CSE 208

Name :Maisha Rahman

ID: 1705060

Sec: A2

Machine Specifications:

Processor : Intel® Core™ i3-7100U CPU @ 2.40 GHz 2.40 GHz

Installed memory (RAM) : 4.00 GB (3.88 GB usable)

OS : Windows 10 Home (64 Bit)

Compiler : GNU GCC

Runtime Analysis of BFS:

When number of vertices is 1000:

|  |  |  |
| --- | --- | --- |
| Number of Edges | Runtime for Adjacency Matrix  (micro seconds) | Runtime for Adjacency list  (microseconds) |
| 1000 | 179 | 50 |
| 2000 | 161 | 50 |
| 4000 | 20446 | 638 |
| 8000 | 40845 | 11133 |
| 16000 | 50674 | 15880 |
| 32000 | 57610 | 22160 |
| 64000 | 71050 | 37050 |

When number of vertices is 2000:

|  |  |  |
| --- | --- | --- |
| Number of Edges | Runtime for Adjacency Matrix  (micro seconds) | Runtime for Adjacency list  (microseconds) |
| 2000 | 2146 | 147 |
| 4000 | 6571 | 110 |
| 8000 | 11193 | 5169 |
| 16000 | 13869 | 7832 |
| 32000 | 19060 | 9989 |
| 64000 | 20738 | 13967 |
| 128000 | 19079 | 18668 |
| 256000 | 20889 | 19897 |

When number of vertices is 4000:

|  |  |  |
| --- | --- | --- |
| Number of Edges | Runtime for Adjacency Matrix  (micro seconds) | Runtime for Adjacency list  (microseconds) |
| 4000 | 800 | 24 |
| 8000 | 5099 | 20 |
| 16000 | 20171 | 278 |
| 32000 | 38987 | 578 |
| 64000 | 47896 | 720 |
| 128000 | 48008 | 2923 |
| 256000 | 49799 | 7713 |
| 512000 | 50884 | 21619 |
| 1024000 | 54963 | 39486 |

When number of vertices is 8000:

|  |  |  |
| --- | --- | --- |
| Number of Edges | Runtime for Adjacency Matrix  (micro seconds) | Runtime for Adjacency list  (microseconds) |
| 8000 | 199 | 48 |
| 16000 | 181 | 43 |
| 32000 | 111912 | 709 |
| 64000 | 182054 | 1814 |
| 128000 | 203959 | 9604 |
| 256000 | 204010 | 17810 |
| 512000 | 205941 | 19307 |
| 1024000 | 210975 | 38359 |
| 2048000 | 220925 | 61857 |
| 4096000 | 232899 | 173067 |

When number of vertices is 16000:

|  |  |  |
| --- | --- | --- |
| Number of Edges | Runtime for Adjacency Matrix  (micro seconds) | Runtime for Adjacency list  (microseconds) |
| 16000 | 364 | 98 |
| 32000 | 352 | 86 |
| 64000 | 559613 | 3330 |
| 128000 | 771720 | 10146 |
| 256000 | 772034 | 18638 |
| 512000 | 774911 | 30748 |
| 1024000 | 771103 | 69050 |
| 2048000 | 779834 | 109089 |
| 4096000 | 791847 | 179392 |

Graphs:

Ans to the Questions:

1. What is the impact on runtime if we keep |V| unchanged and double |E| for

adjacency list? Why is it so?

* If we keep the |v| unchanged and double |E|, then the runtime of BFS doubles.

This is because, when we double the number of edges, we have to search for twice the number of edges. Now, 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.

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

- If we keep |E| unchanged and double |V|, then the run time of BFS doubles.

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 vertices, the increases. Because, if 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.

3. What is the impact on runtime if we keep |V| unchanged and double |E| for

adjacency matrix? Why is it so?

* In adjacency Matrix, if we keep |V| unchanged and double |E|, the Run time for BFS increases slightly. But when |V| is low it has a significant change.

This is because, when the number of vertices increase, it really does not bring any significant change in adjacent matrix implementation in light of runtime. Because, searching for edge between two vertices in Adjacent Matrix implementation is O(1), and can be done in constant time. So, even if we increase the number of edges, it does not really matter. 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 significant. But, as the number of edges becomes significantly high, the time required becomes almost the same, even if we double the number of edges

4. What is the impact on runtime if we keep |E| unchanged and double |V| for

adjacency matrix? Why is it so?

* In adjacency Matrix representation, if we keep |E| unchanged and double |V|, the run time for BFS will be nearly 4 times.

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 is 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.

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?

-For the same number of edges and vertices, the runtime for adjacency list

representation is higher than the runtime for adjacency matrix.

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 checking specific cells of the matrix we created and stored this information in. 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.