1. Problem
   (a) Directed graph, find $k^{th}$ shortest path from $s$ to $t$. Paths may have loops.
   (b) Why? Additional constraints in optimization that’s hard to incorporate.
   (c) Simple: track length of $k$ shortest paths at each vertex.
   (d) Can view as $k$ layers of graph, deviating from best path in current layer brings one to next layer.
   (e) Goal: $O((m + k)\log n)$ time, due to Eppstein ‘94.

2. Additional Structure
   (a) Recall: vertex labels, $y: V \rightarrow \mathbb{R}$.
   (b) Grow shortest path tree into $t$. $y$: distance to $t$. Shortest path satisfy $y_v + d_{uv} \geq y_u$.
   (c) New distances: $d'_{uv} = y_v + d_{uv} - y_u$; $d'_{uv} \geq 0$.
   (d) Lengths of $s$-$t$ paths change by constant.
   (e) All edges of the shortest path tree have 0 weight.
   (f) Can get from any vertex to $t$ with 0 cost.
   (g) Each path corresponds to a sequence of non-tree edges, and weight equals to the (modified) weight of those non-tree edges.
   (h) Can take any non-tree edge belonging to a parent.

3. Modified Problem
   (a) $k^{th}$ smallest sequence of edges $u_1v_1, u_2v_2 \ldots u_lv_l$ such that $u_1$ is ancestor of $s$, $u_{i+1}$ is ancestor of $v_i$.
   (b) Simple case: tree has low depth.
      i. Can have many edges incident to a vertex.
      ii. Process them in order suffice.
      iii. Global list of ‘possible’ edges, pick minimum weight one.
      iv. Priority queue, $O(m \log n + kh \log n)$.
   (c) Data structure needed: for each node, a sorted list of all edges incident to all of its ancestors.
   (d) Hammer: heavy light decomposition, then binary decomposition along each path. Each node maps to $O(\log^2 n)$ sorted lists.
   (e) How to choose which list: priority queue, e.g. binary heap.
   (f) Only need to access a node if its parent has been accessed.
   (g) Better solution: maintain binary min-heap for a traversal of a tree. Add all edges when we discover a node, remove those edges when we finish with that node.
   (h) ‘State’ of the data structure after discovering each node is what we need.
(i) Persistent data structures: can access any historical version.

(j) Simple solution for binary heaps with $O(\log n)$ memory overhead: recreate new access path.

(k) Can remove memory overhead, Driscoll, Sarnak, Sleator, and Tarjan, ‘89.