MAT344 Lecture 22

2019/Aug/1

1 Announcements

2 This week

This week, we are talking about

1. Minimal spanning trees

3 Recap

Last time we talked about

1. Prim's algorithm

4 Directed Graphs (Ch. 12.3 in [KT17])

If we want to model flows in a network using graphs, we would like to be able to identify a "direction" of the flow. Or, you might want to model streets in a city as a graph, but some of the streets may be one way streets. This is the motivation for *directed graphs*.

Definition 4.1. *A* ***directed graph****, or* ***digraph*** *G is a pair (V, E) where V is a set (the set of vertices of G) and $E \subseteq V \times V$ is a set of* ordered *pairs (x, y) of vertices of G.*

When drawing directed graphs, we will indicate the direction by drawing an arrow on the edge. We can also assign weights to the edges of a directed graph. The weight may also represent distance (or travel time) between vertices. If P is a (directed) path in our digraph, then the **length** of the path is defined to be the sum of the weights of the edges that it contains. If r and x are vertices, then the **distance** from r to x is defined to be the minimal length of a path from r to x.

5 Dijkstra's Algorithm (Ch. 12.4 in [KT17])

In many cases, we want to find the distance between two vertices a and x, or a shortest path. Dijkstra's algorithm will do this for us. The idea behind the algorithm is to start with a and add the vertices (along with the shortest paths to them) one by one.

Let's say we have the graph on Figure 1 and we want to find the shortest path (and distance) from a to all the other vertices. We select a as the starting vertex, and mark it as *visited* (see Figure 2a).

We will have to keep record of the tentative shortest distances to all the vertices in our graph. For a vertex x, we will denote by $\delta(x)$ this tentative distance. Since we start at a, we set $\delta(a) = 0$, and we set $\delta(x) = \infty$ for all other vertices.

The next step is to **scan** from a to see if we can improve one of our distances. So we look at all directed edges from a (see Figure 2b) to see if we can find a shorter distance to a vertex than before.

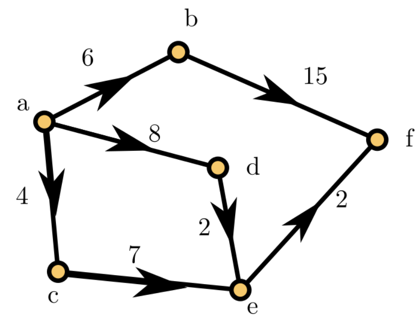
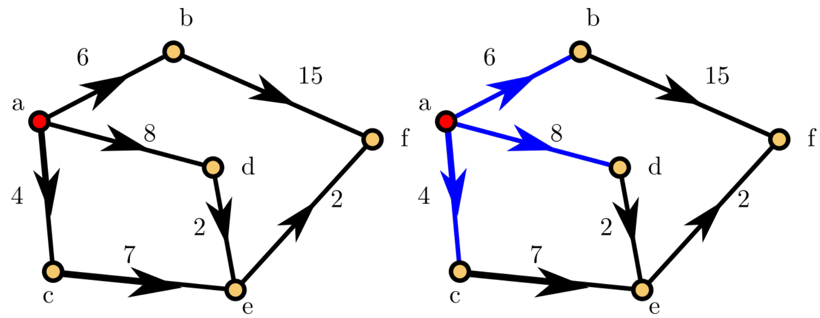


Figure 1: A directed graph



(a) Starting at a

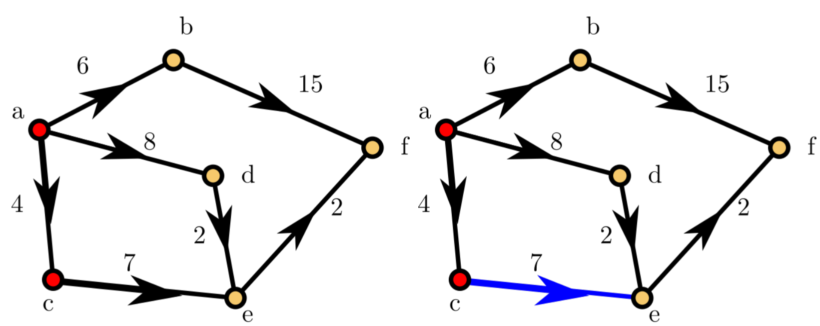
(b) Scanning from a

Figure 2

We update the distances

$$\delta(a) = 0$$
$$\delta(b) = 6$$
$$\delta(c) = 4$$
$$\delta(d) = 8$$
$$\delta(e) = \infty$$
$$\delta(f) = \infty$$

Then we select the vertex with the least distance not already visited (in this case, c), and mark it as visited (see Figure 3a). For a vertex that we add to the visited vertices, we record the shortest path from a (in this case, $P(c) = (a, c)$.



(a) c is now visited

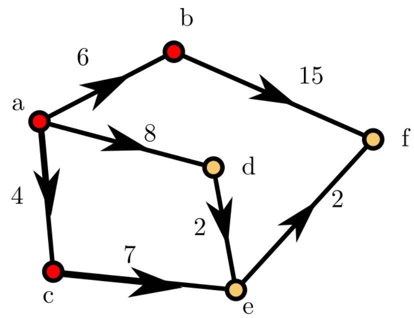
(b) Scanning from c

Figure 3

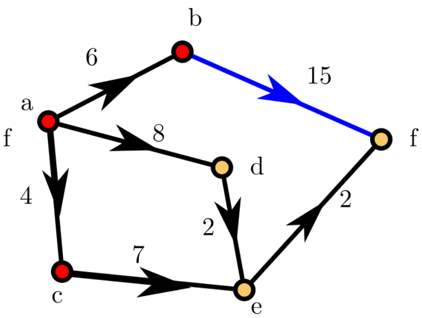
The next step is to scan from the newly added vertex (Figure 3b) to see if we can update one of our distances The vertex e is now reachable, so we record the current shortest distance to it

$$\delta(a) = 0$$
$$\delta(b) = 6$$
$$\delta(c) = 4$$
$$\delta(d) = 8$$
$$\delta(e) = 4 + 7 = 11$$
$$\delta(f) = \infty$$

The vertex with the smallest distance not already added is b, so we add it as visited (Figure 4a). We record $P(b) = (a, b)$.



(a) b is now visited



(b) Scanning from b

Figure 4

We scan from the newly added vertex (b this time, Figure 4b)
We update the distancesn

$$\delta(a) = 0$$
$$\delta(b) = 6$$
$$\delta(c) = 4$$
$$\delta(d) = 8$$
$$\delta(e) = 11$$
$$\delta(f) = 6 + 15 = 21$$

The closest vertex not already visited is d, so we add it ($P(d) = (a, d)$), and scan from it (Figure 5).
We update the distances (note that the distance to e has decreased)

$$\delta(a) = 0$$
$$\delta(b) = 6$$
$$\delta(c) = 4$$
$$\delta(d) = 8$$
$$\delta(e) = 10$$
$$\delta(f) = 21$$

We add e ($P(e) = (a, d, e)$), scan from it

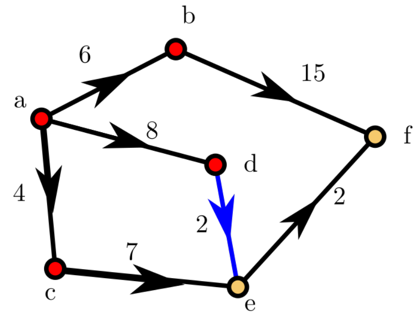


Figure 5: d is now visited, scanning

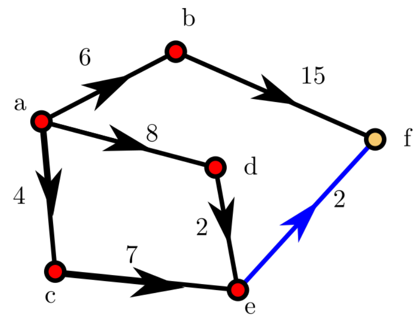


Figure 6: Scanning from e

and after a last update of distances,

$$\begin{aligned}\delta(a) &= 0\\ \delta(b) &= 6\\ \delta(c) &= 4\\ \delta(d) &= 8\\ \delta(e) &= 10\\ \delta(f) &= 10+2=12\end{aligned}$$

we add f to the visited vertices $P(f) = (a, d, e, f)$. We have now found the distance and a shortest path from a to every other vertex. (Figure 7).

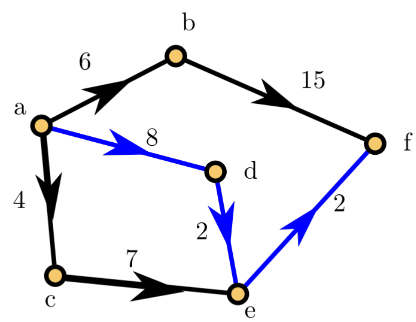


Figure 7: A shortest path from a to f

Theorem 5.1 (Theorem 12.8 in [KT17])**.** *Dijkstra's algorithm finds the distance for every vertex x in G. Moreover, the path $P(x)$ is a shortest path.*

Proof. Let $(v_1, v_2, \dots, v_n)$ be the list of vertices in the order in which they are visited by the algorithm (so, for example, $a = v_1$). Note that this means $\delta(v_1) \le \delta(v_2) \le \dots \le \delta(v_n)$.

We will do induction on the *minumum number of edges in a shortest path* to a vertex. Let $x \ne a$ and let $P(x)$ and $\delta(x)$ be the output of Dijkstra's algorithm for x. The base case is $k = 0$, and this is clearly true for a, as $\delta(a) = 0$ and $P(a) = (a)$ is a shortest path from a to a.

Assume for induction that any vertex whose minimum number of edges in a shortest path is at most k, Dijkstra's algorithm produces the correct distance and shortest path. Let x be a vertex for which the minimum number of edges in a shortest path is $k+1$. Fix a shortest path $Q = (a, u_1, u_2, \dots u_k, x)$. Since this is a shortest path, the path $R = (a, u_1, u_2, \dots u_{k-1}, u_k)$ must also be a shortest path. Therefore by the inductive hypothesis, $\delta(u_k)$ is the distance from a to u_k and $P(u_k)$ is a shortest path from a to u_k (note: $P(u_k)$ need not be $(a, u_1, u_2, \dots, u_k)$.

Also, the distance from a to x is $\delta(u_k) + w(u_k, x) \ge \delta(x)$

Let i, j be the integers for which $u_k = v_i$ and $x = v_j$ (i.e. the step at which u_k and x are added as visited).

- If $j < i$ then
$$\delta(x) = \delta(v_j) \le \delta(v_i) = \delta(u_k) \le \delta(u_k) + w(u_k, x),$$
and by our previous observation, this is the distance from a to x. Therefore the algorithm has found a path that is at most $\delta(u_k) + w(u_k, x)$ long (therefore it must be shortest), and a shortest path $P(x)$ (note: this need not be the same path as Q, but it has the same length).

- If $i < j$ then scanning from $u_k = v_i$ results in
$$\delta(x) \le \delta(v_i) + w(v_i, x) = \delta(u_k) + w(u_k, x).$$
which is the distance from a to x again, so the same argument applies.

Q.E.D.

6 Network flows (Ch. 13 in [KT17])

We will continue looking at weighted directed graphs and study the flows in them. The standard terminology in this area is to refer to the weights as **capacities** and denote them $c(e)$. In most of the examples we will be considering, there will be a **source** vertex s and a **sink, or target** vertex t. All edges incident to the source are oriented away from it and all edges incident to the sink are oriented towards it.

Definition 6.1. *A **flow** in a network is a function f which assigns to each directed edge $e = (x, y)$ a non-negative value $f(e) = f(x, y) \le c(e)$ such that for every vertex which is neither the source nor the sink,*
$$\sum_x f(x, y) = \sum_x f(y, x),$$
*i.e. the amount leaving y is equal to the amount entering y. The **value** or **strength** of a flow is denoted by $|f|$.*

Example 6.2. *The flow in Figure 8 illustrates a flow. There are two numbers on each edge, the first is the capacity, the second one is the value of the flow through that edge.*

Theorem 6.3. *For any flow f in a network G, the total flow out of the source is equal to the total flow into the sink.*

Proof. Consider the sum
$$S = \sum_{x \in V \setminus \{t\}} \left(\sum_{y \in V} f(x, y) - \sum_{y \in V} f(y, x) \right).$$
We can rewrite the summation as
$$S = \sum_{x \in V \setminus \{t\}} \sum_{y \in V} f(x, y) - \sum_{x \in V \setminus \{t\}} \sum_{y \in V} f(y, x)$$

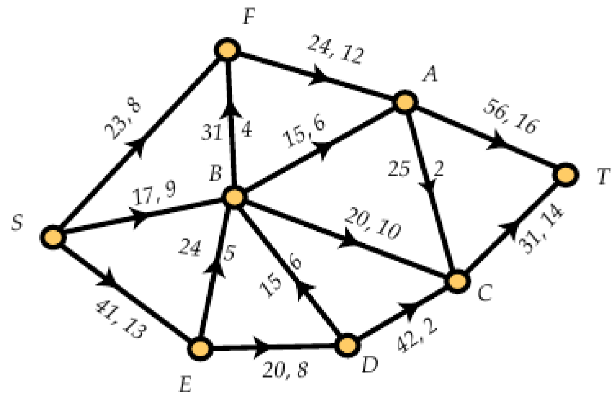


Figure 8: A network flow

every arc $e = (x, y)$ appears in both sums unless $y = t$ (and there are no arcs with $x = t$), so it contributes 0 to the total value. So the entire sum is equal to $\sum_{x \in V \setminus \{t\}} f(x, t)$.

$$\sum_{y \in V} f(s, y) = S = \sum_{y \in V} f(y, t).$$

Q.E.D.

References

[KT17] Mitchel T. Keller and William T. Trotter. *Applied Combinatorics*. Open access, 2017. Available at http://www.rellek.net/appcomb/. 1, 4, 5