

# **Minimum spanning tree**

Pramesh Kumar

IIT Delhi

March 14, 2024

## Definitions

**Definition (Spanning Tree).** A **spanning tree** of undirected network  $G$  is connected acyclic subgraph that spans all the nodes.

**Definition (Minimum spanning tree (MST) problem).** Given an undirected network  $G(N, A)$  and costs  $c : E \mapsto \mathbb{R}$ , determine a spanning tree  $T$  with minimum cost  $\sum_{(i,j) \in T} c_{ij}$ .

**Remark.** We consider the **undirected network** for spanning tree. In case of **directed network**, the problem (much more difficult problem) is known as **rooted arborescence**. For a node  $r$ ,  $r$ -arborescence is a spanning tree directed away from  $r$ . There is only one directed path from  $r$  to every other node.

**Remark.** For maximum spanning tree, just multiply each cost with by  $-1$  and compute the MST.

# Applications

1. Creating a minimal transit network,
2. Connecting different spatial areas with electricity connection,
3. Clustering (based on Kruskal's algorithm),  
and so on...

# LP formulation

Primal

$$\begin{aligned} \min_{\mathbf{x}} \quad & \sum_{(i,j) \in A} c_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_{(i,j) \in A} x_{ij} = n - 1 \\ & \sum_{(i,j) \in A(S)} x_{ij} \leq |S| - 1, \forall S \subset N \\ & x_{ij} \geq 0 \end{aligned}$$

Dual

$$\begin{aligned} \min_{\lambda, \mu_S} \quad & (n - 1)\lambda + \sum_{S \subset N} (1 - |S|)\mu_S \\ \text{s.t.} \quad & \lambda - \sum_{S: (i,j) \in A(S)} \mu_S \leq c_{ij}, \forall (i, j) \in A \\ & \mu_S \geq 0, \forall S \subset N \\ & \lambda \text{ free} \end{aligned}$$

## Generic MST algorithm

- 1: **Input:**  $G, c$
- 2: (*Initialization*)  $A = \phi$
- 3: **while**  $A$  does not form a tree **do**
- 4:     find a "safe" edge  $(i, j)$  for  $A$
- 5:      $A = A \cup \{(i, j)\}$
- 6: **end while**
- 7: **return**  $A$

We maintain the following loop invariant.

*Prior to each iteration,  $A$  is a subset of some MST*

**Definition (Safe edge):** Any edge added to  $A$  satisfying the above loop invariant is called a **safe edge**.

1. *Initialization:* After line 2, the loop invariant is trivially satisfied.
2. *Maintenance:* The lines 3-5 maintain the loop invariant by only adding "safe" edges.
3. *Termination:* All the edges added to  $A$  were part of MST, so after termination, the loop invariant must hold.

Q. How to find safe edge?

## A few more definitions

**Definition (Cut).** Any partition  $(S, N \setminus S)$  is a **cut**. We say that an edge  $(i, j)$  **crosses** the cut  $(S, N \setminus S)$  if one of the endpoints is in  $S$  and other endpoint in  $N \setminus S$ .

**Definition (Tree/Non-tree edges).** Edges in a given spanning tree are **tree** edges, otherwise they are **non-tree** edges.

### Important observations

1. For every non-tree edge  $(i, j)$ , a spanning tree  $T$  has a unique path connecting  $i$  and  $j$ . Adding edge  $(i, j)$  to  $T$  will create a cycle.
2. Removing any tree edge from a spanning tree will create a cut.

## Optimality conditions

### Theorem (Cut optimality conditions)

A spanning tree  $T^*$  is a minimum spanning tree (MST) if and only if it satisfies the following optimality conditions: For every tree edge  $(i, j) \in T^*$ ,  $c_{ij} \leq c_{kl}$  for every edge  $(k, l)$  contained in the cut formed by removing the edge  $(i, j)$  from  $T^*$ .

### Proof.

$\Rightarrow$  Assume that  $T^*$  is MST. Further, assume that the cut optimality conditions are not satisfied, i.e.,  $\exists$  a tree edge  $(i, j) \in T^*$  removing which creates a cut and  $\exists$  an edge  $(k, l)$  (in original graph) crossing the cut which has cost  $c_{kl}$  strictly less than  $c_{ij}$ . Then, replacing the edge  $(i, j)$  by  $(k, l)$  will produce another tree  $T'$  whose overall cost strictly less than  $T^*$ , which is a contradiction that  $T^*$  is MST.

$\Leftarrow$  We need to show that if any tree  $T^*$  satisfies the cut optimality conditions, then it must be MST. Suppose  $T'$  is a MST such that  $T' \neq T^*$ . Then, there must exist an edge  $(i, j)$  in  $T^*$  which is not present in  $T'$ . Removing the edge  $(i, j)$  from  $T^*$  creates a cut  $(S, N \setminus S)$ . Note that if we add  $(i, j)$  to  $T'$ , then it will create a cycle that must contain another edge  $(k, l)$  crossing the cut. Since  $T^*$  satisfies the cut optimality conditions  $c_{ij} \leq c_{kl}$  and since  $T'$  is MST,  $c_{ij} \geq c_{kl}$  implies that  $c_{ij} = c_{kl}$ . Now, we replace  $(i, j)$  by  $(k, l)$  in  $T^*$ , we produce a different spanning tree which has one or more edges common with  $T'$ . Repeating this argument, we can transform  $T^*$  into MST  $T'$ . This shows that  $T^*$  is also a MST.

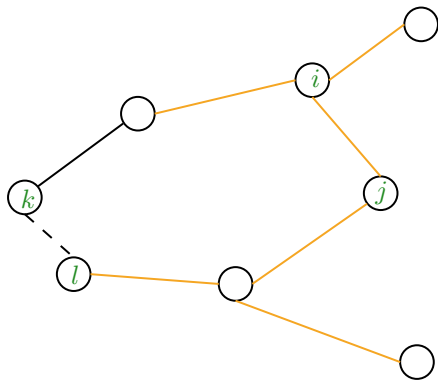


Figure: Replacing  $(i, j)$  by  $(k, l)$

**Remark.** The cut optimality conditions imply that every edge in a MST is a minimum cost edge across the cut that is defined by removing it from the tree.



## Optimality conditions

### Theorem

Let  $F$  be a subset of edges in some MST and let  $S$  be a set of nodes in some component of  $F$ . Suppose  $(i, j)$  is a minimum cost edge in the cut  $(S, N \setminus S)$ . Then some MST contains all the edges of  $F$  as well as edge  $(i, j)$ .

### Proof.

Let  $F \subseteq T^*$  (MST). If  $(i, j) \in T^*$ , we are done. Therefore, suppose  $(i, j) \notin T^*$ . Then, adding  $(i, j)$  to  $T^*$  creates a cycle and therefore,  $\exists (k, l) \neq (i, j) \in (S, N \setminus S)$ . By assumption,  $c_{ij} \leq c_{kl}$  and also  $T^*$  must satisfy the cut optimality conditions which says  $c_{ij} \geq c_{kl}$ . So replacing  $(k, l)$  by  $(i, j)$  will produce another MST that contains  $F$  as well as  $(i, j)$ . □

## Optimality conditions

### Theorem (Path optimality conditions)

A spanning tree  $T^*$  is a MST if and only if satisfies the following path optimality conditions: For every non-tree edge  $(k, l)$  of  $G$ ,  $c_{ij} \leq c_{kl}$  for every edge  $(i, j)$  contained in the path in  $T^*$  connecting nodes  $k$  and  $l$ .

### Proof.

$\Rightarrow$  Suppose  $T^*$  is a MST and  $\exists$  a non-tree edge  $(k, l)$  and a tree edge  $(i, j)$  contained in the path connecting  $k$  and  $l$  such that  $c_{ij} > c_{kl}$ . In that case, we can remove  $(i, j)$  and add  $(k, l)$  creating another tree  $T'$  with cost  $c(T') < c(T^*)$ , contradicting the assumption that  $T^*$  is a MST.

$\Leftarrow$  We'll show that  $T^*$  satisfying the path optimality conditions also satisfy the cut optimality conditions, implying that  $T^*$  is a MST using previous theorem. Let  $(i, j) \in T^*$  and let  $S$  and  $\bar{S}$  be the set of connected nodes produced by removing edge  $(i, j)$  from  $T^*$ . Suppose  $i \in S$  and  $j \in \bar{S}$ . Consider any edge  $(k, l) \in (S, \bar{S})$ . Since  $T^*$  contains a unique path joining nodes  $k$  and  $l$  and since  $(i, j)$  is the only edge connecting a node in  $S$  and a node in  $\bar{S}$ , edge  $(i, j)$  must belong to this path. The path optimality conditions implies that  $c_{ij} \leq c_{kl}$ ; since this condition must be valid for every nontree edge  $(k, l)$  in the cut  $(S, \bar{S})$  formed by removing any tree edge  $(i, j)$ ,  $T^*$  satisfy the cut optimality conditions and so it must be MST.  $\square$

## Kruskal's algorithm

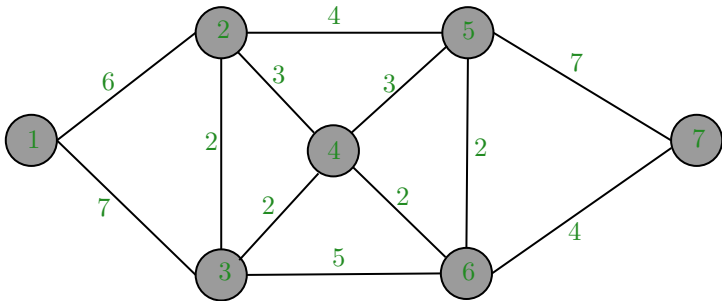
```
1: Input:  $G, c$ 
2: (Initialization)  $A \leftarrow \phi$ 
3: for each  $i \in N$  do
4:   MAKESET( $i$ )
5: end for
6: Let  $A$  be the set of links sorted in increasing order by their costs  $c$ .
7: for each  $(i, j) \in A$  do
8:   if FINDSET( $i$ )  $\neq$  FINDSET( $j$ ) then
9:      $A = A \cup \{(i, j)\}$ 
10:    UNION( $i, j$ )
11:   end if
12: end for
13: return  $A$ 
```

▷ Links in MST

### Theorem

Kruskal's algorithm can be implemented in  $O(m \log n)$  time.

## Example



## Prim's algorithm

- ▶ Based on the cut-optimality condition.
- ▶ Maintains for every node  $d(i)$  and  $pred(i)$  representing minimum cost of any edge connecting  $i$  to another node in tree and predecessor respectively.
- ▶ Also maintains a heap  $Q$  of all nodes not in the tree yet.
  - Heap is a data structure having a collection of objects with unique  $key$ .
  - We can perform operations such as  $CREATEHEAP()$ ,  $INSERT(i, Q)$ ,  $DECREASEKEY(Q, j, c_{ij})$ , etc.
  - Check out [heapq](#) in Python.

# Prim's algorithm

```
1: Input:  $G, c, s$ 
2:  $d(i) \leftarrow \infty, \forall i \in N \setminus \{s\}; d(s) \leftarrow 0$ 
3:  $pred(i) \leftarrow \text{NA}, \forall i \in N \setminus \{s\}$ 
4:  $Q \leftarrow \text{CREATEHEAP}()$ 
5: for each  $i \in N$  do
6:    $\text{INSERT}(Q, i)$ 
7: end for
8: while  $Q$  do
9:    $i \leftarrow \text{FINDMIN}(Q)$ 
10:   $\text{DELETE}(Q, i)$ 
11:  for  $j \in FS(i)$  do
12:    if  $j \in Q$  and  $d(j) > c_{ij}$  then
13:       $d(j) \leftarrow c_{ij}$ 
14:       $pred(j) \leftarrow i$ 
15:       $\text{DECREASEKEY}(Q, j, c_{ij})$ 
16:    end if
17:  end for
18: end while
19: return  $A = \{(pred(i), i) : i \in N \setminus \{s\}\}$ 
```

► Graph, costs, source

► Creates a heap  $Q$

► Inserts node  $i$  into heap  $Q$

► Reduces the key of  $j$  in  $Q$  to  $c_{ij}$

## Theorem

Above algorithm runs in  $O(m \log n)$  time.

Lines 4-6:  $O(n)$  time; while runs:  $O(n)$  times; 9-10:  $O(\log n)$  time; Line 14:  $O(\log n)$  time. For loop runs:  $O(m)$  times;

## Sollin's algorithm

- ▶ Sollin's algorithm combines ideas from both Kruskal's and Prim's algorithm.
- ▶ It maintains a set of forests (like Kruskal's) but only selects the edge with minimum cost (like Prim's).
- ▶ Running time  $O(m \log n)$ .

## Suggested reading

- ▶ AMO Chapter 13



Thank you!