Project 4

Written by Qi Chu, qichu6@gwu.edu

1 Problem Statement

Solve the problem: parking lot

2 Experimental Analysis

2.1 programming code —— mainly three classes

There are three classes to implement the program: class called vehicle is used to store the attributies of the cars in the layout; class called area is used to reflect the layout situation; class called BFS is used to implement the algorithm. The Graph search is accomplished by BFS.

The vehicle class and area class includes some constructors to initialize data. Particularly, the four directions: up, down, left, right are corresponding to 1, 2, 3, 4 to be easy to express later.

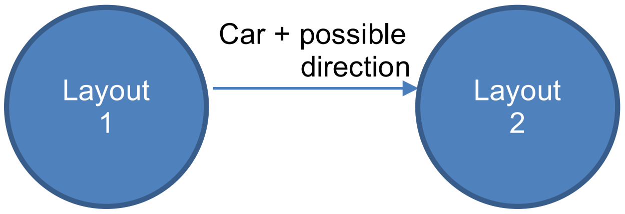
In the vehicle class, it stores some fixed data including the length, direction, coordinates of the cars in the layout. Besides, the cars are separated into two kinds according to whether it is the target one or not.

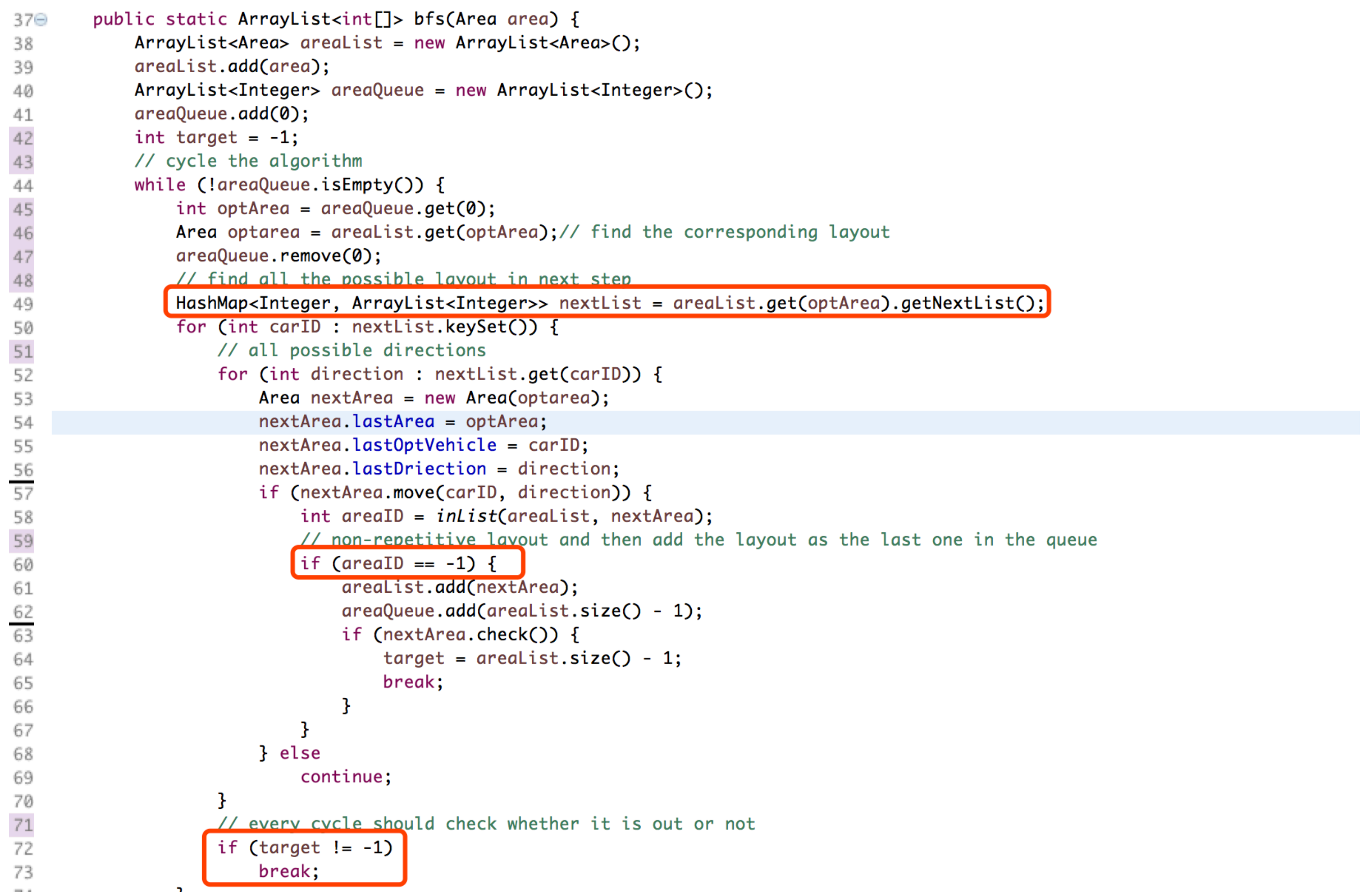
In the area class, first of all, it stores some constructors for the layout generating preparation by using arrayList to store those data. ArrayList<Vehicle> is a kind of arrayList<object> which can be used to store objects with its attributes. Every layout should be copied before it changes to the next step. resetArea is a function used to traceback to check whether the layout is reasonable and the cars can be placed in certain positions, if not, it will reset the layout into zeros without cars. The setCarSite function is used to put cars properly into one layout randomly. Then it comes to the important parts of the class. The getNextList function is used to calculate the adjacency lists (the arraylist is used to store all the edges of one node). To be specific, the function generates all the possible movement of the cars in one layout and it will be used to find all possibilities of the next node in BFS algorithm. I used HashMap to store the data as shown below. Since one car can move to different directions, the cars number is the key of HashMap while the value is a arryList stores its corresponding directions that can be moved. All data constitute the certain layout.

```
public HashMap<Integer, ArrayList<Integer>> getNextList() {
    HashMap<Integer, ArrayList<Integer>> result = new HashMap<Integer, ArrayList<Integer>>();
```

The move function is used to find the next layout according to current layout. It expresses the process moving from one node to another one. In the graph, the process means moving in one edge which represents one car and its possible moving directions. The check function is used to check whether the target car is out or not. Those two methods are all divided into four cases1, 2, 3, 4 according to its directions. The same function prevents searching repeatedly so that the layout can be trace back to a repetitive one.

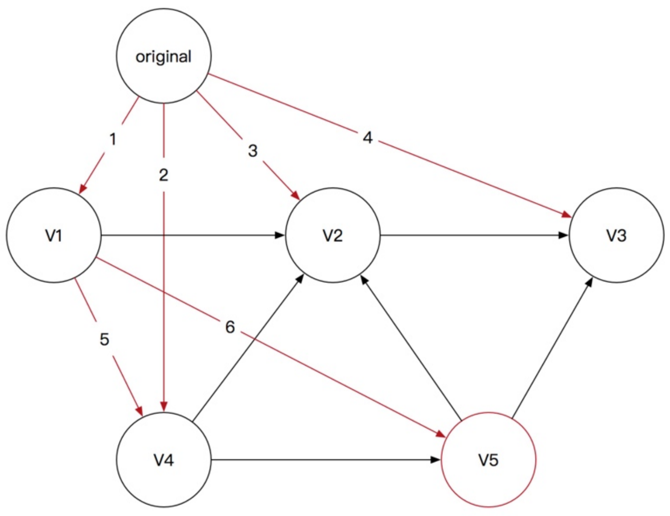
In the algorithm part which is BFS class, the ArrayList<int[]> bfs is the main part of the class. The nextList is used to get the adjacent lists to find the edges in the graph which one node can access to. One node is a layout and the adjacent list is used to store the moving method in graph. One element of nextList is one node's edge which is all possibilities from one layout to another. It can be described as the following diagram.





```java
    public static ArrayList<int[]> bfs(Area area) {
        ArrayList<Area> areaList = new ArrayList<Area>();
        areaList.add(area);
        ArrayList<Integer> areaQueue = new ArrayList<Integer>();
        areaQueue.add(0);
        int target = -1;
        // cycle the algorithm
        while (!areaQueue.isEmpty()) {
            int optArea = areaQueue.get(0);
            Area optarea = areaList.get(optArea);// find the corresponding layout
            areaQueue.remove(0);
            // find all the possible layout in next step
            HashMap<Integer, ArrayList<Integer>> nextList = areaList.get(optArea).getNextList();
            for (int carID : nextList.keySet()) {
                // all possible directions
                for (int direction : nextList.get(carID)) {
                    Area nextArea = new Area(optarea);
                    nextArea.lastArea = optArea;
                    nextArea.lastOptVehicle = carID;
                    nextArea.lastDriection = direction;
                    if (nextArea.move(carID, direction)) {
                        int areaID = inList(areaList, nextArea);
                        // non-repetitive layout and then add the layout as the last one in the queue
                        if (areaID == -1) {
                            areaList.add(nextArea);
                            areaQueue.add(areaList.size() - 1);
                            if (nextArea.check()) {
                                target = areaList.size() - 1;
                                break;
                            }
                        }
                    } else
                        continue;
                }
                // every cycle should check whether it is out or not
                if (target != -1)
                    break;
```

When there is a new layout (a new node), it should be checked whether it is the target layout (node) or not (target car is out). If it is, it will jump out of the loop and the algorithm will end. Otherwise, the new layout will be added into the queue.



The algorithm will begin from the original node and the target node is v5. Firstly, put the original node into the queue and begin the cycle. Get one node from the queue and get its adjacent list. Then, get the connected nodes from the list one by one and judge whether it is the exit or not. If it isn't the target node, this node will be put into queue. Otherwise, the algorithm will finish since we have found the target node. One cycle will be finished after every node has been judged then comes to another cycle. From the diagram, the adjacent list of original node are v1 v4 v2 v3 but none of them is target node. So they will be put into the queue in order. The adjacent list of v1 are V4 V5. V5 is the target node. Therefore, the searching sequence is original-1-2-3-4-5-6 while the final path is original->1->6->v5.

2.2 programming list

I used the layout with areaWidth being 5, areaHight being 6 and car number being 6. The max Car Length is 3. The original layout with different directions as well as length cars are generated randomly for 5000 times. I record the searchCout as the number of edges in the graph, the nodeCount as the number of nodes and the program run time. Since the initialization of running the program for the first time will take longer time, I repeated the experiment for 10 times and calculated the average results.

There are two parts of the analyzed data: has solution and no solution. When it has solution, since every layout should be checked whether it is the target one or not and the repeated layout will be delete, it will not search completely (some edges are deleted). If there is no solution, all edges will be searched and it will more trend to the theoretical values.

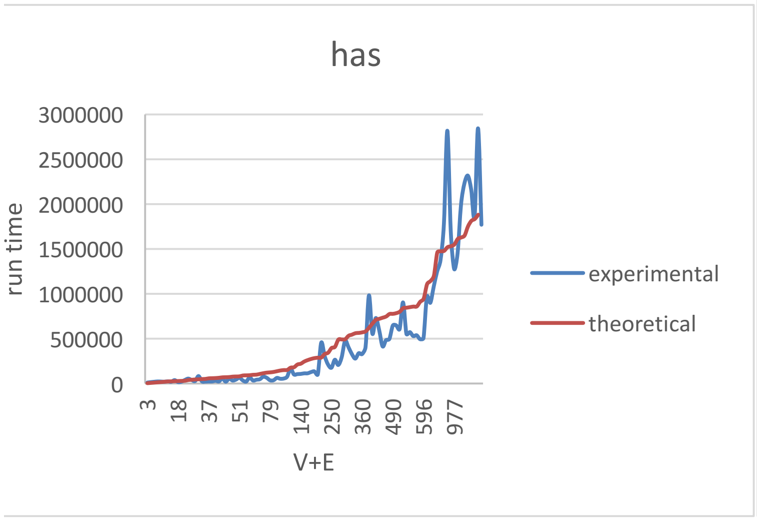
		has solution		
V+E	experimenta	theoretical	scaling	adjusted
3	10611	3		4756.04367
5	15239.7143	5		7926.73944
7	19309	7		11097.4352
9	22596.125	9		14268.131
11	21565.75	11		17438.8268
13	20206.8	13		20609.5226
15	23241	15		23780.2183
16	19105	16		25365.5662
17	37686	17		26950.9141
18	15859	18		28536.262
19	20208	19		30121.6099
20	39420	20		31706.9578
......	......	......	......	......
	604402.339	381.242718	1585.34789	

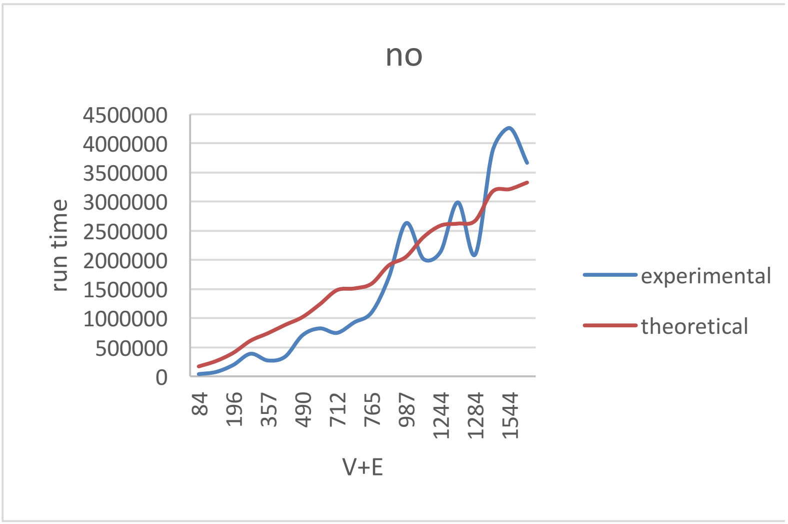
		no solution		
V+E	experimenta	theoretical	scaling	adjusted
84	41281	84		174776.57
128	77856	128		266326.201
196	198666	196		407811.996
296	390675	296		615879.341
357	274876	357		742800.421
426	341054.5	426		886366.889
490	707070	490		1019529.99
595	825713	595		1238000.7
712	748616	712		1481439.5
......	......	......	......	......
	1917435.16	921.545455	2080.67345	

2.3 Data Normalization Notes

The values were normalized by the scaling constant which was derived by the exception of experimental runtime result divided the exception of theoretical results. The scaling constant was shown as the table.

2.4 Graph and graph observation





As we can see, the experimental results of nosolution situation is more approached to the adjusted theoretical ones. The difference of two lines: the experimental result has lots of fluctuations because deleting the repeated layouts and storing the nextList layout. The initialization and manipulation of HashMap, especially the data copying can take up much time. Also, the CPU utilization will influence the program running in eclipse largely. The original layout is generated randomly each time so numbers of V+E can be different each time. Therefore, the experimental results are not very stable and higher than the adjusted theoretical ones.