**CSE 208**

**Data Structures & Algorithms-II**

**Runtime analysis of BFS for adjacency matrix and adjacency list representations**

Time taken to perform **BFS** operation with various number of vertices and edges is given in the below tables:

Adjacency Matrix representation with undirected edges:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Number of Vertices | Time taken for **N\*|V|** number of Edges ( µs) | | | | | | | | | | |
| |V| | 2|V| | 4|V| | 8|V| | 16|V| | 32|V| | 64|V| | 128|V| | 256|V| | 512|V| | 1024|V| |
| 1000 | 2689.4 | 2596.4 | 2596.5 | 2592.9 | 2596.5 | 2596.4 | 2596.5 |  |  |  |  |
| 2000 | 10468.7 | 10578 | 10472 | 10472.1 | 10471.9 | 10475.2 | 10475.3 | 10372.2 |  |  |  |
| 4000 | 41692.1 | 41688.3 | 41588.6 | 41888 | 41787.8 | 41791.9 | 41588.8 | 41688.3 | 41691.8 |  |  |
| 8000 | 166156 | 165856 | 165860 | 165956 | 165856 | 165657 | 166053 | 166355 | 165757 | 166050 |  |
| 16000 | 654649 | 654848 | 654549 | 654150 | 657238 | 660234 | 655151 | 654047 | 654449 | 654250 | 654150 |

Adjacency Matrix representation with directed edges:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Number of Vertices | Time taken for **N\*|V|** number of Edges ( µs) | | | | | | | | | | |
| |V| | 2|V| | 4|V| | 8|V| | 16|V| | 32|V| | 64|V| | 128|V| | 256|V| | 512|V| | 1024|V| |
| 1000 | 2692.8 | 2692.6 | 2692.8 | 2596.2 | 2593.1 | 2692.6 | 2696.3 |  |  |  |  |
| 2000 | 10477.9 | 10475.6 | 10372.1 | 10472 | 10471.9 | 10471.9 | 10471.9 | 10671.5 |  |  |  |
| 4000 | 41588.7 | 42286.7 | 41588.8 | 41588.8 | 41888.1 | 41488.7 | 41788.2 | 41788.2 | 41987.9 |  |  |
| 8000 | 166255 | 166355 | 166156 | 166455 | 166155 | 166352 | 165856 | 166654 | 166056 | 165857 |  |
| 16000 | 655447 | 655546 | 654749 | 654849 | 653854 | 654253 | 667366 | 658252 | 658398 | 658446 | 656947 |

Adjacency List representation with undirected edges:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Number of Vertices | Time taken for **N\*|V|** number of Edges ( µs) | | | | | | | | | | |
| |V| | 2|V| | 4|V| | 8|V| | 16|V| | 32|V| | 64|V| | 128|V| | 256|V| | 512|V| | 1024|V| |
| 1000 | 0 | 99.8 | 99.7 | 99.8 | 203.2 | 399.2 | 698.1 |  |  |  |  |
| 2000 | 99.6 | 99.7 | 99.7 | 299.2 | 398.9 | 794.5 | 1496 |  |  |  |  |
| 4000 | 199.5 | 199.5 | 299.2 | 498.7 | 894.2 | 1595.7 | 3091.7 |  |  |  |  |
| 8000 | 299.3 | 498.8 | 598.2 | 897.8 | 2000.6 | 3191.5 | 6180.3 |  |  |  |  |
| 16000 | 598.4 | 797.9 | 1196.9 | 1795.3 | 3391 | 6684.7 | 12167.3 |  |  |  |  |

Adjacency List representation with directed edges:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Number of Vertices | Time taken for **N\*|V|** number of Edges ( µs) | | | | | | | | | | |
| |V| | 2|V| | 4|V| | 8|V| | 16|V| | 32|V| | 64|V| | 128|V| | 256|V| | 512|V| | 1024|V| |
| 1000 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  |  |  |  |
| 2000 | 0 | 0 | 0 | 0 | 99.7 | 0 | 0 |  |  |  |  |
| 4000 | 0 | 0 | 0 | 99.7 | 0 | 0 | 99.8 |  |  |  |  |
| 8000 | 99.7 | 99.7 | 99.7 | 99.6 | 0 | 99.8 | 0 |  |  |  |  |
| 16000 | 99.8 | 99.8 | 99.9 | 99.8 | 199.5 | 99.9 | 99.6 |  |  |  |  |

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

**Ans:** In Adjacency list representation the time complexity of Breadth first search algorithm is O(|V|+|E|). If we keep |V| unchanged and double |E|, the result winds up in a more densely connected graph. The event has no impact on the time complexity of the algorithm asymptotically. But it has a huge effect on the runtime as we can see from the above table. Here, in case of directed graph it mostly depends on the source vertex or the starting point. In a directed graph the source may not have any adjacent vertices, in such cases the BFS(source) will fail to travel all the vertices. That is why in directed graph even if we double the edges the runtime may remain the same. On the contrary, significant changes can be observed in undirected adjacent graph. In undirected adjacent graph, the runtime almost doubles when |E| becomes two times the number of vertices.

It is because in undirected graph each vertex has more adjacent vertices than its directed counterpart. So it is more likely that all the vertices will be visited.

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

**Ans:** Like the previous answer, we can focus on undirected adjacency List, as the directed graph has very less impact on the runtime. When |V| is much smaller than |E|, doubling |V| has very little effect to no effect on the runtime. However, if |E| and |V| are almost similar or |V| has a dominant figure over |E|, then keeping |E| unchanged and doubling |V| will have a noticeable impact on the runtime. It can be perceived from the table of adjacency list with undirected edges.

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

**Ans:** Previously we have seen how much effect the adjacent list creates on the runtime by changing |E| or |V|. Now, in case of adjacency matrix this kind of |E| or |V| changing has very less impact on the runtime, which is not as much as we perceived in case of adjacency list representation. The effect is so less is sometimes it is almost unnoticeable. It is because the time complexity of BFS in adjacency matrix is O(|V|2). So, no matter the number of edges is, the algorithm searches for adjacent vertices |V| times in every vertex. Although the runtime may slightly vary, which is negligible. The only reason behind this is, the runtime of BFS in adjacency matrix mostly depends on |V|.

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

**Ans:** As we have discussed in the answer to the earlier question that, the runtime of BFS in adjacency matrix mostly depends on |V| with time complexity of O(|V|2). Therefore, it goes without saying that, doubling the value of |V| with unchanged |E| will cause a serious impact on the runtime.

The runtime almost quadruples in such a scenario.

**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:** Adjacency list uses time O(|V|+|E|) while adjacency matrix uses O(|V|2). So in general adjacency matrix takes more time. It is because of the different ways these two representations store and manipulate data.

In BFS with adjacency list representation, we only look for an adjacent node to shove into a queue. Thus, it takes O(|V|+|E|). However, we run a loop over all the vertices, present in the graph in adjacency matrix representation and then check which vertex is adjacent and which is not. This event takes place for all the vertices. Thus the time it takes is O(|V|2).

By this way the runtime of adjacency matrix representation is higher. But as the graph becomes dense, the runtime of both become approximately equal.