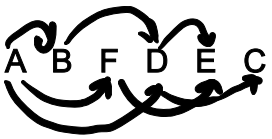
CS 61BL
Summer 2021

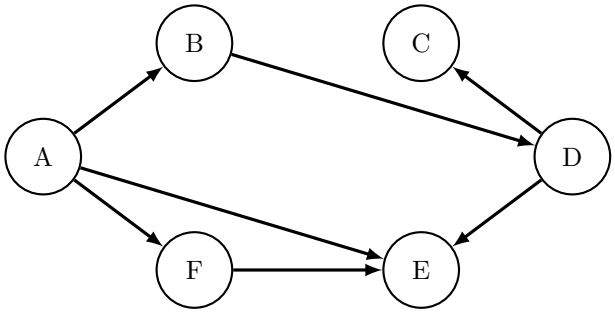
Shortest Paths and MSTs

Recurring Section 11: Thursday August 5, 2021

1 Topological Sorting



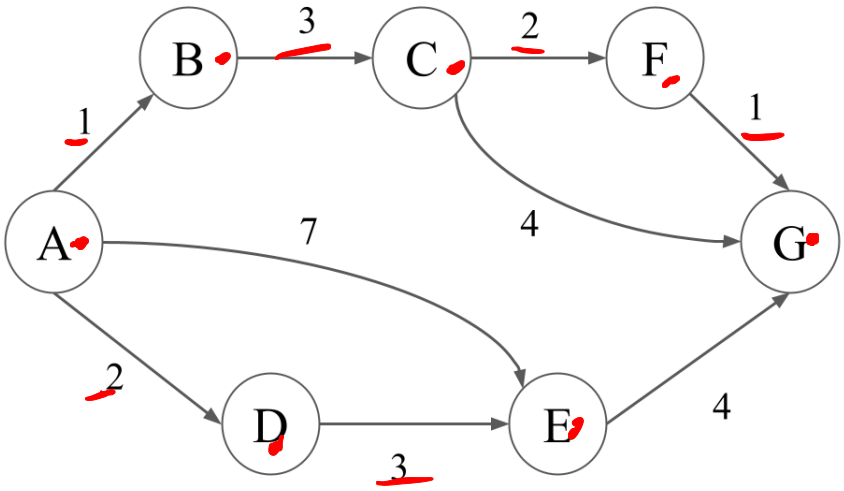
Give a valid topological ordering of the graph below.



Is the topological ordering of the graph unique? **No**

2 The Shortest Path To Your Heart

For the graph below, let `g(u, v)` be the weight of the edge between any nodes `u` and `v`. Let `h(u, v)` be the value returned by the heuristic for any nodes `u` and `v`.



Below, the pseudocode for Dijkstra's and A* are both shown for your reference throughout the problem.

Dijkstra's Pseudocode

```
PQ = new PriorityQueue()
PQ.add(A, 0)
PQ.add(v, infinity) # (all nodes except A).

distTo = {} # map
distTo[A] = 0
distTo[v] = infinity # (all nodes except A).

while (not PQ.isEmpty()):
  popNode, popPriority = PQ.pop()

  for child in popNode.children:
    if PQ.contains(child):
      potentialDist = distTo[popNode] +
        edgeWeight(popNode, child)
      if potentialDist < distTo[child]:
        distTo.put(child, potentialDist)
        PQ.changePriority(child, potentialDist)
```

A* Pseudocode

```
PQ = new PriorityQueue()
PQ.add(A, h(A))
PQ.add(v, infinity) # (all nodes except A).

distTo = {} # map
distTo[A] = 0
distTo[v] = infinity # (all nodes except A).

while (not PQ.isEmpty()):
  poppedNode, poppedPriority = PQ.pop()
  if (poppedNode == goal): terminate

    for child in poppedNode.children:
      if PQ.contains(child):
        potentialDist = distTo[poppedNode] +
          edgeWeight(poppedNode, child)

        if potentialDist < distTo[child]:
          distTo.put(child, potentialDist)
          PQ.changePriority(child, potentialDist + h(child))
```

(a) Run Dijkstra's algorithm to find the shortest paths from A to every other vertex. You may find it helpful to keep track of the priority queue and make a table of current distances.

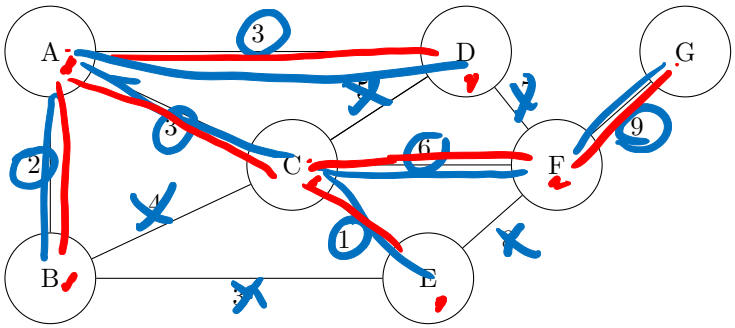
	A	B	C	D	E	F	G
DistTo	0	1	4	2	5	6	7
EdgeTo							

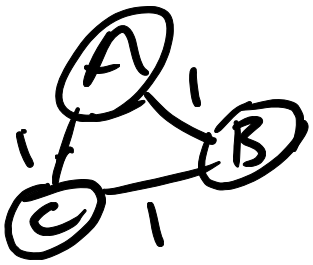
(b) Given the weights and heuristic values for the graph above, what path would A* search return, starting from A and with G as a goal?

Edge weights	Heuristics
$g(A,B) = 1$	$h(A,G) = 7$
$g(B,C) = 3$	$h(B,G) = 6$
$g(C,F) = 2$	$h(C,G) = 3$
$g(C,G) = 4$	$h(F,G) = 1$
$g(F,G) = 1$	$h(D,G) = 6$
$g(A,D) = 2$	$h(E,G) = 3$
$g(D,E) = 3$	
$g(E,G) = 4$	
$g(A,E) = 7$	

	A	B	C	D	E	F	G
DistTo							
EdgeTo							

3 Minimum Spanning Trees





(a) Given the graph above, run Kruskal's and Prim's algorithm to determine the minimum spanning tree of this graph. For Prim's algorithm, assume we start at node A and fill in the following chart including the value cost(v) for all vertices v for that iteration as well as which node was popped off of the fringe for that iteration. (Note: Ties are broken in alphabetical order)

v	init	Pop a	Pop b	Pop c	Pop e	Pop d	Pop f	Pop g
cost(a)	**0**	0	0	0	0	0	0	0
cost(b)	∞	2	2	2	2	2	2	2
cost(c)	∞	3	3	3	3	3	3	3
cost(d)	∞	3	3	3	3	3	3	3
cost(e)	∞	∞	3	1	1	1	1	1
cost(f)	∞	∞	∞	6	6	6	6	6
cost(g)	∞	∞	∞	∞	∞	∞	9	9

(b) Run Kruskal's algorithm on the same graph.

(c) Does Kruskal's algorithm for finding the minimum spanning tree work on graphs with negative edge weights? Does Prim's?

(d) True or False: A graph with unique edge weights has a unique minimum spanning tree.

4 Extra: Fiat Lux

After graduating from Berkeley with solid understanding of CS61B topics, Josh became a billionaire and wants to build power stations across Berkeley campus to help students survive from PG&E power outages. Josh want to minimize his cost, but due to the numerous power outages when he took CS61B, he did not learn anything about Prim's or Kruskal's algorithm and he is asking for your help! We must meet the following constrains to power the whole campus:

- There are V locations where Josh can build power stations, and it costs v_i dollars to build a power station at the i^{th} position.
- There are E positions we can build wires and it cost e_{ij} to build a wire between location i and j.
- All locations must have a power station itself or be connected to another position with power station.
- $e_{ij} << v_i, \forall i, j$

Use the Prim's or Kruskal's algorithm taught in class to find a strategy that will minimize the cost while still fulfilling the constrains above.