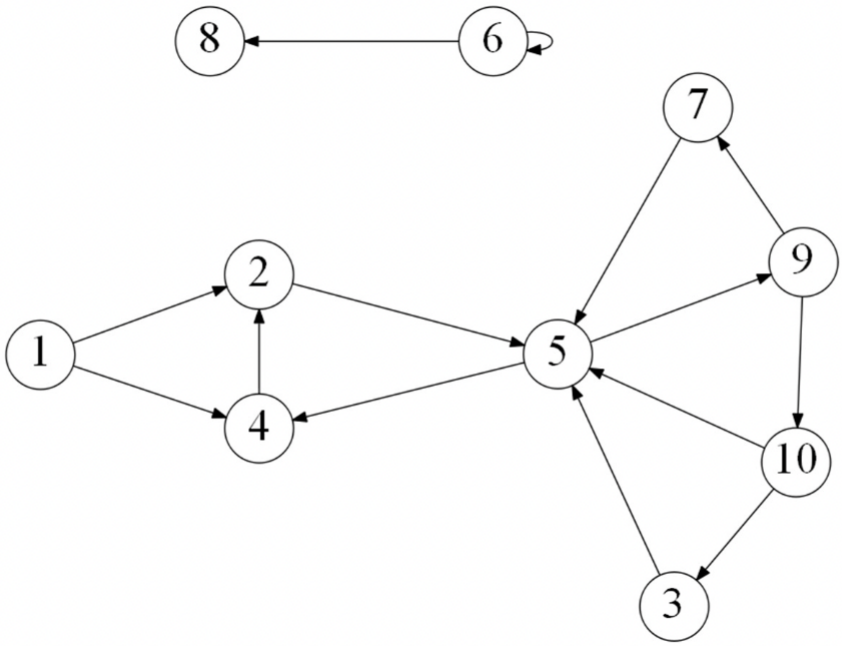
Homework 3: Graph Algorithms

Arjun Koshal

November 11, 2022

Consider the following graph:



Problem 1

Draw how the graph would look if represented by an adjacency matrix. You may assume the indexes are from 1 through 10. (5 points)

Solution:

		To									
		1	2	3	4	5	6	7	8	9	10
From	1	0	1	0	1	0	0	0	0	0	0
	2	0	0	0	0	1	0	0	0	0	0
	3	0	0	0	0	1	0	0	0	0	0
	4	0	1	0	0	0	0	0	0	0	0
	5	0	0	0	1	0	0	0	0	1	0
	6	0	0	0	0	0	1	0	1	0	0
	7	0	0	0	0	1	0	0	0	0	0
	8	0	0	0	0	0	0	0	0	0	0
	9	0	0	0	0	0	0	1	0	0	1
	10	0	0	1	0	1	0	0	0	0	0

Problem 2

Draw how the graph would look if represented by an adjacency list. You may assume the indexes are from 1 through 10. (5 points)

Solution:

1	$\rightarrow 2 \rightarrow 4$
2	$\rightarrow 5$
3	$\rightarrow 5$
4	$\rightarrow 2$
5	$\rightarrow 4 \rightarrow 9$
6	$\rightarrow 6 \rightarrow 8$
7	$\rightarrow 5$
8	
9	$\rightarrow 7 \rightarrow 10$
10	$\rightarrow 3 \rightarrow 5$

Problem 3

List the order in which the vertices are visited with a breadth-first search. If there are multiple vertices adjacent to a given vertex, visit the adjacent vertex with the lowest value first. (5 points)

Solution:

Counter: 0

1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0

Queue: empty

Counter: 1

1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	0	0	0

Queue: 1

Counter: 2

1	2	3	4	5	6	7	8	9	10
1	2	0	0	0	0	0	0	0	0

Queue: 1 2

Counter: 3

1	2	3	4	5	6	7	8	9	10
1	2	0	3	0	0	0	0	0	0

Queue: 1 2 4

Counter: 3

1	2	3	4	5	6	7	8	9	10
1	2	0	3	0	0	0	0	0	0

Queue: 2 4

Counter: 4

1	2	3	4	5	6	7	8	9	10
1	2	0	3	4	0	0	0	0	0

Queue: 2 4 5

Counter: 4

1	2	3	4	5	6	7	8	9	10
1	2	0	3	4	0	0	0	0	0

Queue: 4 5

Counter: 4

1	2	3	4	5	6	7	8	9	10
1	2	0	3	4	0	0	0	0	0

Queue: 5

Counter: 5

1	2	3	4	5	6	7	8	9	10
1	2	0	3	4	0	0	0	5	0

Queue: 5 9

Counter: 5

1	2	3	4	5	6	7	8	9	10
1	2	0	3	4	0	0	0	5	0

Queue: 9

Counter: 6

1	2	3	4	5	6	7	8	9	10
1	2	0	3	4	0	6	0	5	0

Queue: 9 7

Counter: 7

1	2	3	4	5	6	7	8	9	10
1	2	0	3	4	0	6	0	5	7

Queue: 9 7 10

Counter: 7

1	2	3	4	5	6	7	8	9	10
1	2	0	3	4	0	6	0	5	7

Queue: 7 10

Counter: 7

1	2	3	4	5	6	7	8	9	10
1	2	0	3	4	0	6	0	5	7

Queue: 10

Counter: 8

1	2	3	4	5	6	7	8	9	10
1	2	8	3	4	0	6	0	5	7

Queue: 10 3

Counter: 8

1	2	3	4	5	6	7	8	9	10
1	2	8	3	4	0	6	0	5	7

Queue: 3

Counter: 8

1	2	3	4	5	6	7	8	9	10
1	2	8	3	4	0	6	0	5	7

Queue: empty

Counter: 9

1	2	3	4	5	6	7	8	9	10
1	2	8	3	4	9	6	0	5	7

Queue: 6

Counter: 10

1	2	3	4	5	6	7	8	9	10
1	2	8	3	4	9	6	10	5	7

Queue: 6 8

Counter: 10

1	2	3	4	5	6	7	8	9	10
1	2	8	3	4	9	6	10	5	7

Queue: 8

Counter: 10

1	2	3	4	5	6	7	8	9	10
1	2	8	3	4	9	6	10	5	7

Queue: empty

---

1 2 4 5 9 7 10 3 6 8

Problem 4

List the order in which the vertices are visited with a depth-first search. If there are multiple vertices adjacent to a given vertex, visit the adjacent vertex with the lowest value first. (5 points)

Solution:

Counter: 0

1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0

Stack: empty

Counter: 1

1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	0	0	0

Stack: dfs(1)

Counter: 1

1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	0	0	0

Stack: dfs(1), dfs(2)

Counter: 2

1	2	3	4	5	6	7	8	9	10
1	2	0	0	0	0	0	0	0	0

Stack: dfs(1), dfs(2)

Counter: 2

1	2	3	4	5	6	7	8	9	10
1	2	0	0	0	0	0	0	0	0

Stack: dfs(1), dfs(2), dfs(5)

Counter: 3

1	2	3	4	5	6	7	8	9	10
1	2	0	0	3	0	0	0	0	0

Stack: dfs(1), dfs(2), dfs(5)

Counter: 3

1	2	3	4	5	6	7	8	9	10
1	2	0	0	3	0	0	0	0	0

Stack: dfs(1), dfs(2), dfs(5), dfs(4)

Counter: 4

1	2	3	4	5	6	7	8	9	10
1	2	0	4	3	0	0	0	0	0

Stack: dfs(1), dfs(2), dfs(5), dfs(4)

Counter: 4

1	2	3	4	5	6	7	8	9	10
1	2	0	4	3	0	0	0	0	0

Stack: dfs(1), dfs(2), dfs(5)

Counter: 4

1	2	3	4	5	6	7	8	9	10
1	2	0	4	3	0	0	0	0	0

Stack: dfs(1), dfs(2), dfs(5), dfs(9)

Counter: 5

1	2	3	4	5	6	7	8	9	10
1	2	0	4	3	0	0	0	5	0

Stack: dfs(1), dfs(2), dfs(5), dfs(9)

Counter: 5

1	2	3	4	5	6	7	8	9	10
1	2	0	4	3	0	0	0	5	0

Stack: dfs(1), dfs(2), dfs(5), dfs(9), dfs(7)

Counter: 6

1	2	3	4	5	6	7	8	9	10
1	2	0	4	3	0	6	0	5	0

Stack: dfs(1), dfs(2), dfs(5), dfs(9), dfs(7)

Counter: 6

1	2	3	4	5	6	7	8	9	10
1	2	0	4	3	0	6	0	5	0

Stack: dfs(1), dfs(2), dfs(5), dfs(9)

Counter: 6

1	2	3	4	5	6	7	8	9	10
1	2	0	4	3	0	6	0	5	0

Stack: dfs(1), dfs(2), dfs(5), dfs(9), dfs(10)

Counter: 7

1	2	3	4	5	6	7	8	9	10
1	2	0	4	3	0	5	0	4	7

Stack: dfs(1), dfs(2), dfs(5), dfs(9), dfs(10)

Counter: 7

1	2	3	4	5	6	7	8	9	10
1	2	0	4	3	0	6	0	5	7

Stack: dfs(1), dfs(2), dfs(5), dfs(9), dfs(10), dfs(3)

Counter: 8

1	2	3	4	5	6	7	8	9	10
1	2	8	4	3	0	6	0	5	7

Stack: dfs(1), dfs(2), dfs(5), dfs(9), dfs(10), dfs(3)

Counter: 8

1	2	3	4	5	6	7	8	9	10
1	2	8	4	3	0	6	0	5	7

Stack: dfs(1), dfs(2), dfs(5), dfs(9), dfs(10)

Counter: 8

1	2	3	4	5	6	7	8	9	10
1	2	8	4	3	0	6	0	5	7

Stack: dfs(1), dfs(2), dfs(5), dfs(9)

Counter: 8

1	2	3	4	5	6	7	8	9	10
1	2	8	4	3	0	6	0	5	7

Stack: dfs(1), dfs(2), dfs(5)

Counter: 8

1	2	3	4	5	6	7	8	9	10
1	2	8	4	3	0	6	0	5	7

Stack: dfs(1), dfs(2)

Counter: 8

1	2	3	4	5	6	7	8	9	10
1	2	8	4	3	0	6	0	5	7

Stack: dfs(1)

Counter: 8

1	2	3	4	5	6	7	8	9	10
1	2	8	4	3	0	6	0	5	7

Stack: empty

Counter: 8

1	2	3	4	5	6	7	8	9	10
1	2	8	4	3	0	6	0	5	7

Stack: dfs(6)

Counter: 9

1	2	3	4	5	6	7	8	9	10
1	2	8	4	3	9	6	0	5	7

Stack: dfs(6)

Counter: 9

1	2	3	4	5	6	7	8	9	10
1	2	8	4	3	9	6	0	5	7

Stack: dfs(6), dfs(8)

Counter: 10

1	2	3	4	5	6	7	8	9	10
1	2	8	4	3	9	6	10	5	7

Stack: dfs(6), dfs(8)

Counter: 10

1	2	3	4	5	6	7	8	9	10
1	2	8	4	3	9	6	10	5	7

Stack: dfs(6)

Counter: 10

1	2	3	4	5	6	7	8	9	10
1	2	8	4	3	9	6	10	5	7

Stack: empty

---

1 2 5 4 9 7 10 3 6 8

Problem 5

a) What is the running time of breadth-first search with an adjacency matrix? (2 points)
b) What is the running time of breadth-first search with an adjacency list? (2 points)

Solution:

a) Adjacency matrix: $\Theta(V^2)$
In reference to the graph at the top of the page, the running time of breadth-first search is $\Theta(10^2) = \Theta(100)$.

b) Adjacency matrix: $\Theta(V + E)$
In reference to the graph at the top of the page, the running time of breadth-first search is $\Theta(10 + 14) = \Theta(24)$.

Problem 6

a) What is the running time of depth-first search with an adjacency matrix? (2 points)
b) What is the running time of depth-first search with an adjacency list? (2 points)

Solution:

a) Adjacency matrix: $\Theta(V^2)$
In reference to the graph at the top of the page, the running time of breadth-first search is $\Theta(10^2) = \Theta(100)$.

b) Adjacency matrix: $\Theta(V + E)$
In reference to the graph at the top of the page, the running time of breadth-first search is $\Theta(10 + 14) = \Theta(24)$.

Problem 7

While an adjacency matrix is typically easier to code than an adjacency list, it is not always a better solution. Explain when an adjacency list is a clear winner in the efficiency of your algorithm? (2 points)

Solution:

Adjacency lists are more efficient when using larger and sparse graphs. For example, if the graph is not connected and the adjacency matrix is filled primarily with zeroes, then it would be optimal to use an adjacency list over an adjacency matrix.

Problem 8

Explain how one can use a breadth-first to determine if an undirected graph contains a cycle. (5 points)

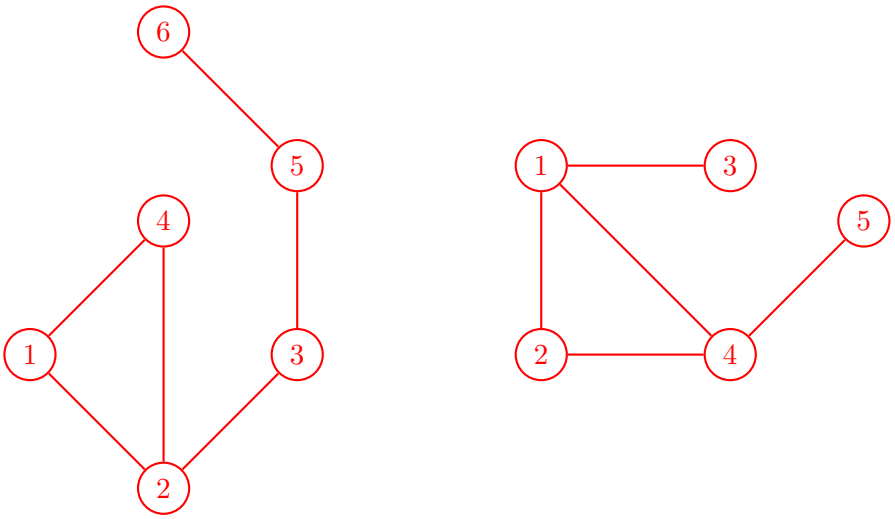
Solution:

We can use breadth-first search to determine if an undirected graph contains a cycle through checking if every visited vertex a, has an adjacent vertex b, such that b has already been visited and b is not a parent of a. If this condition is satisfied, then we can state there is a cycle in the graph. If we traverse the entire graph and find no adjacent vertex b, then the graph does not contain a cycle.

Problem 9

On undirected graphs, does either of the two traversals, DFS or BFS, always find a cycle faster than the other? If yes, indicate which of them is better and explain why it is the case; if not, draw two graphs supporting your answer and explain the graphs. (5 points)

Solution:



On the left graph, BFS will find the cycle at vertices 1, 2 and 4 much faster than DFS would. BFS would go from 1 to 2 and 1 to 4. Then at 2 it would go to 3 and 4. After, it would go from 4 to 2 and notice there is a cycle at vertices 1, 2, and 4. DFS on the other hand would travel from 1 to 2 to 3 to 5 to 6, and then come back to vertex 2. Once we reach 2, we will travel to 4 and then 1 and finally notice that there is a cycle.

On the right graph, DFS will find the cycle at vertices 1, 2 and 4 much faster than BFS. DFS would go from 1 to 2 to 4. Once at 4, it would go to 1 and notice that there is a cycle. BFS on the other hand would travel from 1 to 2 and 1 to 3. From 2 it would go to 4. Then it would check 3 and notice there are no neighbors. At 4 BFS would go to 1 and 5 and notice that there is a cycle at vertices 1, 2 and 4.

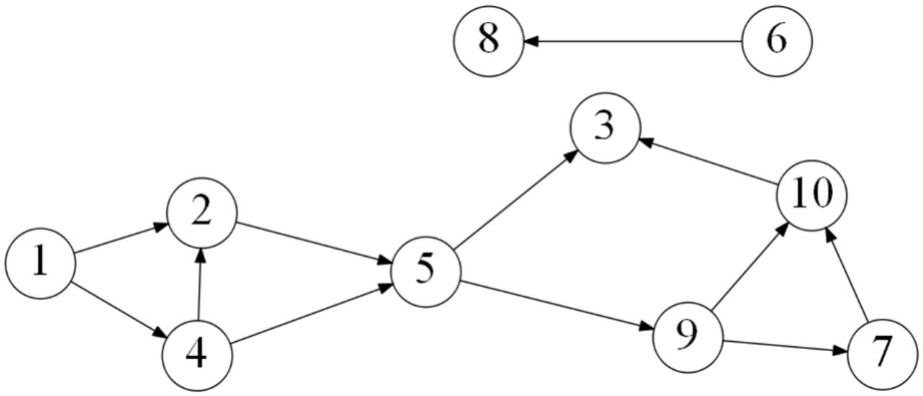
Problem 10

Explain why a topological sort is not possible on the graph at the very top of this document. (5 points)

Solution:

A topological sort is not possible on the graph at the very top of this document because the graph is cyclic (it has a cycle at vertices 5, 9 and 7). A topological order cannot exist when a graph has a cycle because it contradicts the defintion of topological ordering. Topological ordering is a linear ordering of its vertices, such that every directed edge ab from vertex a to vertex b, a must come before b in the ordering. Assuming we have a directed graph with a cycle {(a, b), (b, a)}, this means that topological sort with ordering (a, b) would be contradicted by the edge ba, and topological sort with ordering (b, a) would be contradicted by the edge ab.

Consider the following graph:



Problem 11

List the order in which the vertices are visited with a topological sort. Break ties by visiting the vertex with the lowest value first. (5 points)

Solution:

1	2	3	4	5	6	7	8	9	10
0	2	2	1	2	0	1	1	1	2

Set: 1 6
List: [ ]

1	2	3	4	5	6	7	8	9	10
0	2	2	1	2	0	1	1	1	2

Set: 6
List: [1]

1	2	3	4	5	6	7	8	9	10
0	1	2	0	2	0	1	1	1	2

Set: 6 4
List: [1]

1	2	3	4	5	6	7	8	9	10
0	1	2	0	2	0	1	0	1	2

Set: 4
List: [1 6]

1	2	3	4	5	6	7	8	9	10
0	1	2	0	2	0	1	0	1	2

Set: 4 8
List: [1 6]

1	2	3	4	5	6	7	8	9	10
0	0	2	0	1	0	1	0	1	2

Set: 8
List: [1 6 4]

1	2	3	4	5	6	7	8	9	10
0	0	2	0	1	0	1	0	1	2

Set: 8 2
List: [1 6 4]

1	2	3	4	5	6	7	8	9	10
0	0	2	0	1	0	1	0	1	2

Set: 2
List: [1 6 4 8]

1	2	3	4	5	6	7	8	9	10
0	0	2	0	0	0	1	0	1	2

Set: empty
List: [1 6 4 8 2]

1	2	3	4	5	6	7	8	9	10
0	0	2	0	0	0	1	0	1	2

Set: 5
List: [1 6 4 8 2]

1	2	3	4	5	6	7	8	9	10
0	0	1	0	0	0	1	0	0	2

Set: empty
List: [1 6 4 8 2 5]

1	2	3	4	5	6	7	8	9	10
0	0	1	0	0	0	1	0	0	2

Set: 9
List: [1 6 4 8 2 5]

1	2	3	4	5	6	7	8	9	10
0	0	1	0	0	0	0	0	0	1

Set: empty
List: [1 6 4 8 2 5 9]

1	2	3	4	5	6	7	8	9	10
0	0	1	0	0	0	0	0	0	1

Set: 7
List: [1 6 4 8 2 5 9]

1	2	3	4	5	6	7	8	9	10
0	0	1	0	0	0	0	0	0	0

Set: empty
List: [1 6 4 8 2 5 9 7]

1	2	3	4	5	6	7	8	9	10
0	0	1	0	0	0	0	0	0	0

Set: 10
List: [1 6 4 8 2 5 9 7]

1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0

Set: empty
List: [1 6 4 8 2 5 9 7 10]

1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0

Set: 3
List: [1 6 4 8 2 5 9 7 10]

1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0

Set: empty
List: [1 6 4 8 2 5 9 7 10 3]

---

1 6 4 8 2 5 9 7 10 3