest colors. In this map, each region must be colored with exactly one color.
The following below is the map of the 20 arrondissements in Paris.

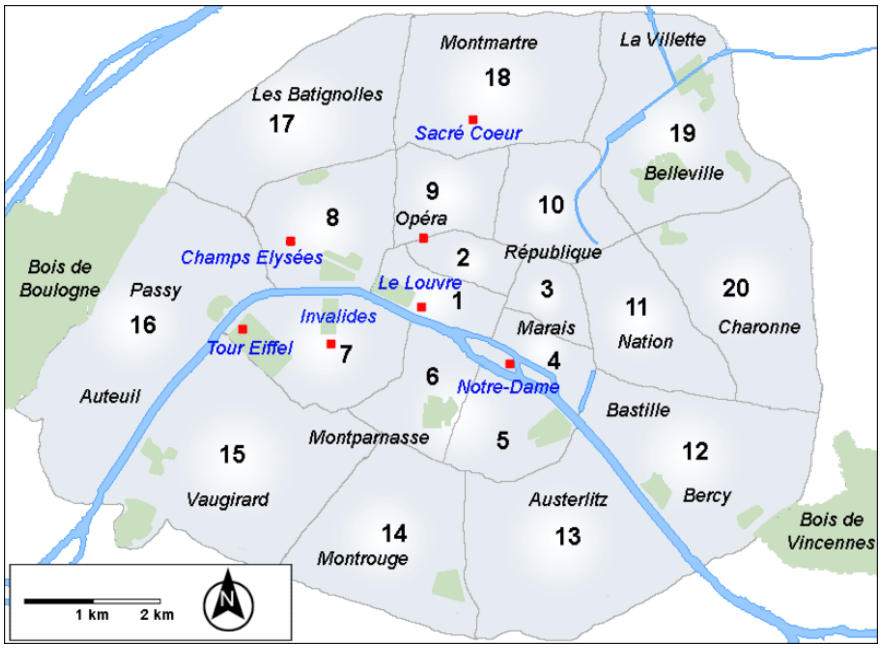

Figure 1: The map of Paris [1]

To solve this problem, I want to design a graph in which each arrondissment corresponds to a vertex, and there is an edge between two vertices if these two regions represented by the vertices shared a border.

1b. Design the Graph

To define our graph, let's suppose an undirected graph $G = (V, E)$ with V represents the set of vertices and E represents the set of edges.

[1] The map is from the link: https://fr.m.wikipedia.org/wiki/Fichier:Arrondissements-de-Paris.png

Since this the Paris has 20 regions, so there are exactly 20 elements in the set V.
For $v \in \{1, 2, 3, ...20\}$ we use v_i to represent the vertex of the region i.
Then we can express V as $V = \{v_1, v_2, \ldots, v_{20}\}$.

For $i, j \in \{1, 2, 3, ...20\}$ and $i \neq j$, we use (v_i, v_j) to represent the edge between vertex i and vertex j.
Since this is a basic graph, so there is no difference between edge (v_i, v_j) and edge (v_j, v_i).
Based on the map above, I can count the edges in the set E.
Before counting the edges, I want to clarify a convention that two regions that borders at a point can be colored with the same color.
In other word, we consider there is no edge between two regions that borders at a point.
In this map, I found four cases that four arrondissements border at a single point.
These cases include Arrondissements 1, 4, 5, and 6; Arrondissements 4, 5, 12, and 13; Arrondissements 8, 9, 17, and 18; Arrondissements 10, 11, 19, 20.
Let me use the first case as an example.
In this case, these four Arrondissements border at a point on the southwestern riverside of island.
Based on the convention, we consider there is no edge between v_1 and v_5, and there is also no edge between v_4 and v_6.
Keeping the convention in mind, I can find all the edges in the set E:

$$E = \left\{ \begin{array}{l} (1,2),(1,3),(1,4),(1,6),(1,7),(1,8),(1,9),(2,3),(2,9),(2,10),(3,4),(3,10),(3,11), \\ (4,5),(4,11),(4,12),(5,6),(5,13),(5,14),(6,7),(6,14),(6,15),(7,8),(7,15),(7,16), \\ (8,9),(8,16),(8,17),(9,10),(9,18),(10,11),(10,18),(10,19),(11,12),(11,20), \\ (12,13),(12,20),(13,14),(14,15),(15,16),(16,17),(17,18),(18,19),(19,20) \end{array} \right\}$$

With the set of vertices V and the set of edges E, I drew a graph for the twenty Arrondissements in Paris.
For the convenience of checking whether my graph exactly represent the map, the map of Paris is provided on the left.
The followings below shows the map of Paris and the graph for Paris.

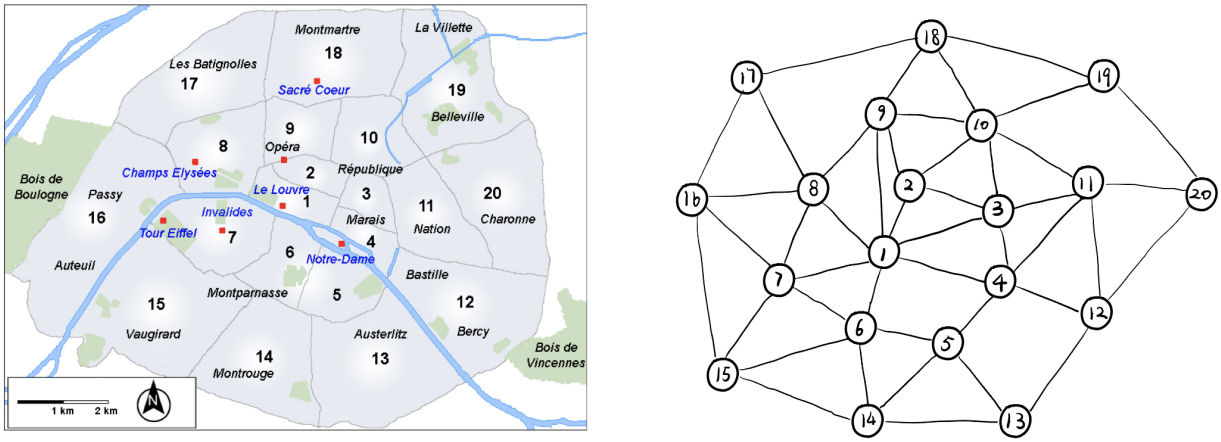

Figure 2: The map and graph of Paris

With the graph, I want to design a LP to solve this problem.

2. LP Formation

2a. Constraints

Based on the constraints of this problem, let's suppose we have s colors.
To define our LP, I firstly use a binary variable y_k with $k = 1, 2, \ldots, s$ to indicate whether we use the color

k.
That is, $y_k = 1$ if and only if we used color k; otherwise we did not use color k.
For $i = 1, 2, \ldots, 20$ and $k = 1, 2, \ldots, s$, let's also define a binary variable x_{ik} with $x_{ik} = 1$ if and only if we used color k for vertex i; otherwise we did not use color k.

Based on the graph, we found that vertices $v_1, v_6, v_{15}, v_{16}, v_8$, and v_7 forms wheel graph W_6 that centered at v_7.
In this wheel graph, let's suppose we colored vertices v_7 and v_8 with Color 1 and 2.
Since we require that the adjacent regions must use different colors, so we may color vertex v_1 with Color 3, vertex v_6 with Color 2, vertex v_15 with Color 3.
However, for the vertex v_{16}, we found that its three adjacent vertices already used three different colors.
That is, vertex v_8, v_7, and v_{15} used Colors $2, 1$, and 3 respectively.
In this case, we have to let v_{16} uses Color 4.
Based on this wheel graph w_6, we may set 4 as the lower bound for s, the total number of colors we use.

Since we want to minimize the number of colors we used, so the objective function would be the sum of y_k.
We can express our objective function as:

$$\sum_{i=1}^{s} y_i.$$

We want to minimize the objective functions subject to the following constraints.
I express our LP as:

Minimize $\sum_{i=1}^{s} y_i$ subject to

$$\sum_{k=1}^{s} x_{ik} = 1, \quad i = 1, 2, \ldots, 20 \tag{1}$$

$$x_{ik} \leq y_k, \quad i = 1, 2, \ldots, 20 \text{ and } k = 1, 2, \ldots, s \tag{2}$$

$$x_{ik} + x_{jk} \leq 1 \text{ for all } (v_i, v_j) \in E \text{ and } k = 1, 2, \ldots, s \tag{3}$$

$$\sum_{k=1}^{s} y_k \leq 4 \tag{4}$$

$$y_{ik}, x_{ik} \in \{0, 1\}, \ i = 1, 2, \ldots, 20, k = 1, 2, \ldots, s \tag{5}$$

The Constraint `(1)` requires that each region can at most be colored with exactly one color.

The Constraint `(2)` requires that all the regions can not use unused color.
That is, for $i = 1, 2, \ldots, 20$ and $k = 1, 2, \ldots, s$, if $x_{ik} = 1$, then y_k must be 1.
If $x_{ik} = 0$, then there is no restriction on y_k.

The Constraint `(3)` requires that the adjacent regions can not share the same color.
This means for two adjacent vertices v_i and v_j, if $x_{ik} = 1$, then $x_{jk} \neq 1$.

To speed up the computations, I introduces Constraint `(4)`, which can reduce the combinations of colors that `lp_solve` will try while doing the calculation.
For example, suppose we use 3 colors out of 4 colors to color a graph, then there could be four possible combinations of colors.
These combinations include $\{1, 2, 3\}$, $\{2, 3, 4\}$, $\{1, 2, 4\}$, and $\{1, 3, 4\}$.
With Constraint `(4)`, `lp_solve` only need to try the combination $\{1, 2, 3\}$.
This greatly reduced the amount of calculations.

At last, Constraint (5) requires all of our variables to be binary.

With the objective function and these constraints, solving this LP can find the minimum number we need to use to color the map of the twenty Arrondissements of Paris.

3. Solutions

With the LP above, I can use R codes to print a file that describe these constraints.
As we discussed in previous section, we set the lower bound for the total number of colors s to be 4.
Then I started with setting $s = 4$ in my R code to see whether my problem can be solved by lp_solve.
(The R codes, lp_solve inputs and outputs can be viewed in the Appendix).

Based on the output from lp_solve, I found that the minimum number of colors we need to color the map for the twenty Arrondissements of Paris is 4.
While we know that we at least need to use 4 colors to color the map, so we can conclude that this results is the optimal solutions.
The followings below are the colored map and the colored graph of the twenty Arrondissements of Paris.

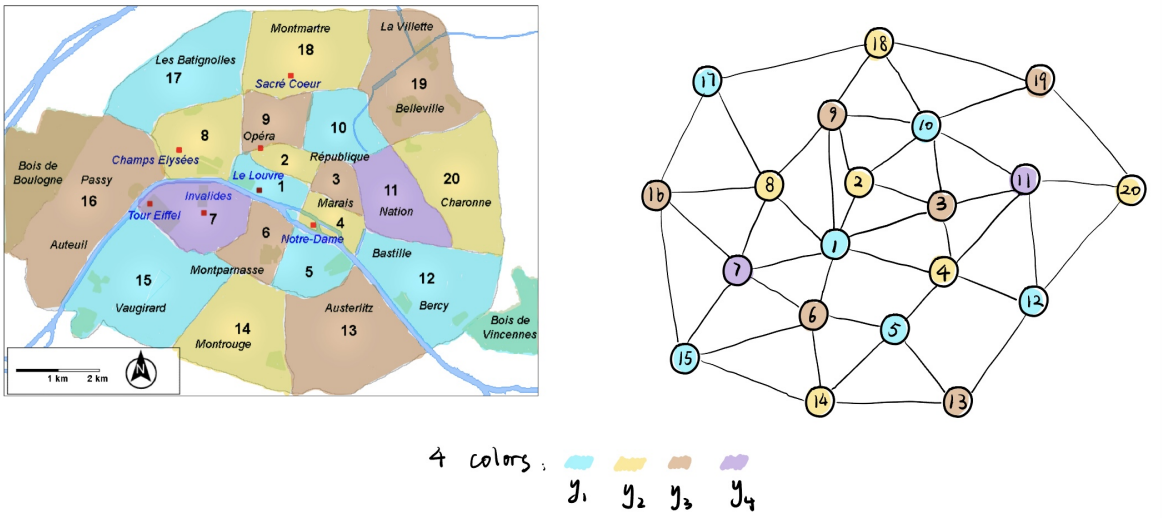

Figure 3: The colored map and colored graph of Paris [2]

Based on the graph, we can validate that all the adjacent regions have different colors.

4. Part II: New Constraint

4a. Problem Description and LP

Based on the current constraints, I want to add an additional constraint that two regions that border the same region cannot be colored with the same color.

[2]Note that on the map of Paris, the areas Bois de Vincennes in Arrondissement 12 and Bois de Boulogne in Arrondissement 16 are originally colored with green color to represent the forestation.
These overlapping colors create a color blend.

This means for all $i, j, m = 1, 2, ..., 20$, and $i \neq j \neq m$, if $(v_i, v_j), (v_j, v_m) \in E$, then vertices v_i, v_j, and v_m can not have the same colors.
For example, if vertex v_i used color k, then $x_{ik} = 1$ and $x_{jk} = x_{mk} = 0$.
This constraint can be expressed as

$$x_{ik} + x_{jk} + x_{mk} \leq 1 \quad \text{for all } (v_i, v_j), (v_j, v_m) \in E \text{ and } k = 1, 2, \ldots, s \tag{6}$$

With this new constraint, we also found that for a vertex v_i with d adjacent vertices, the adjacent vertices of v_i must have different colors.
At the same time, these adjacent vertices must also have colors different from v_i.
In this case, we need at least $(1 + d)$ colors to color the graph.
In this graph, the vertex v_1 has the greatest degree, 7.
This means we at least 8 colors to color this graph.
Then I set 8 as the lower bound of colors we need to use.
I express this constraint as:

$$\sum_{k=1}^{s} y_k \geq 8. \tag{7}$$

Then our LP would simply be minimizing the objective function subject to Constraint `(1)`, `(2)`, `(3)`, `(4)`, `(5)`, `(6)`, and `(7)` as described above.

4b. Solutions

With my new LP, I can use `R` codes to print my LP into files.
Then I used `lp_solve` to compute the solutions based on this file.
Based on the output, I found that the minimum number I need to color the map is 8.
The following figure is the colored map and graph under the new constraints.

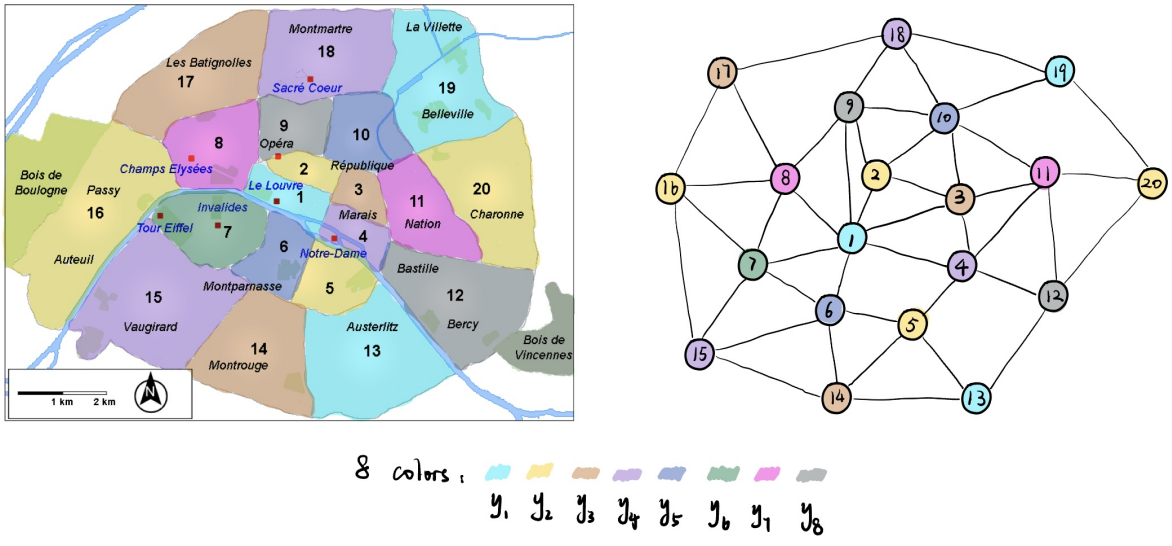

Figure 4: The colored map and colored graph of Paris

Based on the graph, we can validate that all the adjacent regions have different colors and regions that border the same region use different colors.

Since we know that the lower bound for the total number of colors is 8, so we can conclude that this coloring is the optimal results.

5. Part III: Two Colors on Every Region

5a. Problem Description and LP

Now, based on the rules of `Part I`, I want to slightly change my way of graph coloring.
I want to color my map in a way that every region must be colored with exactly two colors.
That for every vertex i on the graph, $\sum_{k=1}^{s} x_{ik} = 2$ where s is the total number of colors we can use.
Then I removed Constraint `(1)` and expressed this new constraint as:

$$\sum_{k=1}^{s} x_{ik} = 2, \quad i = 1, 2, \ldots, 20 \tag{8}$$

I also want to set a bound of the total number of colors to speed up the calculation.
Based on the results from `Part I` that we need a maximum of four colors to color the map when every region is colored with exactly one color and adjacent regions have different colors. Let's suppose we use another different four colors to color the map of the twenty Arrondissements in Paris.
Then we overlap this new colored map with the colored map in `Part I`.
In this case, we found that we only need at most eight colors to color the map such that every region is colored with two colors while adjacent regions have different colors.
Then I can set the the upper bound of the number of colors $s = 8$.

At the same time, for the wheel graph W_6 that centered at vertex v_7, let's suppose we color the vertex v_7 with Color 1 and 2.
Based on the constraints in this problem, let's also color the vertex v_8 with Color 3 and 4;
color the vertex v_1 with Color 6 and 7; color the vertex v_6 with Color 4 and 5;
color the vertex v_{15} with Color 3 and 6; color the vertex v_{16} with Color 5 and 7.
In this way, we need 7 colors to color the wheel graph W_6.
This also means we may need 7 or even more colors to color the whole graph.
The following figure shows the way I color the wheel graph W_6.

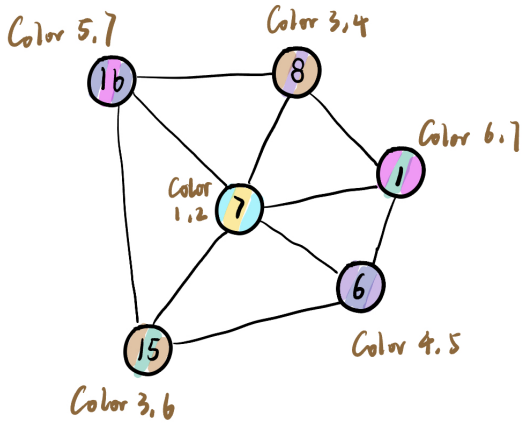

Figure 5: The possible coloring of W_6 uses 7 different colors

Based on the discussion above, in my `R` code, I set the number of colors $s = 8$, which is the upper bound of the number of colors.
I design my LP as minimizing the objective function subject to the Constraints `(2), (3), (4), (5), (8)`

5b. Solutions

With this new LP, I adjusted my `R` codes and used `lp_solve` to compute the solutions.
Based on the output, I found that the minimum number I need to color the map is 7.
Since we know that we need at least 7 colors to color the graph, thus my can conclude that this coloring is the best possible solution.
The following figure is the colored map and graph under the new constraints.

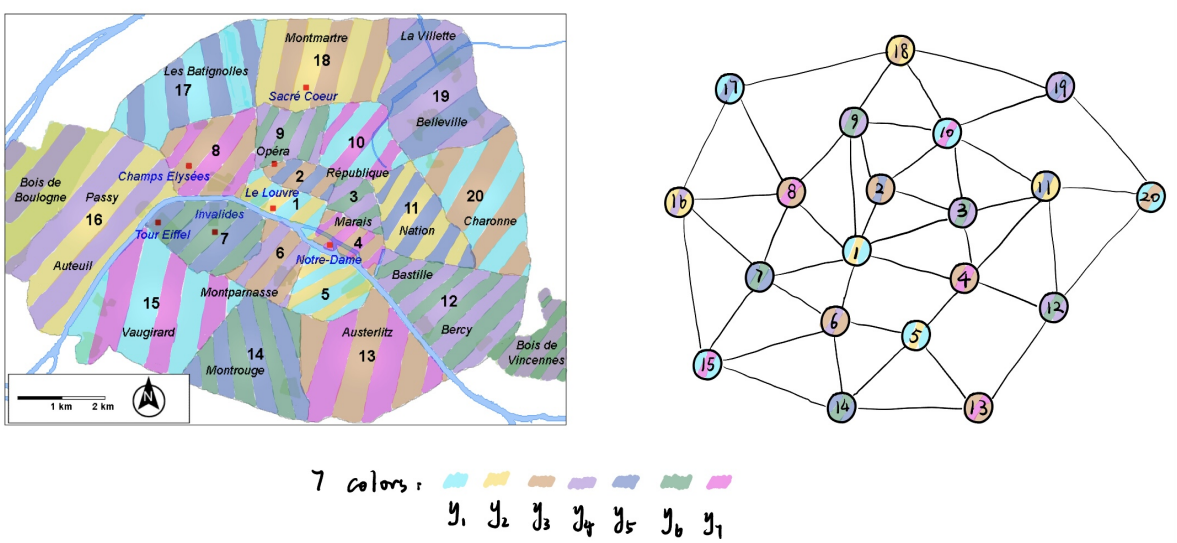

Figure 6: The colored map and colored graph of Paris

6. Apendix

6a. R Codes for Creating the Graph, Adjacency Matrix, and Other Functions

The following below is the `R` code I wrote to create the graph and adjacency matrix.
I used a adjacency matrix to represent the graph.

```
library(igraph)

# Set the working directory
setwd("~/Desktop/Math_381/osx64")

# Build a graph for the 20 Arrondissements in Paris
# There can at most be one edge between two vertices with no direction
#   Edge between region 1 and 2 is the same as between region 2 and 1,
#   Edge 1-2 can be expressed as either c(1,2) or c(2,1) exclusively

set.seed(381)
paris_graph <- graph( edges = c(1,2, 1,3, 1,4, 1,6, 1,7, 1,8, 1,9,
                        2,3, 2,9, 2,10,
                        3,4, 3,10, 3,11,
                        4,5, 4,11, 4,12,
```

```
                        5,6, 5,13, 5,14,
                        6,7, 6,14, 6,15,
                        7,8, 7,15, 7,16,
                        8,9, 8,16, 8,17,
                        9,10, 9,18,
                        10,11, 10,18, 10,19,
                        11,12, 11,20,
                        12,13, 12,20,
                        13,14,
                        14,15,
                        15,16,
                        16,17,
                        17,18,
                        18,19,
                        19,20),
                       directed= F)

V(paris_graph)$color <- "white"
V(paris_graph)$size <- 20

plot(paris_graph)

mat <- paris_graph[]

# The adjacency matrix to represent the graph
mat
```

6b. R codes, lpsolve input and output for Part I

(6b-1). The following is the `R` code I wrote to generate the input file for `lp_solve`.
Based on the upper bound we described above, I set $s = 4$ in this code.
The `R` code for functions to print constraints can be viewed in section `6e`.

```
# Part I

# Set the number of colors to be 4
s <- 4

# Print objective function
print_objf(s, output_file)

# Label vertex 1,2,3 with color 1,2,3
# This speeds up calculation
set_color_v123(output_file)

# Print Constraint 1
# All vertices must have exactly one color
print_color_per_vertex(s, 1, output_file)

# Print Constraint 2
# Each vertex can not have an unused color
print_cons2(s, output_file)
```

```
# Print Constraint 3
# Adjacent vertices must have different colors
print_cons3(s, output_file)

# Print Constraint 4
# Reduce the combinations of colors that need to check
# This makes the computation faster
print_cons4(s, output_file)

# Print Constraint 5
# Require all y_k and x_ik to be binary
print_cons5(s, output_file)
```

(6b-2). The following is the input file for `lp_solve` in `Part I`.

```
min: + y_1+ y_2+ y_3+ y_4;

(3 lines of the following types:
   assign vertex 1,2,3 with Color 1,2,3 respectively.)
label: x_1_1 = 1;

(20 lines of the following types:
   ensure all vertices have exactly one color.)
+x_1_1+x_1_2+x_1_3+x_1_4 = 1;

(80 lines of the following types:
   ensure each vertex can not have an unused color.)
x_1_1 <= y_1;

(352 lines of the following types:
   ensure adjacent vertices must have different colors.)
x_1_1 + x_2_1 <= 1;

(3 lines of the following types:
   ensure that we use the colors with index from the low to high.)
y_2 <= y_1;

(ensure all variables are binary.)
bin y_1, y_2, ..., y_4, x_1_1, x_1_2, ..., x_1_4, x_2_1, x_2_2, ..., x_20_4;
```

(6b-3). The following is the output from `lp_solve` in `Part I`.

```
Value of objective function: 4.00000000

Actual values of the variables:
y_1                             1
y_2                             1
y_3                             1
y_4                             1
x_1_1                           1
x_2_2                           1
x_3_3                           1
```

```
x_4_2                          1
x_5_1                          1
x_6_3                          1
x_7_4                          1
x_8_2                          1
x_9_3                          1
x_10_1                         1
x_11_4                         1
x_12_1                         1
x_13_3                         1
x_14_2                         1
x_15_1                         1
x_16_3                         1
x_17_1                         1
x_18_2                         1
x_19_3                         1
x_20_2                         1

real    0m0.055s
user    0m0.010s
sys     0m0.009s
```

6c. R codes, lpsolve input and output for Part II

(6c-1). The following is the `R` code I wrote to generate the input file for `lp_solve` for `Part II`.
In this part, we found the minimum number we need to color the map is 8.
In the code below, I used $s = 8$ as an example.

```
# Part II
#
# What if one can not share the same color with
#   the other zone that borders the same place?

output_file <- file("paris.txt", open = "w-")

# We found that the min number of color is 8
# This example sets s = 8 as a demonstration
s <- 8

# Print objective function
print_objf(s, output_file)

# Label vertex 1,2,3 with color 1,2,3
set_color_v123(output_file)

# Print Constraint 1
# All vertices must have exactly one color
print_color_per_vertex(s, 1, output_file)

# Print Constraint 2
# Each vertex can not have an unused color
print_cons2(s, output_file)
```

```
# Print Constraint 3
# Adjacent vertices must have different colors
print_cons3(s, output_file)

# Print Constraint 4
# Reduce the combinations of colors that need to check
# This makes the computation faster
print_cons4(s, output_file)

# Print Constraint 6
# Set the lower bound of total number of colors to be
#     1 + the maximum degree of the nodes.
#     That is, the upper bound is 1 + 7 = 8
bound <- " >= 8;"
print_bound(s, bound, output_file)

# Print Constraint 7
# Two regions that border the same region
#     can not be colored with the same color.
#     x_ik + x_jk + x_mk <= 1
print_cons7(s, output_file)

# Print Constraint 5
# Require all y_k and x_ik to be binary
print_cons5(s, output_file)
```

(6c-2). The following is the input file for `lp_solve` in `Part II`.

```
min: + y_1+ y_2+,...,+ y_8;

(3 lines of the following types:
   assign vertex 1,2,3 with Color 1,2,3 respectively.)
label: x_1_1 = 1;

(20 lines of the following types:
   ensure all vertices have exactly one color.
 )
+x_1_1+x_1_2+x_1_3+x_1_4+x_1_5+x_1_6+x_1_7+x_1_8 = 1;

(160 lines of the following types:
   ensure each vertex can not have an unused color.)
x_1_1 <= y_1;

(704 lines of the following types:
   ensure adjacent vertices must have different colors.)
x_1_1 + x_2_1 <= 1;

(7 lines of the following types:
   ensure that we use the colors with index from the low to high.)
y_2 <= y_1;

(set 8 as the lower bound for the number of colors.)
+ y_1 + y_2 + ... + y_8 >= 8;
```

```
(2560 lines of the following types:
      ensure that two regions that border the same region
      can not be colored with the same color.)
   x_10_2 + x_18_2 + x_9_2 <= 1;

(ensure all variables are binary)
bin y_1, y_2, ..., y_8, x_1_1, x_1_2, ..., x_1_8, x_2_1, x_2_2, ..., x_20_8;
```

(6c-3). The following is the output from `lp_solve` in `Part II`.

```
Value of objective function: 8.00000000

Actual values of the variables:
y_1                             1
y_2                             1
y_3                             1
y_4                             1
y_5                             1
y_6                             1
y_7                             1
y_8                             1
x_1_1                           1
x_2_2                           1
x_3_3                           1
x_4_4                           1
x_5_2                           1
x_6_5                           1
x_7_6                           1
x_8_7                           1
x_9_8                           1
x_10_5                          1
x_11_7                          1
x_12_8                          1
x_13_1                          1
x_14_3                          1
x_15_4                          1
x_16_2                          1
x_17_3                          1
x_18_4                          1
x_19_1                          1
x_20_2                          1

real    0m0.149s
user    0m0.048s
sys     0m0.012s
```

6d. R codes, lpsolve input and output for Part III

(6d-1). The following is the `R` code I wrote to generate the input file for `lp_solve` for `Part II`.
In this part, we have the upper bound for the total number of colors, which is 8.
In the code below, I set $s = 8$ as an example.

```
# Part 3
#
# What if we assign two color to each region such that
#   two adjacent regions still share no color

output_file <- file("paris.txt", open = "w-")

# The upper bound for the total number of colors is 8
# In this code, I set s = 8
s <- 8

# Print objective function
print_objf(s, output_file)

# Label vertex 1,2,3 with color 1,2,3
set_color_v123(output_file)

# Print Constraint 8
# All vertices must be colored with exactly two colors
print_color_per_vertex(s, 2, output_file)

# Print Constraint 2
# Each vertex can not have an unused color
print_cons2(s, output_file)

# Print Constraint 3
# Adjacent vertices must have different colors
print_cons3(s, output_file)

# Print Constraint 4
# Reduce the combinations of colors that need to check
# This makes the computation faster
print_cons4(s, output_file)

# Print Constraint 5
# Require all y_k and x_ik to be binary
print_cons5(s, output_file)
```

(6d-2). The following is the input file for `lp_solve` in `Part III`.

```
min: + y_1+ y_2+,...,+ y_8;

(3 lines of the following types:
   assign vertex 1,2,3 with Color 1,2,3 respectively.)
label: x_1_1 = 1;

(20 lines of the following types:
   ensure all vertices have exactly one color.
 )
+x_1_1+x_1_2+x_1_3+x_1_4+x_1_5+x_1_6+x_1_7+x_1_8 = 2;

(160 lines of the following types:
   ensure each vertex can not have an unused color.)
```

```
x_1_1 <= y_1;

(704 lines of the following types:
   ensure adjacent vertices must have different colors.)
x_1_1 + x_2_1 <= 1;

(7 lines of the following types:
   ensure that we use the colors with index from the low to high.)
y_2 <= y_1;

(ensure all variables are binary)
bin y_1, y_2, ..., y_8, x_1_1, x_1_2, ..., x_1_8, x_2_1, x_2_2, ..., x_20_8;
```

(6d-3). The following is the output from `lp_solve` in `Part II`.

```
Value of objective function: 7.00000000

Actual values of the variables:
y_1                             1
y_2                             1
y_3                             1
y_4                             1
y_5                             1
y_6                             1
y_7                             1
x_1_1                           1
x_1_2                           1
x_2_3                           1
x_2_5                           1
x_3_4                           1
x_3_6                           1
x_4_3                           1
x_4_7                           1
x_5_1                           1
x_5_2                           1
x_6_3                           1
x_6_4                           1
x_7_5                           1
x_7_6                           1
x_8_3                           1
x_8_7                           1
x_9_4                           1
x_9_6                           1
x_10_1                          1
x_10_7                          1
x_11_2                          1
x_11_5                          1
x_12_4                          1
x_12_6                          1
x_13_3                          1
x_13_7                          1
x_14_5                          1
x_14_6                          1
```

```
x_15_1                          1
x_15_7                          1
x_16_2                          1
x_16_4                          1
x_17_1                          1
x_17_5                          1
x_18_2                          1
x_18_3                          1
x_19_4                          1
x_19_5                          1
x_20_1                          1
x_20_3                          1

real    0m0.102s
user    0m0.059s
sys     0m0.012s
```

6e. R codes for the functions

The `R` code includes the functions I used in section `6b`, `6c`, and `6d`.

```r
# Print the objective function
# min: y_1 + y_2 + ... + y_s
print_objf <- function(num_colors, output_file) {
  obj_str <- "min: "
  for (k in 1:num_colors) {
    obj_str <- paste(obj_str, "+ y_", k, sep = "")
  }
  cat(obj_str, ";", "\n", sep = "", file = output_file)
}

# Set vertices 1, 2, and 3 form an triangle
# Assign vertex 1 with color 1, vertex 2 with color 2, vertex 3 with color 3
# This speeds up computations
set_color_v123 <- function(output_file) {
  cat("label: x_1_1 = 1;", "\n", file = output_file)
  cat("label: x_2_2 = 1;", "\n", file = output_file)
  cat("label: x_3_3 = 1;", "\n", file = output_file)
}

# Constraint 1: All vertices must have exactly a specified number of colors, rhs.
#               x_i1 + x_i2 + ... + x_ik = rhs
print_color_per_vertex <- function(num_colors, rhs, output_file) {
  for (i in 1:20) {
    vertex_con <- ""
    for (k in 1:num_colors) {
      vertex_con <- paste(vertex_con, "+x_", i, "_", k, sep = "")
    }
    cat(vertex_con, " = ", rhs, ";", "\n", sep = "", file = output_file)
  }
}
```

```
# Constraint 2: If x_ik = 1, then y_k = 1
#               Each vertex can not have an unused color
print_cons2 <- function(num_colors, output_file) {
  for (i in 1:20) {
    for (k in 1:num_colors) {
      cat("x_", i, "_", k, " <= ", "y_", k, ";", "\n",
          sep = "", file = output_file)
    }
  }
}

# Constraint 3: Adjacent vertices must have different colors
print_cons3 <- function(num_colors, output_file) {
  for (i in 1:20) {

    for (j in 1:20) {

      # Vertices i and j are connected if mat[i,j] is 1
      if (mat[i,j] == 1) {

        for (k in 1:num_colors) {
          cat("x_", i, "_", k, " + x_", j, "_", k, " <= 1;", "\n",
              sep = "", file = output_file)
        }
      }
    }
  }
}

# Constraint 4: Use Color k if Color k-1 is not available
#               We consider to use the colors with index from the low to high
#               Makes the computation faster
print_cons4 <- function(num_colors, output_file) {
  for (k in 2:num_colors) {
    cat("y_", k, " <= y_", k-1, ";", "\n", sep = "", file = output_file)
  }
}

# Print the Bound of the total number of colors
print_bound <- function(num_colors, bound, output_file) {

  bound_str <- ""
  for (k in 1:num_colors) {
    bound_str <- paste(bound_str, " + y_", k, sep = "")
  }
  cat(bound_str, bound, "\n", sep = "", file = output_file)
}

# Constraint 5: All of our variables are binary
print_cons5 <- function(num_colors, output_file) {
  bin_str <- "bin "
  for (k in 1:num_colors) {
    bin_str <- paste(bin_str, "y_", k, ", ", sep = "")
```

```
  }
  for (i in 1:20) {
    for (k in 1:num_colors) {
      bin_str <- paste(bin_str, "x_", i, "_", k, ", ", sep = "")
    }
  }
  bin_substr <- substring(bin_str, 1, nchar(bin_str) - 2)
  cat(bin_substr, ";", "\n", sep = "", file = output_file)
}

# Constraint 7: Two regions that border the same region
#               must have different colors.
#               x_ik + x_jk + x_mk <= 1
print_cons7 <- function(num_colors, output_file) {
  for (i in 1:20) {
    for (j in 1:20) {
      # Find the adjacent regions for region i
      if (mat[i,j] == 1) {

        # Find the adjacent regions for region j
        for (m in 1:20) {

          # Find the region m that border j
          if (mat[j, m] == 1 && m != i) {

            # Print the constraint for each color
            for (k in 1:num_colors) {
              cat("x_", i, "_", k, " + x_", j, "_", k,
                  " + x_", m, "_", k, " <= 1;", "\n",
                  sep = "", file = output_file)
            }
          }
        }
      }
    }
  }
}
```