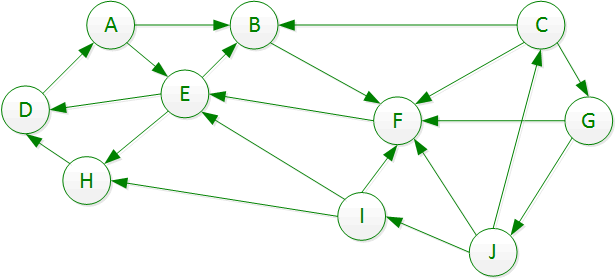
Nalongsone Danddank Student ID : 14958950 StarID: jf3893pd

Email: [nalongsone.danddank@my.metrostate.edu](mailto:nalongsone.danddank@my.metrostate.edu)

Homework 5 (15 pts)[[1]](#footnote-0)

1. [12 pts, 2 points per part]: Consider the directed graph below.



Part A: Do a depth first search of this graph, starting at node A. List the discovery and finishing times of each node, starting with node A at time 1. When you have a choice among two or more unvisited nodes to visit next, visit the one whose name comes first in the alphabet. Note that the entire graph is not reachable from node A. When you have completed node A, restart the DFS from the unvisited node that comes first in the alphabet.

***Part A Solution:***

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Node | A | B | C | D | E | F | G | H | I | J |
| Discovery Time | 1 | 2 | 13 | 5 | 4 | 3 | 14 | 7 | 16 | 15 |
| Finishing Time | 12 | 11 | 20 | 6 | 9 | 10 | 19 | 8 | 17 | 18 |

Part B: For each edge listed in the table, classify it as a Tree edge (T), Forward edge (F), Back edge (B), or Cross edge (C).

***Part B Solution:*** .

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Edge | AB | AE | BF | CB | CF | CG | DA | EB | ED | EH | FE | GF | GJ | HD | IE | IF | IH | JC | JF | JI |
| Type | T | F | T | C | C | T | B | B | T | T | T | C | T | B | C | C | C | B | C | T |

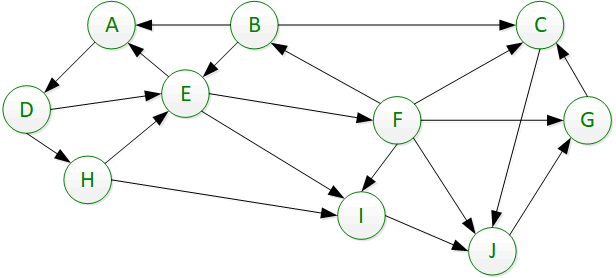
Note: Parts C and beyond are a step-by-step way to use Kosaraju’s algorithm to find the strongly connected components of this graph.

Part C: Draw the stack used in Kosaraju’s algorithm in the following array:

***Part C Solution:*** .

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Stack Top | * 🡪 🡪 🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪🡪 | | | | | | | | Stack Bot. |
| C | G | J | I | A | B | F | E | H | D |

Part D: Here is the complementary graph.



Calculate the discovery and finish times for the nodes in the table below. Remember to use the stack of Part C to guide your decision which node to process in which order. As before, break ties alphabetically.

***Part D Solution:***

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Node | A | B | C | D | E | F | G | H | I | J |
| Discovery Time | 9 | 13 | 1 | 10 | 11 | 12 | 3 | 19 | 7 | 2 |
| Finishing Time | 18 | 14 | 6 | 17 | 16 | 15 | 4 | 20 | 8 | 5 |

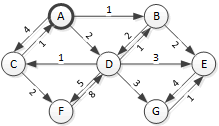
Part E: List the SCCs. You may not need all the lines.

***Part E Solution:***

\_\_\_\_CJG\_\_\_\_ \_\_I\_\_ \_ADEFB\_ \_H\_\_

Part F: What is the worse-case asymptotic running time of this algorithm, in terms of the numbers of vertices V and edges E in the graph? Justify your answer.:

***Part F Solution: The worse-case Time complexity is O(2V+2E) because we run all node V and edge E in the first graph, then we run the reversed graph all node V and edge E again second time. Or so Θ(V+E) for average time complexity.***

Questions 2 and 3 will involve this graph:

1. [3 pts]: Do a breadth first search of this graph, ignoring the edge weights. When choosing which node on the frontier to explore next, choose the node that comes earliest in the alphabet. See appendix for an example of how I want the table filled out.

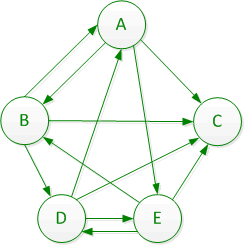
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| . Node | Edges from start | Prev Node | Queue after processing node | | | | | | | | | | | | | | |
| Node @ Head | Edges from start | Prev node | Node | Edges from start | Prev node | Node | Edges from start | Prev node | Node | Edges from start | Prev Node | Node | Edges from start | Prev Node |
| (init) |  |  | A | 0 | - |  |  |  |  |  |  |  |  |  |  |  |  |
| A | 0 | - | B | 1 | A | C | 1 | A | D | 1 | A |  |  |  |  |  |  |
| B | 1 | A | C | 1 | A | D | 1 | A | E | 2 | B |  |  |  |  |  |  |
| C | 1 | A | D | 1 | A | E | 2 | B | F | 2 | C |  |  |  |  |  |  |
| D | 1 | A | E | 2 | B | F | 2 | C | G | 2 | D |  |  |  |  |  |  |
| E | 2 | B | F | 2 | C | G | 2 | D |  |  |  |  |  |  |  |  |  |
| F | 2 | C | G | 2 | D |  |  |  |  |  |  |  |  |  |  |  |  |
| G | 2 | D |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

1. [3 pts] Do a lowest cost first search of all the nodes in this graph (ie, do Dijkstra’s algorithm), using the edge weights. When choosing which node on the frontier to explore next, break ties alphabetically. See appendix for an example of how I want the table filled out.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| . Node | Dist from start | Prev Node | Queue after processing node | | | | | | | | | | | | | | |
| Node @ Head | Dist from start | Prev node | Node | Dist from start | Prev node | Node | Dist from start | Prev node | Node | Dist from start | Prev Node | Node | Dist from start | Prev Node |
| (init) |  |  | A | 0 | - |  |  |  |  |  |  |  |  |  |  |  |  |
| A | 0 | - | B | 1 | A | D | 2 | A | C | 4 | A |  |  |  |  |  |  |
| B | 1 | A | D | 2 | A | D | 3 | B | E | 3 | B | C | 4 | A |  |  |  |
| D | 2 | A | D | 3 | B | E | 3 | B | C | 4 | A | G | 5 | D | F | 7 | D |
| D | 3 | B | E | 3 | B | C | 4 | A | G | 5 | D | F | 7 | D |  |  |  |
| E | 3 | B | C | 4 | A | G | 5 | D | F | 7 | D |  |  |  |  |  |  |
| C | 4 | A | G | 5 | D | F | 7 | D |  |  |  |  |  |  |  |  |  |
| **G** | **5** | D | F | 7 | D |  |  |  |  |  |  |  |  |  |  |  |  |

***Appendix:*** Here is how I would fill out the various tables for the graphs used in the exercises during class.

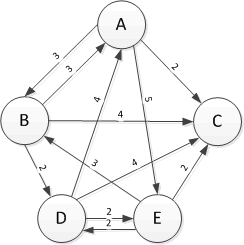
**Breadth first search example**



Note that for this solution, I care that you choose the next node properly, that the “Node at Head” is the highest-priority node in the queue, and that every other node on the queue is listed. Although for a pure BFS the nodes will be stored in the order discovered in an array, I’m not enforcing this (because it confuses people when we do Lowest-Cost-First search).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| . Node | Edges from start | Prev Node | Queue after processing node | | | | | | | | | | | | |
| Node at Head | Edges from start | Prev node | Node | Edges from start | Prev node | Node | Edges from start | Prev node | Node | Edges from start | Prev Node |
| (init) |  |  | A | 0 |  |  |  |  |  |  |  |  |  |  |
| A | 0 | - | B | 1 | A | C | 1 | A | E | 1 | A |  |  |  |
| B | 1 | A | C | 1 | A | E | 1 | A | D | 2 | B |  |  |  |
| C | 1 | A | E | 1 | A | D | 2 | B |  |  |  |  |  |  |
| E | 1 | A | D | 2 | B |  |  |  |  |  |  |  |  |  |
| D | 2 | B |  |  |  |  |  |  |  |  |  |  |  |  |

**Lowest Cost First Search example**



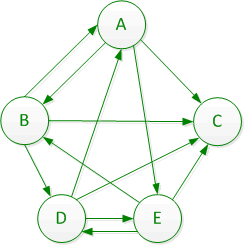
***Solution***:

Note that for this solution, I care that you choose the next node properly, that the “Node at Head” is the highest-priority node in the queue, and that every other node on the queue is listed. The order of any nodes after the head is completely irrelevant (it will be stored in a heap or something, anyway).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| . Node | Dist from start | Prev Node | Queue after processing node | | | | | | | | | | | | |
| Node at Head | Dist from start | Prev node | Node | Dist from start | Prev node | Node | Dist from start | Prev node | Node | Dist from start | Prev Node |
| (init) |  |  | A | 0 |  |  |  |  |  |  |  |  |  |  |
| A | 0 | - | C | 2 | A | B | 3 | A | E | 5 | A |  |  |  |
| C | 2 | A | B | 3 | A | E | 5 | A |  |  |  |  |  |  |
| B | 3 | A | E | 5 | A | D | 5 | B |  |  |  |  |  |  |
| E | 5 | A | D | 5 | B |  |  |  |  |  |  |  |  |  |
| D | 5 | B |  |  |  |  |  |  |  |  |  |  |  |  |

***Comment:*** Note that this is a priority queue, not a “plain” queue. New items are always added in their proper place in the priority scheme. Since the priority is nearer distance to the start node, with the tie-breaker being earlier letter in the alphabet, when node D is added to the priority queue is goes in front of node E. The distances from start node A are the same, so the tiebreaker comes into play . Contrast this with the “plain” queue for the BFS search above, where everything is added to the tail of the queue.

**Depth First Search example**



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Vertex | A | B | C | D | E |
| Discovery Time | 1 | 2 | 3 | 5 | 6 |
| Finishing Time | 10 | 9 | 4 | 8 | 7 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Edge | AB | AC | AE | BA | BC | BD | DA | DC | DE | EB | EC | ED |
| Type (BCFT) | T | F | F | B | T | T | B | C | T | B | C | B |

1. Note that 17 points are possible, so basically this homework includes a little extra credit. [↑](#footnote-ref-0)