Winter School

Minimum Spanning Trees

Tobias Weinzierl

14.2.2003

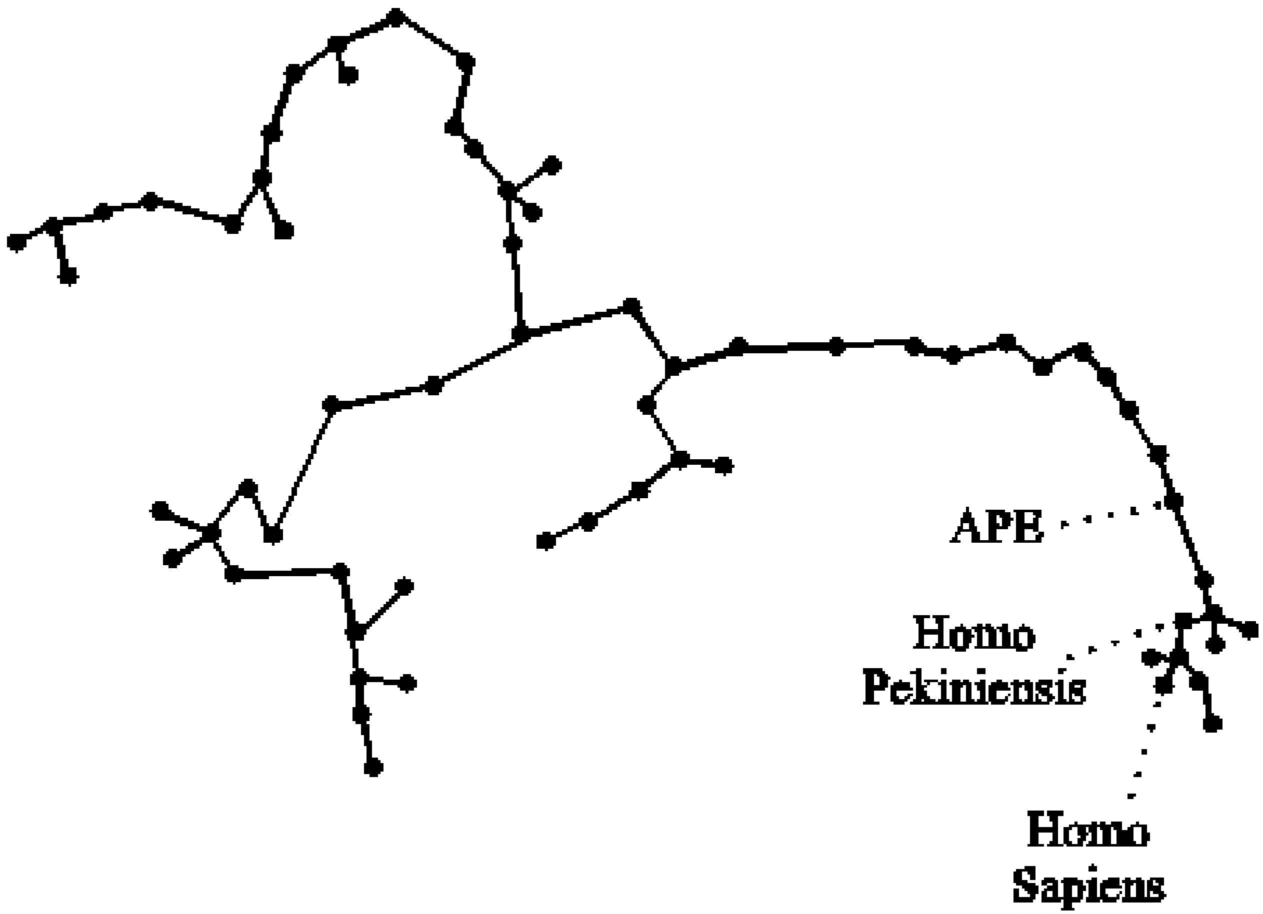
MST

- Given an undirected, connected graph G=(V,E) and a function $w : V \mapsto K$ the minimum spanning tree is a spanning tree (a tree containing all vertices $e \in E$) with minimum total weight.
- The total weight of a tree T=(V,E) is $w(T) := \sum_{e \in E} w(e)$.
- Good polynomial-time algorithms are known since 1930

Applications: Euclidean minimal spanning tree (EMST)

- Set of nodes V each representing a d-dimensional vector
- EMST is the MST for the complete graph K_n on V where edge weights are Euclidean distances between vectors
- Applications: Network design, visualizing multidimensional data, clustering, classification, taxonomy

Applications



MST properties: Strong cycle property

- e is the heaviest edge on some cycle of G $\Rightarrow e \notin E_{MST}$
- Rule is also called red rule
- Proof: By contradiction

MST properties: Cut property

- For any proper nonempty subset $\|V\|'$ of the vertices, the lightest edge with exactly one endpoint in $\|V\|'$ belongs to the MST
- Rule is also called blue rule
- Proof: By contradiction

MST properties

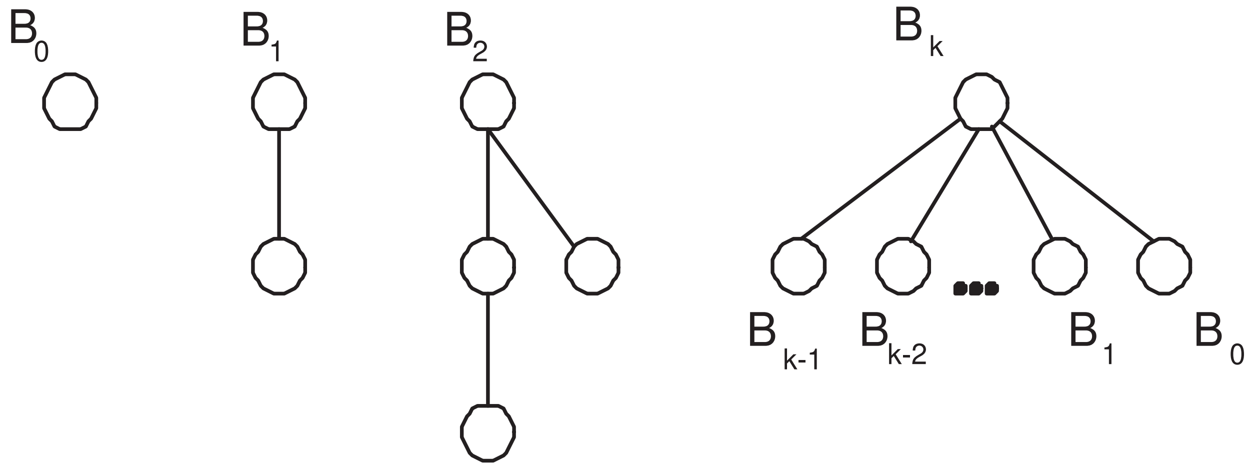
If the edge weights are distinct, the MST is unique

Queues and heaps: Definition

- Queue = first in first out (FIFO)
- Heap: Tree, each node is associated a value, value of parent node is greater than value of child node
- Operations: insert, removeMin (deleteMin), union

Queues and heaps: Binomial heap

- Binomial tree B_k: Tree where the root node has k children. Children are again root nodes of binomal trees $\{B_{k-1}, B_{k-2}, ..., B_1, B_0\}$.
- Binomial heap: Set of binomial trees fullfilling heap property. For every $k \in N$ there's at most one binomial tree B_k.



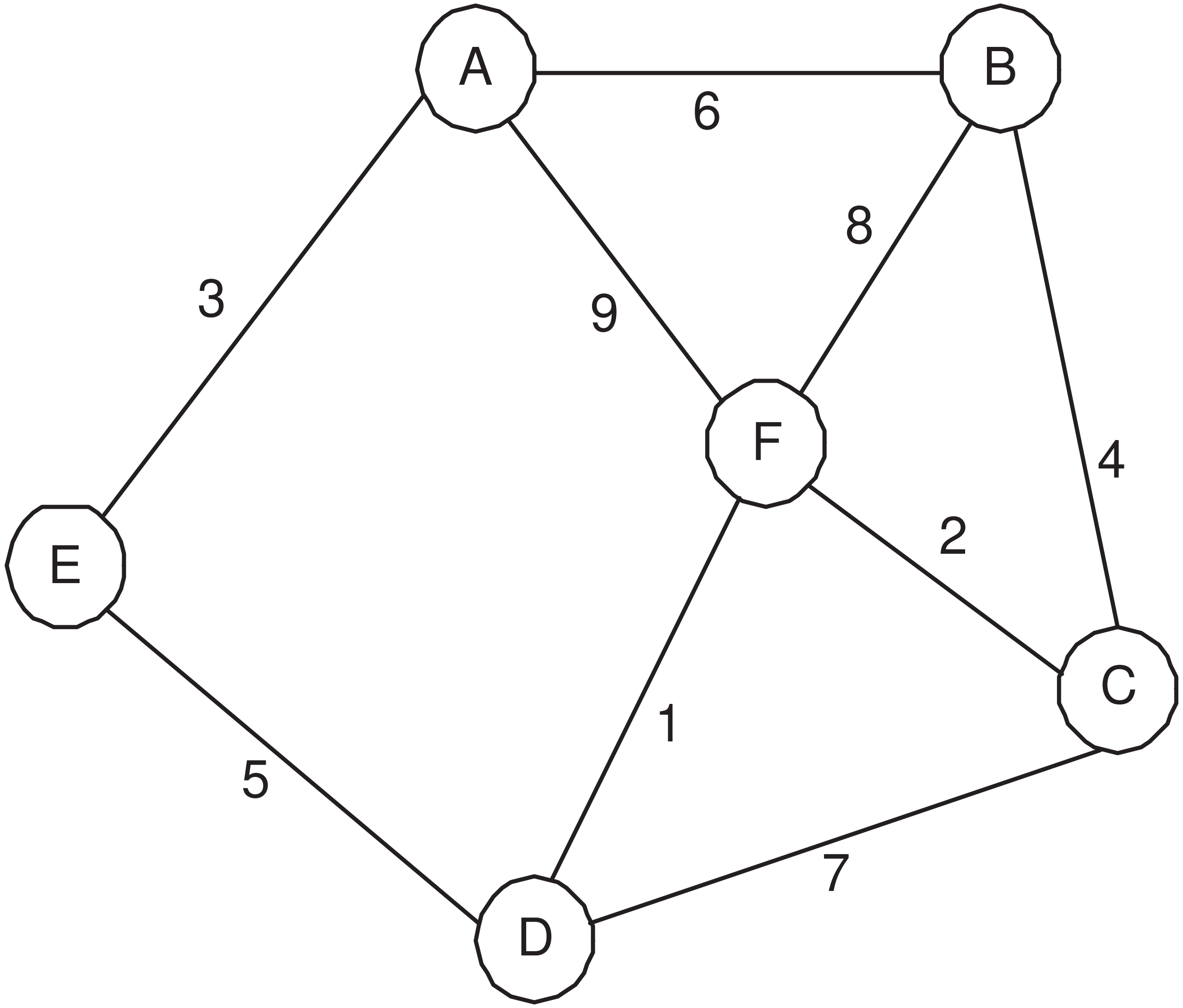
Queues and heaps: Binomial heap

- Binomial heap: Set of binomial trees fullfilling heap property. For every $k \in N$ there's at most one binomial tree B_k.
- All operations call union
- Union: One after another insert the trees of the heaps the operation is applied on in the result heap always taking the smallest one (the one with smallest k). If two heaps are equal with respect to their height in the result heap, union them.
- union $\in O(logn)$

Classical methods: Kruskal's algorithm

- Greedy algorithm on sorted edge set
- Correctness: See blue and red rule
- Pseudocode:

```
for all edges e: insert(E,e,w(e))
while |E(mst)|<|V|-1
  {x,y} := deleteMin(E)
  if x and y are not in same component
    E(mst) := E(mst) + e
```

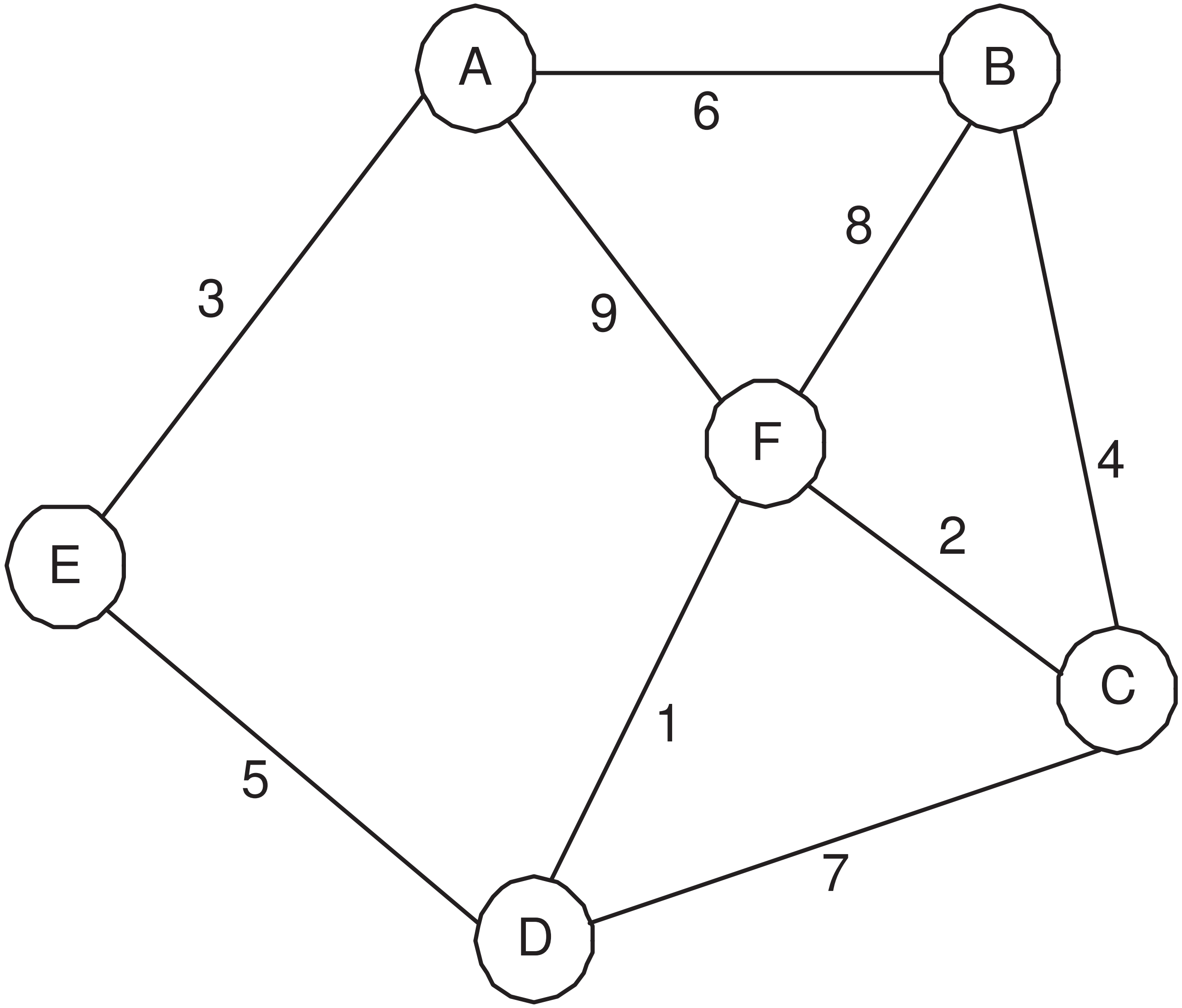

A
B
6
8
3
9
F
4
E
2
C
1
5
7
D

Classical methods: Prim's algorithm

- Start with an arbitrary vertex. Always add cheapest (least weight) edge not creating a cycle (DJP step)
- Correctness: See blue rule
- Pseudocode:

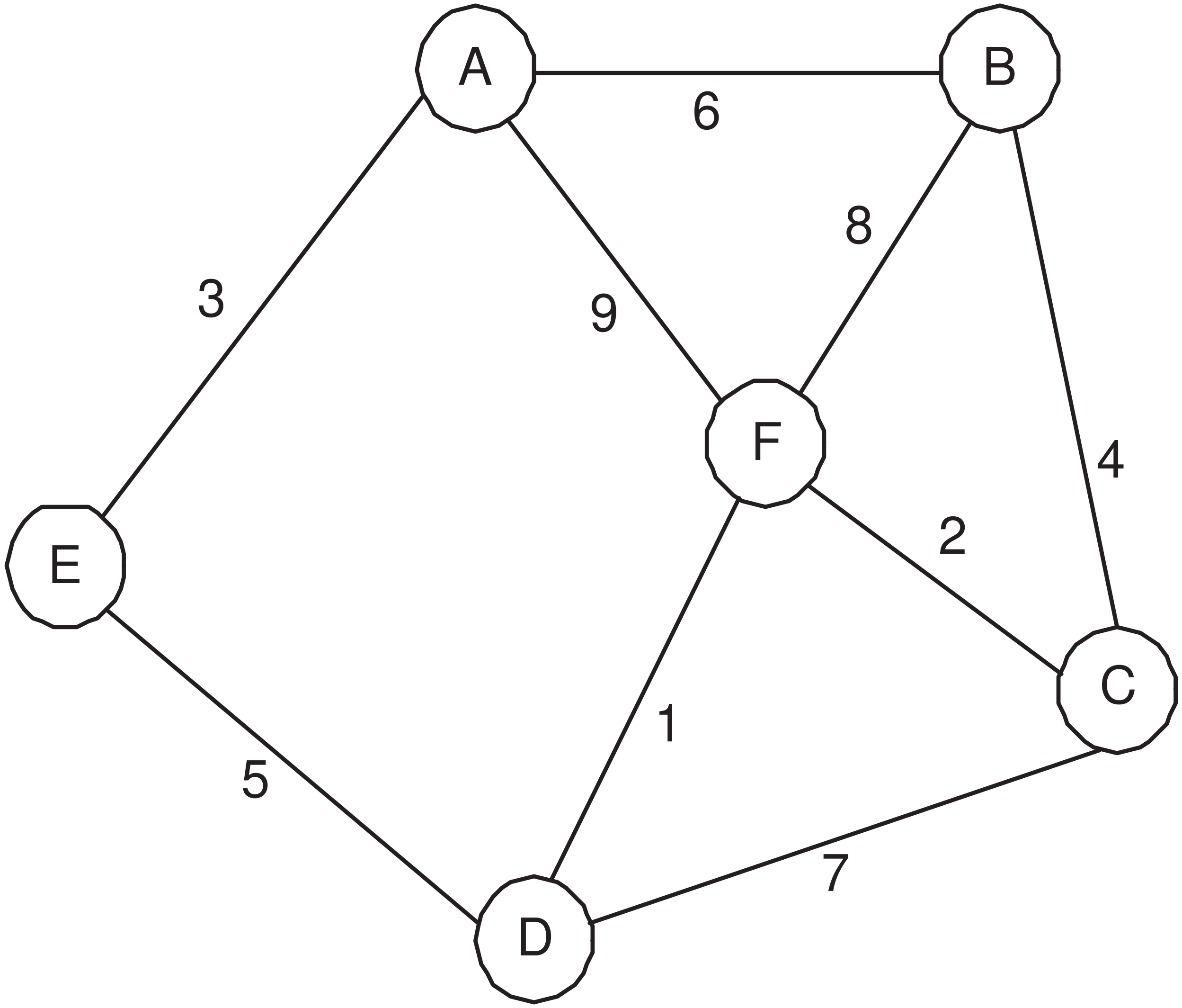
```
vertex e[|V|] := [null, ...]
bool   t[|V|] := [false, ...]
vertex v := arbitrary vertex
t[v] = true
for each vertex v' incident to v
  insert(N, v', w({v,v'}))
```

```
  e[v’] := v
while size(N)>0
  vertex v := smallest node of N
  t[v]     := true
  MST      := MST + {v,e[v]}
  for each vertex v’ incident to v
    if t[v’]=false
      if e[v’]=null
        insert(N, v’, w({v,v’}))
        e[v’] := w({v,v’})
      else
        decreaseKey(N, v’, w({v,v’}))
        e[v’] := w({v,v’})
```


A
B
6
8
3
9
F
4
E
2
C
1
5
7
D

Classical methods: Borúvka’s algorithm

- For every vertex a tree is created.
- For every tree choose least weight edge. Union trees (Borúvka step)
- Correctness: See blue rule

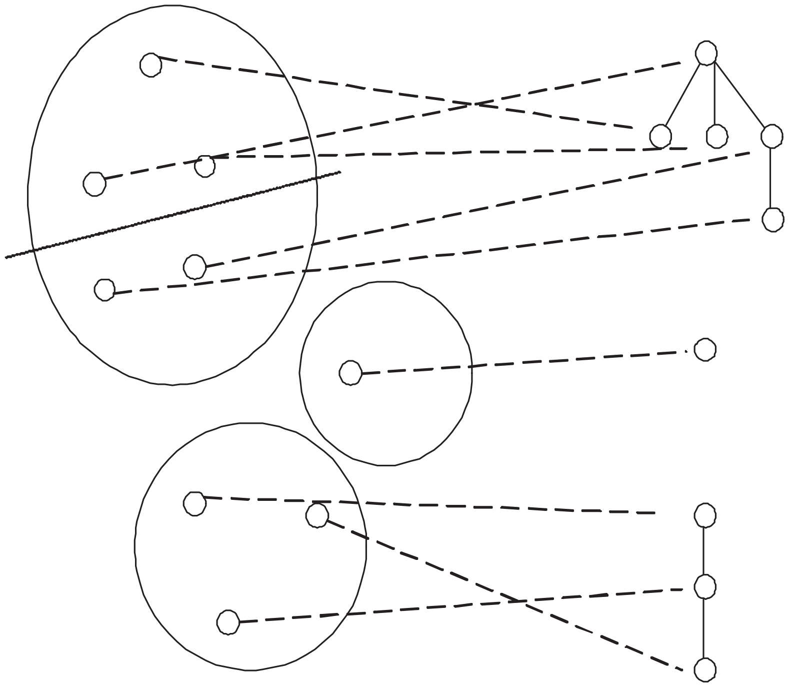

A
B
6
8
3
9
F
4
E
2
C
1
5
7
D

Union-find data structures

- Determine if two points are in the same set
- Find operation returns representative
- Union merges two sets each given by its representative
- Representatives are stored in trees. Root node of tree is set representative
- On an union make root node of smaller tree child of the other one

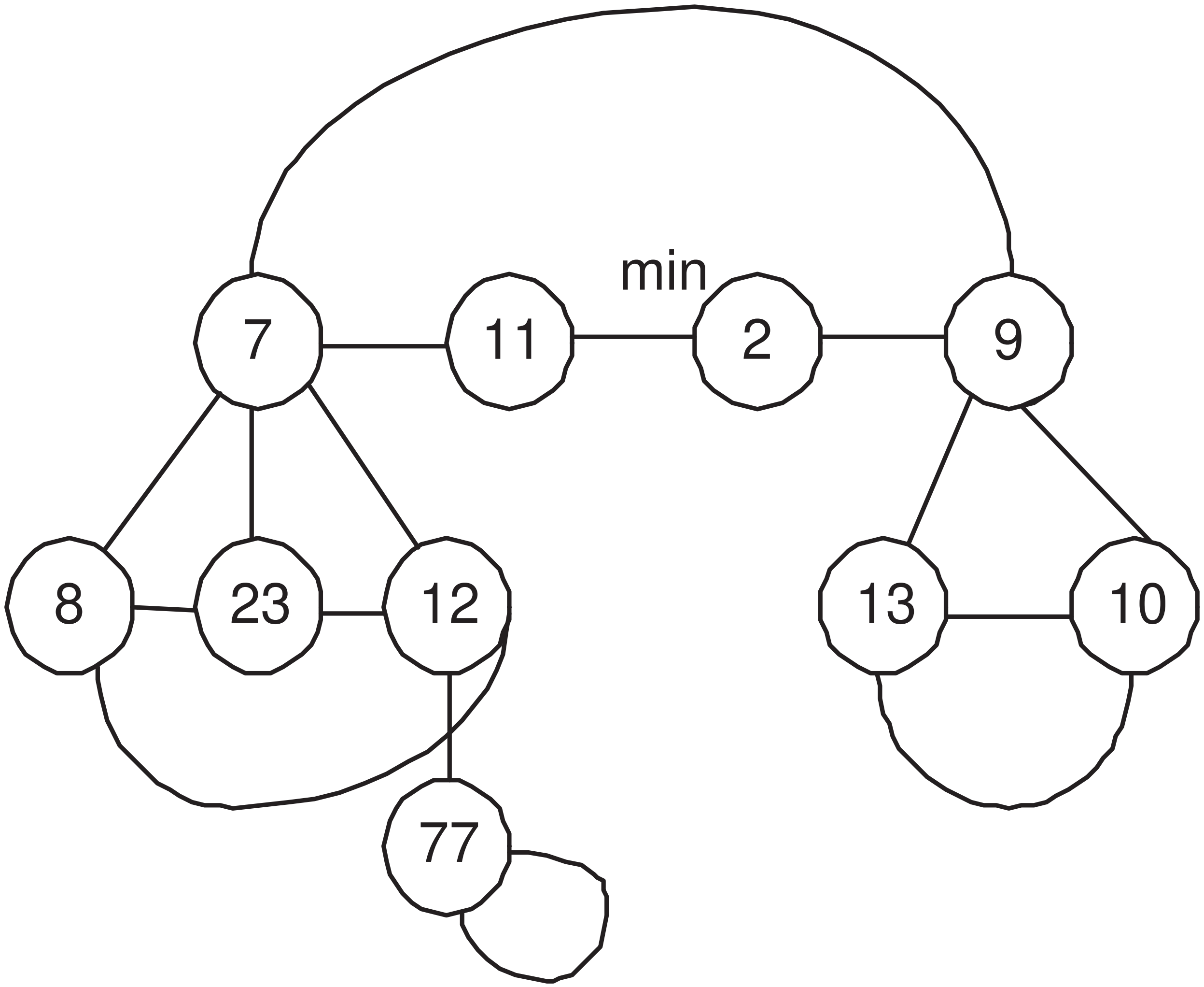
Union-find data structures

- union $\in O(1)$
- find $\in O(logn)$



Fibonacci heap

A Fibonacci heap is a heap. In the trees all nodes in the same tree level - those sharing one parent - are connected bidirectional and cyclic. The tree roots in the heap tree set are connected bidirectional in a cycle, too. A min-pointer identifies the smallest root element stored in the heap. There may be more than one tree in the heap set with height k for any k.

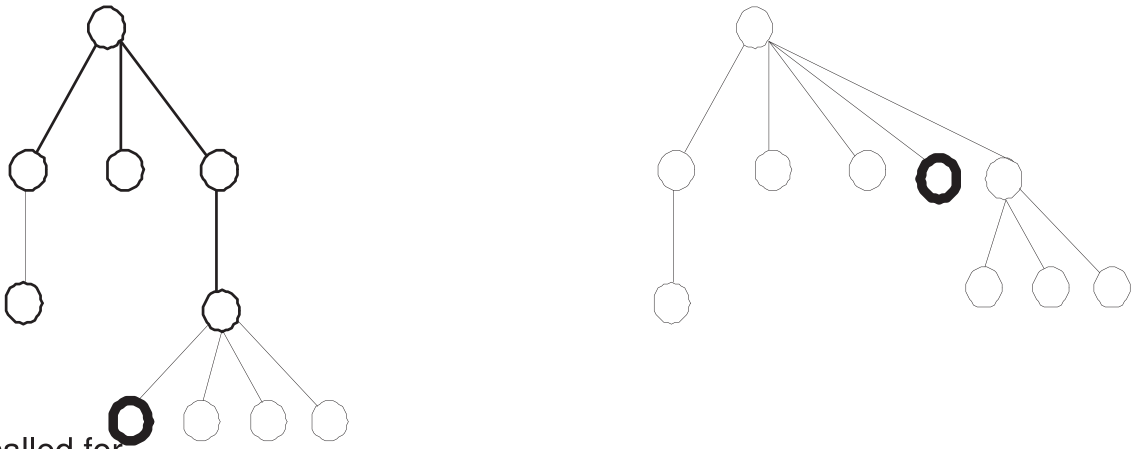

min
7
11
2
9
8
23
12
13
10
77

Fibonacci heap

- insert $\in O(1)$: Add one node in root node cycle. Update min pointer
- union $\in O(1)$: Concatenate root node lists analyzing min elements.
- deleteMin $\in O(logn)$: Add child trees of min element into root list.

Path compression

- Idea: Once the root node for an element is found, make element child of root node. Next time find on element is called, root can be returned in one step.
- Can be done for every node in a path
- m finds and n nodes $\in O(mlog^*n)$ with $log^*n = min(i \in N | log^i n \leq 1)$



find is called for

Complexity

- Kruskal: $O(\|E\| log \|V\|)$ (for sorted edges)
- Prim: $O(\|E\| + \|V\| log \|V\|)$
- Borúvka: $O(min\{\|V\|^2, \|E\| log \|V\|\})$

Faster deterministic algorithms: Fredman & Tarjan's algorithm

- Idea: Use Prim's algorithm but reduce heap size
- Every time heap size exceeds constant k, start growing a new tree
- If tree acquires vertex already contained in another tree, start growing a new tree
- Time complexity: $O(\|E\|\beta(\|E\|, \|V\|))$,
  $\beta(a, b) := min\{i : log^i b \leq a/b\}$

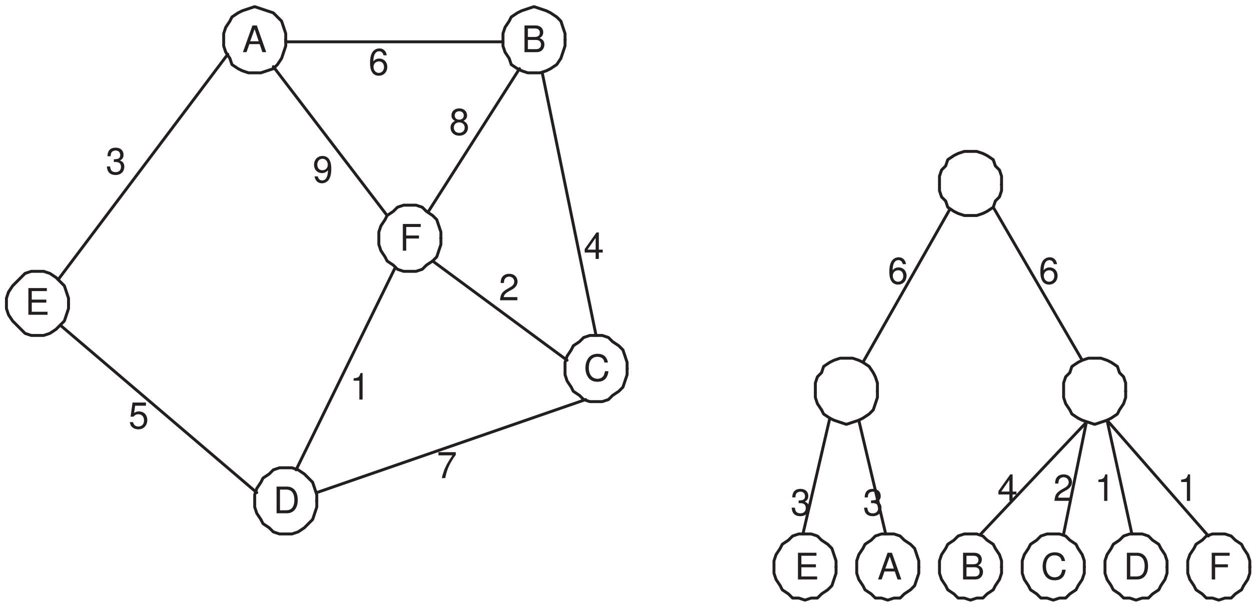
Linear time MST verification

- Verify given spanning tree is minimal
- Concept: Verify red rule for every edge not in the given spanning tree
- Precondition: The heaviest edge of any path in the given tree has to be known

Linear time MST verification: Borúvka tree

- Borúvka's algorithm is in $O(\|V_T\|)$ on any tree
- Tree $B = (V_B, E_B)$ is a branching tree every node represents a tree itself. If the nodes a and b are chilren of r ($a, b, r \in V_B$), the trees a and b were merged into one tree in one step. The edge $(a, root) \in E_B$ is assigned the weight of the edge connecting the tree represented by a and the other subtrees of the tree represented by root.
- Let $T = (\|V_T\|, \|E_T\|)$ be any spanning tree and let B be the branching tree constructed as described above by a function f: The number of nodes in the branching tree $B = (\|V_B\|, \|E_B\|)$ does not exceed $2\|V_T\|$.

Linear time MST verification: Borúvka tree



Linear time MST verification: Borúvka tree

B defined as before. For any pair $x, y \in V_T$, the weight of the heaviest edge in T(x,y) (that is the path from x to y in T) equals the weight of the heaviest edge in $B(f(x), f(y))$.

Proof:

- $\forall e_B \in B(f(x), f(y)) : \exists e_T \in T(x, y) : w(e_T) \geq w(e_B)$
- e_T is heaviest edge on a path $T(x, y)$.
  $\exists e_B \in B(f(x), f(y)) : w(e_B) = w(e_T)$

Linear time MST verification: Maximum weight table

- Idea: Each node is assigned a map. For each ancestor the maximum weight on the path is stored
- Map could be implemented by (sparse) array
- Arrays are constructed in $O(\|V_B\| log \frac{\|V_B\|+\|E_B\|}{\|V_B\|})$ using binary search.

Linear time MST verification: Verification

For each edge $e = \{x, y\}$ not in the given spanning tree:

1. Find $f(x)$ and $f(y)$
2. Find lowest common ancestor a of $f(x)$ and $f(y)$
3. Compare heaviest edges of half paths $(a, f(x))$ and $(a, f(y))$

A randomized algorithm

- Motivation: Reduce number of edges in the graph
- A red edge in any subgraph is a red edge in the whole graph, too
- Use deterministic algorithm on sparse graph

A randomized algorithm: Light edges

Let $G = (V_G, E_G)$ be a graph with weighted edges. Let F be any forest in G. $w_F(\{x,y\})$ is the maximum weight of an edge on $F(\{x,y\})$ with the convention that $w_F(\{x,y\}) = \infty$ if x and y are not connected in F. An edge $e = \{x,y\}$ is F-heavy if $w(\{x,y\}) > w_F(\{x,y\})$. Every non F-heavy edge is F-light.

A randomized algorithm: Light edges

Let G’ be a subgraph of G by including each edge with probability p. Now create the $MST_{G'}$. The number of $MST_{G'}$-light edges exceeds any k with probabilty at most

$$\sum_{i=0}^{n-1} \binom{k}{i} p^i (1-p)^{k-i}$$

with $n = \|V\|$. Therefore the expected number of $MST_{G'}$-light edges in G is at most n/p.

A randomized algorithm: Definition

1. Apply two successive Borúvka steps to the graph, thereby reducing the number of vertices (using contraction) by at least a factor of two. All loops or multiple edges are removed.

2. Choose a subgraph by selecting each each independently with probabilty 1/2. Apply the algorithm recursively to the subgraph H producing a minimal spanning forest MST_H. Find all the MST_H-heavy edges in the whole graph and delete them.

3. Apply the algorithm recursively on the subgraph produced in step 2 and return the result of the algorithm added the edges identified in step 1.

A randomized algorithm: Complexity

- Worst case: $O(min\{\|V\|^2, \|E\|log\|V\|\})$ (see Borúvka)
- Average case: $O(\|E\|)$
  - Total running time is bounded by the total number of edges in the original problem and the number of edges handled in the subproblems
  - Expected size of random subgraph is half the nodes of the calling problem
  - Total number of vertices in tree is $\sum_{i=0}^{\infty} 2^i \frac{\|V\|}{4^i} = 2\|V\|$ (Borúvka)
  - Use upper bound for number of light edges

An optimal MST algorithm: Soft Heaps

- Definition: Binomial Heap where each node represents a (sorted) list of values.
- For each tree node with rank k four attributes are stored:
  1. rank=k
  2. highest value of associated list (ckey)
  3. pointer to root of tree with rank k (refered as child)
  4. pointer to element of rank k the node has been parent before, too (referred as next).
- If a list of a node with rank k is empty and deleteMin is called for, execute the sift-operation

An optimal MST algorithm: The sift operation

1. rank=k
2. if ckey of child is smaller ckey of next exchange next and child
3. append elements of list of next and update ckey
4. call sift on next element
5. if rank exceeds integer parameter r (defined by ϵ) and
6. if rank is odd or rank of child is less rank-1 call sift on child

There are at most ϵn corrupted elements in the heap. n is the number of inserts. All operations are supported in constant amortized time except for insert, which takes $O(log 1/\epsilon)$ time. $r = log\frac{1}{\epsilon} + 3\ \epsilon n$

An optimal MST algorithm: Decision Trees

- Definition: A MST decision tree is a rooted binary tree having an edge-weigth comparison associated with each internal node. An MST decision tree for a graph G is said to be correct if the edge-weight comparisons encountered on any path from the root to a leaf uniquely identify the spanning tree at that leaf as the MST.

- A decision tree is said to be optimal, if it is correct and there's no other correct decision tree with lesser depth.

An optimal MST algorithm: Decision Trees

- There are $2^{\binom{r}{2}}$ graphs with r vertices.
- A DJP algorithm uses no more than r(r-1) comparisons to compute the MST on any graph with r vertices. Hence decision tree has fewer than $\sum_{i=0}^{r^2} 2^i \leq 2^{r^2}$ inner nodes
- For every node there are $\leq r^4$ possibilities (two edges = four vertices have to be selected)
- There are $\leq (r^4)^{2^{r^2}} = r^{2^2 * 2^{r^2}} = r^{2^{r^2+2}}$ decision trees

An optimal MST algorithm: Decision Trees

- Place all possible permutations of a graph at the root. Apply decision tree
- If for each leaf all permutations sharing that leaf agree on the MSF, the decision tree is correct! This process takes no longer than $(r^2+1)!$
- Total time rquired to find optimal decision tree for all graphs is bounded by $2^{r^2} * r^{2^{r^2+2}} * (r^2+1)! \leq 2^{2^{r^2+o(r)}}$
- Setting $r = log^3\|V\|$ allows to precompute all decision trees in $o(\|V\|)$

An optimal MST algorithm: Some theorems

- If T is a tree of MSF edges, you can contract T into one single vertex. The MSF of the resulting graph $G_{contracted(T)}$ plus T is $MSF(G)$
- $G \Uparrow M$ is the graph derived from G by raising the weight of each edge of M by arbitrary amounts.
- Let M be a set of edges in a graph G. If C is a subgraph of G that is DJP-contractible with respect to $G \Uparrow M$, then MSF(G) is a subset of $MSF(C) \cup MSF(G_{contracted(C)} - M_C) \cup M_C$ where M_C is the set of edges with exact one endpoint in C.

An optimal MST algorithm: Algorithm

Idea:

1. Find DJP-contractible subgraphs $C_1, C_2, \ldots$ using Soft Heaps (algorithm resembles Kruskal's). Maximum number of vertices of subgraphs is bounded by $log^3\|V\|$.

2. Find MSF of the subgraphs using precomputed decision trees.

3. Find MSF of $G_{contracted(C)} - M_C) \cup M_C$. For this graph is dense e.g. Algorithm of Fredman and Tarjan is very efficient.

4. Find some MSF edges of the merge of the substep results using a finite number of Borúvka steps and recurse on graph contracting these edges.

Lessons learned

- Mix different algorithms for optimal performance
- Reduce data structure size ’divide et impera’ (heap size)
- Precalculate values used often (verficiation, path compression)
- For different classes of input data (dense graphs, sparse graphs) use different types of algorithms
- Loose invariants for data structures (Fibonacci heaps, heaps instead of sorted lists)

Lessons learned