**ФЕДЕРАЛЬНОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ**

**УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО**

**ОБРАЗОВАНИЯ**

«Московский технологический университет»

Студент: Шмелёв К.А.

Специальность: 09.03.04

(Программная инженерия)

Форма обучения: заочная

**Курсовая работа по дисциплине:**

**“Структуры и алгоритмы обработки данных”**

МГУПИ – 2018

Оглавление

[Задание на курсовую работу 3](#_Toc509611949)

[Введение 3](#_Toc509611950)

[Теоретическая часть 3](#_Toc509611951)

[Описание очереди с головой и хвостом 3](#_Toc509611952)

[Алгоритм сортировки Шелла 4](#_Toc509611953)

[Методика построения f(n) 4](#_Toc509611954)

[Практическая часть 4](#_Toc509611955)

[Вывод f(n) 4](#_Toc509611956)

[Общая схема эксперимента 5](#_Toc509611957)

[Листинг программы 6](#_Toc509611958)

## Задание на курсовую работу

В рамках курсовой работы по дисциплине «Структуры и алгоритмы обработки данных» требуется программно реализовать абстрактный тип данных (АТД) в соответствии с заданием. Абстрактный тип данных должен позволять осуществлять только операции, присущие типу линейного связанного списка: получить значение первого элемента (на выходе), добавить элемент (в конец списка), удалить элемент из списка (на выходе), проверить – список пуст, обнулить (проинициализировать) список. Используя разработанный АТД и указанный набор операций, необходимо реализовать заданный алгоритм сортировки последовательности динамических элементов.

На основе предложенной в рамках В курсовой работе программы получить аналитическую оценку трудоемкости работы алгоритма сортировки, используя О-символику.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Указатели | Очередь с головой и хвостом | Алгоритм Шелла |

Исполнитель : К.А. Шмелев

Утверждаю:

## Введение

Вначале были компьютеры, но перед ними были алгоритмы. Теперь же, когда есть множество компьютеров, есть еще больше алгоритмов которые лежит в основе вычислений.

Каждый алгоритм обладает большой практической ценностью. Часто в зависимости от различной реализации данного алгоритма сильно изменяется их сложность. Таким образом подсчет сложности для определенного проекта остается актуальным по текущий момент.

Поскольку один из важнейших критериев разработки программного обеспечения является их эффективность необходимо уметь прогнозировать и улучшать их. Эффективность программного обеспечения, в большинстве случаев, измеряется эффективностью алгоритма, применённого в нем.

Эффективность алгоритма может определяться по нескольким критериям, таким как время, количество тактовых операций процессора, количество используемой памяти.

Так же следует учитывать различные способы оценки эффективности алгоритма, таким образом в данной курсовой работе мы будет исходить из подхода “наихудшего” случая.

## Теоретическая часть

### Описание очереди с головой и хвостом

Стеки и очереди представляют собой динамические множества, элементы который удаляются заранее предопределенной операцией. В очереди всегда удаляется элемент который содержится в множестве дольше других, что описывается стратегией FIFO («first input, first output»,” первый вошел, первый вышел”). Существует несколько эффективных способов реализации очередей в программах, в нашем случае мы будем реализовывать очередь с помощью класса **Que**, в теле которого будет содержаться голова **head,** хвост **tail.**

### Алгоритм сортировки Шелла

Алгоритм сортировки, являющийся усовершенствованным вариантом сортировки вставками. Идея метода состоит в сравнении элементов не только рядом стоящих, но и на определенном расстоянии друг от друга. Иными словами, это сортировка вставками с предварительными грубыми проходами. Аналогичный метод усовершенствования пузырьковой сортировки называется сортировкой расческой.

При сортировке Шелла сначала сравниваются и сортируются значения стоящие на каком-либо расстоянии **d,** после этого процедура повторяется для некоторых меньших **d.** А завершается сортировкой элементов при **d=1** обычной сортировкой вставками.

Эффективность сортировки обеспечивается тем что элементы могут быстрее встать на свои места.

Несмотря на то что сортировка Шелла медленнее чем быстрая сортировка, она имеет ряд преимуществ:

* Отсутствие потребности в памяти под стек
* Отсутствие деградации при неудачных наборах данных, так быстрая сортировка легко деградирует до О(n^2), что хуже, чем гарантированное время сортировки алгоритмом Шелла.

### Методика построения f(n)

В качестве базовых операций будет использоваться:

1. Обращение к элементу – 1 операция
2. Присваивание - 2 операции
3. Арифметические операции – 2 операции
4. Выделение памяти – 2 операции
5. Логические операции – 1 операция
6. Открытие потока данных -1 операция
7. Использование стандартных средств ввода-вывода – 1 операция
8. Назначение языка по умолчанию – 1 операция

Таким образом мы получим базовые методы класса и их сложность:

1. bool is\_empty(); - 2
2. bool push(int); - 9 || 8
3. bool pop(); - 6||3
4. void print(); - n+4
5. bool init(int); - 19n-2
6. int size(); - 2n+3
7. int head\_ret(); - 2
8. int tail\_ret(); - 2
9. void set(int ,int); - 3
10. int get(int); - 1
11. friend void shell\_sort(Que &); - 25n^2/2
12. bool print\_to\_file(); - 2n+1

## Практическая часть

### Вывод f(n)

Следуя из методики построения f(n) и листинга самой программы получаем:

int count = 1000000000; - 2

setlocale(LC\_ALL, "RUS"); -1

Que pQ; -1

pQ.init(10000); - 19n-2

pQ.print(); n+4

pQ.print\_to\_file(); 2n+1

shell\_sort(pQ); 25n^2/2

pQ.print(); n+4

pQ.print\_to\_file();2n+1

cout << endl; 2

cout << pQ.head\_ret() << endl; 2+ 2

cout << pQ.tail\_ret() << endl; 2+2

getch(); 1

итого **f(n)** = 6n+21 + 12.5 \*n^2 = **12.5n^2 +6n +21**

### Общая схема эксперимента

Рассмотрим зависимость алгоритма от кол-ва входных данных. Таким образом определим время затраченное нашим алгоритмом и кол-во операций которое он фактически исполняет. Построим график зависимости. Проверим верно ли мы подсчитали кол-во операций. Построим графики зависимостей.

|  |  |  |
| --- | --- | --- |
| Кол-во элементов(N) | Время / T(n) | Кол-во операций(operation) |
| 100 | 54ms | 9097 |
| 1000 | 295ms | 107500 |
| 10000 | 2493ms | 1462536 |
| 22000 | 5415ms | 3383869 |

Сравним полученные данные с нашей формулой.

**f(100)** = 125000+ 600+21 = 125621

**f(1000)**= 12500000 + 6000+21 = 12506021

**f(10000)** = 1250000000 + 60000+21 = 1250060021

**f(22000)** = 6050000000+ 132000 +21 = 6050132021

Дополним таблицу:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во элементов(N) | Время / T(n) | Кол-во операций(operation) | F(n) | O(f(n)) |
| 100 | 54ms | 9097 | 125621 | 125000 |
| 1000 | 295ms | 107500 | 12506021 | 12500000 |
| 10000 | 2493ms | 1462536 | 1250060021 | 1250000000 |
| 22000 | 5415ms | 3383869 | 6050132021 | 6050000000 |

Получим следующие зависимости:

|  |  |  |  |
| --- | --- | --- | --- |
| C1=T(n)/F(n) | C2=T(n)/operation | C3=oper/F(n) | C4 =oper/O(f(n)) |
| 4,298644334943998e-7 | 5,936022864680664e-6 | 0,0724162361388621 | 0,072776 |
| 2,358863216638638e-8 | 2,744186046511628e-6 | 0,0085958574843611 | 0,0086 |
| 1,994304235106074e-9 | 1,705940913591187e-6 | 0,0011699726188508 | 0,0011700288 |
| 1,352397602397602e-9 | 1,600239252760671e-6 | 8,452e-4 | 5,5931719e-4 |

## Листинг программы

#include <conio.h>

#include <time.h>

#include <iostream>

#include <ostream>

#include <fstream>

#include <chrono>

//#define count 10000;

using namespace std;

using namespace std::chrono;

int operation = 0;

int SIZE = 22000;

int\* create\_massive(int, int, int);

int\* shell\_sort(int\*,int);

class Que

{

public:

//static const int SIZE = 100; // кол - во элементов - 1, тк хеад не определен

Que();

~Que();

bool is\_empty();

bool push(int); //добавить tail

bool pop(); //Удалить head

void print();

bool init(int);

int size();

int head\_ret();

int tail\_ret();

void set(int ,int);

int get(int);

friend void shell\_sort(Que &);

bool print\_to\_file();

bool read\_fr\_file();

private:

int\* que;

int head;

int tail;

};

Que::Que() //5

{

operation += 5;

que = new int[SIZE];//пока работаем с SIZE элементами

head = tail = 0;

//TEST::вначале очередь состоит из одного 0 = FALSE

//que[head] = que[tail] = 0;

}

Que::~Que()

{

}

bool Que::is\_empty() //2

{

operation += 2;

if (head == tail) //1

{

return false; //1

}

return true; //1

}

bool Que::push(int new\_el) // 9 || 8

{

operation += 9;

if (tail + 1 == head || (tail + 1 == SIZE && !head)) // 7

{

return false; //очередь заполнена //2

}

else

{

tail++; //2

if (tail == SIZE) tail = 0; //3

que[tail] = new\_el; //1

}

return true; // успешно добавлен элемент в хвост //2

}

bool Que::pop() // 6||3

{

operation += 6;

if (head == tail) //1

{

return false; //очередь пустая //2

}

else

{

head++; //2

if (head == SIZE) //1

{

head = 0; //прошли весь que //1

}

}

return true; //2

}

void Que::print() //n+4

{

operation += SIZE+4;

cout << "Вывод очереди: " << endl; //2

for (int i = head+1;i < tail+1;i++) // n

{

cout << " " << que[i]; //2

}

}

bool Que::init(int count) // 19n-2

{

operation += 19\*SIZE - 2;

int\* mass1 = new int[count]; //2

mass1 = create\_massive(count, 500, 0); //1 +10n+3

//заполняем список

for (int i = 0;i < count - 1;i++) //n-1

{

push(mass1[i]); //9 || 8

}

return false; //1

}

int Que::size() // 2n+3

{

operation += 2\*SIZE +3;

int count = 0; //2

for (int i = head; i < tail; i++) //n

{

count++; //2

}

return count; //1

}

int Que::head\_ret()

{

operation += 1;

return que[head+1];

}

int Que::tail\_ret()

{

operation += 1;

return que[tail];

}

void Que::set(int i,int j)

{

operation += 2;

que[i] = j;

}

int Que::get(int i)

{

operation += 1;

return que[i];

}

bool Que::print\_to\_file() // 2n+1

{

operation += 2\*SIZE +1;

ofstream file\_out; //1

file\_out.open("test.txt"); //1

if (!file\_out.is\_open()) //4

{

cout << "file cant be opened" << endl;

getch();

return false;

}

else

{

for (int i = head + 1;i < tail + 1;i++) // n

{

file\_out << " " << que[i]; //2

}

return true;

}

}

bool Que::read\_fr\_file()

{

operation += 1;

return false;

}

void shell\_sort(Que &q) // n/2\*n(6+(n/n-3)\*4) = 4n/n +21 \* n^2/2 = 25n/n\*n^2/2 = 25n^2/2

//сортировка для очереди

//Переделать что бы не было произвольного доступа или нет?

{

operation += 12.5 \* SIZE + 1;

int count = q.size(); //2

int i, j, step;

int mass\_change;

for (step = count / 2; step > 0; step /= 2) // n/2

for (i = step; i <= count; i++) //n

{

mass\_change = q.get(i); //3

for (j = i; j >= step; j -= step) // n /n-3

{

if (mass\_change < q.get(j - step)) //3

{

q.set(j, q.get(j - step)); //3+ +1

}

else

break; //1

}

q.set(j, mass\_change); //3

}

}

void main()

{

auto start = steady\_clock::now();

int count = SIZE; //изменяем вместе с SIZE

setlocale(LC\_ALL, "RUS");

Que pQ;

pQ.init(count);

pQ.print();

pQ.print\_to\_file();

shell\_sort(pQ);

pQ.print();

pQ.print\_to\_file();

cout << endl;

cout << pQ.head\_ret() << endl;

cout << pQ.tail\_ret() << endl;

auto finish = steady\_clock::now();

cout << duration\_cast<milliseconds>(finish - start).count() << "ms" << endl;

cout << operation << endl;

getch();

/\*TEST:массив = TRUE

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

{

cout << " " << mass1[i];

file\_out << " " << mass1[i];

}

cout << endl << endl;

shell\_sort(mass1,count);

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

{

cout << " " << mass1[i];

file\_out << " " << mass1[i];

}

getch();

\*/

}

int\* create\_massive(int count, int range\_max, int range\_min) //10n+3

{

int i;

int \*p;

p = new int[count]; //выделяем память для count элементного массива //2

srand((unsigned)time(NULL)); //1

for (i = 0; i < count; i++) //n\*10

{

p[i] = (double)rand() / (RAND\_MAX + 1) \* (range\_max - range\_min) + range\_min; // 10

p[i + 1] = '\0'; //2

}

return p;//2

}

int\* shell\_sort(int\* mass, int count)

{

int i, j, step;

int mass\_change;

for (step = count / 2; step > 0; step /= 2)

for (i = step; i < count; i++)

{

mass\_change = mass[i];

for (j = i; j >= step; j -= step)

{

if (mass\_change < mass[j - step])

mass[j] = mass[j - step];

else

break;

}

mass[j] = mass\_change;

}

return mass;

}

/\* TEST:: шел сорт при установленном d = FALSE

for (d1=count/2;d1>0;d1=d1/2)//пробегаем половину элементов +1 на всякий

{

for (int j = d1;j <count/2;j++)

{

mass\_change = mass[j];

//cout << j << " прическа d1: ";

for (i = j;i >= d1;i = i - d1)

{

if (mass\_change <mass[i + d1])

{

mass[i] = mass[i + d1];

}

else break;

}

mass[i] = mass\_change;

}

\*/