

Chapter 5 : Minimum Spanning Tree

Omar Saadi
omar.saadi@um6p.ma

UM6P - College of Computing

1st year of the engineering degree of UM6P-CS
Academic year 2024-2025

Outline

- 1 Introduction
- 2 Kruskal's Algorithm
- 3 Prim's Algorithm

Outline

- 1 Introduction
- 2 Kruskal's Algorithm
- 3 Prim's Algorithm

Introduction

Definition

A **weighted graph** is a graph with numerical labels on the edges. For a weighted connected graph, a **minimal spanning tree** is a spanning tree that has the smallest (total) weight possible.

Example : Consider a graph such that :

- Each vertex is a city
- When two cities can be connected by a road, then there is an edge between them in the graph, and its weight is the cost of constructing that road.

Here, connecting all the cities with the smallest total construction cost is exactly the minimum spanning tree problem.

In the following the weights are allowed to be positive or negative.

Outline

- 1 Introduction
- 2 **Kruskal's Algorithm**
- 3 Prim's Algorithm

Kruskal's Algorithm

Input : A weighted connected graph.

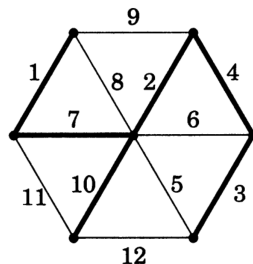
Idea : Maintain an acyclic spanning subgraph H , enlarging it by edges with low weight to form a spanning tree.

Initialization : Set $V(H) = V(G)$ and $E(H) = \emptyset$.

Iteration : If the next cheapest edge joins two components of H , then include it in H ; otherwise, discard it. Terminate when H is connected.

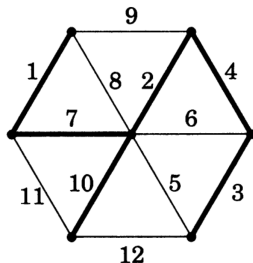
Example

The following example shows a weighted connected graph with the spanning tree given by Kruskal's algorithm. The edges are added in the following order : 1, 2, 3, 4, 7, 10.



Example

The following example shows a weighted connected graph with the spanning tree given by Kruskal's algorithm. The edges are added in the following order : 1, 2, 3, 4, 7, 10.



Remark

Unsophisticated locally optimal heuristics are called **greedy algorithms**. They usually don't guarantee optimal solutions, but Kruskal's algorithm is an example of a greedy algorithm that gives an optimal solution (see next theorem).

Theorem (Kruskal [1956])

In a connected weighted graph G , Kruskal's algorithm constructs a minimum-weight spanning tree.

Proof

First we show that the algorithm produces a tree. Indeed, at each iteration, while H is not yet connected, there is at least an edge that connects two different components of H (because G is connected), so at each iteration an edge is added to H . It becomes connected exactly after adding $n - 1$ edges, and so it becomes a tree.

Let T be the resulting tree, and let T' be a spanning tree of minimum weight. If $T = T'$, we are done. If $T \neq T'$, let e be the first edge chosen for T that is not in T' . Adding e to T' creates one cycle C . Since T has no cycle, C has an edge $e' \notin E(T)$. Consider the spanning tree $T'' = T' + e - e'$.

Proof (Cont.)

Since T' contains e' and all the edges of T chosen before e , then e' could be added to T when the algorithm choosed e (without creating a cycle). So both e' and e were available when the algorithm choosed e , and hence $w(e) \leq w(e')$. Thus $T'' = T' + e - e'$ is a spanning tree with weight that doesn't exceed the weight of T' and that agrees with T for a longer initial list of edges than T' does. In particular T'' is also a minimum spanning tree.

Repeating this argument yields a minimum-weight spanning tree that agrees completely with T . Therefore T is a minimum spanning tree.

Let $n = |V(G)|$ and $m = |E(G)|$.

Implementation steps :

- First sort the m edge weights.
- Maintain for each vertex the label of the component containing it.
- Add the next cheapest edge if its endpoints have different labels, and merge the two components by changing the label of each vertex in the smaller component to the label of the larger one.

Complexity :

- Since the size of the component at least doubles when a label changes (we are changing the smaller one), each label changes at most $\log_2(n)$ times, and the total number of changes is at most $n \log_2(n)$.
- With this labeling method, the running time for large graphs depends on the time to sort m weights of the edges, which is $O(m \log(m))$.

Outline

- 1 Introduction
- 2 Kruskal's Algorithm
- 3 Prim's Algorithm**

Input : A weighted connected graph.

Idea : Growing a subgraph H from one isolated vertex by iteratively adding the cheapest edge from a vertex already reached to a vertex not yet reached to form a spanning tree.

Initialization : $V(H) = \{v\}$ and $E(H) = \emptyset$ for a given starting vertex v .

Iteration : Add to H the cheapest edge (with its other endpoint) from a vertex already in H to a vertex outside H . Terminate when H is spanning.

Input : A weighted connected graph.

Idea : Growing a subgraph H from one isolated vertex by iteratively adding the cheapest edge from a vertex already reached to a vertex not yet reached to form a spanning tree.

Initialization : $V(H) = \{v\}$ and $E(H) = \emptyset$ for a given starting vertex v .

Iteration : Add to H the cheapest edge (with its other endpoint) from a vertex already in H to a vertex outside H . Terminate when H is spanning.

Example : In the above example, if we start from the vertex in the bottom left, we obtain the same spanning tree in the following order of edges : 10, 2, 4, 3, 7, 1.

Theorem (Jarnik [1930], Prim [1957])

In a connected weighted graph G , Prim's algorithm constructs a minimum-weight spanning tree.

Proof

First we show that the algorithm produces a spanning tree. Indeed, at each iteration, we are keeping H connected, and we are adding one edge that does not create any cycle. So H is always a tree. After exactly $n - 1$ iterations, it becomes a spanning tree.

Using the same idea of proof as Kruskal's algorithm :

Let T be the resulting tree from Prim's algorithm, and let T' be a spanning tree of minimum weight. If $T = T'$, we are done. If $T \neq T'$, let e be the first edge chosen (at iteration i) for T that is not in T' . At iteration $i - 1$, we denote the set obtain by $\{v_0, \dots, v_{i-1}\}$. the edge e connects $\{v_0, \dots, v_{i-1}\}$ to $V(G) \setminus \{v_0, \dots, v_{i-1}\}$.

Proof (Cont.)

Now, adding e to T' creates one cycle C . So, $C - e$ is a path from a vertex in $\{v_0, \dots, v_{i-1}\}$ to a vertex in $V(G) \setminus \{v_0, \dots, v_{i-1}\}$. Therefore, $C - e$ contains an edge e' that connects $\{v_0, \dots, v_{i-1}\}$ to $V(G) \setminus \{v_0, \dots, v_{i-1}\}$. Then e' could be added to T when the algorithm chose e (at iteration i), and hence $w(e) \leq w(e')$.

Thus $T'' = T' + e - e'$ is a spanning tree with weight that doesn't exceed the weight of T' and that agrees with T for a longer initial list of edges than T' does. In particular T'' is also a minimum spanning tree.

Repeating this argument yields a minimum-weight spanning tree that agrees completely with T . Therefore T is a minimum spanning tree.

Implementation steps and complexity :

Using the weights matrix $w(u, v)$ giving the weight of the edge (u, v) (with $w(u, v) = +\infty$ if $(u, v) \notin E(H)$).

Idea :

Construct and update a first list $d[v], v \notin H$, with $d[v]$ being the cheapest weight of an edge that connects v to some vertex u already in H .

Also consider a second list $p[v], v \notin H$, with $p[v] = u$ being the neighbor in H to which v is connected with the cheapest weight $d[v]$. The steps and the corresponding complexity are as follows (next slide) :

- ① Start with $H = v_0$ (any initial vertex v_0). [This takes $O(1)$]
- ② For all $v \neq v_0$, initialize $d[v] = w(v_0, v)$, and $p[v] = v_0$ if $(v, v_0) \in E(H)$ and $p[v] = \emptyset$ otherwise. [This takes $O(n)$]
- ③ While H is still not spanning do the following :
 - Extract the cheapest weight in d corresponding to some vertex v_i .
Remove $d[v_i]$ and $p[v_i]$ from d and p . Add the vertex v_i and the edge $(p[v_i], v_i)$ to H . [this takes $O(n)$].
 - Update d and p :
For all neighbours v of v_i such that $v \notin H$, do :
If $w(v, v_i) < d[v]$, then $d[v] = w(v, v_i)$ and $p[v] = v_i$.
[this takes $O(\deg(v_i)) = O(n)$].

Complexity : Combining the above complexities, we get that the total time complexity is $O(n^2)$.

Remark : This complexity can be improved to $O(m \log(n))$ using other data structures to represent the graph (where $m = |E(G)|$).