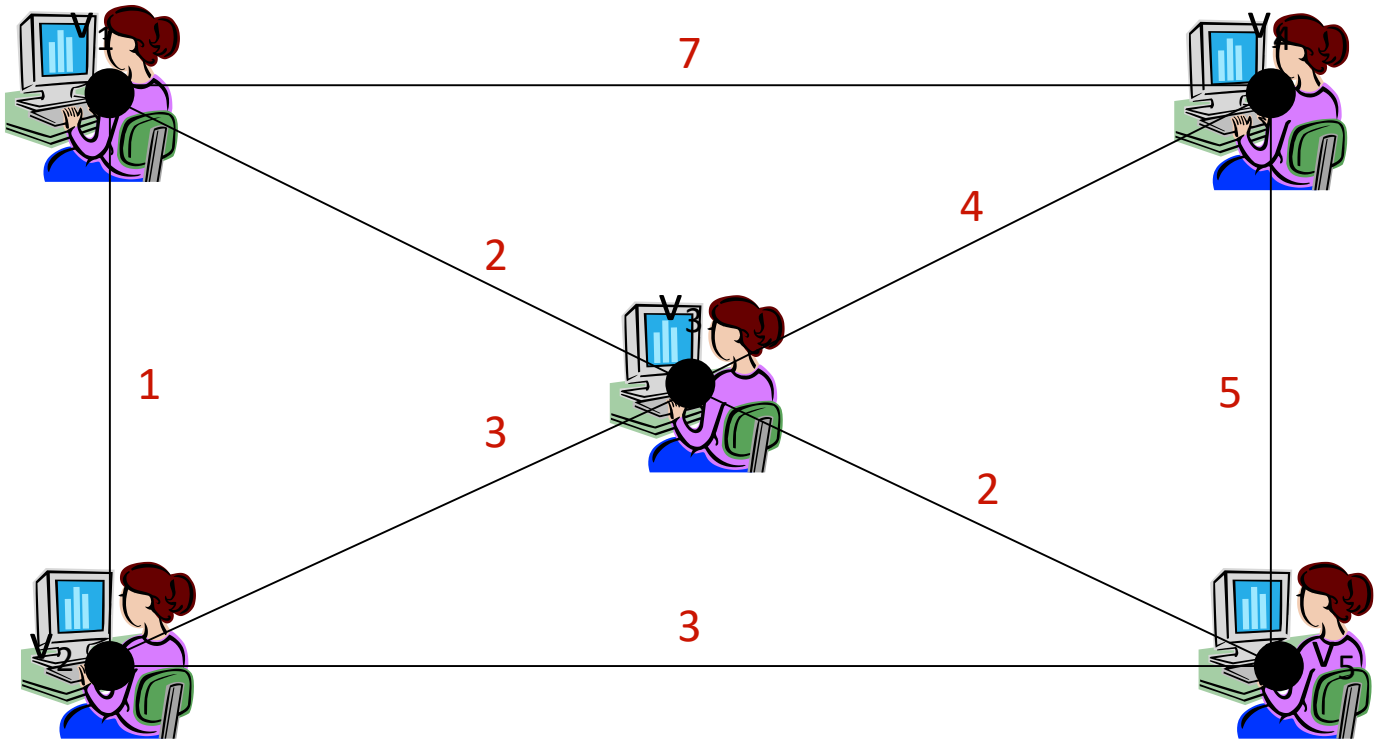
Algorithm and Data Structure Analysis (ADSA)

Minimum Spanning Trees



Graph problem

Establish wired network of minimal cost such that persons can communicate with each other

Minimum Spanning Tree Problem:

- Given a connected undirected graph G=(V,E) with positive edge costs c: E -> R_+.
- Find a connected subgraph T of G that has minimal cost.

Example:

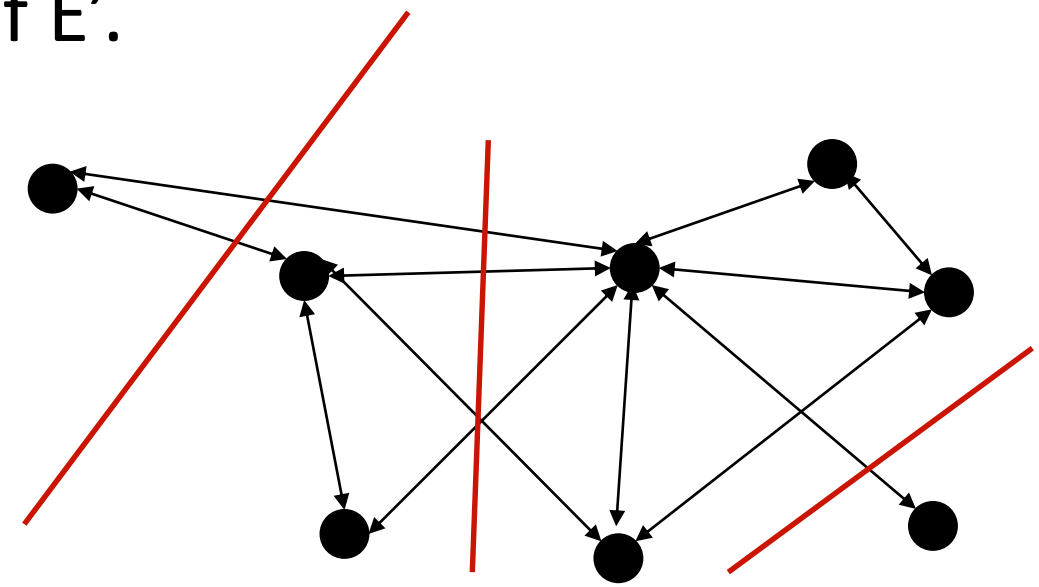
- Nodes are the computers of the network.
- Edges are possible connections between computers.
- Costs are the costs of establishing a particular connection.

Minimum Spanning Trees

- Edge costs are positive.
- Implies that the connected subgraph of minimal cost does not contain a cycle
- It is a tree spanning all nodes of the graph (called a spanning tree).
- A spanning tree of minimal cost is called a minimum spanning tree (MST).

Cuts

A cut in a connected graph G=(V,E) is a subset E’ of edges such that G’=(V, E \ E’) is not connected. G’ is obtained from G by deleting the edges of E’.



Properties of MSTs

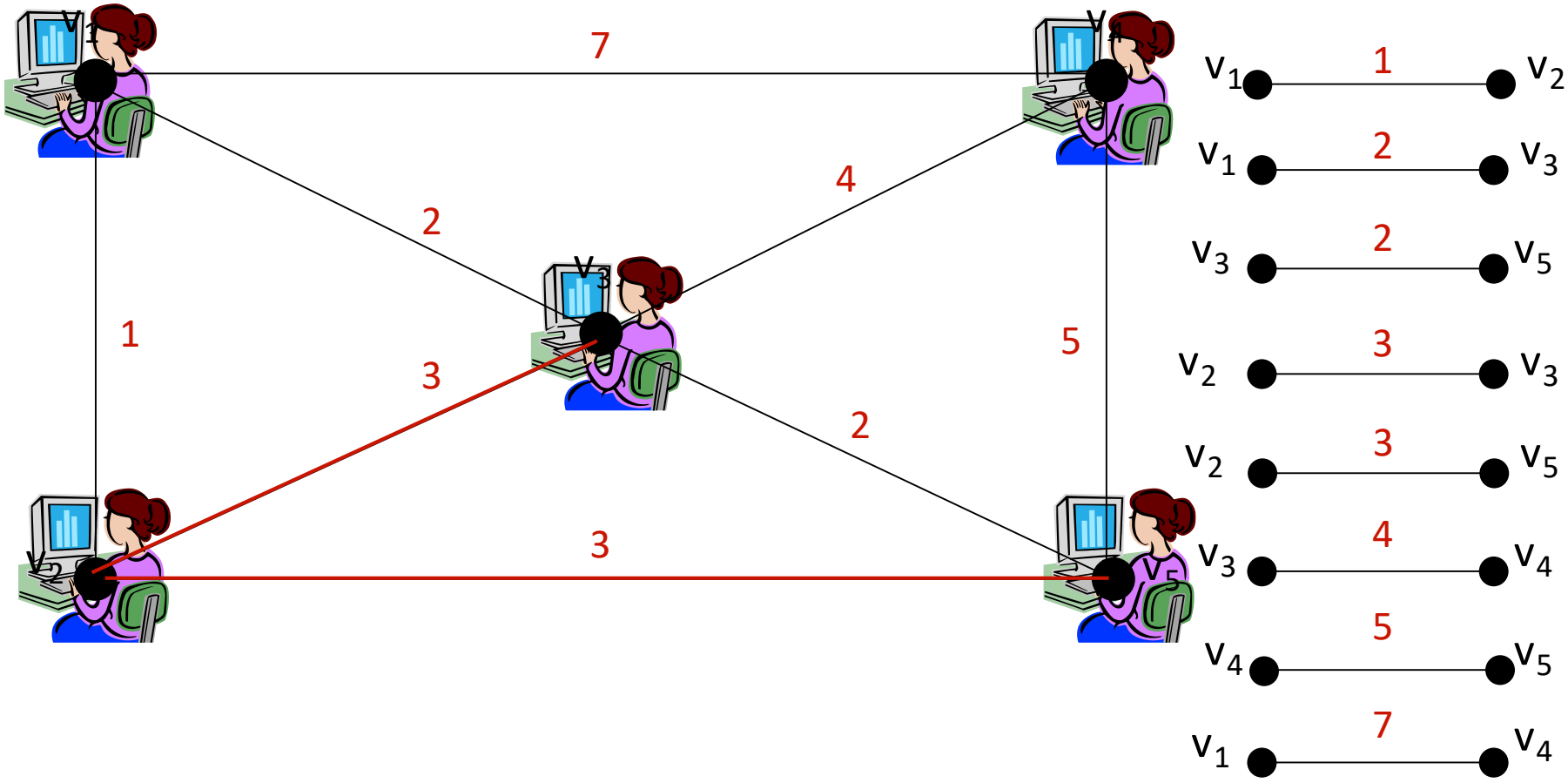
An MST of a given graph G can be constructed by greedy algorithms.

Crucial properties:

- Cut property (Let e be an edge of minimum cost in a cut C. Then there is an MST that contains e)
- Cycle property (an edge of maximal cost in any cycle does not need to be considered for computing an MST)

1. Sort the edges with respect to increasing weight

2. Introduce edges in ascending order that do not create cycles



Description of Kruskal's algorithm:

- Sort in the edges of the graph with respect to increasing weights.
- Introduce the edges in ascending order that do not create cycles.

Function *kruskalMST*(*V, E, c*) : *Set* **of** *Edge*
    $T := \emptyset$
    invariant *T is a subforest of an MST*
    foreach $(u,v) \in E$ in ascending order of cost **do**
        if u and v are in different subtrees of T **then**
            $T := T \cup \{(u,v)\}$
    return T

Figure 11.4 Mehlhorn/Sanders

Cut Property

An edge of minimal cost in a cut belongs to an MST.

Lemma 11.1 (cut property). *Let E' be a cut and let e be a minimal-cost edge in E'. There is then an MST T of G that contains e. Moreover, if T' is a set of edges that is contained in some MST and T' contains no edge from E', then $T' \cup \{e\}$ is also contained in some MST.*

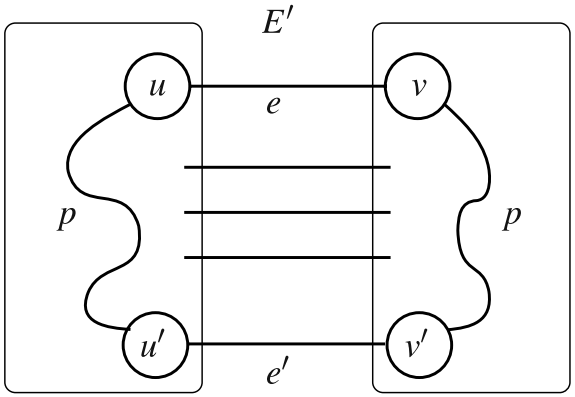
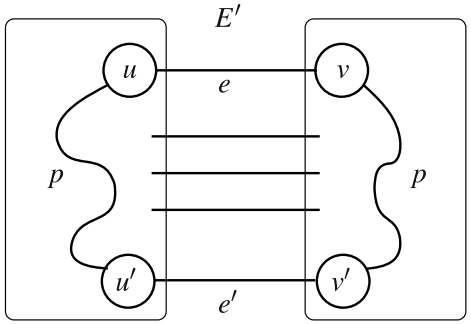


Fig 11.1 Mehlhorn/Sanders

Fig 11.1 Mehlhorn/Sanders



Proof (of Lemma 11.1)

- Consider any MST T of G with $T' \subseteq T$ and e={u,v} be the edge with endpoints u and v.
- Consider the path p from u to v in T (it exists as T is a spanning tree)
- E' is a cut separating u and v which implies that p must contain an edge from E', say e'.
- The tree $T'' := (T \setminus e') \cup e$ is also a spanning tree as the removal of e' splits T into two subtrees which are joined by e.
- $c(e) \leq c(e')$ implies $c(T'') \leq c(T)$ and hence T'' is also a minimum spanning tree of G
- Setting $T' = \emptyset$ implies that e belongs to some MST.

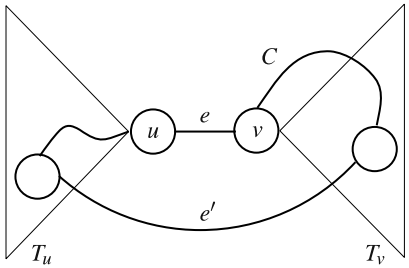


Fig 11.1 Mehlhorn/Sanders

Cycle Property

Lemma 11.2 (cycle property). *Consider any cycle $C \subseteq E$ and an edge $e \in C$ with maximal cost among all edges of C. Then any MST of $G' = (V, E \setminus \{e\})$ is also an MST of G.*

Proof:

- Consider any MST T of G.
- Suppose T contains an edge e={u,v} that splits T into two subtrees T_u and T_v.
- There must be another edge e'={u',v'} from C connecting T_u and T_v.
- $T' := (T \setminus e) \cup e'$ is a spanning tree that does not contain e.
- $c(e') \leq c(e)$ implies that T' is also an MST.

Runtime

Runtime of Kruskal's Algorithm:

- Sorting of the edges takes time O(m log m).

We need to

- test whether an edge connects two different components.
- join the two components when an edge is introduced.
- We want to do this in time O(m + n log n).
- Overall runtime for Kruskal's algorithm is O(m log m).

Consider Union-Find Data structure.

Union-Find Data Structure

- Given n elements 1, 2, …,n
- We start with n sets {1}, {2}, …, {n} consisting of the different n elements.

We want to support the following operations:

FIND(x), $1 \leq x \leq n$: Find the name of the set to which x belongs.

UNION(A,B): Join the sets A and B. The name of the union of the two sets is either A or B.

We want to have a data structure that efficiently supports UNION and FIND operations.

For the computation of minimum spanning trees by Kruskal's algorithm we need 2m FIND and n-1 UNION operations.

- FIND: For each edge we find the sets containing the two nodes.
- UNION: When we introduce an edge we join the two sets.

List-oriented data structure

- Use and array R of size n
- Store at R[i] to name of the set to which element i belongs.
- For each set A we use a list L(A) storing all elements in A.
- Let s(A) the size of A.

Time:

- Initialization takes time O(n)
- FIND(x): Takes constant time (Look up at R[x])

UNION

UNION(A,B)

If $s(A) \leq s(B)$ then

- Run through L(A) and set for each x in L(A), R(x):=B and add x to L(B). Set s(B):= s(A) + s(B).

else

- Run through L(B) and set for each x in L(B), R(x):=A and add x to L(A). Set s(A):= s(A) + s(B).

Theorem: Using the list-oriented data structure each sequence of n-1 UNION and m FIND operations can be carried out in time O(n log n + m).

Proof:

- FIND operations: Lookup in array R (time O(m) for 2m operations).
- Show that n-1 UNION operations take time O(n log n).

n-1 UNION operations:

- Let A_1, …, A_{n-1} be the smaller set in the application of the UNION operations.
- The total time is $O(s(A_1) + ….+ s(A_{n-1}))$
- Show that this sum is O(n log n) using amortized analysis (bookkeeping).

Upper bound for $O(s(A_1) + ...+ s(A_{n-1}))$

- We assign costs to each of the n objects 1, …, n. Each element gets a cost of 0 in the beginning.
- Each object x starts in a set of size $1=2^0$.
- The size of the set of an object x doubles every time we assign an additional cost 1 to x.
- $2=2^1$, $4=2^2$, $8= 2^3$…

- Let a(x) be the total cost assign to x
- Then the set to which x belongs has at least $2^{a(x)}$ elements.
- Size of a set is at most n.
- $2^{a(x)} \leq n$ implies $a(x) \leq \log n$.
- Total cost for n elements and therefore n-1 UNION operations is O(n log n).

□