9.8 Node Arrays (node_array)

1. Definition

An instance A of the parameterized data type *node_array*<E> is a partial mapping from the node set of a graph G to the set of variables of type E, called the element type of the array. The domain I of A is called the index set of A and $A(v)$ is called the element at position v. A is said to be valid for all nodes in I. The array access operator $A[v]$ checks its precondition (A must be valid for v). The check can be turned off by compiling with the flag `-DLEDA_CHECKING_OFF`.

#include < *LEDA/node_array.h* >

2. Creation

node_array<E> A; creates an instance A of type *node_array*<E> with empty index set.

node_array<E> A(*const graph_t*& G);
creates an instance A of type *node_array*<E> and initializes the index set of A to the current node set of graph G.

node_array<E> A(*const graph_t*& G, E x);
creates an instance A of type *node_array*<E>, sets the index set of A to the current node set of graph G and initializes $A(v)$ with x for all nodes v of G.

node_array<E> A(*const graph_t*& G, *int* n, E x);
creates an instance A of type *node_array*<E> valid for up to n nodes of graph G and initializes $A(v)$ with x for all nodes v of G.
Precondition: $n \geq |V|$.
A is also valid for the next $n - |V|$ nodes added to G.

3. Operations

const graph_t& A.get_graph( ) returns a reference to the graph of A.

E& A[*node* v] returns the variable $A(v)$.
Precondition: A must be valid for v.

void A.init(*const graph_t*& G) sets the index set I of A to the node set of G, i.e., makes A valid for all nodes of G.

void A.init(*const graph_t*& G, E x)
makes A valid for all nodes of G and sets $A(v) = x$ for all nodes v of G.

void A.init(*const graph_t*& G, *int* n, E x)

makes A valid for at most n nodes of G and sets $A(v) = x$ for all nodes v of G.
Precondition: $n \geq |V|$.
A is also valid for the next $n - |V|$ nodes added to G.

bool A.use_node_data(*const graph_t*& G)

use free data slots in the nodes of G (if available) for storing the entries of A. If no free data slot is available in G, an ordinary *node_array*<E>is created. The number of additional data slots in the nodes and edges of a graph can be specified in the *graph*::*graph*(*int n_slots*, *int e_slots*) constructor. The result is *true* if a free slot is available and *false* otherwise.

bool A.use_node_data(*const graph_t*& G, E x)

use free data slots in the nodes of G (if available) for storing the entries of A and initializes $A(v) = x$ for all nodes v of G. If no free data slot is available in G, an ordinary *node_array*<E> is created. The number of additional data slots in the nodes and edges of a graph can be specified in the *graph*::*graph*(*int n_slots*, *int e_slots*) constructor. The result is *true* if a free slot is available and *false* otherwise.

4. Implementation

Node arrays for a graph G are implemented by C++vectors and an internal numbering of the nodes and edges of G. The access operation takes constant time, *init* takes time $O(n)$, where n is the number of nodes in G. The space requirement is $O(n)$.

Remark: A node array is only valid for a bounded number of nodes of G. This number is either the number of nodes of G at the moment of creation of the array or it is explicitely set by the user. Dynamic node arrays can be realized by node maps (cf. section 9.11).

9.9 Edge Arrays (edge_array)

1. Definition

An instance A of the parameterized data type *edge_array*<E> is a partial mapping from the edge set of a graph G to the set of variables of type E, called the element type of the array. The domain I of A is called the index set of A and $A(e)$ is called the element at position e. A is said to be valid for all edges in I. The array access operator $A[e]$ checks its precondition (A must be valid for e). The check can be turned off by compiling with the flag `-DLEDA_CHECKING_OFF`.

#include < *LEDA/edge_array.h* >

2. Creation

edge_array<E> A; creates an instance A of type *edge_array*<E> with empty index set.

edge_array<E> A(*const graph_t*& G);
creates an instance A of type *edge_array*<E> and initializes the index set of A to be the current edge set of graph G.

edge_array<E> A(*const graph_t*& G, E x);
creates an instance A of type *edge_array*<E>, sets the index set of A to the current edge set of graph G and initializes $A(v)$ with x for all edges v of G.

edge_array<E> A(*const graph_t*& G, *int* n, E x);
creates an instance A of type *edge_array*<E> valid for up to n edges of graph G and initializes $A(e)$ with x for all edges e of G.
Precondition: $n \geq |E|$.
A is also valid for the next $n - |E|$ edges added to G.

3. Operations

const graph_t& A.get_graph( ) returns a reference to the graph of A.

E& A[*edge* e] returns the variable $A(e)$.
Precondition: A must be valid for e.

void A.init(*const graph_t*& G) sets the index set I of A to the edge set of G, i.e., makes A valid for all edges of G.

void A.init(*const graph_t*& G, E x)
makes A valid for all edges of G and sets $A(e) = x$ for all edges e of G.

void *A*.init(*const graph_t*& *G*, *int n*, *E x*)

makes A valid for at most n edges of G and sets $A(e) = x$ for all edges e of G.
Precondition: $n \geq |E|$.
A is also valid for the next $n - |E|$ edges added to G.

bool *A*.use_edge_data(*const graph_t*& *G*, *E x*)

use free data slots in the edges of G (if available) for storing the entries of A. The number of additional data slots in the nodes and edges of a graph can be specified in the *graph*::*graph*(*int n_slots*, *int e_slots*) constructor. The result is *true* if a free slot is available and *false* otherwise.

4. Implementation

Edge arrays for a graph G are implemented by C++vectors and an internal numbering of the nodes and edges of G. The access operation takes constant time, *init* takes time $O(n)$, where n is the number of edges in G. The space requirement is $O(n)$.

Remark: An edge array is only valid for a bounded number of edges of G. This number is either the number of edges of G at the moment of creation of the array or it is explicitely set by the user. Dynamic edge arrays can be realized by edge maps (cf. section 9.12).