|  |  |
| --- | --- |
|  | **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)** |

ФАКУЛЬТЕТ:  *Информатика и системы управления*

КАФЕДРА:  *Программное обеспечение ЭВМ и информационные технологии*

Отчет по лабораторной работе №5 по предмету

“Типы и структуры данных”

Обработка очередей.

Выполнила:

Кузнецова Анастасия

Группа:

ИУ7-31Б

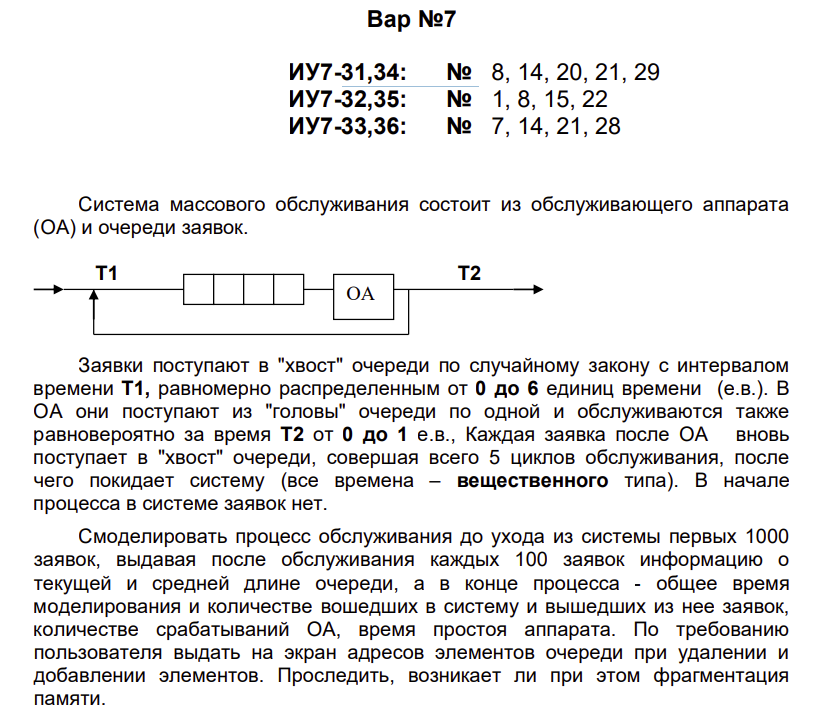
Вариант 7

Проверил(а):

Силантьева Александра Васильевна

**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.

**Описание условия задачи**



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.

**2. Описание ТЗ:**

Описание исходных данных и результатов

*Входные данные:*

Программа представляет собой меню, пользователь должен ввести свой выбор (целое число от 0 до 4)

Если пользователь хочет изменить время прихода или время обработки заявки (выбор 3), то на вход подаётся вещественное число – граница интервала.

Ограничения для входных данных:

* Граница интервала для времени должна быть числом

*Выходные данные:*

Программа выводит меню и в зависимости от выбора пользователя выводит:

1. Вывод результатов моделирования очереди с помощью массива:

* после обслуживания каждых 100 заявок информацию о текущей и средней длине очереди
* 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 адреса элементов очереди при удалении и добавлении элементов.

1. Вывод результатов моделирования очереди с помощью списка:

* после обслуживания каждых 100 заявок информацию о текущей и средней длине очереди
* 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 адреса элементов очереди при удалении и добавлении элементов.

1. Вывод текущих границ времени
2. Вывод результатов сравнения времени и памяти при двух реализациях

Описание задачи, реализуемой программой

Программа выводит меню и выполняет задачу, запрошенную пользователем. В программе реализовано меню с выборами действий:

1. Выход
2. Моделирование очереди с помощью массива
3. Моделирование очереди с помощью списка
4. Изменение границ времени прихода или времени обработки заявки
5. Сравнение времени и памяти при двух реализациях

Способ обращения к программе

Программу следует вызывать из командной строки с помощью команды:

./app.exe

Описание возможных аварийных ситуаций и ошибок пользователей

* Ввод нечисловой границы времени для пункта 3
* Ввод такого отрезка времени, при котором возникает переполнение очереди, реализованной с помощью массива

**3. Описание внутренних СД**

Очередь-массив:

#define MAX\_SIZE 1000  
  
typedef struct {  
 int array[MAX\_SIZE];  
 int p\_in;  
 int p\_out;  
} queue\_arr\_t;

Очередь-список:

struct node {  
 int value;  
 struct node \*next;  
};  
  
typedef struct node node\_t;  
typedef struct {  
 node\_t \*p\_in;  
 node\_t \*p\_out;  
} queue\_list\_t;

**4. Описание алгоритма**

1. Вывод меню

2. В зависимости от выбора пользователя:

2.0 Выход

2.1.1 Пока количество вышедших заявок не станет равным 1000:

2.1.2 Случайная генерация времени прихода новой заявки

2.1.3 Обработка заявок из очереди до достижения необходимого времени прихода п.2.1.1

2.1.4 Добавление новой заявки в очередь

2.2.1 Алгоритм совпадает с пунктами 2.1.1, 2.1.2, 2.1.3, 2.1.4

2.2.2 Вывод адресов удалённых элементов по запросу пользователя

2.3.1 Запрос на изменение времени прихода или времени обработки

2.4.1 Вывод результатов сравнения времени и памяти при двух реализациях

**5. Набор тестов**

Позитивные тесты:

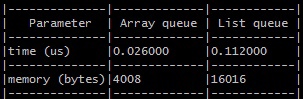
|  |  |  |  |
| --- | --- | --- | --- |
| № | Описание | Input | Output |
| 1 | Время прихода от 0 до 6  Время обработки от 0 до 1  Моделирование с помощью массива | Enter a choice:  1 |  |
| 2 | Время прихода от 0 до 6  Время обработки от 0 до 1  Моделирование с помощью списка | Enter a choice:  2 |  |
| 3 | Вывод удалённых адресов памяти при списке (отслеживание фрагментации) | Enter a choice:  2  1 |  |
| 4 | Время прихода от 0 до 4  Время обработки от 0 до 1  Моделирование с помощью массива | Enter a choice:  1 |  |
| 5 | Время прихода от 0 до 4  Время обработки от 0 до 1  Моделирование с помощью списка | Enter a choice:  2 |  |
| 6 | Время прихода от 0 до 6  Время обработки от 0 до 0.5  Моделирование с помощью массива | Enter a choice:  1 |  |
| 7 | Время прихода от 0 до 6  Время обработки от 0 до 0.5  Моделирование с помощью списка | Enter a choice:  2 |  |

Негативные тесты:

|  |  |  |  |
| --- | --- | --- | --- |
| № | Описание | Input | Output |
| 1 | Время прихода от 0 до 1  Время обработки от 0 до 1  Моделирование с помощью массива | Enter a choice:  1 |  |

**6. Сравнение времени и памяти при двух реализациях.**

Я сравнивала время добавления и удаления элемента в очередь-массив и в очередь-список. Замеры проводила на 1000 повторах брала среднее значение времени. Заполняла 1000 элементов. Результаты представлены ниже:



Массив оказывается эффективнее и по времени (примерно в 4 раза), и по памяти в случае полного заполнения (примерно в 4 раза).

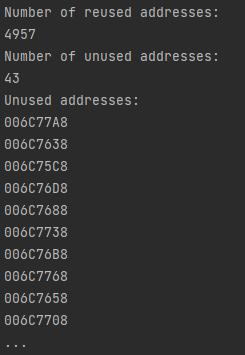
**6. Выводы по проделанной работе**

В результате выполненной лабораторной работы я приобрела навыки работы с типом данных «очередь», который представлен в виде одномерного массива и односвязного линейного списка, оценила эффективность программы (при различной реализации) по времени и по используемому объему памяти.

Массив оказывается эффективнее и по времени (примерно в 4 раза), и по памяти в случае полного заполнения (примерно в 4 раза). Однако очередь, реализованная с помощью линейного односвязного списка, не имеет ограничений по длине в отличие от статического массива.

Эффективность списка по памяти можно заметить только если элементов будет меньше максимального количества элементов в статическом массиве, так как в статическом массиве память выделяется под весь размер, а в списке только под введённое количество элементов.

При работе программы фрагментация памяти почти не происходит.



При реализации принципа LIFO в 4 лабораторной работе выигрыш массива по времени был практически такой же (он был примерно в 4-5 раз), а по памяти выигрыш был больше в 4 раза, что связано с типом хранимых элементов. В 4 лабораторной значениями элементов стека были символы char, а в 5 лабораторной значение int.

Таким образом эффективность реализаций LIFO и FIFO примерно одинаковая, эффективность по памяти зависит от типа элемента.

**7. Ответы на вопросы**

1. Что такое FIFO и LIFO?

FIFO (First In – First Out) – принцип работы, заключающийся в правиле: первым пришёл – первым вышел. По этому принципу реализована очередь.

LIFO (Last In – First Out) – принцип работы, заключающийся в правиле: последним пришел – первым ушел. По этому принципу реализован стек.

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

Если очередь реализована в виде статического или динамического массива (вектора), то для её хранения обычно отводится последовательная область памяти из m мест по L байт, где L – размер поля данных для одного элемента размещаемого типа. В моей реализации — это размер структуры, хранящей индекс pin, индекс pout, являющиеся началом и концом очереди соответственно, и статический массив значений int.

Если очередь реализована в виде односвязного линейного списка, то в статической памяти можно либо хранить адрес начала и конца очереди, либо – адрес начала очереди и количество элементов.

В моей реализации – это структура, содержащая указатели pin и pout, являющиеся началом и концом очереди соответственно. При каждом добавлении элемента выделяется новая область памяти, адрес которой записывается в указатель pin.

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

При реализации очереди с помощью статического массива память выделяется при компиляции и не меняется во время работы программы.

При реализации очереди с помощью линейного односвязного списка при удалении элемента сначала по указателю стека считывается информация об исключаемом элементе, а затем указатель смещается к предыдущему элементу. После чего освобождается память, выделенная под элемент.

1. Что происходит с элементами очереди при её просмотре?

При просмотре очереди чтобы обратиться ко всем элементам очереди надо удалить предыдущий элемент, так как принцип очереди FIFO мы не можем обратиться к элементу стека из середины.

Таким образом при просмотре очереди все её элементы поочередно удаляются.

1. От чего зависит эффективность физической реализации очереди?

При полностью заполненном массиве эффективнее реализовывать очередь с помощью массива.

Эффективность зависит от того, насколько заполнен массив. Если размер массива намного превышает количество введённых элементов стека, то эффективнее по памяти использовать односвязный список.

Однако классическое представление стека - односвязный линейный список.

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

При добавлении элемента в очередь, реализованную массивом, переполнение очереди наступает, когда указатель «хвоста» Pin перейдет границу последнего элемента, т. е. возникает ситуация, при которой Pin = Qm+L. Причем, переполнение наступает независимо от состояния Pout, т. е. когда, несмотря на наличие свободной памяти слева от Pout нет возможности поместить следующий элемент. Чтобы не случалось переполнения, при каждом исключении из очереди обычно производят сдвиг всех элементов к «голове». Тогда переполнение произойдет лишь в том случае, если все m мест выделенной памяти заняты. Но на эти последовательные сдвиги затрачивается время. Поэтому более эффективно использовать кольцевую очередь. Добавление происходит в первые ячейки массива, если «хвост» массива занят. При равенстве Pin = Qm+L, означающем, что «хвост» вышел за конец очереди, производят коррекцию: Pin = Q1. Если Pout > Q1, то еще имеется свободное место. В кольцевой структуре нет необходимости сдвигать элементы, но зато сложнее реализовать алгоритмы включения и исключения элементов. Но даже в этом случае проблема переполнения радикально все равно не решается, так как при заполнении очереди возможна ситуация, когда следующий элемент будет «затирать» первый, что потребует дополнительных проверок заполнения очереди.

Большинство проблем, возникающих при реализации очереди в виде массива, устраняется при реализации очереди на основе односвязного линейного списка. В этом случае при добавлении элементов размер очереди ограничивается только объемом доступной памяти.

Удаление элемента при этом удобнее при реализации очереди массивом, так как при этом нет необходимости следить за освобождением памяти.

1. Что такое фрагментация памяти, и в какой части ОП она возникает?

Фрагментация – это возникновении "дыр" - участков памяти, которые не могут быть использованы.

Фрагментация возникает в куче.

1. Для чего нужен алгоритм «близнецов».

Вариант подхода для случая, когда различные размеры блоков являются степенями 2-ки (512 байт, 1Кб, 2К и т.д.)

Алгоритм близнецов значительно снижает фрагментацию памяти и резко ускоряет поиск блоков. Наиболее важным преимуществом этого подхода является то, что даже в наихудшем случае время поиска не превышает О(log(Smax)- log(Smin)), где Smax, Smin - соответственно максимальный и минимальный размеры используемых блоков. Это делает алгоритм близнецов труднозаменимым для ситуаций, когда необходимо гарантированное время реакции – например, для задач реального времени.

1. Какие дисциплины выделения памяти вы знаете?

Две основные: "самый подходящий (best fit) и "первый подходящий" (first fit).

По дисциплине "самый подходящий" выделяется тот свободный участок, размер которого равен запрошенному или превышает его на минимальную величину.

По дисциплине "первый подходящий" выделяется первый же найденный свободный участок, размер которого не меньше запрошенного (эффективнее).

1. На что необходимо обратить внимание при тестировании программы?

При тестировании необходимо:

• Реализовать полный тест: т.е. подобрать такие входные данные, чтобы каждый оператор, особенно стоящий в цикле или ветви, был выполнен хотя бы один раз.

• Проверять граничные значения для индексов массивов.

• Отслеживать варианты неверных (ошибочных) данных в математических функциях (корень из отрицательного числа, логарифм отрицательного числа, деление на 0 и т.п.).

• Проверять граничные значения данных (минимумов и максимумов), пустые данные, при этом проверяется очистка памяти (зануление).

• Проверять правильности ввода-вывода

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

Память представляет собой бинарную кучу с признаком — занятость-незанятость ячейки. Динамический запрос меняет признак ячейки.