**Типы и структуры данных.**

**Лабораторная работа №4**

# Обработка очередей

**Выполнил**: Кузнецов Александр

**Группа**: ИУ7-33

**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.

Требуется смоделировать процесс обслуживания первых 1000 заявок первого типа, выдавая после обслуживания каждых 100 заявок первого типа информацию о текущей и средней длине каждой очереди и о среднем времени пребывания заявок каждого типа в очереди. В конце процесса необходимо выдать на экран общее время моделирования, время простоя ОА, количество вошедших в систему и вышедших из нее заявок первого и второго типов.

**Вариант 13 (задача 6)**

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

# T1 T3

OA

# T2

# T4

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 немедленно поступает на обслуживание; обработка заявки 2-го типа прерывается и она возвращается в "хвост" своей очереди (система с **абсолютным** приоритетом и **повторным** обслуживанием).

Смоделировать процесс обслуживания первых 1000 заявок **1-го типа**, выдавая после обслуживания каждых 100 заявок **1-го типа** информацию о текущей и средней длине каждой очереди, а в конце процесса - общее время моделирования и количестве вошедших в систему и вышедших из нее заявок обоих типов, среднем времени пребывания заявок в очереди, количестве «выброшенных» заявок второго типа. Обеспечить по требованию пользователя выдачу на экран адресов элементов очереди при удалении и добавлении элементов. Проследить, возникает ли при этом фрагментация памяти.

**Входные данные:**

Номер выполнения команды;

**Пример:**

//ввод команды для очереди

>>1

**Выходные данные**

Время работы очереди 1 типа, 2 типа;

Количество обработанных заявок;

Адреса;

**Функция программы**

Обработка очередей, измерение времени выполнения массива и списка, вывод адресов списка в файл.

**Аварийные ситуации**

**-** Закончилось место в оперативной памяти

**-** Переполнение очереди

**Описание использованных структур данных**

//структура (очередь массива)

typedef struct{

double data[2048];

int size;

}QueueArray;

//узел списка

struct Node{

double data;

struct Node \*next;

};

//структура(очередь списка)

typedef struct{

struct Node \*head, \*tail;

int size;

}List;

**Алгоритм**

**Обработка ввода:**

1. Вывести интерфейс программы
2. Проверить корректность ввода(Интерфейс)
3. Запустить выбранную команду

-//////- Вывод сообщение об ошибке, если она есть -//////-

Иначе

**Обработка операций над очередью:**

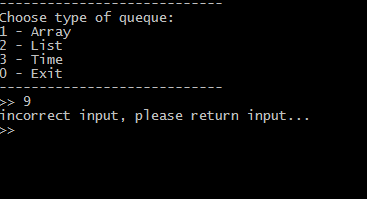
1. Добавление элемента в очередь
2. Удаление элемента из очереди
3. Расчет времени
4. Вывод времени выполнения, через каждые 100 элементов
5. Вывод конечного результата выполнения
6. Вывод таблицы адресов списка в файл

**Обработка времени:**

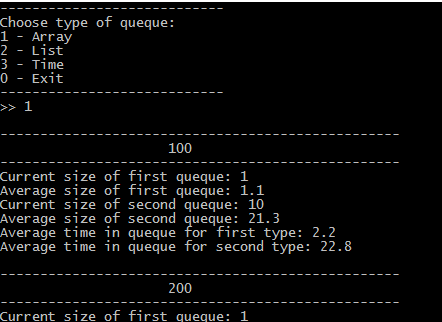
1. Расчет среднего времени для первой очереди
2. Расчет среднего времени для второй очереди
3. Вывод времени работы всей программы

**Тесты**

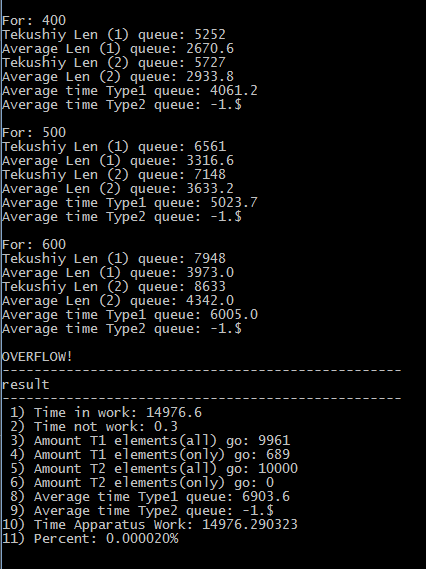
**Тест ввода:**Некорректный ввод команды очереди:



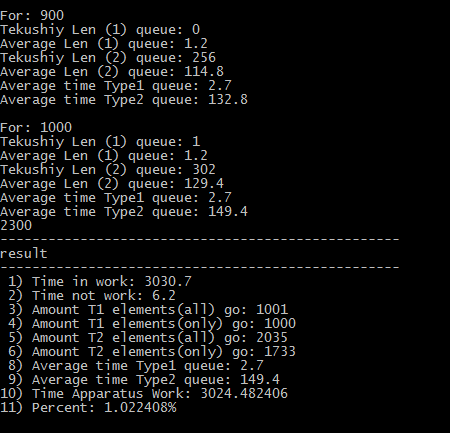
Корректный ввод номера операции над таблицей:



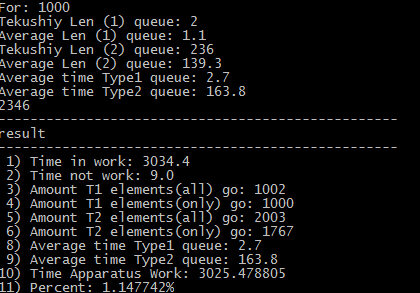
Тест на переполнение



**Результат:**

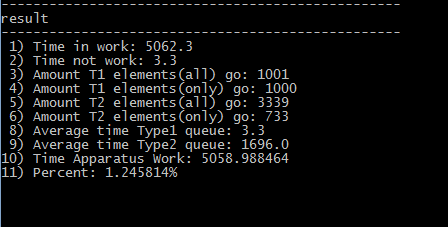
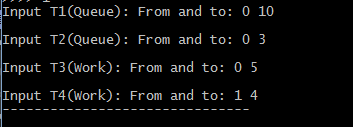


100\*(3030.7 – ((5+1)/2)\*1000)/(((5+1)/2)\*1000) = 1.0224%



100\*(3034.4 – ((5+1)/2)\*1000)/(((5+1)/2)\*1000) = 1.1477%

В теории (5+1)/2 \* 1000 = 3000

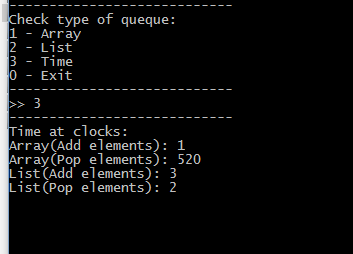


100\*(5058.98 – ((10+0)/2)\*1000)/(((10+0)/2)\*1000) = 1.2458%

В теории (10+0)/2 \* 1000 = 5000

**Сравнение методов**

По результатам выполнения программы, видно, что реализация очереди списком дает прирост по времени работы при 10000 элементов, т.к. в массиве при удалении происходит сдвиг всего массива на 1 позицию влево.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N(элем) | Arr\_pop(ms) | Arr\_add(ms) | List\_pop(ms) | List\_arr(ms) |
| 10000 | 520 | 1 | 3 | 2 |

Отсюда видно, что

**Эффективность по памяти**

При реализации очереди с помощью списка, на каждый узел выделяется дополнительная память в виде указателя на предыдущий элемент, но при этом не выделяется лишняя память.

При реализации очереди с помощью массива, в самой структуре не нужно выделять лишнюю переменную, но при такой реализации выделяется очень много памяти, которая может не пригодиться.

**Вопрос – Ответ**

**1. Что такое очередь?**

Очередь – последовательной список переменной длины. Включение элементов идёт с «хвоста» списка, исключение – с «головы» списка. Принцип работы: первым пришёл – первым вышел, FirstInFirstOut.

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

При реализации очереди списком, под каждый новый элемент выделяется память из кучи, элементы связываются указателями.

При реализации очереди массивом, выделяется блок памяти из N\*sizeof(element) байт, где N–максимальное количество элементов в очереди, элементы следуют друг за другом последовательно.

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

При реализации очереди списком, головной элемент считывается, указатель на «голову» очереди переходит на следующий элемент, считанный элемент удаляется.

При реализации очереди массивом, головной элемент считывается, остальные элементы массива сдвигаются на 1 – длина очереди уменьшается на 1, элемент [1] массива «затирает» головной элемент [0].

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

При просмотре очереди, головной элемент из неё удаляется. Остальные элементы сдвигаются (массив), либо указатель на начало передвигается на следующий элемент (список).

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

При реализации очереди списком, проще всего добавлять и удалять из неё элементы, однако может возникнуть фрагментация памяти. При реализации очереди массивом дефрагментации не возникает, однако может возникнуть переполнение памяти, а добавление и удаление элементов сложнее. Способ реализации зависит от того, в чем мы сильнее ограничены – в памяти или во времени выполнения операций.

**6. В каком случае лучше реализовать очередь посредством указателей, а в каком – массивом?**

Очередь лучше реализовывать с помощью указателей, если новые элементы в среднем появляются реже, чем происходит полное очищение очереди – в общем случае фрагментация не возникает. Реализация с помощью указателей применима если требуется строгий контроль фрагментации.

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

При реализации очереди массивом не возникает фрагментации памяти, однако может произойти переполнение очереди, а также затрачивается дополнительное время на сдвиг элементов. Сдвига можно избежать, если использовать кольцевой массив, однако при этом усложняются операции добавления и удаления элементов. Наконец, при реализации списком проще всего реализуются алгоритмы добавления и удаления элементов, но может возникнуть фрагментация.

**8. Что такое фрагментация памяти?**

Фрагментация – чередование участков памяти при последовательных запросах на выделение и освобождение памяти. «Занятые» участки чередуются со «свободными» - однако последние могут быть недостаточно большими для того, чтобы сохранить в них нужное данное.

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

При реализации очереди списком необходимо следить за освобождением памятти при удалении элемента из очереди. Если новые элементы приходят чаще, чем удаляются старые, очередь растёт и может происходить фрагментации памяти.

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

Программа даёт запрос ОС на выделение блока памяти необходимого размера. ОС находит подходящий блок, записывает его адрес и размер в таблицу адресов, а затем возвращает данный адрес в программу.

При запросе на освобождение указанного блока программы, ОС убирает его из таблицы адресов, однако указатель на этот блок может остаться в программе. Попытка считать данные из этого блока может привести к непредвиденным последствиям, поскольку они могут быть уже изменены.