Lecture 2 (2011-10-13)**:**

Definition 2.1 (connected)**:**
A graph is called *connected* (zusammenhängend) if there exists a [s,t]-Path between all pairs of vertices $s, t \in V$.

Definition 2.2 (forrest, tree, spanning, forest problem, minimum spanning tree)**:**
A *forest* (Wald) is a graph that does not contain a cycle (Kreis). A connected forest is called a *tree* (Baum). A tree in a graph (as subgraph) is called *spanning* (aufspannend), if it contains all vertices.

Given a graph $G = (V, E)$ with edge weights $c_e \in \mathbb{R}$ for all $e \in E$, the task to find a forest $W \subset E$ such that $c(W) := \sum\limits_{e \in W}$ is maximal, is called the *Maximum Forest Problem* (Problem des maximalen Waldes). The task to find a tree $T \subset E$ which spans G and which weight $c(T)$ is minimal, is called the *Minimum Spanning Tree* (MST) problem (minimaler Spannbaum).

Lemma 2.3:
A tree $G = (V, E)$ with at leat 2 vertices has at least 2 vertices of degree 1.

Proof. Let v be arbitrary. Since G is connected, $deg(v) \geq 1$. Assume $deg(v) = 1$. So $\delta(v) = \{vw\}$. If $deg(w) = 1$, we found two vertices with $degree$ 1. If $deg(w) > 1$, there exist a neighbour of w, different from $v : u$. Now, again u has $degree$ 1 or higher. If we repeat this procedure we either find a vertix of degree 1 or find again *new* vertices. Hence, after at most $n - 1$ vertices we end up at a vertex of degree 1. Now, if $deg(v) \geq 2$, we do the same and find a vertex of degree 1, say w. Then repeat the above, staring from w to find a second vertex of degree 1. □

Corollary 2.4:
A tree $G = (V, E)$ with maximum degree Δ has at least Δ vertices of degree 1.

Lemma 2.5: (a) For every graph $G = (V, E)$ it holds that $2|E| = \sum\limits_{u \in V} deg(u)$

(b) for every tree $G = (v, E)$ it holds that $|E| = |V| - 1$.

Proof. (a) trivial

(b) Proof by induction. Clearly, if $|V| = 1$ or $|V| = 2$ it holds. Assumption: true for $n \geq 2$. Let G be a tree with $n + 1$ vertices. By Lemma 2.3, there exists a vertex $v \in G$ with $deg(v) = 1 . G - v = G[V \setminus \{v\}$ is a tree again with n vertices and thus $|E(G - v)| = V(G - v)| - 1$. Since G differs by one vertex and one edge from $G - v$, the claim holds got G as well.

□

Lemma 2.6:
If $G = (V, E)$ whith $|V| \geq 2$ has $|E| < |V| - 1, G$ is not connected.

Algorithm MST

$min_{x \in X} = -max_{x \in X} - c(x)$ ~~maximal forest~~
X spanning trees
$min_{x \in X} + (n-1)D = -max_{x \in X} - c(x)(n-1)D = max_{x \in X} \sum \underbrace{D - C_{ij} x_{ij}}_{\geq 0 if D \geq max_{ij \in E} c_{ij}}$

Theorem 2.7:
Kruskal's Algorithm returns the optimal solution.

Proof. Let T be Kruskal's tree and assume there exists a tree T' with $c(T') < c(T)$. Then there exist an edge $e' \in T' \setminus T$. Then $T \cup \{e'\}$ contains a cycle $\{e_1, e_2, \dots, e_k, e'\}$. Let $c_f = max_{i=1,\dots k} c_{l_i}$. At the moment Kruskal chooses edge f, edge e' cannot be added yet and therefore $c(e') \geq c(f)$. Now exchange e' by f in T'. Hence the number of differences beetween T' and T is reduced by one, $C(T'_{new}) \leq c(T') < c(T)$. Repeating the procedure results in $c(T) \leq \dots < c(T)$, a contradiction. □

Lecture 3 (2011-10-17)**:**

Definition 2.7(+1):
The *running time of algorithms* (Laufzeit) of an algorithm is measured by the number of operations needed in worst case of a function of the input size. We use the $O(\cdot)$ notation (Big-O-notation) ot focus on the most important factor of the running time, ignoring constants and smaller factors.

Example 2.7(+2):
If the running time is $3n \cdot \log n + 26n$, the algorithm runs in $O(n \cdot \log n)$. If the running time is $3n \cdot \log n + 25n^2$, the algorirthm runs in $O(n^2)$.

For graph Problems, the running is expressed in the number of vertices $n = |V|$ and the number of edges $m = |E|$. Sometimes m is approximated by n^2.

Example 2.7(+3) (Kruskal's Algorithm)**:**
First, the edged are sorted according to nondecreasing weights. This can be done in $O(m \cdot \log m)$. Next, we repeatedly select an edge or reject its selection until $n-1$ edges are selected. Since the last selected edge might be after m steps, this routine is performed at most $O(m)$ times.

Checking whether the end nodes of $\{u, v\}$ are already in the same tree can be done in constant time, if we label the vertices of the trees selected so far: $r(u) = \#trees\,containing\,u$. If $r(u) \neq r(v)$, the trees are connected by $\{u, v\}$ to a new tree.

Without going into details, the resetting of labels in one of the old trees, can be done $O(\log n)$ on average. Since this update has to be done at most $n-1$ times, it takes $O(n \cdot \log n)$.

Overall, Kruskal runs in

$$O(n \log m + m + n \cdot \log n) = O(m \cdot \log m) = O(m \cdot \log n^2) = O(m \cdot \log n)$$

Definition 2.7(+4) (Shortest paths in acyclic digraphs)**:**
A directed graph (digraph) $D = (V, A)$ is called *acyclic* (azyklisch) if it does not contain any *directed cycles*, i.e. a *chain* (Kette) $(v_0, a_1, v_1, a_2, v_2, \dots a_k, v_k)$, $k \geq 0$, with $a_i(v_{i-1}, v_i) \in A$ and $v_k = v_0$. In particular, D does not contain *antiparallel* arcs: if $(u, v) \in A$, $(v, u) \notin A$. With $\delta^+_D(v)$ we denote the arcs leaving vertex v:

$$\delta^+_D(v) = \{(u, w) \in A : u = v\}$$

similarly:

$$\delta^-_D(v) = \{(u, w) \in A : w = v\}$$

are the arcs entering v.

The *outdegree* of v is $\deg^+_D(v) = |\delta^+(v)|$ (assuming simple digraph)

The *indegree* of v is $\deg^-_D(v) = |\delta^-(v)|$

Definition 3.1:
The *shortest path* problem in a acyclic digraph is, given an acyclic digraph $D = (V, A)$, a length function $C : A \to \mathbb{R}$ and two vertices $s, t \in V$, find a $[s, t]$-path of minimal length.

Question 1:
Does there exist a $[s, t]$-path at all?

Theorem 3.2:
A digraph $D = (V, A)$ is acyclic, if and only if there exists a permutaion $\sigma : V \to \{1, .., n\}$ of the vertices such that $\deg^-_{D[v_1,...,v_n]}(v_i) = 0$ for all $i = 1, \dots, n$ with $v_i = \sigma^{-1}(i)$.

Proof. By induction:

For digraph with $|V| = 1$, the statement is true. Assume the statement is true for all digraphs with $|V| \leq n$ and consider $D = (V, A)$ acyclic with $n+1$ vertices. If there does not exist a vertex with $\deg^-_D(v) = 0$, a directed cycle can be detected by following incoming arcs backwards until a vertex is repeated, a contradiction regarding the acyclic property of D.

Hence, let v be a vertex with $\deg^-_D(v) = 0$. Set $v_1 = v$. The digraph $D - v_1$ has n vertices and is acyclic, and thus has a permutation $(v_2, \dots, v_{n+1})$ with

$$\deg^-_{D[v_i,...,v_{n+1}]}(v_i) = 0 \qquad \forall i = 2, \dots, n+1$$

Now, $(v_1, \dots, v_{n+1})$ is a permutation fulfilling the condition.

In reverse, if there exists a permutation $(v1, \dots, v_{n+1})$, $\deg^-_D(v_1) = 0$ and there cannot exist a directed cycle containing v_1. By induction, neither cycles containing $v_i, i = 2, \dots, n+1$ exist. ☐

Theorem 3.3:
A $[s, t]$-path exists in a acyclic Digraph $D = (V, A)$ if and only if in all permutations $\sigma : V \to \{1, \dots, n\}$ with $\deg^-_{D[v_i,...,v_n]}(v_i) = 0$ for all $i = 1, \dots, n$, it holds that $\sigma(s) < \sigma(t)$.

Proof. Assume there exists a permutation σ with $\sigma(s) > \sigma(t)$. Since outgoing arcs only go to higher ordered vertices, there does not exist a path from s to t in D.

In reverse, if there does not exist a path from s to t, we order all vertices with paths to t first, followed by t and s afterwards. ☐

Question 2:
How do we find the shortest $[s, t]$-path if it exists?

To simplify notation, let $V = \{1, \dots, n\}, s = 1, t = n$ and $(i, j) \in A \Rightarrow i < j$. Let $D(i)$ be the distance from i to n and $NEXT(i)$ be the next vertex on the shortest path from i to n.

Bellmann's Algorithm

(a) $D(i) = \{\infty : i < n and NEXT(i) = NIL, 0 : i = n\}$

(b) FOR $i = n - 1$ DOWNTO 1 DO

(c) $\qquad D(i) = \min_{j=i+1,...,n} \{D(j) + c(i, j)\}$ with $c(i, j) = \infty$ if $(i, j) \notin A$

(d) $NEXT(i) = argmin_{j=i+1,...,n}\{D(j) + c(i,j)\}$

Theorem 3.4:
Bellmann's Algorithm is correct and runs in $O(m+n)$ time.

Proof. Every path from 1 to n passes through vertices of increasing ID. Assume there exists a path $(a_1, \ldots, a_k)$ with $\sum_{i=1}^{k} c(a_i) < D(1)$. Let $a_1 = (1, j_1)$. Since $D(1) \leq c(a_1) + D(j_1)$, it should hold that

$$\sum_{i=2}^{2} c(a_i) < D(j_1)$$

But $D(j_1) \leq c(a_2) + D(j_2)$ with $a_2 = (j_1, j_2)$, etc.
In the end, $c(a_k) < D(j_{k-1})$ but $D(j_{k-1}) \leq c(a_k) + D(n) = c(a_k)$, contradiction. $\square$