Data Structures - 20407 - Maman 13

Generated by Doxygen 1.9.1

1 Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

- **MergeableHeap< T >** 5
  - **LazyBinomialHeap< T >** 2

2 Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

LazyBinomialHeap< T >
A mergeable heap data structure implementation using lazy binomial heap 2

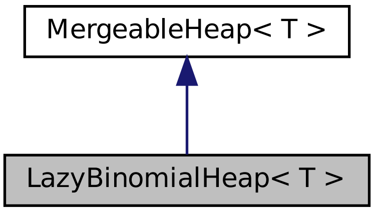
MergeableHeap< T >
A constexpr-friendly interface for a mergeable heap data structure 5

3 Class Documentation

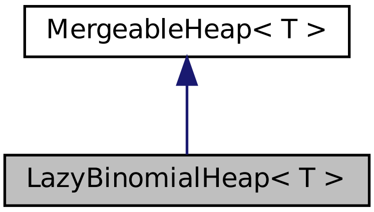
3.1 LazyBinomialHeap< T > Class Template Reference

A mergeable heap data structure implementation using lazy binomial heap.

Inheritance diagram for LazyBinomialHeap< T >:



Collaboration diagram for LazyBinomialHeap< T >:



Public Member Functions

- constexpr LazyBinomialHeap () noexcept

  Constructs a new empty heap.
- constexpr ∼LazyBinomialHeap ()=default

  Destroys the heap.
- constexpr void insert (T key) override

  Inserts a key into the heap.
- constexpr std::optional< std::reference_wrapper< const T > > minimum () const noexcept override

  Returns the minimum key in the heap.
- constexpr std::optional< T > extract_min () override

  Removes and returns the minimum key in the heap.
- constexpr void merge (MergeableHeap< T > &other) override

  Merges another heap into this heap.

3.1.1 Detailed Description

template<typename T>
class LazyBinomialHeap< T >

This mergeable heap is implemented using the binomial heap data structure. A binomial heap is a collection of binomial trees, where each binomial tree is a heap-ordered tree. A binomial tree of order k is a tree with 2^k nodes, where each node has a key and a pointer to its parent and its leftmost child. The binomial heap is a collection of binomial trees of different orders, where each order is represented by a linked list of binomial trees. The root list of the heap is a linked list of the roots of the binomial trees in the heap, hence its size is logarithmic in the number of nodes in the heap in the average case. The current implementation of the binomial heap is lazy, meaning that the heap is consolidated only when needed, during the extract_min operation. This allows for O(1) insertions and merges, and an amortized O(log n) extract_min. Amortized analysis is used to analyze the time complexity of the consolidate operation,

Template Parameters

T	The type of the elements stored in the heap. `T` must be movable, as it is moved into the heap in the insert method.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 LazyBinomialHeap()

```
template<typename T >
constexpr LazyBinomialHeap< T >::LazyBinomialHeap ( )  [inline], [constexpr], [noexcept]
```

This constructor initializes the heap to be empty. The head, tail, and minimum node pointers are all set to `nullptr`, and the size is set to 0.

The time complexity of this operation is O(1).

3.1.2.2 ∼LazyBinomialHeap()

```
template<typename T >
constexpr LazyBinomialHeap< T >::∼LazyBinomialHeap ( )  [constexpr], [default]
```

This destructor deletes the head node of the heap, which recursively deletes all nodes in the heap.

The time complexity of this operation is O(n), where n is the number of nodes in the heap.

3.1.3 Member Function Documentation

3.1.3.1 extract_min() `template<typename T > constexpr std::optional<T> LazyBinomialHeap< T >::extract_min ( ) [inline], [constexpr], [override], [virtual]`

This method removes the minimum key from the heap and returns it. If the minimum key has children, they are added to the root list. The heap is then consolidated to maintain the heap property. If the heap is empty, `std::nullopt` is returned.

The time complexity of this operation is O(log n) in the average case (amotrized), and O(n) in the worst case where n is the number of nodes in the heap. This is because the heap is consolidated after the minimum key is removed, which involves linking binomial trees of the same degree until there are no two trees with the same degree.

Returns

The minimum key, or `std::nullopt` if the heap is empty.

Implements MergeableHeap< T >.

3.1.3.2 insert() `template<typename T > constexpr void LazyBinomialHeap< T >::insert ( T key ) [inline], [constexpr], [override], [virtual]`

This method inserts a new key into the heap and updates the minimum accordingly.

The time complexity of this operation is O(1), because it only involves creating a new node and updating the head and minimum pointers. The heap is not consolidated after each insertion, so the heap may become unbalanced. This is fine because the heap is consolidated lazily only when needed, during the extract_min operation.

Parameters

key	The key to insert. This key is moved into the heap.

Implements MergeableHeap< T >.

3.1.3.3 merge() `template<typename T > constexpr void LazyBinomialHeap< T >::merge ( MergeableHeap< T > & other ) [inline], [constexpr], [override], [virtual]`

This method merges another heap into this heap. The root list of the other heap is concatenated to the root list of this heap, and the size of this heap is increased by the size of the other heap, and the minimum is updated accordingly.

Note

The other heap is left empty after the merge.

The time complexity of this operation is O(1), because it only involves updating the head, tail, minimum and size pointers of this heap. The heap is not consolidated after the merge, so the heap may become unbalanced. This is fine because the heap is consolidated lazily only when needed, during the extract_min operation.

Parameters

other	The heap to merge into this heap.

Implements MergeableHeap< T >.

3.1.3.4 minimum()

```
template<typename T >
constexpr std::optional<std::reference_wrapper<const T> > LazyBinomialHeap< T >::minimum ( )
const [inline], [constexpr], [override], [virtual], [noexcept]
```

This method returns a reference to the minimum key in the heap, or `std::nullopt` if the heap is empty. The reference is wrapped in a `std::reference_wrapper` to allow it to be used in contexts where a reference cannot be used.

The time complexity of this operation is O(1), because the minimum key is stored in the `min` pointer, which is updated whenever a new minimum key is inserted.

Returns

A reference to the minimum key, or `std::nullopt` if the heap is empty.

Implements MergeableHeap< T >.

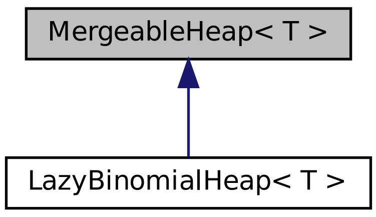
The documentation for this class was generated from the following file:

- lazy.h

3.2 MergeableHeap< T > Class Template Reference

A constexpr-friendly interface for a mergeable heap data structure.

Inheritance diagram for MergeableHeap< T >:



Public Member Functions

- constexpr MergeableHeap ()=default
- constexpr virtual ~MergeableHeap ()=default
- constexpr MergeableHeap (const MergeableHeap &)=delete

  Deleted special member functions.
- constexpr MergeableHeap & **operator=** (const MergeableHeap &)=delete
- constexpr **MergeableHeap** (MergeableHeap &&)=delete
- constexpr MergeableHeap & **operator=** (MergeableHeap &&)=delete
- constexpr virtual void insert (T key)=0
- constexpr virtual std::optional< std::reference_wrapper< const T > > minimum () const =0
- constexpr virtual std::optional< T > extract_min ()=0
- constexpr virtual void merge (MergeableHeap< T > &other)=0

3.2.1 Detailed Description

template<typename T>
class MergeableHeap< T >

A mergeable heap is a type of heap that supports the following operations:

1. MAKE-HEAP creates a new (empty) heap.
2. INSERT inserts a new element into the heap.
3. MINIMUM returns the element with the minimum key in the heap.
4. EXTRACT-MIN removes and returns the element with the minimum key in the heap.
5. UNION merges two heaps into a single heap.

Template Parameters

T	The type of the elements stored in the heap. `T` must be movable, as it is moved into the heap in the insert method.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 MergeableHeap() [1/2]

```
template<typename T >
constexpr MergeableHeap< T >::MergeableHeap ( )  [constexpr], [default]
```

Constructs a new empty heap.

3.2.2.2 ~MergeableHeap()

```
template<typename T >
constexpr virtual MergeableHeap< T >::~MergeableHeap ( )  [constexpr], [virtual], [default]
```

Destroys the heap.

3.2.2.3 MergeableHeap() [2/2]

```
template<typename T >
constexpr MergeableHeap< T >::MergeableHeap (
            const MergeableHeap< T > & ) [constexpr], [delete]
```

MergeableHeap objects should not be copied or moved.

3.2.3 Member Function Documentation

3.2.3.1 extract_min()

```
template<typename T >
constexpr virtual std::optional<T> MergeableHeap< T >::extract_min ( ) [constexpr], [pure
virtual]
```

Removes and returns the minimum key in the heap.

Implemented in LazyBinomialHeap< T >.

3.2.3.2 insert()

```
template<typename T >
constexpr virtual void MergeableHeap< T >::insert (
            T key ) [constexpr], [pure virtual]
```

Inserts a key into the heap.

Implemented in LazyBinomialHeap< T >.

3.2.3.3 merge()

```
template<typename T >
constexpr virtual void MergeableHeap< T >::merge (
            MergeableHeap< T > & other ) [constexpr], [pure virtual]
```

Merges another heap into this heap.

Implemented in LazyBinomialHeap< T >.

3.2.3.4 minimum()

```
template<typename T >
constexpr virtual std::optional<std::reference_wrapper<const T> > MergeableHeap< T >::minimum
( ) const [constexpr], [pure virtual]
```

Returns the minimum key in the heap.

Implemented in LazyBinomialHeap< T >.

The documentation for this class was generated from the following file:

- mergeable_heap.h

Index