CSE331 Introduction to Algorithms
Lecture 23
Minimum Spanning Trees

Antoine Vigneron
antoine@unist.ac.kr

Ulsan National Institute of Science and Technology

July 23, 2021

Introduction

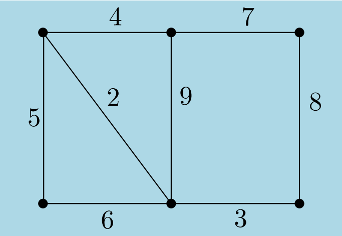
- This lecture presents Prim's algorithm for computing a minimum spanning tree.
- It is a *greedy* algorithm.
- **Reference**: Chapter 23 of the textbook

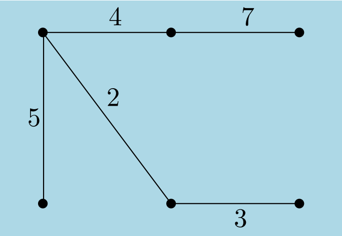
  Introduction to Algorithms by Cormen, Leiserson, Rivest and Stein.

- I will not be following this textbook closely in this lecture.

Introduction

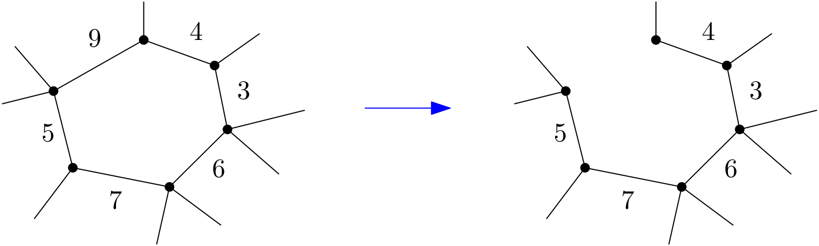
- Suppose that you want to build a communication network between locations $v_1, v_2, \ldots, v_n$.
- Some pairs v_i, v_j can be connected by a direct link with cost $w(v_i, v_j) > 0$.
- The network should be connected, and be as cheap as possible.
- Example:





Introduction

- The network has to be a tree, because if it contained a cycle, we could remove the heaviest edge on this cycle, and the network would remain connected.



Proposition

An undirected graph is a tree iff it is connected and has no cycle.

Problem Statement

Definition

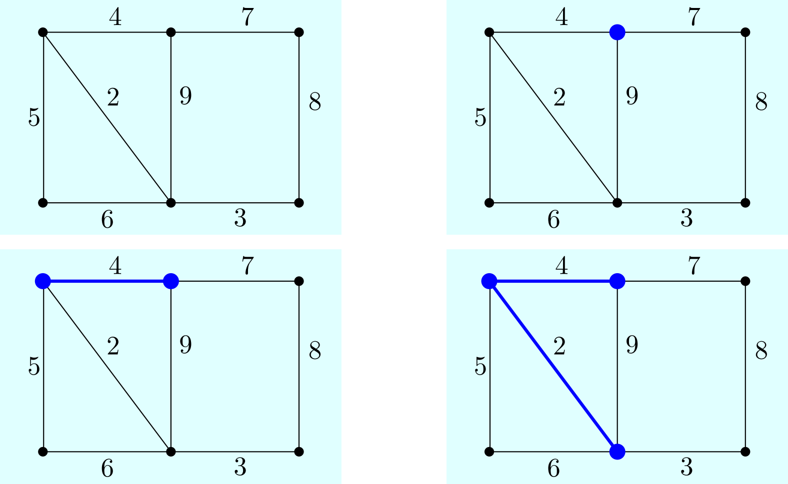
A *spanning tree* of a graph $G(V, E)$ is a tree whose edges are in E, and that connects all the vertices in V.

Problem (Minimum Spanning Tree)

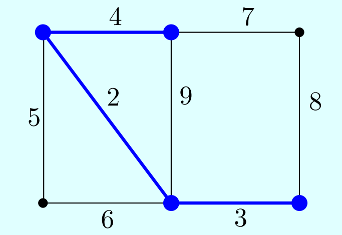
INPUT: a connected, undirected graph $G(V, E)$ with weighted edges.
OUTPUT: a spanning tree T with minimum weight, called the *minimum spanning tree*.

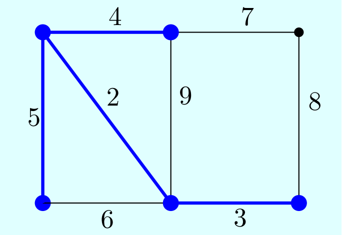
- The brute force approach runs in exponential time as there is an exponential number of spanning trees.
- We will give a polynomial-time algorithm.

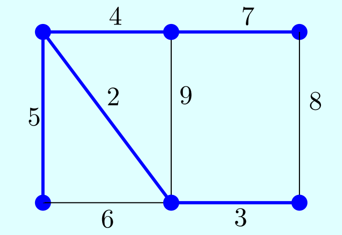
Prim's Algorithm



Prim's Algorithm





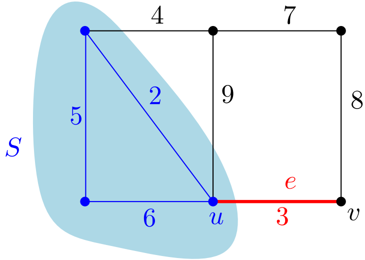


Prim's Algorithm

Pseudocode

procedure $\textsc{Prim}(G(V,E))$
  $S \leftarrow \{r\}$ for some arbitrary $r \in V$
  $F \leftarrow \emptyset$
  while $S \neq V$ **do**
    find an edge $(u,v) \in S \times (V \setminus S)$ with minimum weight
    $F \leftarrow F \cup \{(u,v)\}$
    $S \leftarrow S \cup \{v\}$
  return $T = (V,F)$

Proof of Correctness

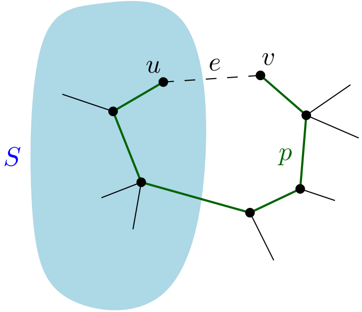


Lemma

Assume that all edge weights are distinct. Let $S \subset V$ be a subset of vertices such that $S \neq \emptyset$ and $S \neq V$. Let $e = (u, v)$ be the edge with smallest weight that starts at a vertex $u \in S$ and ends at a vertex $v \notin S$. Then any MST contains e.

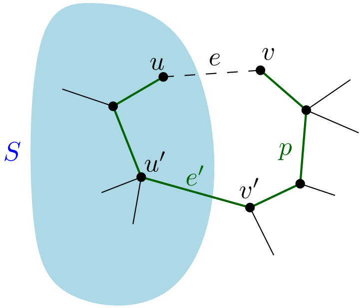
Proof of the Lemma

- Proof by contradiction: Suppose that there is an MST $T(V, F)$ such that $e \notin F$.
- There is a unique path p in T between the endpoints $u \in S$ and $v \notin S$ of e.



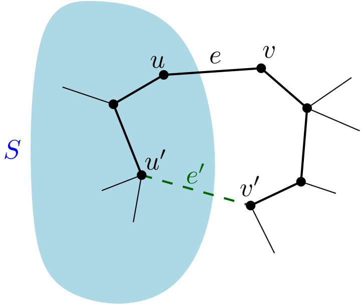
Proof of the Lemma

- There must be an edge $e' = (u', v')$ along p such that $u' \in S$ and $v' \notin S$.



Proof of the Lemma

- Consider $T'(V, F')$ obtained from T by replacing e' with e. So $F' = F \cup \{e\} \setminus e'$.



- Then T' is a spanning tree with weight $w(T) + w(e) - w(e') < w(T)$, a contradiction.

Proof of Correctness

- When all the weights are distinct, this lemma proves that Prim's algorithm picks an edge that lies in any MST at each step.
- At the end, we have $n-1$ edges that belong to any MST.
- So they form the unique MST, as any MST has exactly $n-1$ edges.

Theorem

If all the weights are distinct, then the MST is unique, and Prim's algorithm computes it.

- If the weights are not distinct, then there can be several MSTs, and Prim's algorithm returns one of them. (Why?)

Efficient Implementation

- Let $n = |V|$ and $m = |E|$ denote the number of vertices and edges of G.
- Then a naive implementation of Prim's algorithm (Slide 9) runs in time $\Omega(nm)$.
- We now show how to make it run in $O(m \log n)$ time using a heap-based priority queue $\mathcal{Q}$.
- At any time during the course of the algorithm, $\mathcal{Q}$ records for each $v_j \in V \setminus S$ the lightest edge (p_j, v_j) where $p_j \in S$.
- The vertices in S are marked. (For instance, using a Boolean flag.)
- At Line 14, we update the key of an element of the queue. It can be done in $O(\log n)$ time by deleting and reinserting it with the new key value. There is also a more direct way of implementing it.

Prim's Algorithm: Detailed Pseudocode

```
procedure PRIM2(G(V, E))
    Q ← empty priority queue
    p[1, ..., n] ← array of vertices
    F ← empty list of edges
    for i = 1, n do
        insert v_i into Q with key(v_i) = ∞
    while Q ≠ ∅ do
        v_i ← extract-min(Q)                          ▷ new vertex in S
        mark v_i
        if key(v_i) ≠ ∞ then
            F ← F ∪ {(p[i], v_i)}
        for each unmarked node v_j adjacent to v_i do
            if w(v_i, v_j) < key(v_j) then
                CHANGEKEY(Q, v_j, w(v_i, v_j))
                p[j] ← v_i
    return T = (V, F)
```

Prim's Algorithm

- We assume that $m \geqslant n-1$ as otherwise G is disconnected, and hence there is no MST. We also assume that $n \geqslant 2$ as otherwise the problem is trivial.

Theorem

Prim's algorithm, as implemented in Slide 16, runs in $O(m \log n)$ time.

Proof.

Each queue operation takes $O(\log n)$ time. So, not counting the inner **for** loop, the algorithm runs in $O(n \log n)$ time.
The inner for loop is iterated once for every v_i and every v_j adjacent to v_i. In other words, it is iterated twice for each edge (v_i, v_j). Each iteration takes time $O(\log n)$ due to the ChangeKey operation. So overall, the inner **for** loop contributes $O(m \log n)$ time. □

Greedy Algorithms

- A *greedy algorithm* builds the solution step by step, making a locally optimal choice at each step.
- In other words, a greedy algorithm does not look ahead.
- Prim's algorithm is a greedy algorithm: At each step, it extends the MST by adding the shortest available edge.
- In the case of MSTs, we obtain a globally optimal solution thanks to the lemma on Slide 10.
- Unfortunately, for most problems, greedy algorithms do not return a globally optimal solution.
- In practice, they are often used to obtain approximate solutions.