13 Multiple-Source Shortest Paths[α]

13.1 Problem Statement

Let $\Sigma = (V, E, F)$ be a planar map with outer face o, where each edge e is assigned a non-negative weight $w(e)$.[1] Call any vertex incident to o a *boundary* vertex of Σ. The *multiple-source shortest-path* problem asks for an implicit representation of the shortest paths from s to t, for all boundary vertices s and all vertices t. An explicit representation of these shortest paths, for example as a shortest-path tree rooted at every node on the outer face, requires $\Omega(n^2)$ space in the worst case. Nevertheless, the multiple-source shortest-path problem can be solved in only $O(n \log n)$ time, in any of the following forms:

- Given a collection of k vertex pairs (s_i, t_i), where each s_i is a boundary vertex, we can report all k shortest-path distances $\mathsf{dist}(s_i, t_i)$ in $O(n \log n + k \log n)$ time.
- Assuming the outer face o has k vertices, we can report the $O(k^2)$ shortest-paths distances between every pair of boundary vertices in $O(n \log n + k^2)$ time.
- We can preprocess Σ in $O(n \log n)$ time into a data structure using $O(n \log n)$ space, that can report the shortest-path distance from an arbitrary boundary vertex to an arbitrary vertex in $O(\log n)$ time.

The multiple-source shortest-path problem was first posed and solved by Philip Klein in 2005. Here I'm describing a variant of Klein's algorithm published by Sergio Cabello and Erin Chambers in 2007, which more easily generalizes to graphs on non-planar surface maps. This algorithm plays an essential role in several efficient algorithms for planar maps and surface maps.

To ease presentation, I will make two simplifying assumptions about the input graph:

(1) The boundary of the outer face o is a simple cycle. This assumption can be enforced if necessary by inserting additional edges with very large weight.
(2) For every vertex s and every vertex t, there is ***exactly one*** shortest path from s to t. (Thus, the algorithm I'll describe here cannot be used verbatim on unweighted graphs.) I'll describe two easy methods to enforce this assumption at the end of this note.

13.2 Shortest paths and slacks

The MSSP algorithm relies on a characterization of shortest paths developed by Lester Ford in the mid-1950s. Fix a *source* vertex s. For each vertex v, let $\mathsf{dist}(v)$ denote the shortest-path distance from s to v. Let $\mathsf{pred}(v)$ denote the predecessor of vertex v (if any) in the unique shortest path from s to v. Let T_s denote the tree of shortest paths from s to other vertices, defined so that $\mathsf{pred}(v)$ is the parent of v in T_s. Finally, define the *slack* of each dart $u \to v$ as

$$\mathsf{slack}(u \to v) := \mathsf{dist}(u) + w(u \to v) - \mathsf{dist}(v)$$

A dart whose slack is negative is called *tense*.

Ford's generic single-source shortest path algorithm starts by assigning $\mathsf{dist}(s) = 0$ and *tentatively* assigning $\mathsf{dist}(v) = \infty$ for every vertex $v \neq s$. Then as long as the graph contains at least one tense dart, the algorithm *relaxes* one tense dart $u \to v$ by reassigning $\mathsf{dist}(v) \leftarrow \mathsf{dist}(u) + w(u \to v)$

[1] In fact the algorithm I'm about to describe can be extended to directed graphs, where a dart and its reversal may have different weights, but for ease of exposition, I'll stick to undirected graphs in this lecture.

and pred(v) $\leftarrow u$. When no more darts are tense, every value dist(v) is the correct shortest-path distance, and the predecessor pointers define a correct shortest-path tree T_s.

Lemma (Ford 1956): *The following invariants hold for any shortest-path tree T_s in any edge-weighted graph G:*

(a) *Every dart in G has non-negative slack.*

(b) *Every dart in a shortest path tree T_s (directed away from s) has slack zero.*

(c) *If shortest paths are unique, then every dart that is not in the unique shortest-path tree T_s has positive slack.*

13.3 Compact Output

Disk-tree Lemma: *Let T be any tree embedded on a disk with boundary cycle B; call any vertex in $T \cap B$ a boundary vertex. Let e be any edge of T, and let U and W be the components of $T \setminus e$. Either U contains no vertices, or U contains every boundary vertex, or boundary vertices in U induce a path in B.*

Proof: Let Σ be the planar map induced by $T \cup B$. Trivially, T is a spanning tree of Σ. The complementary dual spanning tree C^* of Σ^* is a star, with the outer face of Σ at the center and other faces of Σ at the leaves.

The dual subgraph C^*/e^* contains a cycle γ of length 2 that separates all vertices in U from all vertices in W. If γ does not intersect B, then U either contains every boundary vertex or none. Otherwise, γ intersects B exactly twice, so U contains an interval of boundary vertices. □

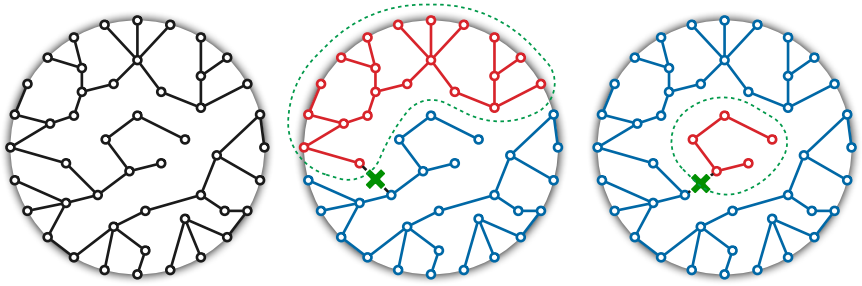

Figure 1: The disk-tree lemma.

Now suppose our original planar map Σ has h boundary vertices, indexed $s_0, s_1, s_2, \ldots, s_{h-1}$ in cyclic order. For each index i, let T_i denote the shortest-path tree rooted at s_i.

Corollary: *Every directed edge $x \rightarrow y$ is either in every shortest path tree T_i, in no shortest path tree T_i, or in an interval of shortest path trees $T_i, T_{i+1 \bmod h}, \ldots, T_{i+j \bmod h}$.*

Proof: Let T be the unique tree of directed shortest paths *into* vertex y, and apply the disk-tree lemma to the components of $T - xy$. □.

It follows that we can encode all k shortest paths using only $O(n)$ space, either by recording the first and last trees T_i that contain each directed edge, or by recording the initial tree T_1 followed by the differences $T_2 \setminus T_1, T_3 \setminus T_2 \ldots T_h \setminus T_{h-1}$.

13.4 Parametric Shortest Paths

[[Double- and triple-check directions for consistency: Source vertex x moves ccw around boundary. Darts pivoting into T_λ point from new parent to child. Dual dart d^* is right(d)→left(d). But left and right are reversed in the dual plane!]]

To solve the multiple-source shortest path problem, imagine moving the source vertex s *continuously* around the outer face and maintaining the shortest-path tree T_s rooted at s. Although the shortest-path *distances* vary continuously as s moves, the *structure* of the shortest-path tree changes only at discrete events. (This approach is a variant of the *parametric shortest-path* problem first proposed by Karp and Orlin (1981).)

Now consider a single edge uv on the outer face. Suppose we have already computed the shortest-path tree T_u rooted at u, and we want to maintain the shortest path tree T_s as the source vertex s moves along uv from u to v. We insert s as a new vertex, partitioning uv into two edges us and sv with parametric weights

$$w_\lambda(us) = \lambda \cdot w(uv) \qquad \text{and} \qquad w_\lambda(sv) = (1-\lambda)w(uv)$$

Every other edge xy has constant parametric weight $w_\lambda(xy) = xy$. We then maintain the shortest-path tree T_λ rooted at s, with respect to the weight function w_λ, as the parameter λ increases continuously from 0 to 1. The initial shortest-path tree T_0 is equal to T_u, and the final tree T_1 is equal to T_v.

Fix a parameter value $\lambda \in [0,1]$. For any vertex x, let $\mathsf{dist}_\lambda(x)$ denote the shortest-path distance from s to x with respect to the weight function w_λ. Similarly, for any dart $x \to y$, let

$$\mathsf{slack}_\lambda(x \to y) = \mathsf{dist}_\lambda(x) + w_\lambda(x \to y) - \mathsf{dist}_\lambda(y).$$

Color each vertex x *red* if $\mathsf{dist}_\lambda(x)$ is an increasing function of λ (with derivative 1), and *blue* if $\mathsf{dist}_\lambda(x)$ is an decreasing function of λ (with derivative -1). For generic values of λ, every vertex except s is either red or blue. Finally, call a dart $x \to y$ *active* if $\mathsf{slack}_\lambda(x \to y)$ is a decreasing function of λ.

Lemma: *The following invariants hold for all $\lambda \in [0,1]$:*

(a) *If $s \to v \notin T_\lambda$, then every vertex except s is red, and the only active dart is $s \to v$.*

(a) *If $s \to u \notin T_\lambda$, then every vertex except s is blue, and there are no active darts.*

(a) *Otherwise, every descendant of u is red, every descendant of v is blue, and $x \to y$ is active if and only if x is blue and y is red.*

Without loss of generality, assume $o = \mathsf{right}(u \to v)$ is the outer face of Σ. Let $p = \mathsf{left}(u \to v)$ be the other face incident to uv. Let $C_\lambda^* = (E \setminus T_\lambda)^*$ denote the spanning tree of Σ^* complementary to T_λ. Finally, let π_λ denote the unique directed path in C_λ^* from o^* to p^*.

Lemma: *If T_λ has both red and blue vertices, then a dart is active if and only if its dual is in the directed path π_λ.*

Corollary: *If T_λ has both red and blue vertices, the next dart to become tense (if any) is the dart with minimum slack whose dual is in the directed path π_λ.*

Thus, we can execute a single phase of the MSSP algorithm as follows. Initially, we set $s = u$. We repeatedly find the tensest active dart $d = x \to y$, move s distance $\mathsf{slack}(d)/2$ along uv, increase

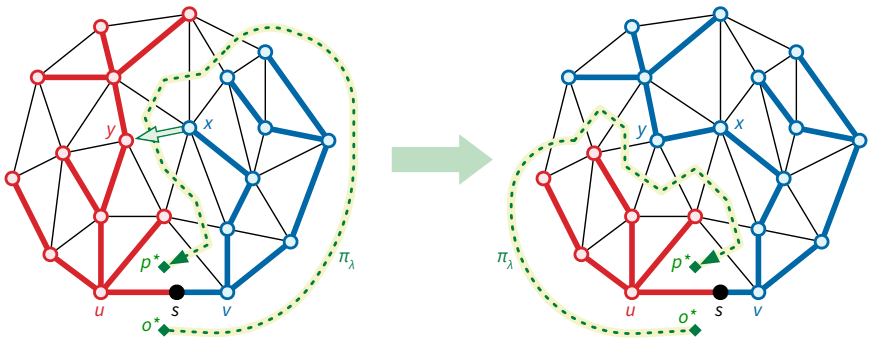

Figure 2: A single pivot in a planar shortest-path tree.

all red distances and decrease all blue distances by $\mathsf{slack}(d)/2$, decrease the slacks of all active darts and increase the slacks of their reversals by $\mathsf{slack}(d)$, and finally pivot d into the tree by assigning $\mathsf{pred}(y) \leftarrow x$. The loop ends either when there are no more active darts, or when the source vertex s reaches v.

Each pivot changes at least one node y from red to blue, and no pivot changes any node from blue to red. Thus, once a dart is pivoted into the shortest-path tree, it is not pivoted out during that phase. Thus, the darts that are pivoted into the tree are precisely the darts in $T_v \setminus T_u$. The disk-tree lemma now immediately implies that the *total* number of pivots over all phases is only linear.

Lemma: *The MSSP algorithm performs a total of $O(n)$ pivots.*

13.5 Dynamic Forest Data Structures

To achieve a running time of $O(n \log n)$, we need to perform each pivot quickly. We maintain both the shortest-path tree T_λ and the complementary dual spanning tree C_λ^* in data *dynamic forest* data structures that implicitly maintain dart values (slacks) or vertex values (distances) under edge insertions, edge deletions, and updates to the values in certain substructures.

We maintain the shortest path tree T_λ as a directed tree rooted at s, with dist values associated with each vertex, in a data structure that supports the following operations:

- $\mathsf{Cut}(x \to y)$: Remove the edge $x \to y$ from T_λ, breaking it into two rooted trees. The component containing x is still rooted at s; the other component is rooted at y.
- $\mathsf{Link}(x, y)$: Add a directed edge from x to y. This operation assumes that y a root. We always call Link immediately after Cut so that T_λ remains a single spanning tree.
- $\mathsf{GetDist}(x)$: Return the distance value associated with vertex x.
- $\mathsf{AddSubtreeDist}(x)$: For every descendant y of x, add Δ to $\mathsf{dist}(y)$.

Similarly, we maintain the complementary dual spanning tree C_λ^* as an undirected unrooted tree, with slack values associated with every dart, in a data structure that supports the following operations.

- $\mathsf{Cut}(xy)$: Remove the edge xy from C_λ^*, breaking it into two trees.

- Link(x, y, α, β): Add the edge xy and assign slack$(x \to y) = \alpha$ and slack$(y \to x) = \beta$. We always call Link immediately after Cut so that C_λ^* remains a single dual spanning tree.
- GetSlack$(x \to y)$: Return the slack value associated with dart $x \to y$.
- MinPathSlack(Δ, x, y): Return the d on the directed path from x to y such that slack(d) is minimized.
- AddPathSlack(Δ, x, y): For each dart d on the directed path from x to y, add Δ to slack(d) and subtract Δ from slack(rev(d)).

There are several dynamic-forest data structures that support the operations we need in $O(\log n)$ amortized time each; my favorite is Tarjan and Werneck's *self-adjusting top tree*. (Most of the others also have Tarjan's name on them.) A description of self-adjusting top trees (or the *splay trees* they use under the hood) is unfortunately beyond the scope of this note (or this course).

13.6 The Pivoting Algorithm

With these data structures in hand, we can identify the tensest active dart and perform the necessary updates to pivot it into T_λ in $O(\log n)$ amortised time. The following figure shows the algorithm to perform the next pivot, with all data structure operations in place.

```
NEXTPIVOT:
  if pred(u) ≠ s
      return 1
  if pred(v) ≠ s
      d ← s→v
      Δ ← GETSLACK(d*)
  else
      d* ← MINPATHSLACK(o*, p*)
      Δ ← GETSLACK(d*)/2
  if λ + Δ/w(uv) < 1
      PIVOT(d, Δ)
      return λ + Δ/w(uv)
  else
      return 1
```

```
PIVOT(x→y, Δ):
  ⟨⟨Update distances⟩⟩
  if pred(u) = s then ADDSUBTREEDIST(Δ, u)
  if pred(v) = s then ADDSUBTREEDIST(−Δ, v)
  ⟨⟨Update slacks⟩⟩
  ADDPATHSLACK(−Δ, o*, p*)
  ⟨⟨Update primal and dual trees⟩⟩
  z ← pred(y)
  pred(y) ← x
  CUT(zy)
  JOIN(x, y)
  CUT((xy)*)
  JOIN((z→y)*, 0, w(yz))
```

Figure 3: The MSSP pivoting algorithm.

As we already argued, the total number of pivots is $O(n)$, so the overall MSSP algorithm runs in $O(n \log n)$ time, as claimed.

13.7 Applications

Computing all k^2 boundary-to-boundary distances $O((n + k^2) \log n)$ time is straightforward.

[[More detail. Reduce time to $O(n \log n + k^2)$. Say anything at all about persistence?]]

13.8 Enforcing Unique Shortest Paths

So far our presentation has assumed that there is a *unique* shortest path between any two vertices of Σ; in particular, for any source vertex s, there is a *unique* shortest-path tree T_s. More subtly, we have also assumes that there is always a unique tensest active dart. These assumption obviously

do not hold in general, but we can enforce them if necessary using any of several standard *perturbation* techniques. I'll describe two such techniques here, but there are other possibilities.

Standard perturbation methods either explicitly or implicitly define a secondary weight $w'(u \to v)$ for each each dart $u \to v$ in Σ. The *perturbed weight* of a dart d is then defined as

$$\tilde{w}(d) := w(d) + w'(d) \cdot \varepsilon$$

for some sufficiently small real number $\varepsilon > 0$. Rather than computing a particular value of ε, we consider the limiting behavior as ε approaches zero. Thus, we can consider each perturbed weight $\tilde{w}(d)$ to be an ordered pair or vector

$$\tilde{w}(d) := (w(d), w'(d)).$$

We compute lengths of paths by summing these vectors normally, but we compare path lengths *lexicographically*. That is, we consider one path π to be shorter than another path π' if either of the following conditions holds:

(a) $w(\pi) < w(\pi')$
(b) $w(\pi) = w(\pi')$ and $w'(\pi) < w'(\pi')$

13.8.1 Random Perturbation

The simplest perturbation method chooses *random* secondary weights for each edge. For example, if we choose each secondary weight $w'(e)$ uniformly at random from the real interval $[0, 1]$, then the lengths of all simple paths (indeed, the lengths of all finite walks) are distinct with probability 1.

Somewhat more realistically, the following lemma implies that we can choose small random *integers* for the secondary weights.

Isolation Lemma (Mulmuley, Vazirani, and Vazirani 1987): *Let $\mathscr{F}$ be any family of subsets of $[n]$. For each index $i \in [n]$, let $w'(i)$ be chosen independently and uniformly at random from $[N]$. Define the weight $w'(S)$ of any subset $S \subseteq [n]$ as $w'(S) = \sum_{i \in S} w'(i)$. With probability at least $1 - n/N$, the minimum-weight set in $\mathscr{F}$ is unique.*

[[Include the proof, which is relatively straightforward]]

Corollary: *If each perturbation weight $w'(e)$ is chosen independently and uniformly at random from $[n^4]$, then with probability $1 - 1/O(n)$, all shortest paths with respect to the perturbed weight function (w, w') are unique.*

Let me emphasize that the Isolation Lemma *only* implies that *shortest* paths are distinct; other pairs of paths may still have equal length, even after perturbation.

[[We also need unique tensest darts!!]]

13.8.2 Cotree Perturbation

Let $T \sqcup C$ be a tree-cotree decomposition of Σ. Root the dual spanning tree C^* at the dual of the outer face o^*, and direct all edges of C^* away from the root. We define the weight $w'(d)$ of each dart d of Σ as follows:

- If $d^* \in C^*$, let $w'(d)$ be the number of descendants of $\mathsf{head}(d^*) = \mathsf{left}(d)^*$ in C^*.

- If $\mathsf{rev}(d^*) \in C^*$, let $w'(d) = -w'(\mathsf{rev}(d))$.
- Otherwise, let $w'(d) = 0$.

Equivalently, for any dart $d \notin T$, let $\mathsf{cycle}_T(d)$ be the unique directed cycle in $T + d$. Then $w'(d)$ is the number of faces of Σ inside $\mathsf{cycle}_T(d)$ if that cycle is oriented counterclockwise, the negation of the number of interior cycles if the cycle is clockwise, and zero if d is in T.

Winding Lemma: *For any closed walk W in Σ, we have $\sum_{d \in W} w'(d) = \sum_{f \in F} \mathit{wind}(W, f)$.*

Proof: This is essentially the shoelace algorithm. We can compute the winding number of W around f by traversing the path in C^* from f^* to o^* and counting crossings. We can write W as the sum of the fundamental directed cycles determined by the non-tree edges of W. ***[[Incomplete]]***

The previous lemma implies that although the secondary weights w' depend on the choice of tree-cotree decomposition, the resulting shortest paths do not!

Corollary: *For any two paths π and π' with the same endpoints, and any two spanning trees T and T', we have $w'_T(\pi) < w'_T(\pi')$ if and only if $w'_{T'}(\pi) < w'_{T'}(\pi')$. Thus, shortest paths with respect to w'_T and $w'_{T'}$ coincide.*

[[Figure!]]

Theorem: *Cotree perturbation makes shortest paths unique.*

Proof: Let π and π' be two shortest paths from some vertex s to some other vertex t. By definition, we have $w(\pi) = w(\pi')$ and $w'(\pi) = w'(\pi')$. The latter condition implies that $\sum_{f \in F} \mathsf{wind}(\pi - \pi', f) = 0$. There are two cases to consider.

First, suppose π and π' do not cross. Then the closed walk $\pi - \pi'$ is a weakly simple closed curve, which implies that the non-zero winding numbers $\mathsf{wind}(\pi - \pi', f)$ are either all 1 or all -1. It follows that $\mathsf{wind}(\pi - \pi', f) = 0$ for *every* face f, which is only possible if π and π' use the same subset of edges. In other words, $\pi = \pi'$.

Now suppose π and π' cross at some vertex x. The prefixes $\alpha = \pi[s, x]$ and $\alpha' = \pi'[s, x]$ must be shortest paths, otherwise we could shorten one of π and π'. Similarly, the suffixes $\beta = \pi[x, t]$ and $\beta' = \pi'[x, t]$ must be shortest paths. The inductive hypothesis now implies that $\alpha = \alpha'$ and $\beta = \beta'$. Again, we conclude that $\pi = \pi'$. □

[[We also need unique tensest darts!!]]

13.9 Leftmost shortest paths

In fact, we can implement cotree perturbation without explicit secondary weights. Suppose π and π' are two paths with the same endpoints. We say that π is *to the left* of π' if the closed walk $\pi - \pi'$ winds negatively (clockwise) around at least one face, but does not wind positively (counterclockwise) around any face. If π is to the left of π', we immediately have $w'(\pi) < w'(\pi')$.

It is not hard to show that for any two paths π and π' with the same endpoints, either one path is the left of the other, or a third path is to the left of both of them. Thus, shortest paths with respect to cotree perturbation are always *leftmost* shortest paths.

We can simulate cotree perturbation by always breaking ties to the left. In the MSSP algorithm, we always pivot the *leafmost* tensest active dart.

[[Need more details here]]

13.9.1 Caveat Emptor!

Cotree perturbation is attractive both because it is deterministic and because it can often be implemented implicitly, but its asymmetry can be a disadvantage. Unless shortest paths are already unique, cotree perturbation yields shortest paths that are not symmetric, even when the original graph is undirected. The reversal of the *leftmost* shortest path from s to t is the *rightmost* shortest path from t to u. Thus, algorithms that rely on the usual behavior of undirected shortest paths cannot automatically use this technique.

As a simple example, consider the *non-crossing* shortest paths problem. Given an undirected planar map Σ with weighted edges and several pairs of vertices $(s_1, t_1), \ldots, (s_k, t_k)$ on the outer face, we want to compute shortest paths between each pair s_i and t_i that are pairwise non-crossing. If shortest paths in Σ are already unique, then it suffices to independently compute the shortest path in Σ from s_i to t_i for each index i. But suppose the terminals appear in order s_1, t_1, s_2, t_2 on the outer face, and we use cotree perturbation to *enforce* uniqueness. Then the shortest path from s_1 to t_1 might cross the shortest path from s_2 to t_2.

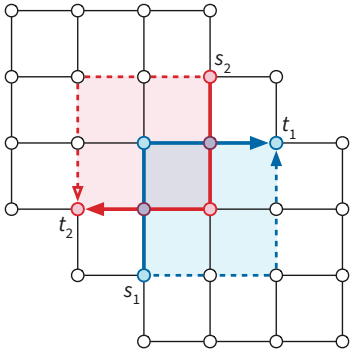

Figure 4: Leftmost shortest paths in undirected planar graphs can cross

13.10 References

1. Sergio Cabello and Erin W. Chambers. Multiple source shortest paths in a genus g graph. *Proc. 18th Ann. ACM-SIAM Symp. Discrete Algorithms*, 89–97, 2007.

2. Sergio Cabello, Erin W. Chambers, and Jeff Erickson. Multiple-source shortest paths in embedded graphs. *SIAM J. Comput.* 42(4):1542–1571, 2013. arXiv:1202.0314.

3. David Eisenstat and Philip N. Klein. Linear-time algorithms for maxflow and multiple-source shortest paths in unit-weight planar graphs. *Proc. 45th Ann. ACM Symp. Theory Comput.*, 735–744, 2013.

4. Lester R. Ford. Network flow theory. Paper P-923, The RAND Corporation, Santa Monica, California, August 14, 1956.

5. Richard M. Karp and James B. Orlin. Parametric shortest path algorithms with an application to cyclic staffing. *Discrete Appl. Math.* 3:37–45, 1981.

6. Philip N. Klein. Multiple-source shortest paths in planar graphs. *Proc. 16th Ann. ACM-SIAM Symp. Discrete Algorithms*, 146–155, 2005.

7. Yaowei Long and Seth Pettie. Planar distance oracles with better time-space tradeoffs. *Proc. 32nd Ann. ACM-SIAM Symp. Discrete Algorithms*, 2517–2537, 2021. arXiv:2007.08585.

8. Ketan Mulmuley, Umesh Vazirani, and Vijay Vazirani. Matching is as easy as matrix inversion. *Combinatorica* 7:105–113, 1987.

9. Robert E. Tarjan and Renato F. Werneck. Self-adjusting top trees. *Proc. 16th Ann. ACM-SIAM Symp. Discrete Algorithms*, 813–822, 2005.

13.11 Sir not appearing

- Subtleties for directed graphs and/or graphs with negative edge lengths.
- Post-hoc distance and path queries via persistence
- Klein's leafmost pivot strategy
- Unweighted MSSP in linear time [Eisenstat Klein]
- $\Omega(n \log n)$ lower bound [Eisenstat Klein]
- Space-time tradeoff for Klein's algorithm (but *not* CCE) using different dynamic-forest data structures: $O(kn^{1+1/k})$ preprocessing time and $O(k \log\log n)$ query time. [Long Pettie 2021]