Alon Averbouch
Khush Negandhi
Kylie Wong
Mr Grigorov
ICS4UE-02
May 1 2022

Algorithm Assignment

The task is to decide where to build fire stations with a minimum number of fire stations, such that all cities are protected. The map is stored as a HashMap with Integer keys representing the city index and ArrayList<Integer> value representing the cities that are connected to it. We tried solving it in three different approaches: **brute-force algorithm** and two **recursive algorithms**.

Regarding the **data structure** of the map, we used an adjacency list of ArrayList<Node> at first, which the Node inner class has instance variables: ArrayList<Integer>, booleans of protected, visited, isFireStation. It is easy to implement, but inefficient in memory and time because it creates many objects, and insertion/deletion in ArrayList has a worst case time complexity of O(n). We replaced it with HashMap, considering the time complexity of insertion and deletion that only takes O(1). The locations of FS are stored in a HashSet so that there will be no duplicate. In removing connections between cities, HashSets are used to remove all intersections between delete targets and the elements in connection list.

For the **brute-force approach**, we used binary strings to generate **every possible combination** of fire stations and store the solution if it works and has a minimum number of FS. This approach returns the best possible result. However, it only works with a small map due to its inefficiency in time, with a time complexity of O(2^n). Then, we tried using recursion that travels through the Nodes, and uses if-statements to put the fire stations. It is fairly efficient in time with a time complexity of O(2^m), m represents the numbers of protected cities. But it didn't give the best solution because of the variety of connections. It is also inefficient in memory because we are using Node Objects and no object is deleted after every recursive call.

The final approach is a **recursive algorithm** that breaks the connection between cities after the city is a fire station or protected, so that **the map gets smaller each time and eventually becomes empty when every city is protected**. We also use ArrayList to store the location of the fire stations and the node that has been visited, instead of using the Node class. The time complexity is similar to the second approach that also uses a recursive algorithm,with O(mlog(m)2^n), but O(m^n) for the worst case when there is no node with a single child. But overall in comparison, it is slightly faster and uses much less memory.

Pseudocode of the recursive algorithm:

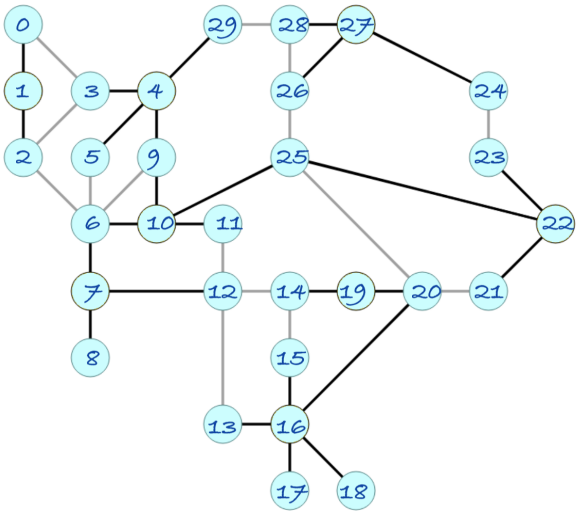
Base case: If the map is empty, return the fireStationList
If currentNode only has one child
    Set its child into a fire station(FS)
    Recursive call the protected nodes's children
Else - transverse the map until we find a node that only has one child
    If every node is visited, but there is no one node that satisfy the condition
        Set the node that with the connection a FS
        Recursive call the protected nodes's children

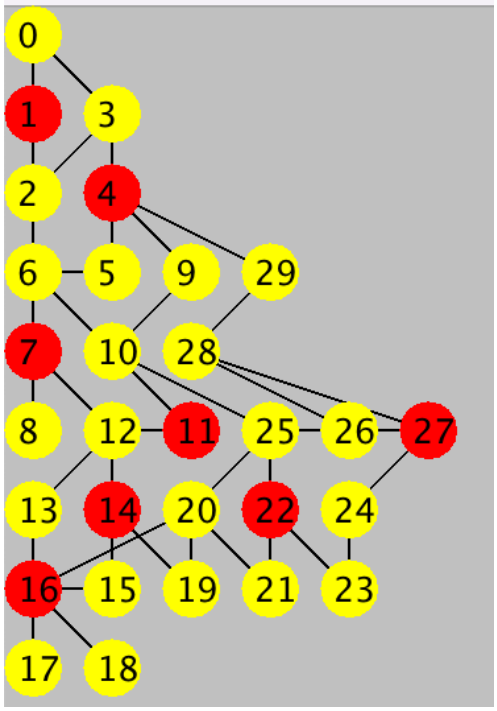
NOTE - Setting a node as FS requires these steps:
Add the node into fireStationList
Disconnect all node that is protected or is a FS, remove the nodes that have no connection from the map

---

DEMO CASES

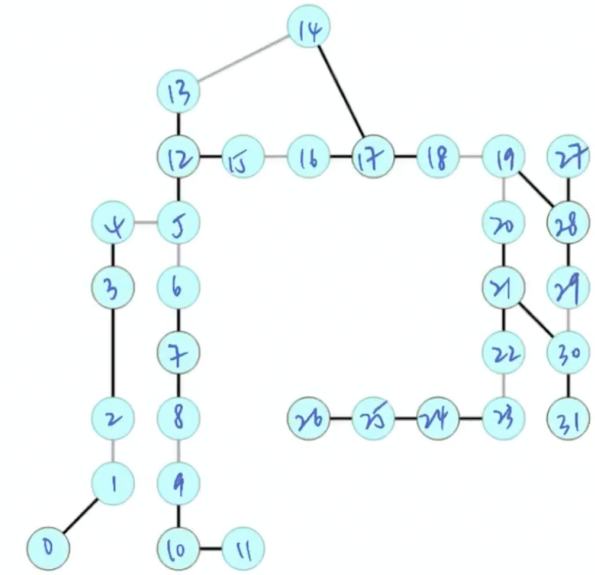
CommunityLayout_1:

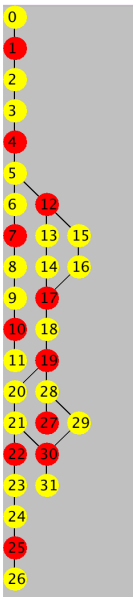





```
> run Main
Fire Station at: [1, 4, 7, 11, 14, 16, 22, 27]
   0 - 1, 3
FS 1 - 0, 2
   2 - 1, 3, 6
   3 - 0, 2, 4
FS 4 - 3, 5, 9, 29
   5 - 4, 6
   6 - 2, 5, 7, 10
FS 7 - 6, 8, 12
   8 - 7
   9 - 4, 10
   10 - 6, 9, 11, 25
FS 11 - 10, 12
   12 - 7, 11, 13, 14
   13 - 12, 16
FS 14 - 12, 15, 19
   15 - 14, 16
FS 16 - 13, 15, 17, 18, 20
   17 - 16
   18 - 16
   19 - 14, 20
   20 - 16, 19, 21, 25
   21 - 20, 22
FS 22 - 21, 23, 25
   23 - 22, 24
   24 - 23, 27
   25 - 10, 20, 22, 26
   26 - 25, 27, 28
FS 27 - 24, 26, 28
   28 - 26, 27, 29
   29 - 4, 28
```

CommunityLayout_2:





```
> run Main
Fire Station at: [1, 4, 7, 10, 12, 17, 19, 22, 25, 27, 30]
   0 - 1
FS 1 - 0, 2
   2 - 1, 3
   3 - 2, 4
FS 4 - 3, 5
   5 - 4, 6, 12
   6 - 5, 7
FS 7 - 6, 8
   8 - 7, 9
   9 - 8, 10
FS 10 - 9, 11
   11 - 10
FS 12 - 5, 13, 15
   13 - 12, 14
   14 - 13, 17
   15 - 12, 16
   16 - 15, 17
FS 17 - 14, 16, 18
   18 - 17, 19
FS 19 - 18, 20, 28
   20 - 19, 21
   21 - 20, 22, 30
FS 22 - 21, 23
   23 - 22, 24
   24 - 23, 25
FS 25 - 24, 26
   26 - 25
FS 27 - 28
   28 - 19, 27, 29
   29 - 28, 30
FS 30 - 21, 31
   31 - 30
```