HASH TABLES

1) Using int index = (int) cos(value); is terrible
because it will only return {-1,0,1} which wouldn't
be good for a hashTable. Plus it's returning a negative index.

2) For Days of the week, we took the decimal value of the first
letter and added it with the length of the word.

Day	Index	Bucket
Sunday	115	5
Monday	123	3
Tuesday	128	8
Wednesday	124	4
Thursday	108	8
Friday	123	3
Saturday	121	1

For Months of the year, We took the hashstring2 function from
Homework 6.

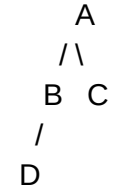
Month	Index	Bucket
January	3114	9
February	3987	12
March	1561	1
April	1623	3
May	666	6
June	1074	9
July	1148	8
August	1680	0
September	4796	11
October	2995	10
November	3842	2
December	3755	5

3) Assuming we are matching hash values, you would have to start with bucket 0 and go to bucket N,
looking and comparing all the hashLinks in each bucket. This best case could be O(1),
worst is O(count).

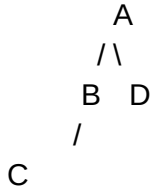
4) 0 1 0 1 0 0 0 0 — 1: {2,3}
0 0 1 0 0 0 0 0 — 2: {3}
0 0 0 0 1 1 0 0 — 3: {5,6}
0 0 0 0 1 0 0 0 — 4: {5}
0 0 0 0 0 0 0 0 — 5: {}
0 0 0 0 0 0 1 1 — 6: {7,8}
0 0 0 0 1 0 0 0 — 7: {5}
0 0 0 0 0 0 0 0 — 8: {}

5) We start at one and look to the left child first then right child. Solution is D.

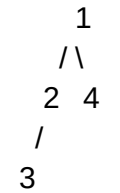
Depth(Faster)

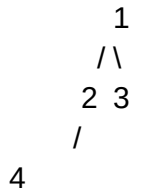


Breadth(Faster)



Numbers are the order in which each letter is found





6)
Depth

Iteration	Stack(T-B)	Reachable
0	1	
1	6,2	1
2	11,2	1,6
3	16,12,2	1,6,11
4	21,12,2	1,6,11,16
5	22,12,2	1,6,11,16,21
6	23,17,12,2	1,6,11,16,21,22
7	17,12,2	1,6,11,16,21,22,23
8	12,12,2	1,6,11,16,21,22,23,17
9	13,12,2	1,6,11,16,21,22,23,17,12
10	18,8,12,2	1,6,11,16,21,22,23,17,12,13
11	8,12,2	1,6,11,16,21,22,23,17,12,13,18

12	3,12,2	1,6,11,16,21,22,23,17,12,13,18,8
13	4,2,12,2	1,6,11,16,21,22,23,17,12,13,18,8,3
14	9,5,2,12,2	1,6,11,16,21,22,23,17,12,13,18,8,3,4
15	14,5,2,12,2	1,6,11,16,21,22,23,17,12,13,18,8,3,4,9
16	15,5,2,12,2	1,6,11,16,21,22,23,17,12,13,18,8,3,4,9,14
17	20,10,5,2,12,2	1,6,11,16,21,22,23,17,12,13,18,8,3,4,9,14,15
18	19,10,5,2,12,2	1,6,11,16,21,22,23,17,12,13,18,8,3,4,9,14,15,20
19	24,10,5,2,12,2	1,6,11,16,21,22,23,17,12,13,18,8,3,4,9,14,15,20,19
20	25,10,5,2,12,2	1,6,11,16,21,22,23,17,12,13,18,8,3,4,9,14,15,20,19,24
21	10,5,2,12,2	1,6,11,16,21,22,23,17,12,13,18,8,3,4,9,14,15,20,19,24,25
22	5,5,2,12,2	1,6,11,16,21,22,23,17,12,13,18,8,3,4,9,14,15,20,19,24,25,10
23	5,2,12,2	1,6,11,16,21,22,23,17,12,13,18,8,3,4,9,14,15,20,19,24,25,10,5
24	2,12,2	1,6,11,16,21,22,23,17,12,13,18,8,3,4,9,14,15,20,19,24,25,10,5
25	7,12,2	1,6,11,16,21,22,23,17,12,13,18,8,3,4,9,14,15,20,19,24,25,10,5,2
26	12,2	1,6,11,16,21,22,23,17,12,13,18,8,3,4,9,14,15,20,19,24,25,10,5,2,7
27	2	1,6,11,16,21,22,23,17,12,13,18,8,3,4,9,14,15,20,19,24,25,10,5,2,7
28		1,6,11,16,21,22,23,17,12,13,18,8,3,4,9,14,15,20,19,24,25,10,5,2,7

Breadth

Iteration	Stack(F-B)	Reachable
0	1	
1	2,6	1,
2	6,3,7	1,2
3	3,7,11	1,2,6
4	7,11,4,8	1,2,6,3
5	11,4,8	1,2,6,3,7
6	4,8,12,16	1,2,6,3,7,11
7	8,12,16,5,9	1,2,6,3,7,11,4
8	12,16,5,9,13	1,2,6,3,7,11,4,8
9	16,5,9,13,13,17	1,2,6,3,7,11,4,8,12
10	5,9,13,13,17,21	1,2,6,3,7,11,4,8,12,16
11	9,13,13,17,21,10	1,2,6,3,7,11,4,8,12,16,5
12	13,13,17,21,10,14	1,2,6,3,7,11,4,8,12,16,5,9
13	13,17,21,10,14,18	1,2,6,3,7,11,4,8,12,16,5,9,13
14	17,21,10,14,18	1,2,6,3,7,11,4,8,12,16,5,9,13
15	21,10,14,18,22	1,2,6,3,7,11,4,8,12,16,5,9,13,17
16	10,14,18,22,22	1,2,6,3,7,11,4,8,12,16,5,9,13,17,21
17	14,18,22,22,15	1,2,6,3,7,11,4,8,12,16,5,9,13,17,21,10
18	18,22,22,15,15	1,2,6,3,7,11,4,8,12,16,5,9,13,17,21,10,14
19	22,22,15,15	1,2,6,3,7,11,4,8,12,16,5,9,13,17,21,10,14,18
20	22,15,15,23	1,2,6,3,7,11,4,8,12,16,5,9,13,17,21,10,14,18,22

21	15,15,23,23	1,2,6,3,7,11,4,8,12,16,5,9,13,17,21,10,14,18,22
22	15,23,23,20	1,2,6,3,7,11,4,8,12,16,5,9,13,17,21,10,14,18,22,15
23	23,23,20,20	1,2,6,3,7,11,4,8,12,16,5,9,13,17,21,10,14,18,22,15
24	23,20,20	1,2,6,3,7,11,4,8,12,16,5,9,13,17,21,10,14,18,22,15,23
25	20,20	1,2,6,3,7,11,4,8,12,16,5,9,13,17,21,10,14,18,22,15,23
26	20,19	1,2,6,3,7,11,4,8,12,16,5,9,13,17,21,10,14,18,22,15,23,20
27	19,19	1,2,6,3,7,11,4,8,12,16,5,9,13,17,21,10,14,18,22,15,23,20
28	19,24	1,2,6,3,7,11,4,8,12,16,5,9,13,17,21,10,14,18,22,15,23,20,19
29	24,24	1,2,6,3,7,11,4,8,12,16,5,9,13,17,21,10,14,18,22,15,23,20,19
30	24,25	1,2,6,3,7,11,4,8,12,16,5,9,13,17,21,10,14,18,22,15,23,20,19,24
31	25,25	1,2,6,3,7,11,4,8,12,16,5,9,13,17,21,10,14,18,22,15,23,20,19,24
32	25,	1,2,6,3,7,11,4,8,12,16,5,9,13,17,21,10,14,18,22,15,23,20,19,24,25
33		1,2,6,3,7,11,4,8,12,16,5,9,13,17,21,10,14,18,22,15,23,20,19,24,25

7) Dijkstras Algorithm

Iteration	Stack	Reachable
1	Pensacola(0)	
2	Phoenix(5)	Pensacola(0)
3	Pueblo(8),Peoria(9),Pitts(15)	Phoenix(5)
4	Pierre(11),Peoria(9),Pitts(15)	Pueblo(8)
5	Pendel(13),Peoria(9),Pitts(15)	Pierre(11)
6	Peoria(9),Pitts(15)	Pendelton(13)
7	Pitts(14),Pitts(15)	Peoria(9)
8	Prince(16),Pitts(15)	Pittsburgh(14)
9	Pitts(15)	Prince(16)
10		

Final Reachable
{Pensacola(0), Phoenix(5), Pueblo(8), Pierre(11), Pendelton(13), Peoria(9), Pittsburgh(14), Prince(16)}

8) Since Dijkstra's algorithm requires a weighted graph, then you need a priority queue to find the shortest(or lighest) path to each node.

9) For Edge List, you get O(V+E)

10) For Adjacency Matrix, you get O(V^2)

11) Depth cannot guarantee because it could go down the wrong infinite path so it could never go on the next one. Breadth can guarantee because it will look at all possible paths as it travels through the maze.