Classification of data structures

In case of a database model of a complex problem we not only have to store the attributes as data, but the logic relationships between them also have to be visualized. To put this into practice a complex data structure has to be formed which later we can make operations with efficiently, including the support of the chosen programming language.

Complex data structures can be classified by these aspects:

1. According to the type of the components the data structure can be

- **homogeneous/solid:** if all the components are from the same type, or
- **heterogeneous:** if the components are not from the same type.

2. According to operation executed with the data structure

- **without a structure:** there are not any relationships between the elements of the data structure (e.g.: set)
- **associative:** there are no essential relationships between the elements of the data structure, the elements are individually addressable (e.g.: array)
- **sequential:** in a data structure every element – except the first and the last ones – are attainable exactly from another element and from every element exactly one element can be attainable. Only the two highlighted elements are the exceptions, because there is no element of the data structure which the first or the last element would be attainable (e.g.: simplelist).
- **hierarchical:** in the data structure there is a highlighted element (the root) which giving equals the giving of the data structure. Every element, except the root can be attainable from exactly one other element, but from one element arbitrary (finite) number of additional elements can be attainable. Exceptions to this are the endpoints of the data structure (e.g.: binary tree).
- **lattice/net:** the elements of the data structure can be attainable from several other elements and from a given element more further elements can be attainable (e.g.: graph).

3. According to the number of the elements

- **static:** the number of elements of the data structure is constant. The size of the used space does not change during the operations.
- **dynamic:** the number of elements of the data structure is finite too, because the space we have limits the possibility of expansion. At the same time the number of elements can change during the operations. The empty state of the data structure is interpreted and when the space, reserved for this purpose, run out, then we say the data structure is full. The speciality of this type of data structure is that in case of an augment of a new element we have to reserve a proper space for that element and if an element goes useless we have to make the free space ready to reverse again.

4. According to the location where the data elements are stored

- **continual:** in case of a representation, that smallest, contiguous space, this includes all elements of the data structure, and no other elements.
- **diffused/linked:** in case of a representation, the elements of the data structure can be on the arbitrary part of the space, the information needed to reach certain elements are stored by other elements.