# Обработка очередей

### Цель работы

### Приобрести навыки работы с типом данных «очередь», представленным в виде одномерного массива и односвязного линейного списка, 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.

**Входные данные:**

**EndGenerateFReq(**вещественное число, большее 0)- максимальное время генерации заявки 1-го типа, **UseRequest** (вещественные число, большее 0)-максимальное время обработки заявки 1-го или 2-го типа. **MaxSReqPos**(натуральное число)- позиция элемента, не далее которого заявка второго типа вернётся в очередь.

**Выходные данные:**

Общее время моделирования(в условных единицах времени), время простоя аппарата(в секундах), количество вошедших в систему и вышедших из нее заявок первого типа и количество обращений заявок второго типа.

**Описание задачи:**

Система массового обслуживания состоит из обслуживающего аппарата (ОА) и очереди заявок двух типов.

Заявки 1-го типа поступают в "хвост" очереди по случайному закону с интервалом времени Т1, 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 Т2 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 Т3 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.

**Способ обращения к программе:**

Взаимодействие с программой происходит посредством командной строки

**Описание возможных аварийных ситуаций:**

* Возможная нехватка памяти при слишком большой длине очереди(реализация-список)
* Переполнение массива при слишком большой длине очереди(реализация очереди-массив)

**Сравнение по времени выполнения:**

**Обслуживание первых 1000 элементов**

|  |  |  |
| --- | --- | --- |
| Количество обслуженных элементов 1ого типа | Реализация | Время выполнения |
| 1000 | Список | 3.053000e+003 ms |
| 1000 | Список | 2.680000e+003 ms |
| 1000 | Список | 5.351000e+003 ms |
| 1000 | Список | 3.269000e+003 ms |
| 1000 | Список | 3.004000e+003 ms |
| Среднее время | | 3,476000e+003 ms |
| 1000 | Массив | 4.179000e+003 ms |
| 1000 | Массив | 2.861000e+003 ms |
| 1000 | Массив | 2.610000e+003 ms |
| 1000 | Массив | 2.772000e+003 ms |
| 1000 | Массив | 2.195000e+003 ms |
| Среднее время | | 2.482000e+003 ms |

**Сравнение по памяти**

|  |  |  |
| --- | --- | --- |
|  | Количество элементов | Занимаемая память(байт) |
| Список | 64(Максимальная длина очереди) | 512 |
| Список | 123 | 984 |
| Список | 126 | 1008 |
| Массив | 500 | 2000 |

**Внутренняя структура данных**

**Реализация список**

**struct** Queue {

**int** **size**; *//счетчик* *размера* *очереди*

Node \***first**; *//указатель* *на* *начало* *очереди*

Node \***last**; *//указатель* *на* *конец* *очереди*

};

**struct** Node *//описание* *узла* *списка*

{

**int** **data**; *//информационное* *поле*

Node \***next**; *//указатель* *на* *следующий* *элемент*

};

**Реализация массив**

**struct** Queue1

{

**int** **data**[N]; *//массив* *данных*

**int** **last**; *//указатель* *на* *начало*

};

**Функции работы с очередью, реализация –список**

**Создание**

**void** Creation(Queue \*Q) *//создание* *очереди* *список*

{

Q->**first**=**new** Node;

Q->**first**->**next**=**NULL**;

Q->**last**=Q->**first**;

Q->**size**=0;

}

**Добавление элемента**

**void** Add(Queue \*Q,**int** value) *//добавление* *элемента*

{

Q->**last**->**next**=**new** Node;

Q->**last**=Q->**last**->**next**;

Q->**last**->**data**=value; *//добавление* *элемента* *в* *конец*

Q->**last**->**next**=**NULL**; *//обнуление* *указателя* *на* *следующий* *элемент*

Q->**size**++;

}

**Удаление элемента**

**void** Delete(Queue \*Q) *//удаление* *элемента*

{

Node \*z=Q->**first**;

Q->**first**=Q->**first**->**next**; *//смещение* *указателя*

free(z);

Q->**size**--;}

**Вставка элемента на определенную позицию списка**

**void** Insert(Queue \*Q, **int** number, **int** value)

{

**int** p=0;

Node \*y;

y=Q->**first**;

**while** (p!=number)

{

p++;

y=y->**next**;

}

Node \*z=y->**next**;

y->**next**=**new** Node;

y->**next**->**next**=z;

y->**next**->**data**=value;

Q->**size**++;

**return**;

}

**Размер очереди**

**int** Size(Queue \*Q) *//размер* *очереди*

{ **return** Q->**size**; }

**Верхний элемент списка**

**bool** Full(Queue \*Q) *//проверка* *очереди* *на* *пустоту*

{

**if** (Q->**first**==Q->**last**) **return** **true**;

**else** **return** **false**;

}

**int** Top(Queue \*Q) *//вывод* *начального* *элемента*

{ **if** (!Full(Q))**return** Q->**first**->**next**->**data**;

**else** **return** 0;}

**Функции работы с очередью, реализация –массив**

**Создание**

**void** Creation(Queue1 \*Q) *//создание* *очереди* *МАССИВ*

{ Q->**last**=0; }

**Добавление элемента**

**void** Add(Queue1 \*Q, **int** val) *//добавление* *элемента* *МАССИВ*

{

Q->**data**[Q->**last**++]=val;

}

**Удаление элемента**

**void** Delete(Queue1 \*Q) *//удаление* *элемента* *МАССИВ*

{

**for** (**int** i=0; i<Q->**last** && i<N; i++) *//смещение* *элементов*

Q->**data**[i]=Q->**data**[i+1]; Q->**last**--;

}

**void** Delete(Queue \*Q) *//удаление* *элемента*

{

Node \*z=Q->**first**;

**if** (ai<1499) a[ai++]=Q->**first**;

Q->**first**=Q->**first**->**next**; *//смещение* *указателя* *----------------------------------------------добавить* *удаление*

free(z);

Q->**size**--;

}

**Вставка элемента на определенную позицию списка**

**void** Insert(Queue \*Q, **int** number, **int** value)

{

**int** p=0;

Node \*y;

y=Q->**first**;

**while** (p!=number)

{

p++;

y=y->**next**;

}

Node \*z=y->**next**;

y->**next**=**new** Node;

y->**next**->**next**=z;

y->**next**->**data**=value;

Q->**size**++;

**return**;

}

**Размер очереди**

**int** Size(Queue1 \*Q) *//размер* *очереди* *МАССИВ*

{ **return** Q->**last**; }

**Верхний элемент очереди**

**int** Top(Queue1 \*Q)

{ **return** Q->**data**[0];}

**Вывод**

Реализация очереди в виде массива оказалась на 40% быстрее, чем реализация в виде списка, даже несмотря на очевидные потери в памяти вследствие организации статического массива и того, что при удалении очередного элемента из массива происходит сдвиг всего массива на элемент влево(на что так же тратится время).