**CSE 207:**

**DATA STRUCTURES & ALGORITHMS II SESSIONAL**

**OFFLINE**: 01

**TITLE**: IMPLEMENTATION OF GRAPH DATA STRUCTURE, IN ADJACENT MATRIX AND ADJACENT LIST FORMAT

**SUBMITTED BY,**

**SOUROV JAJODIA**

**STUDENT NO:** 1705072

**LEVEL** – 2, **TERM** – 2

DEPARTMENT OF CSE

BUET

**Objective:** To learn about “Graph Data Structure” and its two implementations, the Adjacent Matrix and Adjacent List method. Both implementations have their ups and downs. No one is better than the other fully. Both of their run times depend on the density of edges of the graph, and other factors. For space efficiency and sparse graphs, Adjacency list format is better. But, for time efficiency and denser graphs, Adjacency matrix format is much better. It depends on the usage on which implementation to use. Most of the graphs are usually sparse graphs. So, it is wise to use Adjacency list in real life applications, for its space efficiency.

**Machine Configuration:**

* **Processor:** Intel(R) Core(TM) i7 – 4600 CPU @ 2.10 GHz
* **Installed RAM:** 8.00 GB
* **OS:** Windows 10 Pro, Version 1903
* **Compiler:** g++ 5.4.1 c++11, CodeBlocks C/C++ Compiler

**Dataset:**

**FOR UNDIRECTED GRAPH: FOR DIRECED GRAPH:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | VERTICES:  1000 |  |  | VERTICES:  1000 |  |
| EDGES | ADJMATRIX  TIME IN MICROSECONDS | ADJLIST  TIME IN MICROSECONDS | EDGES | ADJMATRIX  TIME IN MICROSECONDS | ADJLIST  TIME IN MICROSECONDS |
| 1000 | 1599.1 | 5396.8 | 1000 | 0 | 100.2 |
| 2000 | 2397.1 | 11593.2 | 2000 | 1562.2 | 5197.1 |
| 4000 | 2499.9 | 21187.9 | 4000 | 1564.2 | 12643.2 |
| 8000 | 2398.5 | 40475.7 | 8000 | 3123.4 | 20388.3 |
| 16000 | 2698.5 | 89157.4 | 16000 | 1562.5 | 37878.1 |
| 32000 | 2998.8 | 168357.7 | 32000 | 3124.5 | 86397.2 |
| 64000 | 3797.9 | 304896.6 | 64000 | 3124.8 | 156254.6 |
|  | VERTICES:  2000 |  |  | VERTICES:  2000 |  |
| EDGES | ADJMATRIX  TIME IN MICROSECONDS | ADJLIST  TIME IN MICROSECONDS | EDGES | ADJMATRIX  TIME IN MICROSECONDS | ADJLIST  TIME IN MICROSECONDS |
| 2000 | 6197.8 | 28183 | 2000 | 0 | 201.3 |
| 4000 | 9396.1 | 48932.8 | 4000 | 6248.8 | 22396 |
| 8000 | 9694.5 | 85495.1 | 8000 | 9374.4 | 46575.4 |
| 16000 | 9794.4 | 166618.7 | 16000 | 9376 | 79896.2 |
| 32000 | 10195.6 | 395698.2 | 32000 | 9374.3 | 153271.8 |
| 64000 | 10693.9 | 688889.6 | 64000 | 9372.8 | 347462.6 |
| 128000 | 11894.7 | 1283111.3 | 128000 | 10729.2 | 639256.3 |
| 256000 | 14691.7 | 2456700.8 | 256000 | 11894.6 | 1209393 |
|  | VERTICES:  4000 |  |  | VERTICES:  4000 |  |
| EDGES | ADJMATRIX  TIME IN MICROSECONDS | ADJLIST  TIME IN MICROSECONDS | EDGES | ADJMATRIX  TIME IN MICROSECONDS | ADJLIST  TIME IN MICROSECONDS |
| 4000 | 21487.7 | 90149.7 | 4000 | 199.8 | 199.8 |
| 8000 | 33685.9 | 206161.6 | 8000 | 20396 | 68434.2 |
| 16000 | 38608.8 | 338837.3 | 16000 | 37495.7 | 184402 |
| 32000 | 42783.4 | 673400.3 | 32000 | 37234 | 318689.4 |
| 64000 | 40422.1 | 1547430.9 | 64000 | 39059.5 | 610416.1 |
| 128000 | 42193.1 | 2762849.7 | 128000 | 40427.7 | 1432738.4 |
| 256000 | 43876.3 | 5321959.8 | 256000 | 39779 | 2578765.6 |
| 512000 | 48475.5 | 9643919.6 | 512000 | 40662.6 | 4877126.7 |
| 1024000 | 63166.8 | 15287839.2 | 1024000 | 47234.1 | 9470437.8 |
|  | VERTICES:  8000 |  |  | VERTICES:  8000 |  |
| EDGES | ADJMATRIX  TIME IN MICROSECONDS | ADJLIST  TIME IN MICROSECONDS | EDGES | ADJMATRIX  TIME IN MICROSECONDS | ADJLIST  TIME IN MICROSECONDS |
| 8000 | 114015.9 | 305285.9 | 8000 | 3124.7 | 400.2 |
| 16000 | 156474.3 | 754655.4 | 16000 | 95614.4 | 294010.4 |
| 32000 | 155781.1 | 1343448.7 | 32000 | 146757.7 | 676728.6 |
| 64000 | 155906.8 | 2454219.7 | 64000 | 148303 | 1242781.3 |
| 128000 | 155496.5 | 5529173.1 | 128000 | 152134.2 | 2432923.8 |
| 256000 | 155488.8 | 10158346.2 | 256000 | 150261.9 | 5524041.6 |
| 512000 | 165765.2 | 19916692.3 | 512000 | 153863 | 10293525.9 |
| 1024000 | 172496.9 | 31833384.8‬ | 1024000 | 156565.5 | 19739589.1 |
| 2048000 | 192016.8 | 56666769.6‬ | 2048000 | 165086.9 | 38819251.4 |
| 4096000 | 240062.9 | 101333539.2‬ | 4096000 | 184007.7 | 73073714.8 |
|  | VERTICES:  16000 |  |  | VERTICES:  16000 |  |
| EDGES | ADJMATRIX  TIME IN MICROSECONDS | ADJLIST  TIME IN MICROSECONDS | EDGES | ADJMATRIX  TIME IN MICROSECONDS | ADJLIST  TIME IN MICROSECONDS |
| 16000 | 418483.8 | 1206989.4 | 16000 | 0 | 2100.4 |
| 32000 | 622753.4 | 3912365.2 | 32000 | 381215.8 | 1345230.6 |
| 64000 | 605165.6 | 7257605.5 | 64000 | 756140.3 | 2876951.3 |
| 128000 | 601438.2 | 11504040.1 | 128000 | 844537.9 | 4957216 |
| 256000 | 603082.5 | 22468080.7 | 256000 | 683909 | 9655756 |
| 512000 | 607737.6 | 39962314.4 | 512000 | 622650.4 | 17311512 |
| 1024000 | 616071.4 | 73924628.8 | 1024000 | 631346.5 | 32623024 |
| 2048000 | 654057.9 | 140849257.6 | 2048000 | 623498.1 | 62246048 |
| 4096000 | 687145.2 | 269698515.2 | 4096000 | 631884 | 114492096 |
| 8192000 | 815876.2 | 1288492576 | 8192000 | 669618.2 | 226984192 |
| 16384000 | 1025419.4 | 2416985152 | 16384000 | 825287.5 | 455968384 |

\*\*These data represent the time required to run 1 BFS on average for both directed and undirected graph, and also Adjacent List and Adjacent Matrix implementation.

**Question / Answer:**

**1. What is the impact on runtime if we keep |V| unchanged and double |E| for adjacency list? Why is it so?**

**Answer:**

**For Directed Graph:** If we double |E|, then the runtime is little bit more than double.

**For Undirected Graph:** If we double |E|, then the runtime is almost double.

This is because, here, for searching a particular vertex in an arrayList, we have to run the searchItem function. The searchItem function of the Class ArrayList searches an item in linear time. So, it has O(n) complexity. So, as the number of edges increase, it is probable that the number of vertices in the arrayList of each vertex will double as the number of edges increase in double. So, this signifies that the number “n” will also double, leading to double the runtime. But still, a little more time is required, for additional computation.

**2. What is the impact on runtime if we keep |E| unchanged and double |V| for adjacency list? Why is it so?**

**Answer:**

**For Directed Graph:** if we double |V|, then the runtime is almost double.

**For Undirected Graph:** If we double |V|, then the runtime is slightly more than double.

This is because, when we double the number of vertices, but still keep a fixed number of edges, the searchItem function for the arrayList does not take more time, as the number of edges is fixed. But, as we increase the number of vertex, the complexity increases. Because, in BFS operation, we check if each vertex has an edge with each vertex we’re operating on. As the number of vertex increases, this loop becomes bigger. So, as we double the number of vertices, the time required also doubles, leading to almost double the time in double vertices scenarios.

**3. What is the impact on runtime if we keep |V| unchanged and double |E| for adjacency matrix? Why is it so?**

**Answer:**

**For Directed Graph:** If we double edges, at lower vertices number, the time is a little bit greater than previous one, but as the number is higher, the time increases gradually.

**For Undirected Graph:** It’s run time analysis is almost like Directed Graph situation.

This is because, when the number of vertices increase, it really does not bring any significant change in runtime for adjacent matrix implementation. Because, searching for edge between two vertices in Adjacent Matrix implementation is O(1), and can be done in constant time. As the number of edges increase, the time for BFS increases in constant time. So, when the number of edges is low, the constant time increase is insignificant. But, as the number of edges becomes significantly high, the time required becomes almost the same, even if we double the number of edges. For undirected graph, the time is just a little bit high, because we have to check edge both ways.

**4. What is the impact on runtime if we keep |E| unchanged and double |V| for adjacency matrix? Why is it so?**

**Answer:**

**For Directed Graph:** If we double |V|, the time required is almost 4 times of the one before.

**For Undirected Graph:** If we double |V|, the time required is slightly more than 4 times of the one before.

This is because, when the number of vertices are doubled, the BFS function runs higher. For each vertex the function visits, it runs a loop up to the total number of vertices to check if there was an edge between them. So, as the number of vertices increase, this loop runs at almost double the time. And, as this is an adjacent matrix, if the number of vertices are doubled, the size of the matrix becomes 4 times the previous size. This means, that we have to check 4 times more cells than the previous number of vertices. This means, that the runtime must also increase by 4 times the previous time, which is exactly the case. And for undirected graph, we have to check both ways to check if there is an edge between two vertices. So, slightly more time is needed.

**5. For the same |E| and |V|, why are the runtimes for adjacency list and adjacency matrix representation different? Which one is higher and why?**

**Answer:** This is because, in BFS, we actually run a search operation to check if there exists an edge between to vertices, which is a major step in running BFS.

In adjacency matrix representation, we do this search by just check if by accessing a specific cell in a matrix is a constant time (O(1)) operation.

But for adjacency list representation, we do this search by running the searchItem function, which is a function which checks the whole arrayList that we are working with at that moment to check if a specific edge exists. So, this function has to traverse the whole list to find the edge, which is a linear time (O(n)) operation.

Thus, it is significantly higher than the matrix representation’s search operation, and also takes significantly more time. This is also observed in the table given above, where we can see that the time required for adjacency list representation for same number of |E| and |V| is always higher than the time required for adjacency matrix representation.