priority_queue

Generado por Doxygen 1.7.5

Lunes, 22 de Octubre de 2012 14:54:32

Índice general

1. Priority Queue **1**

1.1. Introduccion 1

1.2. Ejemplo de Uso: 1

1.3. Representaciones 2

1.4. Primera Representación: 2

1.4.1. Función de Abstracción : 2

1.4.2. Invariante de la Representación: 2

1.5. Segunda Representación: 2

1.5.1. Función de Abstracción : 2

1.5.2. Invariante de la Representación: 2

1.6. Tareas a Realizar. 3

2. Lista de tareas pendientes **5**

3. Índice de clases **7**

3.1. Lista de clases 7

4. Documentación de las clases **9**

4.1. Referencia de la Clase priority_queue 9

4.1.1. Descripción detallada 10

4.1.2. Documentación de los 'Typedef' miembros de la clase 10

4.1.2.1. size_type 10

4.1.3. Documentación del constructor y destructor 10

4.1.3.1. priority_queue 10

4.1.3.2. priority_queue 10

4.1.3.3. ∼priority_queue 11

4.1.4. Documentación de las funciones miembro 11
4.1.4.1. empty 11
4.1.4.2. operator= 11
4.1.4.3. pop 11
4.1.4.4. push 12
4.1.4.5. size 12
4.1.4.6. top 12

Capítulo 1

Priority Queue

Versión

v0

Autor

Estructuras de Datos

1.1. Introduccion

En esta practica se pretende avanzar en el uso de las estructuras de datos, para ello comenzaremos con un tipo de datos simplificado que llamaremos priority_queue.

Una priority_queue (cola_con_prioridad) es un contenedor que proporciona un subconjunto restringido de métodos (inserción de elementos además del borrado y consulta del elemento “mayor”). Se garantiza que el elemento “mayor” de la priority_queue se encuentra en el tope de la misma. Priority_queue no permite la iteración a través de sus elementos.

1.2. Ejemplo de Uso:

```
#include "priority_queue.h"

priority_queue<int> q1;


for (i=1;i<10;i++) q1.push(i);

for (i=20;i>11;i--) q1.push(i);

while (!q1.empty())  { // Imprime los números del 20 al 1.
     cout << q1.top() << " ";
     q1.pop();
   }
```

1.3. Representaciones

El alumno deberá realizar dos implementaciones distintas de una cola con prioridad, utilizando como base el TDA vector de la STL, en la primera de ellas los elementos se almacenarán sin tener en cuenta su valor mientras que en la segunda debemos garantizar que los elementos se encuentran ordenados en el vector

1.4. Primera Representación:

1.4.1. Función de Abstracción :

Funcion de Abstraccion: AF: Rep => Abs

dada qq=(vector<T> V, int mayor) ==> priority_queue PQ;

Un objeto abstracto, Q, se instancia en la cola con prioridad pq teniendo:

```
pq.V[mayor] = PQ.top()
```

1.4.2. Invariante de la Representación:

Propiedades que debe cumplir cualquier objeto

```
pq.size() == pq.V.size();

Para todo i, 0 <= i < pq.V.size() se cumple
pq.V[mayor] >= pq.V[i];
```

1.5. Segunda Representación:

En este caso, la representación que se utiliza es un vector ordenado de elementos

1.5.1. Función de Abstracción :

Funcion de Abstraccion: AF: Rep => Abs

dada qq=(vector<T> V) ==> priority_queue PQ;

Un objeto abstracto, Q, se instancia en la cola con prioridad pq teniendo:

```
pq.V[0] = PQ.top()
```

1.5.2. Invariante de la Representación:

Propiedades que debe cumplir cualquier objeto

```
pq.size() == pq.V.size();

Para todo i, 0 <= i < pq.V.size()-1 se cumple
pq.V[i] >= pq.V[i+1];
```

1.6. Tareas a Realizar.

Se pide implementar las dos representaciones del TDA priority_queue y analizar la eficiencia del siguiente código que permite ordenar un vector de elementos.

Para ello se puede utilizar el código que se ha proporcionado en la práctica 1.

```
// Ordena un vector de string en orden creciente
//@param[in,out] V vector a ordenar
//
void ordenar(vector<string> & V){

 priority_queue aux;
 int pos;

 for (int i=0;i<V.size(); i++)
   aux.push(V[i]);

 pos = V.size()-1;
 while (!aux.empty()) {
   V[pos]=aux.top();
   aux.pop();
   pos--;
 }
}
```

Capítulo 2

Lista de tareas pendientes

Miembro priority_queue::empty () const
implementa esta función

Miembro priority_queue::operator= (const priority_queue< T > &org)
implementa esta función

Miembro priority_queue::pop ()
implementa esta función

Miembro priority_queue::priority_queue ()
implementa esta función

Miembro priority_queue::priority_queue (const priority_queue< T > &org)
implementa esta función

Miembro priority_queue::push (const T &t)
implementa esta función

Miembro priority_queue::size () const
implementa esta función

Miembro priority_queue::top () const
implementa esta función

Miembro priority_queue::~priority_queue ()
implementa esta función

Capítulo 3

Índice de clases

3.1. Lista de clases

Lista de las clases, estructuras, uniones e interfaces con una breve descripción:

priority_queue 9

Capítulo 4

Documentación de las clases

4.1. Referencia de la Clase priority_queue

`#include <priority_queue.h>`

Tipos públicos

- typedef unsigned int size_type

Métodos públicos

- bool empty () const

  vacia Chequea si el priority_queue esta vacio (size()==0)
- priority_queue< T > & operator= (const priority_queue< T > &org)

  operador de asignación
- void pop ()

  sacar Saca el elemento al tope del priority_queue
- priority_queue ()

  Constructor primitivo.
- priority_queue (const priority_queue< T > &org)

  Constructor de copia.
- void push (const T &t)

  insertar Añade el elemento al priority_queue
- size_type size () const

  tamaño Devuelve el numero de elementos en el priority_queue
- const T & top () const

  tope de la cola Se garantiza que el elemento en el tope es el elemento más grande de la cola con prioridad, según determina el criterio de comparación.
- ~priority_queue ()

  Destructor. Destruye el receptor liberando los recursos que ocupaba.

4.1.1. Descripción detallada

prioirty_queue<T>

prioirty_queue<T>:: priority_queue, size, capacity, empty, push, pop, top, operator=

Descripción

Una priority_queue (cola_con_prioridad) es un contenedor que proporciona un subconjunto restringido de métodos (inserción de elementos además del borrado y consulta del elemento "mayor"). Se garantiza que el elemento "mayor" de la priority_queue se encuentra en el tope de la misma. Priority_queue no permite la iteración a través de sus elementos.

El número de elementos en el priority_queue puede variar dinámicamente; la gestión de la memoria es automática.

4.1.2. Documentación de los 'Typedef' miembros de la clase

4.1.2.1. typedef priority_queue::size_type

Hace referencia al tipo asociado al tamaño de la cola con prioridad

4.1.3. Documentación del constructor y destructor

4.1.3.1. priority_queue::priority_queue ( )

Constructor primitivo.

Funcion de Abstraccion: AF: Rep => Abs

dada qq=(vector<T> V, int mayor) ==> priority_queue PQ;

Un objeto abstracto, Q, se instancia en la cola con prioridad pq teniendo:

pq.V[mayor] = PQ.top()

Invariante de la Representación:

Propiedades que debe cumplir cualquier objeto priority_queue pq.size() == pq.V.size();

Para todo i, 0 <= i < pq.V.size() se cumple pq.V[mayor] >= pq.V[i];

Tareas pendientes implementa esta función

4.1.3.2. priority_queue::priority_queue (const priority_queue< T > & *org*)

Constructor de copia.

Parámetros

in	*org*	priority_queue que se copia

Crea un priority_queue duplicado exacto de org.

Tareas pendientes implementa esta función

4.1.3.3. priority_queue::~priority_queue ( )

Destructor. Destruye el receptor liberando los recursos que ocupaba.

Tareas pendientes implementa esta función

4.1.4. Documentación de las funciones miembro

4.1.4.1. bool priority_queue::empty ( ) const

vacia Chequea si el priority_queue esta vacio (size()==0)

Tareas pendientes implementa esta función

4.1.4.2. priority_queue< T > & priority_queue::operator= (const priority_queue< T > & *org*)

operador de asignación

Parámetros

in	*org*	priority_queue a copiar. Crea un priority_queue duplicado exacto de org.

Tareas pendientes implementa esta función

4.1.4.3. void priority_queue::pop ( )

sacar Saca el elemento al tope del priority_queue

Precondición

empty() == false;

Postcondición

el size() será decrementado en 1.

Tareas pendientes implementa esta función

4.1.4.4. void priority_queue::push (const T & *t*)

insertar Añade el elemento al priority_queue

Parámetros

in	*t*	elemento a insertar en la cola

Postcondición

el size() será incrementado en 1.

Tareas pendientes implementa esta función

4.1.4.5. priority_queue< T >::size_type priority_queue::size ( ) const

tamaño Devuelve el numero de elementos en el priority_queue

Tareas pendientes implementa esta función

4.1.4.6. const T & priority_queue::top ( ) const

tope de la cola Se garantiza que el elemento en el tope es el elemento más grande de la cola con prioridad, según determina el criterio de comparación.

Precondición

empty() == false;

Devuelve

Devuelve una referencia al elemento en el tope del priority_queue

Tareas pendientes implementa esta función

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- priority_queue.h
- priority_queue_v0.hxx