РАБОТА № 1. СТЕКИ. ОЧЕРЕДИ. ДЕКИ

Цель работы – изучить основные способы реализации стеков, очередей и деков, выполняемые над ними операции; получить практические навыки программирования задач с использованием стеков, очередей, деков.

Внимание! Для первых шести пунктов подготовки к работе необходимо разработать только алгоритмы (на уровне псевдокода), а не программы. Программная реализация требуется только для седьмого пункта.

При программной реализации в данной работе ЗАПРЕЩЕНО использовать имеющиеся в среде программирования классы (библиотеки) для работы со стеками, очередями и деками.

Подготовка к работе

1. Реализовать стек с использованием последовательного (массив) и связного (динамические списковые структуры) распределений элементов. Для каждого способа представления разработать алгоритмы, реализующие выполнение следующих операций над стеками:

- *Empty*(*S*) – проверка на наличие элементов (проверка пустоты стека);
- *Push*(*S*, *x*) – в вершину стека *S* помещает элемент *x*;
- *Pop*(*S*) – удаляет элемент из вершины стека *S* и возвращает его значение (не определена для пустого стека);
- *StackTop*(*S*) – возвращает значение элемента в вершине стека *S* без его удаления (не определена для пустого стека).

2. Реализовать очередь с использованием последовательного (массив) и связного (динамические списковые структуры) распределений элементов. Для каждого способа представления разработать алгоритмы, реализующие выполнение следующих операций над очередями:

- *EmptyQueue*(*Q*) – проверка на наличие элементов (проверка пустоты очереди);
- *InsQueue*(*Q*, *x*) – помещает элемент *x* в конец очереди *Q*;
- *DelQueue*(*Q*) – удаляет элемент из начала очереди *Q* и возвращает его значение (не определена для пустой очереди).

3. *Дек* (от англ. *deque – double ended queue*) – структура данных, представляющая из себя список элементов, в который добавление новых элементов и удаление существующих производится с обоих концов. Эта структура поддерживает как FIFO, так и LIFO, поэтому на ней можно реализовать как стек, так и очередь. В первом случае нужно использовать только методы головы или хвоста, во втором – методы *push* и *pop* двух разных концов. Дек можно воспринимать как двустороннюю очередь.

Реализовать дек с использованием последовательного (массив) и связного (динамические списковые структуры) распределений элементов. Для каждого способа представления разработать алгоритмы, реализующие выполнение следующих операций над деками:

- *Empty*(*D*) – проверка на наличие элементов (проверка пустоты дека);
- *PushBack*(*D, x*) – помещает элемент *x* в конец дека *D*;
- *PushFront*(*D, x*) – помещает элемент *x* в начало дека *D*;
- *PopBack*(*D*) – удаляет элемент из конца дека *D* и возвращает его значение (не определена для пустого дека);
- *PopFront* (*D*) – удаляет элемент из начала дека *D* и возвращает его значение (не определена для пустого дека).

4. Разработать метод поддержания в одном линейном массиве двух стеков, при котором ни один из стеков не переполняется до тех пор, пока весь массив не будет заполнен. При этом стек никогда не перемещается внутри массива на другие позиции. Написать подпрограммы, реализующие операции *Push*1, *Push*2, *Pop*1, *Pop*2, *Empty*1 и *Empty*2, манипулирующие обоими стеками, учитывая, что стеки растут навстречу друг другу.

5. Реализовать очередь на базе двух стеков. Определить время работы (асимптотику) операций с очередью.

6. Реализовать стек на базе двух очередей. Определить время работы (асимптотику) стековых операций.

7. Разработать алгоритмы и программы решения задач в соответствии с заданными вариантами (***из каждого блока по 1 задаче***), используя одно из представлений стеков, очередей или деков.

Номер варианта N определяется по следующей формуле (i – номер студента по списку группы, k – число задач в блоке, **mod** – остаток целочисленного деления)

$$N = (i - 1) \textbf{ mod } k + 1.$$

Варианты заданий

Блок 1.

1. Преобразование инфиксной формы записи арифметического выражения в постфиксную. Например, выражение в инфиксной форме $((a + b) \times c - (d - e)) \times (f + g)$ в постфиксной форме имеет вид $ab+c\times de- -fg+\times$.

2. Преобразование инфиксной формы записи арифметического выражения в префиксную. Например, выражение в инфиксной форме $((a + b) \times c - (d - e)) \times (f + g)$ в префиксной форме имеет вид $\times-\times+abc-de+fg$. Для упрощения перевода рекомендуется инфиксную строку анализировать справа налево, префиксную строку также создавать справа налево.

3. Преобразование постфиксной формы записи выражения в инфиксную.

4. Преобразование префиксной формы записи выражения в инфиксную.

5. Преобразование префиксной формы записи выражения в постфиксную.

6. Преобразование постфиксной формы записи выражения в префиксную.

7. Вычисление значения арифметического выражения (без переменных), записанного в инфиксной форме, без преобразования в другую.

8. Вычисление значения арифметического выражения (без переменных), записанного в постфиксной форме.

9. Вычисление значения арифметического выражения (без переменных), записанного в префиксной форме.

10. Длинное целое число представлено односвязным списком, старший разряд – первый элемент списка. Реализовать умножение на целое число N ($0 \le N \le 9$).

11. Имеется произвольный текст, сбалансированный по круглым скобкам. Требуется для каждой пары соответствующих открывающей и закрывающей скобок определить номера их позиций в тексте, упорядочив пары в порядке возрастания номеров позиций открывающих скобок. Кроме того, должна проверяться сбалансированность скобок с указанием, какие скобки не имеют пары. Например, для текста $((a + b) \times c – d \times \min(f, g)) \times h$ результат будет выглядеть следующим образом: (1, 20), (2, 6), (15, 19);

12. Имеется произвольный текст, сбалансированный по круглым скобкам. Требуется для каждой пары соответствующих открывающей и закрывающей скобок определить номера их позиций в тексте, упорядочив пары в порядке возрастания номеров позиций закрывающих скобок. Кроме того, должна проверяться сбалансированность скобок с указанием, какие скобки не имеют пары. Например, для текста $((a + b) \times c – d \times \min(f, g)) \times h$ результат будет выглядеть следующим образом: (2, 6), (15, 19), (1, 20).

Блок 2.

1. Дана произвольная последовательность чисел. За один просмотр этой последовательности сформировать новую последовательность в следующем порядке: сначала – все числа, меньшие a, затем – все числа из отрезка $[a, b]$, и наконец – все остальные числа, сохраняя исходный взаимный порядок в каждой из этих трех групп чисел (a и b – заданные числа, $a < b$).

2. Содержимое текста, разделенного на строки, преобразовать в новый текст, перенося при этом в конец каждой строки все входящие в нее цифры (с сохранением исходного взаимного порядка, как среди цифр, так и среди остальных символов строки).

3. Имеется список $S = \{s_1, s_2, \ldots, s_n\}$ имен людей, находящихся в родственных отношениях (все имена в списке различны) и матрица отношений $D = [d_{ij}]$ размера $n \times n$, в которой $d_{ij} = 1$, если человек с именем i является отцом человека с именем j, и $d_{ij} = 0$ в противном случае. Необходимо сформировать список всех потомков человека с заданным именем в следующем порядке: сначала – имена всех его детей, затем – всех его внуков, затем – всех правнуков и т. д.

4. В игре в пьяницу карточная колода раздается поровну двум игрокам. Далее они вскрывают по одной верхней карте, и тот, чья карта старше, забирает себе обе вскрытые карты, которые кладутся под низ его колоды. Тот, кто остается без карт – проигрывает. Для простоты будем считать, что все карты различны по номиналу, а также, что самая младшая карта побеждает самую старшую карту. Игрок, который забирает себе карты, сначала кладет под низ своей колоды карту первого игрока, затем карту второго игрока (то есть карта второго игрока оказывается внизу колоды). Смоделировать игру и определить, кто выигрывает. В игре участвует 10 карт, имеющих значения от 0 до 9, большая карта побеждает меньшую, карта со значением 0 побеждает карту 9.

5. На железнодорожном вокзале работают K касс, однако очередь в них всего одна. Обслуживание происходит следующим образом. Изначально, когда все кассы свободны, первые K человек из очереди подходят к кассам. Остальные ждут своей очереди. Как только кто-нибудь будет обслужен и соответствующая касса освободится, следующий человек из очереди подходит к этой кассе. Так продолжается до тех пор, пока не будут обслужены все клиенты. Задано время T_i обслуживания каждого клиента i. Определите время, за которое будут обслужены все клиенты.

Содержание отчета

1. Цель работы.
2. Информация в соответствии с подготовкой к работе.
3. Скриншот с результатами работы программ.
4. Выводы.

Примечания:

1). В отчет можно не включать алгоритмы, приведенные в методических материалах доцента кафедры вычислительной техники Павлова Л. А. (учебные пособия, методические указания, презентации лекций и т. п.).

2). В отчете не надо описывать всю теорию. Достаточно привести заголовок алгоритма, описание используемой алгоритмом структуры данных, непосредственно сам алгоритм (псевдокод), его краткое описание (входные и выходные данные, назначение важных переменных). **Обязательно указывать временную сложность алгоритма!**

Ниже приведен пример описания алгоритмов.

2. Реализация очереди

Определены следующие процедуры и функции:

- функция *EmptyQueue* (*Q*) возвращает **true**, если очередь *Q* пуста, **false** в противном случае, сложность *O*(1);
- процедура *InsQueue*(*Q*,*x*) помещает элемент *x* в конец очереди *Q*, сложность *O*(1);
- функция *DelQueue* (*Q*) удаляет элемент из начала очереди *Q* и возвращает его значение (не определена для пустой очереди), сложность *O*(1).

2.1. Массив

Массив из m компонентов $Q_0, Q_1, Q_2, \ldots, Q_{m-1}$, свернутый в кольцо в соответствии с операцией **mod**. Считается, что Q_0 следует за Q_{m-1}. Переменная f – указатель позиции в массиве, расположенной непосредственно перед началом очереди, переменная r –указатель ее конца, т. е. очередь состоит из элементов $Q_{f+1}, Q_{f+2}, \ldots, Q_r$.

EmptyQueue(*Q*)	*InsQueue*(*Q*, *x*)	*DelQueue*(*Q*)
EmptyQueue:= $f = r$	$r \leftarrow (r+1)$ **mod** m **if** $r = f$ **then**// переполнение **else** $Q_r \leftarrow x$	$f \leftarrow (f+1)$ **mod** m $x \leftarrow Q_f$

2.2. Связный список

Используется связанный список с заголовком, узел которого состоит из поля *info*, содержащего элемент очереди, и поля связи *next* для указания элемента, идущего после данного. При таком представлении очереди поле *info* заголовка не используется, поле *next* заголовка указывает на первый элемент очереди, указатель *f* – на заголовок списка, указатель *r* – на последний элемент очереди. Пустая очередь состоит из одного узла-заголовка, на который ссылаются указатели *f* и *r*, и значение его поля *next* равно Λ.

EmptyQueue(*Q*)	*InsQueue*(*Q*, *x*)	*DelQueue*(*Q*)
$EmptyQueue = f = r$	$new(next(r))$ $r \leftarrow next(r)$ $info(r) \leftarrow x$ $next(r) \leftarrow \Lambda$	$x \leftarrow info(next(f))$ $l \leftarrow f$ $f \leftarrow next(f)$ $dispose(l)$

Здесь алгоритмы представлены с помощью редактора формул Word. Если Вам это кажется слишком трудоемким, можно записывать алгоритмы с помощью моноширинного шрифта Courier New. В качестве оператора присваивания вместо стрелки можно использовать его обозначение из используемого языка программирования. Также для доступа к элементу массива вместо нижнего индекса можно указывать индекс в квадратных (или круглых) скобках.

Например, вместо алгоритма в редакторе формул

$r \leftarrow (r+1) \bmod m$
if $r = f$
  then // переполнение
  else $Q_r \leftarrow x$

можно записать его шрифтом Courier New:

```
r := (r+1) mod m
if r = f
  then //переполнение
  else Q[r] := x
```