Time to Leave Louvre

Ashmitha Ambastha, ashmitha@kth.se
Ismail Demirkoparan, ismaild@kth.se
Xiaoying Sun, xsu@kth.se

With an increase in maintaining distance and increasing COVID-19 restrictions, it's essential to develop an efficient exit/evacuation plan which conforms to COVID-19 restrictions. The logic built in this project can be applied to all kinds of enclosed buildings such as museums, malls, offices etc. Here we analyse and find the fastest way to exit a building without having to manually guide visitors which can be a very inefficient task. For our use case, we have chosen to build our solution on a very popular museum - The Louvre.

Problem Statement

After understanding and analysing the problem in-depth, we've have come to the following list of issues that we need to work on:

1. Designing a model which allows visitors and staff from exiting the Lourve as soon as possible if there is any emergency.
2. This model should let staff enter the building without blocking any exits as soon as possible.
3. The design model should be adaptable, as it should be able to be solve any potential roadblocks.
4. Be able to propose this design as a recommendation to the safety and security staff at the Lourve.

We will work with GraphX to process graph-based data (Lourve's exits). Our MAIN goal is to find the shortest path to exit the museum for the visitors in case of emergencies.

Hypothesis

Considering the very complicated structure of the Lourve along iwth several exhibition points in the museum, we need to make an ideal model which can aid in simplifying the problem. These are assumptions which have been taken into account for the successful completion of the project

1. Each person exiting the museum are not and can not circle during the leaving process. They should be moving only towards the designated exit.
2. We assume that there are no obstacles in the corridor towards the exit or on the staircase to exit.
3. We assume that everyone walks in the same speed and hence the the time in which they leave the building can be a direct summation.
4. We assume that elevators are not functioning during the exiting process is underway.

Tools

In this subsection, we explain the tools which we have used widely in our project.

1. First and foremost, we use the logic of weighted graphs in graph theory. As we need to abstract multiple exhibition halls in the museum into a set number of nodes.
2. We take the four main exits as the four export nodes and buikd the exit path connecting two nodes as a directed node.
3. We use Spark GraphX to build the directed weight graph. Spark GraphX is utilised by creating VertexRDDs and EdgeRDDs,
4. At every node, we record the distance between the current node and other nodes. We the use the Dijkstra algorithm to find the shortest path.
5. Finally, we use MapReduce to compare the data flow and visualize the data distribution using R.

Methodology

Model construction

Firstly, we abstracted the adjacent exhibition halls as a virtual node and eventually turn all exhibition halls into 28 nodes.

Secondly, we divided the passages in Louvre into three parts: upstairs, downstairs and corridor and build a directed multi-distance weighted graph, L_{ij}.

$$L_{ij} = \begin{bmatrix} L1_{ij} \\ L2_{ij} \\ L3_{ij} \end{bmatrix} \tag{1}$$

where,

$L1_{ij}$ represents the upstairs distance from node i to j.

$L2_{ij}$ represents the downstairs distance from node i to j.

$L3_{ij}$ represents the corridor distance from node i to j.

After consulting relevant literature(1) , we got the relationship between personnel density and personnel flow velocity. In the ideal case, the speed of walking downstairs, upstairs and in the corridor is 1m/s, 0.8m/s and 1.4m/s respectively.

Then we combine with the Directed multi-distance weighted graph to get a different Directed time weighted graph. Each attribute of the Edge in Directed time weighted graph can be calculated by formula 2.

$$t(i,j) = \sum_{3}^{k=1} \frac{L_k(i,j)}{v_k} \tag{2}$$

where, i and j represents srcId and dstId, k represents different types of speed.

Algorithm description

There are four known exits, and the direction of our path is from the exit to the abstract node of each exhibition hall. We use the Dijkstra algorithm to calculate the shortest distance from the exit to the exhibition hall.

1. Initialize the starting vertex to distance zero and all other vertices to distance infinity.
2. Set the current vertex to be the starting vertex.
3. For all the vertices adjacent to the current vertex, set the distance to be the lesser of either its current value or the sum of the current vertex's distance plus the length of the edge that connects the current vertex to that other vertex.
4. Mark the current vertex as having been visited.
5. Set the current vertex to be the unvisited vertex of the smallest distance value. If there are no more unvisited vertices, stop.
6. Go to step 3.

Some exhibition halls can lead to multiple exits, we use MapReduce to keep the shortest path of each node.

Data

We used Github to share the project with each other and to keep track of the versions of it.

We used Louvre museum during the map building. There are several floors in that building which can be seen below in figure 1.

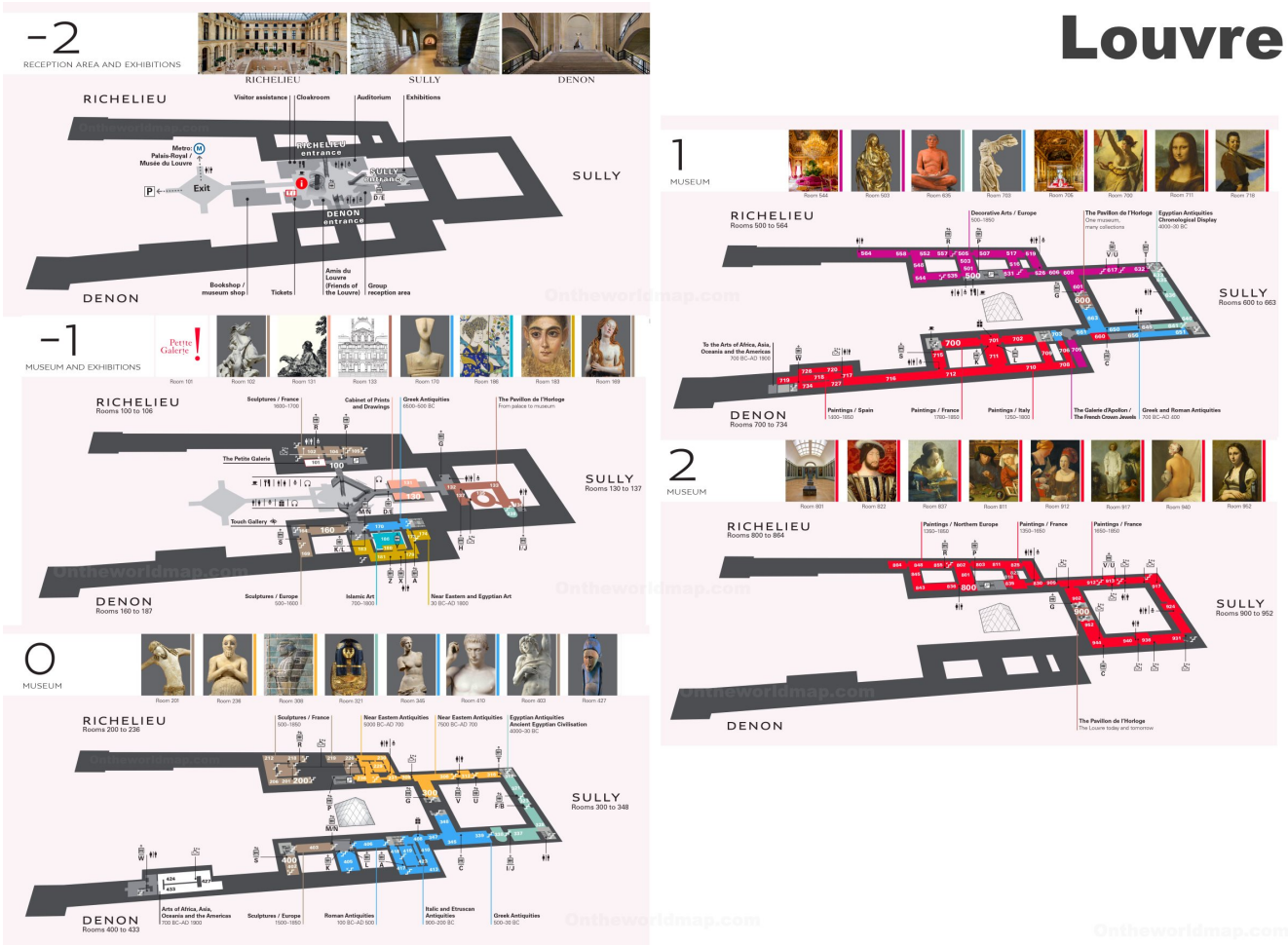

Figura 1: This is the layered map of Louvre (4)

As seen, the map have several floors, elevators and obstacles. Because of time constraint, we opt to reuse a map created in a study. As earlier mentioned, because of time constraint we overlooked the obstacles. The elevators were considered being is out of service during the exit process since it could be risky to use it in case of an evacuation.

The map we used to create the vertices and edges for calculating the shortest path can be seen below in figure 2

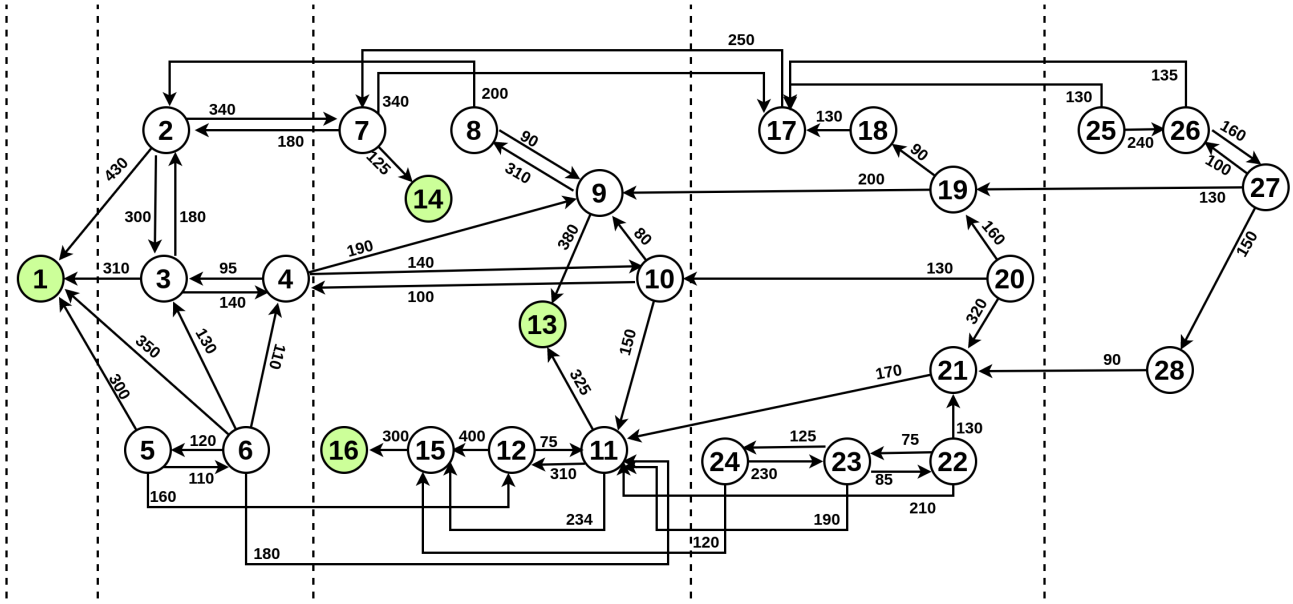

Figura 2: Map for the Louvre museum (4). The green vertices are the exits, while white ones are spots/rooms.

Using the map in figure 2 as reference, we created vertices and edges which can be seen below in the pseudo code of the implementation.

```
// Create an RDD for the vertices
val nodes: RDD[(VertexId, String)] =
  sc.parallelize(Seq(
    (1L, "N1"),
    (2L, "N2"),
...))

// Create an RDD for edges
val edges: RDD[Edge[Double]] =
  sc.parallelize(Seq(
       Edge(1L, 2L, 150.0),
       Edge(3L, 2L, 70.0),
...))

// Build the Louvre Graph
val myGraph = Graph(vertices, edges)
```

Dijkstra: By using vertices and edges combined with Dijkstra we calculated the shortest path to the exists. We first focused on each exit vertex and calculated the shortest path for the other connected vertices and lastly saved the paths locally. There could be situations when one path could be better than another, this could be seen when every exit path were calculated.

Map-Reduce: The situation above was not a desired situation, thus by using Map-Reduce and a function checking the lowest value we found the shortest paths for each vertex. The result was written to a file locally.

Paralyse: We also simulated a situation when one node could be paralyzed, meaning that the node were not possible to pass through. The Dijkstra and map-reduce were recalculated and written locally.

Results

Tables 1, 2 and 3 below shows the final result of the shortest path. The table 1 is sorted with the respect to the vertices, while table 2 is sorted with the respect to the time. As earlier mentioned in the , all of the tables are also map reduced.

Table 1: Paths with shortest time (With respect to Source)

Source	Time(sec)	Path
N1	0.0	
N2	150.0	1
N3	120.0	1
N4	195.0	1 → 3
N5	180.0	1
N6	200.0	1
N7	125.0	14
N8	160.0	13 → 9
N9	100.0	13
N10	175.0	13 → 9
N11	175.0	16 → 15
N12	150.0	16 → 15
N13	0.0	
N14	0.0	
N15	75.0	16
N16	0.0	
N17	325.0	14 → 7
N18	385.0	14 → 7 → 17
N19	260.0	13 → 9
N20	270.0	13 → 9 → 10
N21	315.0	16 → 15 → 11
N22	350.0	16 → 15 → 11
N23	320.0	16 → 15 → 24
N24	195.0	16 → 15
N25	455.0	14 → 7 → 17
N26	470.0	13 → 9 → 19 → 27
N27	390.0	13 → 9 → 19
N28	395.0	16 → 15 → 11 → 21

Table 2: Paths with shortest time (With respect to time)		
Source	Time(sec)	Path
N1	0.0	
N13	0.0	
N14	0.0	
N16	0.0	
N15	75.0	16
N3	120.0	1
N7	125.0	14
N2	150.0	1
N12	150.0	16 → 15
N8	160.0	13 → 9
N10	175.0	13 → 9
N11	175.0	16 → 15
N5	180.0	1
N4	195.0	1 → 3
N24	195.0	16 → 15
N6	200.0	1
N19	260.0	13 → 9
N20	270.0	13 → 9 → 10
N21	315.0	16 → 15 → 11
N23	320.0	16 → 15 → 24
N17	325.0	14 → 7
N22	350.0	16 → 15 → 11
N18	385.0	14 → 7 → 17
N27	390.0	13 → 9 → 19
N28	395.0	16 → 15 → 11 → 21
N25	455.0	14 → 7 → 17
N26	470.0	13 → 9 → 19 → 27

Table 3: Paths with shortest time with paralyzed vertex (With respect to time)		
Source	Time(sec)	Path
N1	0.0	
N3	120.0	1
N2	150.0	1
N4	195.0	1 → 3
N6	200.0	1
N10	265.0	1 → 3 → 4
N20	360.0	1 → 3 → 4 → 10

In table 3, the paths are focused on the exit vertex N1.

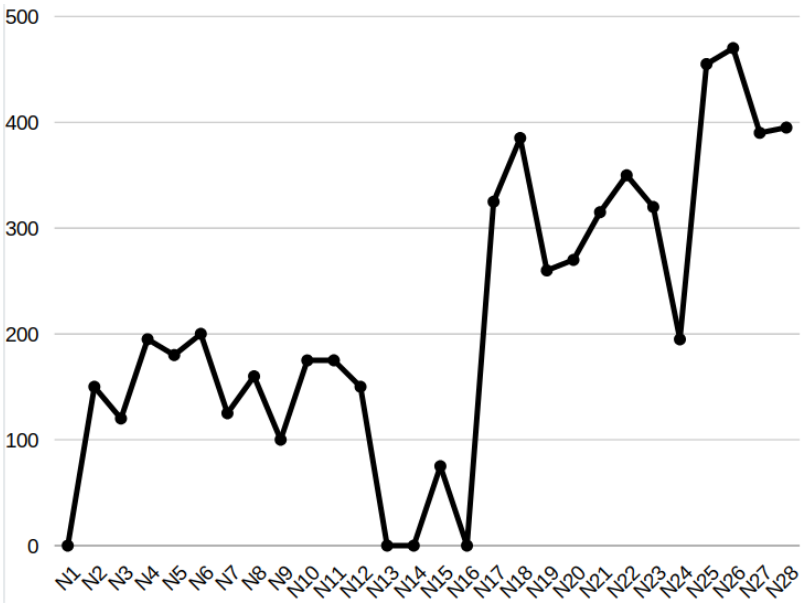

Figura 3: Graph that shows the shortest exit time from each vertex

From the figure 3, we believe that the node has a waiting time exceeding 300 should be considered as a dangerous area. we recommend to limit the flow of people and close the museum early to prevent the tragedy of untimely evacuation.

Referencias

[1] Xu Hui, Tian Wei, Wang Yong. Simulation study on emergen-cy evacuation of dense passenger flow in rail transit trans-fer station [J/OL]. Journal of System Simulation:1-9[2019-01-29].

[2] "Interactive Floor Plans." Louvre - Interactive Floor Plans — Louvre Museum — Paris, https://www.louvre.fr/en/visit/map-entrances-directionsmuseum-map.

[3] Number of visitors to the Louvre in Paris from 2007 to 2020: https://www.statista.com/statistics/247419/yearly-visitors-to-the-louvre-in-paris/

[4] Louvre Museum. Map. Floor plan. - Paris Digest https://www.parisdigest.com/museums/louvre$_{i}nformation_{p}lan.$