Министерство образования и науки 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 (ИУ)

КАФЕДРА Программное обеспечение ЭВМ и информационные технологии (ИУ7)

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №5
ОЧЕРЕДЬ
ДИСЦИПЛИНА: ТИПЫ И СТРУКТУРЫ ДАННЫХ

Студент, группа **Боренко А. ИУ7-32Б**

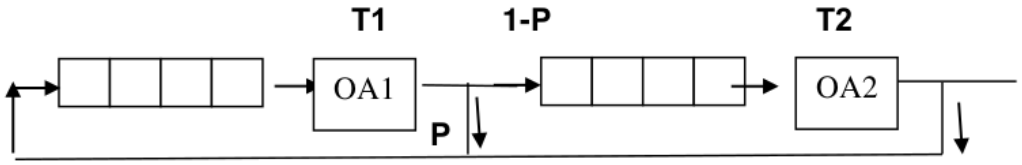
2020 г.

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 реализации) по времени и по используемому объему памяти.

Описание условия задачи

Вариант 3



Система массового обслуживания состоит из двух обслуживающих аппаратов (ОА1 и ОА2) и двух очередей заявок. Всего в системе обращается 100 заявок. Заявки поступают в "хвост" каждой очереди; в ОА они поступают из "головы" очереди по одной и обслуживаются по случайному закону за интервалы времени Т1 и Т2, равномерно распределенные от 0 до 6 и от 1 до 8 единиц времени соответственно. (Все времена – вещественного типа). Каждая заявка после ОА1 с вероятностью P=0.7 вновь поступает в "хвост" первой очереди, совершая новый цикл обслуживания, а с вероятностью 1-P входит во вторую очередь. В начале процесса все заявки находятся в первой очереди.

Смоделировать процесс обслуживания до выхода из ОА2 первых 1000 заявок. Выдавать на экран после обслуживания в ОА2 каждых 100 заявок информацию о текущей и средней длине каждой очереди, а в конце процесса - общее время моделирования, время простоя ОА2, количество срабатываний ОА1, среднее времени пребывания заявок в очереди. Обеспечить по требованию пользователя выдачу на экран адресов элементов очереди при удалении и добавлении элементов. Проследить, возникает ли при этом фрагментация памяти.

Описание Технического Задания

Описание исходных данных и результатов

Исходные данные:

- Ввод пользователя (Выбор действия, Элементы очереди)

Реультаты:

- В зависимости от выбора пользователя
  - Работа с очередью /Вывод сохраненных адресов
  - Моделирование работы автоматов и очередей
  - Тестирование — сравнение реализаций списком и массивом по памяти и по времени

Описание задачи, реализуемой программой

Вывод меню, для реализации выбора пользователя. Ввод очереди. Моделирование работы автоматов и очередей (подробнее в описании задания). Тестирование.

Способ обращения к программе

Обращение к программе происходит через терминал. В командную строку вводится «./app.exe».

Описание возможных аварийных ситуаций и ошибок пользователя

1. Ввод буквы вместо пункта меню — (Расценивается как выход)
2. Ввод буквы при вводе стека — (Расценивается как выход в главное меню)
3. Ввод цифры не из меню (тоже что и ввод буквы)
4. Ошибка при вводе стека (тоже что и ввод буквы)

Описание внутренних структур данных

Типы данных для работы с очередью

```
typedef struct tw_node tw_node;
typedef struct tw_list tw_list;
typedef struct tw_array tw_array;

struct tw_node
{
      int element;
      tw_node *next;
};

struct tw_list
{
      tw_node *pin;
      tw_node *pout;
};

struct tw_array
{
      int size;
      int *pin;
      int *pout;
};
```

Типы данных для с адресами (узел списка адресов)

```
typedef struct adress_stack_node_t adress_stack_node_t;
struct adress_stack_node_t
{
      long int element;
      adress_stack_node_t *next;
};
```

Типы данных для вывода

```
// Структуры вывода
typedef struct intermediat_result_t intermediat_result_t;
typedef struct final_result_t final_result_t;

struct intermediat_result_t
```

```
{
      double current_len;
      double average_len;
};
struct final_result_t
{
      my_time_t result_system_time;
      my_time_t average_request_time1;
      my_time_t average_request_time2;
      int a1_calling;
      my_time_t a2_down_time;
};
```

Типы данных для моделирования работы автоматов

```
// Очередь
struct array_turn_t
{
      int id;
      request_t *pout;
      request_t *pin;

      request_t *start;
      int max_len;

      int len;
      average_t time_av;
      average_t len_av;
};
// Очередь
struct list_turn_t
{
      int id;
      list_node_t *pout;
      list_node_t *pin;
      int len;
      average_t time_av;
      average_t len_av;
};
// Заявка
struct request_t
{
      int id;
      my_time_t time_entry;
};
// Автомат
struct automat_t
{
      int id;

      my_time_t work_ending_time;
      my_time_t down_time;

      my_time_t tmax;
      my_time_t tmin;

      int call_count;

      list_node_t *current_request;

      list_node_t *next_turns;

      int (*redirection)();
};
// Узел односвязного списка
```

```
struct list_node_t
{
      void *element;
      list_node_t *next;
};
```

Описание алгоритма

1. Вывести меню
2. Получить введенное число
3. Если это 1
    1. Вывести меню работы со очередью
    2. Получить число — выбор пользователя
    3. Если 1
        1. Добавить элемент
    4. Если 2
        1. Удалить
    5. Если 3
        1. Вывести очередь
    6. Если 4
        1. Вывести адреса
    7. Если 5 или ошибка
        1. Вывести адреса использованных ячеек памяти
        2. Освободить все списки/массивы
4. Если это 2
    1. Провести моделирование реализуя очереди массивами
    2. Вывести результаты
5. Если это 3
    1. Провести моделирование реализуя очереди списками
    2. Вывести результаты
6. Если это 4
    1. Протестировать
    2. Вывести результаты
7. Иначе
    1. Закончить работу

Алгоритм моделирования

1. Заполнить перую очередь заявками
2. Пока автомат 2 не вызван 1000 раз
    1. Если первый автомат пуст и очередь к нему пуста
        1. Посчитать время простоя первого автомата
        2. Посчитать время работы второго автомата и общее время работы системы
    2. Если второй автомат пуст и очередь к нему пуста
        1. Посчитать время простоя второго автомата
        2. Посчитать время работы второго автомата и общее время работы системы
    3. Если автоматы не простаивают
        1. Если время завершения работы первого автомата больше времени завершения работы второго автомата
            1. Работает второй автомат
        2. Если время завершения работы второго автомата меньше времени завершения работы второго автомата
            1. Работает первый автомат
        3. Если времена равны
            1. Работают оба атомата
        4. Обновить время системы (взять наименьшее из времен работы автоматов 1 и 2)
3. Вернуть результаты

Алгоритм работы автомата

1. Если автомат непуст
    1. Взять элемент из автомата
    2. Положить его в очередь
    3. Добавить вызов автомату
2. Если очередь непуста
    1. Взять эелемент из очереди
    2. Положить его в автомат

Основные функции

`adress_work.h`

- `int add_adresses(adress_stack_node_t **head, long int adress);`

- void output_adresses(adress_stack_node_t *head);
- void free_adresses(adress_stack_node_t **head);

help_functions.h

- my_time_t random_time(my_time_t tmax, my_time_t tmin);
- int create_request(request_t **new_request, int id, int time_entry);
- void delete_request(request_t **del_request);
- int redirection1();
- int redirection2();
- int create_node(list_node_t **new_node, void *element);
- void delete_node(list_node_t **new_node);
- void add_node(list_node_t **last, list_node_t *new_node);
- list_node_t *remove_node(list_node_t **head);
- void output_intermediat_result(int a2_calling, void *turn1, void *turn2, intermediat_result_t *(*get_intermediate)(void *turn));
- final_result_t *get_final(void *turn1, void *turn2, automat_t a1, automat_t a2, my_time_t sys_time,
- my_time_t (*get_av_time)(void *turn));
- void output_final_result(void *turn1, void *turn2, automat_t a1, automat_t a2, my_time_t sys_time,
- my_time_t (*get_av_time)(void *turn));
- void clearInputBuf(void);
- void output_line(int size);

list_model.h

- int list_create_turn(list_turn_t **turn, int id);
- void list_delete_turn(list_turn_t **turn);
- void list_start_fill_turn(list_turn_t **turn);
- list_node_t *list_find_turn_by_id(list_node_t *head, int turn_id);
- void list_prepearing_turns(list_turn_t **t1, list_turn_t **t2);
- void list_prepearing_automats(automat_t *a1, automat_t *a2, list_turn_t **t1, list_turn_t **t2);
- my_time_t list_modeling(list_turn_t **t1, list_turn_t **t2, automat_t *a1, automat_t *a2, int condition);
- intermediat_result_t *get_list_intermediate(void *turn);
- my_time_t list_get_average_time(void *turn);

array_model.h

- list_node_t *array_find_turn_by_id(list_node_t *head, int turn_id);
- int array_create_turn(array_turn_t **turn, int id);
- void array_delete_turn(array_turn_t **turn);
- void array_start_fill_turn(array_turn_t **turn);
- void array_add_request(array_turn_t **turn, request_t request);
- request_t array_remove_request(array_turn_t **turn);
- void array_output_turn(array_turn_t t1);
- intermediat_result_t *get_array_intermediate(void *turn);
- my_time_t array_get_average_time(void *turn);
- void array_prepearing_turns(array_turn_t **t1, array_turn_t **t2);

- `void array_prearing_automats(automat_t *a1, automat_t *a2, array_turn_t **t1,array_turn_t **t2);`
- `void array_output_turn(array_turn_t t1);`
- `my_time_t array_modeling(array_turn_t **t1, array_turn_t **t2, automat_t *a1, automat_t *a2, int condition);`

`experiment.h`

- `void array_experiment();`
- `void list_experiment();`

`testing.h`

- `void testing();`

Тесты

Главное Меню

- *Негативные (Результат для всех — завершение программы)*
  - *Ввод буквы*
  - *Ввод числа не из приведенного списка*
- *Позитивные*
  - *Ввод 1 — действие — работа с ояередью*
  - *Ввод 2 — действие — моделирование работы автоматов*
  - *Ввод 3 — действие — Тестирование*

Меню Работа с очередью

- *Негативные (Результат для всех — выход в главное меню)*
  - *Ввод буквы*
  - *Ввод числа не из приведенного списка*
- *Позитивные*
  - *Ввод 1 — действие — Добавление элемента*
  - *Ввод 2 — действие — Удаление элемента*
  - *Ввод 3 — действие — Вывод стека*
  - *Ввод 4 — действие — Вывод списка адресов*
  - *Ввод 5 — действие — Освобождение стека, вывод списка адресов и егое освобождение*

Моделирование

- *Расчеты*
  - *Общее время системы 9999*
  - *Количество вызовов A1: 3333*
  - *Время простоя 45000*

- *Время заявки в очереди 1: 297*
- *Время заявки в очереди 2: 1,4*

Выводы о проделанной работе

Результаты тестирования (сравнение)

```
-------------------------------------------------
Таблица с результатами тестирования (кол-во повторов
для вычисления времени = 100)
-------------------------------------------------
|     type |     time |   memory |
-------------------------------------------------
|    array |   154433 |     3608 |
|     list |    81244 |     3592 |
-------------------------------------------------
```

Массивы и списки заняли приблизительно одинаково по памяти, по времени же реализация очереди списком обходит релизацию массивом в 2 раза. Поэтому очереди зачастую выгоднее реализовывать списком.
Разница во времени обусловлена необходимостью сдвигать все элементы массива при удалении очередного элемента.

Фрагментация памяти происходит.
Отрывок из сохраненных адресов удаленных ячеек:

94251969962960
94251969967824
94251969962896
94251969962832
94251969967856
94251969962768
94251969962704

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?

При реализации очереди списком выделяется память под каждый элемент непосредственно при его добавлении.

При реализации очереди массивом выделяется наибольшая память, которая возможно будет использована, а возможно нет.

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

При реалиации стеком память освобождается при удалении каждого элемента.

При реалиации массивом память освобождается только при удалении всей очереди, а в случае удаления одного элемента память не освобождается, а просто переносится указатель, если очередь не переполнена.

4. Что происходит с элементами очереди при ее просмотре?

Они вынимаются из головы и кладутся в хвост — просмотр всех элементов возращает очередь в исходное состояние.

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

Эффективнее в большинстве случаев реализация списком. Зависит от наличия точного кол-ва элементов очереди.

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

Лучше реализовать массивом, если известен заранее размер очереди.

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

Реализация очереди массивом увеличивает время работы программы, т. к. элементы очереди надо сдвигать при удалении одного элемента.

Массив может иметь преимущество по памяти перед стеком, если известен точный размер очереди, и он почти неменяется в процессе работы.

Список выгоден по памяти, если размер очереди неизвестен, или меняется с течением времени.

Список выгоден по памяти, потому что не требуется сдвигать элементы очереди при удалении и добавлении элементов.

8. Что такое фрагментация памяти?

Фрагментация это выделение не последовательных адресов в памяти

9. На что необходимо обратить внимание при тестировании программы?

При тестировании программы необходимо:

-проверить правильность работы программы при различном заполнении очередей, т.е., когда время моделирования определяется временем обработки заявок и когда определяется временем прихода заявок;

-отследить переполнение очереди, если очередь в программе ограничена.

10. 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.