I Definitions and problem statement.

A weighted graph is a graph $G$ with real numbers, called weights, assigned to its edges. In the discussion that follows, $w(e)$ will denote the weight of edge $e$ in a weighted graph.

In the following example, the weight of each edge is written next to the edge. The edges represented by the heavier line are for the discussion that ensues.

Let $P$ be a path in a weighted graph. The weight $W(P)$ of $P$ is defined to be the sum total of the weights of the edges in $P$. For example, the weight of the path $a, v_2, v_1, v_8$ in the example is $3 + 3 + 1 = 7$.

In the example, all edge weights are nonnegative, but in the general definition of a weighted graph there is no requirement that they necessarily be
so. However, for the theory we are about to develop, it is important to assume that weights are nonnegative. Henceforth in this lecture, it shall be a standing assumption that all edge weights are nonnegative.

Let \( x \) and \( y \) be two vertices in a connected, weighted graph with nonnegative weights. The (weighted) distance between \( x \) and \( y \) is the smallest weight achieved by an \((x,y)\)-path. This distance will be denoted \( d_W(x,y) \). Thus,
\[
d_W(x,y) = \min \{ W(P) ; P \text{ is an } (x,y)\text{-path} \}
\]
Always, we take \( d_W(x,x) = 0 \); indeed it is useful to think of the single vertex \( x \) as a path of length 0 from \( x \) to \( x \), which, since it contains no edges, has therefore weight 0. We say that an \((x,y)\)-path is a shortest route between \( x \) and \( y \) if it is an \((x,y)\)-path of minimum weight; that is, if \( W(P) = d_W(x,y) \).

Observe that if we take an unweighted graph and assign weight 1 to each edge, the weight of a path is just the number of edges it has, and therefore the weighted distance reduces to the ordinary distance \( d(x,y) \), namely, the number of edges in the shortest path between \( x \) and \( y \). Shortest route paths are then the same thing as geodesics. To illustrate these concepts in the graph above, consider the two vertices \( u \) and \( a \). It is clear by inspection that \( d_W(u,a) = 4 \), and that \( u,b,a \) is a shortest \((u,a)\) route.

Suppose we are given a vertex \( u \) and a vertex \( v \) in a weighted graph \( G \) with nonnegative weights and we want a shortest route between them. An obvious problem of great practical interest is to find the shortest route between \( x \) and \( y \). It turns out that to solve this problem efficiently it is worthwhile to study a more general problem, even though it seems more complex: given a vertex \( u \), find shortest routes between \( u \) and all other vertices in the graph. We shall see in this lecture that one can find a single spanning tree of the graph that supplies the shortest route from \( u \) to each other vertex. Thus, the following definition is useful.

**Definition.** A tree \( T \) that is a subgraph of a weighted graph \( G \) is a shortest route tree for vertex \( u \) if, for every vertex \( v \) in \( T \), the unique \((u,v)\)-path in \( T \) is a shortest route between \( u \) and \( v \).

For an example, we again refer to the weighted graph defined above. We claim that the tree indicated by the thick lines is a shortest route tree for vertex \( u \). This can be seen by inspection. We have already seen that \( d_W(u,a) = 4 \) and the path \( u,b,a \) in the tree is a shortest path from \( u \) to \( a \). Likewise, it is clear that the path consisting of the single edge \( ub \) is a shortest
\((u, b)\)-path. Once can see similarly that \(u, b, c\) and \(u, b, c, d\) are shortest paths from \(u\) to \(c\) and \(d\), respectively.

A trivial, but basic, example of a shortest routes tree from \(u\) is the tree \(T = \{u\}\) consisting of the single vertex \(u\) and no edges; the only path is the null path from \(u\) to itself and it has weight 0, which is the distance from \(u\) to \(u\). Another simple example is the tree consisting of the single edge \(uv\), where \(uv\) is an edge of minimum weight incident on \(u\). To show this is a shortest routes tree we must show that the path \((u, v)\) traversing the single edge \(uv\) is the shortest path between \(u\) and \(v\). Observe that any other path from \(u\) to \(v\) must start with an edge \(ux\) where \(x\) is different from \(v\). Observe also that since all edge weights are nonnegative (here is where the nonnegativity assumption is important) the weight of any path is at least as great as the weight of any one of its edges. By definition of \(v\) as the vertex adjacent to \(u\) such that \(uv\) has minimum weight, \(w(ux) \geq w(uv)\). It follows that any path from \(u\) to \(v\) has weight at least \(w(uv)\) and hence \(uv\) is a shortest path. This general argument, for example, shows that the edge \(ub\) must be the shortest route from \(u\) to \(b\) in the example.

One consequence of Dijkstra’s algorithm to be presented below is the following fact: given a weighted graph \(G\) with nonnegative weights and a vertex \(u\) in \(G\), there is a spanning tree of \(G\) that is a shortest weights tree for \(u\). This is an interesting fact, because it is not immediately obvious that just one tree can supply shortest routes from \(u\) to every other vertex in \(G\). But this indeed is the case. In fact, Dijkstra’s algorithm builds a shortest routes spanning tree from a starting vertex \(u\).

II. Dijkstra’s algorithm

Let \(G\) be a given, connected graph with nonnegative edge weights, and let \(u\) be a vertex in \(G\). Dijkstra’s algorithm is a tree growing algorithm. Suppose we have a subgraph \(T\) of \(G\) that is a shortest routes tree for \(u\). The basic step of the algorithm is to add one edge to \(T\) to make a bigger tree \(T'\) that is again a shortest routes tree for \(u\). Dijkstra’s algorithm is simply to start with the tree \(T = \{u\}\) and keep adding edges using the basic step until a spanning tree is obtained.

We have already seen the main idea of the algorithm in the fact explained above that the edge of minimum weight incident to \(u\) constitutes a shortest routes tree for \(u\) of size 1. We want now to generalize to the situation in which we have already a shortest routes tree \(T\) for \(u\), but in which \(T\) is not
a spanning tree, so it does not contain all the vertices of $G$. Recall that, by
definition of a shortest routes tree, $d_W(u, y)$ is the weight of the unique $(u, y)$
path in $T$, for every vertex $v$ in $T$. As we go along explaining the general
method, we shall illustrate on the weighted graph defined above, taking $T$ to
be the tree defined by the thicker edges. For convenience, here is the graph
again:

![Graph](image)

The ultimate object is to show how to add an edge to $T$ to get a larger
tree that is still a shortest routes tree from $u$. First, we show how to get
to neighbors of $T$ in an optimal way from $T$. Here, by a *neighbor* of $T$ we
mean any vertex in $G − T$ that is adjacent to a vertex in $T$. In the example,
the neighbors of $T$ are the vertices labelled $v_1, v_2, ..., v_8$. A $(u, v)$-path all of
whose edges are in $T$, except for the last edge $xv$ from a neighbor $x$ in $T$ to $v$, is said to *reach $v$ directly from $T$*. For instance, the path $u, b, a, v_5$ reaches
the vertex $v_5$ directly from $T$; so also do the paths $u, b, v_5$ and $u, b, c, d, v_5$.
For each $x$ in $T$ that is a neighbor of $v$, there is a unique path that reaches $v$
directly from $T$ and ends with edge $xv$, because the there is a unique $(u, x)$
path in $T$. The weight of this path is

$$d_W(u, x) + w(xv),$$

because the total weight of the edges from $u$ to $x$ is $d_W(u, x)$, by virtue of the fact that $T$ is a shortest routes tree, and the weight of the edge from $x$ to $v$ is $w(xv)$. In the example, the weight of the path reaching $v_5$ directly from $T$ by way of vertex $a$ is $d_W(u, a) + w(av_5) = 4 + 0 = 4$, the weight of the path that reaches $v_5$ directly from $T$ by way of $b$ is $d_W(u, b) + w(bv_5) = 1 + 4 = 5$, while the weight of the path that reaches $v_5$ from $d$ is $d_W(u, d) + w(dv_5) = 5 + 6 = 11$.

The key to Dijkstra’s algorithm is to look for the shortest paths from $u$ reaching neighbors of $T$ directly from $T$. For each neighbor $v$ of $T$, define $t(v)$ to be the minimum weight of a $(u, v)$ path that reaches $v$ directly from $T$: since each such path reaches $v$ from a vertex in $T$ adjacent to $v$,

$$t(v) = \min \{d_W(u, x) + w(xv); x \in T, x \text{ is adjacent to } v\}.$$

To illustrate in the case of the example, look at the vertex $v_5$, as we have already done some of the necessary calculations:

$$t(v_5) = \min \{d_W(u, x) + w(xv_5); x = a, b, \text{ or } d\} = \min \{4, 5, 11\} = 4.$$

The student should also check that

$$t(v_1) = \min \{d_W(u, x) + w(xv_1); x = u, a, \text{ or } b\} = \min \{4, 7, 5\} = 4;$$

$$t(v_2) = \min \{d_W(u, x) + w(xv_2); x = a\} = d_W(u, a) + w(av_2) = 7;$$

$$t(v_3) = 5, \quad t(v_4) = 5, \quad t(v_6) = 5, \quad t(v_7) = 9, \quad t(v_8) = 7.$$

Here now are the crucial definitions for Dijkstra’s algorithm. Let $v^*$ denote a neighbor of $T$ that minimizes $t(v)$ over all neighbors of $v$. In other words,

$$t(v^*) = \min \{t(v); v \text{ is a neighbor of } T\}.$$

Let $x^*$ be the vertex of departure from $T$ of a path reaching $v^*$ directly from $T$ of weight $t(v^*)$. That is,

$$t(v^*) = d(u, x^*) + w(x^*v^*).$$

To illustrate in the case of Figure, $v^* = v_1$, because $t(v_1) = 4$ is the smallest value among $\{t(v_1), t(v_2), \ldots, t(v_8)\}$ and $x^* = b$, because it is the path reaching $v_1$ by way of $b$ that has weight 4.
The importance of these definitions and the heart of the Dijkstra algorithm is expressed by the following result.

Claim. The tree $T'$ obtained by adding the edge $x^*v^*$ to $T$ is a shortest routes tree.

This claim implies that if the edge $bv_1$ is added to the tree $T$ in our example, the new tree is still a shortest routes tree from $u$. On can see by inspection that this will indeed be the case.

Here is the argument in support of this claim. If $y$ is a vertex of $T$, the path in $T'$ from $u$ to $y$ is contained in $T$ and hence is already a shortest route. Thus we only need to prove that the unique $(u,v^*)$ path in $T'$ is a shortest path. This path consists of the unique $(u,x^*)$ path in $T$ followed by the edge $x^*v^*$ and it has weight $t(v^*)$. By definition of weighted distance

$$t(v^*) \geq d(u,v^*)$$  \hspace{1cm} (1)

To prove the claim we must prove in fact that

$$t(v^*) = d(u,v^*)$$ \hspace{1cm} (2)

Consider any vertex $z$ not in $T$. Consider any path $P$ from $u$ to $z$. At some point this path must leave $T$, so let $v$ be the first vertex in $P$ not in $T$. Then, since edge weights are non-negative, the weight $W(P)$ is at least the weight of that part of the path from $u$ to $v$, that is it is at least $t(v)$. Thus, since $t(v^*)$ is by definition the minimum value of $t(v)$ over all neighbors of $T$,

$$w(P) \geq t(v) \geq t(v^*)$$ \hspace{1cm} (3)

Since (3) is true for all paths from $u$ to $z$ and for all vertices $z$ it follows that

$$d_W(u,z) \geq t(v^*) \quad \text{for all vertices } z \text{ in } G - T.$$

In particular, this applies to the case in which $z = v^*$, and so

$$d_W(u,v^*) \geq t(v^*).$$

But this inequality in conjunction with (1) implies $t(v^*) = d(u,v^*)$, which is what we needed to show.

We are ready for a formal summary of Dijkstra’s algorithm.
Dijkstra’s algorithm

- **Input:** A connected graph with non-negative edge weights and a vertex $u$.
- **Output:** A shortest routes spanning tree of $G$ and the distances $\{d_W(u, v) ; v \in G\}$.
- **Initialization:** Set $T = \{u\}$ and $d_W(u, u) = 0$.
- **Iteration:** As long as $G - T$ contains vertices, compute $t(v)$ for all neighbors of $T$, find $v^*$ and $x^*$ as described above, augment $T$ by adding edge $x^*v^*$, and set $d(u, v^*) = t(v^*)$.

Example. We will apply a few iterations of Dijkstra’s algorithm to the weighted graph of the example above, with starting vertex $u$. (Although the tree $T$ we worked with so far is a shortest routes tree, Dijkstra’s algorithm will not reproduce it at the third iteration.)

In the calculations $T$ will denote the current tree of the algorithm, not the bold edge subgraph defined above.

Initially, $T = \{u\}$. The first iteration: the neighbors of $T$ are $b, c, v_1, v_6, v_7, v_8$ and $t(b) = 1, t(c) = 5, t(v_1) = 5, t(v_6) = 7, t(v_7) = 9, t(v_8) = 7$. Thus $v^* = b$ and $x^* = u$ and we add to $ub$ to the current tree $\{u\}$ and set $d_W(u, b) = 1$.

The second iteration: Now $T$ consists of the vertices $u$ and $b$ and the single edge $ub$. The neighbors of $T$ are now $v_1, a, d, v_5, v_6, v_7, v_8$, and $t(a) = 4, t(c) = 4, t(d) = 6, t(v_1) = 4, t(v_5) = 5, t(v_6) = 7, t(v_7) = 9, t(v_8) = 7$. There is a 3-way tie for minimum here; we can choose $v^*$ to be either $a$, $c$, or $v_1$. Let us choose it to be $a$. Then $x^* = b$, we add edge $ba$ to $T$, and find $d_W(u, a) = 4$.

The third iteration: $T$ now consists of the two edges $ub$ and $ba$ and their vertices. The neighbors of $T$ are then $c, d$ and $v_1, \ldots, v_8$, and $t(c) = 4, t(d) = 6, t(v_1) = 4, t(v_2) = 7, t(v_3) = 5, t(v_4) = 6, t(v_5) = 4, t(v_6) = 7, t(v_7) = 9, t(v_8) = 7$. Again there is a 3-way tie. Keeping with lexicographic ordering, let $v^* = c$, add $bc$ to $T$, and find $d_W(u, c) = 4$.

The third iteration: the neighbors of $T$ are $d$ and $v_1, \ldots, v_8$, and $t(d) = 5, t(v_1) = 4, t(v_2) = 7, t(v_3) = 5, t(v_4) = 6, t(v_5) = 4, t(v_6) = 6, t(v_7) = 9, t(v_8) = 7$. 7
There is a tie between $v_1$ and $v_5$. We choose to add $bv_1$ to the tree and calculate $d_W(u, v_1) = 4$.

The student should continue this example for several steps.

**Remark.** Dijkstra's algorithm is an instance of a general optimization technique called *dynamic programming*, which is of great importance in applied mathematics.