

Motivation

- many algorithms can be applied to unweighted trees
 - [[Spannbaumalgorithmen]]
 - [[Breadth-First Search]]
 - [[Depth-First Search]]
 - ...
- does not work for weighted graphs
- useful if direct connections from A to B not possible
 - road networks
 - electrical circuits
 - ...

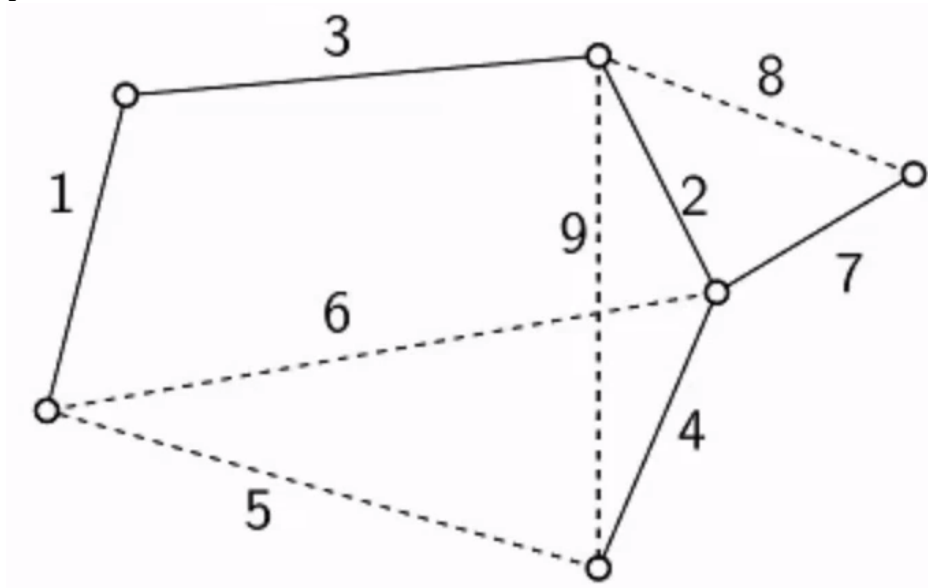
Definition

- tree which connects all points which minimizes the total length of all edges
A **minimum spanning tree** of a weighted graph $G = (V, E, w)$ is a tree $T = (V, E')$ with $E' \subseteq E$ and with minimal total edge length among all spanning trees in G :

$$w(T) = \sum_{e \in E'} w(e)$$

– is minimized over all trees in G with vertex set V .

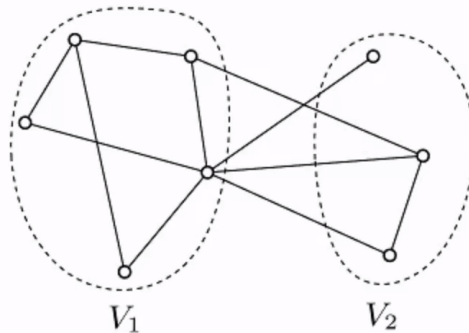
- crossing free (planar)
 - otherwise using different edges would lead to a fewer length
- example



Algorithm Idea

- cut

A **cut** of a graph $G = (V, E)$ is a partition of V into V_1, V_2 .



An edge e is called **external** for the cut (V_1, V_2) if it has one endpoint in V_1 and one in V_2 ; otherwise e is called **internal**.

Theorem: Let E' be a subset of edges of an MST of $G = (V, E, w)$. Let (V_1, V_2) be a cut of G for which all edges of E' are internal. Then the external edge of the cut with **minimum weight** is a good edge for E' .

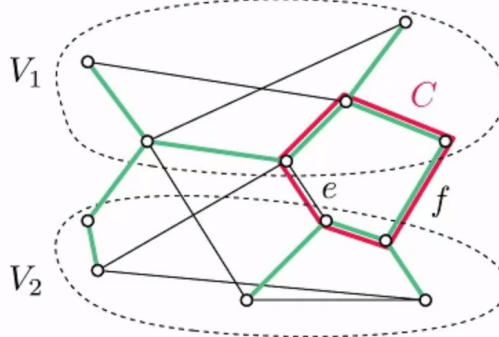
- proof

Proof:

Assume there is an MST T with E' and without e .

$\Rightarrow e$ closes a cycle C in T .

The cycle C contains at least one edge f of the tree T that is external for the cut.



By definition of e , the weight $w(f) \geq w(e)$.

$\Rightarrow T \setminus \{f\} \cup \{e\}$ is an MST of G . \square

If $w(f) > w(e)$ then T is not an MST of G .

*

Prim's Algorithm

- greedy
- also works with negative weights

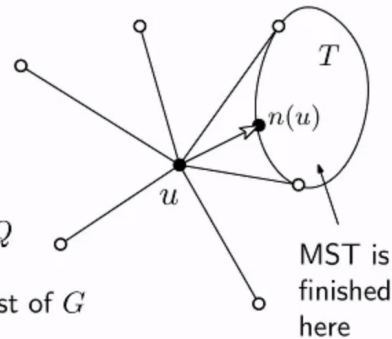
- Start with an arbitrary vertex s of G and iteratively 'grow' an MST T from s .
- **Iterative step:**
Choose the 'cheapest' edge with exactly one node in T .
- Cut: $V_1 =$ vertices of T , $V_2 =$ vertices not yet in T .
- For each vertex $v \notin T$ we maintain:
 - Priority $p(v)$: weight of the shortest edge from v to a vertex in T (initially: ∞).
 - Nearest $n(v)$: vertex in T realizing $p(v)$: $w(v, n(v))$ is min. among neighbors of v in T (initially no vertex).
- A **queue** Q contains all vertices not yet in T , organized by priorities (e.g., in a min-heap; initially all vertices).

• pseudo code

PRIM-MST (G, s)

```

for all  $v \in V$  do  $p(v) = \infty$  od
 $p(s) = 0$ ,  $n(s) = \text{nil}$ 
 $Q = V$  // build up  $Q$ 
while  $Q \neq \emptyset$  do
   $u = \text{MIN}(Q)$ 
  remove  $u$  from  $Q$  // reorganize  $Q$ 
  write  $u, n(u)$ 
  for all  $v \in A(u)$  do //  $A$ : adj. list of  $G$ 
    if  $v \in Q$  and  $p(v) > w(u, v)$  then
       $p(v) = w(u, v)$  // reorganize  $Q$ 
       $n(v) = u$ 
    fi
  od
od
  
```



Run-time-analysis:

n ... number of vertices of G

m ... number of edges of G

$d(v)$... Degree of vertex v in G

- Initialization, construction of the heap Q : $\Theta(n)$
- n times removing the minimum from Q : $O(n \log n)$
- report MST edges: $\Theta(n)$
- Update priorities for all neighbors of v : $O(d(v) \cdot \log n)$

\Rightarrow Altogether:

$$O(n + n \log n + \sum_{v \in V} d(v) \log n) = O(m \log n)$$

Memory requirements: $\Theta(m + n) = \Theta(m)$

Graph + queue + priorities + nearests + constant additional

Remarks:

- MST always begins at the start vertex s and grows from there as a connected tree.
- Shrinking $p(v)$ causes v to move up in the heap.
 $\Rightarrow O(\log n)$ time.
- Test for $v \in Q$ in $O(1)$ time when bit vector is used to store which vertices are already in T .
- Runtime can be changed to $O(n^2)$. This is useful for dense graphs (see notes on Dijkstra's algorithm in the next chapter).

Kruskal's Algorithm

- greedy
- also works with negative weights
- uses [[Union Find]] data structure
 - Label the vertex set of G as v_1, v_2, \dots, v_n (arbitrary)
 - Initially there are n disjoint sets M_1, M_2, \dots, M_n (each with one vertex)
 - $\text{FIND}(v)$: returns index i if vertex v is in M_i
 - $\text{UNION}(i, j)$: join sets M_i and M_j : $M_i = M_i \cup M_j$ (index of resulting set: minimum of i and j)
 - End of the algorithm: one component M_1 with all vertices of G .
-
- Creating a 1-element set needs $\Theta(1)$ time.
- f FIND and u UNION operations need $O(f + u \log u)$ time in total.
- The total memory requirement of the data structure is linear in the number of initial 1-element sets.
-

- Start with empty edge set E' .
- Sort edge set E of $G = (V, E, w)$ in increasing order of their weights (edges will be considered in this order): e_1, e_2, \dots, e_m with $w(e_1) < w(e_2) < \dots < w(e_m)$.
- **Iterative step:**
 - E' forms a forest F (= set of disjoint subtrees, acyclic) in G and in the MST to be constructed.
 - Edge e that is added to E' is the shortest edge in $E \setminus E'$ that does not form a cycle with edges from E' .
- pseudo code

KRUSKAL-MST(G)

sort edges by weight: $\{e_1, e_2, \dots, e_m\}$

for $i = 1$ **to** n **do** $M_i = \{v_i\}$ **od**

for $k = 1$ **to** m **do**

$(u, v) = e_k$

$i = \text{FIND}(u)$

$j = \text{FIND}(v)$

if $i \neq j$ **then**

 write e_k

 UNION(i, j)

fi

od

Runtime analysis:

- Sorting of the edges: $O(m \log m)$
 - Initialize UNION-FIND data structure for vertices: $\Theta(n)$
 - In total $2m$ FIND operations and $n - 1$ UNION operations: $O(m + n \log n)$
 - Extract edges + write MST edges: $\Theta(m)$
- \Rightarrow Altogether $O(m \log m)$ time.
- \Rightarrow Sorting of the edges dominates the runtime.

Memory requirements:

- $\Theta(n + m) = \Theta(m)$ in total.
-