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»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №5 ПО ДИСЦИПЛИНЕ: ТИПЫ И СТРУКТУРЫ ДАННЫХ

“Работа с очередью”

Студент **Зернов Георгий Павлович**

Группа **ИУ7-34Б**

Название предприятия **НУК ИУ МГТУ им. Н. Э. Баумана**

Студент _____ **Зернов Г.П.**

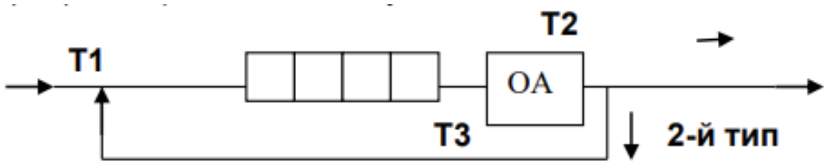
Преподаватель _____ **Никульшина Т.А.**

2024

Оглавление

Условие задачи

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 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.

Техническое задание

Реализуемая в программе задача

Цель работы: отработка навыков работы с типом данных «очередь», представленным в виде одномерного статического массива (представление динамическим массивом можно добавить по желанию) 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 объему памяти.

Входные данные

В случае выбора опции в меню целое число, соответствующее одной из указанных опций:

- 1 - Добавить число в очередь на массиве
- 2 - Удалить число из очереди на массиве
- 3 - Вывести очередь на массиве
- 4 - Добавить число в очередь на списке
- 5 - Удалить число из очереди на списке
- 6 - Вывести очередь на списке
- 7 - Освобождённые адреса
- 8 - Анализ эффективности
- 9 - Симуляция на массиве
- 10 - Симуляция на списке
- 11 - Симуляция на массиве(время)
- 12 - Симуляция на списке(время)
- 13 - Выход

В случае добавления числа в очередь:

- Добавляемый элемент – вещественное число.

Выходные данные

1. Вывести очередь на массиве – выведенная на экран очередь на массиве.
2. Вывести очередь на списке – выведенная на экран очередь на списке.
3. Освобождённые адреса – выведенные на экран освобождённые адреса.
4. Анализ эффективности – выведенные на экран результаты измерения времени работы операций push/pop для разных реализаций очереди.
5. Симуляция на массиве – выведенные на экран результаты симуляции, проведенной с помощью очереди на массиве.
6. Симуляция на списке – выведенные на экран результаты симуляции, проведенной с помощью очереди на списке.
7. Симуляция на массиве(время) – выведенное на экран время симуляции с использованием очереди на массиве.
8. Симуляция на списке(время) – выведенное на экран время симуляции с использованием очереди на списке.

Способ обращения к программе

Чтобы обратиться к программе, пользователю необходимо запустить файл app.exe.

Возможные аварийные ситуации или ошибки пользователя

- Введено невалидное значение опции – вывод на экран информирующего сообщения о неверном вводе и предоставление пользователю возможности повторно ввести значение.

- Введено невалидное значение добавляемого элемента – вывод на экран информирующего сообщения о неверном вводе и предоставление пользователю возможности повторно ввести значение.
- Попытка добавления в очередь максимальной длины элемента – сообщение об ошибке.
- Попытка удалить элемент из пустой очереди или её вывод – соответствующее сообщение об ошибке.
- Нехватка памяти при её выделении – соответствующее сообщение об ошибке и остановка текущего процесса с возвратом программы в состояние до его начала.
- Нехватка длины очереди при симуляции – соответствующее сообщение об ошибке.

Внутренние структуры и типы данных

Структура, представляющее тип данных “очередь на массиве”.

- double *front_ptr – указатель на начало очереди.
- double *back_ptr – указатель на конец очереди.
- size_t queue_size – указатель на вершину правого стека.
- double data[MAX_QUEUE_SIZE]– массив находящихся в очереди значений. MAX_QUEUE_SIZE = 10^6

Листинг структуры:

```
typedef struct
{
    double *front_ptr;
    double *back_ptr;
    size_t queue_size;
    double data[MAX_QUEUE_SIZE];
} array_queue_t;
```

Структура, представляющее тип данных “узел списка”.

- double data – значение узла.

- struct node_t* next – указатель на следующий узел.

Листинг структуры:

```
typedef struct node_t
{
    double data;
    struct node_t* next;
} node_t;
```

Структура, представляющее тип данных “очередь на списке”.

- node_t *front_ptr – указатель на начало очереди.
- node_t *back_ptr – указатель на конец очереди.
- size_t queue_size – текущая длина очереди.

Листинг структуры:

```
typedef struct
{
    node_t *front_ptr;
    node_t *back_ptr;
    size_t queue_size;
} list_queue_t;
```

Алгоритм программы

- Общий алгоритм программы:

1. Запуск основного цикла программы.
2. Вывод меню.
3. Запрос у пользователя опции для выполнения.
4. Выполнение выбранной опции.
5. Переход к новой итерации цикла или завершение основного цикла при соответствующем выборе пользователя.

- Алгоритм добавления элемента в очередь на массиве:

1. Проверка переполнения очереди.

2. При наличии достижении указателя на конец очереди конца массива – сдвиг очереди к началу массива.
2. Запись по указателю на конец очереди и его увеличение.

- Алгоритм удаления элемента из очереди на массиве:

1. Проверка очереди на пустоту.
2. Уменьшение указателя на конец очереди.

- Алгоритм добавления элемента в очередь на списке:

1. Добавление узла списка с новыми данными в его конец. Обновление указателя на конец очереди указателем на новый узел.

- Алгоритм удаления элемента из очереди на списке:

1. Проверка очереди на пустоту.
2. Разрушение узла очереди по указателю на её начало. Обновление указателя на начало.

- Алгоритм проведения симуляции:

Запуск основного цикла симуляции. Условием завершения цикла является обработка требуемого числа заявок первого типа (1000.) На каждой итерации цикла:

1) До тех пор, пока переменная, обозначающая время до прихода следующей заявки первого типа равна 0, генерация времени до прихода заявки первого типа и генерация заявки первого типа с помещением её в конец очереди.
2) До тех пор, пока переменная обозначающая время до конца обработки текущей заявки равна 0.
   а) Если очередь пуста или обработано 4 заявки первого типа подряд – генерация времени обработки заявки второго типа и помещение его

в обработчик. Увеличение счётчиков заявок второго типа и суммарного времени их обработки.

b) Иначе, если в очереди есть заявки первого типа – изъятие из очереди заявки первого типа. Увеличение счётчика заявок первого типа и суммарного времени их обработки.

c) Иначе увеличение счётчика времени простоя автомата.

3) Уменьшение времени до прихода следующей заявки первого типа и времени до конца обработки текущей заявки на 1.

После завершения цикла – вывод результатов симуляции.

Функции программы

- void create_arr_queue(array_queue_t *queue) – инициализирует очередь на массиве.
- int push_back_arr(array_queue_t *queue, double value) – добавляет элемент в очередь на массиве. Возвращает код ошибки.
- int pop_front_arr(array_queue_t *queue) – удаляет элемент из очереди на массиве. Возвращает код ошибки.
- int front_arr(array_queue_t *queue, double *value) – записывает значение начала очереди на массиве в value. Возвращает код ошибки.
- int back_arr(array_queue_t *queue, double *value) – записывает значение конца очереди на массиве в value. Возвращает код ошибки.
- void create_queue(list_queue_t *queue) – инициализирует очередь на списке.
- void destroy_queue(list_queue_t *queue) – разрушает очередь на списке.
- int push_back(list_queue_t *queue, double value) – добавляет элемент в очередь на списке. Возвращает код ошибки.
- int pop_front(list_queue_t *queue) – удаляет элемент из очереди на списке. Возвращает код ошибки.

- int front(list_queue_t *queue, double *value) – записывает значение начала очереди на списке в value. Возвращает код ошибки.
- int back(list_queue_t *queue, double *value) – записывает значение конца очереди на списке в value. Возвращает код ошибки.
- int time_measurement(void) – проводит анализ времени работы очередей.
- int simulate_arr(int verbose) – проводит симуляцию на очереди на массиве. Возвращает код ошибки.
- int simulate_list(int verbose) – проводит симуляцию на очереди на списке. Возвращает
- код ошибки.
- void print_menu(void) – выводит меню.
- int main(void) – выполняет основной алгоритм программы.

Тесты

Тестовый случай	Ожидаемый результат
Введена невалидная опция	Соответствующее сообщение об ошибке, возможность снова ввести требуемое значение.
Попытка добавить элемент в очередь при достижении максимальной длины.	Соответствующее сообщение об ошибке.
Попытка удалить элемент из очереди при отсутствии в ней элементов.	Соответствующее сообщение об ошибке.
Попытка считать значение начала или конца очереди при отсутствии в ней элементов.	Соответствующее сообщение об ошибке.
Попытка вывести очередь при отсутствии в ней элементов.	Соответствующее сообщение об ошибке.
Добавление элемента в пустую очередь Очередь: - Элемент: 42	Полученная очередь: 42

Добавление элемента в очередь Очередь: 5 4 3 2 1 Элемент: 7	Полученная очередь: 5 4 3 2 1 7
Удаление элемента из очереди, в которой единственный элемент Очередь: 55	Полученная очередь: -
Удаление элемента из очереди Очередь: 43 123 1 4 21 1	Полученная очередь: 123 1 4 21 1
Вывод очереди Очередь: 9 0 9 0 9	Выведенная очередь: 9 0 9 0 9

Анализ эффективности

Производится 10^6 операций добавления элемента, потом производятся столько же удалений элемента и считается среднее для соответствующей операции. Элементами очереди являются вещественные числа.

Очередь на массиве:

- push: 12(нс)
- pop: 8(нс)
- Размер элемента: 8 байт

Очередь на списке:

- push: 33(нс)
- pop: 25(нс)
- Размер элемента: 16 байт

Таким образом очередь на массиве в 3 раза эффективнее по времени. При размере очереди на списке до 50% от размера очереди на массиве, очередь на списке будет эффективнее по памяти.

Симуляция

Теоретический расчёт:

- Среднее время ожидания заявки первого типа: (0 + 5) / 2 = 2.5 е.в.
- Среднее время обработки заявки первого типа: (0 + 4) / 2 = 2 е.в.
- Среднее время обработки заявки второго типа: (0 + 4) / 2 = 2 е.в.

Т.к. заявка второго типа возвращается не далее 4 позиции от головы очереди, то можно считать, что они составят четверть от заявок первого типа, значит будет обработано 250 заявок второго типа и 1000 первого. (Время простоя будет равно 0 так как если в обработчике нет заявки первого типа, там есть заявка второго типа)

Среднее время работы аппарата: 1000 * 2 + 250 * 2 = 2500 е.в.

Результаты симуляции:

- Обработано 100 заявок первого типа:

  Вошло заявок первого типа: 102

  Вышло заявок первого типа: 100

  Обращений заявок второго типа: 33

  Текущая длина очереди: 2

  Средняя длина очереди: 1.849057

  Среднее время заявки: 1.992481
- Обработано 200 заявок первого типа:

  Вошло заявок первого типа: 223

  Вышло заявок первого типа: 200

  Обращений заявок второго типа: 59

  Текущая длина очереди: 23

  Средняя длина очереди: 8.525520

  Среднее время заявки: 2.042471
- Обработано 300 заявок первого типа:

  Вошло заявок первого типа: 318

Вышло заявок первого типа: 300

Обращений заявок второго типа: 84

Текущая длина очереди: 18

Средняя длина очереди: 12.651223

Среднее время заявки: 2.023438

- Обработано 400 заявок первого типа:

  Вошло заявок первого типа: 414

  Вышло заявок первого типа: 400

  Обращений заявок второго типа: 109

  Текущая длина очереди: 14

  Средняя длина очереди: 13.364956

  Среднее время заявки: 2.029470

- Обработано 500 заявок первого типа:

  Вошло заявок первого типа: 508

  Вышло заявок первого типа: 500

  Обращений заявок второго типа: 134

  Текущая длина очереди: 8

  Средняя длина очереди: 13.276545

  Среднее время заявки: 1.990536

- Обработано 600 заявок первого типа:

  Вошло заявок первого типа: 618

  Вышло заявок первого типа: 600

  Обращений заявок второго типа: 159

  Текущая длина очереди: 18

  Средняя длина очереди: 13.111402

  Среднее время заявки: 2.010540

- Обработано 700 заявок первого типа:

  Вошло заявок первого типа: 721

Вышло заявок первого типа: 700

Обращений заявок второго типа: 184

Текущая длина очереди: 21

Средняя длина очереди: 13.817873

Среднее время заявки: 2.000000

- Обработано 800 заявок первого типа:

  Вошло заявок первого типа: 823

  Вышло заявок первого типа: 800

  Обращений заявок второго типа: 209

  Текущая длина очереди: 23

  Средняя длина очереди: 15.168071

  Среднее время заявки: 1.999009

- Обработано 900 заявок первого типа:

  Вошло заявок первого типа: 921

  Вышло заявок первого типа: 900

  Обращений заявок второго типа: 234

  Текущая длина очереди: 21

  Средняя длина очереди: 15.829300

  Среднее время заявки: 2.004409

Время моделирования: 2519

Время простоя: 0

Вошло заявок первого типа: 1021

Вышло заявок первого типа: 1000

Обращений заявок второго типа: 258

Погрешность: 0.760000

Заметим, что симуляция с высокой точностью подтверждает данные теоретического расчёта.

Время и память симуляции:

Очередь на массиве:

- Время: 48402 (нс)
- Память: 8000 (байт)

Очередь на списке:

- Время: 60350
- Память: 392 (байт)

Заметим, что реализация очереди на массиве имеет значительно меньшую разницу по времени с реализацией очереди на списке, чем в замерах времени операций, но значительно проигрывает по памяти.

Вывод

Реализация очереди на массиве, теоретически значительно эффективнее по времени, чем реализация на списке. Разница в эффективности по памяти напрямую зависит от размера элемента очереди – чем меньше размер элемента, тем при меньшем проценте заполнения реализация на массиве будет эффективнее. На практике эффективность значительно зависит от условий применения. Разница во времени между реализациями будет с высокой вероятностью меньше, так как в реальной программе операции производятся не только с очередью (при этом время работы программы с реализацией очереди на списке может быть даже меньше, если частоты добавлений и удалений примерно равны, и элементы очереди приходится часто сдвигать к началу массива). Но разница по памяти может быть значительна, во-первых, реализация очереди на списке не может переполниться (только в случае нехватки памяти самой программа), а во-вторых реализация очереди на списке может занимать значительно меньше памяти, если заполненность очереди невелика.

Контрольные вопросы:

Что такое FIFO и LIFO?

- FIFO (first in, first out): «первым пришёл — первым ушёл».
- LIFO (last-in, first-out): «последним пришёл — первым ушёл».

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

При хранении очереди на массиве память выделяется статически, в объёме достаточном для хранения полностью заполненной очереди. При хранении очереди на списке память выделяется динамически при добавлении элемента в очередь.

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

При реализации очереди на массиве память не освобождается напрямую, доступные для записи ячейки определяются указателями на начало и конец. При удалении элемента указатель на начало смещается, освобождая ячейку для записи. При реализации очереди на списке память освобождается при удалении элемента из очереди путём разрушения области памяти, выделенной под удаляемый узел.

Что происходит с элементами очереди при ее просмотре?

Считать можно только первый или последний элемент очереди, при просмотре очереди элементы из неё удаляются.

От чего зависит эффективность физической реализации очереди?

Эффективность физической реализации зависит от алгоритмов добавления и удаления из очереди и затрачиваемой на элемент памяти.

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

Очередь на массиве:

- Достоинства: Быстрые операции добавления и удаления элемента. Минимальные затраты памяти в пересчёте на элемент.
- Недостатки: Большой объём постоянно занимаемой памяти. Заранее определённая максимальная длина очереди.

Очередь на списке:

- Достоинства: Оптимальный расход памяти при низком проценте заполнения очереди.
- Недостатки: Более медленные операции добавления и удаления элемента. Большие затраты памяти при одинаковой длине.

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

Фрагментация памяти — это наличие большого числа несмежных участков свободной памяти очень маленького размера (фрагментов). Фрагментация возникает только в куче.

Для чего нужен алгоритм «близнецов».

Алгоритм близнецов (или метод двойников) — алгоритм динамического распределения памяти, при котором память поделена на блоки размером 2^n и при запросе выделяется блок, размер которого максимально приближен к требуемой памяти. Нужен для уменьшения фрагментации, при его использовании не возникает внешней фрагментации.

Какие дисциплины выделения памяти вы знаете?

Best fit (самый подходящий): Выделяется тот свободный участок, размер которого равен запрошенному или превышает его на минимальную величину. First fit (Первый подходящий: Выделяется первый же найденный свободный участок, размер которого не меньше запрошенного.

На что необходимо обратить внимание при тестировании программы?

При тестировании программы следует обратить внимание на:

- Корректность работы программы при переполнении очереди.
- Корректность работы с памятью.
- Разницу между ожидаемым и полученным временем симуляции (не более 2-3%).

Каким образом физически выделяется и освобождается память при динамических запросах?

При выделении памяти адреса помечаются как занятые, при освобождении – наоборот.