

Отчет по 5 лабораторной работе
По дисциплине «Типы и структуры данных»

Подготовил Пересторонин Павел
Группа ИУ7-33Б
Вариант 15

Цель работы

Отработка навыков работы с типом данных «очередь», представленным в виде одномерного массива 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.

Техническое задание

Заявки 1-го типа поступают в "хвост" очереди по случайному закону с интервалом времени T_1 , 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 T_2 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 T_3 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.

Входные данные

Отсутствуют.

Выходные данные

Статистика работы автомата (время моделирования, время простоя, время работы), статистика очередей (средняя длина, среднее время в очереди, кол-во потерянных заявок), а так же информация о фрагментации.

Возможные аварийные ситуации

Задание некорректных значений обработки (в коде).

Структуры данных

Структура элемента очереди:

```
typedef struct elem_t
{
    int value;
    double income_time;
} elem_t;
```

income_time — время прихода в очередь (нужно для измерения среднего времени заявки в очереди)

value — тип заявки (некоторое численное значение (0 или 1 в конкретно моем случае))

Структура узла очереди-списка:

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

data — значение элемента очереди

next — указатель на следующий элемент очереди

Структура очереди-списка:

```
typedef struct
{
    node_t *pin;
    node_t *pout;
    int size;
} queue_l;
```

pin — указатель на хвост

pout — указатель на голову

size — текущий размер очереди

Структура очереди-массива:

```
typedef struct
{
    elem_t data[QUEUE_SIZE];
    int is_empty;
    int pin;
    int pout;
} queue_a;
```

data — массив, содержащий элементы очереди

is_empty — флаг, равный 1, если очередь пуста, иначе 0 (нужен для того, чтобы определять, пуста очередь или полностью заполнена при равенстве pout == pin)

pin — указатель за последний элемент очереди (хранится индекс кольцевого массива)

pout — указатель на голову очереди (хранится так же индекс)

Алгоритм

При симуляции работы обрабатываются исключительные ситуации. В моем случае их 2: окончание времени работы автомата и поступление новой заявки в очередь. Таким образом алгоритм сводится к следующему: обработка исключительной ситуации или, если ее нет, проматывание времени моделирования до ближайшей исключительной ситуации.

Что касается обработки этих ситуаций, то она заключается в следующем:

1. Обработка освобождения автомата:

1.1. Проверка, какой тип заявки вышел и замер статистики для нее (время простоя в очереди и т. п.) (если заявка 2 типа, то вставляем ее в очередь)

1.2. Взятие заявки из головы очереди в автомат или, если ее нет, замер времени простоя автомата до получения следующей заявки.

2. Поступление заявки в очередь:

2.1. Проверить, есть ли место в очереди.

2.2. Вставить элемент при выполнении (2.1)

Расчеты времени и памяти.

Фрагментация:

В большем количестве случаев фрагментация была минимальной (на 1500 старых участков памяти приходится порядка 10 новых участков)

Работа аппарата:

В среднем случае отклонения работы автомата от расчетных данных — 1-3%. В худшем случае — 6%.

Время (в тиках):

push

| Количество добавляемых элементов | очередь-список | очередь-массив |
|----------------------------------|----------------|----------------|
| 100 | 13110 | 8923 |
| 500 | 62369 | 41352 |
| 1000 | 130145 | 80332 |

pop

| Количество удаляемых элементов | очередь-список | очередь-массив |
|--------------------------------|----------------|----------------|
| 100 | 20421 | 11953 |
| 500 | 105432 | 60543 |
| 1000 | 195347 | 124832 |

Память:

| Количество элементов в очереди | очередь-список | очередь-массив |
|--------------------------------|----------------|----------------|
| 100 | 2000 + 20 | 1200 + 12 |
| 1000 | 20000 + 20 | 12000 + 12 |
| 10000 | 200000 + 20 | 120000 + 12 |

Выводы по проделанной работе

При реализации очереди в целях экономии времени и (или) памяти выгоднее использовать массив, однако если нам заранее не известен хотя бы примерный объем данных, то лучше использовать список (так как выделять непрерывную область памяти, которую требует массив, не всегда возможно (в особенности, если эта область требуется большой))

Контрольные вопросы

1. Что такое очередь?

Очередь — структура данных, работающая по принципу FIFO и поддерживающая 2 операции вставки в хвост и удаления из головы.

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

При реализации очереди списком в моем случае требуется 20 байт на хранение одного элемента (+ 20 байт на хранения указателя на голову, хвост и размер).

При реализации очереди массивом на один элемент расходуется 12 байт (+ 12 байт на хранения указателей на голову, хвост и флага (пуст ли массив или нет))

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

При реализации статическим массивом у нас нет выделения памяти, а удаление элемента происходит простым сдвигом указателя на голову вперед (кольцевой массив*)

При реализации списком у нас удаляется узел (и очищается память), на который указывает голова и голова начинает указывать на следующий узел.

4. Что происходит с элементами очереди при ее просмотре?

Учитывая, что у нас реализованы только 2 функции, то мы можем просмотреть очередь только удаляя элементы (ну или при помощи 2 очередей сохранить очередь)

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

При реализации очереди в целях экономии времени и (или) памяти выгоднее использовать массив, однако если нам заранее не известен хотя бы примерный объем данных, то лучше использовать список (так как выделять непрерывную область памяти, которую требует массив, не всегда возможно (в особенности, если эта область требуется большой))

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

В виде указателей — при неизвестном хотя бы примерном размере или при сильной пульсации очереди (когда максимальный размер очереди сильно больше среднего размера).

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

8. Что такое фрагментация памяти?

Показатель того, как делится память в ОЗУ. Говорят, что если память разделена на много маленьких кусочков, то присутствует фрагментация памяти. Когда память выделяется большими кусками, то фрагментация, можно сказать,

отсутствует.

9. На что необходимо обратить внимание при тестировании программы?

Правильность добавления в очередь, удаления из очереди, корректность работы с пустой очередью и переполненной очередью.

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

При работе со статическим массивом нет выделения и освобождения памяти.

При работе со списком выделение памяти происходит под узел при поступлении заявки и освобождение происходит при удалении элемента (освобождается память, выделенная под целый узел). Постоянная память (под хранение указателей на голову, хвост и размер) выделяется статически один раз.