Time Complexity Analysis of Breadth First Search Algorithm

Objective:

This report is based on the analysis the time complexity of Breadth First Search algorithm in two different representation named “Adjacency List “and “Adjacency Matrix”. Theoretically, when graph is represented in Adjacency list way then time complexity for BFS algorithm is O (|V| + |E|) in big-Oh sense and when the graph is represented in matrix form then the complexity is O(|V|2) in big-Oh sense. Depending on the size of the list of and matrix, from our time limit analysis we have to determine best algorithm for these two representations and this was done in this assignment.

Machine Configuration:

Windows edition : Windows 10 Pro

Processor : Inter® Core™ i5-7200U CPU @ 2.50GHz

Installed memory (RAM): 8.00 GB (7.88 GB usable)

System Type : 64-bit operation System, x64-based processor

Runtime Table of BFS Algorithm:

|  |  |  |  |
| --- | --- | --- | --- |
| Vertices | Edges | Time for Adj. List (nanoseconds) | Time for Adj. Matrix  (nanoseconds) |
| 1000 | 1000 | 0 | 0 |
| 1000 | 2000 | 99770 | 2892670 |
| 1000 | 4000 | 99640 | 3191430 |
| 1000 | 8000 | 199370 | 3291310 |
| 1000 | 16000 | 299200 | 3390910 |
| 1000 | 32000 | 698420 | 3793130 |
| 1000 | 64000 | 1196510 | 4887830 |
| 2000 | 2000 | 99990 | 0 |
| 2000 | 4000 | 199540 | 11369670 |
| 2000 | 8000 | 299160 | 12267170 |
| 2000 | 16000 | 399210 | 12362500 |
| 2000 | 32000 | 997300 | 13563620 |
| 2000 | 64000 | 1396050 | 13367970 |
| 2000 | 128000 | 2489990 | 16755120 |
| 2000 | 256000 | 4588000 | 19353720 |
| 4000 | 4000 | 0 | 299190 |
| 4000 | 8000 | 399050 | 48968890 |
| 4000 | 16000 | 598250 | 48669730 |
| 4000 | 32000 | 897580 | 54250140 |
| 4000 | 64000 | 1495710 | 52191840 |
| 4000 | 128000 | 2689710 | 52066060 |
| 4000 | 256000 | 4886920 | 59453730 |
| 4000 | 512000 | 9374930 | 66727060 |
| 4000 | 1024000 | 18251100 | 76192380 |
| 8000 | 8000 | 103140 | 99690 |
| 8000 | 16000 | 901460 | 175231600 |
| 8000 | 32000 | 1196780 | 205283430 |
| 8000 | 64000 | 1997740 | 191370280 |
| 8000 | 128000 | 3288300 | 188495610 |
| 8000 | 256000 | 6881470 | 192382020 |
| 8000 | 512000 | 10174690 | 195521910 |
| 8000 | 1024000 | 19843580 | 209439510 |
| 8000 | 2048000 | 36906860 | 243942150 |
| 8000 | 4096000 | 75104450 | 322437240 |
| 16000 | 16000 | 99690 | 4288660 |
| 16000 | 32000 | 2398570 | 708906070 |
| 16000 | 64000 | 3186550 | 739319170 |
| 16000 | 128000 | 4294120 | 789399730 |
| 16000 | 256000 | 7081020 | 835990820 |
| 16000 | 512000 | 11967970 | 749694980 |
| 16000 | 1024000 | 21239710 | 762062080 |
| 16000 | 2048000 | 39597440 | 784328380 |
| 16000 | 4096000 | 75791350 | 827983270 |
| 16000 | 8192000 | 147807020 | 943042610 |
| 16000 | 16384000 | 292965180 | 1207277830 |

Analysis:

From the analysis of time it is clear that time taken for adjacency list represented graph is much less than adjacency matrix represented graph as complexity for adj. matrix is O(|V|2) which is significantly larger than adj. list represented graph as complexity is O(|V| + |E|) when |V| >> |E|.

So, depending on the above discussion, it is clear that adj. list is better than adj. matrix.

Question Answer:

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

Ans.

From mathematical analysis, we know that time complexity for Breadth First Search (BFS) is O (|V| + |E|) where G = (V, E) and here V = # of vertices and E = # of edges of the graph.

Now, if we keep |V| unchanged and double |E| for adjacency list then two different cases arise for dense graph and sparse graph.

Case-1: When graph is dense:

When graph is dense graph then |E|>> |V| mathematically |E| = O(|V|2) so, O (|V| + |E|) becomes nearly equal to O(|E|).

So, for this case if we double |E| then runtime for the graph will become almost double.

Case-2: When graph is sparse:

A sparse graph is a graph G = (V, E) in which |E| = O(|V|) So for this case if we double |E| then runtime for the graph will rise slowly.

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

Ans:

From first answer it is clear that if |v| remains same and |E| is being changed then time will change depending on the edge density of the graph.

But for this case, the run time will also increase when |E| remains same and |V| is being changed.

When, |V| >> |E| then O(|V|+|E|) become O(|V|). So, when we double the number of vertices then run will increase almost twice.

But for |E| >> |V| then O(|V|+|E|) become O(|E|). For this case, runtime will increase slowly.

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

Ans:

For analysis of time complexity for BFS algorithm when the graph is represented in adjacent matrix representation then runtime is in O (|V|2).

So, if the number of edges is doubled but number of vertices remain constant then runtime will increase but not in polynomial order of 2. It will increase slightly.

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

Ans:

For analysis of time complexity for BFS algorithm when the graph is represented in adjacent matrix representation then runtime is in O (|V|2)

So, if the number of vertices is doubled but number of edges remain constant then runtime will increase and will be almost twice.

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?

Ans:

For the same |E| and |V|, the runtime for adjacency matrix is higher because of the more iteration for each vertex.

Mathematically it is clear that, time complexity of BFS algorithm

When represented in Adjacency List form is O(|V|+|E|) and

When represented in Adjacency Matrix form is O(|V|2)

As during bfs algorithm, we have to iterate all the vertices and as in matrix representation we keep records in a matrix of size |V|\*|V| so we have to iterate all over the vertices for V\*V times so time complexity grows as vertices grows but in list representation it grows in linear combination of # of edges and vertices so time taken for adjacency list is reasonably smaller than adjacency matrix.