CS 344: Design and Analysis of Computer Algorithms **Rutgers: Spring 2022**

Lecture 20

April 05, 2022

Instructor: Sepehr Assadi

1 Single-Source Shortest Path

Recall from the last lecture that we were considering the **shortest path** problem:

Problem 1. Given a graph $G = (V, E)$ (directed or undirected) with *positive* weight $w_e > 0$ on each edge $e \in E$, and a designated vertex $s \in V$, called a *source*, find the value of the shortest path from s to every vertex $v \in V$, denoted by $dist(s, v)$. In other words, for any path P, we define $w(P) = \sum_{e \in P} w_e$, namely, the total weight of the edges in this path, and our goal is to find the weight of the minimum weight path (minimizing $w(P)$) from s to any other vertex v (thus we are actually looking for minimum weight path but the convention is to still call this shortest path).

We now consider another algorithm for the shortest path problem, namely the Dijkstra's algorithm.

2 Dijkstra's Algorithm

The algorithm is quite similar to Prim's algorithm with a minor yet crucial modification.

1. Let $mark[1:n] = FALSE$ and s be the designated source vertex.
2. Let $d[1:n] = +\infty$ and set $d[s] = 0$.
3. Set $mark[s] = TRUE$ and let S (initially) be the set of edges incident on s and assign a value $value(e) = d[s] + w_e$ to each of these edges.
4. While S is non-empty:
   (a) Let $e = (u, v)$ be the *minimum value* edge in S and remove e from S.
   (b) If $mark[v] = TRUE$ ignore this edge and go to the next iteration of the while-loop.
   (c) Otherwise, set $mark[v] = TRUE$, $d[v] = value(e)$, and insert all edges e' incident on v to S with $value(e') = d[v] + w'_e$.
5. Return d.

It is worth mentioning that the "only difference" between Dijkstra's and Prim's algorithms is in the notion of $value(e)$: in Dijkstra's we set $value(e) = d[v] + w_e$ where v is the endpoint of edge e we already marked while in Prim's algorithm we set $value(e) = w_e$. This has the following effect: Even though both algorithms attempt to "grow" a component of vertices reachable from s, Dijkstra do this by exploring edges that are "closer" to s and adding their endpoints to the component of s while Prim finds the edges that are simply the "lightest" at the moment. See Figure 1 for an illustration.

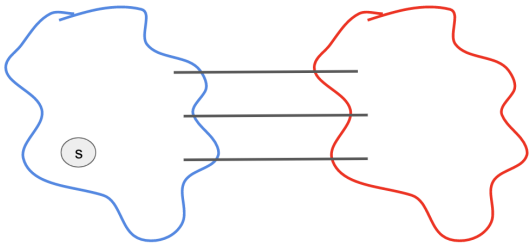

Figure 1: Both Prim's and Dijkstra's algorithms expand a set of vertices reachable from a source further and further by picking the "best" (cut) edge in the cut specified by the vertices currently reachable from source. However, the definition of the "best" edge is different between the two algorithms: Prim's algorithm picks the edge with minimum weight among the cut edges of this cut while Dijkstra's algorithm picks the the edge whose other endpoint would be closest to the source. This is the reason why Prim's algorithm returns an MST while Dijkstra's algorithm returns shortest paths.

Proof of Correctness: The proof of correctness of Dijkstra's algorithm is quite similar to that of BFS and Prim's algorithm. We prove by induction that in every iteration of the while-loop in the algorithm, the set C of all marked vertices (i.e., $C = \{v \mid mark[v] = TRUE\}$) has the following two properties:

- For any $u \in C$ and $v \in V - C$, $dist(s, u) < dist(s, v)$;
- For every $v \in C$, $d[v] = dist(s, v)$, i.e., distances are computed correctly.

The base case of the induction, namely for iteration 0 of the while-loop (i.e., before we even start the while-loop) is true since s is the closest vertex to s (satisfying part one) and $d[s] = dist(s, s) = 0$ satisfying part two. Now suppose this is true for some iteration i of the while-loop and we prove it for iteration $i + 1$. Let $e = (u, v)$ be the edge removed from S in this iteration. If $mark[v] = TRUE$ we simply ignore this edge and hence the set C remains the same after this step and by induction hypothesis, we have the above two properties. Now suppose $mark[v] = FALSE$. In this case:

1. Every edge among the cut edges of $(C, V - C)$ belong to the set S at this point (we have not removed any of those cut edges yet as otherwise C would have contained the other endpoint of that edge as well and we included all those edges when we marked their first endpoint and hence added them to C).
2. We are setting $d[v] = value(e) = d[u] + w_e = dist(s, u) + w_e$ (by the induction hypothesis for u) where e is the edge with minimum $value(e)$ in S and hence minimum $value(e)$ among cut edges of $(C, V - C)$.
3. Since $dist(s, w) > dist(s, u)$ for all $w \in V - C$ by induction hypothesis, we know that the shortest path from s to v does not visit any of the vertices in $V - C$ (otherwise the edge leading to that vertex will have a smaller value than the edge e leading to v). Hence, by setting $d[v] = dist(s, u) + w_e$ where edge e minimizes the right hand side of this equation, we will have that $d[v] = dist(s, v)$. This proves the second part of the induction hypothesis. For the first part also, notice that $value(e)$ is minimized and hence v is the "closest" vertex to s in $V - C$ and hence after adding v to C, $dist(s, w) > dist(s, v)$ for all $w \in V - C$. This proves the second part of the induction hypothesis.

This concludes the proof of correctness of Dijkstra's algorithm as at the end of the last iteration, by induction hypothesis (second part), $d[v] = dist(s, v)$ for all $v \in V$.

Runtime Analysis: Similar to the Prim's algorithm, we can implement the set S with a *min-heap*: this allows us to implement Dijkstra's algorithm in $O(n + m \log m)$ time as well[1].

[1]There is a way to implement Dijkstra's and Prim's algorithms to run in $O(n \log n + m)$ instead of $O(n + m \log m)$ by using *Fibonacci heaps* instead of min-heaps, but Fibonacci heaps are seriously complicated and we ignore them in this course.