Template Class Graph

Table of contents

- Defined in File graph.hpp

Inheritance Relationships

Derived Type

- `public holoscan::FlowGraph< NodeT, EdgeDataElementT >` (Template Class FlowGraph)

Class Documentation

template<typename NodeT = OperatorNodeType, typename EdgeDataElementT = OperatorEdgeDataElementType>
class Graph

Abstract base class for all graphs.

Subclassed by holoscan::FlowGraph< NodeT, EdgeDataElementT >

Public Types

using NodeType = NodeT

using NodePredicate = std::function<bool(const NodeT&)>

using EdgeDataElementType = EdgeDataElementT

using EdgeDataType = std::shared_ptr<EdgeDataElementT>

Public Functions

Graph() = default

virtual ~Graph() = default

virtual void add_node(const NodeT &node) = 0

Add the node to the graph.

Parameters

node – The node to add.

virtual void add_flow(const NodeType &node_u, const NodeType &node_v, const EdgeDataType &port_map) = 0

Add an edge to the graph.

Parameters

- **node_u** – A source node.
- **node_v** – A destination node.
- **port_map** – A map from the source node's port name to the destination node's port name(s).

virtual std::optional<EdgeDataType> get_port_map(const NodeType &node_u, const NodeType &node_v) = 0

Get a mapping from the source node's port name to the destination node's port name(s).

Parameters

- **node_u** – A source node.
- **node_v** – A destination node.

Returns

A map from the source node's port name to the destination node's port name(s).

inline virtual bool is_empty()

Check if the graph is empty.

Returns

true if the graph is empty. Otherwise, false.

virtual bool is_root(const NodeType &node) = 0

Check if the node is a root node.

Parameters

node – A node in the graph.

Returns

true if the node is a root node.

virtual bool is_user_defined_root(const NodeType &node) = 0

Check if the node is a user-defined root node. A user-defined root is the first node that is added to the graph.

Parameters

node – A node in the graph.

Returns

true if the node is a user-defined root node.

virtual bool is_leaf(const NodeType &node) = 0

Check if the node is a leaf node.

Parameters

node – A node in the graph.

Returns

true if the node is a leaf node.

virtual std::vector<NodeType> has_cycle() = 0

Returns a vector of root nodes of the cycles if the graph has cycle(s). Otherwise, an empty vector is returned.

Returns

Returns a vector of root nodes of cycles.

virtual std::vector<NodeType> get_root_nodes() = 0

Get all root nodes.

Returns

A vector of root nodes.

virtual std::vector<NodeType> get_nodes() = 0

Get all nodes.

The order of the nodes is not guaranteed.

Returns

A vector of all nodes.

virtual std::vector<NodeType> get_next_nodes(const NodeType &node) = 0

Get the next nodes of the given node.

Parameters

node – A node in the graph.

Returns

A vector of next nodes.

virtual NodeType find_node(const NodePredicate &pred) = 0

Find a node in the graph that satisfies the given predicate.

Parameters

pred – A predicate.

Returns

The node if found, otherwise nullptr.

virtual NodeType find_node(const NodeType &node) = 0

Find a node in the graph that is equal to the given node.

Parameters

node – The node to find.

Returns

The node in the graph if found, otherwise nullptr.

virtual NodeType find_node(std::string name) = 0

Find a node in the graph whose name is equal to the given name.

Parameters

name – The name to find.

Returns

The node in the graph if found, otherwise nullptr.

virtual std::vector<NodeType> get_previous_nodes(const NodeType &node) = 0

Get the previous nodes of the given node.

Parameters

op – A node in the graph.

Returns

A vector of next nodes.

inline virtual void context(void *context)

Set the context.

Parameters

context – The context.

inline virtual void *context()

Get the context.

Returns

The context.

Protected Attributes

void *context_ = nullptr

The context.