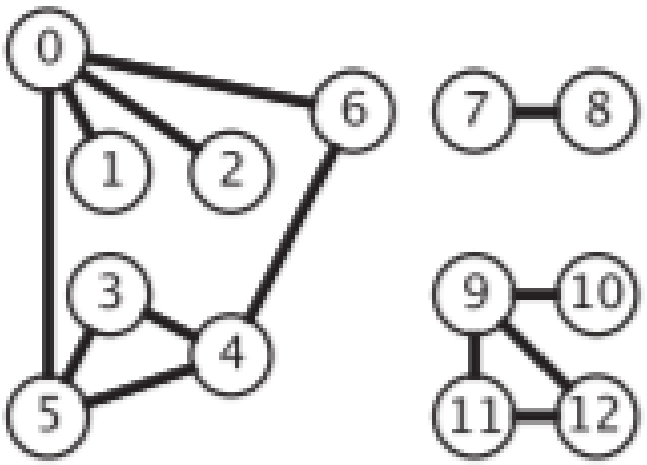
CSC 225 Assignment 4,
Julian Rocha, V00870460
A01, B02

1. *Show a trace of DFS on this graph starting with vertex 0 by showing the contents of the arrays, marked [], EdgeTo [], and id [] during its execution. Show the DFS spanning forest constructed at the end of the algorithm. Assume that the DFS continues by picking the smallest unmarked vertex until all vertices are visited.*



- Dfs (0)

Marked []

0	1	2	3	4	5	6	7	8	9	10	11	12
T												

EdgeTo []

0	1	2	3	4	5	6	7	8	9	10	11	12
0												

Id []

0	1	2	3	4	5	6	7	8	9	10	11	12
0												

- Dfs (1)

Marked []

0	1	2	3	4	5	6	7	8	9	10	11	12
T	T											

EdgeTo []

0	1	2	3	4	5	6	7	8	9	10	11	12
0	0											

Id []

0	1	2	3	4	5	6	7	8	9	10	11	12
0	0											

- Check 0
- 1 done
- Dfs (2)

Marked []

0	1	2	3	4	5	6	7	8	9	10	11	12
T	T	T										

EdgeTo []

0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0										

Id []

0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0										

- Check 0
- 2 done
- Dfs (5)

Marked []

0	1	2	3	4	5	6	7	8	9	10	11	12
T	T	T			T							

EdgeTo []

0	1	2	3	4	5	6	7	8	9	10	11	12

0	0	0			0							

Id []

0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0			0							

- Check 0
- Dfs (3)

Marked []

0	1	2	3	4	5	6	7	8	9	10	11	12
T	T	T	T		T							

EdgeTo []

0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	5		0							

Id []

0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0		0							

- Dfs (4)

Marked []

0	1	2	3	4	5	6	7	8	9	10	11	12
T	T	T	T	T	T							

EdgeTo []

0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	5	3	0							

Id []

0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0							

- Check 3
- Check 5
- Dfs (6)

Marked []

0	1	2	3	4	5	6	7	8	9	10	11	12
T	T	T	T	T	T	T						

EdgeTo []

0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	5	3	0	4						

Id []

0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0						

- Check 0
- Check 4
- 6 done
- Check 3
- Check 5
- 4 done
- Check 5
- 3 done
- Check 0
- 5 done
- 0 done
- Dfs (7)

Marked []

0	1	2	3	4	5	6	7	8	9	10	11	12
T	T	T	T	T	T	T	T					

EdgeTo []

0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	5	3	0	4	7					

Id []

0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	1					

- Dfs (8)

Marked []

0	1	2	3	4	5	6	7	8	9	10	11	12
T	T	T	T	T	T	T	T	T				

EdgeTo []

0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	5	3	0	4	7	7				

Id []

0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	1	1				

- Check 7
- 8 done
- 7 done
- Dfs (9)

Marked []

0	1	2	3	4	5	6	7	8	9	10	11	12
T	T	T	T	T	T	T	T	T	T			

EdgeTo []

0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	5	3	0	4	7	7	9			

Id []

0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	1	1	2			

- Dfs (10)

Marked []

0	1	2	3	4	5	6	7	8	9	10	11	12
T	T	T	T	T	T	T	T	T	T	T		

EdgeTo []

0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	5	3	0	4	7	7	9	9		

Id []

0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	1	1	2	2		

- Check 9
- 10 done
- Dfs (11)

Marked []

0	1	2	3	4	5	6	7	8	9	10	11	12
T	T	T	T	T	T	T	T	T	T	T	T	

EdgeTo []

0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	5	3	0	4	7	7	9	9	9	

Id []

0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	1	1	2	2	2	

- Check 9
- Dfs (12)

Marked []

0	1	2	3	4	5	6	7	8	9	10	11	12
T	T	T	T	T	T	T	T	T	T	T	T	T

EdgeTo []

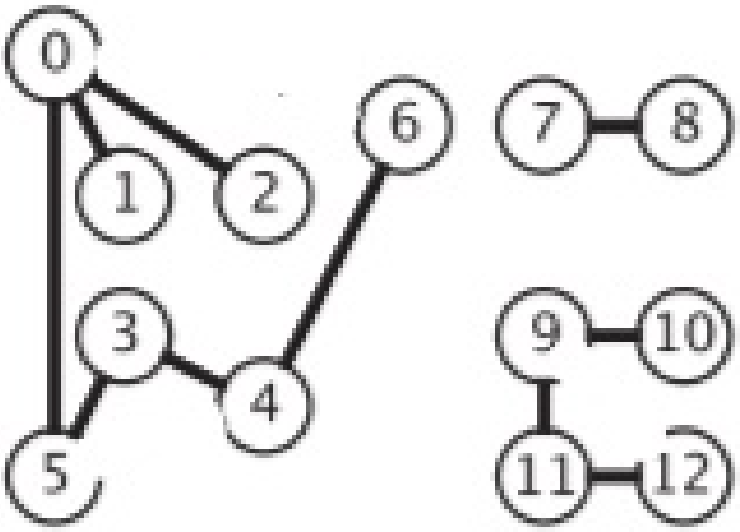
0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	5	3	0	4	7	7	9	9	9	11

Id []

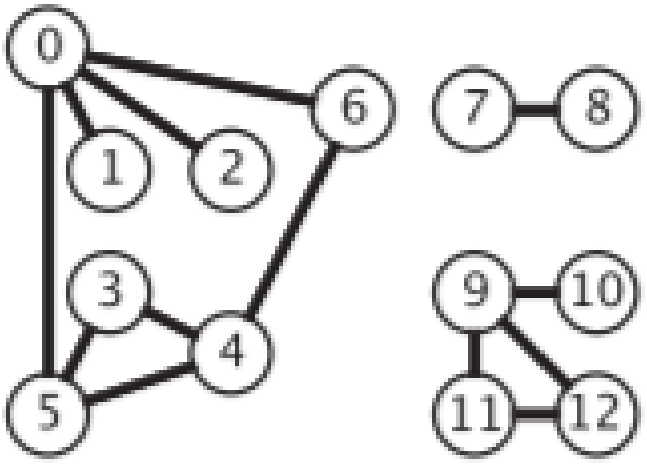
0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	1	1	2	2	2	2

- Check 9
- Check 11
- 12 done
- Check 9
- 11 done
- 9 done

Spanning DFS forest:



2. *Show a trace of BFS on this graph starting with vertex 0 by showing the contents of the arrays, marked [], EdgeTo [] during its execution. Show the BFS spanning forest constructed at the end of the algorithm. Assume that the BFS continues by picking the smallest unmarked vertex until all vertices are visited.*



- Bfs (0)
- Check 1, enqueue
- Check 2, enqueue
- Check 5, enqueue
- Check 6, enqueue
- 0 done

Marked []

0	1	2	3	4	5	6	7	8	9	10	11	12
T	T	T			T	T						

EdgeTo []

0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0			0	0						

- Bfs (1)
- Check 0
- 1 done

Marked []

0	1	2	3	4	5	6	7	8	9	10	11	12
T	T	T			T	T						

EdgeTo []

0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0			0	0						

- Bfs (2)
- Check 0
- 2 done

Marked []

0	1	2	3	4	5	6	7	8	9	10	11	12
T	T	T			T	T						

EdgeTo []

0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0			0	0						

- Bfs (5)
- Check 0
- Check 3, enqueue
- Check 4, enqueue
- 5 done

Marked []

0	1	2	3	4	5	6	7	8	9	10	11	12
T	T	T	T	T	T	T						

EdgeTo []

0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	5	5	0	0						

- Bfs(6)
- Check 0
- Check 4
- 6 done

Marked []

0	1	2	3	4	5	6	7	8	9	10	11	12
T	T	T	T	T	T	T						

EdgeTo []

0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	5	5	0	0						

- Bfs(3)
- Check 4
- Check 5
- 3 done

Marked []

0	1	2	3	4	5	6	7	8	9	10	11	12
T	T	T	T	T	T	T						

EdgeTo []

0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	5	5	0	0						

- Bfs(4)
- Check 3
- Check 5
- 4 done

Marked []

0	1	2	3	4	5	6	7	8	9	10	11	12
T	T	T	T	T	T	T						

EdgeTo []

0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	5	5	0	0						

- Bfs(7)
- Check 8, enqueue
- 7 done

Marked []

0	1	2	3	4	5	6	7	8	9	10	11	12
T	T	T	T	T	T	T	T	T				

EdgeTo []

0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	5	5	0	0	7	7				

- Bfs(8)
- Check 7
- 8 done

Marked []

0	1	2	3	4	5	6	7	8	9	10	11	12
T	T	T	T	T	T	T	T	T				

EdgeTo []

0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	5	5	0	0	7	7				

- Bfs(9)
- Check 10, enqueue
- Check 11, enqueue

- Check 12, enqueue
- 9 done

Marked []

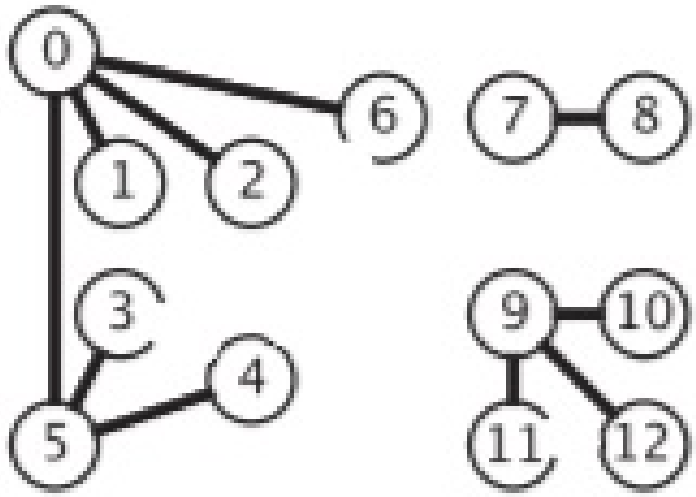
0	1	2	3	4	5	6	7	8	9	10	11	12
T	T	T	T	T	T	T	T	T	T	T	T	T

EdgeTo []

0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	5	5	0	0	7	7	9	9	9	9

- Bfs(10)
- Check 9
- 10 done
- Bfs(11)
- Check 9
- Check 12
- 11 done
- Bfs(12)
- Check 9
- Check 11
- 12 done

Spanning BFS forest:



3. *Prove that every connected graph has a vertex whose removal (including all adjacent edges) will not disconnect the graph by describing a DFS method that finds such a vertex.*

   Assumption: Every DFS search of a graph can be represented by a spanning forest. However, we know that if the graph is connected then we will simply have a single spanning tree.

   In every DFS traversal of a graph, there will be a node which is marked as visited last, at which point the algorithm backtracks to confirm so. This last node will be a leaf node in the spanning tree. If we remove this node, all other nodes will remain where they are in the spanning tree and thus, they will all still be connected.

4. *Design a BFS based algorithm to compute the girth of a bipartite graph. The girth of a graph is the length of its shortest cycle.*

   Since the graph is bipartite, we know that any cycles will be even, and they will be of length 4 or greater.

   With a BFS traversal, nodes can be separated into levels on a spanning tree and we can detect cycles in the graph by finding edges that connect 2 nodes of the same level OR 2 edges connecting one node to 2 nodes in the level above. However, since we know that with bipartite graphs, cycles will be even, we only need to look for the second case. Each level corresponds to one of the two groups from the bipartite graph so we will never have an edge connecting nodes of the same level.

The above can be simplified by simply looking for edges in BFS which visit a node which has already been marked. If we keep track of the distance from the root node to this marked node, once this edge is detected, we know the cycle length will be twice as large as the distance from the root. We then perform this modified BFS on every node in the graph to determine all cycles and keep track of the shortest one.

5. *Design an algorithm to determine whether a DAG has a Hamiltonian path. A path in a directed graph is called Hamiltonian if it visits each vertex exactly once.*

   Every DAG can be topologically sorted so we will do this first. This ordering produces a list of nodes where every vertex v with directed edge to u will always appear before u in the ordering. Thus, a Hamiltonian path will only exist if each node in the topological sort has a directed edge to the next edge in order.
   A topological sort is achieved using a stack along with a modified DFS traversal. In performing a DFS, once a visited node has no possible unvisited nodes to visit, it is pushed onto the stack. This is done recursively. Once the DFS visits every node, the stack is recursively popped, and this ordering produces a topological sort.