**Hierarchies and trees**

• We talked about XML and HTML, which are hierarchical data structures.

• A hierarchy can consist of different elements, but also of the same type

an element can be repeated in the hierarchy.

• For example, the content of an HTML table cell can be a new table:

HTML->BODY->TABLE->TR->TD->TABLE->...

*• If you look at tags abstractly, all elements of the hierarchy are the same*

*type: tags, which can contain new tags.*

• A tree corresponds to this structure.

• To represent a tree in a programming language, it is necessary to define a node element (or top element).

• A subpart of such an element is five of the same type as the followingto the elements.

**Node / vertex (node)**

• A node contains some dataset and in addition references to the next ones of the same kind to nodes.

• The first node in the structure is the root element (root), nodes without descendants are leaves and the branches between them.

• Each subsequent layer increases the capacity of the structure exponentially.

• The tree must run out at some point. So the references must allow a situation where this node has no descendants.

• The possible number of references sets limits on what can be represented in the dataset.

• If there are at most two references, it is a binary tree.

• If there is at most one reference, we don't have a tree, like that the data structure is called a chained list.

• In XML, the number of references is not directly limited, the limiting factor is memory capacity

**Sorted binary tree**

• A binary tree is sorted if the following rules are followed:

• A key value is stored at each vertex.

• Values smaller than the key value can only go to the left branch.

• Values greater than the key value can only go to the right branch.

• The advantage of a sorted tree is that you can easily add elements to it and the tree remains sorted.

• The disadvantage of the binary tree is that it can easily become "sloppy",the left and right branches of the root element are very different in size.

• In this case, tree balancing would be required and binary tree balancing would not be easy.

**Tree properties**

• Searching in a sorted tree is linearly dependent on the depth of the tree.

• The depth of the binary tree, in turn, is roughly logarithmic of the number of elements.

• So tree search is more or less equivalent to binary search from a sorted array.

• Keeping a sorted tree sorted is slightly better complexity than best sorting algorithms.

• The disadvantage of the tree is that it is scattered along the memory. Memory access operations are expensive and getting more expensive. Therefore, there is no array to sort the point of building a tree since memory operations zero out all win which a tree structure compared to an array gives.

**N tree**

• In a binary tree, a node has one key value and up to two children.

• A generalized variant of this is an N-tree, the vertex has up to N descendants and the vertex is kept N-1 key values.

• An N-tree reduces the number of vertices that need to be traversed, but now each vertex has also more key values.

• The advantage of an N-tree is that it is easier to balance.

• B-tree (B-tree) is a tree whose node has at least 2 key values (so at least 3 offspring). Adding to the tree is done only at leaf vertices and if more key values accumulate at the top of the leaf than can fit, then the tree will be rebalanced.

• B-trees are widely used in databases for table indexes for storage.

**Traversing a whole tree**

• Suppose we want to print out the contents of an entire binary tree, for example.

• For each tree vertex, we are interested in the vertex to the left of the vertex, the vertex itself and peak to the right.

• But vertices on the left and right can in turn have descendants.

• Therefore, the entire branch to the left of the top must be traversed first, and only then return to the top.

• This task generally requires a stack automaton (push-down automaton).

• To realize a stack automaton, it is necessary to supplement the finite state machine with a stack where vertices and also machine states can be saved.

• An easier option is to use recursion. In recursion, C builds the compiler itself gives us a stack.

**Node / Vertex in C**

struct node{ // binary tree with resolved pointers

int keyvalue;

struct node \*left, \*right;

};

struct node2{ // binary tree with resolved indices

int keyvalue;

int leftidx, rightidx;

};

struct node3{ /\* n-tree resolved by convention or array of pointers\*/

char \*value;

int childcount;

struct node3 \*children; // or also \*\*children;

};

**Saving a Tree (Serialization**)

• If the tree is resolved by index and the max number of descendants is fixed, then it is an array that is easy to store in both binary and text form.

• If the tree is resolved with pointers and/or the number of descendants is not limited, then it is spreading the tree along the memory and storing it in binary form is very inconvenient.

• XML and HTML allow such a tree to be saved in text form. A node inside are all his offspring and offspring's offspring.

• In order to save the tree, it is necessary to go through it in its entirety, recursion again demanding task.

**Decision diagram (decision diagram)**

• A decision diagram is one type of static tree, although they are rather called charts into graphs.

• At each vertex of the decision diagram, the condition associated with the vertex is checked and it as a result, one of the possible branches is selected and moved to the next one to the top.

• There are few "leaves" or end states compared to the tree.

• Decision diagrams can have cycles where one branch from the top leads back

somewhere in the beginning. As a rule, the structure of the tree does not allow cycles.

• One example of a decision diagram is dialogue trees in computer games.

The player makes one choice, as a result of which some additional actions are performed and moves to the next vertex where the player can make more choices.

**Binary decision diagrams**

• A variant of the decision diagram where there are only two choices.

• Binary decision diagrams can be used to describe binary logic functions and, in this regard, also the functionality of the hardware.

• At the peak, the signal value is checked and the next peak is selected until it is reached end which is a constant 0 or a constant 1