DATA STRUCTURES - CS 261 (SPRING 2015)

Assignment #7 – Solutions

Keeley Abbott
Theodore Duchow-Pressley

June 2, 2015

1 In searching for a good hash function over the set of integer values, one student thought he could use the following:

```
int index = (int) cos(value); // Cosine of 'value'
```

What was wrong with this choice?

The possible integers derived from `(int) math.cos(value)` are $\{-1, 0, 1\}$, which would result in a high rate of collision.

2 Can you come up with a perfect hash function for the names of days of the week? The names of the months of the year? Assume a table size of 10 for days of the week and 15 for names of the months. In case you cannot find any perfect hash functions, we will accept solutions that produce a small number of collisions (< 3).

There are many solutions for this problem, but for simplicity we began by using the sum of the first three letters of each day of the week (since each of these is unique to begin with):

Day of the week	First three	Values of letters	Sum of letters
Monday	MON	$12+14+13$	39
Tuesday	TUE	$19+20+4$	43
Wednesday	WED	$22+4+3$	29
Thursday	THU	$19+7+20$	46
Friday	FRI	$5+17+8$	30
Saturday	SAT	$18+0+19$	37
Sunday	SUN	$18+20+13$	51

Table 1: *Perfect hash for weekdays – question 2*

This gives us a set of unique integers that look like this:

$$\forall x \in A = \{39, 43, 29, 46, 30, 37, 51\}$$

3 The function `containsKey()` can be used to see if a dictionary contains a given key. How could you determine if a dictionary contains a given value? What is the complexity of your procedure?

4 Represent the following graph as both an adjacency matrix and an edge list:

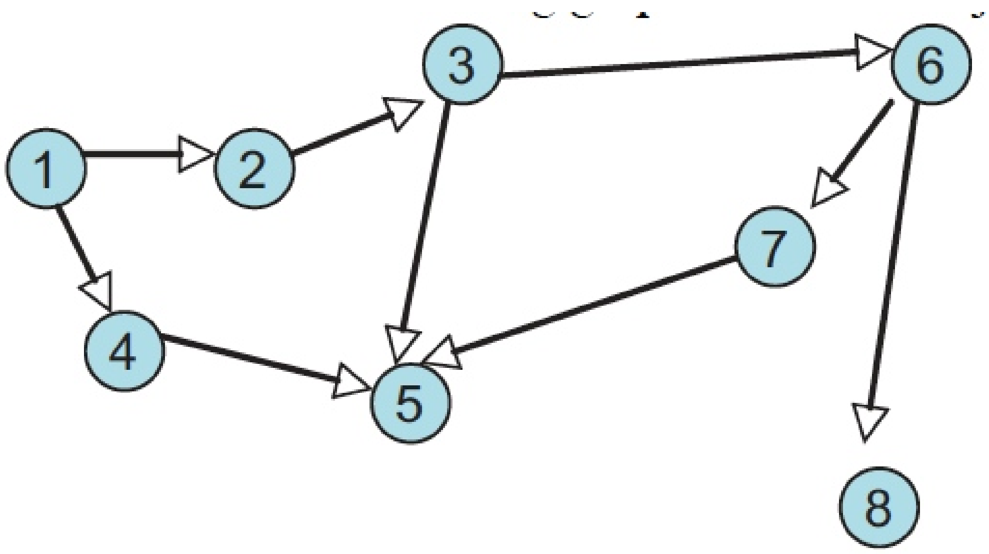


Figure 1: *Graph – question 4*

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

(a) *Adjacency matrix*

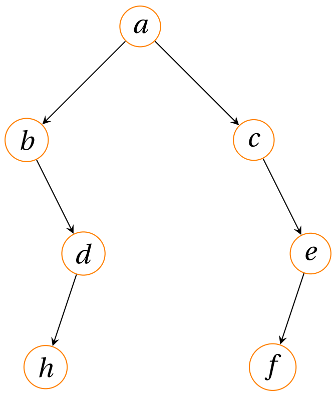
Node	Edges
1:	$\{2,4\}$
2:	$\{3\}$
3:	$\{5,6\}$
4:	$\{5\}$
5:	$\{\varnothing\}$
6:	$\{7,8\}$
7:	$\{5\}$
8:	$\{\varnothing\}$

(b) *Edge list*

Table 2: *Adjacency matrix & edge list – question 4*

5 Construct a graph in which a depth first search will uncover a solution (discover reachability from one vertex to another) in fewer steps than will a breadth first search. You may need to specify an order in which neighbor vertices are visited. Construct another graph in which a breadth-first search will uncover a solution in fewer steps.

Figure 2: *Example graph – question 5*



5.1 Depth-first search case

Using the above graph with a starting path at 'a' and ending at 'h', and assuming a clockwise push order for the stack. For a depth-first search, we will uncover the solution by visiting $\{a, b, d, h\}$. While for a breadth-first search we will uncover the solution after visiting $\{a, b, c, d, e, h, f\}$. Thus depth-first search is clearly completed in fewer steps in this case.

5.2 Breadth-first search case

Again using the above graph with a starting path at 'a' and ending at 'c', and again assuming a clockwise push order for the stack. With a depth-first search, we will uncover the solution by visiting $\{a, b, d, h, c\}$. Whereas with a breadth-first search we will uncover the solution after visiting $\{a, b, c\}$. Thus breadth-first search is clearly completed in fewer steps in this case.

6 Complete Worksheet 41 (2 simulations). Show the content of the stack, queue, and the set of reachable nodes.

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

(a) *Worksheet 41 – DFS (Stack)*

Iteration	Queue (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	9,13,13,17,21,10,14	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

(b) *Worksheet 41 – BFS (Queue)*

Table 3: *Worksheet 41 – question 6*

7 Complete Worksheet 42 (1 simulation). Show the content of the priority queue and the cities visited at each step.

Iteration	Priority queue	Added to reachable
0	Pensacola(0)	
1	Phoenix(5)	Pensacola(0)
2	Pueblo(8), Peoria(9), Pittsburgh(15)	Phoenix(5)
3	Peoria(9), Pierre(11), Pittsburgh(15)	Pueblo(8)
4	Pierre(11), Pueblo(12), Pittsburgh(14), Pittsburgh(15)	Peoria(9)
5	Pueblo(12), Pendleton(13), Pittsburgh(14), Pittsburgh(15)	Pierre(11)
6	Pendleton(13), Pittsburgh(14), Pittsburgh(15)	
7	Pittsburgh(14), Pittsburgh(15), Phoenix(19), Pueblo(21)	Pendleton(13)
8	Pittsburgh(15), Pensacola(18), Phoenix(19), Pueblo(21)	Pittsburgh(14)
9	Pensacola(18), Phoenix(19), Pensacola(19), Pueblo(21)	
10	Phoenix(19), Pensacola(19), Pueblo(21)	
11	Pensacola(19), Pueblo(21)	
12	Pueblo(21)	
13		

Table 4: *Worksheet 42 – question 7*

8 Why is it important that Dijkstra's algorithm stores intermediate results in a priority queue, rather than in an ordinary stack or queue?

It is important, because the intent is to take the path with the lowest associated cost – prioritizing intermediate edges – which is a property that neither a stack nor a queue posses. of the three, only a priority queue possesses this property (this is also assuming that priorities are determined by cumulative cost from the point of origin).

9 How much space (in big-O notation) does an edge-list representation of a graph require?

$O(|V|+|E|)$

10 For a graph with V vertices, how much space (in big-O notation) will an adjacency matrix require?

$O|V|^2$

11 Suppose you have a graph representing a maze that is infinite in size, but there is a finite path from the start to the finish. Is a depth first search guaranteed to find the path? Is a breadth-first search guaranteed to find the path? Explain why or why not.

For an infinite size maze, a *depth-first search* does not guarantee this path will be found. This algorithm will search along a path until it reaches an end, and since the size of the maze can be infinite, it is possible for the path of a depth-first search to be infinitely long.

However, using a *breadth-first search* does guarantee the path will be found, because this algorithm will search the nearest neighbor points – beginning with the starting point – and gradually enlarge the search region. This ensures that the finite path you are looking for will eventually be found.