1. Introduction

(a) Why study graphs and graph algorithms?
   i. Graphs are (reasonably) accurate models for many real world problems: traffic,
      social structure, images.
   ii. Real world graphs are huge. Standard medical image has about $10^9$ vertices, web
      graph has $10^{12}$ vertices. Almost arbitrary pairwise associations makes the introduc-
      tion of parallelism non-trivial too.
   iii. Many fundamental questions on graphs that we don’t fully understand.

(b) Why these topics?
   i. Plan is to group the topics into the following categories: combinatorial algorithms,
      data structures and randomized algorithms.
   ii. Many of the more recent algorithms use tools from all 3 of these categories. So in
      some sense ideas involved in them are essential for designing graph algorithms.
   iii. Algorithms in later categories would often use techniques from earlier ones, making
      this a natural order of presentation as well.
   iv. Topics chosen are a bit heavy on the optimization side. Any preferences?

(c) What’s the right size for a problem set?

(d) What is a graph? Simplified description: a collection of objects and pairwise associations
   between them.

(e) Notation: $G = (V, E)$ for graph, $w$ for weights, $c$ for costs or capacities, $u, v$ for vertices.
    $|V| = n, |E| = m$. Typically assume $m = O(n \text{ poly log } n)$.

(f) Model of computation: pointer model, references to memory addresses treated as distinct
   from data. For example, an edge is a pair of pointers to vertices and its weight is
   considered data. Can follow pointers in $O(1)$ time.

2. Shortest Path

(a) Problem ($s$-$t$ shortest path): given directed graphs with costs along edges, $G = (V, E, c)$,
   and two vertices $s$ and $t$. Find a path from $s$ to $t$ of minimum cost.

(b) Assumption: no negative weight cycles.

(c) Alternate view: Label each vertex $u$ with a distance $y_u$. $y_s = 0$, maximize $y_t$ subject to
   the condition that none of the distance labels are ‘improvable’. Edge $uv$ can be used to
   improve the labels if $y_u + c_{uv} \leq y_v$.

(d) Shortest path tree
   i. Tree: $n - 1$ edges that connect the graph.
   ii. All shortest paths from $s$ form a tree.
   iii. Proof: for each vertex $u \neq s$, consider the second last vertex on the shortest path
       to $u$.

(e) Bellman-Ford algorithm
i. start with $y_u = \infty$ except $y_s = 0$.
ii. Iteratively go through all edges $uv$, $y_v \leftarrow \min(y_v, y_u + c_{uv})$ (relaxing $uv$).
iii. If we have level $i$ of the tree correct, then level $i+1$ becomes correct in one iteration, $O(m)$ time and parallelizable.
iv. Tree have at most $n$ level gives $O(nm)$ time total.

(f) Dijkstra’s algorithm
i. Only works on graphs with non-negative edge weights.
ii. Each step pick vertex with minimum value of $y$, mark it as done, and relax all edges $uv$.
iii. Non-negativity and induction means nodes that we pick already has correct $y$ value.
iv. $O(n^2)$ time, can run in $O(n \log n + m)$.
v. In unweighted case, becomes breath first search (BFS). Implementable in $O(m)$ using a queue.

(g) **Open**: practical shortest path algorithm faster than $O(n \log n + m)$ in the pointer model.

3. Minimum Spanning Trees

(a) Problem: given undirected, weighted graph $G = (V, E, w)$, find a tree of minimum total weight.

(b) Alternative definition: minimum weight of edges needed to connect the graph.

(c) Assume all edges have distinct weights (by adding small perturbations to edge weights)

(d) Characterizations:

i. Based on the fact that removing an edge from a tree and add an edge connecting the two resulting pieces, we still get a tree. Related to matroids.

ii. Edge $uv$ in tree if it’s the minimum in some cut $\delta(S, \bar{S})$. Proof: consider optimum tree and add $uv$ to it, the resulting cycle must have an edge $u'v'$ that cross the cut $\delta(S, \bar{S})$. Removing that edge leads to a tree with smaller weight.

iii. Edge $uv$ not in tree if there is a path from $u$ to $v$ using edges of smaller weight. Proof: for a tree containing $uv$, some edge on this path, $u'v'$ must be not included (or we have a cycle). Remove $uv$ and add $u'v'$ gives a tree of smaller weight.

(e) Algorithms:

i. Prim’s algorithm: Start at any vertex, add minimum weight edge out of current connected set of vertices, add other end point to connected set. (proven using characterization 1, won’t discuss)

ii. Kruskal’s algorithm: sort edges by weight in increasing order, add edge if its two end points have not been connected yet. (proven using characterization 2, won’t discuss)

iii. Borůvková’s algorithm: for all vertices, find minimum weight edge with it as one end point. Add edges to tree and contract, repeat. Proven using characterization 1 by considering cut $\delta(u, V \setminus \{u\})$. Runtime proof: edges picked ‘cover’ at least $n$ vertices, so at least $n/2$ edges picked. This halves the number of vertices, so we terminate in $O(\log n)$ iterations.

(f) **Open**: deterministic, $O(m)$ time algorithm for minimum spanning tree in the comparison model.