Algorithms

Homework 2 Report

자유전공학부 김지원
2019-11563

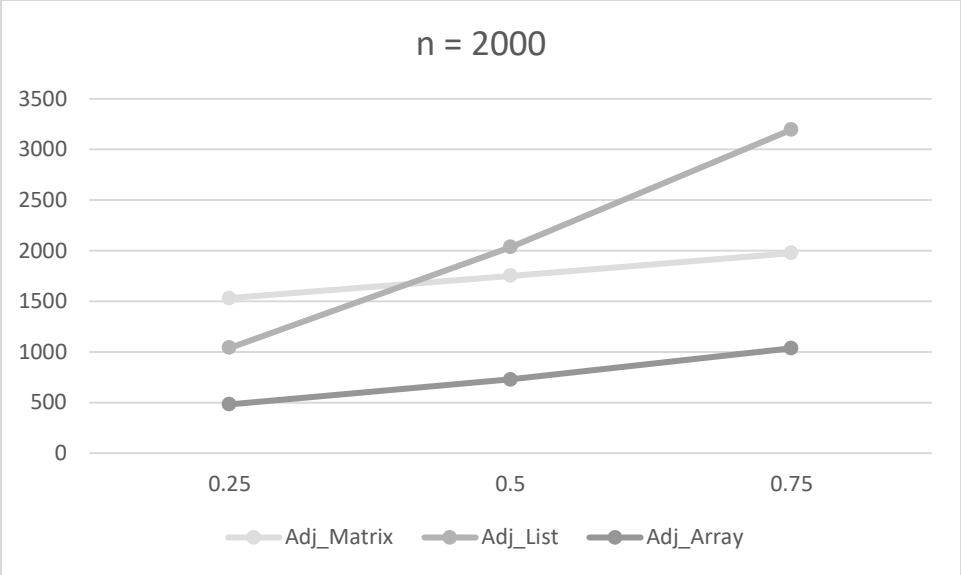
1. Compare the time usage of the three different data types; adjacency matrix, adjacency list, and adjacency array. (unit : ms)

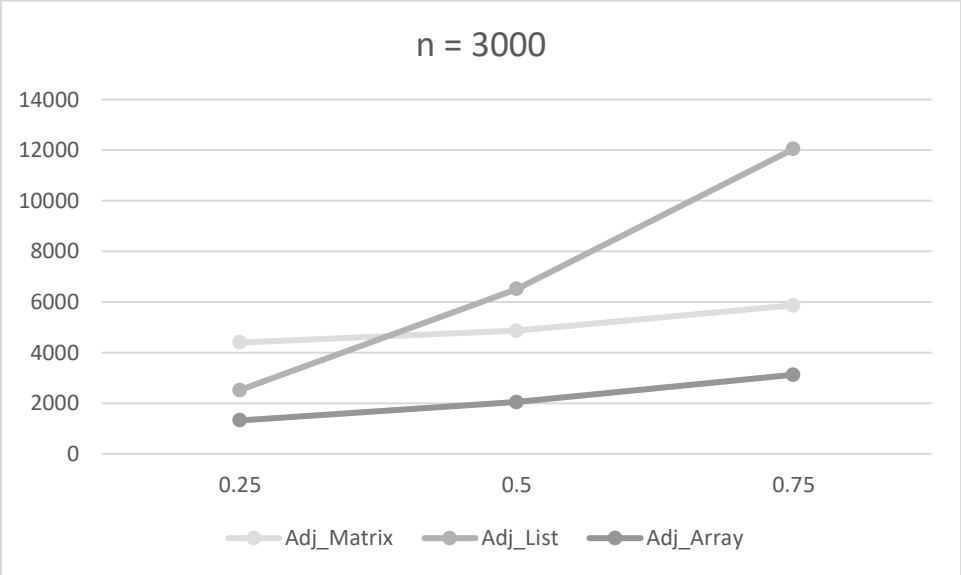
    a. Different values of **n** (**vertices**)

n, density	**Adj Matrix**	**Adj List**	**Adj Array**
1000, 0.25	154.01	5.98	5.21
1000, 0.5	179.89	56.25	26.21
1000, 0.75	198.37	107.74	67.56
2000, 0.25	1532.28	1039.12	481.64
2000, 0.5	1750.76	2035.52	727.07
2000, 0.75	1975.63	3194.82	1035.76
3000, 0.25	4398.83	2518.89	1317.48
3000, 0.5	4858.91	6504.79	2038.84
3000, 0.75	5867.62	12042.58	3123.54

Time is calculated by measuring both the time spent to create and populate a given data type with the given input, plus the time spent to retrieve arrays of strongly connected components. The input are generated randomly. The above matrix shows the different times depending on the number of vertices (value of n). Matrix are populated by measuring the time of one particular n value (ex. 1000) with three different densities(number of edges 3000, 30,000, and 300,000).
We can see that the adj list and adj array are approximately O(V + E) time, and adj matrix approximately O(V^2) time. The difference lies because of the data structure of adj_list, adj_array and adj_matrix. When initiating the data into each data structure and when iterating through the data to find sccs, adj_list and adj_array takes linear tine since it has an list/array with the size proportionate to the number of vertex, while adj_matrix has to initialize n*n size matrix and iterate through the matrix to find sccs.

    b. Different values of **m** / density (**edges**)





Regardless of the type, the time increases when it is denser, obviously because it needs to process more edges when initializing, and finding sccs. We need to focus that the time of adj_list and adj_array increases dramatically as the number of edges increase. However, adj_matrix time doesn't. It is because regardless of the number of edges, the data structure of adj_matrix is fixed to n*n. Rather for adj_list and adj_array, the list/array size increases as it is denser. That's why when the number of edges are small (sparse), the adj_matrix is worse than adj_list, but as the graph becomes denser, adj_matrix time becomes better than adj_list.

Also, based on the time above, we can see that the adj_array generally is the quickest since we can quickly access the adjacent vertices to each vertex.

2. Environment of the program, how to execute the program.

Open jdk – 19

Run the program by compiling the program ($ javac Main.java). Execute it with 3 command arguments. First argument with the version, second and last argument with each input and output file paths (ex.) $ java Main 1 ./input/1.in ./output/1.out)

Also, below is the code I used to generate random input for the time test. I used probability ‘density’ (0.25, 0.5, 0.75) to generate edges so the numbers are bigger than the given scope for N and M and it increases quadratically. Therefore the output time might also seem bigger than others.

```java
class InputGenerator {
    no usages
    public static void main(String[] args) {
        int numV = 1000;
        double density = 0.5;
        String outputPath = "./input/1.in";
        int numE = 0;

        Random random = new Random( seed: 125);

        try (FileWriter fw = new FileWriter(outputPath)) {
            fw.write( str: numV + "\n");

            StringBuilder line = new StringBuilder();
            for (int i = 1; i <= numV; i++) {
                //List<Integer> adj = new ArrayList<>();
                for (int j = 1; j <= numV; j++) {
                    if (i != j && random.nextDouble() < density) {
                        line.append(i).append(" ").append(j).append("\n");
                        numE ++;
                    }
                }
            }
            fw.write( str: numE + "\n");
            fw.write(line.toString());
```