1 Introduction

We’ve discussed shortest paths and Dijkstra’s Algorithm, which finds the minimal cost tree from a given starting vertex to all other vertices, but these involve choosing an initial vertex.

Suppose now we have a weighted graph $G$ and we wish to find a subgraph of $G$ which is not only a tree but which spans the original graph (includes all vertices), and also has minimal cost, where “minimal” means taken over all possible trees.

Consider the following graph:

**Example 1.1.** Consider this weighted graph:

There are many ways to get a spanning tree. Here are two:

Observe that the spanning tree on the left has a total cost of $20 + 10 + 100 + 100 + 80 + 30 + 100 + 120 = 560$ while the spanning tree on the right has a total cost of $150 + 20 + 10 + 20 + 120 + 10 + 20 + 80 = 420$.

Clearly the one on the right costs less, but could we have done better?
Before proceeding further:

**Theorem 1.0.1.** Every tree with \( n \) vertices has \( n - 1 \) edges.

**Proof.** By structural induction. A single vertex is a tree with \( n = 1 \) vertices and \( n - 1 = 0 \) edges. A new tree is created by adding an edge to an existing vertex and a new vertex at the other end. This contributes 1 to the vertex count and the edge count. \( \square \)

### 2 Prim’s Algorithm

#### 2.1 The Algorithm

Prim’s Algorithm works by growing a tree \( T \subseteq G \). In what follows denote by \( v(H) \) the vertex set of a graph \( H \) and by \( e(H) \) the edge set of a graph \( H \):

1. Define \( T \) to be a graph consisting of one vertex in \( G \). That is, \( v(T) = x \) for any \( x \in v(G) \) and \( e(T) = \emptyset \).

2. While \( v(T) \neq v(G) \) pick a minimal weight edge \( e(x, y) \) with \( x \in v(T) \) and \( y \in v(G) - v(T) \) and add the edge and the vertex to \( T \), so \( v(T) = v(T) + y \) and \( e(T) = e(T) + edge(x, y) \).

**Note 2.1.1.** Note that the condition that the minimum cost edge adds a new vertex to the tree is equivalent to insisting that the minimum cost edge does not form a cycle.
Example 2.1. Consider the example from earlier. Here we have excluded all edges by dashing them. We’ll start with an arbitrary choice of vertex 6.

We now choose a minimum weight edge going between a vertex in \{6\} and a vertex in \{0, 1, 2, 3, 4, 5, 7, 8\}. We choose edge(6, 7) which picks up vertex 7:

We now choose a minimum weight edge going between a vertex in \{6, 7\} and a vertex in \{0, 1, 2, 3, 4, 5, 8\}. We choose edge(7, 8) which picks up vertex 8:
We now choose a minimum weight edge going between a vertex in \( \{6, 7, 8\} \) and a vertex in \( \{0, 1, 2, 3, 4, 5\} \) We choose \( \text{edge}(4, 7) \) which picks up vertex 4:

![Graph 1](image1.png)

We now choose a minimum weight edge going between a vertex in \( \{4, 6, 7, 8\} \) and a vertex in \( \{0, 1, 2, 3, 5\} \) We choose \( \text{edge}(3, 4) \) which picks up vertex 3:

![Graph 2](image2.png)

We now choose a minimum weight edge going between a vertex in \( \{3, 4, 6, 7, 8\} \) and a vertex in \( \{0, 1, 2, 5\} \) We choose \( \text{edge}(0, 3) \) which picks up vertex 0:

![Graph 3](image3.png)
We now choose a minimum weight edge going between a vertex in \{0, 3, 4, 6, 7, 8\} and a vertex in \{1, 2, 5\} We choose edge \((0, 1)\) which picks up vertex 1:

![Graph](image)

We now choose a minimum weight edge going between a vertex in \{0, 1, 3, 4, 6, 7, 8\} and a vertex in \{2, 5\} We choose edge \((1, 2)\) which picks up vertex 2:

![Graph](image)

We now choose a minimum weight edge going between a vertex in \{0, 1, 2, 3, 4, 6, 7, 8\} and a vertex in \{5\} We choose edge \((5, 8)\) which picks up vertex 8:

![Graph](image)

We have now picked up all vertices and we are done. We now have a minimum cost spanning tree. The total cost is 240.
2.2 Mathematics for Prim

**Theorem 2.2.1.** Prim’s Algorithm creates a minimal spanning tree.

**Proof.** Let $G$ be a weighted and connected graph. Since the process of Prim’s Algorithm starts with a node and builds a subgraph by adding edges while keeping the subgraph connected we know that Prim’s Algorithm actually does yield a tree $T \subseteq G$.

Applying Prim’s Algorithm involves adding a sequence of edges $e_1, e_2, \ldots, e_{n-1}$. Suppose we apply Prim’s Algorithm to get a tree $T$ which is not a minimal weight spanning tree. At this point it’s clear that $T$ cannot be extended to create a minimal weight spanning tree but since the single edge $e_1$ can, at some point in the algorithm $e_1, e_2, \ldots, e_{i-1}, e_i$ cannot. Suppose in what follows that $e_i = \text{edge}(x, y)$.

Let $U$ be the set of vertices included when Prim’s Algorithm added $e_1, e_2, \ldots, e_{i-1}$. Observe that $x \in U$ since $e_i = \text{edge}(x, y)$ was next to be added, and Prim’s Algorithm works by adding an edge from a vertex already in the tree (that’s $x$) to a new vertex.

Since the sequence $e_1, e_2, \ldots, e_{i-1}$ can be expanded to a minimal weight spanning tree let’s suppose $M$ is such a tree. Since $M$ is a minimum weight spanning tree there must be a path $P$ from $x$ to $y$ following edges in $M$. However since $y \not\in U$ there is some first edge $\text{edge}(x', y')$ along $P$ which has $x' \in U$ and $y' \not\in U$.

Define the set:

$$M' = M - \{\text{edge}(x', y')\} + \{\text{edge}(x, y)\}$$

Observe that the deletion disconnects the tree $M$ because it removes the only connection from $x'$ to $y'$ and then the addition reconnects it because $x'$ and $y'$ are now connected by following $P$ from $x'$ to $x$, then going to $y$, then following $P$ from $y$ to $y'$.

Consider now:

- If $w(x, y) < w(x', y')$ then the total cost of $M'$ is less than that of $M$ which contradicts the fact that $M$ is a minimal spanning tree.

- If $w(x, y) > w(x', y')$ then Prim’s Algorithm would have chosen $(x', y')$ (which it could have done since $x' \in U$) instead of $(x, y)$ but it didn’t.

- If $w(x, y) = w(x', y')$ then $M'$ is also a minimal spanning tree. However this means that $e_1, e_2, \ldots, e_{i-1}, e_i$ can be extended to a minimal spanning tree (that being $M'$) which contradicts the fact that $e_1, \ldots, e_{i-1}$ was assumed to be the longest string of edges for which this was possible.

In all cases we have contradictions and we are done.

$\Box$
2.3 Pseudocode and Time Complexity

Loosely speaking, the pseudocode is easy:

```
\PRE: G is a graph
T = graph consisting of an arbitrary vertex in G
while vertices in T != vertices in G
    select (u,v) = min cost edge (u,v) with u in T and v in G-T
    T = T + (u,v)
end
\POST: T is a minimal spanning tree
```

The devil is in the details, however. We can represent a graph with an adjacency matrix or with an adjacency list or with other data structures that we have not discussed. These choices will determine the time complexity of the pseudocode.

If $V$ is the number of vertices in the graph and if $E$ is the number of edges in the graph, then here are some options:

- If an adjacency matrix is used, then the `while` loop iterates $V$ times and nested within that we are finding a minimum cost edge involves selecting from $V$ items in $AM$. In this case Prim’s Algorithm has time complexity $O(V^2)$.

- If we use a min-heap to organize the edges, then although the `while` loop iterates $V$ times, edge selection is $\lg E$ as this is the process of taking the root node off the min-heap and fixing the heap. In this case Prim’s Algorithm has time complexity $O(V \lg E)$.

- We’re venturing away from the pseudocode here but if we use a min heap to value the vertices by their current distance from the growing tree. We then update their values (at $\lg V$ time) as the edges are added. This approach can then be shown to be $O(E \lg V)$.

- If a Fibonacci heap is substituted in the previous method we can obtain $O(E + V \lg V)$. 
3 Kruskal’s Algorithm

3.1 The Algorithm

Kruskal’s Algorithm works as follows:

1. For now, exclude all edges.

2. Pick an excluded edge of minimum cost. If including it does not form a cycle then include it.

3. Repeat step 2 until we span the original graph.

Note 3.1.1. We will eventually span the original graph because the graph spans itself. Moreover we can do this without adding a cycle because cycles are not necessary to achieve spanning.
**Example 3.1.** Consider the example from earlier. Here we have excluded all edges by dashing them:

Since there are 9 vertices we’ll need 8 edges so this will be an 8-step procedure. We add edge 0 — 1 which does not create a cycle:

We add edge 6 — 7 which does not create a cycle:
We add edge $0 - 3$ which does not create a cycle:

![Diagram](image1)

We add edge $1 - 2$ which does not create a cycle:

![Diagram](image2)

We add edge $7 - 8$ which does not create a cycle:

![Diagram](image3)

We add edge $3 - 4$ which does not create a cycle:
We add edge 4 − 7 which does not create a cycle:

The next minimum cost edge is 4 − 8 but adding it creates the cycle 4 − 8 − 7 so we skip over it. and instead we add edge 5 − 8 which does not create a cycle:

We now have a minimum cost spanning tree. The total cost is 240.
3.2 Mathematics for Kruskal’s Algorithm

Theorem 3.2.1. Kruskal’s Algorithm creates a minimal spanning tree.

Proof. The fact that we actually obtain a spanning tree follows from the fact that we proceed until we span the graph and from the fact that we do not add an edge if it would create a cycle, therefore we get a tree.

We now claim that at each iteration of the algorithm that if $S$ is the set of edges we have included then $S$ is contained in a minimal spanning tree.

If this is true then when we have obtained a spanning tree at the end then this spanning tree is contained in a minimal spanning tree but since the only spanning tree it it contained in is itself, it must itself be a minimal spanning tree.

We proceed by structural induction, meaning we show that at the start $S$ is contained in a minimal spanning tree and that if at any iteration this is true then it is still true after Kruskal adds another edge.

The statement is obviously true at the start because no edges are included and hence any minimal spanning tree will work.

Suppose the statement is true at some stage and let $T \supseteq S$ be the corresponding minimal spanning tree. Kruskal’s Algorithm adds some edge, $S + \{e\}$.

- If $e \in T$ then we are done since $T$ works for $S + \{e\}$ as well.
- If $e \notin T$ then $T + \{e\}$ contains a set of edges which form a cycle (because adding an edge to a minimal spanning tree forces a cycle). Since $S + \{e\}$ does not contain a cycle (due to how Kruskal works) there is some edge $f$ in that cycle in $T + \{e\}$ which is not in $S + \{e\}$. It follows that $f \in T$ and $f \notin S$ and since the algorithm chose $e$ and not $f$ we must have $w(f) \geq w(e)$. Then $T + \{e\} - \{f\}$ is also a spanning tree (we took a spanning tree, added an edge to create a cycle and then removed an edge within that cycle) and $w(T + \{e\} - \{f\}) \leq w(T)$. However $w(T + \{e\} - \{f\}) \neq w(T)$ since $T$ is a minimal spanning tree and so $w(T + \{e\} - \{f\}) = w(T)$ and so $T + \{e\} - \{f\}$ is a minimal spanning tree containing $S + \{e\}$ as desired.

QED
3.3 Graph Data, Pseudocode, Time Complexity

Loosely speaking the pseudocode is easy:

```plaintext
\< PRE: G is a graph
T = empty graph
while vertices in T != vertices in G
  (u,v) = minimum cost edge in G-T
  if T + (u,v) does not contain a cycle
    T = T + (u,v)
  end
end
\< POST: T is a minimal spanning tree
```

Of course as before the devil is in the details. Keeping track of the edges in G-T is straightforward and selecting a minimum cost edge is not hard (this could be done with a simple linear search or with a binary heap) the significant challenge here is determining when \( T + (u,v) \) does not contain a cycle.

- Fairly simple data structures can be used to achieve this and such an implementation can achieve a time complexity of \( O(E \lg E) \). This is equivalent to \( O(E \lg V) \) because \( E \lg E = O(E \lg V^2) = O(2E \lg V) = O(E \lg V) \) and \( E \lg V = O(E \lg (2E)) = O(E \lg E) \).

- A more technical approach here is to use a disjoint-set data structure which is an data structure that handles sets well, and quickly, and allows such calculations with low time complexity. This is beyond the scope of this course, however such an implementation of Kruskal’s Algorithm can be shown to run in \( O(E \alpha(V)) \) time, where \( \alpha \) is the inverse of the single-valued Ackermann function. Without diving too deeply into details this function \( \alpha \) is “almost constant”.
4 Prim v Kruskal

There are a few considerations that come into play when choosing between Prim and Kruskal.

1. If Prim ends early then we still have a tree.

2. Kruskal’s selection process allows us to be much more flexible if we wish to interact and make specific choices of edges. This is because we can choose any minimal weight edge which does not form a cycle, whereas in Prim we are restricted to growing the tree.

3. Prim allows us to select a starting vertex which gives us a different sense of control, especially when combined with the first point.

4. Prim tends to run faster in dense graphs (lots of edges). This is because even though there are more edges, Prim requires us to grow a tree which restricts the edges we can choose from. Especially earlier on in the algorithm Prim will have far fewer choices even in a very dense graph.

5. Kruskal tends to run faster in sparse graphs (few edges). This is because it’s fairly quick to choose from a sparse set of edges.

6. Kruskal requires us to do cycle detection and this in and of itself can be challenging, or at least obscure.
5 Thoughts, Problems, Ideas

1. In Prim’s Algorithm we need to check if our minimum-cost edge joins a
vertex in $T$ to a vertex in $G - T$. This is equivalent to determining if
the edge creates a cycle. However in Kruskal’s Algorithm these are not
equivalent. Explain

2. In the Graphs chapter exercises you need to write the pseudocode to detect
if a graph contains a cycle. Integrate that pseudocode into Kruskal’s
Algorithm and discuss the time complexity.

3. (a) If $G$ has 5 nodes and 5 edges with weights $\{1, 2, 3, 4, 5\}$ and you con-
struct a Kruskal minimal weight spanning tree, what are the mini-
mum and maximum total possible weights? Explain and draw exam-
ple graphs. Do not do this by attempting to list all graphs!

(b) Repeat the previous question for 6 nodes and 6 edges with weights
$\{1, 2, 3, 4, 5, 6\}$.

4. ...more to come...