Лабораторная работа № 5

Методические указания

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.

Краткие теоретические сведения

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).

Для «хвоста» и «головы» очереди используют соответственно два указателя Pin и Pout, то есть исключается элемент с адресом Pout и включается элемент по адресу Pin. Моделировать простейшую линейную очередь можно как на основе вектора (одномерного массива), так и на основе динамического списка.

Реализация очереди в виде массива. При моделировании простейшей линейной очереди на основе одномерного массива выделяется последовательная область памяти из m мест по L байт, где L – размер поля данных для одного элемента размещаемого типа. В каждый текущий момент времени выделенная память может быть вся свободна, занята частично или занята полностью.

В начале процесса очередь пуста, при этом: Pin = Pout =Q1, где Q1 – адрес (индекс) первого элемента очереди, а количество элементов очереди (K) равно нулю. При включении очередного элемента в очередь он располагается по адресу Pin, а сам указатель Pin перемещается на длину типа данных к началу следующего элемента

(рис. 1). На рисунке символом X обозначена память, занятая элементами очереди, при этом считаем, что под массив была выделена область памяти от Q1 до Qm.

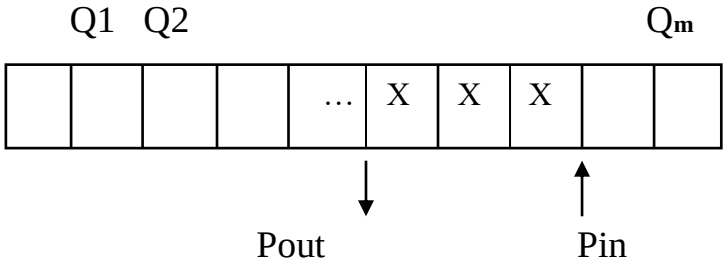

Рис. 1

В очереди исключается элемент, находящийся по адресу (индексу) Pout, а указатель Pout, как и в случае добавления, перемещается к следующему элементу. 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. (рис. 2). Добавление происходит в первые ячейки массива, если «хвост» массива занят.

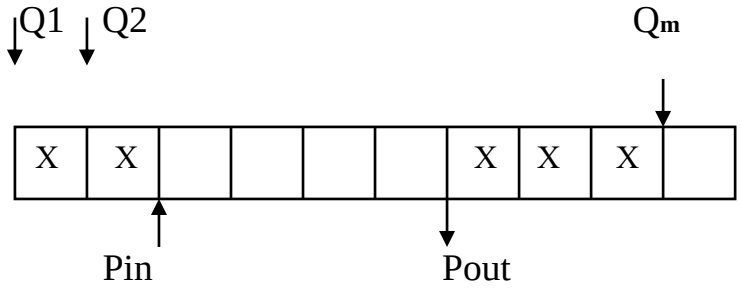

Рис. 2

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 случае кольцевой очереди может быть Pin > Pout, (т. е. «хвост» располагается дальше от начала массива, чем «голова») и Pin < Pout, (т. е. «хвост» ближе к началу массива,

чем «голова»). При равенстве Pin = Pout, означающем, что «хвост» и «голова» совпали, очередь может быть как пустой, так и заполненной (т. е. K=0 или K=m).

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. Граничные адреса очереди являются параметрами физической структуры. Эта структура обычно дополняется дескриптором очереди, в котором хранятся имя очереди, адреса нижней и верхней границ очереди, адреса включения и исключения элементов (Pin и Pout), а также максимальное количество заполненных элементов и описание (тип) элемента очереди.

Реализация очереди в виде линейного списка. Большинство проблем, возникающих при реализации очереди в виде массива, устраняется при реализации очереди на основе односвязного линейного списка, каждый элемент которого содержит информационное поле и поле с указателем «вперед» (на следующий элемент). В этом случае в статической памяти можно либо хранить адрес начала и конца очереди, либо – адрес начала очереди и количество элементов. Исходное состояние очереди: Pout = Pin – пустой указатель, K = 0.

Основные операции над очередью, как над абстрактным типом данных: **включение** элемента в очередь и **исключение** элемента из очереди.

Как было сказано выше, включение элемента в очередь осуществляется по адресу Pin, указатель «вперед» у включаемого элемента – пустой. В этом случае размер очереди ограничивается только объемом доступной памяти. Исключается элемент, находящийся по адресу Pout. Указатель Pout при этом переместится на предыдущий элемент. При исключении элемента необходимо следить за опустошением очереди, а также не забывать освобождать память под удаляемый элемент.

При физической реализации очереди в виде односвязного линейного списка дескриптор будет отличаться отсутствием верхней границы очереди. В этом случае объем очереди ограничивается только объемом доступной оперативной памяти.

В процессе моделирования очереди может оказаться, что при последовательных запросах на выделение и освобождении памяти под очередной элемент выделяется не та память, которая была только что освобождена при удалении элемента. Участки свободной и занятой памяти могут чередоваться, т.е. может возникнуть **фрагментация** памяти.

Задание

Система массового обслуживания состоит из обслуживающих аппаратов (ОА) и очередей заявок двух типов, различающихся временем прихода и обработки. Заявки поступают в очереди по случайному закону с различными интервалами времени (в зависимости от варианта задания), равномерно распределенными от начального значения (иногда от нуля) до максимального количества единиц времени. В ОА заявки поступают из «головы» очереди по одной и обслуживаются за указанные в задании времена, распределенные равновероятно от минимального до максимального значений (все времена – **вещественного типа**).

Требуется смоделировать процесс обслуживания первых 1000 заявок первого типа, выдавая после обслуживания каждых 100 заявок первого типа информацию о текущей и средней длине каждой очереди и о среднем времени пребывания заявок каждого типа в очереди. В конце процесса необходимо выдать на экран общее время моделирования, время простоя ОА, количество вошедших в систему и вышедших из нее заявок первого и второго типов.

Очередь необходимо представить в виде **статического** вектора и списка. Все операции должны быть оформлены подпрограммами. **Алгоритм для реализации задачи один, независимо от формы представления очереди.** Необходимо сравнить эффективность различного представления очереди по времени выполнения программы и по требуемой памяти. При реализации очереди списком нужно проследить, каким образом происходит выделение и освобождение участков памяти, для чего по запросу пользователя необходимо выдать на экран адреса памяти, содержащие элементы очереди при добавлении или удалении очередного элемента.

Длительности обработки заявок и интервалы между их приходом (единицы времени - е.в.) – случайные равномерно распределенные числа ***вещественного типа*** в указанном диапазоне (например, от t1 до t2). Для получения случайной величины в указанном диапазоне значений можно использовать генератор случайных чисел с параметром функции от 0 до 1, возвращающий значение t. Используя линейное преобразование time = (t2–t1)*t + t1 можно получить необходимое время. Имея время прихода первой заявки, допустим, $t_{прихода1}$= time, можно определить время прихода следующей заявки как $t_{прихода2} = t_{прихода1}$+ time.

Следовательно, время прихода 1000 заявок будет равно:

$$t_{прихода} = \sum_{i=1}^{i=1000} t_{приходаi}$$

Среднее время прихода или обработки заявок можно подсчитать как среднее арифметическое временного диапазона (например, если t1 = 1, а t2 = 4, то среднее время будет равно $t_{ср}$ = (1+4)/2 = 2.5). Аналогичным образом рассчитывается время окончания обработки заявок и время выхода заявок из системы.

В один и тот же момент времени одна заявка может придти в очередь, а другая – начать обрабатываться или выйти из системы. Допустим, пришла заявка первого типа, а в этот же момент заявка второго типа закончила обработку и вышла из системы, т.е. $t_{прихода1} = t_{обработки2} = t_{выхода2}$. Таким образом, модельное время прихода заявки первого типа в систему и модельное время выхода заявки второго типа из системы совпали. Процессы обработки одной заявки и прихода другой заявки идут одновременно, т.е. **протекают во времени параллельно, а не последовательно!**

Указания к выполнению работы

Рассогласование между средними ожидаемыми временами и временами, полученными в моделирующей программе должно быть **не больше 2–3%.**

При проверке работы программы по входу заявок ожидаемое время моделирования равно среднему интервалу между приходом заявок, умноженному на количество вошедших заявок. Если есть две очереди, то проверка ведется по каждой из очередей.

При проверке работы программы по выходу заявок ожидаемое время обработки в ОА должно быть равно среднему времени обработки заявки, умноженному на количество обработанных заявок. Если есть две очереди и один аппарат, то это будет сумма времен обслуживаний заявок каждого типа.

Если среднее время обработки заявок больше среднего интервала между их поступлением, то очередь будет расти, и время моделирования будет определяться временем обработки заявок. При этом количество вошедших заявок должно быть равно времени моделирования, деленному на средний интервал между приходом заявок.

Если средний интервал между поступлением заявок больше или равен среднему времени их обработки, то длина очереди стабилизируется (может стремиться к нулю, к единице или к какой-то другой величине) и время моделирования будет определяться временем прихода заявок. При этом ОА может работать с простоем, если время окончания обработки заявки будет меньше, чем время прихода очередной

заявки. Тогда время простоя будет определяться разницей этих времен, т. е. $t_{простоя} = t_{обработки} - t_{прихода.}$ Общее время простоя ОА будет равно сумме простоя перед обслуживанием каждой заявки, если простой имел место.

Если есть одна очередь и два аппарата, соединенных последовательно, то время моделирования будет определяться временем обработки в наиболее загруженном аппарате.

Необходимо привести тестирование программы. Рассмотрим пример.

Программа представляет собой модель очереди. Допустим, никакие данные в программу не вводятся.

В течение всего времени работы в очередь поступают заявки по случайному закону со временами в интервале от 0 до 10 е. в., обслуживаются заявки по случайному закону со временами в интервале от 0 до 2 е. в.

Каждая заявка после обслуживания в аппарате возвращается в «хвост» очереди для повторной обработки и после 4-х циклов обслуживания покидает систему.

После выхода из системы обработки каждой 100-й заявки выводится текущее состояние очереди, а именно количество вошедших и вышедших заявок, текущая длина очереди и среднее время пребывание заявок в очереди.

Система завершает свою работу по выходу из нее 1000 заявок. На экране при этом отображается общее время моделирования, время простоя ОА, количество вошедших и вышедших из очереди заявок.

После окончания работы следует проверить правильность результатов моделирования.

Теоретически рассчитываем время моделирования. Время обработки заявки лежит в интервале от 0 до 2 е.в., значит, среднее значение времени обработки одной заявки 1 е.в. Так как каждая заявка обрабатывается 4 раза до выхода из системы, то среднее время обработки одной заявки будет: 1*4=4 е.в., а общее время обработки 1000 заявок будет: 1000*4 = 4000 е.в. Для прихода 1000 заявок, если каждая из них приходит в среднем за 5 е.в., потребуется 5000 е.в. Следовательно, ОА работает с простоем и время моделирования будет определяться временем прихода заявок, т.е. оно должно быть равно 5000 е.в. Время простоя будет равно разнице между временем обработки заявок и временем их обслуживания: 5000 – 4000 = 1000 е.в.

На практике при выполнении программы получены следующие результаты:

время моделирования − 5054.458 е.в.

время простоя составило −- 987.971 е.в.

вошедших заявок было − 1018

вышедших заявок – 1000

аппарат сработал – 4019 раз

время работы ОА составило t = 1*4019 = 4019 е.в.

Сравниваем практические результаты с теоретическими расчетами.

Для проверки правильности работы системы по входу общее время моделирования делим на время прихода одной заявки:

5054.458 / 5 = 1010.8 заявок

Таким образом, определяем предполагаемое количество вошедших заявок. Фактически их пришло 1018, т.е. погрешность составляет

|100%(1018-1010.8)/1010.8| = 0.712%

Проверим правильность работы системы по выходу. За время моделирования аппарат работал 4019 е.в. и простаивал – 987.971 е.в.. Время моделирования должно было составить: 5006.971 е.в., а оно составило – 5054.458. Получаем погрешность:

100%(5054.458−5006.971)/5006.971 = 0.9484%.

Можем вычислить погрешность по-другому. Расчетное время работы равно 5000е.в., а фактическое − 5054.458. То есть, погрешность составит

100%(5054.458-5000)/5000 = 1,08916%.

Таким образом, проверка работы системы по входным и выходным данным показала, что погрешность работы системы составляет не более 1.1%, что удовлетворяет поставленному условию.

Интерфейс программы должен быть понятен неподготовленному пользователю. При разработке интерфейса программы следует предусмотреть:

- возможность изменения **входных** данных с экрана: времен прихода и обработки заявок, вероятностей обработки и возврата заявок;
- наличие пояснений при выводе результата;
- вывод процента погрешности работы программы.

Кроме того, нужно вывести на экран время выполнения программы при реализации очереди списком и массивом, а также требуемый при этом объем памяти.

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.

При реализации очереди списком необходимо тщательно следить за освобождением памяти при удалении элемента из очереди. При этом по запросу

пользователя должны выдаваться на экран адреса памяти, содержащие элементы очереди при добавлении или удалении элемента очереди. Следует проверять, происходит ли при этом фрагментация памяти (выделение непоследовательных адресов память).

Содержание отчета

Отчет по лабораторной работе должен содержать выводы о том, в каких случаях применение какой структуры данных более целесообразно для реализации очереди, какую выгоду дает та или иная реализация. Выводы должны быть подтверждены результатами (оформленными таблицей) числовых сравнений расходования памяти и времени выполнения схожих операций.

Сравните эффективность реализации FIFO и LIFO в ваших лабораторных работах 4 и 5 и сделайте соответствующий вывод.

В отчете по лабораторной работе также должны быть даны ответы на следующие вопросы:

1. Что такое FIFO и LIFO?
2. Каким образом, и какой объем памяти выделяется под хранение очереди при различной ее реализации?
3. Каким образом освобождается память при удалении элемента из очереди при ее различной реализации?
4. Что происходит с элементами очереди при ее просмотре?
5. От чего зависит эффективность физической реализации очереди?
6. Каковы достоинства и недостатки различных реализаций очереди в зависимости от выполняемых над ней операций?
7. Что такое фрагментация памяти, и в какой части ОП она возникает?
8. Для чего нужен алгоритм «близнецов».
9. Какие дисциплины выделения памяти вы знаете?
10. На что необходимо обратить внимание при тестировании программы?
11. Каким образом физически выделяется и освобождается память при динамических запросах?

Отчет должен быть представлен в электронном или печатном виде.

Список рекомендуемой литературы

1. *Вирт Н.* Алгоритмы и структуры данных: Пер. с англ. СПб.: Невский диалект, 2001, С. 69–71, 205–235.

2. *Ахо А.,. Хопкрофт Д., Ульман Д.* Структуры данных и алгоритмы.: Пер. с англ. М.: Издат. дом «Вильямс», 2000 , С. 58–61.
3. *Кормен Т., Лейзерсон Ч., Ривест Р.*, Алгоритмы: построение и анализ: Пер. с англ. М.: МЦНМО, 2001, С. 194 –198.