ckpttncpp Documentation

Release 0.2.3

Wai-Shing Luk

Jul 02, 2021

CONTENTS

Welcome to the CkPttnCpp website.

Tip: The webpage you are viewing used the `html_theme` of `bootstrap` in `conf.py`.

CHAPTER

ONE

LIBRARY API

Welcome to the developer reference to Exhale Companion. The code being documented here is largely meaningless and was only created to test various corner cases e.g. nested namespaces and the like.

Note: The text you are currently reading was fed to `exhale_args` using the `afterTitleDescription` key. Full reStructuredText syntax can be used.

Tip: Sphinx / Exhale support unicode! You're `conf.py` already has it's encoding declared as `# -*- coding: utf-8 -*-` **by default**. If you want to pass Unicode strings into Exhale, simply prefix them with a `u` e.g. `u""` (of course you would actually do this because you are writing with åçćëñtß or non-English).

1.1 Class Hierarchy

1.2 File Hierarchy

Below the hierarchies comes the full API listing.

1. The text you are currently reading is provided by `afterHierarchyDescription`.
2. The Title of the next section *just below this* normally defaults to `Full API`, but the title was changed by providing an argument to `fullApiSubSectionTitle`.
3. You can control the number of bullet points for each linked item on the remainder of the page using `fullToctreeMaxDepth`.

1.3 Custom Full API SubSection Title

1.3.1 Namespaces

Namespace fun

Page Contents

Classes

Functions

Namespace py

Page Contents

Classes

Functions

- *Template Function py::len(const dict<Key, T>&)*
- *Template Function py::len(const set<Key>&)*
- *Template Function py::operator<(const Key&, const set<Key>&)*
- *Template Function py::operator<(const Key&, const dict<Key, T>&)*
- *Template Function py::range(T)*
- *Template Function py::range(T, T)*

Namespace ranges

Namespace xn

Page Contents

Detailed Description

View Classes provide node, edge and degree "views" of a graph. Views for nodes, edges and degree are provided for all base graph classes. A view means a read-only object that is quick to create, automatically updated when the graph changes, and provides basic access like `n : V`, `for n : V`, `V[n]` and sometimes set operations. The views are read-only iterable containers that are updated as the graph is updated. As with dicts, the graph should not be updated while (iterating through the view. Views can be iterated multiple times. Edge and Node views also allow data attribute lookup. The resulting attribute dict is writable as `G.edges[3, 4]["color"]="red"` Degree views allow lookup of degree values for single nodes. Weighted degree is supported with the `weight` argument. *Template Class NodeView*

V = G.nodes (or *V = G.nodes()*) allows *len(V)*, *n : V*, set operations e.g. "G.nodes & H.nodes", and *dd = G.nodes[n]*, where *dd* is the node data dict. Iteration is over the nodes by default.

NodeDataView

To iterate over (node, data) pairs, use arguments to *G.nodes()* to create a DataView e.g. *DV = G.nodes(data="color", default="red")*. The DataView iterates as *for n, color : DV* and allows *(n, "red") : DV*. Using *DV = G.nodes(data=true)*, the DataViews use the full datadict : writeable form also allowing contain testing as *(n, {"color": "red"}) : VD*. DataViews allow set operations when data attributes are hashable.

DegreeView

V = G.degree allows iteration over (node, degree) pairs as well as lookup: *deg=V[n]*. There are many flavors of DegreeView for In/Out/Directed/Multi. For Directed Graphs, *G.degree* counts both : and out going edges. *G.out_degree* && *G.in_degree* count only specific directions. Weighted degree using edge data attributes is provide via *V = G.degree(weight="attr_name")* where any string with the attribute name can be used. *weight=None* is the default. No set operations are implemented for degrees, use NodeView.

The argument *nbunch* restricts iteration to nodes : nbunch. The DegreeView can still lookup any node even if (nbunch is specified.

Template Class EdgeView

V = G.edges or *V = G.edges()* allows iteration over edges as well as *e : V*, set operations and edge data lookup *dd = G.edges[2, 3]*. Iteration is over 2-tuples *(u, v)* for Graph/DiGraph. For multigraphs edges 3-tuples *(u, v, key)* are the default but 2-tuples can be obtained via *V = G.edges(keys=false)*.

Set operations for directed graphs treat the edges as a set of 2-tuples. For undirected graphs, 2-tuples are not a unique representation of edges. So long as the set being compared to contains unique representations of its edges, the set operations will act as expected. If the other set contains both *(0, 1)* and *(1, 0)* however, the result of set operations may contain both representations of the same edge.

EdgeDataView

Edge data can be reported using an EdgeDataView typically created by calling an EdgeView: *DV = G.edges(data="weight", default=1)*. The EdgeDataView allows iteration over edge tuples, membership checking but no set operations.

Iteration depends on *data* and *default* and for multigraph *keys* If *data == false* (the default) then iterate over 2-tuples *(u, v)*. If *data is true* iterate over 3-tuples *(u, v, datadict)*. Otherwise iterate over *(u, v, datadict.get(data, default))*. For Multigraphs, if (*keys is true*, replace *u, v* with *u, v, key* to create 3-tuples and 4-tuples.

The argument *nbunch* restricts edges to those incident to nodes : nbunch. Exceptions Base exceptions and errors for XNetwork.

Classes

- *Struct AmbiguousSolution*
- *Struct ExceededMaxIterations*
- *Struct HasACycle*
- *Struct NodeNotFound*
- *Struct object*
- *Struct XNetworkAlgorithmError*
- *Struct XNetworkError*
- *Struct XNetworkException*
- *Struct XNetworkNoCycle*
- *Struct XNetworkNoPath*
- *Struct XNetworkNotImplemented*
- *Struct XNetworkPointlessConcept*
- *Struct XNetworkUnbounded*
- *Struct XNetworkUnfeasible*
- *Template Class AtlasView*
- *Template Class EdgeView*
- *Template Class grAdaptor*
- *Template Class Graph*

Typedefs

Variables

1.3.2 Classes and Structs

Template Struct Fraction

- Defined in file_py2cpp_fractions-new.hpp

Page Contents

Inheritance Relationships

Base Type

- `public boost::totally_ordered< Fraction< Z >, boost::totally_ordered2< Fraction< Z >, Z, boost::multipliable2< Fraction< Z >, Z, boost::dividable2< Fraction< Z >, Z > > > >`

Template Parameter Order

1. `typename Z`

Struct Documentation

template<typename **Z**>

struct **fun::Fraction** : public boost::totally_ordered<*Fraction*<*Z*>, boost::totally_ordered2<*Fraction*<*Z*>, *Z*, boost::multipliable2<*Fraction*<*Z*>, *Z*, boost::dividable2<*Fraction*<*Z*>, *Z*>>>>

Public Types

using **_Self** = *Fraction*<*Z*>

Public Functions

inline constexpr **Fraction**(const *Z* &numerator, const *Z* &denominator)
Construct a new *Fraction* object.

Parameters

- **numerator** – **[in]**
- **denominator** – **[in]**

inline explicit constexpr **Fraction**(const *Z* &numerator)
Construct a new *Fraction* object.

Parameters **numerator** – **[in]**

constexpr **Fraction**() = default
Construct a new *Fraction* object.

inline constexpr const *Z* &**numerator**() const

Returns const Z&

inline constexpr const *Z* &**denominator**() const

Returns const Z&

inline constexpr *_Self* **abs**() const

Returns _Self

inline constexpr void **reciprocal**()

inline constexpr *_Self* **operator-**() const

Returns _Self

inline constexpr *_Self* **operator+**(const *_Self* &frac) const

Parameters **frac** – **[in]**

Returns _Self

inline constexpr *_Self* **operator-**(const *_Self* &frac) const

Parameters **frac** – **[in]**

Returns _Self

inline constexpr *_Self* **operator***(const *_Self* &frac) const

Parameters **frac** – **[in]**

Returns _Self

inline constexpr *_Self* **operator/**(*_Self* frac) const

Parameters **frac** – **[in]**

Returns _Self

inline constexpr *_Self* **operator+**(const *Z* &i) const

Parameters **i** – **[in]**

Returns _Self

inline constexpr *_Self* **operator-**(const *Z* &i) const

Parameters **i** – **[in]**

Returns _Self

inline constexpr *_Self* **operator***(const *Z* &i) const

Parameters **i** – **[in]**

Returns _Self

inline constexpr *_Self* **operator/**(const *Z* &i) const

Parameters **i** – **[in]**

Returns _Self

inline constexpr *_Self* **operator+=**(const *_Self* &frac)

Parameters **frac** – **[in]**

Returns _Self

inline constexpr *_Self* **operator-=**(const *_Self* &frac)

Parameters **frac** – **[in]**

Returns _Self

inline constexpr *_Self* **operator*=**(const *_Self* &frac)

Parameters **frac** – **[in]**

Returns _Self

inline constexpr *_Self* **operator/=**(const *_Self* &frac)

Parameters **frac** – **[in]**

Returns _Self

inline constexpr *_Self* **operator+=**(const *Z* &i)

Parameters **i** – **[in]**

Returns _Self

inline constexpr *_Self* **operator-=**(const *Z* &i)

Parameters **i** – **[in]**

Returns _Self

inline constexpr *_Self* **operator*=**(const *Z* &i)

Parameters **i** – **[in]**

Returns _Self

inline constexpr *_Self* **operator/=**(const *Z* &i)

Parameters **i** – **[in]**

Returns _Self

template<typename **U**>
inline constexpr auto **cmp**(const *Fraction*<*U*> &frac) const
Three way comparison.

Parameters **frac** – **[in]**

Returns auto

template<typename **U**>
inline constexpr bool **operator==**(const *Fraction*<*U*> &frac) const

Template Parameters **U** –

Parameters **frac** – **[in]**

Returns true

Returns false

template<typename **U**>
inline constexpr bool **operator!=**(const *Fraction*<*U*> &frac) const

Template Parameters **U** –

Parameters **frac** – **[in]**

Returns true

Returns false

template<typename **U**>
inline constexpr bool **operator<**(const *Fraction*<*U*> &frac) const

Template Parameters **U** –

Parameters **frac** – **[in]**

Returns true

Returns false

template<typename **U**>
inline constexpr bool **operator>**(const *Fraction*<*U*> &frac) const

Template Parameters **U** –

Parameters **frac** – **[in]**

Returns true

Returns false

template<typename **U**>
inline constexpr bool **operator<=**(const *Fraction*<*U*> &frac) const

Template Parameters **U** –

Parameters **frac** – **[in]**

Returns true

Returns false

template<typename **U**>
inline constexpr bool **operator>=**(const *Fraction*<*U*> &frac) const

Template Parameters **U** –

Parameters **frac** – **[in]**

Returns true

Returns false

inline constexpr auto **cmp**(const *Z* &c) const

Parameters **c** – **[in]**

Returns auto

inline constexpr bool **operator==**(const *Z* &c) const

Parameters **c** – **[in]**

Returns true

Returns false

inline constexpr bool **operator!=**(const *Z* &c) const

Parameters **c** – **[in]**

Returns true

Returns false

inline constexpr bool **operator<**(const *Z* &c) const

Parameters **c** – **[in]**

Returns true

Returns false

inline constexpr bool **operator>**(const *Z* &c) const

Parameters **c** – **[in]**

Returns true

Returns false

inline constexpr bool **operator<=**(const *Z* &c) const

Parameters **c** – **[in]**

Returns true

Returns false

inline constexpr bool **operator>=**(const *Z* &c) const

Parameters **c** – **[in]**

Returns true

Returns false

inline explicit constexpr **operator double**()

Returns double

inline constexpr **Fraction**(*Z* &&numerator, *Z* &&denominator) noexcept
Construct a new *Fraction* object.

Parameters

- **numerator** – **[in]**
- **denominator** – **[in]**

inline constexpr **Fraction**(const *Z* &numerator, const *Z* &denominator)
Construct a new *Fraction* object.

Parameters

- **numerator** – **[in]**
- **denominator** – **[in]**

inline constexpr void **normalize**()

inline explicit constexpr **Fraction**(*Z* &&numerator) noexcept
Construct a new *Fraction* object.

Parameters `numerator` – [in]

inline explicit constexpr **Fraction**(const *Z* &numerator)
Construct a new *Fraction* object.

Parameters `numerator` – [in]

inline constexpr auto **numerator**() const -> const *Z*&

Returns const Z&

inline constexpr auto **denominator**() const -> const *Z*&

Returns const Z&

inline constexpr auto **abs**() const -> *Fraction*

Returns *Fraction*

inline constexpr void **reciprocal**()

inline constexpr auto **operator-**() const -> *Fraction*

Returns *Fraction*

inline constexpr auto **operator+**(const *Fraction* &frac) const -> *Fraction*

Parameters `frac` – [in]

Returns *Fraction*

inline constexpr auto **operator-**(const *Fraction* &frac) const -> *Fraction*

Parameters `frac` – [in]

Returns *Fraction*

inline constexpr auto **operator***(const *Fraction* &frac) const -> *Fraction*

Parameters `frac` – [in]

Returns *Fraction*

inline constexpr auto **operator/**(*Fraction* frac) const -> *Fraction*

Parameters `frac` – [in]

Returns *Fraction*

inline constexpr auto **operator+**(const *Z* &i) const -> *Fraction*

Parameters `i` – [in]

Returns *Fraction*

inline constexpr auto **operator-**(const *Z* &i) const -> *Fraction*

Parameters `i` – **[in]**

Returns *Fraction*

inline constexpr auto **operator+=**(const *Fraction* &frac) -> *Fraction*&

Parameters

- `i` – **[in]**
- `i` – **[in]**
- `frac` – **[in]**

Returns *Fraction*

Returns *Fraction*

Returns *Fraction*

inline constexpr auto **operator-=**(const *Fraction* &frac) -> *Fraction*&

Parameters `frac` – **[in]**

Returns *Fraction*

inline constexpr auto **operator*=**(const *Fraction* &frac) -> *Fraction*&

Parameters `frac` – **[in]**

Returns *Fraction*

inline constexpr auto **operator/=**(const *Fraction* &frac) -> *Fraction*&

Parameters `frac` – **[in]**

Returns *Fraction*

inline constexpr auto **operator+=**(const *Z* &i) -> *Fraction*&

Parameters `i` – **[in]**

Returns *Fraction*

inline constexpr auto **operator-=**(const *Z* &i) -> *Fraction*&

Parameters `i` – **[in]**

Returns *Fraction*

inline constexpr auto **operator*=**(const *Z* &i) -> *Fraction*&

Parameters `i` – **[in]**

Returns *Fraction*

inline constexpr auto **operator/=**(const *Z* &i) -> *Fraction*&

Parameters `i` – **[in]**

Returns *Fraction*

template<typename **U**>
inline constexpr auto **cmp**(const *Fraction*<*U*> &frac) const
Three way comparison.

Parameters **frac** – **[in]**

Returns auto

inline constexpr auto **operator==**(const *Fraction*<*Z*> &rhs) const -> bool

inline constexpr auto **operator<**(const *Fraction*<*Z*> &rhs) const -> bool

inline constexpr auto **operator==**(const *Z* &rhs) const -> bool

inline constexpr auto **operator<**(const *Z* &rhs) const -> bool

inline constexpr auto **operator>**(const *Z* &rhs) const -> bool

Public Members

Z **_numerator**

Z **_denominator**

Friends

inline friend friend constexpr _Self operator+ (const Z &c, const _Self &frac)

Parameters

- **c** – **[in]**
- **frac** – **[in]**

Returns Fraction<Z>

inline friend friend constexpr _Self operator- (const Z &c, const _Self &frac)

Parameters

- **c** – **[in]**
- **frac** – **[in]**

Returns Fraction<Z>

inline friend friend constexpr _Self operator* (const Z &c, const _Self &frac)

Parameters

- **c** – **[in]**
- **frac** – **[in]**

Returns Fraction<Z>

`inline friend friend constexpr _Self operator+ (int &&c, const _Self &frac)`

Parameters

- **c** – [in]
- **frac** – [in]
- **c** – [in]
- **frac** – [in]
- **c** – [in]
- **frac** – [in]

Returns Fraction<Z>

Returns Fraction<Z>

Returns Fraction<Z>

`inline friend friend constexpr _Self operator- (int &&c, const _Self &frac)`

Parameters

- **c** – [in]
- **frac** – [in]

Returns Fraction<Z>

`inline friend friend constexpr _Self operator* (int &&c, const _Self &frac)`

Parameters

- **c** – [in]
- **frac** – [in]

Returns Fraction<Z>

`template<typename _Stream> inline friend friend _Stream & operator<< (_Stream &os, const _Self &frac)`

Template Parameters

- **_Stream** –
- **Z** –

Parameters

- **os** – [in]
- **frac** – [in]

Returns _Stream&

Template Struct MoveInfo

- Defined in file_ckpttncpp_netlist.hpp

Page Contents

Template Parameter Order

1. `typename Node`

Struct Documentation

template<typename **Node**>

struct **MoveInfo**

Public Members

Node **net**

Node **v**

std::uint8_t **fromPart**

std::uint8_t **toPart**

Template Struct MoveInfoV

- Defined in file_ckpttncpp_netlist.hpp

Page Contents

Template Parameter Order

1. `typename Node`

Struct Documentation

template<typename **Node**>

struct **MoveInfoV**

Public Members

Node **v**

std::uint8_t **fromPart**

std::uint8_t **toPart**

Template Struct Netlist

- Defined in file_ckpttncpp_netlist.hpp

Page Contents

Inheritance Relationships

Derived Type

- `public HierNetlist< graph_t >` (*Template Class HierNetlist*)

Template Parameter Order

1. `typename graph_t`

Struct Documentation

template<typename **graph_t**>

struct **Netlist**

Netlist.

Netlist is implemented by *xn::Graph*, which is a networkx-like graph.

Subclassed by *HierNetlist< graph_t >*

Public Types

using **nodeview_t** = typename *graph_t*::nodeview_t

using **node_t** = typename *graph_t*::node_t

using **index_t** = typename *nodeview_t*::key_type

Public Functions

Netlist(*graph_t* G, const *nodeview_t* &modules, const *nodeview_t* &nets)

Construct a new *Netlist* object.

Parameters

- **G** – **[in]**
- **module_list** – **[in]**
- **net_list** – **[in]**
- **module_fixed** – **[in]**
- **G** – **[in]**
- **modules** – **[in]**
- **nets** – **[in]**

Template Parameters

- **nodeview_t** –
- **nodemap_t** –

Netlist(*graph_t* G, uint32_t numModules, uint32_t numNets)

Construct a new *Netlist* object.

Parameters

- **G** – **[in]**
- **num_modules** – **[in]**
- **num_nets** – **[in]**

inline auto **begin**() const

inline auto **end**() const

inline auto **number_of_modules**() const -> size_t
Get the number of modules.

Returns size_t

inline auto **number_of_nets**() const -> size_t
Get the number of nets.

Returns size_t

inline auto **number_of_nodes**() const -> size_t
Get the number of nodes.

Returns size_t

inline auto **get_max_degree**() const -> size_t
Get the max degree.

Returns index_t

Returns size_t

inline auto **get_max_net_degree**() const -> size_t
Get the max net degree.

Returns index_t

inline auto **get_module_weight**(const *node_t* &v) const -> unsigned int
Get the module weight.

Parameters **v** – **[in]**

Returns int

inline auto **get_net_weight**(const *node_t*&) const -> int
Get the net weight.

Returns int

Public Members

graph_t **G**

nodeview_t **modules**

nodeview_t **nets**

size_t **num_modules** = {}

size_t **num_nets** = {}

size_t **num_pads** = 0U

size_t **max_degree** = {}

size_t **max_net_degree** = {}

std::vector<unsigned int> **module_weight**

bool **has_fixed_modules** = {}

py::*set*<*node_t*> **module_fixed**

Template Struct key_iterator

- Defined in file_py2cpp_py2cpp.hpp

Page Contents

Inheritance Relationships

Base Type

- `public Iter`

Template Parameter Order

1. `typename Iter`

Struct Documentation

template<typename **`Iter`**>

struct `py::key_iterator` : public *Iter*
Template Deduction Guide.

tparam Key

Public Functions

inline explicit **`key_iterator`**(*Iter* it)

inline auto **`operator*`**() const

inline auto **`operator++`**() -> *key_iterator*&

Struct robin::iterable_wrapper

- Defined in file_ckpttncpp_robin.hpp

Page Contents

Nested Relationships

This struct is a nested type of *Template Class robin*.

Struct Documentation

struct `robin::iterable_wrapper`

Public Functions

inline auto **begin**()

inline auto **end**()

Public Members

robin<T> ***rr**

T **fromPart**

Struct robin::iterator

- Defined in file_ckpttncpp_robin.hpp

Page Contents

Nested Relationships

This struct is a nested type of *Template Class robin*.

Struct Documentation

struct `robin::iterator`

Public Functions

inline auto **operator!=**(const *iterator* &other) const -> bool

inline auto **operator==**(const *iterator* &other) const -> bool

inline auto **operator++**() -> *iterator*&

inline auto **operator***() const -> const T&

Public Members

slnode ***cur**

Struct robin::slnode

- Defined in file_ckpttncpp_robin.hpp

Page Contents

Nested Relationships

This struct is a nested type of *Template Class robin*.

Struct Documentation

struct `robin::slnode`

Public Members

slnode ***next**

T **key**

Template Struct Snapshot

- Defined in file_ckpttncpp_netlist.hpp

Page Contents

Template Parameter Order

1. `typename Node`

Struct Documentation

template<typename **Node**>

struct **Snapshot**

Public Members

py::*set*<*Node*> **extern_nets**

py::*dict*<index_t, std::uint8_t> **extern_modules**

Struct AmbiguousSolution

- Defined in file_xnetwork_exception.hpp

Page Contents

Inheritance Relationships

Base Type

- `public xn::XNetworkException` (*Struct XNetworkException*)

Struct Documentation

struct `xn::AmbiguousSolution` : public xn::*XNetworkException*
Raised if (more than one valid solution exists for an intermediary step of an algorithm.

In the face of ambiguity, refuse the temptation to guess. This may occur, for example, when trying to determine the bipartite node sets in a disconnected bipartite graph when computing bipartite matchings.

Public Functions

inline explicit **AmbiguousSolution**(std::string_view msg)

Struct ExceededMaxIterations

- Defined in file_xnetwork_exception.hpp

Page Contents

Inheritance Relationships

Base Type

- `public xn::XNetworkException` (*Struct XNetworkException*)

Struct Documentation

struct `xn::ExceededMaxIterations` : public xn::*XNetworkException*
Raised if (a loop iterates too many times without breaking. This may occur, for example, in an algorithm that computes progressively better approximations to a value but exceeds an iteration bound specified by the user.

Public Functions

inline explicit **ExceededMaxIterations**(std::string_view msg)

Struct HasACycle

- Defined in file_xnetwork_exception.hpp

Page Contents

Inheritance Relationships

Base Type

- `public xn::XNetworkException` (*Struct XNetworkException*)

Struct Documentation

struct **xn::HasACycle** : public xn::*XNetworkException*
Raised if (a graph has a cycle when an algorithm expects that it will have no cycles.

Public Functions

inline explicit **HasACycle**(std::string_view msg)

Struct NodeNotFound

- Defined in file_xnetwork_exception.hpp

Page Contents

Inheritance Relationships

Base Type

- `public xn::XNetworkException` (*Struct XNetworkException*)

Struct Documentation

struct **xn::NodeNotFound** : public xn::*XNetworkException*
Exception raised if (requested node is not present : the graph

Public Functions

inline explicit **NodeNotFound**(std::string_view msg)

Struct object

- Defined in file_xnetwork_classes_graph.hpp

Page Contents

Inheritance Relationships

Base Type

- `public py::dict< const char *, std::any >` (*Template Class dict*)

Derived Type

- `public xn::Graph< _nodeview_t, adjlist_t, adjlist_outer_dict_factory >` (*Template Class Graph*)

Struct Documentation

struct **object** : public py::*dict*<const char*, std::any>

Base class for undirected graphs.

A *Graph* stores nodes and edges with optional data, or attributes.

Graphs hold undirected edges. Self loops are allowed but multiple (parallel) edges are not.

Nodes can be arbitrary (hashable) C++ objects with optional key/value attributes. By convention `None` is not used as a node.

Edges are represented as links between nodes with optional key/value attributes.

Parameters

node_container : input graph (optional, default: None) Data to initialize graph. If None (default) an empty graph is created. The data can be any format that is supported by the to_networkx_graph() function, currently including edge list, dict of dicts, dict of lists, NetworkX graph, NumPy matrix or 2d ndarray, SciPy sparse matrix, or PyGraphviz graph.

See Also

DiGraph MultiGraph MultiDiGraph OrderedGraph

Examples

Create an empty graph structure (a “null graph”) with 5 nodes and no edges.

G can be grown in several ways.

Nodes:**

Add one node at a time:

Add the nodes from any container (a list, dict, set or even the lines from a file or the nodes from another graph).

In addition to strings and integers any hashable C++ object (except None) can represent a node, e.g. a customized node object, or even another *Graph*.

Edges:**

G can also be grown by adding edges.

Add one edge,

a list of edges,

or a collection of edges,

If some edges connect nodes not yet in the graph, the nodes are added automatically. There are no errors when adding nodes or edges that already exist.

Attributes:**

Each graph can hold key/value attribute pairs in an associated attribute dictionary (the keys must be hashable). By default these are empty, but can be added or changed using direct manipulation of the attribute dictionaries named graph, node and edge respectively.

{'day': 'Friday'}

Subclasses (Advanced):**

The *Graph* class uses a container-of-container-of-container data structure. The outer dict (node_dict) holds adjacency information keyed by node. The next dict (adjlist_dict) represents the adjacency information and holds edge data keyed by neighbor. The inner dict (edge_attr_dict) represents the edge data and holds edge attribute values keyed by attribute names.

Each of these three dicts can be replaced in a subclass by a user defined dict-like object. In general, the dict-like features should be maintained but extra features can be added. To replace one of the dicts create a new graph class by changing the class(!) variable holding the factory for that dict-like structure. The variable names are node_dict_factory, node_attr_dict_factory, adjlist_inner_dict_factory, adjlist_outer_dict_factory, edge_attr_dict_factory and graph_attr_dict_factory.

node_dict_factory : function, (default: dict) Factory function to be used to create the dict containing node attributes, keyed by node id. It should require no arguments and return a dict-like object

node_attr_dict_factory: function, (default: dict) Factory function to be used to create the node attribute dict which holds attribute values keyed by attribute name. It should require no arguments and return a dict-like object

adjlist_outer_dict_factory : function, (default: dict) Factory function to be used to create the outer-most dict in the data structure that holds adjacency info keyed by node. It should require no arguments and return a dict-like object.

adjlist_inner_dict_factory : function, (default: dict) Factory function to be used to create the adjacency list dict which holds edge data keyed by neighbor. It should require no arguments and return a dict-like object

edge_attr_dict_factory : function, (default: dict) Factory function to be used to create the edge attribute dict which holds attribute values keyed by attribute name. It should require no arguments and return a dict-like object.

graph_attr_dict_factory : function, (default: dict) Factory function to be used to create the graph attribute dict which holds attribute values keyed by attribute name. It should require no arguments and return a dict-like object.

Typically, if your extension doesn't impact the data structure all methods will inherit without issue except: `to_directed/to_undirected`. By default these methods create a DiGraph/Graph class and you probably want them to create your extension of a DiGraph/Graph. To facilitate this we define two class variables that you can set in your subclass.

to_directed_class : callable, (default: DiGraph or MultiDiGraph) Class to create a new graph structure in the `to_directed` method. If `None`, a NetworkX class (DiGraph or MultiDiGraph) is used.

to_undirected_class : callable, (default: *Graph* or MultiGraph) Class to create a new graph structure in the `to_undirected` method. If `None`, a NetworkX class (*Graph* or MultiGraph) is used.

Examples

Create a low memory graph class that effectively disallows edge attributes by using a single attribute dict for all edges. This reduces the memory used, but you lose edge attributes.

... all_edge_dict = {'weight': 1} ... def single_edge_dict(self): ... return self.all_edge_dict ... edge_attr_dict_factory = single_edge_dict {'weight': 1} True

Please see :mod:~networkx.classes.ordered for more examples of creating graph subclasses by overwriting the base class dict with a dictionary-like object.

Subclassed by *xn::Graph< _nodeview_t, adjlist_t, adjlist_outer_dict_factory >*

Struct XNetworkAlgorithmError

- Defined in file_xnetwork_exception.hpp

Page Contents

Inheritance Relationships

Base Type

- `public xn::XNetworkException` (*Struct XNetworkException*)

Derived Types

- `public xn::XNetworkUnbounded` (*Struct XNetworkUnbounded*)
- `public xn::XNetworkUnfeasible` (*Struct XNetworkUnfeasible*)

Struct Documentation

struct **xn::XNetworkAlgorithmError** : public xn::*XNetworkException*

Exception for unexpected termination of algorithms.

Subclassed by *xn::XNetworkUnbounded*, *xn::XNetworkUnfeasible*

Public Functions

inline explicit **XNetworkAlgorithmError**(std::string_view msg)

Struct XNetworkError

- Defined in file_xnetwork_exception.hpp

Page Contents

Inheritance Relationships

Base Type

- `public xn::XNetworkException` (*Struct XNetworkException*)

Struct Documentation

struct **xn::XNetworkError** : public xn::*XNetworkException*
Exception for a serious error : XNetwork

Public Functions

inline explicit **XNetworkError**(std::string_view msg)

Struct XNetworkException

- Defined in file_xnetwork_exception.hpp

Page Contents

Inheritance Relationships

Base Type

- `public runtime_error`

Derived Types

- `public xn::AmbiguousSolution` (*Struct AmbiguousSolution*)
- `public xn::ExceededMaxIterations` (*Struct ExceededMaxIterations*)
- `public xn::HasACycle` (*Struct HasACycle*)
- `public xn::NodeNotFound` (*Struct NodeNotFound*)
- `public xn::XNetworkAlgorithmError` (*Struct XNetworkAlgorithmError*)
- `public xn::XNetworkError` (*Struct XNetworkError*)
- `public xn::XNetworkNotImplemented` (*Struct XNetworkNotImplemented*)
- `public xn::XNetworkPointlessConcept` (*Struct XNetworkPointlessConcept*)

Struct Documentation

struct **xn::XNetworkException** : public runtime_error
Base class for exceptions : XNetwork.

Subclassed by *xn::AmbiguousSolution*, *xn::ExceededMaxIterations*, *xn::HasACycle*, *xn::NodeNotFound*, *xn::XNetworkAlgorithmError*, *xn::XNetworkError*, *xn::XNetworkNotImplemented*, *xn::XNetworkPointlessConcept*

Public Functions

inline explicit **XNetworkException**(std::string_view msg)

Struct XNetworkNoCycle

- Defined in file_xnetwork_exception.hpp

Page Contents

Inheritance Relationships

Base Type

- `public xn::XNetworkUnfeasible` (*Struct XNetworkUnfeasible*)

Struct Documentation

struct **xn::XNetworkNoCycle** : public xn::*XNetworkUnfeasible*
Exception for algorithms that should return a cycle when running on graphs where such a cycle does not exist.

Public Functions

inline explicit **XNetworkNoCycle**(std::string_view msg)

Struct XNetworkNoPath

- Defined in file_xnetwork_exception.hpp

Page Contents

Inheritance Relationships

Base Type

- `public xn::XNetworkUnfeasible` (*Struct XNetworkUnfeasible*)

Struct Documentation

struct **xn::XNetworkNoPath** : public xn::*XNetworkUnfeasible*
Exception for algorithms that should return a path when running on graphs where such a path does not exist.

Public Functions

inline explicit **XNetworkNoPath**(std::string_view msg)

Struct XNetworkNotImplemented

- Defined in file_xnetwork_exception.hpp

Page Contents

Inheritance Relationships

Base Type

- `public xn::XNetworkException` (*Struct XNetworkException*)

Struct Documentation

struct **xn::XNetworkNotImplemented** : public xn::*XNetworkException*
Exception raised by algorithms not implemented for a type of graph.

Public Functions

inline explicit **XNetworkNotImplemented**(std::string_view msg)

Struct XNetworkPointlessConcept

- Defined in file_xnetwork_exception.hpp

Page Contents

Inheritance Relationships

Base Type

- `public xn::XNetworkException` (*Struct XNetworkException*)

Struct Documentation

struct **xn::XNetworkPointlessConcept** : public xn::*XNetworkException*

Raised when a null graph is provided as input to an algorithm that cannot use it.

The null graph is sometimes considered a pointless concept [1]_, thus the name of the exception.

References

.. [1] Harary, F. and Read, R. "Is the Null Graph a Pointless

Concept?" In Graphs and Combinatorics Conference, George Washington University. New York: Springer-Verlag, 1973.

Public Functions

inline explicit **XNetworkPointlessConcept**(std::string_view msg)

Struct XNetworkUnbounded

- Defined in file_xnetwork_exception.hpp

Page Contents

Inheritance Relationships

Base Type

- `public xn::XNetworkAlgorithmError` (*Struct XNetworkAlgorithmError*)

Struct Documentation

struct **xn::XNetworkUnbounded** : public xn::*XNetworkAlgorithmError*

Exception raised by algorithms trying to solve a maximization or a minimization problem instance that is unbounded.

Public Functions

inline explicit **XNetworkUnbounded**(std::string_view msg)

Struct XNetworkUnfeasible

- Defined in file_xnetwork_exception.hpp

Inheritance Relationships

Base Type

- `public xn::XNetworkAlgorithmError` (*Struct XNetworkAlgorithmError*)

Derived Types

- `public xn::XNetworkNoCycle` (*Struct XNetworkNoCycle*)
- `public xn::XNetworkNoPath` (*Struct XNetworkNoPath*)

Struct Documentation

struct **xn::XNetworkUnfeasible** : public xn::*XNetworkAlgorithmError*

Exception raised by algorithms trying to solve a problem instance that has no feasible solution.

Subclassed by *xn::XNetworkNoCycle*, *xn::XNetworkNoPath*

Public Functions

inline explicit **XNetworkUnfeasible**(std::string_view msg)

Template Class AdjacencyView

- Defined in file_xnetwork_classes_coreviews.hpp

Page Contents

Inheritance Relationships

Base Type

- `public AtlasView< Atlas >` (*Template Class AtlasView*)

Template Parameter Order

1. `typename Atlas`

Class Documentation

template<typename **Atlas**>

class **AdjacencyView** : public *AtlasView*<*Atlas*>
An *AdjacencyView* is a Read-only Map of Maps of Maps.

It is a View into a dict-of-dict-of-dict data structure. The inner level of dict is read-write. But the outer levels are read-only.

See Also

AtlasView - View into dict-of-dict MultiAdjacencyView - View into dict-of-dict-of-dict-of-dict

Public Functions

inline explicit **AdjacencyView**(*Atlas* &d)

Template Class AtlasView

- Defined in file_xnetwork_classes_coreviews.hpp

Page Contents

Inheritance Relationships

Derived Type

- `public AdjacencyView< Atlas >` (*Template Class AdjacencyView*)

Template Parameter Order

1. `typename Atlas`

Class Documentation

template<typename **Atlas**>

class **AtlasView**

An *AtlasView* is a Read-only Mapping of Mappings.

It is a View into a dict-of-dict data structure. The inner level of dict is read-write. But the outer level is read-only.

See Also

AdjacencyView - View into dict-of-dict-of-dict MultiAdjacencyView - View into dict-of-dict-of-dict-of-dict

Interface: Mapping

Subclassed by *AdjacencyView< Atlas >*

Public Functions

inline explicit **AtlasView**(*Atlas* &d)

inline auto **size**() const -> size_t

inline auto **begin**() const

inline auto **end**() const

template<typename **T**>
inline auto **operator[]**(const *T* &key) const -> const auto&

template<typename **T**>
inline auto **operator[]**(const *T* &key) -> auto&

Public Members

Atlas &**_atlas**

Template Class bpq_iterator

- Defined in file_ckpttncpp_bpqueue.hpp

Page Contents

Template Parameter Order

1. `typename _Tp`
2. `typename Int`

Class Documentation

template<typename **_Tp**, typename **Int** = int32_t>

class **bpq_iterator**
Bounded Priority Queue Iterator.

Bounded Priority Queue Iterator. Traverse the queue in descending order. Detaching queue items may invalidate the iterator because the iterator makes a copy of current key.

Public Functions

inline constexpr **bpq_iterator**(*bpqueue*<*_Tp*, *Int*> &bpq, UInt curkey)
Construct a new bpq iterator object.

Parameters

- **bpq** – **[in]**
- **curkey** – **[in]**

inline constexpr auto **operator++**() -> *bpq_iterator*&
move to the next item

Returns *bpq_iterator*&

inline constexpr auto **operator***() -> Item&
get the reference of the current item

Returns *bpq_iterator*&

Friends

inline friend friend constexpr auto operator== (const bpq_iterator &lhs, const bpq_iterator &rhs) -> bool
eq operator

Parameters **rhs** – **[in]**

Returns true

Returns false

inline friend friend constexpr auto operator!= (const bpq_iterator &lhs, const bpq_iterator &rhs) -> bool
neq operator

Parameters **rhs** – **[in]**

Returns true

Returns false

Template Class bpqueue

- Defined in file_ckpttncpp_bpqueue.hpp

Page Contents

Template Parameter Order

1. `typename _Tp`
2. `typename Int`
3. `typename _Sequence`

Class Documentation

template<typename **_Tp**, typename **Int** = int32_t, typename **_Sequence** = std::vector<*dllink*<std::pair<*_Tp*, std::make_unsigned_t<*Int*>>>>>
class **bpqueue**

bounded priority queue

Bounded Priority Queue with integer keys in [a..b]. Implemented by array (bucket) of doubly-linked lists. Efficient if key is bounded by a small integer value.

Note that this class does not own the PQ nodes. This feature makes the nodes sharable between doubly linked list class and this class. In the FM algorithm, the node either attached to the gain buckets (PQ) or in the waitinglist (doubly linked list), but not in both of them in the same time.

Another improvement is to make the array size one element bigger i.e. (b - a + 2). The extra dummy array element (which is called sentinel) is used to reduce the boundary checking during updating.

All the member functions assume that the keys are within the bound.

: support std::pmr

Public Types

using **value_type** = typename *_Sequence*::value_type

using **reference** = typename *_Sequence*::reference

using **const_reference** = typename *_Sequence*::const_reference

using **size_type** = typename *_Sequence*::size_type

using **container_type** = *_Sequence*

Public Functions

inline constexpr **bpqueue**(*Int* a, *Int* b)

Construct a new bpqueue object.

Parameters

- **a** – **[in]** lower bound
- **b** – **[in]** upper bound

bpqueue(const *bpqueue*&) = delete

~bpqueue() = default

constexpr auto **operator=**(const *bpqueue*&) -> *bpqueue*& = delete

constexpr **bpqueue**(*bpqueue*&&) noexcept = default

constexpr auto **operator=**(*bpqueue*&&) noexcept -> *bpqueue*& = default

inline constexpr auto **is_empty**() const noexcept -> bool
whether the bpqueue is empty.

Returns true

Returns false

inline constexpr auto **set_key**(Item &it, *Int* gain) noexcept -> void
Set the key object.

Parameters

- **it** – **[out]** the item
- **gain** – **[in]** the key of it

inline constexpr auto **get_max**() const noexcept -> *Int*
Get the max value.

Returns T maximum value

inline constexpr auto **clear**() noexcept -> void
clear reset the PQ

inline constexpr auto **append_direct**(Item &it) noexcept -> void
append item with internal key

Parameters **it** – **[inout]** the item

inline constexpr auto **append**(Item &it, *Int* k) noexcept -> void
append item with external key

Parameters

- **it** – **[inout]** the item
- **k** – **[in]** the key

inline constexpr auto **popleft**() noexcept -> Item&
append from list

pop node with the highest key

Parameters **nodes** – **[inout]**

Returns dllink&

inline constexpr auto **decrease_key**(Item &it, UInt delta) noexcept -> void
decrease key by delta

Note that the order of items with same key will not be preserved. For FM algorithm, this is a prefered behavior.

Parameters

- **it** – **[inout]** the item
- **delta** – **[in]** the change of the key

inline constexpr auto **increase_key**(Item &it, UInt delta) noexcept -> void
increase key by delta

Note that the order of items with same key will not be preserved. For FM algorithm, this is a prefered behavior.

Parameters

- **it** – **[inout]** the item
- **delta** – **[in]** the change of the key

inline constexpr auto **modify_key**(Item &it, *Int* delta) noexcept -> void
modify key by delta

Note that the order of items with same key will not be preserved. For FM algorithm, this is a prefered behavior.

Parameters

- **it** – **[inout]** the item
- **delta** – **[in]** the change of the key

inline constexpr auto **detach**(Item &it) noexcept -> void
detach the item from bpqueue

Parameters **it** – **[inout]** the item

inline constexpr auto **begin**() -> *bpq_iterator*<*_Tp*, *Int*>
iterator point to begin

Returns *bpq_iterator*

inline constexpr auto **end**() -> *bpq_iterator*<*_Tp*, *Int*>
iterator point to end

Returns *bpq_iterator*

Template Class dll_iterator

- Defined in file_ckpttncpp_dllist.hpp

Page Contents

Template Parameter Order

1. `typename T`

Class Documentation

template<typename **T**>

class **dll_iterator**
list iterator

List iterator. Traverse the list from the first item. Usually it is safe to attach/detach list items during the iterator is active.

Public Functions

inline explicit constexpr **dll_iterator**(*dllink*<*T*> *cur) noexcept
Construct a new dll iterator object.

Parameters **cur** – **[in]**

inline constexpr auto **operator++**() noexcept -> *dll_iterator*&
move to the next item

Returns dllink&

inline constexpr auto **operator***() noexcept -> *dllink*<*T*>&
get the reference of the current item

Returns dllink&

Friends

inline friend friend auto operator== (const dll_iterator &lhs, const dll_iterator &rhs) noexcept -> bool
eq operator

Parameters **rhs** – **[in]**

Returns true

Returns false

inline friend friend auto operator!= (const dll_iterator &lhs, const dll_iterator &rhs) noexcept -> bool
neq operator

Parameters **rhs** – **[in]**

Returns true

Returns false

Template Class dllink

- Defined in file_ckpttncpp_dllist.hpp

Page Contents

Template Parameter Order

1. `typename T`

Class Documentation

template<typename **T**>

class **dllink**
doubly linked node (that may also be a "head" a list)

A Doubly-linked List class. This class simply contains a link of node's. By adding a "head" node (sentinel), deleting a node is extremely fast (see "Introduction to Algorithm"). This class does not keep the length information as it is not necessary for the FM algorithm. This saves memory and run-time to update the length information. Note that this class does not own the list node. They are supplied by the caller in order to better reuse the nodes.

Public Functions

inline explicit constexpr **dllink**(*T* data) noexcept
Construct a new dllink object.

Parameters **data** – **[in]** the data

constexpr **dllink**() = default
Copy construct a new dllink object (deleted intentionally)

~dllink() = default

dllink(const *dllink*&) = delete

constexpr auto **operator=**(const *dllink*&) -> *dllink*& = delete

constexpr **dllink**(*dllink*&&) noexcept = default

constexpr auto **operator=**(*dllink*&&) noexcept -> *dllink*& = default

inline constexpr auto **lock**() noexcept -> void
lock the node (and don't append it to any list)

inline constexpr auto **is_locked**() const noexcept -> bool
whether the node is locked

Returns true

Returns false

inline constexpr auto **is_empty**() const noexcept -> bool
whether the list is empty

Returns true

Returns false

inline constexpr auto **clear**() noexcept -> void
reset the list

inline constexpr auto **detach**() noexcept -> void
detach from a list

inline constexpr auto **appendleft**(*dllink* &node) noexcept -> void
append the node to the front

Parameters **node** – **[inout]**

inline constexpr auto **append**(*dllink* &node) noexcept -> void
append the node to the back

Parameters **node** – **[inout]**

inline constexpr auto **popleft**() noexcept -> *dllink*&
pop a node from the front

Precondition: list is not empty

Returns dllink&

inline constexpr auto **pop**() noexcept -> *dllink*&
pop a node from the back

Precondition: list is not empty

Returns dllink&

inline constexpr auto **begin**() noexcept -> *dll_iterator*<*T*>
begin

Returns *dll_iterator*

inline constexpr auto **end**() noexcept -> *dll_iterator*<*T*>
end

Returns *dll_iterator*

Public Members

T **data** = {}
data

Class FMBiConstrMgr

- Defined in file_ckpttncpp_FMBiConstrMgr.hpp

Page Contents

Inheritance Relationships

Base Type

- `public FMConstrMgr` (*Class FMConstrMgr*)

Class Documentation

class **FMBiConstrMgr** : public *FMConstrMgr*
Constraint Manager.

Check if (the move of v can satisfied, makebetter, or notsatisfied

Public Functions

inline **FMBiConstrMgr**(const *SimpleNetlist* &H, double BalTol)
Construct a new *FMBiConstrMgr* object.

Parameters

- **H** – **[in]**
- **BalTol** – **[in]**

inline **FMBiConstrMgr**(const *SimpleNetlist* &H, double BalTol, std::uint8_t)
Construct a new *FMBiConstrMgr* object (for general framework)

Parameters

- **H** – **[in]**
- **BalTol** – **[in]**
- **K** – **[in]** (for compatability only)

inline auto **select_togo**() const -> std::uint8_t

Returns std::uint8_t

Class FMBiGainCalc

- Defined in file_ckpttncpp_FMBiGainCalc.hpp

Page Contents

Class Documentation

class **FMBiGainCalc**
FMBiGainCalc.

Public Types

using **node_t** = typename *SimpleNetlist*::node_t

using **Item** = *dllink*<std::pair<*node_t*, uint32_t>>

Public Functions

inline explicit **FMBiGainCalc**(const *SimpleNetlist* &H, std::uint8_t)
Construct a new *FMBiGainCalc* object.

Parameters

- **H** – **[in]**
- **K** – **[in]**

inline auto **init**(gsl::span<const std::uint8_t> part) -> int

Parameters **part** – **[in]**

inline auto **update_move_init**() -> void
update move init

void **init_IdVec**(const *node_t* &v, const *node_t* &net)

auto **update_move_2pin_net**(gsl::span<const std::uint8_t> part, const *MoveInfo*<*node_t*> &move_info) -> *node_t*
update move 2-pin net

Parameters

- **part** – **[in]**
- **move_info** – **[in]**
- **w** – **[out]**

Returns int

auto **update_move_3pin_net**(gsl::span<const std::uint8_t> part, const *MoveInfo*<*node_t*> &move_info) -> std::vector<int>

update move 3-pin net

Parameters

- **part** – **[in]**
- **move_info** – **[in]**

Returns ret_info

auto **update_move_general_net**(gsl::span<const std::uint8_t> part, const *MoveInfo*<*node_t*> &move_info) -> std::vector<int>

update move general net

Parameters

- **part** – **[in]**
- **move_info** – **[in]**

Returns ret_info

Public Members

int **deltaGainW** = {}

FMPmr::vector<*node_t*> **IdVec**

bool **special_handle_2pin_nets** = {true}

Class FMBiGainMgr

- Defined in file_ckpttncpp_FMBiGainMgr.hpp

Page Contents

Inheritance Relationships

Base Type

- `public FMGainMgr< FMBiGainCalc, FMBiGainMgr >` (*Template Class FMGainMgr*)

Class Documentation

class **FMBiGainMgr** : public *FMGainMgr*<*FMBiGainCalc*, *FMBiGainMgr*>
FMBiGainMgr.

Public Types

using **Base** = *FMGainMgr*<*FMBiGainCalc*, *FMBiGainMgr*>

using **GainCalc_** = *FMBiGainCalc*

using **node_t** = typename *SimpleNetlist*::node_t

Public Functions

inline explicit **FMBiGainMgr**(const *SimpleNetlist* &H)

inline **FMBiGainMgr**(const *SimpleNetlist* &H, std::uint8_t)
Construct a new *FMBiGainMgr* object.

Parameters **H** – **[in]**

auto **init**(gsl::span<const std::uint8_t> part) -> int

Parameters **part** – **[in]**

Returns int

inline auto **modify_key**(const *node_t* &w, std::uint8_t part_w, int key) -> void
(needed by base class)

Parameters

- **w** – **[in]**
- **part_w** – **[in]**
- **key** – **[in]**

inline auto **update_move_v**(const *MoveInfoV*<*node_t*> &move_info_v, int gain) -> void

Parameters

- **move_info_v** – **[in]**
- **gain** – **[in]**

inline auto **lock**(uint8_t whichPart, const *node_t* &v) -> void
lock

Parameters

- **whichPart** – **[in]**
- **v** – **[in]**

inline auto **lock_all**(uint8_t fromPart, const *node_t* &v) -> void
lock_all

Parameters

- **fromPart** – [in]
- **v** – [in]

Class FMConstrMgr

- Defined in file_ckpttncpp_FMConstrMgr.hpp

Page Contents

Inheritance Relationships

Derived Types

- `public FMBiConstrMgr` (*Class FMBiConstrMgr*)
- `public FMKWayConstrMgr` (*Class FMKWayConstrMgr*)

Class Documentation

class **FMConstrMgr**

FM Partition Constraint Manager.

Subclassed by *FMBiConstrMgr*, *FMKWayConstrMgr*

Public Types

using **node_t** = typename *SimpleNetlist*::node_t

Public Functions

auto **init**(gsl::span<const std::uint8_t> part) -> void

Parameters **part** – [in]

auto **check_legal**(const *MoveInfoV*<*node_t*> &move_info_v) -> *LegalCheck*

Parameters **move_info_v** – [in]

Returns LegalCheck

auto **check_constraints**(const *MoveInfoV*<*node_t*> &move_info_v) -> bool

Parameters **move_info_v** – **[in]**

Returns true

Returns false

auto **update_move**(const *MoveInfoV*<*node_t*> &move_info_v) -> void

Parameters **move_info_v** – **[in]**

Protected Functions

inline **FMConstrMgr**(const *SimpleNetlist* &H, double BalTol)
Construct a new *FMConstrMgr* object.

Parameters

- **H** – **[in]**
- **BalTol** – **[in]**

FMConstrMgr(const *SimpleNetlist* &H, double BalTol, std::uint8_t K)
Construct a new *FMConstrMgr* object.

Parameters

- **H** – **[in]**
- **BalTol** – **[in]**
- **K** – **[in]**

Protected Attributes

std::vector<unsigned int> **diff**

unsigned int **lowerbound** = {}

std::uint8_t **K**

Template Class FMGainMgr

- Defined in file_ckpttncpp_FMGainMgr.hpp

Page Contents

Template Parameter Order

1. `typename GainCalc`
2. `class Derived`

Class Documentation

template<typename **GainCalc**, class **Derived**>

class **FMGainMgr**

tparam GainCalc

tparam Derived

Public Functions

FMGainMgr(const *SimpleNetlist* &H, std::uint8_t K)
Construct a new *FMGainMgr* object.

Parameters

- **H** – **[in]**
- **K** – **[in]**

auto **init**(gsl::span<const std::uint8_t> part) -> int

Parameters **part** – **[in]**

inline auto **is_empty_togo**(uint8_t toPart) const -> bool

Parameters **toPart** – **[in]**

Returns true

Returns false

inline auto **is_empty**() const -> bool

Returns true

Returns false

auto **select**(gsl::span<const std::uint8_t> part) -> std::tuple<*MoveInfoV*<node_t>, int>

Parameters **part** – **[in]**

Returns std::tuple<MoveInfoV<node_t>, int>

auto **select_togo**(uint8_t toPart) -> std::tuple<node_t, int>

Parameters **toPart** – **[in]**

Returns std::tuple<node_t, int>

auto **update_move**(gsl::span<const std::uint8_t> part, const *MoveInfoV*<node_t> &move_info_v) -> void

Parameters

- **part** – **[in]**
- **move_info_v** – **[in]**

Public Members

GainCalc **gainCalc**

Protected Attributes

Item **waitinglist** = {std::pair{node_t{}, uint32_t(0)}}

const *SimpleNetlist* &**H**

std::vector<*bpqueue*<node_t>> **gainbucket**

std::uint8_t **K**

Class FMKWayConstrMgr

- Defined in file_ckpttncpp_FMKWayConstrMgr.hpp

Page Contents

Inheritance Relationships

Base Type

- `public FMConstrMgr` (*Class FMConstrMgr*)

Class Documentation

class **FMKWayConstrMgr** : public *FMConstrMgr*

FM K-Way Partition Constraint Manager.

Public Functions

inline **FMKWayConstrMgr**(const *SimpleNetlist* &H, double BalTol, std::uint8_t K)
Construct a new *FMKWayConstrMgr* object.

Parameters

- **H** – **[in]**
- **BalTol** – **[in]**
- **K** – **[in]**

inline auto **select_togo**() const -> std::uint8_t

Returns std::uint8_t

inline auto **init**(gsl::span<const std::uint8_t> part) -> void

Parameters **part** – **[in]**

auto **check_legal**(const *MoveInfoV*<node_t> &move_info_v) -> *LegalCheck*

Parameters **move_info_v** – **[in]**

Returns LegalCheck

Class FMKWayGainCalc

- Defined in file_ckpttncpp_FMKWayGainCalc.hpp

Page Contents

Class Documentation

class **FMKWayGainCalc**
FMKWayGainCalc.

Public Types

using **ret_info** = std::vector<std::vector<int>>

Public Functions

inline **FMKWayGainCalc**(const *SimpleNetlist* &H, std::uint8_t K)
Construct a new *FMKWayGainCalc* object.

Parameters

- **H** – **[in]** *Netlist*
- **K** – **[in]** number of partitions

inline auto **init**(gsl::span<const std::uint8_t> part) -> int

Parameters

- **toPart** – **[in]**
- **part** – **[in]**

Returns dllink*

inline auto **update_move_init**() -> void

auto **update_move_2pin_net**(gsl::span<const std::uint8_t> part, const *MoveInfo*<node_t> &move_info) -> node_t

Parameters

- **part** – **[in]**
- **move_info** – **[in]**
- **w** – **[out]**

Returns std::vector<int>

void **init_IdVec**(const node_t &v, const node_t &net)

auto **update_move_3pin_net**(gsl::span<const std::uint8_t> part, const *MoveInfo*<node_t> &move_info) -> *ret_info*

Parameters

- **part** – **[in]**
- **move_info** – **[in]**

Returns ret_info

auto **update_move_general_net**(gsl::span<const std::uint8_t> part, const *MoveInfo*<node_t> &move_info) -> *ret_info*

Parameters

- **part** – **[in]**
- **move_info** – **[in]**

Returns ret_info

Public Members

FMPmr::vector<int> **deltaGainW**

FMPmr::vector<node_t> **IdVec**

bool **special_handle_2pin_nets** = {true}

Class FMKWayGainMgr

- Defined in file_ckpttncpp_FMKWayGainMgr.hpp

Page Contents

Inheritance Relationships

Base Type

- `public FMGainMgr< FMKWayGainCalc, FMKWayGainMgr >` (*Template Class FMGainMgr*)

Class Documentation

class **FMKWayGainMgr** : public *FMGainMgr*<*FMKWayGainCalc*, *FMKWayGainMgr*>
FMKWayGainMgr.

Public Types

using **Base** = *FMGainMgr*<*FMKWayGainCalc*, *FMKWayGainMgr*>

using **GainCalc_** = *FMKWayGainCalc*

using **node_t** = typename *SimpleNetlist*::node_t

Public Functions

inline **FMKWayGainMgr**(const *SimpleNetlist* &H, std::uint8_t K)
Construct a new *FMKWayGainMgr* object.

Parameters

- **H** – **[in]**
- **K** – **[in]**

auto **init**(gsl::span<const std::uint8_t> part) -> int

Parameters **part** – **[in]**

inline auto **modify_key**(const *node_t* &w, std::uint8_t part_w, gsl::span<const int> keys) -> void
(needed by base class)

Parameters

- **w** – **[in]**
- **part_w** – **[in]**
- **keys** – **[in]**

auto **update_move_v**(const *MoveInfoV*<*node_t*> &move_info_v, int gain) -> void

Parameters

- **move_info_v** – **[in]**
- **gain** – **[in]**

inline auto **lock**(uint8_t whichPart, const *node_t* &v) -> void
lock

Parameters

- **whichPart** – **[in]**
- **v** – **[in]**

inline auto **lock_all**(uint8_t, const *node_t* &v) -> void
lock_all

Parameters

- **fromPart** – **[in]**
- **v** – **[in]**

Template Class FMPartMgr

- Defined in file_ckpttncpp_FMPartMgr.hpp

Page Contents

Inheritance Relationships

Base Type

- `public PartMgrBase< GainMgr, ConstrMgr, FMPartMgr >` (*Template Class PartMgrBase*)

Template Parameter Order

1. `typename GainMgr`
2. `typename ConstrMgr`

Class Documentation

template<typename **GainMgr**, typename **ConstrMgr**>

class **FMPartMgr** : public *PartMgrBase*<*GainMgr*, *ConstrMgr*, *FMPartMgr*>
FM Partition Manager.

tparam GainMgr

tparam ConstrMgr

Public Functions

inline **FMPartMgr**(const *SimpleNetlist* &H, *GainMgr* &gainMgr, *ConstrMgr* &constrMgr, size_t K)
Construct a new *FMPartMgr* object.

Parameters

- **H** – **[in]**
- **gainMgr** – **[inout]**
- **constrMgr** – **[inout]**

inline **FMPartMgr**(const *SimpleNetlist* &H, *GainMgr* &gainMgr, *ConstrMgr* &constrMgr)
Construct a new *FMPartMgr* object.

Parameters

- **H** – **[in]**
- **gainMgr** – **[inout]**
- **constrMgr** – **[inout]**

inline auto **take_snapshot**(gsl::span<const std::uint8_t> part) -> std::vector<std::uint8_t>

Parameters `part` – **[in]**

Returns *Snapshot*

inline auto **restore_part**(const std::vector<std::uint8_t> &snapshot, gsl::span<std::uint8_t> part) -> void

Parameters

- **snapshot** – **[in]**
- **part** – **[inout]**

Template Class HierNetlist

- Defined in file_ckpttncpp_HierNetlist.hpp

Page Contents

Inheritance Relationships

Base Type

- `public Netlist< graph_t >` (*Template Struct Netlist*)

Template Parameter Order

1. **`graph_t <exhale_class_classxn_1_1Graph_>`_**

Class Documentation

template<typename **graph_t**>

class **HierNetlist** : public *Netlist*<*graph_t*>
HierNetlist.

HierNetlist is implemented by *xn::Graph*, which is a networkx-like graph.

Public Types

using **nodeview_t** = typename *graph_t*::nodeview_t

using **node_t** = typename *graph_t*::node_t

using **index_t** = typename *nodeview_t*::key_type

Public Functions

HierNetlist(*graph_t* G, const *nodeview_t* &modules, const *nodeview_t* &nets)
Construct a new *HierNetlist* object.

Construct a new *Netlist* object.

Parameters

- **G** – **[in]**
- **module_list** – **[in]**
- **net_list** – **[in]**
- **module_fixed** – **[in]**
- **G** – **[in]**
- **modules** – **[in]**
- **nets** – **[in]**

Template Parameters

- **nodeview_t** –
- **nodemap_t** –

void **projection_down**(gsl::span<const std::uint8_t> part, gsl::span<std::uint8_t> part_down) const
Construct a new *Netlist* object.

projection down

Parameters

- **G** – **[in]**
- **num_modules** – **[in]**
- **num_nets** – **[in]**
- **part** – **[in]**
- **part_down** – **[out]**

void **projection_up**(gsl::span<const std::uint8_t> part, gsl::span<std::uint8_t> part_up) const
projection up

Parameters

- **part** – **[in]**
- **part_up** – **[out]**

inline auto **get_net_weight**(const *node_t* &net) const -> int
Get the net weight.

Returns int

Public Members

const *Netlist*<*graph_t*> ***parent**

std::vector<*node_t*> **node_up_map**

std::vector<*node_t*> **node_down_map**

py::*dict*<*index_t*, *node_t*> **cluster_down_map**

shift_array<std::vector<int>> **net_weight** = {}

Class MLPartMgr

- Defined in file_ckpttncpp_MLPartMgr.hpp

Page Contents

Class Documentation

class **MLPartMgr**
Multilevel Partition Manager.

Public Functions

inline explicit **MLPartMgr**(double BalTol)
Construct a new *MLPartMgr* object.

Parameters **BalTol** – **[in]**

inline **MLPartMgr**(double BalTol, std::uint8_t K)
Construct a new *MLPartMgr* object.

Parameters

- **BalTol** – **[in]**
- **K** – **[in]**

inline void **set_limitsize**(size_t limit)

template<typename **PartMgr**>
auto **run_FMPartition**(const *SimpleNetlist* &H, gsl::span<std::uint8_t> part) -> *LegalCheck*
run_Partition

Template Parameters

- **GainMgr** –
- **ConstrMgr** –

Parameters

- **H** – **[in]**

- **part** – **[inout]**

Returns LegalCheck

Public Members

int **totalcost** = { }

Template Class PartMgrBase

- Defined in file_ckpttncpp_PartMgrBase.hpp

Page Contents

Template Parameter Order

1. `typename GainMgr`
2. `typename ConstrMgr`
3. `template< typename _gainMgr, typename _constrMgr > class Derived`

Class Documentation

template<typename **GainMgr**, typename **ConstrMgr**, template<typename _gainMgr, typename _constrMgr> class **Derived**>
class **PartMgrBase**

Partition Manager Base.

In order to do that, heuristics refer to a measure of the gain (and balance condition) associated to any sequence of changes performed on the current solution. Moreover, the length of the sequence generated is determined by evaluting a suitably defined $stopping rule$ at each iteration.

tparam GainMgr

tparam ConstrMgr

tparam Derived Iterative Improvement Partitioning Base Class. In this partitioning method, the next solution s' considered after solution s is dervied by first applying a sequence of t changes (moves) to s (with t dependent from s and from the specific heuristic method), thus obtaining a sequence of solution $s,...,s_t$ and by successively choosing the best among these solutions.

Reference: G. Ausiello et al., Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties, Section 10.3.2.

Public Types

using **GainCalc_** = typename *GainMgr*::GainCalc_

using **GainMgr_** = *GainMgr*

using **ConstrMgr_** = *ConstrMgr*

using **Der** = *Derived*<*GainMgr*, *ConstrMgr*>

Public Functions

inline **PartMgrBase**(const *SimpleNetlist* &H, *GainMgr* &gainMgr, *ConstrMgr* &constrMgr, size_t K)
Construct a new FDPartMgr object.

Parameters

- **H** – **[in]**
- **gainMgr** – **[inout]**
- **constrMgr** – **[inout]**

void **init**(gsl::span<std::uint8_t> part)

Parameters **part** – **[inout]**

auto **legalize**(gsl::span<std::uint8_t> part) -> *LegalCheck*

Parameters **part** – **[inout]**

Returns LegalCheck

void **optimize**(gsl::span<std::uint8_t> part)

Parameters **part** – **[inout]**

Public Members

int **totalcost** = { }

Protected Attributes

Der &**self** = *static_cast<*Der**>(this)

const *SimpleNetlist* &**H**

GainMgr &**gainMgr**

ConstrMgr &**validator**

size_t **K**

Template Class dict

- Defined in file_py2cpp_py2cpp.hpp

Page Contents

Inheritance Relationships

Base Type

- `public std::unordered_map< Key, T >`

Template Parameter Order

1. `typename Key`
2. `typename T`

Class Documentation

template<typename **Key**, typename **T**>

class `py::dict` : public std::unordered_map<*Key*, *T*>

tparam Key

tparam T

Public Types

using **value_type** = std::pair<const *Key*, *T*>

Public Functions

inline **dict**()
Construct a new dict object.

auto **operator=**(Self&&) noexcept -> *dict*& = default

Returns _Self&

dict(*dict*<*Key*, *T*>&&) noexcept = default
Move Constructor (default)

~dict() = default

dict(const *dict*<*Key*, *T*>&) = default
Construct a new dict object.

Copy through explicitly the public copy() function!!!

Template Class set

- Defined in file_py2cpp_py2cpp.hpp

Page Contents

Inheritance Relationships

Base Type

- public std::unordered_set< Key >

Template Parameter Order

1. typename Key

Class Documentation

template<typename **Key**>

class **py::set** : public std::unordered_set<*Key*>

tparam Key

Public Functions

inline **set**()
Construct a new set object.

auto **operator=**(*set*&&) noexcept -> *set*& = default

Returns _Self&

set(*set*<*Key*>&&) noexcept = default
Move Constructor (default)

set(const *set*<*Key*>&) = default
Copy Constructor (deleted)

Copy through explicitly the public copy() function!!!

Template Class robin

- Defined in file_ckpttncpp_robin.hpp

Page Contents

Nested Relationships

Nested Types

- *Struct robin::iterable_wrapper*
- *Struct robin::iterator*
- *Struct robin::slnode*

Template Parameter Order

1. `typename T`

Class Documentation

template<typename **T**>

class **robin**

Public Functions

inline explicit **robin**(*T* K)

inline auto **exclude**(*T* fromPart)

Class set_partition_

- Defined in file_ckpttncpp_set_partition.hpp

Page Contents

Class Documentation

class **set_partition_**

Public Functions

inline explicit **set_partition_**(Fun yield_)
Construct object.

Parameters **yield_** – **[in]**

inline void **run**(int n, int k)

Parameters

- **n** – **[in]**
- **k** – **[in]**

Template Class shift_array

- Defined in file_ckpttncpp_array_like.hpp

Page Contents

Inheritance Relationships

Base Type

- `public C`

Template Parameter Order

1. `typename C`

Class Documentation

template<typename **C**>

class **shift_array** : public *C*

Public Functions

inline **shift_array**()

inline **shift_array**(*C* &&base)

inline void **set_start**(const size_t &start)

inline auto **operator[]**(const size_t &index) const -> const value_type&

inline auto **operator[]**(const size_t &index) -> value_type&

Template Class AtlasView

- Defined in file_py2cpp_nx2bgl.hpp

Page Contents

Template Parameter Order

1. `typename Vertex`
2. `typename Graph`

Class Documentation

template<typename **`Vertex`**, typename **`Graph`**>

class `xn::AtlasView`

tparam Vertex

tparam Graph

Public Functions

inline **`AtlasView`**(*Vertex* v, const *Graph* &G)
Construct a new Atlas View object.

Parameters

- **`v`** – **[in]**
- **`G`** – **[in]**

inline auto **`begin`**() const

Returns auto

inline auto **`end`**() const

Returns auto

inline auto **`cbegin`**() const

Returns auto

inline auto **`cend`**() const

Returns auto

Template Class EdgeView

- Defined in file_py2cpp_nx2bgl.hpp

Page Contents

Template Parameter Order

1. `typename Graph`

Class Documentation

template<typename **Graph**>

class `xn::EdgeView`

tparam Graph

Public Functions

inline explicit **EdgeView**(const *Graph* &G)
Construct a new Edge View object.

Parameters `G` – **[in]**

inline auto **begin**() const

Returns auto

inline auto **end**() const

Returns auto

inline auto **cbegin**() const

Returns auto

inline auto **cend**() const

Returns auto

Template Class grAdaptor

- Defined in file_py2cpp_nx2bgl.hpp

Page Contents

Inheritance Relationships

Base Type

- `public xn::VertexView< Graph >` (*Template Class VertexView*)

Template Parameter Order

1. **`Graph <exhale_class_classxn_1_1Graph_>`_**

Class Documentation

template<typename **Graph**>

class `xn::grAdaptor` : public xn::*VertexView*<*Graph*>

tparam Graph

Public Types

using **Vertex** = typename boost::graph_traits<*Graph*>::vertex_descriptor

using **node_t** = *Vertex*

using **edge_t** = typename boost::graph_traits<*Graph*>::edge_descriptor

Public Functions

grAdaptor() = delete
Construct a new gr Adaptor object.

inline explicit **grAdaptor**(*Graph* &&G) noexcept
Construct a new gr Adaptor object.

Parameters G – **[in]**

Template Class Graph

- Defined in file_xnetwork_classes_graph.hpp

Page Contents

Inheritance Relationships

Base Type

- `public xn::object` (*Struct object*)

Derived Type

- `public xn::VertexView< Graph >` (*Template Class VertexView*)

Template Parameter Order

1. `typename _nodeview_t`
2. `typename adjlist_t`
3. `typename adjlist_outer_dict_factory`

Class Documentation

template<typename **_nodeview_t**, typename **adjlist_t** = py::*set*<*Value_type*<*_nodeview_t*>>, typename **adjlist_outer_dict_factory** = py::*dict*<*Value_type*<*_nodeview_t*>, *adjlist_t*>>
class **xn::Graph** : public xn::*object*

Subclassed by *xn::VertexView< Graph >*

Public Types

using **nodeview_t** = *_nodeview_t*

using **Node** = typename *nodeview_t*::value_type

using **dict** = py::*dict*<const char*, std::any>

using **graph_attr_dict_factory** = *dict*

using **adjlist_inner_dict_factory** = *adjlist_t*

using **key_type** = typename *adjlist_t*::key_type

using **value_type** = typename *adjlist_t*::value_type

using **edge_t** = std::pair<*Node*, *Node*>

using **node_t** = *Node*

Public Functions

inline explicit **Graph**(const *nodeview_t* &Nodes)
Initialize a graph with edges, name, or graph attributes.

Parameters

node_container : input nodes

Examples

inline explicit **Graph**(uint32_t num_nodes)

inline auto **adj**() const
Graph adjacency object holding the neighbors of each node.

This object is a read-only dict-like structure with node keys and neighbor-dict values. The neighbor-dict is keyed by neighbor to the edge-data-dict. So `G.adj[3][2]['color'] = 'blue'`sets the color of the edge(3, 2)to"blue"`.

Iterating over G.adj behaves like a dict. Useful idioms include `for nbr, datadict in G.adj[n].items():`.

The neighbor information is also provided by subscripting the graph. So `for nbr, foovalue in G[node].data('foo', default=1):` works.

For directed graphs, `G.adj` holds outgoing (successor) info.

inline auto **adj**()

inline auto **_nodes_nbrs**() const

inline auto **get_name**()

inline auto **set_name**(std::string_view s)

inline auto **begin**() const
Iterate over the nodes. Use: "for (const auto& n : G)".

Returns

niter : iterator An iterator over all nodes : the graph.

Examples

[0, 1, 2, 3]; [0, 1, 2, 3];

inline auto **end**() const

inline auto **contains**(const *Node* &n) -> bool
Return true if (n is a node, false otherwise. Use: "n : G".

Examples

true

inline auto **operator[]**(const *Node* &n) const -> const auto&
Return a dict of neighbors of node n. Use: "G[n]".

Parameters

n : node A node in the graph.

Returns

adj_dict : dictionary The adjacency dictionary for nodes connected to n.

Notes

G[n] is the same as G.adj[n] and similar to G.neighbors(n); (which is an iterator over G.adj[n]);

Examples

AtlasView({1: {}});

inline auto **operator[]**(const *Node* &n) -> auto&

inline auto **nodes**()

inline auto **number_of_nodes**() const
Return the number of nodes : the graph.

Returns

nnodes : int The number of nodes : the graph.

See Also

order, **len** which are identical

Examples

3

inline auto **number_of_edges**() const

inline auto **order**()
Return the number of nodes : the graph.

Returns

nnodes : int The number of nodes : the graph.

See Also

number_of_nodes, **len** which are identical

inline auto **has_node**(const *Node* &n)
Return true if (the graph contains the node n.

Identical to `n : G`

Parameters

n : node

Examples

true

template<typename **U** = *key_type*>
inline auto **add_edge**(const *Node* &u, const *Node* &v) -> typename std::enable_if<std::is_same<*U*, *value_type*>::value>::type
Add an edge between u and v.

The nodes u and v will be automatically added if (they are not already : the graph.

Edge attributes can be specified with keywords or by directly accessing the edge"s attribute dictionary. See examples below.

Parameters

u, v : nodes Nodes can be, for example, strings or numbers. Nodes must be hashable (and not None) C++ objects.

See Also

add_edges_from : add a collection of edges

Notes

Adding an edge that already exists updates the edge data.

Many XNetwork algorithms designed for weighted graphs use an edge attribute (by default `weight`) to hold a numerical value.

Examples

The following all add the edge e=(1, 2) to graph G) {

Associate data to edges using keywords) {

For non-string attribute keys, use subscript notation.

template<typename **U** = *key_type*>
inline auto **add_edge**(const *Node* &u, const *Node* &v) -> typename std::enable_if<!std::is_same<*U*, *value_type*>::value>::type

template<typename **T**>
inline auto **add_edge**(const *Node* &u, const *Node* &v, const *T* &data)

template<typename **C1**, typename **C2**>
inline auto **add_edges_from**(const *C1* &edges, const *C2* &data)

inline auto **has_edge**(const *Node* &u, const *Node* &v) -> bool

inline auto **degree**(const *Node* &n) const

inline auto **clear**()
An *EdgeView* of the *Graph* as G.edges().

edges(nbunch=None, data=false, default=None);

The *EdgeView* provides set-like operations on the edge-tuples as well as edge attribute lookup. When called, it also provides an EdgeDataView object which allows control of access to edge attributes (but does not provide set-like operations). Hence, `G.edges[u, v]["color"]` provides the value of the color attribute for edge `(u, v)` while `for (auto [u, v, c] : G.edges.data("color", default="red") {` iterates through all the edges yielding the color attribute with default `"red"` if (no color attribute exists.

Parameters

nbunch : single node, container, or all nodes (default= all nodes); The view will only report edges incident to these nodes. data : string or bool, optional (default=false); The edge attribute returned : 3-tuple (u, v, ddict[data]). If true, return edge attribute dict : 3-tuple (u, v, ddict). If false, return 2-tuple (u, v). default : value, optional (default=None); Value used for edges that don"t have the requested attribute. Only relevant if (data is not true or false.

Returns

edges : *EdgeView* A view of edge attributes, usually it iterates over (u, v); or (u, v, d) tuples of edges, but can also be used for attribute lookup as `edges[u, v]["foo"]`.

Notes

Nodes : nbunch that are not : the graph will be (quietly) ignored. For directed graphs this returns the out-edges.

Examples

[(0, 1), (1, 2), (2, 3)]; EdgeDataView([(0, 1, {}), (1, 2, {}), (2, 3, {"weight": 5})]); EdgeDataView([(0, 1, 1), (1, 2, 1), (2, 3, 5)]); EdgeDataView([(0, 1), (3, 2)]); G.adj[0]?); EdgeDataView([(0, 1)]);

```
A DegreeView for the Graph as G.degree or G.degree().
```

The node degree is the number of edges adjacent to the node. The weighted node degree is the sum of the edge weights for edges incident to that node.

This object provides an iterator for (node, degree) as well as lookup for the degree for a single node.

Parameters

nbunch : single node, container, or all nodes (default= all nodes); The view will only report edges incident to these nodes.

weight : string or None, optional (default=None); The name of an edge attribute that holds the numerical value used as a weight. If None, then each edge has weight 1. The degree is the sum of the edge weights adjacent to the node.

Returns

If a single node is requested deg : int Degree of the node

OR if (multiple nodes are requested nd_view : A DegreeView object capable of iterating (node, degree) pairs

Examples

etc 1 [(0, 1), (1, 2), (2, 2)];

inline auto **is_multigraph**()
Return true if (graph is a multigraph, false otherwise.

inline auto **is_directed**()
Return true if (graph is directed, false otherwise.

Public Members

size_t **_num_of_edges** = 0

nodeview_t **_node**

graph_attr_dict_factory **graph** = { }

adjlist_outer_dict_factory **_adj**

Public Static Functions

static inline auto **end_points**(*edge_t* &e) -> *edge_t*&
For compatible with BGL adaptor.

Parameters **e** – **[in]**

Returns edge_t&

static inline auto **end_points**(const *edge_t* &e) -> const *edge_t*&
For compatible with BGL adaptor.

Parameters **e** – **[in]**

Returns edge_t&

Template Class NodeView

- Defined in file_xnetwork_classes_reportviews.hpp

Page Contents

Template Parameter Order

1. `typename nodeview_t`

Class Documentation

template<typename **nodeview_t**>

class `xn::NodeView`
A *NodeView* class to act as G.nodes for a XNetwork *Graph* Set operations act on the nodes without considering data. Iteration is over nodes. Node data can be looked up like a dict. Use NodeDataView to iterate over node data or to specify a data attribute for lookup. NodeDataView is created by calling the *NodeView*.

Parameters

graph : XNetwork graph-like class

Examples

true 0 1 2 {“color”: “blue”} true (0, “aqua”); (1, “aqua”); (2, “blue”); (8, “red”); true true (0, “aqua”); (1, “aqua”); (2, “blue”); (8, “red”); false

Public Functions

inline explicit **NodeView**(*nodeview_t* &nodes)

inline auto **size**()

inline auto **begin**()

inline auto **end**()

inline auto **operator[]**(const Node &n) const -> const auto&

inline auto **operator[]**(const Node &n) -> auto&

inline auto **contains**(const Node &n) -> bool

Template Class VertexView

- Defined in file_py2cpp_nx2bgl.hpp

Page Contents

Inheritance Relationships

Base Type

- `public xn::Graph< _nodeview_t, adjlist_t, adjlist_outer_dict_factory >` (*Template Class Graph*)

Derived Type

- `public xn::grAdaptor< Graph >` (*Template Class grAdaptor*)

Template Parameter Order

1. **`Graph <exhale_class_classxn_1_1Graph_>`_**

Class Documentation

template<typename **Graph**>

class **xn::VertexView** : public xn::*Graph*<_nodeview_t, adjlist_t, adjlist_outer_dict_factory>

tparam Graph

Subclassed by *xn::grAdaptor< Graph >*

Public Functions

inline explicit **VertexView**(*Graph* &&G) noexcept
Construct a new Vertex View object.

Parameters **G** – **[in]**

inline auto **begin**() const

Returns auto

inline auto **end**() const

Returns auto

inline auto **cbegin**() const

Returns auto

inline auto **cend**() const

Returns auto

1.3.3 Enums

Enum LegalCheck

- Defined in file_ckpttncpp_FMConstrMgr.hpp

Enum Documentation

enum **LegalCheck**
Check if the move of v can satisfied, getbetter, or notsatisfied.

Values:

enumerator **notsatisfied**

enumerator **getbetter**

enumerator **allsatisfied**

1.3.4 Functions

Function create_contraction_subgraph

- Defined in file_ckpttncpp_MLPartMgr.hpp

Function Documentation

Warning: doxygenfunction: Unable to resolve function "create_contraction_subgraph" with arguments (const SimpleNetlist&, const py::set<node_t>&) in doxygen xml output for project "ckpttncpp" from directory: ./doxy-output/xml. Potential matches:

```
- auto create_contraction_subgraph(const SimpleNetlist&, const py::set<node_t>&) ->␣
↪std::unique_ptr<SimpleHierNetlist>
```

Template Function fun::gcd(_Mn, _Mn)

- Defined in file_py2cpp_fractions-new.hpp

Function Documentation

template<typename **_Mn**>
constexpr *_Mn* **fun::gcd**(*_Mn* __m, *_Mn* __n)
Greatest common divider.

Template Parameters **_Mn** –

Parameters

- **__m** – **[in]**
- **__n** – **[in]**

Returns _Mn

Template Function fun::gcd(Mn, Mn)

- Defined in file_py2cpp_fractions.hpp

Function Documentation

Warning: doxygenfunction: Unable to resolve function "fun::gcd" with arguments (Mn, Mn) in doxygen xml output for project "ckpttncpp" from directory: ./doxyoutput/xml. Potential matches:

```
- template<typename Mn> constexpr auto gcd(Mn _m, Mn _n) -> Mn
- template<typename _Mn> constexpr _Mn gcd(_Mn __m, _Mn __n)
```

Template Function fun::lcm(_Mn, _Mn)

- Defined in file_py2cpp_fractions-new.hpp

Function Documentation

template<typename **_Mn**>
constexpr *_Mn* fun::**lcm**(*_Mn* __m, *_Mn* __n)

Least common multiple.

Template Parameters **_Mn** –

Parameters

- **__m** – **[in]**
- **__n** – **[in]**

Returns _Mn

Template Function fun::lcm(Mn, Mn)

- Defined in file_py2cpp_fractions.hpp

Function Documentation

Warning: doxygenfunction: Unable to resolve function "fun::lcm" with arguments (Mn, Mn) in doxygen xml output for project "ckpttncpp" from directory: ./doxyoutput/xml. Potential matches:

```
- template<typename Mn> constexpr auto lcm(Mn _m, Mn _n) -> Mn
- template<typename _Mn> constexpr _Mn lcm(_Mn __m, _Mn __n)
```

Template Function fun::operator*

- Defined in file_py2cpp_fractions.hpp

Function Documentation

Warning: doxygenfunction: Unable to resolve function "fun::operator*" with arguments (int&&, const Fraction<Z>&) in doxygen xml output for project "ckpttncpp" from directory: ./doxyoutput/xml. Potential matches:

```
- template<typename Z> constexpr auto operator*(int &&c, const Fraction<Z> &frac) ->␣
↪Fraction<Z>
```

Template Function fun::operator+(const Z&, const Fraction<Z>&)

- Defined in file_py2cpp_fractions.hpp

Function Documentation

Warning: doxygenfunction: Unable to resolve function "fun::operator+" with arguments (const Z&, const Fraction<Z>&) in doxygen xml output for project "ckpttncpp" from directory: ./doxyoutput/xml. Potential matches:

```
- template<typename Z> constexpr auto operator+(const Z &c, const Fraction<Z> &frac) -
↪> Fraction<Z>
- template<typename Z> constexpr auto operator+(int &&c, const Fraction<Z> &frac) ->␣
↪Fraction<Z>
```

Template Function fun::operator+(int&&, const Fraction<Z>&)

- Defined in file_py2cpp_fractions.hpp

Function Documentation

Warning: doxygenfunction: Unable to resolve function "fun::operator+" with arguments (int&&, const Fraction<Z>&) in doxygen xml output for project "ckpttncpp" from directory: ./doxyoutput/xml. Potential matches:

```
- template<typename Z> constexpr auto operator+(const Z &c, const Fraction<Z> &frac) -
↪> Fraction<Z>
- template<typename Z> constexpr auto operator+(int &&c, const Fraction<Z> &frac) ->␣
↪Fraction<Z>
```

Template Function fun::operator-(const Z&, const Fraction<Z>&)

- Defined in file_py2cpp_fractions.hpp

Function Documentation

Warning: doxygenfunction: Unable to resolve function "fun::operator-" with arguments (const Z&, const Fraction<Z>&) in doxygen xml output for project "ckpttncpp" from directory: ./doxyoutput/xml. Potential matches:

```
- template<typename Z> constexpr auto operator-(const Z &c, const Fraction<Z> &frac) -
↪> Fraction<Z>
- template<typename Z> constexpr auto operator-(int &&c, const Fraction<Z> &frac) ->␣
↪Fraction<Z>
```

Template Function fun::operator-(int&&, const Fraction<Z>&)

- Defined in file_py2cpp_fractions.hpp

Function Documentation

Warning: doxygenfunction: Unable to resolve function "fun::operator-" with arguments (int&&, const Fraction<Z>&) in doxygen xml output for project "ckpttncpp" from directory: ./doxyoutput/xml. Potential matches:

```
- template<typename Z> constexpr auto operator-(const Z &c, const Fraction<Z> &frac) -
↪> Fraction<Z>
- template<typename Z> constexpr auto operator-(int &&c, const Fraction<Z> &frac) ->␣
↪Fraction<Z>
```

Template Function fun::operator<<

- Defined in file_py2cpp_fractions.hpp

Function Documentation

Warning: doxygenfunction: Unable to resolve function "fun::operator<<" with arguments (Stream&, const Fraction<Z>&) in doxygen xml output for project "ckpttncpp" from directory: ./doxyoutput/xml. Potential matches:

```
- template<typename Stream, typename Z> auto operator<<(Stream &os, const Fraction<Z>␣
↪&frac) -> Stream&
```

Template Function get_repeat_array

- Defined in file_ckpttncpp_array_like.hpp

Function Documentation

template<typename **Val**>
inline auto **get_repeat_array**(const *Val* &a, std::ptrdiff_t n)

Template Function min_maximal_matching

- Defined in file_ckpttncpp_netlist_algo.hpp

Function Documentation

Warning: doxygenfunction: Unable to resolve function "min_maximal_matching" with arguments (const Netlist&, const C1&, C2&&, C2&&) in doxygen xml output for project "ckpttncpp" from directory: ./doxyoutput/xml. Potential matches:

```
- template<typename Netlist, typename C1, typename C2> auto min_maximal_
↪matching(const Netlist &H, const C1 &weight, C2 &&matchset, C2 &&dep) -> typename␣
↪C1::mapped_type
```

Template Function min_vertex_cover

- Defined in file_ckpttncpp_netlist_algo.hpp

Function Documentation

Warning: doxygenfunction: Unable to resolve function "min_vertex_cover" with arguments (const Netlist&, const C1&, C2&) in doxygen xml output for project "ckpttncpp" from directory: ./doxyoutput/xml. Potential matches:

```
- template<typename Netlist, typename C1, typename C2> auto min_vertex_cover(const␣
↪Netlist &H, const C1 &weight, C2 &coverset) -> typename C1::mapped_type
```

Template Function py::len(const set<Key>&)

- Defined in file_py2cpp_py2cpp.hpp

Function Documentation

Warning: doxygenfunction: Unable to resolve function "py::len" with arguments (const set<Key>&) in doxygen xml output for project "ckpttncpp" from directory: ./doxyoutput/xml. Potential matches:

```
- template<typename Key, typename T> auto len(const dict<Key, T> &m) -> size_t
- template<typename Key> auto len(const set<Key> &m) -> size_t
```

Template Function py::len(const dict<Key, T>&)

- Defined in file_py2cpp_py2cpp.hpp

Function Documentation

Warning: doxygenfunction: Unable to resolve function "py::len" with arguments (const dict<Key, T>&) in doxygen xml output for project "ckpttncpp" from directory: ./doxyoutput/xml. Potential matches:

```
- template<typename Key, typename T> auto len(const dict<Key, T> &m) -> size_t
- template<typename Key> auto len(const set<Key> &m) -> size_t
```

Template Function py::operator<(const Key&, const set<Key>&)

- Defined in file_py2cpp_py2cpp.hpp

Function Documentation

Warning: doxygenfunction: Unable to resolve function "py::operator<" with arguments (const Key&, const set<Key>&) in doxygen xml output for project "ckpttncpp" from directory: ./doxyoutput/xml. Potential matches:

```
- template<typename Key, typename T> auto operator<(const Key &key, const dict<Key, T>
↪ &m) -> bool
- template<typename Key> auto operator<(const Key &key, const set<Key> &m) -> bool
```

Template Function py::operator<(const Key&, const dict<Key, T>&)

- Defined in file_py2cpp_py2cpp.hpp

Function Documentation

Warning: doxygenfunction: Unable to resolve function "py::operator<" with arguments (const Key&, const dict<Key, T>&) in doxygen xml output for project "ckpttncpp" from directory: ./doxyoutput/xml. Potential matches:

```
- template<typename Key, typename T> auto operator<(const Key &key, const dict<Key, T>
↪ &m) -> bool
- template<typename Key> auto operator<(const Key &key, const set<Key> &m) -> bool
```

Template Function py::range(T, T)

- Defined in file_py2cpp_py2cpp.hpp

Function Documentation

template<typename **T**>
inline constexpr auto **py::range**(*T* start, *T* stop)

Template Function py::range(T)

- Defined in file_py2cpp_py2cpp.hpp

Function Documentation

template<typename **T**>
inline auto **py::range**(*T* stop)

Function set_partition

- Defined in file_ckpttncpp_set_partition.hpp

Function Documentation

coro_t::pull_type **set_partition**(int n, int k)

Template Function Stirling2nd

- Defined in file_ckpttncpp_set_partition.hpp

Function Documentation

template<int **N**, int **K**>
constexpr auto **Stirling2nd**()

1.3.5 Variables

Variable FM_MAX_DEGREE

- Defined in file_ckpttncpp_FMPmrConfig.hpp

Variable Documentation

const auto **FM_MAX_DEGREE** = 65536U

Variable FM_MAX_NUM_PARTITIONS

- Defined in file_ckpttncpp_FMPmrConfig.hpp

Variable Documentation

const auto **FM_MAX_NUM_PARTITIONS** = 255U

Variable xn::__slots__

- Defined in file_xnetwork_classes_reportviews.hpp

Variable Documentation

static const auto **xn::__slots__** = ()

A DataView class for nodes of a XNetwork *Graph*

The main use for this class is to iterate through node-data pairs. The data can be the entire data-dictionary for each node, or it can be a specific attribute (with default) for each node. Set operations are enabled with NodeDataView, but don't work in cases where the data is not hashable. Use with caution. Typically, set operations on nodes use *NodeView*, not NodeDataView. That is, they use `G.nodes` instead of `G.nodes(data="foo")`.

Parameters

graph : XNetwork graph-like class data : bool or string (default=false); default : object (default=None);

```
A View class for degree of nodes : a XNetwork Graph
```

The functionality is like dict.items() with (node, degree) pairs. Additional functionality includes read-only lookup of node degree, and calling with optional features nbunch (for only a subset of nodes); and weight (use edge weights to compute degree).

Parameters

graph : XNetwork graph-like class nbunch : node, container of nodes, or None meaning all nodes (default=None); weight : bool or string (default=None);

Notes

DegreeView can still lookup any node even if (nbunch is specified.

Examples

34 1 70

```
A DegreeView class to act as G.degree for a XNetwork Graph
```

Typical usage focuses on iteration over `(node, degree)` pairs. The degree is by default the number of edges incident to the node. Optional argument `weight` enables weighted degree using the edge attribute named : the `weight` argument. Reporting and iteration can also be restricted to a subset of nodes using `nbunch`.

Additional functionality include node lookup so that `G.degree[n]` reported the (possibly weighted) degree of node `n`. Calling the view creates a view with different arguments `nbunch` or `weight`.

Parameters

graph : XNetwork graph-like class nbunch : node, container of nodes, or None meaning all nodes (default=None); weight : string or None (default=None);

Notes

DegreeView can still lookup any node even if (nbunch is specified.

Examples

34 1 70

A DegreeView class to report out_degree for a DiGraph; See DegreeView

A DegreeView class to report in_degree for a DiGraph; See DegreeView

A DegreeView class for undirected multigraphs; See DegreeView

A DegreeView class for MultiDiGraph; See DegreeView

A DegreeView class for inward degree of MultiDiGraph; See DegreeView

A DegreeView class for outward degree of MultiDiGraph; See DegreeView

EdgeDataView for outward edges of DiGraph; See EdgeDataView

```
A EdgeDataView class for edges of Graph
```

This view is primarily used to iterate over the edges reporting edges as node-tuples with edge data optionally reported. The argument `nbunch` allows restriction to edges incident to nodes : that container/singleton. The default (nbunch=None); reports all edges. The arguments `data` and `default` control what edge data is reported. The default `data == false` reports only node-tuples for each edge. If `data is true` the entire edge data dict is returned. Otherwise `data` is assumed to hold the name of the edge attribute to report with default `default` if (that edge attribute is not present.

Parameters

nbunch : container of nodes, node or None (default None); data : false, true or string (default false); default : default value (default None);

Examples

[(0, 1, “biz”), (1, 2, “bar”)];

An EdgeDataView class for outward edges of DiGraph; See EdgeDataView

An EdgeDataView for outward edges of MultiDiGraph; See EdgeDataView

1.3.6 Typedefs

Typedef coro_t

- Defined in file_ckpttncpp_set_partition.hpp

Typedef Documentation

using **coro_t** = boost::coroutines2::coroutine<ret_t>

Typedef graph_t

- Defined in file_ckpttncpp_netlist.hpp

Typedef Documentation

using **graph_t** = xn::*SimpleGraph*

Typedef index_t

- Defined in file_ckpttncpp_netlist.hpp

Typedef Documentation

using HierNetlist::**index_t** = typename nodeview_t::key_type

Typedef node_t

- Defined in file_ckpttncpp_MLPartMgr.hpp

Typedef Documentation

using *FMBiGainCalc*::**node_t** = typename *SimpleNetlist*::node_t

Typedef ret_t

- Defined in file_ckpttncpp_set_partition.hpp

Typedef Documentation

using *set_partition_*::**ret_t** = std::tuple<int, int>

Typedef SimpleNetlist

- Defined in file_ckpttncpp_netlist.hpp

Typedef Documentation

using **SimpleNetlist** = *Netlist*<*graph_t*>

Typedef SimpleNetlist

- Defined in file_ckpttncpp_PartMgrBase.hpp

Typedef Documentation

using **SimpleNetlist** = *Netlist*<*graph_t*>

Typedef Value_type

- Defined in file_py2cpp_py2cpp.hpp

Typedef Documentation

using **Value_type** = typename T::value_type

Typedef xn::SimpleGraph

- Defined in file_xnetwork_classes_graph.hpp

Typedef Documentation

using **xn::SimpleGraph** = *Graph*<decltype(py::*range*<uint32_t>(uint32_t{})), py::*set*<uint32_t>, std::vector<py::*set*<uint32_t>>>

You read all the way to the bottom?! This text is specified by giving an argument to `afterBodySummary`. As the docs state, this summary gets put in after a **lot** of information. It's available for you to use if you want it, but from a design perspective it's rather unlikely any of your users will even see this text.

CHAPTER

TWO

HOW THIS VERSION OF CKPTTNCPP WAS CREATED

For convenience, I'm going to inline the code used in this configuration from `conf.py` here. The three main things you need to do here are

1. The `requirements.txt` used on read the docs.
2. Setup the `breathe` and `exhale` extensions.
3. Choose your `html_theme`, which affects what you choose for the `exhale` side.

Refer to the Start to finish for Read the Docs tutorial for getting everything setup on RTD.

2.1 requirements.txt

```
# for testing the master branch
# git+git://github.com/svenevs/exhale.git#egg=exhale
# See: https://exhale.readthedocs.io/en/latest/#exhale-version-compatibility-with-python-
↪sphinx-and-breathe
sphinx>=2.0
sphinx-bootstrap-theme>=0.4.0
breathe>=4.13.0
exhale
```

2.2 Extension Setup

```
# Tell Sphinx to use both the `breathe` and `exhale` extensions
extensions = [
    'breathe',
    'exhale'
]

# Setup the `breathe` extension
breathe_projects = {"ckpttncpp": "./doxyoutput/xml"}
breathe_default_project = "ckpttncpp"

# Setup the `exhale` extension
# import textwrap
exhale_args = {
    ############################################################################
```

(continues on next page)

(continued from previous page)

```
    # These arguments are required.                                          #
    ##########################################################################
    "containmentFolder":     "./api",
    "rootFileName":          "library_root.rst",
    "rootFileTitle":         "Library API",
    "doxygenStripFromPath":  "../lib/include",
    ##########################################################################
    # Suggested optional arguments.                                          #
    ##########################################################################
    "createTreeView":        True,
    "exhaleExecutesDoxygen": True,
    "exhaleDoxygenStdin": textwrap.dedent('''
        INPUT       = ../lib/include
        # For this code-base, the following helps Doxygen get past a macro
        # that it has trouble with.  It is only meaningful for this code,
        # not for yours.
        PREDEFINED += NAMESPACE_BEGIN(arbitrary)="namespace arbitrary {"
        PREDEFINED += NAMESPACE_END(arbitrary)="}"
    '''),
    ##########################################################################
    # HTML Theme specific configurations.                                    #
    ##########################################################################
    # Fix broken Sphinx RTD Theme 'Edit on GitHub' links
    # Search for 'Edit on GitHub' on the FAQ:
    #     http://exhale.readthedocs.io/en/latest/faq.html
    "pageLevelConfigMeta": ":github_url: https://github.com/svenevs/exhale-companion",
    ##########################################################################
    # Main library page layout example configuration.                        #
    ##########################################################################
    "afterTitleDescription": textwrap.dedent(u'''
        Welcome to the developer reference to Exhale Companion.  The code being
        documented here is largely meaningless and was only created to test
        various corner cases e.g. nested namespaces and the like.

        .. note::

            The text you are currently reading was fed to ``exhale_args`` using
            the :py:data:`~exhale.configs.afterTitleDescription` key.  Full
            reStructuredText syntax can be used.

        .. tip::

           Sphinx / Exhale support unicode!  You're ``conf.py`` already has
           it's encoding declared as ``# -*- coding: utf-8 -*-`` **by
           default**.  If you want to pass Unicode strings into Exhale, simply
           prefix them with a ``u`` e.g. ``u""`` (of course you would
           actually do this because you are writing with åçćëñtß or
           non-English  ).
    '''),
    "afterHierarchyDescription": textwrap.dedent('''
        Below the hierarchies comes the full API listing.
```

(continues on next page)

(continued from previous page)

```
        1. The text you are currently reading is provided by
           :py:data:`~exhale.configs.afterHierarchyDescription`.
        2. The Title of the next section *just below this* normally defaults to
           ``Full API``, but the title was changed by providing an argument to
           :py:data:`~exhale.configs.fullApiSubSectionTitle`.
        3. You can control the number of bullet points for each linked item on
           the remainder of the page using
           :py:data:`~exhale.configs.fullToctreeMaxDepth`.
    '''),
    "fullApiSubSectionTitle": "Custom Full API SubSection Title",
    "afterBodySummary": textwrap.dedent('''
        You read all the way to the bottom?!  This text is specified by giving
        an argument to :py:data:`~exhale.configs.afterBodySummary`.  As the docs
        state, this summary gets put in after a **lot** of information.  It's
        available for you to use if you want it, but from a design perspective
        it's rather unlikely any of your users will even see this text.
    '''),
    ############################################################################
    # Individual page layout example configuration.                            #
    ############################################################################
    # Example of adding contents directives on custom kinds with custom title
    "contentsTitle": "Page Contents",
    "kindsWithContentsDirectives": ["class", "file", "namespace", "struct"],
    # This is a testing site which is why I'm adding this
    "includeTemplateParamOrderList": True,
    ############################################################################
    # useful to see ;)
    "verboseBuild": True
}

# Tell sphinx what the primary language being documented is.
primary_domain = 'cpp'

# Tell sphinx what the pygments highlight language should be.
highlight_language = 'cpp'
```

2.3 HTML Theme Setup

```
# The name of the Pygments (syntax highlighting) style to use.
# `sphinx` works very well with the RTD theme, but you can always change it
pygments_style = 'sphinx'

# on_rtd is whether we are on readthedocs.org, this line of code grabbed from docs.
↪readthedocs.org
on_rtd = os.environ.get('READTHEDOCS', None) == 'True'

if not on_rtd:  # only import and set the theme if we're building docs locally
    import sphinx_bootstrap_theme
    html_theme = 'bootstrap'
    html_theme_path = sphinx_bootstrap_theme.get_html_theme_path()
```

CHAPTER

THREE

USING INTERSPHINX

The Sphinx intersphinx extension is exceptionally convenient, and typically works out-of-the-box for most projects you would want to link to. This is not limited to linking to documents just within your domain, and if you really want to go the extra mile (and create your own mapping), it doesn't even have to be restricted to linking to documentation that was generated with Sphinx.

Contents

3.1 Setup your `conf.py`

First, how you link to things depends on what your domain is. In the Exhale Quickstart Guide, I encouraged you to add these lines to your `conf.py`:

```
# Tell sphinx what the primary language being documented is.
primary_domain = 'cpp'

# Tell sphinx what the pygments highlight language should be.
highlight_language = 'cpp'
```

This will come up in the next section, but is added to `conf.py` so it is included here.

For this ckpttncpp project, I want to link to two Sphinx generated projects. In the `conf.py`, this means that I have:

```
# In addition to `breathe` and `exhale`, use the `intersphinx` extension
extensions = [
    'sphinx.ext.intersphinx',
    'breathe',
    'exhale'
```

(continues on next page)

(continued from previous page)

```
]

# Specify the baseurls for the projects I want to link to
intersphinx_mapping = {
    'exhale':  ('https://exhale.readthedocs.io/en/latest/', None),
    'nanogui': ('http://nanogui.readthedocs.io/en/latest/', None)
}
```

3.2 Linking to Other Sites Using Intersphinx

This is where understanding your primary domain becomes particularly relevant. Since the `primary_domain` for this project is `cpp`, I can link to things like `:cpp:function:` as just `:function:`. But if I want to link to Python or C domains I need to specify that explicitly. Inlined from the Cross Referencing Syntax docs, there is some syntax you will likely need to wield:

- You may supply an explicit title and reference target: `:role:`title <target>`` will refer to target, but the link text will be title.
- If you prefix the content with `!`, no reference/hyperlink will be created.
- If you prefix the content with `~`, the link text will only be the last component of the target. For example, `:py:meth:`~Queue.Queue.get`` will refer to `Queue.Queue.get` but only display `get` as the link text.

3.2.1 Linking to Python Docs from a cpp Project

Since I've setup intersphinx to point back to the main Exhale site, I'll just link to some from there.

Linking to a Python Class

:py:class:`exhale.graph.ExhaleRoot` Links to `exhale.graph.ExhaleRoot`

:py:class:`graph.ExhaleRoot <exhale.graph.ExhaleRoot>` Links to `graph.ExhaleRoot`

:py:class:`~exhale.graph.ExhaleRoot` Links to `ExhaleRoot`

Linking to a Python Function

:py:func:`exhale.deploy.explode` Links to `exhale.deploy.explode()`

:py:func:`deploy.explode <exhale.deploy.explode>` Links to `deploy.explode`

:py:func:`~exhale.deploy.explode` Links to `explode()`

3.2.2 Linking to Another C++ Project

This is where understanding how to manipulate the link titles becomes relevant. I'll use the NanoGUI docs since I stole the `NAMESPACE_BEGIN` macro from there.

Linking to a C++ Class Using a single `:` does not appear to work, but using the `namespace::ClassName` seems to include a leading `:`. I think this is a bug, but solving it would likely be treacherous so instead just control the title yourself.

:class:`nanogui::Screen` Links to `:Screen`

:class:`nanogui::Screen <nanogui::Screen>` Links to `nanogui::Screen`

:class:`~nanogui::Screen` Links to Screen

Linking to C Domains Even if the other project is primarily C++, things like macros are in the :c: Sphinx domain. I choose the NAMESPACE_BEGIN example to show you how to qualify where Sphinx should link — both **this project** and **NanoGUI** have links to it, so when I just do :c:macro:`NAMESPACE_BEGIN` the link (NAMESPACE_BEGIN) goes to **this project**. Using nanogui:NAMESPACE_BEGIN (since 'nanogui' was a key in our intersphinx_mapping)

:c:macro:`nanogui:NAMESPACE_BEGIN` Links to NAMESPACE_BEGIN

:c:macro:`NanoGUI macro NAMESPACE_BEGIN <nanogui:NAMESPACE_BEGIN>` Links to NanoGUI macro NAMESPACE_BEGIN

:c:macro:`~nanogui:NAMESPACE_BEGIN` Links to NAMESPACE_BEGIN

Tip: These kinds of cross references are **reStructuredText** syntax! You **must** enable the \rst environment for Doxygen (see Doxygen ALIASES) **and** use this in the documentation. For example, in order to get the NAMESPACE_BEGIN link to work, the actual C++ code is as follows:

```
#if !defined(NAMESPACE_BEGIN) || defined(DOXYGEN_DOCUMENTATION_BUILD)
    /**
     * \rst
     * See :c:macro:`NanoGUI macro NAMESPACE_BEGIN <nanogui:NAMESPACE_BEGIN>`.
     * \endrst
     */
    #define NAMESPACE_BEGIN(name) namespace name {
#endif
```

3.3 Finding the Links to Use

For things like classes that are qualified in namespaces, it should be pretty easy for you to figure out what the link is by inspection. However, there is an excellent tool available for you: the Sphinx Objects.inv Encoder/Decoder.

1. Install the utility:

```
$ pip install sphobjinv
```

2. Download the Sphinx objects.inv for the project you want to use. This should be at the location you specified in your intersphinx_mapping. So if the URL you gave was url, the objects.inv should be at url/objects.inv. Sticking with the NanoGUI example:

```
# Go to wherever you want and download the file
$ cd /tmp

# That's a capital 'Oh' not a zero; or use `wget`
$ curl -O http://nanogui.readthedocs.io/en/latest/objects.inv
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 44056  100 44056    0     0   109k      0 --:--:-- --:--:-- --:--:--  109k

# rename it so you know where it hails from
$ mv objects.inv nanogui_objects.inv
```

3. Decode it to plain text and search for what you are trying to link.

```
# decode it so we can search it
$ sphobjinv convert plain nanogui_objects.inv

Conversion completed.
'nanogui_objects.inv' decoded to 'nanogui_objects.txt'.

# search for the thing you are trying to link to
$ grep NAMESPACE_BEGIN nanogui_objects.txt | grep -v -- -1
                vvvvvvv
NAMESPACE_BEGIN c:macro 1 api/define_NAMESPACE_BEGIN.html#c.$ -
                ^^^^^^^
```

Tip: Refer to the sphobjinv syntax section, the reason I am piping to `grep -v -- -1` is because "priority" `-1` means it won't be available to link to. The `-v` tells `grep` to invert the match, and `--` tells `grep` that the command-line options (e.g., `-v`) are finished and what follows is an argument. That is, `-- -1` just makes it so `grep` doesn't think `-1` is a flag.

3.3.1 Custom Links

You can also make your own `intersphinx` mappings. I did this for linking to the BeautifulSoup docs. See the _intersphinx/README.md of Exhale.

This use case was for a dysfunctional `objects.inv`, but you could also easily create your own mapping to index a project that was not created using Sphinx.

3.4 Testing your Intersphinx Links

By default the Sphinx build process does not inform you of broken link targets when you run `make html`. The `sphinx-build` flag you want for testing this is `-n` (for *nitpicky*). You will want to make sure to `clean` first so that all errors get shown.

```
$ make SPHINXOPTS='-n' clean html
```

Tip: There is also a `make linkcheck` target for the Sphinx generated Makefiles!

Note: This was built using Exhale version 0.2.3.

Make sure to view the *Extension Setup* and *HTML Theme Setup* for the different versions, as they vary slightly (e.g., `bootstrap` gets more supplied in the `exhale_args` portion of `conf.py`).

INDEX

S

V

X