David Tigreros

7/29/2016

CS-260 Section 003

Dr. Nowak

Programming Assignment 1

Timing Results

The timing results of each test are displayed in the tables below. As shown in the tables, when the number of elements used in each data structure is increased, there is an increase in the time it takes to complete each operation.

List Forward Iterated Insertion					
Number of Elements	500	1000	1500	2000	2500
Python (milliseconds)	0.6020	0.5231	1.0059	1.5888	2.4240
Array (milliseconds)	2.8620	6.6729	13.7479	24.9481	37.8430
Pointer (milliseconds)	1.4200	2.8200	4.2400	5.5490	6.9370

List Backward Iterated Insertion					
Number of Elements	500	1000	1500	2000	2500
Python (milliseconds)	0.0679	0.1290	0.1922	0.2592	0.3221
Array (milliseconds)	1.5030	5.0879	10.1819	17.9908	28.0380
Pointer (milliseconds)	2.0049	4.0181	6.3992	7.9839	9.9621

List Traversal					
Number of Elements	500	1000	1500	2000	2500
Python (milliseconds)	0.0350	0.0741	0.0939	0.1268	0.1528
Array (milliseconds)	0.2370	0.4690	0.6950	0.9148	1.1649
Pointer (milliseconds)	156.4760	621.7692	1394.7010	2469.4479	3878.2740

List Forward Iterated Deletion					
Number of Elements	500	1000	1500	2000	2500
Python (milliseconds)	0.5131	1.9979	4.5059	0.1680	12.8870
Array (milliseconds)	2.1708	8.9209	19.0032	34.9731	54.8139
Pointer (milliseconds)	139.9810	567.9891	1275.1410	2284.8160	3620.4059

List Backward Iterated Deletion					
Number of Elements	500	1000	1500	2000	2500
Python (milliseconds)	0.0749	0.1361	0.1991	0.2730	0.3531
Array (milliseconds)	2.0490	6.9621	15.1300	27.0760	43.2091
Pointer (milliseconds)	215.6901	862.6850	1949.3451	3528.4832	5496.1209

Stack Iterated Insertion (PUSH)					
Number of Elements	500	1000	1500	2000	2500
Python (milliseconds)	0.2220	0.3750	0.5491	0.7360	0.9949
Array (milliseconds)	17.0751	66.1242	147.7370	261.4639	410.0971
Pointer (milliseconds)	0.6549	1.1640	1.8139	2.5249	3.1531

Stack Iterated Deletion (POP)					
Number of Elements	500	1000	1500	2000	2500
Python (milliseconds)	0.5779	1.2949	2.2290	3.2101	4.5531
Array (milliseconds)	0.4711	1.1649	2.0039	3.0122	4.2961
Pointer (milliseconds)	0.4239	0.8049	1.1849	1.6000	2.0621

List:

Analysis of the forward iterated insertion for the list shows that it is a relatively efficient procedure regardless of implementation type, however the array implementation takes slightly longer to complete than the other implementations. The built in python function is the fastest, followed by the pointer implementation. For the backward iterated insertions, the same conclusion can be drawn, the fastest implementation is the built in python function, the second is the pointer implementation, and the least efficient is the array implementation.

Analysis of the traversal shows that the built in python function is the most efficient, followed by the array implementation. The pointer implementation is much less efficient than either of the other implementations as it takes a much longer time than the others. This has to deal with the fact that temporary pointers have to be set in order to traverse through the list.

The forward iterated deletion of the lists shows that the built in python function is the most efficient, followed by the array implementation, and lastly the pointer implementation. The pointer implementation takes longer due to setting temporary pointers. The same conclusion is drawn for iterated deletion in reverse order (backward). The most efficient is the python built in function, followed by the array implementation, and then the pointer implementation.

Stack:

Analysis of the iterated insertion, or PUSH, operation of the stack data structure shows that least efficient implementation is the array implementation. The most efficient is the built in python function, followed by the pointer implementation. The pointer implementation is more efficient than the array implementation since the stack deals with just the first end (head) of the structure. Because of this it is easier to just change the location of the head pointer, compared to the shifting that occurs in the array implementation.

The iterated deletion, or POP, operation of the stack data structure shows that the most efficient implementation is the pointer implementation, while the python and array implementation are almost the same efficiency, where the python function is only slightly more efficient than the array implementation.