**Типы и Структуры Данных**

**Oтчет**

## *Работа № 5:* Обработка очередей

***Студент: Нгуен Фыок Санг***

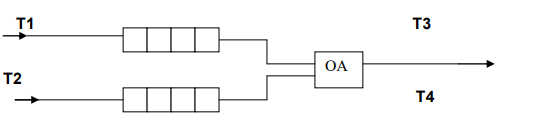
***Группa: ИУ7И-36Б***

*2019*

1. **Oписание условия задачи:**

**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.**

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 любого типа может войти в ОА, если:

♣ а) она вошла в пустую систему;

♣ б) перед ней обслуживалась заявка ее же типа;

♣ в) перед ней из ОА вышла заявка другого типа, оставив за собой пустую очередь (система с чередующимся приоритетом).

Смоделировать процесс обслуживания первых 1000 заявок 1-го типа, выдавая после обслуживания каждых 100 заявок информацию о текущей и средней длине каждой очереди, а в конце процесса - 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.

**II. Oписание ТЗ:**

1. **Исходные данные**: время T1, T2, T3, T4
2. **Результат**: проверки правильности работы системы по входу и по выходу
3. **Задачи, реализуемой программой**:
   * включение элементов
   * исключение

**III. Описание СД:**

1. **Для ввода:** использовать 4 матрицы 1\*2 для хранения времени
2. **Для обработки:** Использовать 2 связанных списка (2 очереди)

***структура узла:***

struct node

{

int type;

float tin;

float t1;

float t2;

float tout;

struct node \*next;

};

***структура очереди***:

struct queue

{

node\_t \*front;

node\_t \*rear;

};

**IV. Oписание алгоритма:**

1. ***Алгоритм вставки (push) node (P)***
   1. Если стек (Q) **заполнен**, появляется сообщение об ошибке
   2. Если стек **пуст**, Q^front 🡨 P; Q^rear 🡨 P;
   3. (Q^rear)^next 🡨 P;
2. ***Aлгоритм отключения (pop)***
   1. Если стек **пуст**, появляется сообщение об ошибке
   2. P 🡨 Q^front;
   3. Q^front 🡨 P^next;
   4. P^next 🡨 Nil;
   5. Возвращает узел (P)

**V. Набор тестов:**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| N = | 100 | 300 | 500 | 700 | 1000 | 3000 | 5000 | 7000 | 10000 | 20000 | 30000 |
| Time1 | 14 | 22 | 37 | 51 | 71 | 210 | 371 | 512 | 696 | 1399 | 2110 |
| Time2 | 7 | 21 | 38 | 52 | 72 | 209 | 343 | 491 | 697 | 1444 | 2117 |
| Aver1 | 0.14 | 0.073 | 0.074 | 0.073 | 0.071 | 0.070 | 0.074 | 0.073 | 0.070 | 0.070 | 0.070 |
| Aver2 | 0.07 | 0.070 | 0.076 | 0.074 | 0.072 | 0.070 | 0.067 | 0.070 | 0.070 | 0.072 | 0.071 |

\*Time1: Время для списка \*Time2: Время для массива

**VI. Ответы на вопросы:**

1. ***Что такое очередь?*** 
   * Очередь – это последовательный список переменной длины, включение элементов в который идет с одной стороны (с «хвоста»), а исключение – с другой стороны (с «головы»). Принцип работы очереди: первым пришел – первым вышел, т. е. First In – First Out (FIFO).
2. ***Каким образом, и какой объем памяти выделяется под хранение очереди при различной ее реализации?*** 
   * Реализация очереди в виде **массива**: При моделировании простейшей линейной очереди на основе одномерного массива выделяется последовательная область памяти из m мест по L байт, где L – размер поля данных для одного элемента размещаемого типа
   * Реализация очереди в виде **линейного списка**: при реализации очереди на основе односвязного линейного списка, каждый элемент которого содержит информационное поле и поле с указателем «вперед» (на следующий элемент). В этом случае в статической памяти можно либо хранить адрес начала и конца очереди, либо – адрес начала очереди и количество элементов.
3. ***Каким образом освобождается память при удалении элемента из очереди при ее различной реализации?***
   * Очереди в виде списка
     + Освободить память для содержимого узла (если применимо).
     + Освободить память для узла
   * Очереди в виде массива
     + переместить указатель (rear) на предыдущий элемент

(rear 🡨 rear – 1)

1. ***Каким образом эффективнее реализовывать очередь. От чего это зависит?*** 
   * Сохранить очереди в виде списка
     + Узлы находятся в разных областях памяти
     + Удалить и вставить быстрее
   * Сохранить очереди в виде массива
     + Простая реализация
     + меньше памяти и времени
2. ***В каком случае лучше реализовать очередь посредством указателей, а в каком – массивом?*** 
   * Связанный список подходит для изменчивого списка, то есть мы часто добавляем и удаляем элементы.
   * Массив при обычном доступе к элементу (поиск, ...)
3. ***Каковы достоинства и недостатки различных реализаций очереди в зависимости от выполняемых над ней операций?*** 
   * Сохранить очереди в виде массива
     + Требует определения количества элементов в массиве.
     + Неэффективное использование памяти
     + Время для вставки или удаления элемента пропорционально количеству элементов после позиции.
     + Доступ к элементу списка занимает только одну постоянную времени
   * Сохранить очереди в виде списка
     + Подходит для колебаний списка, список может быть пустым или большим, в зависимости от максимальной памяти устройства. Тем не менее, это занимает больше памяти для указателей (next).
     + Время для вставки или удаления элемента занимает только постоянную времени.
     + Доступ к элементу списка, должнo найти с начала списка
4. ***Что такое фрагментация памяти?***
   * *фрагментация памяти* - Наличие в ОЗУ большого числа коротких несмежных блоков, не позволяющее удовлетворить запрос на выделение блока большего размера
5. ***Каким образом физически выделяется и освобождается память при динамических запросах?***
   * Во многих ОС существуют инструменты для работы с динамической памятью – набором «кусочков» памяти разного размера, устроенным по типу кучи. Если программа запрашивает память из кучи, и запрос оказывается успешным — ячейка резервируется за программой. Если программа запрашивает память из кучи, все свободные ячейки крупнее запрашиваемой, то менеджер имеет право разделить ячейку (из свойств кучи).
   * А может выделить и ячейку больше. Если программа запрашивает определённый объём памяти, он есть в куче, но не в виде одного кусочка, то менеджер имеет право перетасовать ячейки (для программы, как для внешнего наблюдателя, ничего не меняется) и объединить (из свойств кучи). Если программа освобождает память, то память «возвращается в кучу» – становится свободной для выделения менеджером другой программе.