2.3

Definition

A **weighted graph** G is a graph alongside a function $w : E(G) \to \mathbb{R}^+$.

If G is a weighted graph and $H \subseteq G$, then, $w(H) := \sum_{e \in E(H)} w(e)$.

A **minimum weight spanning tree** (or MWST) is a spanning tree T such that $w(T)$ is minimized among all possible spanning trees. In other words, $w(T) \leq w(T')$ $\forall T' \subseteq G$ where T' is a spanning tree.

Kruskal's Algorithm

INPUT Weighted graph G with n vertices

OUTPUT A MWST, T^* if G is connected, otherwise a message "G is not connected"

STEP 1 Create a list of edges, L_E, in order from smallest weight to largest weight. Start T^* with no edges but all vertices of G.

STEP 2 If the number of edges in T^* is strictly less than $n-1$ AND if there are still edges in L_E, examine the first edge in L_E (i.e., the edge with smallest weight), $e = \{a, b\}$

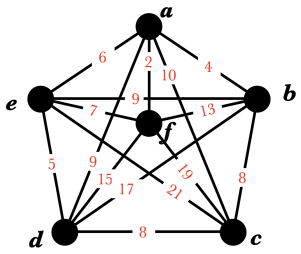
SUBSTEP 2.1 If a and b are in different components of T^*, add e to T^* and remove e from L_E. Return to STEP 2.

SUBSTEP 2.2 If a and b are in the same component, remove e from L_E and do not add to T^*. Return to STEP 2.

STEP 3 If the number of edges in T^* is $n-1$, then output T^*, which is the MWST for G. Otherwise, the number of edges in T^* is strictly less than $n-1$ and G was not connected.

Example

We will find a MWST for the following graph:



First, we create the following table of all the edges in G.

Edge	Cost
af	2
ab	4
de	5
ae	6
ef	7
bc	8
cd	8
ad	9
be	9
ac	10
bf	13
df	15
bd	17
cf	19
ce	21

We can read the following table describing the steps in Kruskal's algorithm from left to right (i.e., we check the edge of lowest weight, then we check the components, then we select our substep).

Edge	Components Of T′	Substep to be used
af	$\{a\},\{b\},\{c\},\{d\},\{e\},\{f\}$	2.1
ab	$\{a,f\},\{b\},\{c\},\{d\},\{e\}$	2.1
de	$\{a,b,f\},\{c\},\{d\},\{e\}$	2.1
ae	$\{a,b,f\},\{c\},\{d,e\}$	2.1
ef	$\{a,b,d,e,f\},\{c\}$	2.2
bc	$\{a,b,d,e,f\},\{c\}$	2.1
—	$\{a,b,c,d,e,f\}$	—

Proof of Kruskal's Algorithm

If G is connected, then Kruskal's Algorithm produces a minimum weight spanning tree.

Let G be a connected graph, and let T_K be the output from Kruskal's algorithm on G. It is easy to check that T_K is a spanning tree, since it is acyclic and has $n(G) - 1$ edges.

Suppose T^* is a MWST with largest edge intersection with T_K — in other words, $|E(T_K) \cap E(T^*)| \geq |E(T_K) \cap E(T')|$ for any other MWST T'.

If $T^* = T_K$, then we are done, since T_K is assumed to be a minimum weight spanning tree. Otherwise, assume toward contradiction that $T^* \neq T_K$. Let e be the first edge chosen by Kruskal's algorithm that is not in T^*. Then, by a previous result, $\exists e' \in E(T^*) - E(T_K)$ such that $T' = T^* + e - e'$ is a spanning tree.

We are assuming, however, that Kruskal's algorithm would choose e over e'. Let $e_1, \ldots, e_k$ be edges in $E(T_K)$ before e. Since e_i was selected before e, we know that $e_i \in E(T^*)$ for each $i \in [k]$.

Let $G_k = (V, \{e_1, \ldots, e_k\})$. we are assuming that Kruskal's algorithm would not choose e' for two reasons:

- If e' shows up before e, then e' would connect two vertices in $G_j = (V, \{e_1, \ldots, e_j\})$. Since $G_j + e' \subseteq T^*$, then T^* contains a cycle and isn't a spanning tree.
- If e' shows up after e in L_E, then $w(e') \geq w(e)$, meaning $w(T') \leq w(T^*)$, meaning that $w(T_K) \leq w(T^*)$, implying that T_K is of a lower weight than T^*.

Dijkstra's Algorithm

We want to find an algorithm to find the shortest path between two vertices in a weighted graph — Dijkstra's Algorithm can solve this.

We know that if P is the shortest u, v path, and $x \in P$, then following P from u to x will yield the shortest u, x path.

Input A weighted graph, G, and $u \in V(G)$

Output For each $z \in V(G)$, the distance $d(u, z)$.

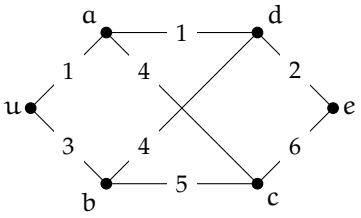
Initialization Extend the weight function such that if $xy \notin E(G)$, then $w(xy) = \infty$. Create S that contains all vertices whose distances from u are known. Let $S := \{u\}$. Let $t : V \to \mathbb{R}^+ \cup \{0, \infty\}$ which will keep track of the tentative distance between u and z. Let $t(z) := w(uz)$ for all $z \neq u$, and $t(u) := 0$.

Condition to Terminate Loop If $t(z) = \infty$ for all $z \notin S$ **or** $S = V$, then go to end.

Loop Else, pick $v \in V - S$ such that $t(v) = \min_{z \notin S} t(z)$. Add v to S. Explore the edges from v to update tentative distances; for each edge vz with $z \notin S$, $t(z) := \min\{t(z), t(v) + w(vz)\}$.

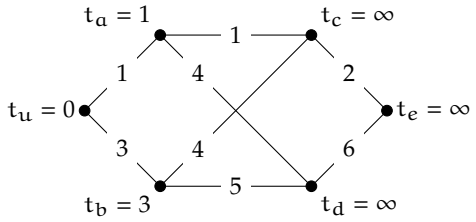
End Set $d(u, v) = t(v)\ \forall v \in V$.

On the following weighted graph, we can do Dijkstra's algorithm as follows:

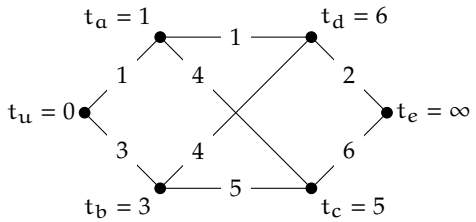


Dijkstra's Algorithm: Worked Example

We let $S = \{u\}$, and include our tentative distances for the first step.

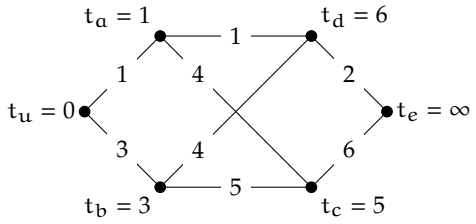


Now, $S = \{u, a\}$ because $t_a \leq t_b$. We now start from a and include our tentative distances.

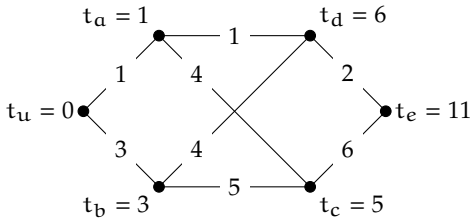


Next, we include b into S, making it $\{u, a, b\}$. We then check our tentative distances from b, where

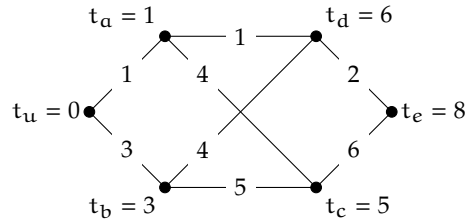
we find that they are no better than tentative distances from a, so we keep the tentative distances on c and d as what they were with a.



Next, we include c into S, making it $\{u, a, b, c\}$, and we update our tentative distances.



Finally, we include d and update tentative distances:



In order to find the direct paths, we can add arrows along the edges that were selected by Dijkstra's algorithm.

The proof can be outlined as follows:

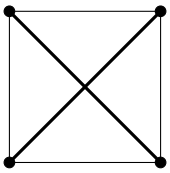
If $z \in S$: then, $t(z) = d(u, z)$.

Else if $z \notin S$: then, $t(z)$ is the length of a shortest u, z path P such that $V(P - z) \subseteq S$.

3.1

Matchings

In a simple graph, a **matching** M is a set of pairwise disjoint edges. In other words, $\forall e_i, e_j \in M$, then $e_i \cap e_j = \emptyset$. In an arbitrary graph, a matching is a set of non-loop edges with no shared endpoints. For example, the thick edges are a matching.



If v is incident to an edge, then v is **saturated** by M, otherwise it is unsatured by M.

A **perfect matching** is a matching that saturates every vertex. We know that all graphs with perfect matchings have even number of vertices, but the alternative case may not be true (for example, we might consider a graph with an isolated vertex).

A **maximal matching** is a matching that cannot be enlarged by adding any other edges. A **maximum** matching is a matching of maximum size among all matchings. In other words, if M^* is a maximum matching, then $|M^*| \geq |M| \; \forall M \in G$.

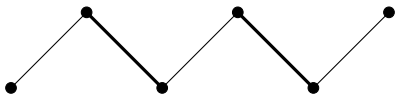
- Not every maximal matching is a maximum matching.
- However, every maximum matching is a maximal matching (because you cannot extend a maximum matching by definition).

Alternating and Augmenting Paths

Given a matching M, an M-**alternating path** is a path that alternates between edges in M and edges not in M. An M-alternating path whose endpoints are unsaturated by M is an M-**augmenting** path.

An M-alternating path:

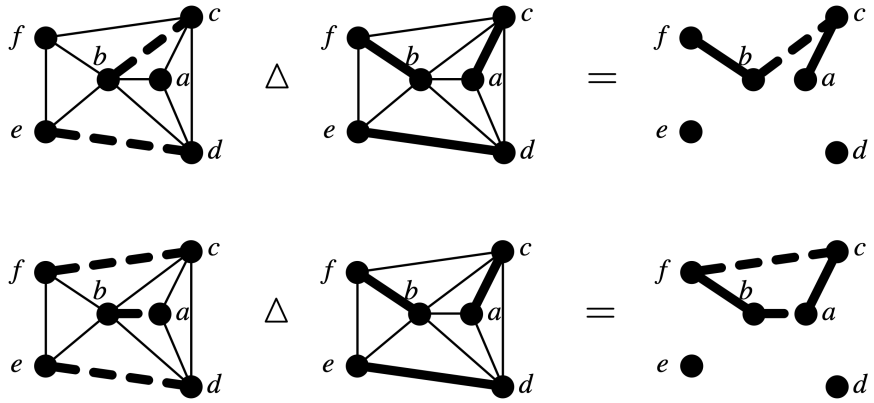
An M-augmenting path:



If M is a matching and there exists an M-augmenting path in G, then M is not a maximum matching (as in the path P that contains M, you can switch the matching edges as follows)

Symmetric Difference

The **symmetric difference** $G \triangle H$ is the subgraph of $G \cup H$ whose edges are the edges of $G \cup H$ that appear in exactly one of G and H. A picture of an example of a symmetric difference between matchings is shown below.



Theorems and Lemmas

Lemma 3.1.9

Every component of the symmetric difference of two matchings is a path or an even cycle.

Let M and M′ be two matchings. Each vertex is incident to at most 1 edge in M and at most one edge in M′. Thus, $d(v) \leq 2$ for each vertex in M and M′. Therefore, each component is either a path or a cycle.

If a component is a cycle, then it must be even because the edges of the cycle must alternate between M and M′.

Theorem 3.1.10

A matching M in a graph G is a *maximum* matching if and only if G has no M-augmenting path.

($\Rightarrow$) Suppose G has an M-augmenting path, P. Exchange the edges of P in M with the edges of P not in M. This action increases the size of the matching by 1, meaning that M was not a maximum matching initially.

($\Leftarrow$) Suppose M is not a maximum matching. Let M′ be a matching with more edges than M (i.e., $|M'| > |M|$). Let $H = (V(G), M \triangle M')$. By Lemma 3.1.9, we know that each component of H is either a path or an even cycle. Since $|M'| > |M|$, there is a component of H that contains more edges from M′ than edges from M, which means it cannot be an even cycle — therefore, this component must be a path that alternates between M′ and M. Because there are more edges from M′ than M in this path, meaning this path is an M-augmenting path.

Perfect Matchings

Let $G = (X, E, Y)$ be a bipartite graph. A matching that saturates X is an X-**perfect matching**.

If there exists a set of vertices $S \subseteq X$ such that $|S| > |N(S)|$, then it is impossible for G to have an

X-perfect matching.

Theorem 3.1.11

A bipartite graph $G = (X, E, Y)$ has an X-perfect matching if and only if $|S| \leq |N(S)|$ for all $S \subseteq X$.

($\Rightarrow$) If G has an X-perfect matching, then each vertex in X is matched to a distinct vertex in $N(S)$. Thus, $|S| \leq |N(S)|$.

($\Leftarrow$)