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)

Дисциплина “Типы и структуры данных”

Лабораторный практикум №5

по теме: «обработка очередей»

Выполнил студент: __*Клименко Алексей Константинович*_____

фамилия, имя, отчество

Группа: _____*ИУ7-35Б*_____

Проверил, к.п.н.: _____

подпись, дата

Оценка _____ Дата _____

2020 г.

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 рализации) по времени и по используемому объему памяти.

описание условия задачи

Создать программу для моделирования работы двух очередей с общим обслуживающим аппаратом. Система должна быть с абсолютным приоритетом и повторным обслуживанием.

Реализовать очереди: а) массивом; б) списком.

Смоделировать процесс обслуживания первых 1000 заявок 1-го типа, выдавая после обслуживания каждых 100 заявок 1-го типа информацию о текущей и максимальной длине каждой очереди, а в конце процесса - общее время моделирования и количество вашедших в систему и вышедших из нее заявок обоих типов, среднем времени пребывания заявок в очереди, количестве "выброшенных" заявок второго типа.

Техническое задание

исходные данные

Исходными данными являются параметры моделирования, вводимые пользователем.

Примеры настройки параметров моделирования:

Ввод пользователя	**Результат**
`config`	переход в режим настроек
`set a 1 5`	задание граничных значений для времени прихода заявки в первую очередь
`set d 2.3 2.4`	задание граничных значений для времени обработки заявки из второй очереди
`back`	выход из режима настроек

результат

Результатом работы программы является моделирование системы из двух очередей с абсолютным приоритетом и повторным обслуживанием, и обслуживающего аппарата.

выходные данные

Выходными данными являются теоритические времена, расчитанные на основе параметров моделирования, а также результаты самого моделирования - времена обработки заявок обоих типов, а также статистические данные используемых при моделировании очередей, и список используемых адресов при реализации очереди на списке.

способы обращения к программе

Для запуска программы необходимо запустить файл **app.exe**. Далее запустить моделирование командой **run A A** или перейти в меню настроек для изменения параметров моделирования.

возможные аварийные ситуации и ошибки пользователя

При неверном вводе команды программа попросит ввести команду снова, а при невозможности выполнения команды при неверных параметрах моделирования сообщает об этом, и предлагает задать другие параметры моделирования.

Структуры данных

Для реализации очереди на массиве была выбрана следующая структура (используется кольцевой массив):

```
struct queue_arr
{
   uint32_t capacity;  // вместимость очереди
   uint32_t size;      // размер очереди
   int32_t *begin;     // начало массива (нулевой элемент)
   int32_t *end;      // указатель за последним элементом в массиве
      int32_t *first;   // указатель на первый элемент в очереди
      int32_t *last;     // указатель за последним элементом в очереди
};
```

Реализация очереди на связном списке:

```
struct __qu_lst_node
{
   struct __qu_lst_node *next; // указатель на следующий узел в очереди
   qdata_t data;               // данные элемента очереди
};

struct queue_lst
{
   uint32_t size;              // размер очереди
   struct __qu_lst_node *first; // указатель на первый элемент в очереди
   struct __qu_lst_node **last; // указатель на поле <next>
                        // в последнем элементе очереди
};
```

Набор функций

```
// Создаёт новую очередь заданной вместимости.
struct queue qu_create(queue_imp_t type, uint32_t capacity);

// Удаляет очередь.
void qu_destroy(struct queue *qu);

// Возвращает размер очереди.
uint32_t qu_get_size(struct queue *qu);

// Добавляет элемент в конец очереди.
int qu_push_back(struct queue *qu, int32_t value);

// Удаляет первый элемент из очереди.
int qu_pop_front(struct queue *qu, int32_t *value);
```

Описание алгоритма обработки данных

Моделирование работы обслуживающего аппарата состоит из итерационной обработки событий среди которых приход заявки в очередь и смена обрабатываемой заявки в обслуживающем аппарате.

```
К := требуемое кол-во заявок 1го типа
пока К > 0:
   если время прихода заявки 1го типа < 0:
      добавить новую заявку в первую очередь;
      обновить время следующего прихода заявки 1го типа;
      если текущая обрабатываемая заявка 2го типа:
         вернуть обрабатываемую заявку во вторую очередь;
         обновить время обработки текущей заявки;
      конец если;
   конец если;
   если время прихода заявки 2го типа < 0:
      добавить новую заявку в первую очередь;
      обновить время следующего прихода заявки 1го типа;
   конец если;
   если время обработки текущей заявки < 0:
      если обработана заявка 1го типа:
         К := К - 1;
      конец если;
      если первая очередь не пуста:
         удалить заявку из первой очереди;
         пометить тип текущей обрабатываемой заявки как 1;
      иначе если вторая очередь не пуста:
         удалить заявку из второй очереди;
         пометить тип текущей обрабатываемой заявки как 2;
      конец если;
      обновить время обработки текущей завки;
   конец если;
   уменьшить все времена ожидания (прихода и обработки заявок);
```

конец пока.

Набор функциональных тестов

№	Описание теста	Входные данные	Выходные данные
1	Некорректная команда	`a`	Сообщение о неверной команде. Возврат в меню
2	Запуск моделирования с настройками по-умолчанию	`run A L`	Вывод информации в процессе моделирования на экран

3	Изменение параметров по-умолчанию	`config`
		`set a 1.0 2.0`
		`set b 3 3`
		`show`
			Вывод измененных параметров на экран

4	Сброс настроек	`config`
		`set a 1.0 2.0`
		`set b 3 3`
		`reset`
		`show`
			Вывод настроек по-умолчанию

5	Установка неверного промежутка времени	`config` `set a 20 18`	Вывод сообщении о неверном временном промежутке
6	Установка нулевого промежутка времени	`config` `set a 0 0`	Вывод сообщении о неверном временном промежутке
7	Переход в ручной режим	`manual` `show`	Вывод пустых очередей
8	Удаление элемента из пустых очередей	`manual` `pop`	Вывод сообщении о том, что нечего удалять

9	Превышение максимального числа элементов в очередях	`manual`
		`push 1`
		`...`
		`push 11`
			Вывод сообщении о переполнении очередей

Тесты эффективности

Так как вставка и удаление элементов из очереди происходит на концах массива или списка, то временная сложность этих операций не зависит от числа элементов в очереди и равна **O(1)**. Однако при реализации очереди с помощью кольцевого массива требуется затратить немного больше времени на то, чтобы правильно расставить указатели на начало и конец очереди в массиве.

Результаты тестирования эффективности по времени операций добавления и удаления из очереди в виде таблицы:

Операция	время массив	время список	эффективность массива
push_back	17 тактов	178 тактов	90,4 %

pop_front	18 тактов	2076 тактов	99,1 %

Такое значительное расхождение времен получается из-за того, что для вставки и удаления из очереди на списке требуется обащение к менеджеру динамической памяти, который тратит соизмеримое время для того, чтобы выделить или очистить блок памяти.

Сравнительная таблица эффективности по памяти обоих решений:

макс. размер	реал. размер	заполнен-ность, %	размер очереди на массиве	размер очереди на списке	эффектив-ность очереди на списке, %
10	4	40	80	88	-10
10	8	80	80	152	-90
100	15	15	440	264	40
100	35	35	440	584	-33
1000	240	24	4040	3864	4
1000	260	26	4040	4184	-4

Таким образом, можно сделать вывод о том, что использование списков для реализации очереди является эффективным решением по памяти в случаях, когда реальный размер очереди зачастую будет менее **25%** от максимально возможного.

Выводы по проделанной работе

Использование той или иной реализации очереди сильно зависит от выполняемых задач. Для представленной задачи моделирования работы обслуживающего аппарата сложно сразу сказать, какая реализация будет наиболее эффективной: у каждой из них есть как достоинства так и недостатки.

Для реализации очереди на массиве характерна быстрота выполняемых операции вставки и удаления. Это решение эффективно по времени всегда, но зачастую проигрывает по памяти реализации на списке, которая показывает свою эффективность при больших скачках размера очереди, но преобладающем малом размере.

Контрольные вопросы

1. *Что такое очередь?*

Очередь - это абстрактная линейная структура данных с двумя концами, имеющая операции вставки в один конец и удаления из другого

конца. В отличие от стека работает по принципу FIFO - первый пришёл - первый вышел.

2. *Каким образом и какой объём памяти выделяется под хранение очереди при различной её реализации?*

При реализации очереди на массиве память выделяется единожды в момент инициализации.

При реализации стека на связном списке память выделяется каждый раз при добавлении нового элемента в стек.

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

При реализации очереди на массиве память очищается только по окончании работы с очередью.

При реализации стека на связном списке память очищается каждый раз при удалении элемента из очереди.

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

В классической реализации очереди просмотр осуществляется поэлементным удалением элементов из одного конца, запоминанием их и вставкой в другой конец.

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

Зависимость выбора реализации очереди обуславливается средним числом элементов в очереди или среднестатистической заполненностью очереди. Если в очереди будет постоянно находиться небольшое число элементов (менее 25% для представленной задачи), и лишь изредка это число будет близко к максимально допустимому, то в этом случае будет гораздо эффективнее использовать список в качестве реализации очереди.

Если же среднестатистическая заполненность очереди будет близка к максимальной (больше 25% для представленной задачи), то следует присмотреться к реализации очереди с помощью массива.

6. *В каком случае лучше реализовывать очередь посредством указателей, а в каком - массивом?*

Если в очереди будет постоянно находиться небольшое число элементов (менее 25% для представленной задачи), и лишь изредка это число будет близко к максимально допустимому, то в этом случае лучше реализовать очередь посредством указателей.

Если же среднестатистическая заполненность очереди будет близка к максимальной (больше 25% для представленной задачи), то лучше реализовать очередь массивом.

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

Достоинства и недостатки реализации на массиве:

+ быстрота выполнения операций вставки и удаления.

- нужно знать максимальный размер очереди заранее.

+/- эффективна по памяти только при большой заполненности.

Достоинства и недостатки реализации на списке:

+ размер ограничен лишь оперативной памятью.

+/- эффективна по памяти при небольшой заполненности.

- операции выполняются медленнее, чем на массиве.

8. *Что такое фрагментация памяти?*

Фрагментация памяти - явление, при котором занятые участки памяти перемешаны со свободными участками. Оно приводит к ситуациям, когда несмотря на то, что физический объем свободной памяти достаточен для выделения блока заданной длины, выделение не может быть осуществлено из-за отсутствия цельного свободного блока памяти заданного размера.

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

При тестировании программы необходимо учесть как можно больше (а в лучшем случае все) классов эквивалентности для входных данных, убедиться, что 100% написанного кода было выполнено, и результат выполнения совпадает с ожидаемым.

Также следует вести учет потребления программой ресурсов, выделяемых операционной системой. Все ресурсы по окончании работы с ними должны быть возвращены системе в полном объеме.

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

При запросе на выделение памяти менеджер памяти проходит по списку блоков памяти в поисках блока требуемого размера, далее если свободный блок слишком большой, он разбивает его на два, возвращая указатель на блок требуемой длины. Если же блок требуемого размера не был найден (например вследствие фрагментации памяти) то менеджер памяти завершает работу с неудачей.

При освобождении памяти менеджер памяти помечает блок как свободный и при необходимости объединяет соседние свободные блоки в один.