КПІ ім. Ігоря Сікорського

Інститут прикладного системного аналізу

Кафедра Системного проектування

Алгоритми та структури даних

**Лабораторна робота №2.1**

**«Дослідження структур даних зв’язний список та динамічний масив»**

Виконав:

Студент групи ДА-02

Рудік Андрій

Варіант №16

Мета роботи:

Ознайомитись і дослідити структури даних зв’язний список та динамічний масив. Поглибити розуміння роботи вказівників, познайомитись з використанням функцій всередині структур. Набути навичок реалізації однозв’язного списку та динамічного масиву мовою програмування C++, порівняти час роботи основних операцій цих структур даних та дослідити їх асимптотичну складність.

Варіант №16.

Завдання №1

Створити структуру для зберігання різнотипних даних відповідно до свого варіанту.

Структура Учень Хогвартсу має наступні поля: назва факультету, тип палички, вірогідність працевлаштування після навчання.

Лістинг

StudentOfHogwarts.h

#pragma once

class StudentOfHogwarts {

private:

char m\_facultyName;

char m\_wandType;

unsigned short m\_epmloymentLikelihood;

public:

StudentOfHogwarts();

StudentOfHogwarts(char facultyName, char wandType, unsigned short epmloymentLikelihood) :

m\_facultyName(facultyName), m\_wandType(wandType), m\_epmloymentLikelihood(epmloymentLikelihood) {}

friend std::ostream& operator<< (std::ostream&, const StudentOfHogwarts&);

};

StudentOfHogwarts.cpp

#include <string>

#include <iostream>

#include "StudentOfHogwarts.h"

#define NUMBER\_OF\_FACULTIES 4

const char FACULTIES[NUMBER\_OF\_FACULTIES] = {

'Г',

'П',

'К',

'С'

};

StudentOfHogwarts::StudentOfHogwarts() {

m\_facultyName = FACULTIES[rand() % NUMBER\_OF\_FACULTIES];

m\_wandType = 65 + (rand() % 26); // От A до Z

m\_epmloymentLikelihood = rand() % 101;

}

std::ostream& operator<< (std::ostream &out, const StudentOfHogwarts &el) {

out << "Факультет — " << el.m\_facultyName

<< ", тип палочки — " << el.m\_wandType

<< ", вероятность трудоустройства — " << el.m\_epmloymentLikelihood << "%";

return out;

}

Завдання №2

Реалізувати динамічний масив (з саморозширюванням):

2.1 Створити структуру DynamicArray для зберігання структур свого варіанту та реалізувати в ній основні функції динамічного масиву:

* push\_back() додавання елементу в кінець
* pop\_back() зчитування та видалення елементу з кінця
* get() зчитування n-го елементу
* size() знаходження кількості елементів
* print() виведення всіх елементів з даними, що зберігаються

2.2 Реалізувати додаткові функції для динамічного масиву:

* push\_front() додавання елементу в початок
* pop\_front() зчитування та видалення елементу з початку

Лістинг

DynamicArray.h

#pragma once

#include <iostream>

#define ALPHA 2

template <class T>

class DynamicArray {

private:

T\* m\_array;

size\_t m\_capacity;

size\_t m\_length;

public:

DynamicArray() : m\_length(0), m\_capacity(1), m\_array(new T[1]) {}

~DynamicArray();

T& operator[] (int);

void push(T);

T pop();

void unshift(T);

T shift();

size\_t getLength();

T getElem(size\_t);

void print();

};

template <class T>

DynamicArray<T>::~DynamicArray() {

delete[] m\_array;

}

template <class T>

T& DynamicArray<T>::operator[] (int idx) {

if (m\_length <= (idx >= 0 ? idx : -(idx + 1))) throw "Index out of range of array";

return idx >= 0 ? m\_array[idx] : m\_array[m\_length + idx];

}

template <class T>

void DynamicArray<T>::push(T el) {

if (m\_capacity <= m\_length) {

T\* tmp = new T[m\_capacity \*= ALPHA];

for (size\_t i = 0; i < m\_length; i++) {

tmp[i] = m\_array[i];

}

delete[] m\_array;

m\_array = tmp;

}

m\_array[m\_length++] = el;

}

template <class T>

T DynamicArray<T>::pop() {

if (!m\_length) throw "Array is empty";

return m\_array[--m\_length];

}

template <class T>

void DynamicArray<T>::unshift(T el) {

if (m\_capacity <= m\_length) {

T\* tmp = new T[m\_capacity \*= ALPHA];

for (size\_t i = 1; i <= m\_length; i++) {

tmp[i] = m\_array[i - 1];

}

delete[] m\_array;

m\_array = tmp;

}

else {

for (size\_t i = m\_length; i; ) {

m\_array[i] = m\_array[--i];

}

}

m\_array[0] = el;

m\_length++;

}

template <class T>

T DynamicArray<T>::shift() {

if (!m\_length) throw "Array is empty";

T tmp = m\_array[0];

for (size\_t i = 0; i < m\_length;) {

m\_array[i] = m\_array[++i];

}

m\_length--;

return tmp;

}

template <class T>

size\_t DynamicArray<T>::getLength() {

return m\_length;

}

template <class T>

T DynamicArray<T>::getElem(size\_t idx) {

return m\_array[idx];

}

template <class T>

void DynamicArray<T>::print() {

if (!m\_length) {

std::cout << "Динамический массив пуст" << std::endl;

return;

}

for (size\_t i = 0; i < m\_length; i++) {

std::cout << i + 1 << ") " << m\_array[i] << std::endl;

}

}

Завдання №3

Реалізувати однозв’язний список:

3.1 Створити структуру Node для базового елементу однозв’язного

списку, вона буде містити в собі структуру відповідно до варіанту.

3.2 Створити структуру LinkedList та реалізувати в ній основні функції

однозв’язного списку:

* push\_front() додавання елементу в початок
* push\_back() додавання елементу в кінець
* get() зчитування n-го елементу
* pop\_front() зчитування та видалення елементу з початку
* pop\_back() зчитування та видалення елементу з кінця
* size() знаходження кількості елементів
* print() виведення всіх елементів з даними, що зберігаються

Лістинг

LinkedList.h

#pragma once

#include <iostream>

template <class T>

class LinkedList {

private:

struct Node {

T value;

Node\* next = NULL;

Node\* prev = NULL;

Node(T v) : value(v), next(NULL), prev(NULL) {}

};

Node\* getNode(int);

Node\* m\_head;

Node\* m\_tail;

size\_t m\_length;

public:

LinkedList() : m\_length(0), m\_head(NULL), m\_tail(NULL) {}

~LinkedList();

T& operator[] (int);

void pushBack(T);

T popBack();

void pushFront(T);

T popFront();

void insert(T, size\_t);

T pop(size\_t);

T getElem(int);

size\_t getLength();

void print();

};

template <class T>

LinkedList<T>::~LinkedList() {

for (Node\* current = m\_head; current != NULL; ) {

Node\* tmp = current;

current = current->next;

delete tmp;

}

}

template <class T>

void LinkedList<T>::pushBack(T value) {

m\_length++;

if (m\_tail == NULL) {

m\_tail = new Node(value);

m\_head = m\_tail;

return;

}

m\_tail->next = new Node(value);

m\_tail->next->prev = m\_tail;

m\_tail = m\_tail->next;

}

template <class T>

T LinkedList<T>::popBack() {

if (m\_tail == NULL) throw "List is empty";

m\_length--;

Node\* tmp = m\_tail;

T value = tmp->value;

m\_tail = m\_tail->prev;

if (m\_tail != NULL) {

m\_tail->next = NULL;

} else {

m\_head = m\_tail;

}

delete tmp;

return value;

}

template <class T>

void LinkedList<T>::pushFront(T value) {

m\_length++;

if (m\_head == NULL) {

m\_head = new Node(value);

m\_tail = m\_head;

return;

}

m\_head->prev = new Node(value);

m\_head->prev->next = m\_head;

m\_head = m\_head->prev;

}

template <class T>

T LinkedList<T>::popFront() {

if (m\_head == NULL) throw "List is empty";

m\_length--;

Node\* tmp = m\_head;

T value = tmp->value;

m\_head = m\_head->next;

if (m\_head != NULL) {

m\_head->prev = NULL;

}

else {

m\_tail = NULL;

}

delete tmp;

return value;

}

template <class T>

void LinkedList<T>::insert(T value, size\_t idx) {

if (idx > m\_length) throw "Index is bigger than a size of list";

m\_length++;

Node\* prev = NULL;

Node\* next = m\_head;

for (size\_t i = 0; i < idx; i++) {

prev = next;

next = next->next;

}

Node\* tmp = new Node(value);

if (prev != NULL) {

prev->next = tmp;

tmp->prev = prev;

}

else {

m\_head = tmp;

}

if (next != NULL) {

next->prev = tmp;

tmp->next = next;

}

else {

m\_tail = tmp;

}

}

template <class T>

T LinkedList<T>::pop(size\_t idx) {

Node\* tmp = getNode(idx);

T value = tmp->value;

m\_length--;

if (tmp->prev) {

tmp->prev->next = tmp->next;

}

else {

m\_head = tmp->next;

}

if (tmp->next) {

tmp->next->prev = tmp->prev;

}

else {

m\_tail = tmp->prev;

}

delete tmp;

return value;

}

template <class T>

typename LinkedList<T>::Node\* LinkedList<T>::getNode(int idx) {

idx = idx >= 0 ? idx : m\_length + idx;

if (m\_length <= idx || idx < 0) throw "Index out of range of list";

Node\* current;

if (idx <= m\_length / 2) {

current = m\_head;

for (int i = 0; i < idx; i++) {

current = current->next;

}

}

else {

current = m\_tail;

for (int i = m\_length - 1; i > idx; i--) {

current = current->prev;

}

}

return current;

}

template <class T>

T& LinkedList<T>::operator[] (int idx) {

return getNode(idx)->value;

}

template <class T>

T LinkedList<T>::getElem(int idx) {

return getNode(idx)->value;

}

template <class T>

size\_t LinkedList<T>::getLength() {

return m\_length;

}

template <class T>

void LinkedList<T>::print() {

if (!m\_length) {

std::cout << "Список пуст" << std::endl;

return;

}

size\_t counter = 1;

for (Node\* current = m\_head; current != NULL; current = current->next, counter++) {

std::cout << counter << ") " << current->value << std::endl;

}

}

Завдання №5

Провести порівняння роботи однозв’язного списку та динамічного масиву (кожен крок виконати для обох структур, заміряти та порівняти час кожного кроку та всіх кроків разом):

5.1 Додати в кінець 50000 елементів

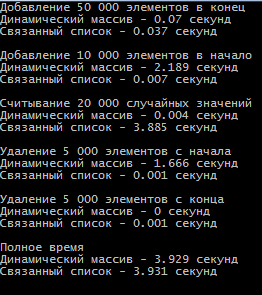
5.2 Додати в початок 10000 елементів

5.3 Зчитати 20000 елементів під випадковими індексами

5.4 Видалити 5000 елементів з початку

5.5 Видалити 5000 елементів з кінця

Результат



Лістинг

main.cpp

#include <iostream>

#include <clocale>

#include <cstdlib>

#include <ctime>

#include "StudentOfHogwarts.h"

#include "DynamicArray.h"

#include "LinkedList.h"

using namespace std;

int main()

{

setlocale(LC\_ALL, "rus");

srand(time(NULL));

DynamicArray<StudentOfHogwarts> arr;

LinkedList<StudentOfHogwarts> lst;

float totalDynamicArrayTime = 0;

float totalLinkedListTime = 0;

// Добавление 50 000 элементов в конец

clock\_t start = clock();

for (int i = 0; i < 5e4; i++) {

arr.push(StudentOfHogwarts());

}

float testDynamicArrayTime = (float(clock() - start)) / CLOCKS\_PER\_SEC;

totalDynamicArrayTime += testDynamicArrayTime;

start = clock();

for (int i = 0; i < 5e4; i++) {

lst.pushBack(StudentOfHogwarts());

}

float testLinkedListTime = (float(clock() - start)) / CLOCKS\_PER\_SEC;

totalLinkedListTime += testLinkedListTime;

cout << "Добавление 50 000 элементов в конец" << endl

<< "Динамический массив — " << testDynamicArrayTime << " секунд" << endl

<< "Связанный список — " << testLinkedListTime << " секунд" << endl << endl;

// Добавление 10 000 элементов в начало

start = clock();

for (int i = 0; i < 1e4; i++) {

arr.unshift(StudentOfHogwarts());

}

testDynamicArrayTime = (float(clock() - start)) / CLOCKS\_PER\_SEC;

totalDynamicArrayTime += testDynamicArrayTime;

start = clock();

for (int i = 0; i < 1e4; i++) {

lst.pushFront(StudentOfHogwarts());

}

testLinkedListTime = (float(clock() - start)) / CLOCKS\_PER\_SEC;

totalLinkedListTime += testLinkedListTime;

cout << "Добавление 10 000 элементов в начало" << endl

<< "Динамический массив — " << testDynamicArrayTime << " секунд" << endl

<< "Связанный список — " << testLinkedListTime << " секунд" << endl << endl;

// Считывание 20 000 случайных значений

start = clock();

for (int i = 0; i < 2e4; i++) {

arr.getElem(rand() % arr.getLength());

}

testDynamicArrayTime = (float(clock() - start)) / CLOCKS\_PER\_SEC;

totalDynamicArrayTime += testDynamicArrayTime;

start = clock();

for (int i = 0; i < 2e4; i++) {

lst.getElem(rand() % arr.getLength());

}

testLinkedListTime = (float(clock() - start)) / CLOCKS\_PER\_SEC;

totalLinkedListTime += testLinkedListTime;

cout << "Считывание 20 000 случайных значений" << endl

<< "Динамический массив — " << testDynamicArrayTime << " секунд" << endl

<< "Связанный список — " << testLinkedListTime << " секунд" << endl << endl;

// Удаление 5 000 элементов с начала

start = clock();

for (int i = 0; i < 5e3; i++) {

arr.shift();

}

testDynamicArrayTime = (float(clock() - start)) / CLOCKS\_PER\_SEC;

totalDynamicArrayTime += testDynamicArrayTime;

start = clock();

for (int i = 0; i < 5e3; i++) {

lst.popFront();

}

testLinkedListTime = (float(clock() - start)) / CLOCKS\_PER\_SEC;

totalLinkedListTime += testLinkedListTime;

cout << "Удаление 5 000 элементов с начала" << endl

<< "Динамический массив — " << testDynamicArrayTime << " секунд" << endl

<< "Связанный список — " << testLinkedListTime << " секунд" << endl << endl;

// Удаление 5 000 элементов с конца

start = clock();

for (int i = 0; i < 5e3; i++) {

arr.pop();

}

testDynamicArrayTime = (float(clock() - start)) / CLOCKS\_PER\_SEC;

totalDynamicArrayTime += testDynamicArrayTime;

start = clock();

for (int i = 0; i < 5e3; i++) {

lst.popBack();

}

testLinkedListTime = (float(clock() - start)) / CLOCKS\_PER\_SEC;

totalLinkedListTime += testLinkedListTime;

cout << "Удаление 5 000 элементов с конца" << endl

<< "Динамический массив — " << testDynamicArrayTime << " секунд" << endl

<< "Связанный список — " << testLinkedListTime << " секунд" << endl << endl;

cout << "Полное время" << endl

<< "Динамический массив — " << totalDynamicArrayTime << " секунд" << endl

<< "Связанный список — " << totalLinkedListTime << " секунд" << endl << endl;

return 0;

}

Висновок

Я ознайомився і дослідив структури даних зв’язний список та динамічний масив. Поглибив розуміння роботи вказівників, познайомився з використанням функцій всередині структур. Набув навичок реалізації однозв’язного списку та динамічного масиву мовою програмування C++, порівняв час роботи основних операцій цих структур даних та дослідив їх асимптотичну складність.