CS 61B Spring 2026

Shortest Paths and MSTs

Discussion 10: March 30, 2026

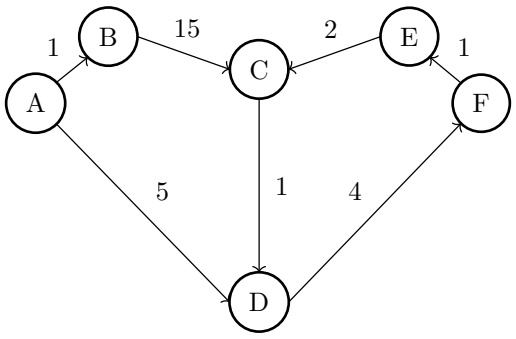
1 Warmup

(a) Draw lines to match each algorithm (top row) to the problem it solves (bottom row).

Dijkstra's	**Prim's**	**Kruskal's**	Reverse DFS post-order traversal
Processes vertices in order of distance from source	Grows a tree by adding the shortest edge to a new vertex	Adds edges in order of weight, skipping those that create cycles	

Topological Sort	**Minimum Spanning Tree**	**Shortest Paths**
Order vertices such that for every directed edge u→v, u comes before v in the ordering.	Find a tree with minimum total edge weight that connects all vertices	Find the minimum-weight path from a source vertex to all other vertices

2 The Shortest Path to your Heart



Dijkstra's Pseudocode Dijkstra's finds the shortest paths from a starting node v to all other nodes. The overall strategy is to process nodes in order of distance from v, starting from the closest.

```
PQ = new PriorityQueue()                    // All vertices we haven't yet finalized the shortest
                                            // path to, ordered by distance to v.

distTo = {}                                 // Map of vertex to best known distance from v.
edgeTo = {}                                 // Map of vertex to the vertex we came from.

PQ.add(v, 0)                                // Distance from the starting node v to itself is 0.
distTo[v] = 0
for all other vertices u:
  PQ.add(u, infinity)                       // Distance infinity because we don't yet
  distTo[u] = infinity                      // know any paths to u.

while (not PQ.isEmpty()):
  u, dist = PQ.pop()                        // u is the closest unprocessed vertex.

  for all vertices w adjacent to u:
    potentialDist = dist + edgeWeight(u, w)

    if potentialDist < distTo[w]:           // See if there's a shorter path to
      distTo.put(w, potentialDist)          // each neighbor via u. If so, update
      PQ.changePriority(w, potentialDist)   // distTo, edgeTo, and the weight in
      edgeTo[w] = u                         // PQ accordingly.
```

(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. We have provided a table to keep track of best distances, and the adjacent vertex that has an edge going to the target vertex in the current shortest paths tree so far.

	A	B	C	D	E	F
distTo						
edgeTo						

(b) Answer the following questions about the runtime of Dijkstra's algorithm. Write your answers in terms of the number of vertices V and the number of edges E.

i. How many total calls are made to `PQ.add()` on lines 5 and 8?

ii. How many total calls are made to `PQ.pop()` on line 12?

iii. How many total calls are made to `PQ.changePriority()` on line 19?

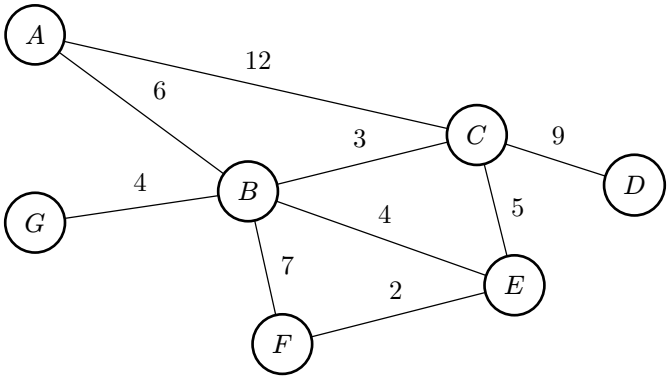
iv. All priority queue operations take $O(\log N)$ time, where N is the number of items in the priority queue. Given that, what's the total runtime? Simplify as much as possible, using the assumption that $V < E$. Your answer should contain only E and V (no N).

3 Introduction to MSTs

- **Prim's algorithm:** Start from any vertex. Repeatedly add the shortest edge that connects a vertex in the tree to a vertex not yet in the tree.
- **Kruskal's algorithm:** Sort all edges by weight. Add edges in order from smallest to largest, skipping any edge that would create a cycle.

(a) Fill in the blank with an expression using V (the number of vertices) and/or E (the number of edges):

Both algorithms stop when all V vertices are connected to the tree, which occurs when the tree contains _____ edges.



(b) For the graph above, list the edges in the order they're added to the MST by Kruskal's and Prim's algorithm. Assume Prim's algorithm starts at vertex A. Assume ties are broken in alphabetical order. Denote each edge as a pair of vertices (e.g. AB is the edge from A to B). The first edge is shown for you.

Prim's algorithm order: AB,

Kruskal's algorithm order: EF,

(c) True/False: Adding 1 to the smallest edge of a graph G with unique edge weights must change the total weight of its MST.

(d) True/False: If all edge weights in a graph are unique, there is only one possible MST.

(e) True/False: The shortest path from vertex u to vertex v in a graph G is the same as the shortest path from u to v using only edges in T, where T is the MST of G.

4 Class Enrollments

Review: DFS for Topological Sort

(a) Run DFS from a vertex with no incoming edges (indegree 0). Record the post-order: add each vertex to a list after visiting all its neighbors.
(b) If any vertices aren't yet in the list, repeat step (a), beginning from a different vertex with no incoming edges.
(c) Reverse the post-order to get a valid topological ordering

You're planning your CS classes for the upcoming semesters, but it's hard to keep track of all the prerequisites! Let's figure out a valid ordering of the classes you're interested in. A valid ordering is an ordering of classes such that every prerequisite of a class is taken before the class itself. Assume we're taking one CS class per semester.

(a) The list of prerequisites for each course is given below (not necessarily accurate to actual courses!). Draw a graph to represent our scenario.

- CS 61A: None
- CS 61B: CS 61A
- CS 61C: CS 61B
- CS 70: None
- CS 170: CS 61B, CS 70
- CS 161: CS 61C, CS 70

(b) Suppose we added a new prerequisite where the student must take CS 161 before CS 170 and CS 170 before CS 61C. Is there still a valid ordering of classes such that no prerequisites are broken? If no, explain.

(c) With the original graph, perform a topological sort to find a valid ordering of the 6 classes. Break ties by going to the lower course number first.