FIT2004: Supplementary notes for lecture 10

Graphs: Minimum Spanning Trees

DANIEL ANDERSON [1]

The minimum spanning tree problem is informally, the problem of finding the cheapest way to connect a set of nodes in a network. Minimum spanning trees arise in applications such as connecting cables in a network, where several locations need to be connected together at a minimum possible cost. We will see two algorithms for solving the minimum spanning tree problem.

Summary: Minimum Spanning Trees

In this lecture, we cover:

- Spanning trees, weighted spanning trees and minimum spanning trees
- Prim's Algorithm for minimum spanning trees
- Kruskal's Algorithm for minimum spanning trees
- Correctness of the greedy strategy for minimum spanning trees

Recommended Resources: Minimum Spanning Trees

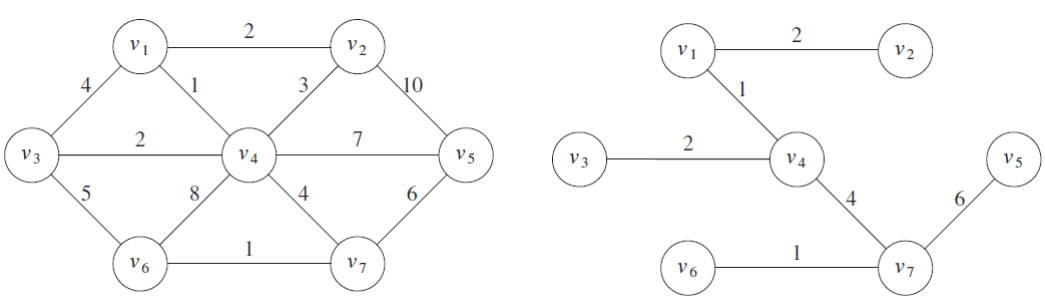
- CLRS, Introduction to Algorithms, Chapter 23
- Weiss, Data Structures and Algorithm Analysis, Section 9.5
- `https://visualgo.net/en/mst` - Visualisation of minimum spanning trees
- `https://youtu.be/tKwnms5iRBU` - MIT 6.046 lecture on minimum spanning trees

Minimum Spanning Trees

A spanning tree of a graph $G = (V, E)$ is a subgraph T consisting of all of the same vertices V and some subset $E' \subset E$ of the edges such that T is a tree. Since spanning trees are trees, they are acyclic (they do not contain cycles.) A spanning tree of an edge-weighted graph is called a weighted spanning tree, and a weighted spanning tree with minimum possible total weight is called a minimum spanning tree. In other words, a minimum spanning tree is a connected subgraph T that minimises the total weight

$$w(T) = \sum_{(u,v)\in T} w(u,v).$$

Although the definition of a minimum spanning tree is a global one, that is we are required to minimise the total weight of the tree, minimum spanning trees have local structure which allows us to find them using greedy algorithms. We will see two such greedy algorithms, Prim's algorithm which is almost identical to Dijkstra's algorithm, and Kruskal's algorithm which makes use of the Union-Find disjoint-sets data structure.



A weighted graph G and a minimum spanning tree of G. Image source: Weiss

[1] FACULTY OF INFORMATION TECHNOLOGY, MONASH UNIVERSITY: `daniel.anderson@monash.edu`. These notes are based on lecture slides by Arun Konagurthu, the textbook by CLRS, some video lectures from MIT OpenCourseware, and many discussions with students.

For both algorithms presented below, we assume that the graph G is connected.

Prim's Algorithm

Prim's algorithm for minimum spanning trees is very similar to Dijkstra's algorithm for single-source shortest paths. Prim's algorithms begins with an arbitrarily chosen source vertex and builds a tree T, adding edges one at a time to to form a spanning tree. The edge to add at each iteration is chosen such that it is a lightest weight edge that connects the current partial spanning tree T to some new vertex v not yet in the tree. To contrast, while Dijkstra's algorithm forms a shortest path tree by selecting a vertex with the smallest total distance from the source, Prim's selects a vertex with minimum possible distance to T.

Algorithm: Prim's Algorithm

```
1:  function PRIM(G = (V[1..N], E[1..M], r))
2:      Set dist[1..N] = ∞
3:      Set pred[1..N] = 0
4:      Set dist[r] = 0
5:      Set Q = priority_queue(V[1..N], key(v) = dist[v])
6:      while Q is not empty do
7:          u = Q.pop_min()
8:          for each edge e = (u,v) adjacent to u do
9:              if v ∈ Q and dist[v] w(u,v)< then
10:                 dist[v] = w(u,v)
11:                 pred[v] = u
12:             end if
13:         end for
14:     end while
15:     return dist, pred
16: end function
```

We may choose the root node `r` to be any node in the graph. Compare the code for Prim's to that of Dijkstra's algorithm. The only difference is that the key used to greedily select the next vertex `v` is taken to be the lightest edge weight $w(u,v)$ connecting `v` to the removed set, rather than the total distance to the source node. Like Dijkstra's algorithm, the complexity of Prim's algorithm using a binary heap to implement the priority queue is $O(|E|\log(|V|))$.

Kruskal's Algorithm

Kruskal's algorithm like Prim's also greedily selects edges, but no longer necessarily builds a single tree T at a time, but instead maintains a forest F of minimum weight subtrees. At each iteration, Kruskal's algorithm selects the lightest weight edge that connects two currently disconnected vertices. In order words, Kruskal's selects the minimum weight edge that does not induce a cycle in the current forest. In order to quickly check whether or not an edge will connect two currently disconnected vertices, we utilise the Union-Find disjoint sets data structure.

Algorithm: Kruskal's Algorithm

```
1:  function KRUSKAL(G = (V[1..N], E[1..M]))
2:      sort(E, key(u,v) = w(u,v))
3:      UnionFind.initialise(N)
4:      Set MST = (V, ∅)
5:      for each edge (u,v) ∈ E[1..M] do
6:          if UnionFind.find(u) ≠ UnionFind.find(v) then
7:              UnionFind.union(u, v)
8:              MST.add_edge(u, v)
9:          end if
10:     end for
11:     return MST
12: end function
```

Sorting the edges of the graph takes $O(|E|\log(|E|))$ time, and the sequence of `UNION` and `FIND` operations takes $O(|E|\alpha(|V|))$. The total time complexity of Kruskal's algorithm is therefore $O(|E|\log(|E|))$, or equivalently $O(|E|\log(|V|))$ since $\log(E) = O(\log(|V|^2)) = O(|\log(|V|))$.

Correctness of the Greedy Strategy

Prim's and Kruskal's algorithms both employ a greedy strategy to select new edges. The correctness of such a greedy strategy is not immediately obvious since the definition of a minimum spanning tree is a global one, not a local one. To prove the correctness of this approach, let's introduce some new definitions and ideas.

- **Cut:** A cut is a partition of the vertex set V into two parts, C and $C \setminus V$.
- **Cross:** An edge $e = (u, v)$ crosses the cut $(C, V \setminus C)$ if the vertices u and v are in different parts, that is either $u \in C$ and $v \notin C$ or $v \in C$ and $u \notin C$.
- **Respect:** A cut respects a set of edges M if no edge in M crosses the cut.
- **Light edge:** An edge is a light edge crossing a cut if its weight is minimum of any edge crossing the cut.

Given some subgraph $M = (V, A \subset E)$ of a minimum spanning tree of G, an edge e is called *safe* if $(V, A \cup \{e\})$ is a subgraph of a minimum spanning tree. The goal of both Prim's and Kruskal's algorithms is to slowly build a minimum spanning tree by maintaining a growing subgraph M which is always a subgraph of some minimum spanning tree. The primary question to answer is therefore how to find safe edges. The following theorems provide the answer and consequently demonstrate the correctness of Prim's and Kruskal's algorithms.

Theorem: Growing a Tree

Let $G = (V, E)$ be a connected, undirected, edge-weighted graph. Let $M \subset E$ be some subset of edges of some minimum spanning tree of G. Let $(C, V \setminus C)$ be any cut of G that respects M. Then, any light edge crossing the cut $(C, V \setminus C)$ is safe for M.

Proof

M is a subset of some minimum spanning tree, let us denote that spanning tree by T. Let $e = (u, v)$ denote a light edge crossing the cut. If T contains e then e must be safe. Suppose then that T does not contain e.

Since T is a tree, there is a path p from $u \rightsquigarrow v$. Since u and v are on opposite sides of the cut, the path p must cross the cut too. Let (x, y) be some such edge on the path p that crosses the cut. The edge (x, y) is not in M since the cut respects M.

Since T is a tree, removing the edge (x, y) disconnects T into two components T_1 and T_2. Connecting the edge (u, v) to T_1 and T_2 reconnects them to form a new spanning tree $T' = T_1 \cup T_2 \cup \{(u, v)\} = T \cup \{(u, v)\} \setminus \{(x, y)\}$. Since (u, v) is a light edge, it has minimum weight of all edges crossing the cut, so in particular it is at least as light as (x, y), in orther words $w(u, v) \leq w(x, y)$. So the total weight of T' is

$$\begin{aligned} w(T') &= w(T) = w(x, y) + w(u, v) \\ &\leq w(T) \end{aligned}$$

But since T is a minimum spanning tree and T' is some other spanning tree, we have $w(T) \leq w(T')$, and hence T' is also a minimum spanning tree.

Therefore, since $M \cup \{(u, v)\}$ is a subset of T' which is a minimum spanning tree, $e = (u, v)$ is a safe edge for M.

Taking our initial cut to be $C = \{r\}$ and growing the contents of the cut with each selected edge, this theorem shows that Prim's algorithm always selects a safe edge and therefore correctly produces a minimum spanning tree.

Theorem: Growing a Forest

Let $G = (V, E)$ be a connected, undirected, edge-weighted graph. Let $M \subset E$ be some subset of edges of some minimum spanning tree of G and let C be some connected component in the forest induced by M on V. If $e = (u, v)$ is a light edge connecting C to another connected component, then e is a safe edge for M.

Proof

Define a cut by the partition $(V_C, V \setminus V_C)$. Since C is a connected component, the cut respects the edge set M. Since e is a light edge connecting C to another component, it is a light edge for the cut. By the first theorem, e is safe for M.

This theorem demonstrates that Kruskal's algorithm always selects a safe edge, and hence also correctly produces a minimum spanning tree.