자료구조론 HW2

김정선 교수님

소프트웨어융합대학 소프트웨어학과

2017012197 여채린

package hw2;

public class DLinkedList<T> {

    private Node<T> header;

    private Node<T> trailer;

    private int size = 0;

    public DLinkedList() {

    header = new Node<>(null, null, null);

    trailer = new Node<>(null, header,null);

    header.setNext(trailer);

    }

    public void setHeaderInfo(T info) {

        header.setItem(info);

    }

    public void setTrailerInfo(T info) {

        trailer.setItem(info);

    }

    public boolean isEmpty() {

        return size == 0;

    }

    public int getSize() { return size; }

    public Node<T> getFirst() {

        return header.getNext();

    }

    public Node<T> getLast() {

        return trailer.getPrev();

    }

    public void addFirst(Node<T> n) {

       addBetween(n.getItem(), header, header.getNext());

    }

    public void addLast(Node<T> n) {

        addBetween(n.getItem(), trailer.getPrev(), trailer);

    }

    public T removeFirst() {

        if (isEmpty()) return null;

        return remove(header.getNext());

    }

    public T removeLast() {

        if (isEmpty()) return null;

        return remove(trailer.getPrev());

    }

    public void addAfter(Node<T> p, Node<T> n) {

       n.setNext(p.getNext());

       n.setPrev(p);

       p.getNext().setPrev(n);

       p.setNext(n);

       size++;

    }

    public void addBefore(Node<T> p, Node<T> n) {

        n.setNext(p);

        n.setPrev(p.getPrev());

        p.getPrev().setNext(n);

        p.setPrev(n);

        size++;

    }

    //추가한 메소드!!

    private void addBetween(T e, Node<T> predecessor, Node<T> successor) {

    Node<T> newest = new Node<>(e, predecessor, successor);

    predecessor.setNext(newest);

    successor.setPrev(newest);

    size++;

    }

    public T remove(Node<T> n) {

        Node<T> predecessor = n.getPrev();

        Node<T> successor = n.getNext();

        predecessor.setNext(successor);

        successor.setPrev(predecessor);

        size--;

        return n.getItem();

    }

    @Override

    public String toString() {

        StringBuilder builder = new StringBuilder(

            "List: size = " + size + " [");

        Node<T> current = header.getNext();

        while (current != trailer) {

            builder.append(current.getItem().toString());

            current = current.getNext();

        }

        builder.append("]");

        return builder.toString();

    }

}

package hw2;

import hw2.DLinkedList;

/\* Block will be used as a type argument \*/

class Block {

    public int size;

    public int start;

    public int end;

    public Block(int size, int start, int end) {

        this.size = size;

        this.start = start;

        this.end = end;

    }

    @Override

    public String toString() {

        return "(" + size +", " + start + ", " + end + ")";

    }

}

public class MemoryManager {

    private DLinkedList<Block> heap = new DLinkedList<>();

    //추가

    public MemoryManager(int capacity) {

    Block freeblock1 = new Block(capacity, 0, capacity-1);

        heap.addFirst(new Node<Block>(freeblock1,null,null));

        if(heap.getFirst() == null) {

        heap.setTrailerInfo(freeblock1);

        }

    }

    public Block malloc(int size) {

    /\*

    if(freeblock.size < size) {

    throw new OutOfMemoryException("size lack");

    }

    Block newblock = new Block(size, freeblock.start, freeblock.start+size-1);

    freeblock = new Block(freeblock.size-size, freeblock.start+size, freeblock.end);

    return newblock;

    \*/

    Node<Block> x = heap.getFirst();

        Block curr = x.getItem();

        for(; x != null; x = x.getNext()){

            curr = x.getItem();

            if(size <= curr.size) break;

        }

        if(x == null){

            throw new OutOfMemoryException("size lack");

        }

        Block rBlock = new Block(size,curr.start,curr.start+size-1);//size를 size-1로 고침.\\

        Block nBlock = new Block(curr.size-size, curr.start+size, curr.end);

        Node<Block> nNode = new Node<Block>(nBlock,x.getPrev(),x.getNext());

        x.getPrev().setNext(nNode);

        x.getNext().setPrev(nNode);

        return rBlock;

        /\*\*/

    }

    public void free(Block block) {

    Node<Block> x = heap.getFirst();

        Block curr = x.getItem();

        for(; x != null; x = x.getNext()) {

            curr = x.getItem();

            if(block.end < curr.start){

                break;

            }

        }

        if(block.end+1 == curr.start){

            Block nBlock = new Block(curr.size+block.size, block.start, curr.end);

            Node<Block> nNode = new Node<Block>(nBlock,x.getPrev(),x.getNext());

            x.getPrev().setNext(nNode);

            x.getNext().setPrev(nNode);

        }else {

            Node<Block> nNode = new Node<Block>(block,x.getPrev(),x);

            x.getPrev().setNext(nNode);

            x.setPrev(nNode);

        }

        /\*\*/

    }

    // for debugging purpose only

    public DLinkedList<Block> getHeap() {

        return heap;

    }

    @Override

    public String toString() {

        return heap.toString();

    }

}