```
/* CS261- Assignment 7
 * Name: Li, Tingzhi & Zhang, Chunyang
 * Date: 6/5/2015
 * Development Environment: Microsoft Word
 * Solution description: Hash Table and Graph implementation
 */
```

Hash tables

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?

Answer: When using the hash function above, the index will be zero for all values except when value is 0, which will produce index 1. This is not preferable because this hash function introduced a huge collision problem.

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).

Answer: Hash function for names of days of the week. Summing the alphabetical values of each character in the string. Table size = 10

sunday	84 % 10 = 4
monday	72 % 10 = 2
tuesday	95 % 10 = 5
wednesday	100 % 10 = 0
thursday	116 % 10 = 6
friday	63 % 10 = 3
saturday	109 % 10 = 9

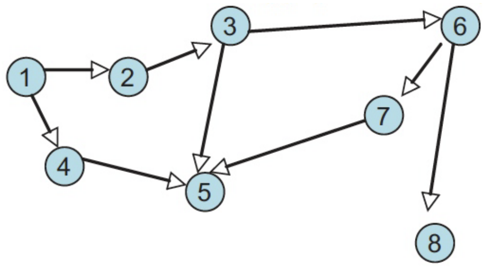
Hash function for names of the months of the year. Summing the alphabetical values of each character in the string. Table size = 15

january	90 % 15 = 0
february	96 % 15 = 6
march	43 % 15 = 13
april	56 % 15 = 11
may	39 % 15 = 9
june	50 % 15 = 5
july	68 % 15 = 8
august	89 % 15 = 14
september	103 % 15 = 13
october	78 % 15 = 3
november	94 % 15 = 4
december	55 % 15 = 10

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?

Answer: We would first put the key into hash function and get an integer, which is the index for the hash table. Then we could compare this key with the key that contained in that bucket of the hash table directed by that calculated index, if they do not match, then we could go through the next link if there is one. We could keep doing this until the next link is NULL. We would return 1 if there is one match, or return 0 if there isn't.
Luckily, if the key is contained in the bucket of hash table, then the complexity of this procedure is O(1), if not, the complexity would be the number of link starting from that index bucket to where we find the key. The worst case is that all the keys in the hash table produced a link list and the wanted key is not contained, which is O(n). Overall, the average complexity is O(1+).

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



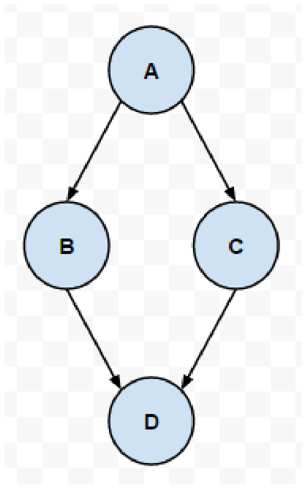
Answer:
Adjacency Matrix:

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

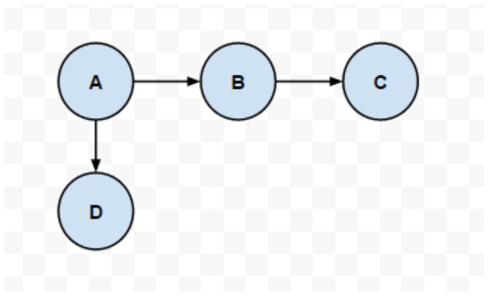
Edge List:
1: {2, 4}
2: {3}
3: {5, 6}
4: {5}
5: { }
6: {7, 8}
7: {5}
8: { }

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.

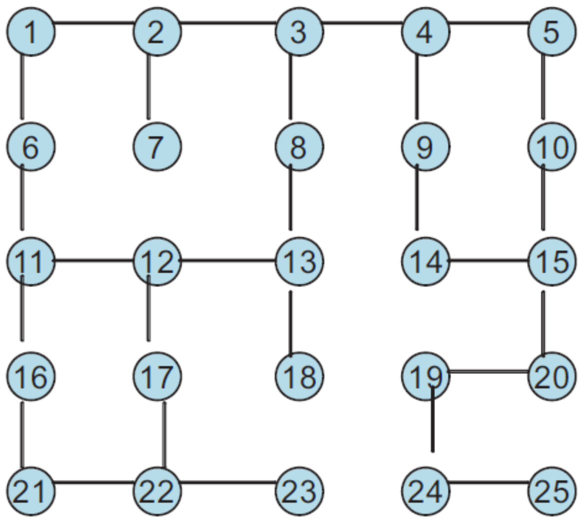
Answer:
a) Fewer steps in depth first search



b) Fewer steps in breadth first search



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



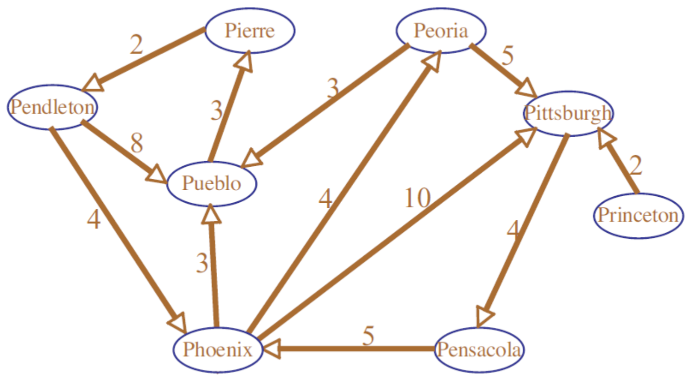
Answer:
Depth first search [First 12 iterations] (stack version)

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

Breadth first search [first 14 iterations] (queue version)

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	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

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



Answer:
Dijkstra’s Algorithm:

Iteration	Pqueue	Reachable with Costs
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, Pittsburgh: 14, Pittsburgh: 15	Peoria: 9
5	Pendleton: 13, Pittsburgh: 14, Pittsburgh: 15	Pierre: 11
6	Pittsburgh: 14, Pittsburgh: 15	Pendleton: 13
7	Pittsburgh: 15	Pittsburgh: 14
8	{}	--

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

Answer:
The reason is that in Dijkstra's algorithm, we want to know what is the lowest cost to reach the vertex rather than whether a give vertex is reachable. To find the lowest cost, we have to us a priority queue to store the cost for each step and find to lowest cost to keep moving.

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

Answer:
An edge-list representation of a graph requires O(v+e) space.

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

Answer:
An adjacency matrix will require $O(V^2)$ space.

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.

Answer:
A breadth-first search is guaranteed to find the path. From the lecture note, we know that the reason is because BFS first checks all paths of length 1, then of length 2, then of length 3, etc. It’s guaranteed to find a path containing the least steps from start to goal. However, if we use DFS, it will check one path to the end before it checks the next path. If the path being checked is to the goal, then we are lucky. But if the path is infinite, then we will never get the chance to check the next path. Therefore, for finding the path from the start to the finish, a BFS is better.