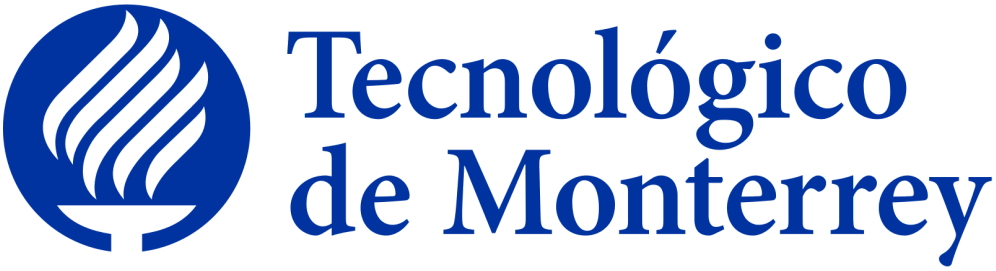
Instituto Tecnológico de Estudios Superiores de Monterrey



Campus Monterrey

Act 2.2 - Verificación de las funcionalidades de una estructura de datos lineal

Programación de estructuras de datos y algoritmos fundamentales (Gpo 610)

Eduardo López Benítez

Fidel Morales Briones A01198630

Monterrey, Nuevo León, a 2 de octubre de 2023

Casos de prueba:

Lista de prioridad:

```
296  int main() {
297      //Lista de prioridad
298      priorityQueue<int> *queue = new priorityQueue<int>();
299      queue->push(10,1);
300      queue->push(15,0);
301      queue->push(5,3);
302
303      cout << "Lista con prioridad\nLista con prioridad original:" << endl;
304      queue->printList();
305
306      cout << "\nSe actualiza cabeza y luego se borra la cabeza" << endl;
307      queue->updateHead(20);
308      queue->peek();
309      queue->pop();
310
311      cout << "\nLista modificada: " << endl;
312      queue->printList();
313
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Lista con prioridad
Lista con prioridad original:
Dato: 5 Prioridad: 3
Dato: 10 Prioridad: 1
Dato: 15 Prioridad: 0

Se actualiza cabeza y luego se borra la cabeza
Nodo en primera posicion:
Dato: 20 Prioridad: 3

Lista modificada:
Dato: 10 Prioridad: 1
Dato: 15 Prioridad: 0
```

Deque:

```
314     //Deque
315     deque<int> *deque1 = new deque<int>();
316     deque1->pushFront(4);
317     deque1->pushFront(9);
318     deque1->pushFront(12);
319
320     cout << "\nDeque\nDeque original: " << endl;
321     deque1->printDeque();
322
323     cout << "\nSe actualiza cabeza y luego se borra la cabeza" << endl;
324     deque1->updateTail(11);
325     deque1->peekTail();
326     deque1->popFront();
327
328     cout << "\nDeque modificada: " << endl;
329     deque1->printDeque();
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Deque
Deque original:
Dato: 12
Dato: 9
Dato: 4

Se actualiza cabeza y luego se borra la cabeza
Dato de la cola: 11

Deque modificada:
Dato: 9
Dato: 11
```

Conclusión:

Se crearon dos clases de estructuras de datos, una lista de prioridad y otra de un deque. Se implementaron con el uso de listas ligadas y con métodos CRUD básicos. También se realizaron casos de prueba para comprobar el funcionamiento adecuado de las estructuras.

Programa:

```
/*
Act 2.2 - Verificación de las funcionalidades de una estructura de
datos lineal
Fidel Morales Briones A01198630
2 de octubre de 2023
*/
#include <iostream>
using namespace std;

template <class T>
```

```
class Node {
    private:
        T data;
        Node<T> *next;
        Node<T> *previous;
        int priority = 0;

    public:
        Node(T data) {
            this->data = data;
            previous = nullptr;
            next = nullptr;
        }

        T getData() {
            return data;
        }

        void setData(T data) {
            this->data = data;
        }

        Node<T> * getNext() {
            return next;
        }

        void setNext(Node<T> *next) {
            this->next = next;
        }

        Node<T> * getPrevious() {
            return previous;
        }

        void setPrevious(Node<T> *previous) {
            this->previous = previous;
        }

        int getPriority() {
            return priority;
        }

        void setPriority(int priority) {
```

```
            this->priority = priority;
        }
};

template <class T>
class priorityQueue {
    private:
        Node<T> *head;
        Node<T> *tail;

    public:
        priorityQueue() {
            head = nullptr;
            tail = nullptr;
        }

        /**
         * Create, push(), agrega un nodo
         *
         * Se agrega un nodo empezando desde la tail de la fila, y se
va
         * revisando iterativamente si la prioridad del nuevo nodo es
menor o igual
         * al actual para instertarlo a su posición correspondiente
         *
         * Entrada: dato del nodo y su prioridad
         * Salida: no tiene
         * Complejidad tiempo: O(n)
         * Complejidad espacio: O(1)
         */
        void push(T data, int priority) {
            Node<T> *new_node = new Node<T>(data);
            new_node->setPriority(priority);

            if (head == nullptr) {
                head = new_node;
                tail = new_node;
            }
            else {
                Node<T> *current = tail;
                while (current != nullptr) {
                    if (current->getPriority() >=
new_node->getPriority()) {
```

```
                        if (current == tail) {
                            new_node->setPrevious(current);
                            current->setNext(new_node);
                            tail = new_node;
                            break;
                        } else {
                            new_node->setPrevious(current);
                            new_node->setNext(current->getNext());
                            current->getNext()->setPrevious(new_node);
                            current->setNext(new_node);
                            break;
                        }
                    }
                    else if (current->getPrevious() == nullptr) {
                        new_node->setNext(head);
                        head->setPrevious(new_node);
                        head = new_node;
                        break;
                    }
                    current = current->getPrevious();
                }
            }
        }

        /**
         * Read, peek(), imprimen los datos de la head de la lista
         *
         * Se imprime el dato del nodo head y su prioridad
         *
         * Entrada: no tiene
         * Salida: la impresión de datos en la terminal
         * Complejidad tiempo: O(1)
         * Complejidad espacio: O(1)
         */
        void peek() {
            cout << "Nodo en primera posicion:\n" <<"Dato: " <<
head->getData() << " Prioridad: " << head->getPriority() << endl;
        }

        /**
         * Read, printList(), imprimir lista
         *
         * Se imprimen los datos de todos los nodos y su prioridad
```

```
         * dentro la lista recorriendo toda la lista
         *
         * Entrada: no tiene
         * Salida: imprimir datos de todos los nodos
         * Complejidad tiempo: O(n)
         * Complejidad espacio: O(1)
         */
        void printList() {
            Node<T> *current = head;
            while (current != nullptr) {
                cout << "Dato: " << current->getData() << " Prioridad:
" << current->getPriority() << endl;
                current = current->getNext();
            }
        }

        /**
         * Update, updateHead(), actualizar dato de la head
         *
         * Se introduce el dato para actualizar la head
         *
         * Entrada: nuevo dato
         * Salida: no tiene
         * Complejidad tiempo: O(1)
         * Complejidad espacio: O(1)
         */
        void updateHead(T newData) {
            head->setData(newData);
        }

        /**
         * Delete, pop(), elimina el nodo en la primera posición de la
fila (head)
         *
         * Se elimina la head y se establece la siguiente nodo que
apunta
         * head como la nueva head de la lista
         *
         * Entrada: no tiene
         * Salida: no tiene
         * Complejidad tiempo: O(1)
         * Complejidad espacio: O(1)
         */
```

```
        void pop() {
            if (head == nullptr) {
                cout << "La lista está vacía" << endl;
            }
            else {
                Node<T> *current = head;
                head = head->getNext();
                head->setPrevious(nullptr);
                delete current;
            }
        }

};

template <class T>
class deque {
    private:
        Node<T> *head;
        Node<T> *tail;

    public:
        deque() {
            head = nullptr;
            tail = nullptr;
        }

        /**
         * Create, pushFront(), agregar un nuevo nodo desde la head
         *
         * Se agrega un nuevo nodo desde el inicio del deque, por la
head,
         *
         * Entrada: dato del nuevo nodo
         * Salida: no tiene
         * Complejidad tiempo: O(1)
         * Complejidad espacio: O(1)
         */
        void pushFront(T data) {
            Node<T> *new_node = new Node<T>(data);
            if (head == nullptr) {
                head = new_node;
                tail = new_node;
            }
```

```
        else {
            head->setPrevious(new_node);
            new_node->setNext(head);
            head = new_node;
        }
    }

    /**
     * Read, peekTail(), leer dato de la tail
     *
     * Se imprime en la terminal el dato de la cola
     *
     * Entrada: no tiene
     * Salida: no tiene
     * Complejidad tiempo: O(1)
     * Complejidad espacio: O(1)
     */
    void peekTail() {
        cout << "Dato de la cola: " << tail->getData() << endl;
    }

    /**
     * Read, printList(), imprimir lista
     *
     * Se imprimen los datos de todos los nodos
     * dentro la lista recorriendo toda la lista
     *
     * Entrada: no tiene
     * Salida: imprimir datos de todos los nodos
     * Complejidad tiempo: O(n)
     * Complejidad espacio: O(1)
     */
    void printDeque() {
        Node<T> *current = head;
        while (current != nullptr) {
            cout << "Dato: " << current->getData() << endl;
            current = current->getNext();
        }
    }

    /**
     * Update, updateTail(), actualizar dato de la tail
     *
```

```
         * Se introduce el dato para actualizar la tail
         *
         * Entrada: nuevo dato
         * Salida: no tiene
         * Complejidad tiempo: O(1)
         * Complejidad espacio: O(1)
         */
        void updateTail(T newData) {
            tail->setData(newData);
        }

        /**
         * Delete, popFront(), elimina el nodo en la primera posición
de la fila (head)
         *
         * Se elimina la head y se establece la siguiente nodo que
apunta
         * head como la nueva head de la lista
         *
         * Entrada: no tiene
         * Salida: no tiene
         * Complejidad tiempo: O(1)
         * Complejidad espacio: O(1)
         */
        void popFront() {
            if (head == nullptr) {
                cout << "La lista está vacía" << endl;
            }
            else {
                Node<T> *current = head;
                head = head->getNext();
                head->setPrevious(nullptr);
                delete current;
            }
        }

};

int main() {
    //Lista de prioridad
    priorityQueue<int> *queue = new priorityQueue<int>();
    queue->push(10,1);
    queue->push(15,0);
```

```
    queue->push(5,3);

    cout << "Lista con prioridad\nLista con prioridad original:" <<
endl;
    queue->printList();

    cout << "\nSe actualiza cabeza y luego se borra la cabeza" << endl;
    queue->updateHead(20);
    queue->peek();
    queue->pop();

    cout << "\nLista modificada: " << endl;
    queue->printList();

    //Deque
    deque<int> *deque1 = new deque<int>();
    deque1->pushFront(4);
    deque1->pushFront(9);
    deque1->pushFront(12);

    cout << "\nDeque\nDeque original: " << endl;
    deque1->printDeque();

    cout << "\nSe actualiza cabeza y luego se borra la cabeza" << endl;
    deque1->updateTail(11);
    deque1->peekTail();
    deque1->popFront();

    cout << "\nDeque modificada: " << endl;
    deque1->printDeque();

};
```

Bibliografía:

GeeksforGeeks. (2023). What is priority queue Introduction to priority queue.

GeeksforGeeks. https://www.geeksforgeeks.org/priority-queue-set-1-introduction/

Programiz. (s. f.). *Deque Data Structure*. programiz.com.

https://www.programiz.com/dsa/deque