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» (ИУ)
КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии» (ИУ7)

Лабораторная работа №5

“Обработка очередей”

Вариант №8

Студент:
Князев Дмитрий Юрьевич, группа ИУ7-33Б

_____
(подпись, дата)

Преподаватель:
Барышникова Марина Юрьевна

_____
(подпись, дата)

2022 г.

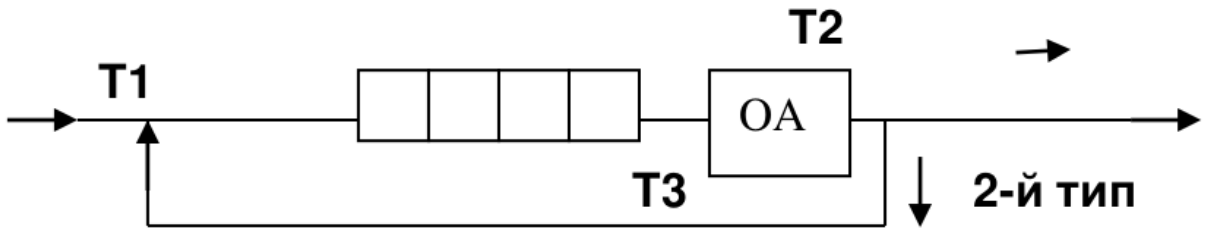
Оглавление

Цель работы:

Отработка навыков работы с типом данных «очередь», представленным в виде одномерного массива и односвязного линейного списка. Сравнительный анализ реализации алгоритмов включения и исключения элементов из очереди при использовании двух указанных структур данных. Оценка эффективности программы (при различной реализации) по времени и по используемому объёму памяти.

Задание по варианту:

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



Заявки 1-го типа поступают в "хвост" очереди по случайному закону с интервалом времени T1, равномерно распределенным от 0 до 5 единиц времени (е.в.). В ОА они поступают из "головы" очереди по одной и обслуживаются также равновероятно за время T2 от 0 до 4 е.в., после чего покидают систему. Единственная заявка 2-го типа постоянно обращается в системе, обслуживаясь в ОА равновероятно за время T3 от 0 до 4 е.в. и возвращаясь в очередь не далее 4-й позиции от "головы". В начале процесса заявка 2-го типа входит в ОА, оставляя пустую очередь (все времена – вещественного типа).

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

Входные данные:

- Пункт меню (число от 0 до 10 включительно)
- Целые числа (в зависимости от пункта меню)

Выходные данные (в зависимости от выбранного пункта меню):

- Содержимое очередей в символьном представлении
- Информация об успешном или неудачном добавлении в очереди на основе списка и массива или об удалении элемента из них
- Результаты симуляции системы массового обслуживания
- Результаты сравнения эффективности алгоритмов сортировки и поиска с использованием ДДП, АВЛ дерева, хеш-таблиц и массива

Функции меню

0. Выход

Функции для очереди в виде массива

1. Добавить элемент в очередь
2. Удалить элемент из очереди
3. Отобразить очередь на экран

Функции для очереди в виде списка

4. Добавить элемент в очередь
5. Удалить элемент из очереди
6. Отобразить очередь на экран
7. Вывести массив освободившихся адресов

Функции для моделирования и вычисления эффективности

8. Смоделировать работу ОА с очередью на основе массива
9. Смоделировать работу ОА с очередью на основе списка
10. Сравнить эффективность работы очередей

Оценка эффективности производится в последнем пункте меню, предыдуще используются лишь для проверки работоспособности структур данных.

Структуры данных

```c
#define INITIAL_SIZE 100
#define EXPAND_FACTOR 2

struct array_queue
{
    int *data;
    size_t size;
    size_t begin;
    size_t end;
    size_t max_size;
};
```

Очередь на основе массива (динамического кольцевого буфера)

`INITIAL_SIZE` — начальный размер массива, передаваемый при создании структуры

`EXPAND_FACTOR` — множитель, определяет во сколько раз будет увеличен массив при его заполнении

`data` — указатель на динамический массив целых чисел, элементов очереди

`size` — текущий размер очереди

`begin` — индекс элемента, который будет удалён из очереди следующим

`end` — индекс элемента, который будет удалён из очереди последним

`max_size` — максимальный размер массива

```c
typedef struct node
{
    int value;
    struct node *next;
} node_t;

struct list_queue
{
```

```
    node_t *head;
    node_t *tail;
};
```

`struct node` — структура узла списка, на котором основана очередь
`value` — целое число, элемент очереди
`next` — указатель на следующий узел в списке
`list_queue` - очередь на основе списка
`head` — указатель на узел, который будет удалён из очереди следующим
`tail` — указатель на узел, который будет удалён из очереди последним

Основные функции

Заголовок функции	Описание
`list_queue_t * new_list_queue(void)`	Возвращает указатель на новую структуру очереди в виде списка
`int list_queue_insert(list_queue_t *list_queue, int value, ssize_t index)`	Вставляет новый элемент в произвольное место очереди на основе списка
`int list_queue_push(list_queue_t *list_queue, int value)`	Вставляет новый элемент в конец списка, возвращает EXIT_SUCCESS в случае успеха, иначе — ненулевой код ошибки
`int list_queue_pop(list_queue_t *list_queue, int *value)`	Удаляет первый элемент из очереди на основе списка, возвращает EXIT_SUCCESS в случае успеха, иначе — ненулевой код ошибки

`void` `list_queue_print(list_queue_t *list_queue)`	Функция для печати очереди на основе списка
`array_queue_t *` `new_array_queue(size_t init_size)`	Возвращает указатель на новую структуру очереди в виде массива
`int` `array_queue_expand(array_queue_t *array_queue)`	Увеличивает максимальный размер кольцевого буфера (с перемещением элементов при необходимости)

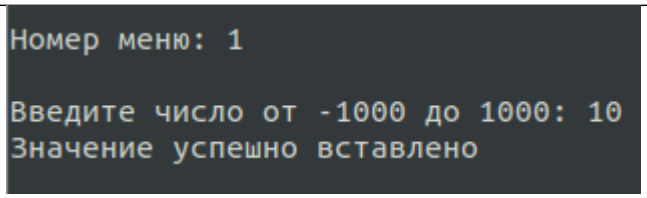
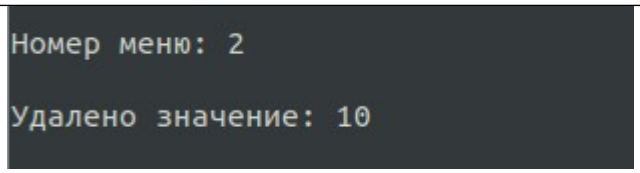
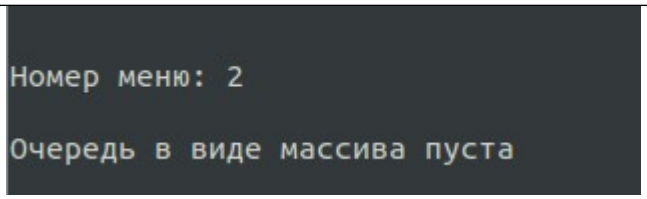
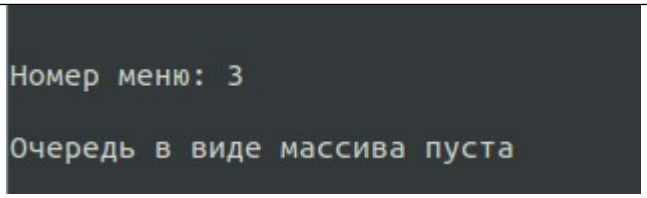
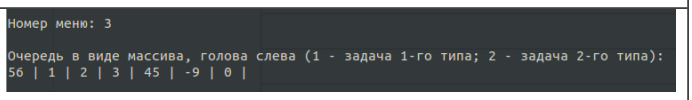
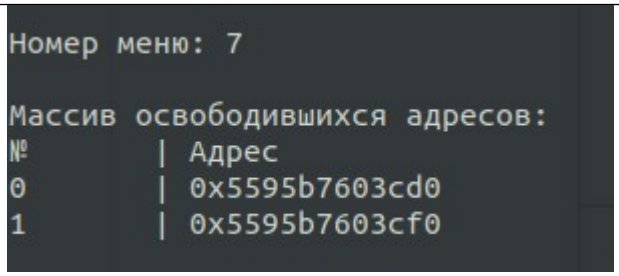
`int`
`array_queue_insert(array_queue_t *array_queue,`
`int value,`
`int index)`
	Вставляет новый элемент в произвольное место очереди на основе кольцевого буфера

`int`
`array_queue_push(array_queue_t *array_queue,`
`int value)`
	Вставляет новый элемент в конец кольцевого буфера, возвращает EXIT_SUCCESS в случае успеха, иначе — ненулевой код ошибки

`int`
`array_queue_pop(array_queue_t *array_queue,`
`int *value)`
	Удаляет первый элемент из очереди на основе кольцевого буфера, возвращает EXIT_SUCCESS в случае успеха, иначе — ненулевой код ошибки

`void` `array_queue_print(array_queue_t *array_queue)`	Функция для печати очереди на основе массива

Функциональные тесты

Описание	Результат
1. Добавить элемент в очередь (На основе массива)	
2. Удалить элемент из очереди (не пустой)	
2. Удалить элемент из очереди (пустой)	
3. Отобразить очередь на экран (пустую)	
3. Отобразить очередь на экран (1 элемент)	
3. Отобразить очередь на экран (несколько элементов)	
7. Вывести массив освободившихся адресов (после удаления нескольких элементов из очереди в виде списка)	

Моделирование системы массового обслуживания

Временные промежутки модели для заявок (распределённые равномерно)
T1: от 0 до 5 — время ожидания заявки первого типа
T2: от 0 до 4 — время выполнения заявки первого типа
T3: от 0 до 4 — время выполнения заявки второго типа

С момента запуска модели очередь будет расти из-за непрерывного поступления заявки 2-го типа на позицию не дальше 4-й.

Трудно точно теоретически рассчитать количество обращений заявки 2-го типа: вначале очередь короткая и эта заявка будет находиться ближе 4-й позиции от головы очереди, а с нарастанием очереди она будет находиться ровно на 4-й позиции. Однако можно представить, что она всегда будет находиться ровно на 4-й позиции (что и будет происходить начиная с некоторого момента времени), если время работы модели достаточно велико. Тогда количество обработанных заявок второго типа будет примерно в три раза меньше обработанных заявок первого типа.

Примерный результат работы модели системы массового обслуживания

```
Вошло задач 1-го типа: 1077
Вышло задач 1-го типа: 1000
Вошло задач 2-го типа: 340
Вышло задач 2-го типа: 339
Текущая длина очереди: 78
Средняя длина очереди: 46.400
Среднее время ожидания заявок 1-го типа в очереди (е.в.): 1.950
Среднее время ожидания заявок 2-го типа в очереди (е.в.): 1.967
Среднее время ожидания заявок в очереди (е.в.): 1.954


Время представлено в единицах времени (е.в.)
Время прихода заявок: 2618.692 (Ожидаемое время прихода заявок: 2666.667, относительная погрешность: 1.799%)
Время работы автомата: 2618.945 (Ожидаемое время работы автомата: 2666.667, относительная погрешность: 1.790%)
Время простоя аппарата: 0.252
Вошло задач 1-го типа: 1078
Вышло задач 1-го типа: 1000
Обращений задач 2-го типа: 339
```

Теоретический расчёт:
Среднее время ожидания заявки первого типа: (0 + 5) / 2 = 2,5 е.в.
Среднее время выполнения заявки первого типа: (0 + 4) / 2 = 2,0 е.в.
Среднее время выполнения заявки второго типа: (0 + 4) / 2 = 2,0 е.в.

Примерное количество обращений заявок 2-го типа:
1000 / 3 ≈ 333,333

За время моделирования обрабатывается 1000 заявок первого типа, значит время моделирования приблизительно равно:
1000 * 2,0 е.в. + 333,333 * 2,0 е.в. = 2666,667 е.в.
Относительная погрешность времени по выходу:
| 2618,945 - 2666,667 | / 2666,667 * 100% ≈ 1,790%

Время простоя обслуживающего аппарата приблизительно равно нулю. Это объясняется тем, что заявка второго типа в данной модели может находится всего в двух состояниях: либо в обслуживающем аппарате, либо в очереди (в модели не учитывается время на добавление и удаление заявок из очереди, на передачу заявки, а также на добавление и удаление заявок из обслуживающего аппарата).
Таким образом, суммарное время ожидания заявок равно времени моделирования.
Относительная погрешность времени по входу:
| 2618,692 - 2666,667 | / 2666,667 * 100% ≈ 1,799%

Оценка эффективности

Среднее время добавления элемента (в тиках) на основе 100 итераций

Количество элементов	Массив	Список
50	24,52	27,99
100	48,30	51,34
500	132,28	141,97
1000	266,28	288,61

Среднее время удаления элемента (в тиках) на основе 100 итераций

Количество элементов	Массив	Список
50	0,78	1,50
100	1,15	1,93
500	4,79	8,66
1000	8,73	16,16

Размер занимаемой памяти (в байтах)

Количество элементов	Массив	Список
50	240	816
100	440	1616
500	2040	8016
1000	4040	16016

Вывод:

Очередь на основе списка ограничена объёмом оперативной памяти, доступной программе, при его использовании может произойти фрагментация памяти.

Если реализовывать очередь на основе статического кольцевого буфера, то его размер будет ограничен размером стека, однако можно организовать очередь в виде кольцевого буфера, хранящегося в динамической памяти, с возможностью расширения и реструктуризации его элементов, что потребует дополнительного времени, но при этом исключается возможная фрагментация, так как память под массив выделяется одним куском.

Если реализовать очередь в виде списка, то операции добавления и удаления элементов будут на нем происходить дольше из-за необходимости выделения или освобождения памяти под узел списка. При этом добавление элемента в массиве происходит примерно на 8% (с учётом реструктуризаций), а удаление быстрее в 1,5 - 2 раза.

Также очередь на основе массива занимает приблизительно в 4 раза меньше памяти, чем очередь на основе списка при одинаковом количестве элементов.

Контрольные вопросы:

1. Что такое FIFO и LIFO?

Правило FIFO означает, что элемент, вошедший первым в структуру данных, также первым из неё и выйдет (как, например, в очереди).

Правило LIFO означает, что элемент, вошедший последним в структуру данных, выйдет из неё первым (как, например, в стеке).

2. Каким образом, и какой объем памяти выделяется под хранение очереди при различной её реализации?

При хранении кольцевым буфером: кол-во элементов * размер одного элемента очереди. Память под такую структуру может выделяться как на стеке, так и в куче (статический или динамический массив).

При хранении списком: кол-во элементов * (размер одного элемента очереди + указатель на следующий элемент). Для списка имеет смысл выделять память в куче для каждого элемента отдельно, хотя возможен и вариант размещения на стеке.

3. Каким образом освобождается память при удалении элемента из очереди при её различной реализации?

При хранении кольцевым буфером память не освобождается, а просто меняется указатель на конец очереди.

При хранении списком память, занимаемая удаляемым элементом, освобождается.

4. Что происходит с элементами очереди при её просмотре?

Как и при работе со стеком, можно увидеть лишь крайний элемент путём его удаления, но это верно только при работе со стеком как с АТД. Разработчик библиотеки, обладая доступом к полям структуры очереди, может реализовать функцию, которая будет проходить по её элементам без их удаления.

5. От чего зависит эффективность физической реализации очереди?

Зная максимальный размер очереди (если этот размер не слишком велик), лучше использовать статический кольцевой буфер.

Не зная максимальный размер (или если этот размер достаточно велик), стоит использовать очередь на основе списка, так как её размер ограничивает лишь доступная программе оперативная память.

6. Каковы достоинства и недостатки различных реализаций очереди в

зависимости от выполняемых над ней операций?

При использовании списка тратится больше времени на обработку операций с очередью, а так же может произойти фрагментация памяти.

При реализации статическим кольцевым буфером очередь всегда ограничена по размеру (хотя возможно и динамическое выделение памяти с последующим расширением массива). Операции в ней выполняются быстрее, нежели на списке.

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

Фрагментация памяти - разбиение памяти на куски, которые лежат не рядом друг с другом. Можно сказать, что это чередование свободных и занятых кусков памяти.

8. Для чего нужен алгоритм «близнецов».

Алгоритм «близнецов» - дисциплина выделения памяти, используется в ситуациях, когда необходимо гарантированное время реакции (например для задач реального времени).

9. Какие дисциплины выделения памяти вы знаете?

Дисциплина "самый подходящий" (best fit), по которой выделяется свободный участок, размер которого равен запрошенному или превышает его на минимальную величину.

Дисциплина "первый подходящий" (first fit), по которой выделяется первый же найденный свободный участок, размер которого не меньше запрошенного (работает быстрее, но менее эффективно выделяется память).

10. На что необходимо обратить внимание при тестировании программы?

При тестировании программы необходимо обратить внимание на переполнение очереди (при реализации очереди в виде массива) и на фрагментацию (при реализации очереди в виде списка)

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

При запросе памяти ОС находит подходящий блок памяти и записывает его в «таблицу» занятой памяти. При освобождении ОС удаляет этот блок памяти из «таблицы» занятой пользователями памяти.