|  |  |
| --- | --- |
| **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

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №5**

**«ОБРАБОТКА ОЧЕРЕДЕЙ»**

Студент Светличная Алина Алексеевна

Группа ИУ7 – 33Б

Проверил Апальков Федор Станиславович

*2021 г.*

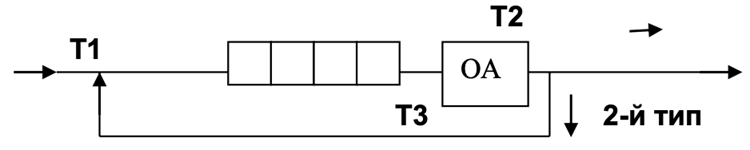
# **Описание технического задания**

Условие задачи:

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.

Техническое задние:

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 Т1, 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 Т2 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 Т3 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.

Пункты меню:

1. Добавить элементы в очередь

2. Добавить случайные элементы в очередь

3. Удалить элементы из очереди

4. Вывести текущее состояние очереди (массив)

5. Вывести текущее состояние очереди (список)

6. Вывести массив освобождённых адресов

0. Выйти из программы

Входные данные:

* Способ реализации очереди (массивом или списком)
* Номер пункта меню
* Входные данные зависят от пункта меню

Выходные данные:

Выходные данные зависят от пункта меню

Действие программы:

* Добавление элемента в очередь
* Удаление элемента из очереди
* Сравнение реализаций по времени и памяти

Обращение к программе:

Запускается через терминал командой ./main.exe в директории с программой.

Возможные аварийные ситуации:

1. Некорректный пункт меню
2. Некорректные символы при вводе
3. Отрицательное количество элементов в пунктах меню
4. Переполнение очереди при добавлении
5. Выход за нижнюю границу очереди при удалении
6. Невыделенная память

# **Описание структур данных**

**general\_action** – переменная для выбора реализации стека

**menu\_action** – переменная для выбора пункта меню

Поля структуры, описанной ниже, для реализации очереди с помощью массива:

\*head – указатель на голову очереди

\*tail – указатель на хвост очереди

size – количество заполненных элементов очереди

**typedef struct turn\_array**

**{**

**int \*head;**

**int \*tail;**

**int size;**

**} turn\_array\_t**

Поля структуры, описанной ниже, для реализации очереди с помощью списка:

\*head – указатель на голову очереди

size – количество заполненных элементов очереди

**typedef struct turn\_list**

**{**

**elem\_turn\_list\_t \*head;**

**int size;**

**} turn\_list\_t**

Поля структуры, описанной ниже, для хранения свободных областей очереди:

\*\*array – динамическая матрица для хранения адресов свободных областей очереди

size – количество свободных областей очереди

**typedef struct array\_freed\_areas**

**{**

**elem\_turn\_list\_t \*\*array;**

**int size;**

**} array\_freed\_areas\_t**

Поля структуры, описанной ниже, для хранения элементов очереди-списка:

elem – элемент очереди

\*next – указатель на следующий элемент очереди

**typedef struct elem\_turn\_list**

**{**

**int elem;**

**struct elem\_turn\_list \*next;**

**} elem\_turn\_list\_t**

# **Описание алгоритма**

1. Вводится способ реализации очереди
2. Вводится пункт меню, отвечающий за определённое действие;
3. Ввод осуществляется до того момента, пока не будет введён 0 – признак выхода из программы.

Очередь-массив: указатель на начало массива, который передвигается вперед или назад при добавлении и удалении соответственно (также увеличивается или уменьшается количество элементов).

Очередь-список: происходит работа с указателем на следующий элемент, который при добавлении сначала инициализируется, а потом двигается, а при удалении сначала очищается, а потом двигается.

**Тесты**

|  |  |  |
| --- | --- | --- |
| Аварийная ситуация | Код ошибки | Уведомление |
| Тип очереди - k | ERROR\_SYMBOLS | Встречены некорректные символы |
| Тип стека - 5 | ERROR\_GENERAL\_TYPE | Некорректный номер команды |
| Пункт меню - № | ERROR\_SYMBOLS | Встречены некорректные символы |
| Пункт меню - -1 | ERROR\_MENU\_TYPE | Некорректный номер команды |
| При добавлении элемента в стек: | | |
| Размер стека - 1000 | ERROR\_OVERLOW | Очередь переполнена |
| Количество добавляемых элементов - we | ERROR\_SYMBOLS | Встречены некорректные символы |
| Количество добавляемых элементов - -3 | ERROR\_NUM\_ADD | Количество добавляемых элементов не может быть отрицательным |
| Количество добавляемых элементов - 1001 | ERROR\_NUM\_ADD | В результате добавления такого числа элементов произойдет переполнение очереди |
| Добавляемый элемент - \n | ERROR\_SYMBOLS | Встречены некорректные символы |
| При удалении элемента из стека: | | |
| Размер очереди - 0 | ERROR\_ EMPTY | Очередь пуста |
| Количество удаляемых элементов - we | ERROR\_SYMBOLS | Встречены некорректные символы |
| Количество удаляемых элементов - -3 | ERROR\_NUM\_REMOVE | Количество удаляемых элементов не может быть отрицательным |
| Количество удаляемых элементов - 1001 | ERROR\_NUM\_ REMOVE | В результате удаления такого числа элементов произойдет выход за нижнюю границу очереди |

**Обработка результатов**

**Методика исследования:** программно проводится 100 замеров одного действия для определения среднего значения выполнения действия.

При добавлении элемента: массивом тратится время только на создание элемента (для этого пункта написана отдельная функция добавления рандомных элементов, чтобы исключить человеческую погрешность), передвижение указателя и увеличение размерности очереди. Списком тратится время еще и на *выделение памяти под каждый элемент* (остальные этапы остаются такие же).

При удалении элемента: массивом тратится время на *полный кольцевой сдвиг*, на передвижение указателя и уменьшение размерности очереди. Списком тратится время только на освобождение памяти из-под каждого элемента и передвижение указателя.

**Добавление элементов (в тактах)**

|  |  |  |  |
| --- | --- | --- | --- |
| Размер | Массив | Список | Отношение списка к массиву (в разах) |
| 10 | 941 | 4229 | 4,5 |
| 100 | 5429 | 21358 | 3,9 |
| 500 | 28755 | 230384 | 8,0 |
| 1000 | 54610 | 199408 | 3,7 |

**Удаление элементов (в тактах)**

|  |  |  |  |
| --- | --- | --- | --- |
| Размер | Массив | Список | Отношение массива к списку (в разах) |
| 10 | 484 | 2263 | 0,2 |
| 100 | 43470 | 21139 | 2,1 |
| 500 | 1064017 | 51057 | 20,8 |
| 1000 | 4165910 | 101344 | 41,1 |

**Сравнение памяти (в байтах)**

|  |  |  |  |
| --- | --- | --- | --- |
| Размер | Массив | Список | Отношение массива к списку (в разах) |
| 10 | 4024 | 168 | 23 |
| 100 | 4024 | 1608 | 2,5 |
| 500 | 4024 | 8008 | 0,5 |
| 1000 | 4024 | 16008 | 0,3 |

# **Тесты**

Интервал времени прихода в очередь заявки 1-го типа (T1): от 0 до 5 е.в.;

Интервал времени обработки заявки 1-го типа в очереди: от 0 до 4 е.в.;

Интервал времени обработки заявки 2-го типа в очереди: от 0 до 4 е.в.

Количество вошедших заявок в очередь 1-го типа – 1001;

Количество вышедших заявок из очереди – 2214.

|  |  |  |  |
| --- | --- | --- | --- |
| Тип сравнения | Ожидаемое время | Реальное время | Погрешность |
| Время моделирования (по входу) | 2502 | 2521 | 0.75% |
| Время работы аппарата | 4428 | 4455 | 0.61% |

Интервал времени прихода в очередь заявки 1-го типа (T1): от 0 до 4 е.в.;

Интервал времени обработки заявки 1-го типа в очереди: от 0 до 5 е.в.;

Интервал времени обработки заявки 2-го типа в очереди: от 0 до 5 е.в.;

Количество вошедших заявок в очередь 1-го типа – 1249;

Количество вышедших заявок из очереди – 1257.

|  |  |  |  |
| --- | --- | --- | --- |
| Тип сравнения | Ожидаемое время | Реальное время | Погрешность |
| Время моделирования | 2498 | 25010 | 0.50% |
| Время работы аппарата | 3142 | 3128 | 0.50% |

**Контрольные вопросы**

1. Что такое очередь?

Очередь – это последовательный список переменной длины, включение в

который идет с одной стороны (с хвоста), а исключение – с другой стороны (с головы). Принцип работы очереди: первым пришел – первым вышел, то есть, First In – First Out (FIFO).

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

При хранении статическим массивом: количество элементов \* размер одного элемента очереди.

Память выделяется на стеке при компиляции – массив статический;

Память выделяется в куче – массив динамический.

При хранении списком: количество элементов \* (размер одного элемента очереди + указатель на следующий элемент).

Память выделяется в куче для каждого элемента отдельно.

3. Каким образом освобождается память при удалении элемента из очереди при ее различной̆ реализации?

При хранении статическим массивом память не освобождается, а лишь меняется указатель конца очереди. При хранении списком, освобождается память из-под удаляемого элемента.

4. Что происходит с элементами очереди при его просмотре?

Классическая реализация очереди предполагает, что просмотреть содержимое очереди без извлечения (удаления) ее элементов невозможно.

5. Каким образом эффективнее реализовывать очередь? От чего это зависит?

Основной причиной выбора типа очереди является размерность очереди, если размер очереди известен заранее, то статический массив будет сочетанием и быстро действенности (если массив будет кольцевым, то удаление тоже будет проходить быстрее) и затрат по памяти, но если же размер очереди абсолютно неизвестен, то статический массив обязывает нас выделять память с большим излишком (до 16 раз).

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

При использовании кольцевого массива тратится больше времени на удаление элементов из очереди, а так же может возникнуть фрагментация памяти. При реализации статическим массивом, очередь всегда ограничена по размеру, однако в малых количествах такое выделение памяти может быть оптимальнее.

Список не ограничен по памяти, но более затратен по памяти (на очередях от 200 элементов). А также медленнее по добавлению элементов.

7. Что такое фрагментация памяти?

Фрагментация памяти – разбиение памяти на участки, которые лежат не рядом друг с другом, то есть происходит чередование свободных и занятых участков памяти.

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

Корректное освобождение памяти.

9. Каким образом физически выделяется и освобождается память при

**динамических запросах?**

При запросе памяти операционная система находит подходящий блок памяти и записывает его в начало массива-таблицы занятой памяти. При освобождении операционная система удаляет этот блок памяти из массива-таблицы занятой пользователем памяти.

# **Вывод**

Если очередь реализована статическим массивом, то добавление в неё новых элементов будет происходить быстрее (до 8 раз). Это связанно с тем, что для хранения очереди в виде списка каждый раз требуется выделить память для указателя на следующий элементы списка.

Удаление элементов из массива происходит медленнее (до 40 раз), так как при удалении элемента из массива, требуется произвести сдвиг всех оставшихся элементов. Для того чтобы удаление элементов из массива тоже было эффективнее по времени, необходимо использовать кольцевой массив.

Однако вариант хранения очереди в виде списка выигрывает пред статическим массивом по занимаемой памяти, также стоит использовать очередь-список, если не известен даже приблизительный размер стека.