Introduction

When an emergency occurs in a closed public space, its devastating effect can take thousands of lives away, causing considerable losses to public wealth. However, in most cases, the occurrence of an emergency is not the sentence to death but the poor emergency management and reaction mechanism. The chaos caused by the panic crowds fleeing in all directions can hinder the evacuation process and may block the way for the emergency personnel to get to the scene, carrying out their work. As a result, an effective and dedicated emergency evacuation plan should be designed for any public spaces, ensuring efficient responses that can help to minimize the losses.

Many models have been developed and tested on the subject of evacuation, such as the famous simulation model based on cellular automation[1]. These established models invariably take a perspective that mainly considers individual human behavior in the face of an emergency. However, such a model focusing on human behavior is more like a scenario simulation of the self-driven evacuation pattern, rather than an emergency evacuation plan designed for a crowded public structure. The latter requires a more comprehensive consideration that is beyond the individual level.

Instead of considering individual choices of evacuation route, we primarily look at the evacuation plan from a macro perspective and think the personnel involved as a whole. We model the evacuation process by comparing it to the flow of networks. By analyzing the similarities and differences between our evacuation model and the typical flow network model, we make some adjustments to the flow network model and work out our evacuation model[2]. Then we start testing it on multiple scenarios, adjusting models accordingly with emphasizing on more detailed features. Further, we examine our model with the occurrence of random unexpected events, e.g., the existence of visitors with mobility difficulties, obstruction of evacuation routes. Our model eventually proves to have a real-time self-adjustment mechanism towards those unexpected situations and is robust in achieving its designed goal of minimizing the total evacuation time against any changes. We take the exact data from the Louvre as a test and analysis of our model and we also find this model's general applicability to any crowded, public structure.

Restatement and clarification of the problem

The 2019 ICM Question D addresses the issue of emergency evacuation, which requests us to design an evacuation model for a crowded public space on two levels.

- First, it sets a specific situation for the design of evacuation plan -- the Louvre in Paris, a city with an increasing number of emergencies as terrorist attacks. This requires incorporation of detailed geographic information and specific designing features regarding this particular museum.
- Second, the question also expects a broader application of our evacuation model that can be adapted and implemented in any large, crowded structures.

General definitions

Graph: The graph is a mathematical definition of a typical structure in which objects are related to each other symmetrically or asymmetrically. Objects in the graph can be connected or disconnected.

Vertex[Vertices]: The vertex is a definition of the object in graph. For example, in this paper, one room is abstracted into a vertex.

Edge: The edge is the connection between two vertices. It can also be called as arc or path.

Route: The route is a sequence of edges that connect a sequence of vertices.

Source: The starting vertex of one route.

Sink: The ending vertex of one route.

Route Depth: The route depth is the minimum number of edges used to connect the source to the sink.

Flow Capacity: The flow capacity is the maximum flow one edge can carry. In this paper, the flow capacity of model means the number of people passing during a unit of time.

Flow Network: The flow network is a directed graph where each edge has a capacity and each vertex can send or receive amount of flow. The amount of flow on one edge is always smaller or equal to the capacity of the edge.

Max Flow (Maximum Flow Problem): The max flow is a way to find a sequence of routes in a single-source, single-sink flow network to make the total flow maximum.

General assumptions

People are rational during the whole evacuation process and only follow the designated evacuation route as assigned.

- When an emergency occurs, not knowing what to do and where to go at such an emergent situation causes panic to spread across the crowd, and thus intensifying the chaos. On the contrary, an ideal situation is that there's a universal system monitoring the general flow of people and the emergency personnel, broadcasting instructions for different parties to take actions under the circumstance. To achieve the success of evacuation, it also requires that all visitors fully understand and trust the instructions given by the museum's emergency management team. That's why primary assumption made here is that visitors will behave and respond rationally to the emergency and the instructions. That is to say, the basic starting point of our model is that we do not consider the individual emotional-driven behavior, but looking at the evacuation process as at a macro-level.

People are generally viewed as being able to understand and follow the instructions from the security.

- A delay in following instructions appears randomly among crowds and can influence the optimal time for evacuation. So we assume that people in general would evacuate by the route as designated at a constant time as calculated. That is to say, any individual behavior opposite to taking the route as designed are viewed as a friction, or a random shock to the model, which we plan to take into consideration in the sensitivity analysis.

Each visitor moves at a consistent speed during the whole evacuation process

- In face of emergency, chaos and unordered evacuation situations are what we want to avoid. That's why even if the crowdedness in different rooms might be different, our model assume that the crowds move at a constant speed. This is also an important factor in real life for a successful evacuation. By making this assumption, we guarantee that whatever route people take, rooms does not squeeze and ways will not be blocked because of over-crowdedness, and the outflow at exit is smooth during the whole evacuation.

People generally follow orders for evacuation

- Though the local overtaking is allowed in our model, we see the whole crowds evacuate with orders. This correspond to the assumption that people move at the same speed. Also, we would view the local overtaking, and the friction it causes as a random shock to the model, which the sensitivity analysis would cover.

We prioritize the evacuation by how close the rooms are from where the emergency happens as well as the depth of the room from the exit

- How close one room towards the exit is measured by "depth" in our model and the depth of the exit is always defined as 0. In this example, room 2 has a depth of 3, room 3 and the emergency spot room 1 has a depth of 2, room 4&5 has a depth of 1.
- In real -life cases for the sake of securing human lifes, we always want to evacuate people at the emergency spot first, which is room 1 in the example. And we establish an evacuation order in the model as to evacuate based on how closely connected it is with the emergency spot. If these rooms have same distance from the emergency spot, we prioritize rooms with higher depth from the exit to evacuate first. For example, after evacuating of room 1, we start evacuating room 2 & 5. Since 2 has depth of 3 (2-3-4-6), and 6 has depth of 1 (5-6), we ensure the flow of room 2 first and room 5 will take the rest part of the max flow. Then evacuating room 3, and 4 would be the last for consideration of evacuation.

People always take the route away from where the emergency happens

- Rooms are considered connected in both ways (undirected), yet to appeal to the real-life situation, not all route can be taken. In our model, we assume that people are always getting away from where the emergency happens. For example (Figure 1), to evacuate people in room 4, we only consider the route 4-6. We don't consider 4-3 since the depth of room 3 is greater than room 4.
- This also guarantees the safety issue in the evacuations process since no crossing emergency spot action is allowed in the model following this rule.

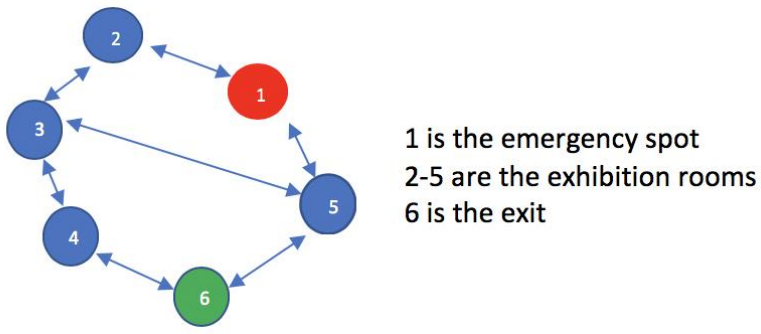


Figure 1

Hallways are modeled as rooms with same flow capacity in both ways

- In the basic flow network model, only vertices [rooms in this case], connections between vertices, and flow capacity are considered and modeled[4,5]. Yet in real life evacuation case, we also have to consider the hallways, which can hold people and also

acts as evacuation path. Also, the length of such hallways affect the evacuation time we want to optimize so that it must be considered. To include such a factor and also to simply the model, we consider such hallways as vertices [rooms] as well, which considers it's capacity of holding people. And as we view them as vertices with the same flow capacity at both sides, we keep the feature of a hallway.

We prioritize the evacuation of visitors inside the museum in our model

- Though the rescue personnels also important in case of emergency, we use the minimized time for only the evacuation of people inside as a criteria for the model. We consider that in most cases of emergency situations in crowded places, evacuating people first is the best way for the sake of human-life saving.

Model discussion:

1. Primary Model

1.1 Model specification

The interior layout of the hallway & exhibition room, and the exact number of people in each place add to the complexity of the model we build, so we start from the simplest threading layout as follows (Figure 2).

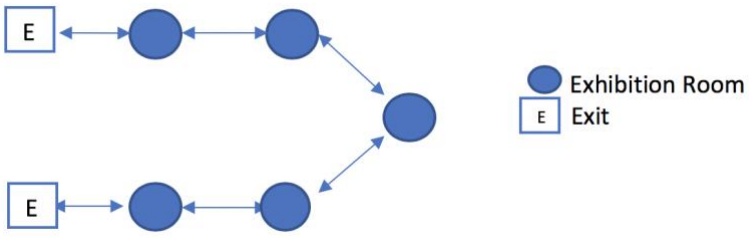


Figure 2

1.2 Model discussion & Result

Based on this threading layout with a symmetric placement of exhibition rooms and exits as Figure 3 shows, once an emergency occurs in any spot, visitors are supposed to and only able to go to the exit that is in the opposite direction of the emergency spot (relative to their positions), leaving a clear evacuation plan for everyone.

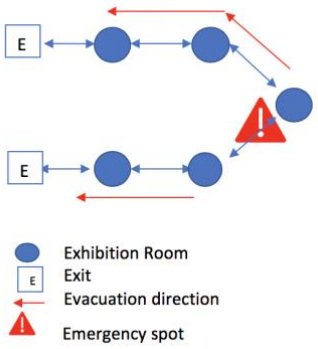


Figure 3

In this model with the simplest threading connection of exhibition area, the best evacuation solution is to go the other way of the emergency position.

2. Incorporation of the layout features of the Louvre

2.1 New feature discussion

The actual layout of the Louvre is much more complex. In the simplest one-threading model, the visitors would have no way to go in the case where both the left side and the right side get blocked simultaneously. The Louvre is generally divided into three thematic sections according to the exhibitors. Also, the real design of the Louvre considers a lobby-function cable that connects the three exhibition rooms in the middle directly connect to each other. Besides, there are also two hallways on each floor that can also be modeled as rooms as well according to the general assumption. Besides, there are many stairs in between floors in case of evacuation. In short, in this refined model, our adopted interior layout of the Louvre is as the follows (Figure 4).

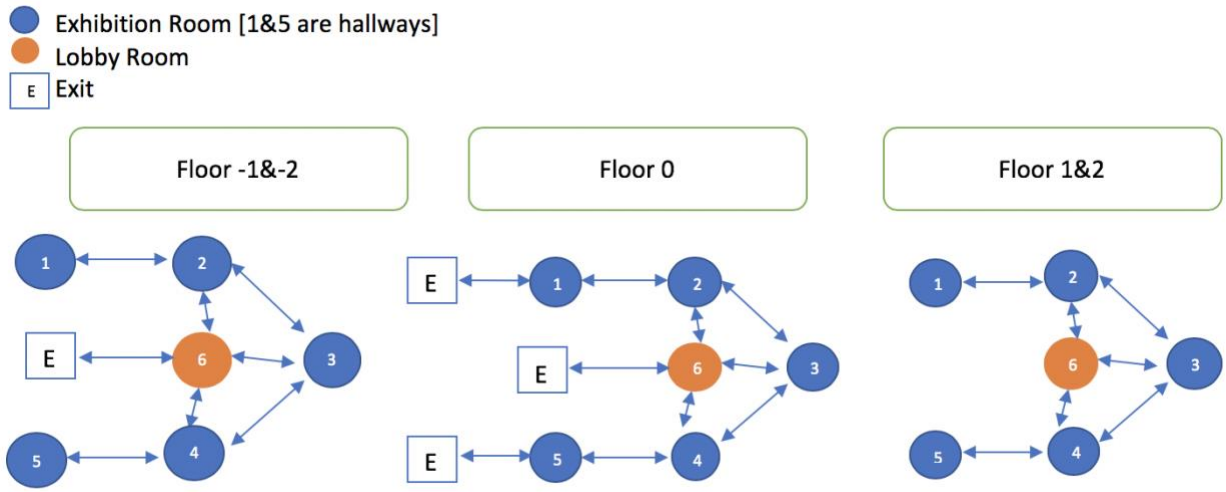


Figure 4

2.2 Model specification & algorithm development

The flow of crowds in an evacuation can be analogous to the flow network, which is a well-established model. Two inputs are required in this model, one is the flow network and the other is the ways that networks are connected. However, there exist some differences between the model we establish and the typical flow network model. In the typical flow network model, there are both inflow and outflow of people. Nevertheless, in the case of our evacuation model, though people can move in between the exhibition rooms, there is no inflow of people. At this point, we would like to keep evacuation personnel outside of our discussion, which we would address later.

Considering the real evacuation necessity, we adopt the parameters from the basic flow network model, which employs the connection of different vertices [the exhibition rooms in this case], and the crowd flow in each network [hallway in this case]. We establish the model based on the following consideration of real-life evacuation (Figure 5.1).

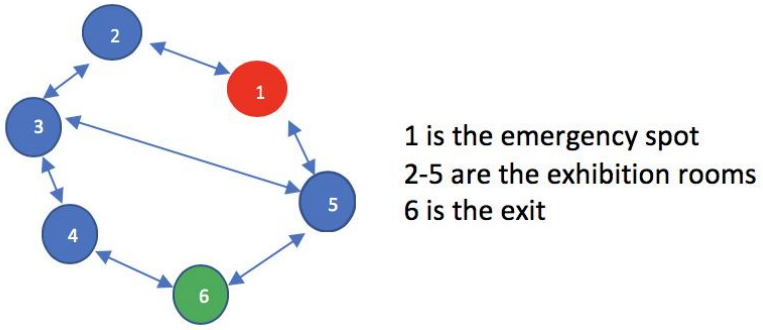


Figure 5.1

The emergency spot (represented as the circle numbered 1 on Figure 5.1) should be evacuated first, and then there are four route choices left:

1-2-3-4-6, 1-2-3-5-6, 1- 5-6, 1-5-3-4-6

The first step is to calculate the maximum flow for evacuation from emergency spot 1, which is route 1-2-3-4-6 and 1-2-3-5-6. Then we can detach the vertex 1 from our graph, which leaves us with the following (Figure 5.2).

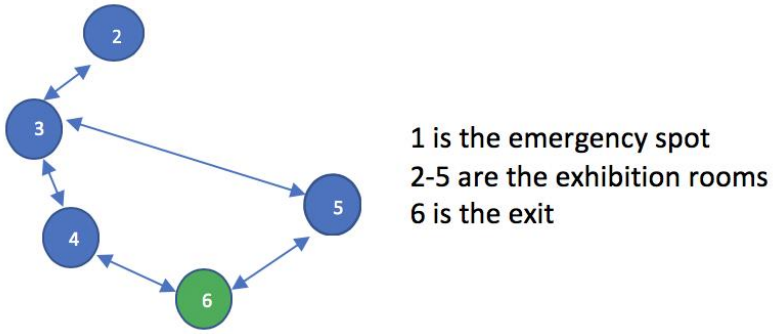


Figure 5.2

Then we evacuate the vertices closest to emergency spot 1 [such as 2 & 5], and the segregated rooms after detaching 1 [such as 2], in this way, we would not leave unconnected rooms such as 2 unevacuated after 3 is evacuated. After the evacuation of 2 ,3, 5, we can then detach the vertices with the greatest depth (most far away from the exit), like 2, 3 in this case. Then after 4 is evacuated, we detach the last two vertices 4 and 5, and eventually get to the exit 6, leaving no other room unevacuated. This "Detaching vertices" Model we established can be compared to the behavior of water flow in a complex network of water pipes. In that case, the emergency spot is considered the water storage at the highest altitude, and the exhibition rooms are considered the temporary water pound in between the water pipes. So the complete evacuation of the crowd can be simulated as the behavior of water in the pipes when the highest water storage is released. Additionally, the prioritized evacuation room (in depth) can be viewed as an order in which water flows into the temporary storage. The farther to the exit, the higher the altitude is, no matter how many networks of water pipes one storage has, water always go down from the highest. Thus, it's a perfect match for out evacuation plan. As water always flows out, our model always guarantees the evacuation of all rooms.

Mathematical Algorithm (Pseudocode)

```
## Initialization
V = len(graph)          # Getting The Size of Vertix(Rooms)
toVisit = [startingPoint] # List Initialized with 1 element
visited = List()        # Initialized by False with length of V
currentStep = 1         # Current Step

## Algorithm Starts
While toVisit is not empty:
   ## Connect toVisit Points By Flow Entry Point 0 And Build a New Graph g
   g = a deep copy of graph
   nextVisit = empty List()
```

```
    ForEach v in toVisit:
      ## Finding a nearest point to v that will not divide the graph into two
      hasVisited = a deep copy of visited
      isolatedPoint = isMid(graph, v, exitPoint, hasVisited)
      If isolatedPoint is not None:
        hasVisited[v] = True
        v = isolatedPoint
        isolatedPoint = isMid(graph, v, exitPoint, hasVisited)
      End If
      ## Connect the new point v to starting vertice
      g[0][v] = INF # Meaning the flow from 0 to v is infinite
      visited[v] = True
      ForEach v' Connected By v Which is not visited:
        nextVisit.push(v')
      End For
    End For

    ## Doing MaxFlow Algorithm Upon The New Graph g
    doMaxFlow(g, 0, t)
    toVisit = nextVisit
    currentStep ++

    ## Remove Visited Nodes from Graph
    ForEach v Where g[0][v] == INF:
      graph[:][v] = 0
    End For
  End While
End Function
```

Another problem we encounter here is that in each step of our detaching process, we need to calculate the maximum flow capacity from the current vertex to the exit. In the first step [from the emergency point to the exit], it is easy to calculate the maximum capacity, yet when detaching the first vertex, we face a problem of calculating maximum capacity from multiple sources, which cannot be solved by the maximum capacity algorithm. To solve this problem, we create an imaginary point 0 to connect all current vertices that are in the face of evacuation. For example, after the detachment of the emergency point, point 0 is considered to be connected with 2&5, so that the maximum capacity can be calculated as the one from 0 to exit again.

2.3 Dealing with multi-source & multi-sink problem

In a classical maximum flow problem, there usually exists only one pair of source and sink (emergency spot and exit). However, in our situation, visitors are evacuated from multiple rooms to multiple exits. To solve this, we create a pair of virtual source and virtual sink and connect sources by the virtual source with infinite capacity, sinks by the virtual sink with infinite capacity as well. By creating the virtual pair of source and sink, we do not limit the flow network

for their outbound and inbound are set to infinity, while it provides a unique entrance and exit to the graph. In that case, we transform the multiple sources and sinks problem into the classical maximum flow problem model (Figure 6).

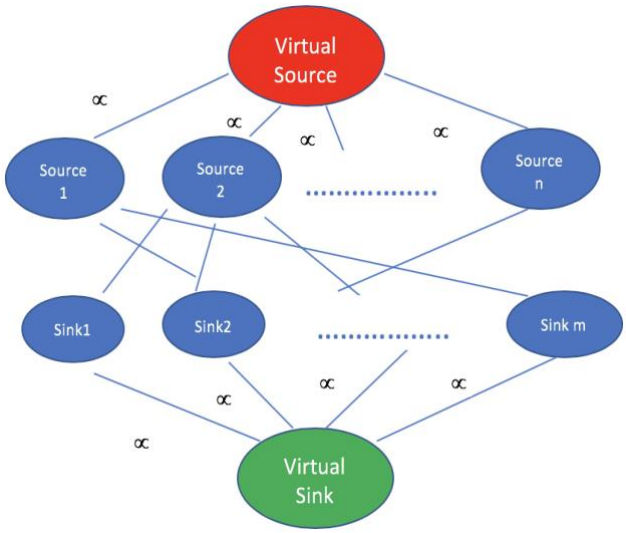
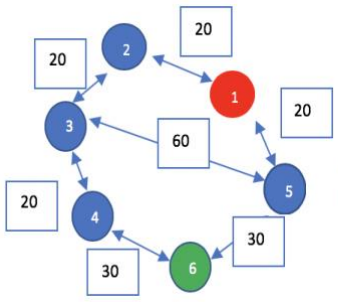


Figure 6

2.4 Simple test for the model

For the Test case [only 5 exhibition rooms, 1 imaginary point 0 and 7 edges]mentioned above, we first assign the flow on each network and build a matrix for each vertex, with row representing each vertex, and each column representing the flow in between vertices. For example, the position in the second row and first column represent the flow in between vertex 1 and vertex 1, which is 0, and the position in the first row and second column represent the flow in between vertex 1 and vertex 2. Moreover, any two vertices without a direct connection are given a flow capacity 0. If we assign the flow network as follows. Then in the test model, we have a 7x7 matrix as below (Figure 7).



1 is the emergency spot
2-5 are the exhibition rooms
6 is the exit

$$capacityMatrix = \begin{vmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 20 & 0 & 0 & 20 & 0 \\ 0 & 20 & 0 & 20 & 0 & 0 & 0 \\ 0 & 0 & 20 & 0 & 20 & 60 & 0 \\ 0 & 0 & 0 & 20 & 0 & 0 & 30 \\ 0 & 20 & 0 & 60 & 0 & 0 & 30 \\ 0 & 0 & 0 & 0 & 30 & 30 & 0 \end{vmatrix}$$

Figure 7

Running the simulation, we have the result for the test case as follows:

```
Step 1
```

```
  1-5-6     [capacity = 20]
  6-4-3-2-1 [capacity = 20]
Step 2
  5-6       [capacity = 30]
  2-3-4-6   [capacity = 20]
Step 3
  3-4-6     [capacity =  20]
Step 4
  4-6       [capacity =30]
```

2.5 More complex situation test

For the general and more realistic model, we have the interior layout of the Louvre as follows (Figure 8).

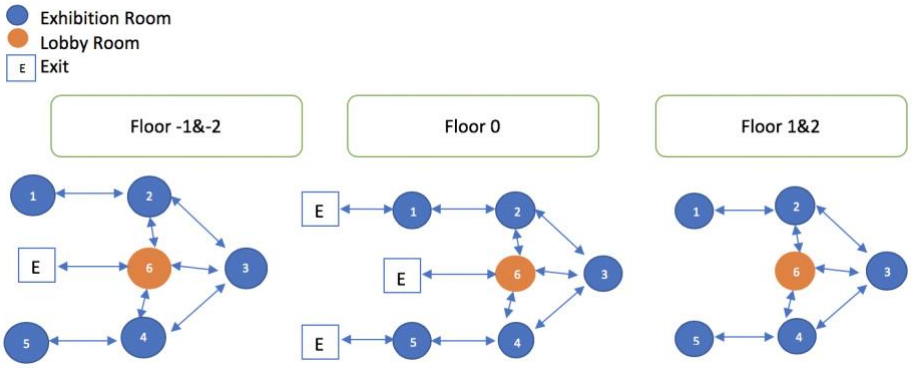


Figure 8

Then we calculate the flow capacity of each hallway and each set of stairs.

According to official statistics, the Louvre had a record 10.2 million visitors in 2018[6]. Given the opening hour of the Louvre in 2018, the total amount of time that the museum is open to visitors in 2018 equals 3250 hours, 195,000 minutes. Dividing the total number of visitors by the sum of opening hours gives us an estimated number of visitors inside the museum every minute. As our model features a main floor with multiple exhibition rooms, we would like to decrease the number of floors from five to three in our estimation of the number of visitors on each main floor every minute in Louvre for the following reasons. Among the five floors of the Louvre, the Napoleon Hall mainly works as a reception and information center, and the lower ground floor as well as the second floor, houses relatively fewer collections; roughly speaking the sum of traffic on these three floors may be comparable to a main exhibiting floor. On each floor, we divide the space into two categories; one is exhibition area where visitors appreciate the museum's collections and the other is the hallway where visitors generally walk through from one exhibition room to the other. There should reasonably exist a difference between the speeds at which visitors are walking in these two different areas; faster in the hallway and

slower in the exhibition room. To recognize such differences, we would like to propose a relative ratio of 5:2 for the speed of visitor in the hallway compared to that in the exhibition area.

$$10{,}200{,}200 \div (3250 \times 60) \div 3 \times \frac{5}{(5+2) \div 2} = 24.9$$

Based on the evidence and rationales above, our estimation for the speed of visitor flow is 25 ppl/min in the hallway, and respectfully 10 ppl/min in the exhibition rooms. Now we assume that in the face of an emergency, people would all move at this speed under the guidance of the security. This assumption is made based on the real-life evacuation situation for several reasons. First, standing at the point of the Museum, it wants a secure, steady evacuation plan. Second, the Louvre is usually crowded with visitors in the real situation, thus evacuating at different speed would be dangerous and chaotic.
Multiplying the speed by the width of the walkable area, we work out the flow capacity as the following result shows:

- 240 ppl/min at exits
- 96 ppl/min in between exhibition rooms
- 76.8 ppl/min for stairs in between exhibition rooms
- 144 ppl/min for stairs in between lobbies

We assume that the emergency point is in floor 2 lobby.
We assume that all exhibition rooms have 1 stair and the lobbies have 2 stairs
Replicating what we did in the test case, we generate a 32X32 matrix [find in Appendix A], which is generated by an algorithm [find in Appendix B].
The result can be easily calculated in just 6 steps.

L means level
B means basement
R means room number

Step 1 [capacity = 432]
[capacity = 144] EXIT <= L1R6 <= L2R6 <= L3R6
[capacity = 76] EXIT <= L1R1 <= L1R2 <= L2R2 <= L3R2 <= L3R6
[capacity = 19] EXIT <= L1R1 <= L2R1 <= L3R1 <= L3R2 <= L3R6
[capacity = 76] EXIT <= L1R6 <= L1R3 <= L2R3 <= L3R3 <= L3R6
[capacity = 76] EXIT <= L1R5 <= L1R4 <= L2R4 <= L3R4 <= L3R6
[capacity = 19] EXIT <= L1R5 <= L2R5 <= L3R5 <= L3R4 <= L3R6
[capacity = 19] EXIT <= L1R1 <= L2R1 <= L3R1 <= L3R2 <= L3R3 <= L3R6

Step 2 [capacity = 528]
[capacity = 144] EXIT <= L1R6 <= L2R6
[capacity = 76] EXIT <= L1R1 <= L1R2 <= L2R2 <= L2R6
[capacity = 19] EXIT <= L1R1 <= L2R1 <= L2R2 <= L2R6
[capacity = 76] EXIT <= L1R6 <= L1R3 <= L2R3 <= L2R6
[capacity = 76] EXIT <= L1R5 <= L1R4 <= L2R4 <= L2R6
[capacity = 19] EXIT <= L1R5 <= L2R5 <= L2R4 <= L2R6
[capacity = 57] EXIT <= L1R1 <= L2R1 <= L2R2 <= L3R2
[capacity = 57] EXIT <= L1R5 <= L2R5 <= L2R4 <= L3R4

Step 3 [capacity = 959]
[capacity = 240] EXIT <= L1R6
[capacity = 144] EXIT <= B1R6 <= L1R6
[capacity = 96] EXIT <= L1R1 <= L1R2 <= L1R6
[capacity = 96] EXIT <= L1R5 <= L1R4 <= L1R6
[capacity = 76] EXIT <= L1R1 <= L2R1 <= L2R2
[capacity = 76] EXIT <= L1R5 <= L2R5 <= L2R4
[capacity = 76] EXIT <= B1R6 <= B1R3 <= L1R3 <= L1R6
[capacity = 19] EXIT <= B1R6 <= B1R2 <= L1R2 <= L2R2
[capacity = 57] EXIT <= B2R6 <= B2R2 <= B1R2 <= L1R2 <= L2R2
[capacity = 76] EXIT <= B2R6 <= B2R4 <= B1R4 <= L1R4 <= L2R4

Step 4 [capacity = 960]
[capacity = 240] EXIT <= B1R6
[capacity = 144] EXIT <= B2R6 <= B1R6
[capacity = 96] EXIT <= L1R1 <= L1R2
[capacity = 96] EXIT <= L1R5 <= L1R4
[capacity = 76] EXIT <= L1R1 <= L2R1
[capacity = 76] EXIT <= L1R5 <= L2R5
[capacity = 76] EXIT <= B2R6 <= B2R2 <= B1R2 <= B1R6
[capacity = 19] EXIT <= L1R1 <= B1R1 <= B1R2 <= B1R6
[capacity = 19] EXIT <= B2R6 <= B2R3 <= B1R3 <= B1R6
[capacity = 67] EXIT <= L1R5 <= B1R5 <= B1R4 <= B1R6
[capacity = 48] EXIT <= L1R1 <= B1R1 <= B1R2 <= L1R2

Step 5 [capacity = 720]
[capacity = 240] EXIT <= B2R6
[capacity = 240] EXIT <= L1R1
[capacity = 240] EXIT <= L1R5

Step 6 [capacity = 240]
[capacity = 240] EXIT <= L1R5

3. Incorporating the number of people in each room into the model

3.1 Feature discussion & model specification

The previous model we established provides the general guidance without adding variables such as the number of people in each room into consideration. That is to say, in the previous model, we assume that rooms at the same level evacuate using the same amount of time. Yet in the real-life case, the number of people in different rooms varies, and they usually evacuate at different amount of time. For example, if room A & room B are at the same level, and room A complete the evacuation ahead of room B, we would detach room A and start evacuating the next-level rooms based on the previous model. However, when we do this, we are limiting the routes for the further evacuation of room B, which does not make sense in real life.

In order to improvise the model, first, we have to quantify the number of people in each room and track them by unit of time & flow capacity at each network.

P_t (The amount of people in a room x at time t) can be formulated as :

$$P_t = P_0 - \sum_{i=t_0}^{t} f(i, x, T)$$

In this formula, means the P_0 means initial number of people. t_0 is the time that the people in the room starts to evacuate. $f(i, x, T)$ is a function to calculate the partial maxflow from vertex x to exit Tat time$i$.

Based on this model, we can observe how long it takes for each room to evacuate. Now, to solve the problem of early detaching for some rooms because of the early completion of evacuation, we design an improvised evacuation model considering the depth of the room as the priority for evacuation.

In this model, we attach depth to each room and prioritize the deeper depth rooms in the evacuation process (Figure 9). For example, in the following room structure, suppose room 1 & room 3 has the same depth 2, room 2 has depth 1. Then we firstly evacuate room 1 & 3, and by monitoring the remaining people at each time spot, we knows that room 1 complete the evacuation first, but we do not detach this room until room 3 finish the evacuation because the remaining people in room 3 take the route through room 1 for evacuation. And this model works especially well when the networks are more complex, and total depth is much larger. For example, based on the previous model, if room 2 evacuates faster than room 5, we just detach room 2 and start the evacuation of room 3, and if room 3 even competes the evacuation before

room 5, we detach room 3 and thus losing the route 5-3-4-6, which disabled the optimization measured by shortest time. However, in our new improvised model, this situation would not exist because 3 won't be detached until 5 is evacuated.

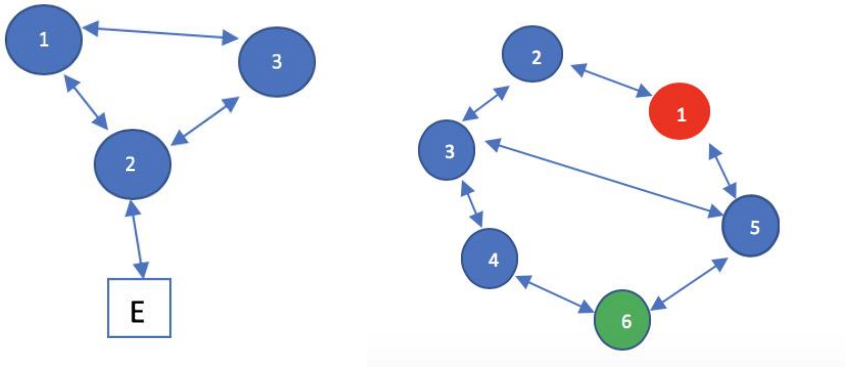


Figure 9

Key Assumptions for the new model

- People wouldn't take the route towards the larger depth compared to the evacuation point. That is to say, if they start evacuating at a depth of 3, they can only evacuate towards the room at depth of 3 or smaller depth.

3.2 Model test by the Louvre

This improvised model adopts the same logic of flow network as it is before, but by bring in the real-time quantified monitor on the remaining people in each evacuating room, it constantly rerun the previous model and brings the best route of evacuation at real-time. So each time with the rerunning, we would see a brand-new evacuation route. Now using the data we assigned for each flow network capacity, we regenerate the simulation with the additional estimate of the number of people in each room (Figure 10). In this simulation, we assume in the total 30 rooms, each has 200 people. So at the beginning of the evacuation, there are in total 6000 people in the Louvre. And the change of people inside the museum is plotted as the following applying the new model. The total amount of time for evacuation is 423s.

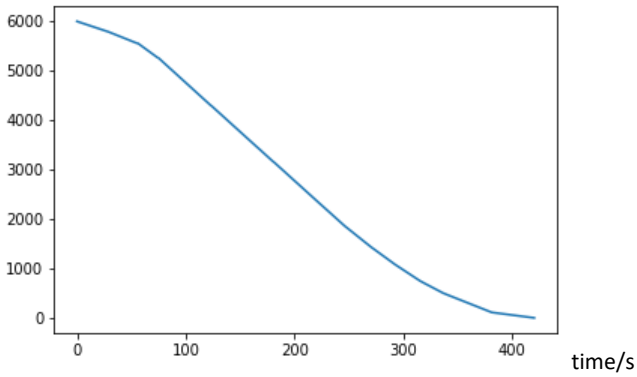


Figure 10

Extending to the general model

Although in the previous test cases, we only tested several specific cases including the Louvre, this model can be used much extensively. To be more precise, this model can be generally applicable to the evacuation plan of any public, crowded places with the following features:

- Any buildings, halls, or outdoor structures can be described by network models (can be described by structure of vertices and edges).
- The flow capacity of each edges are known or are able to be estimated.
- There exists accountable exits connected in the network.
- The amount of people in each vertex (or room) is known or able to be estimated.

Sensitivity test

In our previous model established, we consider a constant flow capacity for each network during the evacuation. However, in a real-life situation, unexpectancy happens such as disabled or injured people blocking the way, partial falling down of the ceiling, or multiple such events happening which would influence the flow capacity. We want to know how the total amount of evacuation time would fluctuate with the shift in flow capacity. To take the most cases into consideration, we here cover the following scenarios.

- **Multiple change of flow capacity is considered**
- **Duration of each change of flow capacity is considered**
- **When each change of flow capacity happen is considered**
- **How much each flow capacity change is considered.**

In order to run the sensitivity test, we give the following assumptions and set ranges for each variables we consider that may affect the optimal time for evacuation

- **Each flow network has a 1% possibility of running into a flow capacity constraint situation**
- **The duration of each flow capacity constraint is any random integer in the range of 10-30s**
- **The time for each flow capacity constraint is any random non-negative integer**
- **The parameter for the change of each flow capacity is any random number in the range of 0.05-0.3.**

Below are two graphs showing the fluctuation of evacuation according to random_parameters as listed above after running the algorithm 100 times and 1000 times (Figure 11).

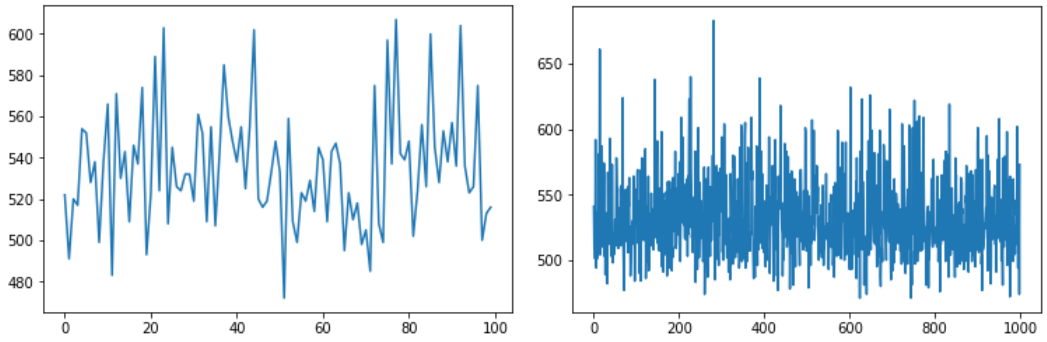


Figure 11

The standard deviation of the 1000 times random test is 30.3405s, and the mean is 532.594s. These statistics indicate that when an unexpectancy happens randomly, using our model, the optimized evacuation time would increase averagely by 109.594s. This suggests that when an unexpected situation such as a temporary obstruction happens, or any situation that affects the network capacity anywhere, the optimized evacuation time using our model would increase on average, yet does not fluctuate much according to how much or how bad the unexpectancies are. This is largely because our model is built upon the real-time situation processing and real-time evacuation plan adjustment. When any expectancies shift the original flow capacity in the network, the time for a specific room's evacuation changes, and according to how the algorithm in our model works, this triggers a rerun of the model, thus correcting the current route for all room evacuation.

The sensitivity analysis shows how strong and flexible our model is in handling the real-time situation change, which is a constant occurrence in the real-life evacuation situation. It also shows how swift our model shifts in responding to the change in situation, thus optimizing in response to the change in the evacuation process.

Rescue personnel in our model

We assumed in our model that the priority for the evacuation plan is to evacuate all people inside the museum. That's why we haven't talk about the inflow of rescue personnels in our model. However, when we use the Louvre test sample to check the total flow capacity at all exits using our model, we find that the max outflow capacity at the exit can't be filled at all times. This means that when extra flow capacity exits, rescue personnels can go in. And since

our model can simulate the flow capacity according to time. Instructions can timely be given to the rescue teams. Specifically in the Louvre case, we have the following observed outflow total capacity at exit observed.

Time	Total capacity at exit	Time	Total capacity at exit
< 30	432.00	< 320	873.60
< 60	528.00	< 341	700.80
< 79	960.00	< 367	528.00
< 251	1200.00	< 385	518.40
< 274	1065.60	< 424	172.80
< 296	969.60	Average: 918.36	Total: 6489.76

This tables the allowance for inflow of personnel in the first 80 seconds and after 252 seconds.

In the worst case, if the emergency requires the urgent handling, then no matter what the outflow evacuation is, the rescue team squeezes in, causing the temporary reduction in the flow capacity. Then like discussed in the sensitivity analysis, our model would allow for the friction in flow capacity and handle it timely.

Strengths and Weaknesses of Model

Strength

1. Our model considers a dynamic evacuation process, allowing for a more comprehensive plan.

    In our model, everytime a room of visitors have successfully evacuated, the room will be shut down and removed from the whole graph. Thus we only need to focus on the rest of the rooms and solve the easier problem. In order to accomplish this goal, we have to make sure each time when we shut down one room, it would not interfere the remaining procedure of evacuation. This means that the removal of the room will not separate the whole graph into multiple disconnected parts, in case that some of them could not be linked to the exit. We use a BFS(Breadth First Searching) algorithm to determine if there are some rooms being isolated (or disconnected to the exit), and

then using DFS(Depth First Searching) to query the nearest vertex (room) that can be evacuated and removed safely. Here is an example:

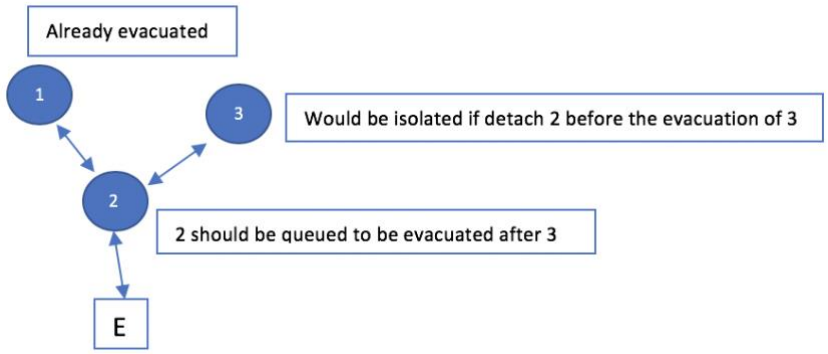


Figure 12

In Figure 12, after vertex 1 is removed, we are going to process vertex 2. By running a BFS algorithm starting from the exit, we could find vertex 3 be isolated if we process 2 ahead. So by using a DFS algorithm we can pick vertex 3 out (since in some complex situations, there might be plenty of vertices followed by vertex 3). Instead of evacuating vertex 2, we can process vertex 3 first to avoid isolation. This means that no room would be left out after the evacuation and the rooms closest to the emergency point would always be evacuated first.

2. Our model can work flexibly for evacuation in any structures and perform real-time optimization.

- This flexibility comes from the fact that our model does not fix the length of hallway or the amount of visitors in each room. Unconstrained variations of these two parameters allow our model to have more general applications in the real life. For example, in our solution presented above, we assume that the evacuation time above the same-level vertices [exhibition rooms] are the same, yet in real life, the length of each hallway and the number of people in each room differs, making it possible that one room evacuates faster. In this case, we can just timely detach that vertex [the room evacuated faster] from the algorithm and rerun it, keeping the evacuation plan up-to-date and flexible for real situation. No matter how many rooms are connected or isolated, our model can always work out a corresponding evacuation plan, which can also be timely updated with real-time monitoring.
- Because of the real-time monitoring and adjusting mechanism in our model, it can also deal with situations where multiple emergencies happen along the timeline.

3. Our model considers complex situation and unexpectancies and offers optimal solution accordingly.

The following bottlenecks that can typically happen in an evacuation are covered in our model:

- Route destruction situation
- Disabled as obstacle
- Considering stairs as a way for evacuation
- Multiple incident in the museum
- Unexpected obstruction of passageway
- Possible one-way network situation
- People not understanding and delayed in evacuating as designated

4. Our algorithm is not computationally intensive and can be calculated for large data sets in a relatively short amount of time.

Weakness

1. Our model assumes that people move at constant speed generally and deal with unusual movements as friction of the model so that it does not offers solution if groups of people change their speed.

2. Our model gives priority to minimizing the amount of time it takes for all the personnel to evacuate, and only allows entry of emergency personnel when the flow capacity start to decrease from the maximum level. In other words, this model does not consider the circumstance where a certain type of incidents that prioritize the earliest arrival of emergency personnel over the fastest evacuation.

Conclusion and Future Work

Our goal is to create a model (algorithm) to determine an optimal evacuation plan that minimizes the evacuation time for both the Louvre and any large, public and crowded structures. To accomplish this, we generate an algorithm that considers the following variables, the flow capacity, vertex connection, the number of people in each vertex, and the emergency spot. This algorithm provides a time-based comprehensive solution as to how many flow capacity should one vertex flow to other at a specific time spot. First, according to the layout of the vertices and how they are connected, we generate an adjacency matrix which shows the flow capacity and network relation in between vertices. Then we input the emergency spots and compute their depths. The vertex which has greater depth is assigned with higher priority to evacuate. Finally, we follow the priority to build the maximum flow and update state

(amount of people) of vertices according to the time. Once we monitor that the vertices with the largest depth are fully evacuated, we detach them and rerun the model until all vertices are evacuated finally.

We also conduct a sensitivity analysis on our model, from which we find that our model, because of the real-time updating algorithm, are flexible in adjusting to different unexpectancies such as the unexpected block downs or the existence of disabled people constraining the normal network flow. And it is tested to handle multi occurrence of unexpectancy with different level of impact on the network flow. Besides, our model even offers the time range for the inflow of rescue personnel as it monitors the real-time flow at exits. Although our model is only tested on the Louvre case, it is versatile and has general applicability towards evacuation at multiple situations.

Our model mainly focuses on the macro arrangement of the crowds to provide the most feasible evacuation plans. Meanwhile, by monitoring the evacuation situation, our model adjusts itself to output the most efficient plan at realtime. However, in real life, human beings are unpredictable. People are not always rational, so there will be more unexpected issues happened. Although we have conducted sensitivity test to simulate the occurrence of unexpectancies during the evacuation process, it only uses an approximate rate to compute the probability of emergencies that happen. To be more precise, it is better to introduce a more micro model that can cooperate with the current model we have. For example, by using the simulated annealing model, we can predict the potential accidental rate dynamically[7].

Reference

[1] Research on The Large Public Place Evacuation Model, https://wenku.baidu.com/view/10e9eb42e53a580217fcfebd.html
[2] Division of Maximum Flow, https://blog.csdn.net/M_AXSSI/article/details/50829795
[3] Mathematical Reflection of Wenchuan Earthquake, https://wenku.baidu.com/view/3b3cab16c850ad02de8041fc.html
[4] Basis of Multiple flow network, https://www.cnblogs.com/victorique/p/8560656.html#autoid-1-2-0
[5] Max-flow, Min-cut Theorem, https://blog.csdn.net/yjr3426619/article/details/82715779
[6] 10.2 MM visitors to the Louvre in 2018, http://presse.louvre.fr/10-2-million-visitors-to-the-louvre-in-2018/
[7] Simulated Annealing Modeling, https://www.cnblogs.com/Qling/p/9326109.html

Appendix A

data.json


```
[
   [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
   [0, 0, 96, 0, 0, 0, 0, 76.8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
   [0, 96, 0, 96, 0, 0, 96, 0, 76.8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
   [0, 0, 96, 0, 96, 0, 96, 0, 0, 76.8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
   [0, 0, 0, 96, 0, 96, 96, 0, 0, 0, 76.8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
   [0, 0, 0, 0, 96, 0, 0, 0, 0, 0, 0, 76.8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
   [0, 0, 96, 96, 96, 0, 0, 0, 0, 0, 0, 0, 144, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 240],
   [0, 76.8, 0, 0, 0, 0, 0, 0, 96, 0, 0, 0, 0, 76.8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
   [0, 0, 76.8, 0, 0, 0, 0, 96, 0, 96, 0, 0, 96, 0, 76.8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
   [0, 0, 0, 76.8, 0, 0, 0, 0, 96, 0, 96, 0, 96, 0, 0, 76.8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
   [0, 0, 0, 0, 76.8, 0, 0, 0, 0, 96, 0, 96, 96, 0, 0, 0, 76.8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
   [0, 0, 0, 0, 0, 76.8, 0, 0, 0, 0, 96, 0, 0, 0, 0, 0, 0, 76.8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
   [0, 0, 0, 0, 0, 0, 144, 0, 96, 96, 96, 0, 0, 0, 0, 0, 0, 0, 144, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 240],
```

```
[
    [0, 0, 0, 0, 0, 0, 0, 76.8, 0, 0, 0, 0, 0, 0, 96, 0, 0, 0, 0, 76.8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 240],
    [0, 0, 0, 0, 0, 0, 0, 0, 76.8, 0, 0, 0, 0, 96, 0, 96, 0, 0, 96, 0, 76.8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 76.8, 0, 0, 0, 0, 96, 0, 96, 0, 96, 0, 0, 76.8, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 76.8, 0, 0, 0, 0, 96, 0, 96, 96, 0, 0, 0, 76.8, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 76.8, 0, 0, 0, 0, 96, 0, 0, 0, 0, 0, 0, 76.8, 0, 0, 0, 0, 0, 0, 240],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 144, 0, 96, 96, 96, 0, 0, 0, 0, 0, 0, 0, 144, 0, 0, 0, 0, 0, 0, 240],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 76.8, 0, 0, 0, 0, 0, 0, 96, 0, 0, 0, 0, 76.8, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 76.8, 0, 0, 0, 0, 96, 0, 96, 0, 0, 96, 0, 76.8, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 76.8, 0, 0, 0, 0, 96, 0, 96, 0, 96, 0, 0, 76.8, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 76.8, 0, 0, 0, 0, 96, 0, 96, 96, 0, 0, 0, 76.8, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 76.8, 0, 0, 0, 0, 96, 0, 0, 0, 0, 0, 0, 76.8, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 144, 0, 96, 96, 96, 0, 0, 0, 0, 0, 0, 0, 144, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 76.8, 0, 0, 0, 0, 0, 0, 96, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 76.8, 0, 0, 0, 0, 96, 0, 96, 0, 0, 96, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 76.8, 0, 0, 0, 0, 96, 0, 96, 0, 96, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 76.8, 0, 0, 0, 0, 96, 0, 96, 96, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 76.8, 0, 0, 0, 0, 96, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 144, 0, 96, 96, 96, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 240, 0, 0, 0, 0, 0, 240, 240, 0, 0, 0, 240, 240, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
]
```

Appendix B

linkedListToMatrix.py

```
#!/usr/bin/env python3

"""
Modified Maximum Network Flow
Author: NHibiki
Date: Jan 24, 2019
"""

import json
from collections import deque

def exportToJson(data, fn="data.json"):
    with open(fn, "w") as f:
        f.write(json.dumps(data))

class Node:
    def __init__(self, id=0):
        self.c = []
        self.w = {}
        self.id = id

def connect(n1, n2, w):
    if n2 not in n1.c:
        n1.c.append(n2)
        n1.w[n2.id] = w
    if n1 not in n2.c:
        n2.c.append(n1)
        n2.w[n1.id] = w

### Generate Floor(n) [0 <= n <= 4]
### 0,1: -2, -1
### 2,3,4: 1,2,3
def generateFloor(n):
    nodes = [Node(i) for i in range(n*6+1, (n+1)*6+1)]
    for i in range(4):
        connect(nodes[i], nodes[i+1], 96)
    for i in [1,2,3]:
        connect(nodes[i], nodes[5], 96)
    return nodes

def dumpMap(size, node):
    queue = deque([node])
    visited = [False for i in range(size+1)]
    matrix = [[0 for j in range(size+1)] for i in range(size+1)]
    while len(queue):
        temp = queue.popleft()
        visited[temp.id] = True
        for c in temp.c:
```

```
            matrix[c.id][temp.id] = temp.w[c.id]
            matrix[temp.id][c.id] = temp.w[c.id]
            if not visited[c.id]:
                queue.append(c)
    return matrix

level = [generateFloor(i) for i in range(5)]
for i in range(4):
    for j in range(5):
        connect(level[i][j], level[i+1][j], 76.8)
    connect(level[i][5], level[i+1][5], 144)

exit = Node(5*6+1)
connect(exit, level[0][5], 240)
connect(exit, level[1][5], 240)
connect(exit, level[2][5], 240)
connect(exit, level[2][0], 240)
connect(exit, level[2][4], 240)

exportToJson(dumpMap(5*6+1, exit))
```

maxflow.py

```
#!/usr/bin/env python3

"""
Modified Maximum Network Flow
Author: NHibiki
Date: Jan 24, 2019
"""

import copy
import json
from collections import deque

def importFromJson(fn="data.json"):
    with open(fn, "r") as f:
        return json.loads(f.read())

def exportToJson(data, fn="data.json"):
    with open(fn, "w") as f:
        f.write(json.dumps(data))

INF = 999999999
EPT = 0

capacityMatrix = importFromJson()

E     = INF
V     = len(capacityMatrix)
```

```
start = 30
exit  = V-1

def hasPath(Gf, s, t, parent):
    # BFS Algorithm
    V = len(Gf)
    visited = [False for i in range(V)]
    visited[s] = True
    queue = deque([s])
    while len(queue):
        temp = queue.popleft()
        if temp == t:
            return True
        for i in range(V):
            if not visited[i] and Gf[temp][i] > 0:
                queue.append(i)
                visited[i] = True
                parent[i] = temp   # record precursor
    return visited[t]

def isMid(graph, s, t, globalVisited):
    # Normal BFS
    V = len(graph)
    visited = [False for i in range(V)]
    visited[s] = True
    visited[t] = True
    count = 2
    queue = deque([t])
    while len(queue):
        temp = queue.popleft()
        for i in range(V):
            if not visited[i] and graph[temp][i] > 0:
                queue.append(i)
                visited[i] = True
                count += 1
        if count == V:
            return False, None
    for i in range(V):
        if not visited[i] and graph[s][i] > 0 and not globalVisited[i]:
            return True, i
    return False, None

def showDef(id):
    level = (id-1) // 6 - 2
    if not level < 0:
        level += 1
    room  = (id-1) % 6 + 1
    if level > 3:
        return "EXIT"
    elif level > 0:
        return "L%dR%d" % (level, room)
```

```
    else:
        return "B%dR%d" % (-level, room)

def doMaxFlow(graph, s, t):
    maxFlow = 0
    answer = ""
    Gf = copy.deepcopy(graph)
    V = len(Gf)
    parent = list(range(V))
    while hasPath(Gf, s, t, parent):
        lineAnswer = ("%2s" % (showDef(t),))
        min_flow = float('inf')

        # find cf(p)
        v = t
        while v != s:
            if parent[v] != 0:
                min_flow = min(min_flow, Gf[parent[v]][v])
                lineAnswer += (" <= %2s" % (showDef(parent[v]),))
            v = parent[v]
        if lineAnswer:
            answer += ("[capacity = %3d] %s\n" % (min_flow, lineAnswer))

        # add flow in every edge of the argument parent
        v = t
        while v != s:
            Gf[parent[v]][v] -= min_flow
            Gf[v][parent[v]] += min_flow
            v = parent[v]

        maxFlow += min_flow
    return maxFlow, answer

def searchFlow(graph, s, t):
    V = len(graph)
    toVisit = [s]
    visited = [False for i in range(V)]
    step = 1
    while len(toVisit):
        # Build New Graph According to toVisit
        Gf = copy.deepcopy(graph)
        toVisitTemp = []
        for v in toVisit:
            globalVisited = copy.deepcopy(visited)
            ismid, parent = isMid(graph, v, t, globalVisited)
            while ismid:
                globalVisited[v] = True
                v = parent
                ismid, parent = isMid(graph, v, t, globalVisited)
            Gf[0][v] = INF
            visited[v] = True
            for i in range(V):
```

```
                if not visited[i] and graph[v][i] > 0 and i != t:
                    toVisitTemp.append(i)
        # Running MaxFlow
        maxFlow, answer = doMaxFlow(Gf, 0, t)
        print("Step %2d " % (step,), "[capacity = %3d]" % (maxFlow,))
        print(answer, "\n")
        step += 1
        toVisit = toVisitTemp
        # Remove Nodes From Graph
        for i in range(V):
            if Gf[0][i] == INF:
                for j in range(V):
                    graph[j][i] = 0

if __name__ == "__main__":
    searchFlow(capacityMatrix, start, exit)
```

generatingPlotData.py

```
import maxflow
import copy

from random import random
from collections import deque

def getDepth(Gf, s, t):
    if s == t:
        return 0
    V = len(Gf)
    visited = [False for i in range(V)]
    visited[s] = True
    queue = [s]
    depth = 1
    while len(queue):
        nextQueue = []
        for temp in queue:
            for i in range(V):
                if not visited[i] and Gf[temp][i] > 0:
                    if i == t:
                        return depth
                    nextQueue.append(i)
                    visited[i] = True
        queue = nextQueue
        depth += 1
    return -1

def hasPath(Gf, s, t, parent):
    # Priority BFS Algorithm
    V = len(Gf)
    visited = [False for i in range(V)]
```

```
    visited[s] = True
    queue = deque([s])
    while len(queue):
        temp = queue.popleft()
        if temp == t:
            return True
        if temp == s:
            # Apply priority on the first selection only
            linkedNodes = list(filter(lambda x: not visited[x] and Gf[temp][x] > 0,
range(V)))
            linkedNodes = sorted(linkedNodes, key=lambda x: getDepth(Gf, x, t),
reverse=True)
            queue = deque(linkedNodes)
        for i in range(V):
            if not visited[i] and Gf[temp][i] > 0:
                queue.append(i)
                visited[i] = True
                parent[i] = temp   # record precursor
    return visited[t]

def doMaxFlow(graph, s, t):
    maxFlow = 0
    answer = []
    Gf = copy.deepcopy(graph)
    V = len(Gf)
    parent = list(range(V))
    while hasPath(Gf, s, t, parent):
        lineAnswer = [t]
        min_flow = float('inf')

        # find cf(p)
        v = t
        while v != s:
            if parent[v] != 0:
                min_flow = min(min_flow, Gf[parent[v]][v])
                lineAnswer.append(parent[v])
            v = parent[v]
        lineAnswer.reverse()
        answer.append({
            "capacity": min_flow,
            "route"   : lineAnswer,
        })

        # add flow in every edge of the argument parent
        v = t
        while v != s:
            Gf[parent[v]][v] -= min_flow
            Gf[v][parent[v]] += min_flow
            v = parent[v]

        maxFlow += min_flow
    return maxFlow, answer
```

```
Graph   = copy.deepcopy(maxflow.capacityMatrix)
V       = len(Graph)
S       = 30
T       = V - 1
INF     = maxflow.INF
toVisit = [S]
visited = [False for i in range(V)]
step    = 1

simulateInterval = 1/60 # 1 second
remainings = [200 for i in range(V)]
remainings[ 0] = 0
remainings[-1] = 0

def init():
    global Graph, V, S, T, INF, toVisit, visited, step, simulateInterval, remainings
    Graph   = copy.deepcopy(maxflow.capacityMatrix)
    V       = len(Graph)
    S       = 30
    T       = V - 1
    INF     = maxflow.INF
    toVisit = [S]
    visited = [False for i in range(V)]
    step    = 1

    simulateInterval = 1/60 # 1 second
    remainings = [200 for i in range(V)]
    remainings[ 0] = 0
    remainings[-1] = 0

def stepFlow(graph):
    global V, S, T, INF, toVisit, visited, step, simulateInterval, remainings
    # Build New Graph According to toVisit
    G = copy.deepcopy(graph)
    isVisiting = [False for i in range(V)]
    for i in range(len(toVisit)):
        v = toVisit[i]
        globalVisited = copy.deepcopy(visited)
        ismid, parent = maxflow.isMid(graph, v, T, globalVisited)
        while ismid:
            globalVisited[v] = True
            v = parent
            ismid, parent = maxflow.isMid(graph, v, T, globalVisited)
        G[0][v] = INF
        isVisiting[v] = True
        if v != toVisit[i]:
            if v not in toVisit:
                toVisit[i] = v
            else:
                toVisit[i] = -1
    toVisit = list(filter(lambda x: x >= 0, toVisit))
```

```
    # Running MaxFlow
    maxFlow, routes = doMaxFlow(G, 0, T)
    # Check Status
    step += 1
    depths = list(map(lambda x: getDepth(graph, x["route"][0], T), routes))
    maxDepth = 0
    maxDepthCount = -1
    if depths:
        maxDepth = max(depths)
        maxDepthCount = len(list(filter(lambda x: x == maxDepth, depths)))
    # print(step-1, toVisit, depths, maxDepth)
    for index in range(len(routes)):
        r = routes[index]
        route = r["route"]
        cap   = r["capacity"]
        room  = route[0]
        # print(room, remainings[room], route, room in toVisit)
        remainings[room] -= cap * simulateInterval
        if remainings[room] < 0:
            remainings[room] = 0
        # Evacuation Complete
        if remainings[room] <= 0 and room in toVisit:
            # Rebuild toVisit
            toVisit.remove(room)
            for i in range(V):
                # Check if to Add new Nodes
                if not visited[i] and (i not in toVisit) and graph[room][i] > 0 and i != T:
                    toVisit.append(i)
                    visited[i] = True
        if remainings[room] <= 0 and depths[index] == maxDepth:
            maxDepthCount -= 1
    # Remove Nodes From Graph
    if maxDepthCount == 0:
        for index in range(len(routes)):
            room = routes[index]["route"][0]
            if remainings[room] <= 0 and depths[index] == maxDepth:
                for i in range(V):
                    graph[i][room] = 0
    return maxFlow, routes

stepResetMap = {}

def noShake(graph, prob, step):
    pass

def shake(graph, prob, step):
    global stepResetMap, Graph
    # Reset Path
    if step in stepResetMap:
        for s in stepResetMap[step]:
            if graph[s[0]][s[1]] == s[2]:
                graph[s[0]][s[1]] = Graph[s[0]][s[1]]
```

```
                # print("Recover @", s[0], "To", s[1])
    # Break Path
    for i in range(len(graph)):
        for j in range(len(graph[i])):
            if graph[i][j] > 0 and graph[i][j] == Graph[i][j] and random() < prob:
                rate   = random()     * 0.25 + 0.05 # 0.05 - 0.3
                during = int(random() * 20)  + 10   # 10   - 30
                reset  = step + during
                reduce = rate * graph[i][j]
                graph[i][j] = reduce
                if not reset in stepResetMap:
                    stepResetMap[reset] = []
                stepResetMap[reset].append((i, j, reduce))
                # print("Emergency @", i, "To", j, "Rate", rate, "During", during)

def searchFlow(shakeRate, shakeFn):
    global step, remainings, Graph
    init()
    g = copy.deepcopy(Graph)
    flow, routes = stepFlow(g)
    plot = [30*200, sum(remainings)]
    while len(routes) and sum(remainings):
        if shakeRate > 0:
            shakeFn(g, shakeRate, step)
        flow, routes = stepFlow(g)
        plot.append(sum(remainings))
        # print("Step:", step, "Remains:", sum(remainings))
    return plot

p = []
for i in range(1000):
    print("Step", i)
    p.append(searchFlow(0.01, shake))

maxflow.exportToJson(p, "plot.json")

# Plot
if __name__ == "__main__":
    import matplotlib.pyplot as plt
    p = list(map(lambda x: len(x), p))
    plt.figure()
    plt.plot(p)
    plt.show()
```