Hassan Harajly

Cis 350

3/24/18

**Germ Lab**

|  |  |  |
| --- | --- | --- |
| **Function** | **Time/comment** | **Space/comment** |
| deleteTopMinNode() | O(log(n))  Calls the dropDown() function | O(1) |
| Returnminnode() | O(1) returns an integer | O(1) returns an integer |
| Priorityqueue() | O(nlog(n)) calls formtree constructor | O(1) |
| Switchitup() | O (log(n)) recursive function that calls itself | O(logn) recursive function calls itself logn times |
| DropDown() | O (log(n)) recursive function that calls | O(logn) recursive function calls itself logn times |
| FormTree() | O(nLog(n)) calls dropdown function n times | O(log(n))  Recursive function calls itself logntimes |
| Push() | O(log(n)  calls the switchitup function which takes log of n time | O(1) |
| Formmatrix() | O(n^2) initializes 2dmatrix array | O(n^2) forms a 2d array denoting a matrix representing a graph structure |
| Update() | O(n^2) updates the matrix according to numbers of germs deleted. | O(n^2) |
| Addnewvertex() | O(n) sets matrix according to the size of germ children | O(1) |
| Topologicalsort() | O(n^2) nested loop that loops according to number of input and calls tied function | O(n^2) tied function uses a 2d vector to store germ numbers |
| Tied() | O(N^2) nested loop calling priority queue deletetopminnode function | O(n^2) 2 d vector is used to store germ numbers |
| Main() | O(n^2) calls the topological sort function | O(n^2) |

|  |  |  |  |
| --- | --- | --- | --- |
| Reason | Input Data | Expected output | Actualoutput |
| Only 1 germ input | 1 | 1 | 1 |
| Sample data | 5  0  4 5 1 0  1 0  5 3 0  3 0 | 2 4 5 3 1 | Asked for input no error message |
| Check for all 0 ties | 5  0  0  0  0  0 | 12345 | 12345 |
| Check for ties | 5  3 5 0  5 4 0  0  0  0 | 1 2 3 4 5 | 1 2 3 4 5 |
| Check for ties while accounting for children | 6  3 5 0  5 4 0  0  0  0  0 | 1 2 6 3 4 5 | 1 2 6 3 4 5 |
| Check for further ties | 6  5 3 0  1 5 4 0  0  0  0  0 | 2 6 1 4 3 5 | 2 6 1 4 3 5 |
| Check for 2 equal ties | 4  2 0  0  4 0  0 | 1 3 2 4 | 1 3 2 4 |
| Check to see if root node doesn’t affect indegree order | 5  3 0  1 0  5 0  2 0  0 | 4 2 1 3 5 | 4 2 1 3 5 |

Defense of data structure:

For the priority queue I used a vector instead of an array since vectors are dynamically allocated, so the space taken up by the vector can be exactly the size of the input but if I used a standard array I would have to allocate more memory to the array, a vector is slower(**because of dynamic allocation memory is allocated randomly forcing your computer to jump from location to another far location in memory as opposed to a static array which just moves to the next memory location in memory, so static arrays are faster**) but more memory efficient than an array.

For the graph I used a dynamically allocated array instead of linked list of pointers to form a graph because for a dynamic array access is faster than say linked list forming a graph structure because I would have to traverse the whole list to access one element but with a dynamic array I get the best of both worlds I can go directly to the memory spot I need and I can set the size at compile time.

And since my program uses dynamically allocated memory there is no (reasonable) limit of input size.