

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**Український державний університет науки і технологій**

Кафедра «Комп’ютерні інформаційні технології»

**Лабораторна робота №4**

**з дисципліни «якість програмного забезпечення та тестування»**

**на тему: «Проектування структур даних з урахуванням програмно-апаратних середовищ їх використання»**

Виконала: ???

Прийняв: ???

Дніпро, 2022

**Тема**. Проектування структур даних з урахуванням програмно-апаратних середовищ їх використання.

**Мета роботи**. Вивчити статистичні методи визначення показників часової ефективності алгоритмів та набути практичних навичок застосування цих методів при вирішенні задач розробки структур даних.

**Постановка задачі**

На основі результатів лабораторної роботи №3, вибрати дві структури даних для зберігання лексем програми.

Для тексту програми на заданій мові програмування (за матеріалами лабораторної роботи № 3) виконати пошук всіх лексем класу, визначеного варіантом. Знайдені лексеми занести в структури даних.

Виконати операції обробки даних з виміром часу. Для цього заповнити структури, розроблені в л.р. 3, лексемами в кількості а шт, де а – випадкове число, яке не перевищує загальну кількість знайдених лексем заданого класу в програмі. Провести серії випробувань за такими сценаріями:

1) додати (а + 1)-елемент в структури, виконавши замір часу виконання операції;

2) видалити елемент, який знаходиться в структурі, порядковий номер якого – випадкове число в діапазоні від 0 до а, виконавши замір часу виконання операції;

3) знайти елемент, який знаходиться в структурі, порядковий номер якого – випадкове число в діапазоні від 0 до а, виконавши замір часу виконання операції.

На основі отриманих даних виконати оцінювання часової ефективності алгоритмів.

Обрані структури даних та їх обчислювальна складність (на основі л.р.3):

|  |  |  |  |
| --- | --- | --- | --- |
| Структура даних | Обчислювальна складність | | |
| Найкращий  випадок | Середній  випадок | Найгірший  випадок |
| Двозв’язний впорядкований список | 6,68 | 15,42 | 25,26 |
| Черга зі зв’язним представленням | 4,28 | 13,92 | 22,56 |
| Стек зі зв’язним представленням | 4,72 | 12,34 | 18,82 |

**Текст програми**

Файл List.h:

#pragma once

#include<string>

class List

{

private:

struct Node

{

std::string field; // поле данных

Node\* next=nullptr; // указатель на следующий элемент

Node\* prev=nullptr; // указатель на предыдущий элемент

};

Node\* head = nullptr;

void add(std::string value, Node\* node);

public:

List() {};

~List() {};

void add(std::string value);

Node\* find\_node(std::string value);

Node\* search(std::string value);

void del(std::string value);

void show\_list();

};

Файл List.cpp:

#include "List.h"

#include <iostream>

List::Node\* List::search(std::string value)

{

Node\* iterator = head;

while (iterator != nullptr)

{

if (iterator->field == value)

break;

iterator = iterator->next;

}

return iterator;

}

List::Node\* List::find\_node(std::string value)

{

Node\* iterator = head;

if (iterator==nullptr || iterator->field > value)

return nullptr;

while (iterator->next != nullptr)

{

if (iterator->next->field > value)

break;

iterator = iterator->next;

}

return iterator;

}

void List::add(std::string value, Node\* node=nullptr)

{

Node\* tmp = new Node;

tmp->field = value;

if (node == nullptr || head==nullptr)

{

tmp->next = head;

if (tmp->next != nullptr)

tmp->next->prev = tmp;

head = tmp;

return;

}

tmp->next = node->next;

tmp->prev = node;

node->next = tmp;

}

void List::add(std::string value)

{

add(value, find\_node(value));

}

void List::show\_list()

{

Node\* info;

info = head;

while (info)

{

std::cout << info->field << " ";

info = info->next;

}

std::cout << "\n";

}

void List::del(std::string value)

{

Node\* tmp = search(value);

if (tmp == nullptr)

return;

if (tmp == head)

head = head->next;

if(tmp->next!=nullptr)

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

if (tmp->prev != nullptr)

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

delete tmp;

}

Файл Queue.h:

#pragma once

#include <iostream>

#include <string>

struct Node

{

std::string data;

Node\* next = nullptr;

};

class Queue

{

private:

Node\* Head = nullptr;

Node\* Tail = nullptr;

public:

Queue() {};

~Queue();

void push(std::string val);

std::string pop();

Node\* find(std::string val);

void remove(std::string a);

void show()

{

Node\* temp = Head; //Временный указатель на начало списка

while (temp != nullptr) //Пока в списке что-то встречается

{

std::cout << temp->data << " "; //Выводим значения из списка на экран

temp = temp->next; //Сдвигаем указатель на начало на адрес следующего элемента

}

std::cout << std::endl;

}

};

Файл Queue.cpp:

#include "Queue.h"

Queue::~Queue()

{

Node\* temp = Head;

while (temp != nullptr)

{

temp = Head->next;

delete Head;

Head = temp;

}

}

void Queue::push(std::string val)//добавление

{

Node\* temp = new Node;

temp->data = val;

if (Head != nullptr)

{

Tail->next = temp;

Tail = temp;

}

else

{

Head = Tail = temp;

}

}

std::string Queue::pop()//удаление

{

std::string data = "";

if (Head != nullptr)

{

Node\* temp = Head;

data = Head->data;

Head = Head->next;

delete temp;

}

else

{

throw "Try to pop from an empty queue";

}

return data;

}

void Queue::remove(std::string a)//удаление

{

Node\* temp = Head;

if (Head->data == a)

{

pop();

return;

}

while (temp != nullptr && temp->next!=nullptr)

{

if (temp->next->data == a)

{

Node\* temp\_next = temp->next;

temp->next = temp\_next->next;

delete temp\_next;

return;

}

temp = temp->next;

}

}

Node\* Queue::find(std::string val)//поиск

{

Node\* temp = Head;

while (temp != nullptr)

{

if (temp->data == val)

return temp;

else

temp = temp->next;

}

return nullptr;

}

Файл stack.h:

#pragma once

#include<string>

class Stack

{

private:

struct Node

{

std::string item;

Node \* next;

};

Node\* top=nullptr; // указатель на вершину стека

public:

Stack() {};

~Stack() {};

void add(std::string value);

bool search(std::string value);

void del();

void show\_stack();

};

Файл stack.cpp:

#include"stack.h"

#include<iostream>

#define stas std

void Stack::add(std::string value)

{

Node\* p=new Node;

p->item = value;

p->next = top; // p указывает на 1-й элемент

top = p;

}

bool Stack::search(std::string value)

{

Node\* iterator = top;

while (iterator != nullptr)

{

if (iterator->item == value)

break;

iterator = iterator->next;

}

return iterator!=nullptr;

}

void Stack::del()

{

if (top == nullptr)

return;

Node\* p2; // дополнительный указатель

std::string item;

item = top->item;

p2 = top;

top = top->next;

// Освободить память, выделенную под 1-й элемент

delete p2;

}

void Stack::show\_stack()

{

if (top == nullptr)

std::cout << "stack is empty." << std::endl;

else

{

Node\* p; // дополнительный указатель

p = top;

while (p != nullptr)

{

stas::cout << p->item << " ";

p = p->next;

}

std::cout << stas::endl;

}

}

Файл Source.cpp:

#include<iostream>

#include"List.h"

#include"stack.h"

#include"Queue.h"

#include <chrono>

#include <ctime>

#include <vector>

int main()

{

std::vector <std::string> words = { "Getmanoid", "a", "i", "j", "m", "n", "s", "st" };

std::vector <double> add;

std::vector <double> del;

std::vector <double> search;

int a, b, c;

for (int i = 0; i < 100; i++)

{

Queue q1;

srand(time(NULL));

a = rand() % (words.size() - 2) + 1;

for (int i = 0; i < a; i++)

{

q1.push(words[i]);

}

auto start = std::chrono::high\_resolution\_clock::now();

q1.push(words[a + 1]);

auto end = std::chrono::high\_resolution\_clock::now();

std::chrono::duration<double, std::micro> fp\_ms = end - start;

add.push\_back(fp\_ms.count());

c = rand() % (a);

start = std::chrono::high\_resolution\_clock::now();

q1.remove(words[c]);

end = std::chrono::high\_resolution\_clock::now();

fp\_ms = end - start;

del.push\_back(fp\_ms.count());

b = rand() % (a);

start = std::chrono::high\_resolution\_clock::now();

q1.find(words[b]);

end = std::chrono::high\_resolution\_clock::now();

fp\_ms = end - start;

search.push\_back(fp\_ms.count());

}

std::vector <double> add1;

std::vector <double> del1;

std::vector <double> search1;

for (int i = 0; i < 100; i++)

{

Stack s1;

for (int i = 0; i < a; i++)

{

s1.add(words[i]);

}

auto start = std::chrono::high\_resolution\_clock::now();

s1.add(words[a + 1]);

auto end = std::chrono::high\_resolution\_clock::now();

std::chrono::duration<double, std::micro> fp\_ms = end - start;

add1.push\_back(fp\_ms.count());

start = std::chrono::high\_resolution\_clock::now();

s1.del();

end = std::chrono::high\_resolution\_clock::now();

fp\_ms = end - start;

del1.push\_back(fp\_ms.count());

start = std::chrono::high\_resolution\_clock::now();

s1.search(words[b]);

end = std::chrono::high\_resolution\_clock::now();

fp\_ms = end - start;

search1.push\_back(fp\_ms.count());

}

double S\_add = 0;

double R\_add = 0;

double S\_del = 0;

double R\_del = 0;

double S\_search = 0;

double R\_search = 0;

for (int i = 0; i < 100; i++)

{

S\_add += (add1[i] - add[i]) / std::max(add1[i], add[i]);

S\_del += (del1[i] - del[i]) / std::max(del1[i], del[i]);

S\_search += (search1[i] - search[i]) / std::max(search1[i], search[i]);

R\_add += ((add1[i] - add[i]) > 0) ? 1 : 0;

R\_del += ((del1[i] - del[i]) > 0) ? 1 : 0;

R\_search += ((search1[i] - search[i]) > 0) ? 1 : 0;

}

std::cout << "S add = " << S\_add << std::endl;

std::cout << "R add = " << R\_add << std::endl;

std::cout << "S del = " << S\_del << std::endl;

std::cout << "R del = " << R\_del << std::endl;

std::cout << "S search = " << S\_search << std::endl;

std::cout << "R search = " << R\_search << std::endl;

std::cout << std::endl;

return 0;

}

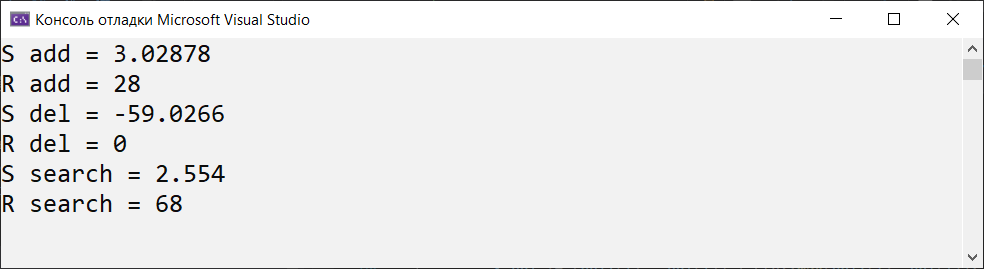
**Результати комп’ютерних експериментів:**

**Обрані структури:**

1) Черга зі зв’язним представленням

2) Стек зі зв’язним представленням

**Визначені показники часової ефективності:**



**Результати розрахунку показників обчислювальної складності**

|  |  |  |
| --- | --- | --- |
| Операція | Показники обчислювальної складності | |
| Оцінка S | Оцінка R |
| Додавання до структури даних | 3.02878% | 28% |
| Пошук у структурі даних | -59.0266% | 0% |
| Видалення зі структури даних | 2.554% | 68% |

**Висновок**

Оцінка S- демонструє перевагу стеку над чергою при пошуку елементу. При додаванні та видалення елементу оцінка S- демонструє перевагу черги над стеком. Оцінка R- демонструє співвідношення областей переваги, а саме вказує на перевагу стеку над чергою при пошуку та додаванні елементу. При видаленні елементу оцінка R- демонструє перевагу черги над стеком.