Типы и Структуры Данных

Лабораторная работа №5

*Работа с очередью*

Вариант 7

Выполнил студент группы: ИУ7-36

Гасанзаде Мухаммедали

2018

1. **Задание:**

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. Реализовать очередь:

а) массивом;

б) списком.

1. **ТЗ:**

Система массового обслуживания состоит из обслуживающего аппарата (ОА) и очереди заявок. T1 T2 Заявки поступают в "хвост" очереди по случайному закону с интервалом времени Т1, равномерно распределенным от 0 до 6 единиц времени (е.в.). В ОА они поступают из "головы" очереди по одной и обслуживаются также равновероятно за время Т2 от 0 до 1 е.в., Каждая заявка после ОА вновь поступает в "хвост" очереди, совершая всего 5 циклов обслуживания, после чего покидает систему. (Все времена – вещественного типа) В начале процесса в системе заявок нет. Смоделировать процесс обслуживания до ухода из системы первых 1000 заявок, выдавая после обслуживания каждых 100 заявок информацию о текущей и средней длине очереди, а в конце процесса - общее время моделирования и количестве вошедших в систему и вышедших из нее заявок, количестве срабатываний ОА, время простоя аппарата. По требованию пользователя выдать на экран адресов элементов очереди при удалении и добавлении элементов.

**Входные данные**

Требуется выбрать реализацию очереди (массив или список). В случае массива также ввести максимальное количество элементов. Далее нужно ввести интервал времени появления новой заявки. Далее ввести интервал времени на обработку заявки ОА. Далее требуемое количество итераций для каждой заявки. В случае списка после выполнения программы можно проверить возникновение фрагментации памяти путем добавления и удаления элементов из очереди.

**Обработка ошибок**

Ввод некорректного пункта меню, некорректный ввод интервалов или размера массива, ошибки памяти, переполнение очереди.

1. **СД:**

**Структура для заявки:**

struct request {

float proc\_time;

int iteration; };

**Структура для массива:**

struct array\_order {

struct request \*pleft;

struct request \*pright;

struct request \*pin;

struct request \*pout;

int len; };

**Структура для списка:**

struct list\_member {

struct request request;

struct list\_member \*next; };

struct list\_order {

struct list\_member \*first;

struct list\_member \*last;

int len; };

1. **Алгоритм**

Считать тип очереди

Если тип очереди – массив:

считать максимальное количество элементов

Считать интервал появления заявки

Считать интервал обработки заявки

Считать количество итераций для заявки

Пока (количество вышедших заявок < 1000):

Если (ОА не работает или время до новой заявки < время

до окончания работы ОА)

Время окончания работы ОА -= время до новой заявки

Общее время += время до новой заявки

Сгенерировать новую заявку

Добавить в очередь новую заявку

Время до новой заявки = сгенерировать время

Если (ОА не работает)

Извлечь из очереди заявку

Время окончания работы ОА = время обработки

заявки

Иначе:

Время до новой заявки -= время окончания работы ОА

Общее время += время окончания работы ОА

Если (заявка не прошла все итерации)

Добавить заявку в очередь

Иначе

Количество вышедших заявок += 1

Извлечь из очереди новую заявку

Время окончания работы ОА = время обработки

заявки

Вывести все метрики модулирования

Вывести затраченное время

Вывести требуемую память

Если (реализация списком)

Предложить проверить фрагментацию

1. **Тесты:**

**Неправильный ввод пункта меню**

Ввод: -1

Вывод: Wrong input, try again

**Неправильный ввод размерности массива**

Ввод: 0

Вывод: Wrong input, try again

**Неправильный ввод интервала времени**

Ввод: 7 -1

Вывод: Wrong input, try again

**Добавление элемента в полностью заполненную очередь**

Вывод:

WARNING!!!

The queue is full

New element was not added to the queue

**Удаление элемента из пустого стека**

Вывод: Stack is empty

1. **Вывод**

Если скорость поступления заявок меньше скорости их обработки, или даже равна, но мы уверены, на каком значении стабилизируется длина очереди, то выгоднее использовать массив, как по памяти, так и по времени исполнения. Но данный подход отличается ненадежностью, так как даже незначительное отступление от этих условий влечет за собой проблемы в виде переполнения массива и необходимости его реаллоцирования, либо же в выделении большого и заведомо достаточного массива. Тогда все плюсы этой реализации могут превратиться в минусы, а программа станет менее эффективной в сравнении с реализацией списком.

А плюсом реализации списком как раз и является её гибкость к исходным данным. Можно быть уверенным в корректной и стабильной работе программы в любой ситуации, пусть и немного менее эффективной в некоторых случаях.

**7.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).

1. Каким образом и сколько памяти выделяется под хранение очереди при различной её реализации?

Очередь может быть реализована на основе динамического списка, её размер не безграничен, он ограничен объемом доступной программе оперативной памяти. В таком случае на каждый элемент очереди выделяется память для хранения самих данных и указателя на предыдущий элемент. При реализации в виде массива заранее выделяется некоторый объем памяти для хранения самих элементов, также требуется хранить адрес нижней границы очереди, адрес верхней границы стека, текущие указатели добавления и удаления.

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

При реализации списком освобождается память под каждый удаляемый элемент непосредственно. При реализации статическим массивом программист вручную не освобождает память, она освобождается после окончания этого блока программы. При динамическом массиве можно его реаллоцировать, если некоторые элементы удалились, но на это затрачивается много времени.

4. Что происходит с элементами очереди при её просмотре?

Классическая реализация очереди предполагает, что просмотр элемента очереди происходит только во время его извлечения (удаления).

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

Для «хвоста» и «головы» очереди используют соответственно два указателя Pin и Pout,. то есть, исключается элемент с адресом Pout и включается элемент по адресу Pin. Моделировать простейшую линейную очередь можно как на основе вектора (одномерного массива), так и на основе динамического списка.

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

Если скорость поступления заявок меньше скорости их обработки, или даже равна, но мы уверены, на каком значении стабилизируется длина очереди, то выгоднее использовать массив, как по памяти, так и по времени исполнения. Но данный подход отличается ненадежностью, так как даже незначительное отступление от этих условий влечет за собой проблемы в виде переполнения массива и необходимости его реаллоцирования, либо же в выделении большого и заведомо достаточного массива. Тогда все плюсы этой реализации могут превратиться в минусы, а программа станет менее эффективной в сравнении с реализацией списком.

А плюсом реализации списком как раз и является её гибкость к исходным данным. Можно быть уверенным в корректной и стабильной работе программы в любой ситуации, пусть и немного менее эффективной в некоторых случаях.

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

При добавлении элемента в реализации массивом нам нужно постоянно отслеживать возможное его переполнение. В реализации списком этого делать не надо, но там надо каждый раз очищать память при удалении элемента, чего не надо делать в массиве.

8. Что такое фрагментация памяти?

В процессе моделирования очереди может оказаться, что при последовательных запросах на выделение и освобождении памяти под очередной элемент выделяется не та память, которая была только что освобождена при удалении элемента. Т.е. участки свободной и занятой памяти могут чередоваться, что и называют фрагментацией памяти.