|  |
| --- |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **"МИРЭА – Российский технологический университет"**  **РТУ МИРЭА** |

Институт кибербезопасности и цифровых технологий

Кафедра КБ-2 «Институт кибербезопасности и цифровых технологий»

**ОТЧЕТ   
о выполнении домашней работе**

**«Реализация сортировки линейных структур данных»**

**по дисциплине   
«Технологии и методы программирования»**

**Вариант № 78**

Выполнил: студент 2 курса

группы БИСО-01-21

Русаков А.М.

шифр ИТ6-4387086

Москва 2022 г.

**Задание на № 1.**

В рамках домашней работы №1 требуется программно реализовать (с помощью массива (однонаправленных/двунаправленный динамический линейный связанный список, массива или используя стандартный контейнер библиотеки STL “stack” или «queue» - по варианту) абстрактный тип данных (АТД) в соответствии с заданием (стек, дек, очередь с одной головой, очередь с головой и хвостом).

Абстрактный тип данных должен позволять осуществлять только операции, присущие типу линейного связанного списка:

* получить значение первого элемента (на выходе),
* добавить элемент (на вход),
* удалить элемент из списка (на выходе),
* проверить – список пуст,
* обнулить (проинициализировать) список (конструктор, при необходимости).
* деструктор (при необходимости)

Используя разработанный АТД и указанный набор операций, необходимо реализовать заданный алгоритм сортировки последовательности элементов заданного типа, при этом следует учитывать, что разрешен доступ (чтение/извлечение) только к элементу на выходе.

На основе исходного текста программы получить аналитическую оценку трудоемкости работы алгоритма сортировки, используя О-символику для каждого реализованного метода АТД и сортировки в целом.

**Вариант № 78.**

**Реализация связи элементов линейного списка: Массив**

**Способ организации линейного связанный список: Стек**

**Алгоритм сортировки: Пузырьковая сортировка**

**Пузырьковая сортировка**

**Обзор быстрой сортировки**

Пузырьковая сортировка – самый простой для реализации и понимания, но эффективный лишь на небольших массивах данных алгоритм, т.к. его сложность: О(n2).

**Листинг программы с расчетами.**

#include <iostream>

#include <random>

#include <chrono>

#include "Adapter.h"

using namespace std::chrono;

class Timer

{

typedef high\_resolution\_clock Clock;

private:

Clock::time\_point start;

public:

Timer()

:start(Clock::now()) {}

long long get\_time\_ms() const

{

return duration\_cast<milliseconds>(Clock::now() - start).count();

}

};

void bubble\_sort(Adapter& adapter, uint64\_t& nop) // F(n) = 56n^3 + 18n^2

{

nop += 1;

for (int i = 0; i < adapter.size() - 1; i++) // n

{

nop += 1;

nop += 2;

for (int j = 0; j < adapter.size() - i - 1; j++) // n

{

nop += 4;

if (adapter.peekElement(j) > adapter.peekElement(j + 1)) // 20n + 8

{

int tmp = adapter.peekElement(j + 1);

adapter.setElement(j, tmp);

adapter.getElement(j + 2);

}

}

}

}

void work(int size)

{

std::mt19937 rng{ std::random\_device()() };

std::uniform\_int\_distribution<int> dist{ -20, 20 };

uint64\_t nop = 0;

Adapter a(nop, size + 10);

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

{

a.push(dist(rng));

}

const Timer t;

bubble\_sort(a, nop);

const auto time = t.get\_time\_ms();

std::cout << "N=" << size << "; Elapsed time: " << time << "ms = " << time / 1000.0f << "s; nop=" << nop <<"\n-------\n";

}

int main()

{

for (int i = 0; i < 10; i++)

{

work( (i+1) \* 300);

}

}

F(n) =

O(F(n)) =

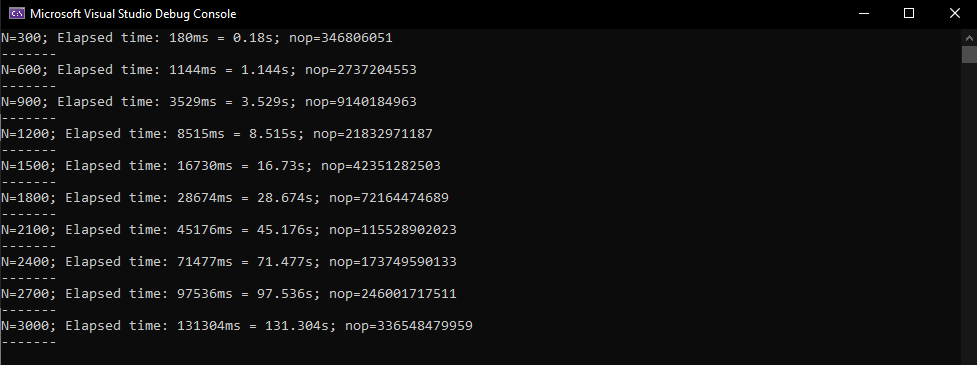
**Таблица результата экспериментов и графики зависимостей**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во элементов | F(n) | O(F(n)) | Т(n) (сек) | N\_op |
| 300 | 1513620000 | 27000000 | 0,18 | 346806051 |
| 600 | 12102480000 | 216000000 | 1,144 | 2737204553 |
| 900 | 40838580000 | 729000000 | 3,529 | 9140184963 |
| 1200 | 96793920000 | 1728000000 | 8,515 | 21832971187 |
| 1500 | 1,89041E+11 | 3375000000 | 16,731 | 42351282503 |
| 1800 | 3,2665E+11 | 5832000000 | 28,674 | 72164474689 |
| 2100 | 5,18695E+11 | 9261000000 | 45,176 | 1,15529E+11 |
| 2400 | 7,74248E+11 | 13824000000 | 71,477 | 1,7375E+11 |
| 2700 | 1,10238E+12 | 19683000000 | 97,536 | 2,46002E+11 |
| 3000 | 1,51216E+12 | 27000000000 | 131,304 | 3,36548E+11 |

**Коэффициенты**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| С1=F(n)/T(n) | С2=O(F(n))/T(n) | С3=F(n)/N\_op | С4=O(F(n))/N\_op | C5=N\_op/T(n) |
| 8409000000 | 150000000 | 4,364456721 | 0,077853313 | 1926700283 |
| 10579090909 | 188811188,8 | 4,421474452 | 0,078912626 | 2392661323 |
| 11572281099 | 206574100,3 | 4,468025556 | 0,079757686 | 2590021242 |
| 11367459777 | 202935995,3 | 4,433382849 | 0,079146351 | 2564060034 |
| 11298816568 | 201721355,6 | 4,46363106 | 0,079690621 | 2531306109 |
| 11391864407 | 203389830,5 | 4,526469865 | 0,080815388 | 2516721584 |
| 11481657960 | 204998229,1 | 4,489745604 | 0,080161759 | 2557307022 |
| 10832123340 | 193404871,5 | 4,456112267 | 0,079562778 | 2430846148 |
| 11302280389 | 201802411,4 | 4,481185055 | 0,080011637 | 2522163278 |
| 11516496070 | 205629683,8 | 4,493147615 | 0,080226183 | 2563124352 |

**Скриншот работы программы:**



**Выводы.**

По результатам экспериментов было установлено, что графики C1, C2, C3, C4, C5 от N имеют не линейную зависимость от количества элементов.

**Литература:**

1. Ахо А. В. Структуры данных и алгоритмы. – Издательский дом Вильямс, 2000.

2. Кнут Д. Э. Искусство программирования: Сортировка и поиск. – Издательский дом Вильямс, 2000. – Т. 3.

**Приложение 1. Применение счетчика операций N\_op.**

Stack.h

#pragma once

#include <iostream>

class Stack

{

int\* data;

int capacity;

int top;

uint64\_t& nop;

public:

Stack(uint64\_t& nop, int size = 5000)

:capacity(size), top(-1), nop(nop)

{

data = new int[capacity];

}

Stack (const Stack& other)

:capacity(other.capacity), top(other.top), nop(other.nop)

{

data = new int[other.capacity];

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

{

data[i] = other.data[i];

}

}

~Stack()

{

delete[] data;

}

//3 + 6n

void display() const

{

std::cout << "Stack (top -> bottom): { ";

nop += 3;

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

{

nop += 6;

std::cout << data[top - i] << ", ";

}

std::cout << data[0] << " }\n";

}

// 5

void push(int element)

{

nop += 3;

top++;

data[top] = element;

}

// 3

int pop()

{

nop += 3;

const int element = data[top];

top--;

return element;

}

// 1

int peek() const

{

nop += 1;

return data[top];

}

// 1

int size() const

{

nop += 1;

return top + 1;

}

// 1

bool isEmpty() const

{

nop += 1;

return top == -1;

}

// 2

bool isFull() const

{

nop += 2;

return top == capacity - 1;

}

};

Adapter.h

#pragma once

#include "Stack.h"

class Adapter

{

Stack stack;

int capacity;

uint64\_t& nop;

public:

Adapter(uint64\_t& nop, int capacity = 5000)

:stack(nop, capacity), capacity(capacity), nop(nop) {}

// 1

int size() const

{

return stack.size();

}

// 1

bool isEmpty() const

{

return stack.isEmpty();

}

// 2

bool isFull() const

{

return stack.isFull();

}

// 10n + 6

int getElement(int ind)

{

int size = stack.size(); // 1

nop += 4;

if (!(ind >= 0 && ind < size))

return -1;

Stack\* tmp = new Stack(nop, capacity);

// 10n + 5

nop += 2;

nop += 2 \* size;

for (int i = 0; i < ind; i++) // 1 + 2n

{

tmp->push(stack.pop()); // 8n

}

int element = stack.pop(); // 3

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

{

stack.push(tmp->pop());

}

//

delete tmp;

return element;

}

// 10n + 4

int peekElement(int ind)

{

int size = stack.size(); // 1

nop += 4;

if (!(ind >= 0 && ind < size))

return -1;

Stack\* tmp = new Stack(nop, capacity);

// 10n + 1

nop += 2;

nop += 2 \* size;

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

{

tmp->push(stack.pop());

}

const int element = stack.peek();

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

{

stack.push(tmp->pop());

}

//

delete tmp;

return element;

}

// 16n

bool setElement(int ind, int element)

{

nop += 4;

int size = stack.size(); // 4

if (!(ind >= 0 && ind <= size))

return false;

Stack\* tmp = new Stack(nop, capacity);

// 16n

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

tmp->push(stack.pop());

tmp->push(element);

for (int i = ind + 1; i < size; i++)

tmp->push(stack.pop());

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

stack.push(tmp->pop());

delete tmp;

return true;

}

void push(int x) // 5

{

stack.push(x);

}

void display() const // 3 + 6n

{

stack.display();

}

};

Source.cpp

#include <iostream>

#include <random>

#include <chrono>

#include "Adapter.h"

using namespace std::chrono;

class Timer

{

typedef high\_resolution\_clock Clock;

private:

Clock::time\_point start;

public:

Timer()

:start(Clock::now()) {}

long long get\_time\_ms() const

{

return duration\_cast<milliseconds>(Clock::now() - start).count();

}

};

void bubble\_sort(Adapter& adapter, uint64\_t& nop) // F(n) = 56n^3 + 18n^2

{

nop += 1;

for (int i = 0; i < adapter.size() - 1; i++) // n

{

nop += 1;

nop += 2;

for (int j = 0; j < adapter.size() - i - 1; j++) // n

{

nop += 4;

if (adapter.peekElement(j) > adapter.peekElement(j + 1)) // 20n + 8

{

int tmp = adapter.peekElement(j + 1);

adapter.setElement(j, tmp);

adapter.getElement(j + 2);

}

}

}

}

void work(int size)

{

std::mt19937 rng{ std::random\_device()() };

std::uniform\_int\_distribution<int> dist{ -20, 20 };

uint64\_t nop = 0;

Adapter a(nop, size + 10);

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

{

a.push(dist(rng));

}

const Timer t;

bubble\_sort(a, nop);

const auto time = t.get\_time\_ms();

std::cout << "N=" << size << "; Elapsed time: " << time << "ms = " << time / 1000.0f << "s; nop=" << nop <<"\n-------\n";

}

int main()

{

for (int i = 0; i < 10; i++)

{

work( (i+1) \* 300);

}

}