Report for Assignment 3

Ben Whitehead

October 25, 2013

1 TwoStackQueue

The first method I will consider is the isEmpty() method. isEmpty() simply checks to see if the in and out are empty stacks using the isEmpty() method from the book. enqueue() pushes an item to the in stack, this will take the same amount of stackoperations reguardless of how many items are in the stack. dequeue() checks to see if the out stack isEmpty(), if so then it copies the in stack to the out one, otherwise it simply pops the out stack.

2 Doubling Ratio

My doubling ratio test is based on the one in the book. In each trial the following happens: I push k items to each of the stacks and record how long that takes. I then add the ratio of the runtimes in avgRatio[], then I double k. My ratio is defined as $arrayTime/listTime$ I do this until NUMDOUBLES is reached, NUMDOUBLES describes how many times I will double k during the experiment. I run this for NUMTESTS. NUMDOUBLES is 5 by default, and NUMTESTS is 100, k starts at 60,000. I found that if k is much lower than this value the run times cannot be divided, because they result in a number that cannot be stored as a double (either NaN or infinity). I am getting times that suggest that the array is about two times as fast as the list. This is because the list has more memory allocation going on, each time it pushes, it has to create a new node. The array only has to resize periodically, and everytime it does, it doubles. Here is the table:

3 Birthday Problem

I tested the birthday problem by creating a boolean array called validate. This array is initialized to the size of NUMTESTS, which is by default 1000. The program will then read in a number from the command line, and send that number to the testHyp() method. This method starts by finding the value of

Test #	Ratio of Runtimes (arrayTime/listTime)
0	0.5480859010270775
1	2.562254901960784
2	0.4085989010989011
3	0.32216775599128544
4	0.43729480164158685
5	0.48874125874125873
6	0.4177317290552585
7	0.384099718111346
8	0.4665441176470588
9	0.2831541218637993
10	0.4454248366013071
11	0.35375816993464054
12	0.49800950683303624
13	0.38709150326797387
14	0.38905228758169935
15	0.44523172905525843
16	0.4903409090909091
17	0.4131565656565657
18	0.3813153917693559
19	0.46912878787878787
20	0.45773172905525844
21	0.38322332090840067
22	0.45514705882352935
23	0.3795454545454545
24	0.4165441176470588
25	0.6260984848484848
26	0.4636140819964349
27	0.32351587301587303
28	0.46912878787878787
29	0.45773172905525844
30	0.37201744334097275
31	0.44523172905525843
32	0.4583036244800951
33	0.44523172905525843
34	0.4260398098633392
35	0.36148989898989903
36	0.41842948717948725
37	0.42300950683303623
38	0.32372848200312987
39	0.4665441176470588
40	0.4721590909090909
41	0.393659281894576
42	0.39285714285714285
43	0.6583333333333333
44	0.4714285714285714
45	0.3198214285714286
46	0.39285714285714285
47	0.7047619047619047
48	0.6082010582010582
49	0.32564205457463885
50	0.5397783251231527

Test #	Ratio of Runtimes (arrayTime/listTime)
51	0.34389038634321656
52	0.39285714285714285
53	0.6082010582010582
54	0.375659281595957
55	0.47857142857142854
56	0.33173501755975987
57	0.5428571428571429
58	0.36868770764119596
59	0.3940476190476191
60	0.42208994708994707
61	0.48201058201058206
62	0.3354761904761905
63	0.3940476190476191
64	0.34674603174603175
65	0.6015599343185549
66	0.4891534391534392
67	0.3435405141555483
68	0.39285714285714285
69	0.6653439153439152
70	0.42857142857142855
71	0.47532044116551164
72	0.5967261904761905
73	0.516931216931217
74	0.6082010582010582
75	0.44198606271776997
76	0.3998677248677248
77	0.3150585628363406
78	0.4714285714285714
79	0.5498677248677248
80	0.39357142857142857
81	0.3964285714285714
82	0.3941194581280788
83	0.47857142857142854
84	0.572962962962963
85	0.42142857142857143
86	0.3490427098674521
87	0.40036945812807884
88	0.43805418719211825
89	0.3869047619047619
90	0.3167027417027417
91	0.43965517241379304
92	0.598111658456486
93	0.5078571428571429
94	0.4428571428571429
95	0.31785714285714284
96	0.4391534391534392
97	0.40572407045009784
98	0.5428571428571429
99	0.5669047619047619
The average ratio of the running times	0.4636541373217238

$\sqrt{N/2}$ where N is the value inputted. I then create an array of length $\sqrt{N/2}$, and I use StdRandom.uniform(N) to generate values for that array from 1 to N. I then take that array and test to see if it has any duplicate values, if it does then I return True, else False. This boolean is then stored into my validate array which I then analyse() the analysis simply prints the number of true and false values stored in validate.

4 Path Compression

To produce a path of length 4. You must submit 5 sites such that you can trace from one to the other. For Example (2,3),(3,7),(7,4),(5,8),(8,10). These inputs would produce a path of length 4.