|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Программное обеспечение ЭВМ и информационные технологии

ДИСЦИПЛИНА Типы и структуры данных

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №5 «ОБРАБОТКА ОЧЕРЕДЕЙ».**

Студент Иванов Павел Александрович

Группа ИУ7 – 35Б

Вариант №2

Проверил

**Оглавление**

[Цель работы 2](#_Toc86744035)

[Описание условия задачи 2](#_Toc86744036)

[Описание технического задания 3](#_Toc86744037)

[Описание структур данных 5](#_Toc86744038)

[Описание алгоритма 8](#_Toc86744039)

[Набор тестов 8](#_Toc86744040)

[Расчёт времени работы очереди 10](#_Toc86744041)

[Оценка эффективности 11](#_Toc86744042)

[Ответы на контрольные вопросы 12](#_Toc86744043)

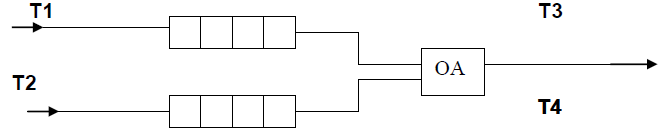
[Вывод 15](#_Toc86744044)

# **Цель работы**

Приобрести навыки работы с типом данных «очередь», представленным в виде одномерного массива и односвязного линейного списка, провести сравнительный анализ реализации алгоритмов включения и исключения элементов из очереди при использовании указанных структур данных, оценить эффективности программы по времени и по используемому объему памяти.

Описание условия задачи

Система массового обслуживания состоит из обслуживающего аппарата (ОА) и двух очередей заявок двух типов.

Заявки 1-го и 2-го типов поступают в "хвосты" своих очередей по случайному закону с интервалами времени Т1 и Т2, равномерно распределенными от 1 до 5 и от 0 до 3 единиц времени (е.в.) соответственно.

В ОА они поступают из "головы" очереди по одной и обслуживаются также равновероятно за времена Т3 и Т4, распределенные от 0 до 4 е.в. и от 0 до 1 е.в. соответственно, после чего покидают систему (все времена – вещественного типа). В начале процесса в системе заявок нет.

Заявка 2-го типа может войти в ОА, если в системе нет заявок 1-го типа. Если в момент обслуживания заявки 2-го типа в пустую очередь входит заявка 1-го типа, то она ждет первого освобождения ОА и далее поступает на обслуживание (система с относительным приоритетом).

Смоделировать процесс обслуживания первых 1000 заявок 1-го типа. Выдать на экран после обслуживания каждых 100 заявок 1-го типа информацию о текущей и средней длине каждой очереди, количестве вошедших и вышедших заявок и о среднем времени пребывания заявок в очереди. В конце процесса выдать общее время моделирования и количество вошедших в систему и вышедших из нее заявок обоих типов. По требованию пользователя выдать на экран адреса элементов очереди при удалении и добавлении элементов. Проследить, возникает ли при этом фрагментация памяти.

Интерфейс программы должен быть понятен неподготовленному пользователю. При разработке интерфейса программы следует предусмотреть:

* возможность изменения времен прихода и обработки заявок с экрана;
* наличие пояснений при выводе результата;
* вывод процента погрешности работы программы.

Кроме того, нужно вывести на экран время выполнения программы при реализации очереди списком и массивом, а также требуемый при этом объем памяти.

При тестировании программы необходимо:

* проверить правильность работы программы при различном заполнении очередей, т.е., когда время моделирования определяется временем обработки заявок и когда определяется временем прихода заявок;
* отследить переполнение очереди, если очередь в программе ограничена.

При реализации очереди списком необходимо тщательно следить за освобождением памяти при удалении элемента из очереди. При этом по запросу пользователя должны выдаваться на экран адреса памяти, содержащие элементы очереди при добавлении или удалении элемента очереди. Следует проверять, происходит ли при этом фрагментация памяти (выделение непоследовательных адресов память).

# **Описание технического задания**

Входные данные:

*Целое число:*

Число от 0 до 4 — пункт меню или число заявок первого типа от 1 до 1 000 000.

*Вещественное число:*

Число от 0 до 10 000 — временная граница обработки или поступления заявки.

Выходные данные:

Числа – число обработанных заявок, длина и средняя длина очередей, число вошедших и вышедших заявок, среднее время, общее время выполнения программы, объем памяти, занимаемой структурами, в байтах.

Действие программы:

Выполнение операций с очередью, решение задачи с двумя очередями и одним обработчиком, над которым установлен относительный приоритет в пользу заявок первой очереди. Также сравнение эффективности операций при использовании различных структур данных.

*Меню программы:*

1 – Решение задачи с использованием массива

2 – Решение задачи с использованием списка

3 – Изменение исходных данных задачи

4 – Измерение времени и эффективности операций

Обращение к программе:

Запускается файл app.exe. Если его нет, из консоли следует вызывать make, и тогда запустить app.exe.

Аварийные ситуации:

* При очистке очистке очередь пустая

Результат: сообщение «Queue is already empty!»

* При добавлении возникло переполнение очереди

Результат: сообщение «Queue is already full!»

* Получение элемента из пустой очереди

Результат: сообщение «Queue is already empty!»

* Некорретный ввод пунта меню

Результат: сообщение «Wrong input!»

* Некорретный ввод числа заявок первой очереди

Результат: сообщение «Wrong input!»

* Некорретный ввод временных границ

Результат: сообщение «Wrong input!»

# **Описание структур данных**

Для хранения очереди — массива (очередь кольцевая) используется следующая структура:

typedef struct arr\_queue

{

int pin; // указатель на конец

int pout; // указатель на начало

int size; // размер массива

int count; // количество элементов в очереди

double \*arr; // массив, элементы коротого - время прихода

} arr\_queue;

Поля структуры:

int pin – указатель (индекс) на последний добавленный элемент;

int pout – указатель (индекс) на первый элемент в очереди;

int size – размер очереди;

int count – количество элементов в очереди;

double \*arr – сам массив, где хранятся элементы очереди.

Для хранения очереди — списка используется следующая структура:

typedef struct list\_queue

{

int count;

list\_node \*pout; // первый элемент на выход

list\_node \*pin; // адрес последнего элемента

} list\_queue;

Поля структуры:

int count — число элементов в очереди;

list\_node \*pout — указатель на первый элемент в очереди;

list\_node \*pin – указатель на последний элемент в очереди.

Для хранения списка свободных адресов используется:

typedef struct free\_mem

{

size\_t data;

struct free\_mem \*next;

} free\_mem;

Поля структуры:

size\_t data – адрес

struct free\_mem \*next – следующий элемент списка

Для обработчика используется следующая структура данных, которая хранит всю необходимую информацию:

typedef struct handler\_t

{

    int type1\_processed;

    int type2\_processed;

    double time\_standby;

    request\_t in\_process;

    float time\_finish;

} handler;

Поля структуры:

int type1\_processed – число обработанных заявок первого типа;

int type2\_processed – число обработанных заявок второго типа;

double time\_standby – время «простоя» обработчика;

request\_t in\_process – тип обрабатываемой заявки;

float time\_finish – время, к которому заявка будет обработана.

Тип заявки – перечислимый тип:

typedef enum { NONE, ONE, TWO } request\_t;

NONE – заявки нет;

ONE – заявка из первой очереди;

TWO – заявка из второй очереди.

*Основные функции программы:*

// Инициализация очереди

int arr\_queue\_init(arr\_queue \*queue);

// Добавление элемента в конец очереди

int arr\_queue\_push(arr\_queue \*queue, double come\_time);

// Удаление элемента из очереди

int arr\_queue\_pop(arr\_queue \*queue, double \*ret\_time);

// Освобождение памяти под очередь

void arr\_queue\_free(arr\_queue \*queue);

// Очистка очереди

void arr\_queue\_clean(arr\_queue \*queue);

// Печать очереди

void arr\_queue\_print(arr\_queue \*queue);

// Инициализвция очереди

void list\_queue\_init(list\_queue \*queue);

// Создание ноды списка

int create\_node(list\_node \*\*node, double tmp);

// Добавление элемента в очередь

int list\_queue\_push(list\_queue \*queue, double tmp, free\_mem \*mem);

// Удаление элемента из очереди

int list\_queue\_pop(list\_queue \*queue, double \*tmp, free\_mem \*mem);

// Очистка очереди

void list\_queue\_clean(list\_queue \*queue, free\_mem \*mem);

// Печать очереди

void list\_queue\_print(list\_queue \*queue, free\_mem \*mem);

// Инициализия списка свободных областей

void free\_mem\_init(free\_mem \*mem);

// Добавление адреса в список свободных адресов

int free\_mem\_add(free\_mem \*mem, size\_t add);

// Удаление адреса из списка свободных адресов при его наличии

void free\_mem\_del(free\_mem \*mem, size\_t del);

// Вывод списка свободных адресов

void free\_mem\_print(free\_mem \*mem);

# **Описание алгоритма**

1. Ввод номера команды;
2. Если выбрано моделирование с помощью массива:
   1. Создается цикл, который идёт пока не обработается нужное число заявок первого типа. В это же время создается счетчик времени. По ходу времени генерируется время поступления следующей заявки в каждую из очередей. Если обработчик свободен или только что освободился, в него добавлется заявка (если есть в первой очереди, то из первой — иначе из второй).
   2. По окончании цикла выводится сравнение полуенных результатов с идеальными
3. Если выбрано моделирование с помощью списка:
   1. Создается цикл, который идёт пока не обработается нужное число заявок первого типа. В это же время создается счетчик времени. По ходу времени генерируется время поступления следующей заявки в каждую из очередей. Если обработчик свободен или только что освободился, в него добавлется заявка (если есть в первой очереди, то из первой — иначе из второй).
   2. По окончании цикла выводится сравнение полуенных результатов с идеальными
4. Если выбрано сравнение по времени, то выводится сравнительная таблица времени добавления и удаления элемента из очереди, а также объём занимаемой структурами памяти при разной наполненности очередей.

# **Набор тестов**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Тестовый случай | Пользовательский ввод | Результат |
| 1 | Введён неверный пункт меню | a | Wrong input! |
| 2 | Введён неверный пункт меню | 5 | Wrong input! |
| 3 | Введено неверное количество заявок первого типа | 0 | Wrong input! |
| 4 | Введено неверное количество заявок первого типа | -10 | Wrong input! |
| 5 | Введено неверное количество заявок первого типа | 1000001 | Wrong input! |
| 6 | Ввод неверного промежутка времени | -1 5 | Wrong input! |
| 7 | Ввод неверного промежутка времени | 0 0 | Wrong input! |
| 8 | Ввод неверного промежутка времени | 0.0 10000.01 | Wrong input! |
| 9 | Ввод неверного промежутка времени | 10.5 10.5 | Wrong input! |
| 10 | Ввод неверного промежутка времени | 10 -5 | Wrong input! |
| 11 | Ввод неверного промежутка времени | 5.0 3.0 | Wrong input! |
| 12 | Выход из программы | 0 | Программа завершилась |
| 13 | Запуск программы, использующей очередь на массиве | 1 | Обработано установленное число заявок, вывдено сравнение реального и ожидаемого времени |
| 14 | Запуск программы, использующей очередь на списке | 2 | Обработано установленное число заявок, вывдено сравнение реального и ожидаемого времени |
| 15 | Вывод результатов сравнения эффективности | 4 | Выведена сравнительная таблица операций по времени и по памяти |

# **Расчёт времени работы очереди**

Расчитаем время работы очереди.

Дано:

T1 : [1, 5]

T2 : [0, 3]

T3 : [0, 4]

T4 : [0, 1]

Число заявок первой очереди — 1000;

*В данном случае* время обработки (2 с.) меньше, чем время прихода (3 с.), а значит обе очереди будут либо пустые на каком-то моменте в времени, либо в них будет поддерживаться постоянное число заявок. В таком случае, время моделирования определяется временем прихода.

Идеальное время моделирования: 3 \* 1000 = 3000 е.в.

Время обработки заявок первого типа: 2 \* 1000 = 2000 е.в.

Тогда под заявки второй очереди выделяется 3000 – 2000 = 1000 е.в., а значит за это время в среднем обработается 2000 заявок второго типа.

В случае же когда *время обработки больше*, чем время прихода, ситуация меняется. Пусть теперь Т3: [0, 10].

Теперь время моделирования определяется временем обработки. Время моделирования = 5 \* 1000 = 5000 е.в.

Число заявок первого типа: 1000

Время их обработки: 1000 \* 5 = 5000

Число заявок второго типа: 5000 / 1.5 = 3300

Обработается заявок второго типа: 0

Время обработки равно времени прихода:

Пусть теперь, в сравнении с первым случаем, Т2: [0, 1]. Тогда время моделирования, как и в первом случае, 3000 е.в. Обработано так же 2000 заявок первого типа. Вторых же заявок вошло 6000, и обработалось 1500.

# **Оценка эффективности**

Оценка эффективно­­сти проводится по операциям добавления элемента и извлечения элемента. Измеряется число усреднённое число тактов:

Операция добавления:

|  |  |  |
| --- | --- | --- |
| Число прогонов | Очередь — массив, такты | Очередь — список, такты |
| 1 000 | 128 | 1387 |
| 1 000 000 | 133 | 1355 |

Операция извлечения:

|  |  |  |
| --- | --- | --- |
| Число прогонов | Очередь — массив, такты | Очередь — список, такты |
| 1 000 | 96 | 1076 |
| 1 000 000 | 97 | 1130 |

Память:

|  |  |  |
| --- | --- | --- |
| Число элементов | Очередь — массив, 1500 элементов, байт | Очередь — список, байт |
| 10 | 12016 | 180 |
| 20 | 12016 | 340 |
| 40 | 12016 | 660 |
| 80 | 12016 | 1300 |
| 160 | 12016 | 2580 |
| 320 | 12016 | 5140 |
| 640 | 12016 | 10260 |
| 1280 | 12016 | 20500 |

# **Ответы на контрольные вопросы**

1. *Что такое FIFO и LIFO?*

FIFO и LIFO – принципы работы структуры данных. FIFO – First in – First Out (первый пришел, первый ушёл). LIFO – Last in – First Out (последний пришел, первый ушёл). По принципу FIFO работают очереди, а по принципу LIFO работает стек.

1. *Каким образом, и какой объем памяти выделяется под хранение очереди при различной ее реализации?*

В случае списка, память под очередь выделяется каждый раз, когда добавляется новый элемент. В этом случае ищется свободный кусочек памяти в куче, который и займет ячейка списка. В это случае объем занимаемой памяти будет равен sizeof(тип данных) + sizeof(указатель на следующую ячейку).

В случае массива, память можно выделить статически и динамически. В первом случае она выделяется при компиляции, во втором во время исполнения программы ищется непрерывный кусок памяти нужного размера. Массив будет занимать (макс. число элементов) \* sizeof(тип данных) байт.

1. *Каким образом освобождается память при удалении элемента из очереди при ее различной реализации?*

В случае списка, возвращается значение данных в удаляемой ячейке, указатель начала очереди смещается на следующий элемент и память под удаляемую ячейку освобождается. В случае массива, при удалении элемента память не освобождается, а просто сдвигается указатель pout. Память под массив освобождается в конце выполнения программы.

1. *Что происходит с элементами очереди при ее просмотре?*

Все элементы очереди при её просмотре удаляются.

1. *От чего зависит эффективность физической реализации очереди?*

Эффективность физической реализации очереди зависит от поставленной задачи и максимального объема данных, который очередь будет принимать. Так, если мы не знаем, какого максимального размера очередь, то может возникнуть переполнение, или наоборот, излишнее использование памяти, поскольку если очередь небольшая, то выделять огромный кусок памяти просто не эффективно. Если же память не так важна, но важно время — массив практически всегда быстрее списка.

1. *Каковы достоинства и недостатки различных реализаций очереди в зависимости от выполняемых над ней операций?*

Недостатки реализации массивом: легко может возникнуть переполнение, трудно угадать размер очереди, и, как следствие, часто излишние траты по памяти. Если очередь кольцевая, реализация операций получается не самая простая в сравнении со списком.

Достоинства реализации массивом: операции выполняются быстро.

Недостатки реализации списком: на больших объёмах данных список занимает больше памяти, чем массив, а также может возникать фрагментация памяти. Скорость операций – медленнее, чем у массива.

Достоинства реализации списком: простота выполнения операций, на малых размераз очередей список не требует больших затрат по памяти.

1. *Что такое фрагментация памяти, и в какой части ОП она возникает?*

Фрагментация – явление чередования «свободных» и «занятых» участков памяти. При фрагментации данные могут записываться не подряд, а быть «разбросанными» по памяти. Она возникает вследствие постоянного выделения и освобождения памяти — тогда возникают свободные участки всё более малых размеров. При сильной фрагментации объём свободной памяти может быть большой, но может не найтись ни одного свободного куска памяти большого размера. Фрагментация возникает в куче.

1. *Для чего нужен алгоритм «близнецов».*

В алгоритме выделения памяти «близнецов» последовательные размеры блоков ограничиваются степенями двойки. Он используется там, где нужна скорость — например, для задач реального времени, также используется для выделения памяти внутри ядра ОС.

1. *Какие дисциплины выделения памяти вы знаете?*

Существуют разные подходы к выделению памяти: «самый подходящий» (выделяется тот участок, размер которого равен или минимально превышается требуемый) и «первый подходящий» (когда выделяется самый первый участок памяти, в котором есть требуемый объем).

1. *На что необходимо обратить внимание при тестировании программы?*

Необходимо првоерять правильность работы программы при различном заполнении очредей (при различных отношениях между временем обработки и временем прихода), а также отследить переполнение очереди. Если используется список, необходимо следить за правильным освобождением и выдлением памяти.

***11. Каким образом физически выделяется и освобождается память при динамических запросах?***

При выделении памяти в куче ищется (в зависмости от дисциплины выделения памяти) необходимый кусок памяти. Если найденный фрагмент памяти больше необходимого, забирается необходимый объем, а оставшаяся часть памяти остается быть свободной. При освобождении памяти, фрагмент добавляется в «карту» свободной памяти, и в следующий раз, когда мы снова запросим память, на это же место может опять записаться какая-то информация.

Вывод

В ходе данной работы я познакомился со структурой данных «очередь», разобрался с операциями, которые можно реализовывать на этой структуре и сравнил реализацию очереди списком и массивом.

Сказать, что же лучше, массив или список для реализации очереди – трудно. В завимости от задачи и требований к ней нужно выбирать правильную структуру данных. Так, к преимуществам массива можно отнести скорость выполнения операций, а к недостаткам — ограниченность и затратность по памяти. В случае списка, удобно его использовать на относительно небольших объемах данных или когда мы не знаем, сколько памяти нам нужно будет. По моим замерам вышло, что где-то до 50% максимального размера очереди имеет смысл использовать список. Из недостатков списка можно отметить его медленную скорость работы по операциям добавления и удаления элемента.

Соотнося результаты с результатами лабораторной работы №4, можно увидеть параллели между эффективностью стека и очереди (структур LIFO и FIFO) на массивах и списках — вышесказанное актуально также и для стека.