Jaydeep Rotithor

Assignment 7

Hash tables.

1. Because all of the keys will hash to different values. The function would be modded by 7 instead of 6.
2. a. Abel- 4.
   Abigail- 8
   Abraham- 17
   Ada- 0
   Adam- 0
   Adrian- 17
   Adrienne- 17
   Agnes- 13
   Albert- 1
   Alex- 4
   Alfred- 5
   Alice- 8

   b. Size 6:
   bucket 0: 2
   bucket 1: 2
   bucket 2: 2
   bucket 3:0
   bucket 4: 2
   bucket 5: 4
   size 13:
   bucket 0: 3
   bucket 1: 1
   bucket 2: 0
   bucket 3:0
   bucket 4: 5
   bucket 5: 1
   bucket 6: 0
   bucket 7: 0
   bucket 8: 2
   bucket 9: 0
   bucket 10:0
   bucket 11: 0
   bucket 12: 0

   c. size 6: 2
   size 13: .9231
3. Since cosine always returns a value between 0 and 1, casting it to an integer will return either 0 or 1, more often 0.

4. Days of the week: take the second letter of each day, convert it to a number between 0 and 25, add a number between 0 and 6, depending on what day of the week it is(Sunday = 0, Monday = 1, …, Saturday = 6) and then mod it by 10.
5. First, use the hash function on the key to go to the correct index in the hashtable. Then, assign a hashlink called temp to equal the first link at that index. Then, compare the value of temp to the value of the key that needs to be found. Go through the entire list, and if the value is not found, then return 0. If the value is found, return 1. The complexity here is O(n), because in the worst case every value in the list will need to be accessed.

Graphs

1. Matrix:

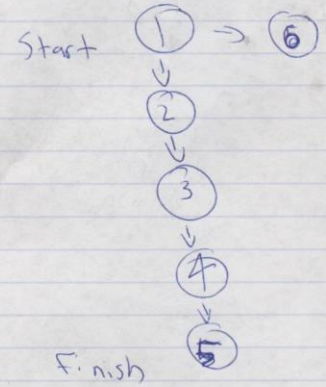
	1	2	3	4	5	6	7	8
1	-	1	0	1	0	0	0	0
2	0	-	1	0	0	0	0	0
3	0	0	-	0	1	0	0	0
4	0	0	0	-	1	0	0	0
5	0	0	0	0	-	0	0	0
6	0	0	0	0	0	-	1	1
7	0	0	0	0	1	0	-	0
8	0	0	0	0	0	0	0	-

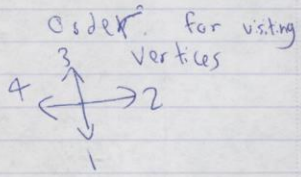
Edge list:

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

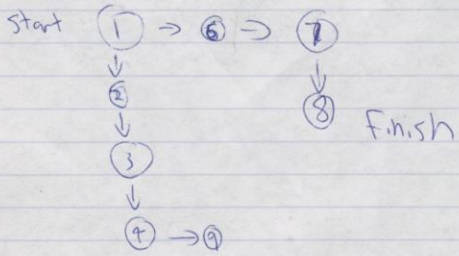
2. Below, in the depth-first graph, the graph simply goes downward along the line in depth-first, but in breadth-first, it accesses the node to the right of the starting node also, so it takes one step longer. In the breadth-first graph, the depth-first solution goes downward to the bottom, hits a dead end, and then comes back up and goes right toward the end. The breadth-first solution accesses both the bottom and the right path equally and will find the solution faster.

2. Depth-first





Breadth-first



3.

Depth-first:

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

Breadth-first:

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

4.

	Pensacola: 0
Pensacola: 0	Phoenix: 5
Phoenix: 5	Pueblo: 8, Peoria: 9, Pittsburgh: 15
Pueblo: 8	Peoria: 9, Pierre: 11, Pittsburgh: 15
Peoria: 9	Pierre: 11, Pueblo: 12, Pittsburgh: 14, Pittsburgh: 15
Pierre: 11	Pueblo: 12, Pendleton: 13, Pittsburgh: 14, Pittsburgh: 15
Pendleton: 13	Pittsburgh: 14: Pittsburgh: 15, Phoenix: 17, Pueblo: 21
Pittsburgh: 14	Pittsburgh: 15, Phoenix: 17, Pueblo: 21

5. Because then the smallest result needs to be removed from the queue, and it has the highest priority, which is the smallest number. It is compared to the values already removed, and the city has already been removed, the next priority item will be removed. If a stack were used, then the first path found would be the one added to the shortest list, and this may not be correct. With a queue, the last path found would be added, and this is also not necessarily the right path.
6. O(V+E)
7. $O(V^2)$
8. A depth-first search is not guaranteed to find the path, but a breadth-first search is. This is because once a breadth-first search selects a path, it goes through the path to the entirety of its depth. If the path is infinitely deep, the search will never reach the end if the path is incorrect. A breadth-first search is guaranteed to find the path. This search does not go very deep into each path, and advances along all potential paths at the same rate. Since it is given that the length of the path from start to finish is finite, one of the search paths will eventually reach the end.