ЛАБОРАТОРНАЯ РАБОТА №8

По дисциплине: Алгоритмы и структуры данных
Тема работы: **Простейшие структуры данных**
Цель работы: выполнить программную реализацию стека и очереди с помощью массива, описать необходимые операции.
Количество часов: 2

Содержание работы:

1. Описание структуры данных
2. Описание главной функции
3. Добавление функций основных операций
4. Анализ времени работы основных операций для разных наборов входных чисел
5. Выводы

Методические указания по выполнению

В некоторых задачах бывает удобно ограничить интерфейс обычного способа хранения данных. Порядок взаимодействия при этом определяется по определенным разработчиком правилам. В работе рассматривается реализация простейших структур данных – стеков и очередей – с помощью массива целочисленного либо структурного типа.

Простейшие структуры данных

Описание структуры данных

Стеки и очереди представляют собой динамические множества.

Первым из ***стека*** удаляется элемент, который был помещен туда последним. В стеке реализуется стратегия «последним вошел – первым вышел» (*last-in, first-out – LIFO*). Пример стека: стопка тарелок, где новую тарелку можно поместить только сверху, и взять можно тоже только сверху.

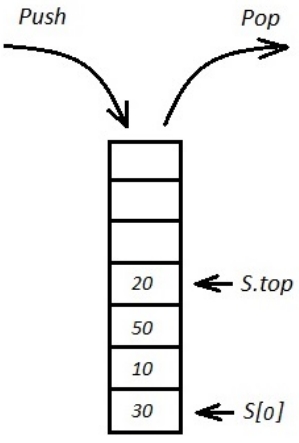
Аналогично в очереди всегда удаляется элемент, который содержится в очереди дольше других. В очереди реализуется стратегия «первым вошел – первым вышел» (*first-in, first-out – FIFO*). Пример очереди: очередь к врачу или к банкомату, где каждый новый человек становится в конец очереди, а первым из нее уходит первый пришедший и простоявший дольше всех.

Здесь показано, как реализовать эти обе структуры данных с помощью обычного массива.

Стеки (Stack)

Основные операции над стеком (рис. 1):

- операция вставки *Push* (запись нового элемента в стек);
- операция удаления элемента *Pop* (снятие со стека).



Стек из n элементов можно реализовать с помощью массива $S[n]$. Массив содержит атрибут ***top*** – индекс последнего помещенного в стек элемента. Стек состоит из элементов $S[0..top]$, где $S[0]$ – элемент на дне стека, $S[top]$ – элемент на его вершине.

Если $top=-1$, стек не содержит ни одного элемента и является ***пустым (empty)***. Протестировать, есть ли в стеке элементы, можно с помощью следующей операции.

```
Stack_Empty (S)
1. if top == -1
2.       return TRUE
3. else return FALSE
```

Если выполняется попытка снятия элемента с пустого стека, говорят, что он ***опустошается (underflow)***, что обычно приводит к ошибке. Если значение *top* равняется *n*, то стек ***переполняется (overflow)***.

```
Push(S,x)
```

```
1. top = top + 1
2. S[top] = x
```

Pop(S)

```
1. if STACK_EMPTY(S)
2.       error "underflow"
3. else top = top - 1
      4.      return S[top+1]
```

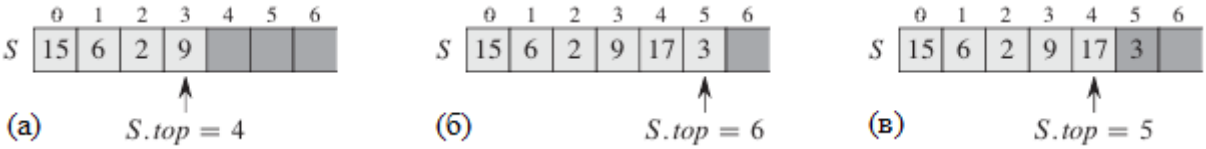
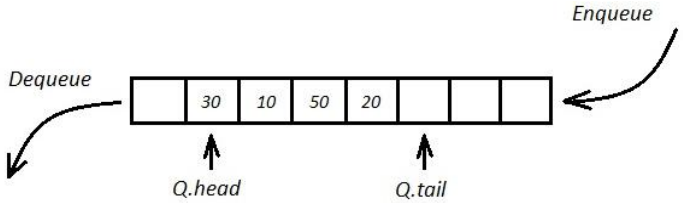


Рис. 2. Реализация стека с помощью массива. Элементы стека только в светлых ячейках. (а) стек состоит из 4-х элементов; (б) стек после вызовов Push(S,17), Push(S,3); (в) стек после вызова функции Pop(S) – элемент 3 есть в массиве, но не принадлежит стеку

Очереди (Queue)

Основные операции над очередью (рис. 3):

- операция вставки *Enqueue* (помещение нового элемента в очередь);
- операция удаления *Dequeue* (выведение элемента из очереди).



У очереди имеется ***голова (head)*** и ***хвост (tail).*** Когда элемент помещается в очередь, он занимает место в ее хвосте. Из очереди всегда выводится элемент, который находится в ее голове.

Q[*0..n*] – массив, реализующий очередь из *n* элементов, но содержащий *n+1* элемент, где ***head*** –указывает на головной элемент очереди, а ***tail*** – указывает на место в очереди, куда будет добавляться новый элемент. Одна ячейка резервируется дополнительно для эффективной работы функций (закрашена на рисунке). Элементы очереди расположены в ячейках *Q*[*head*], *Q*[*head] +1, … Q[tail]-1*, которые циклически замкнуты, т.е. ячейка 0 следует сразу же после ячейки *n* в циклическом порядке. Очередь пуста, если *head = tail,* а также изначально, когда выполняется соотношение: *head = 0, tail = 0.*

Если очередь пуста, то при попытке удалить из нее элемент, происходит ошибка опустошения. Если *head = tail + 1* или *head = 0* и *tail = n*, то очередь заполнена, и попытка добавить в нее элемент приводит к ее переполнению.

В функциях проверка ошибок опустошения и переполнения не проводится.

Enqueue(Q,x)

```
1. Q[tail] = x
2. if tail == n - 1      // n – длина Q
3.    tail = 0
4. else tail = tail + 1
```

Dequeue(Q)

```
1. x = Q[head]
2. if head == n          // n – длина Q
3.    head = 0
4. else head = head + 1
5. return x
```

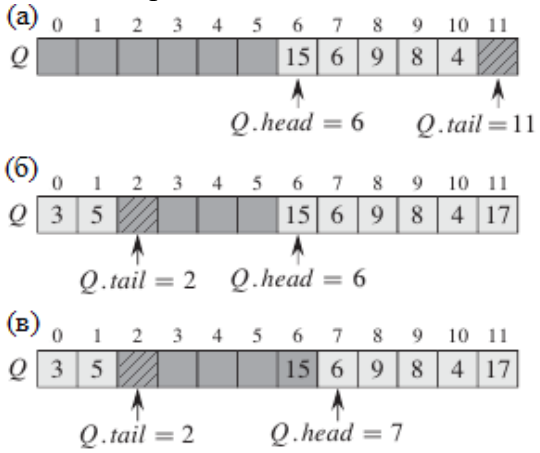


Рис. 4. Реализация очереди с помощью массива. Элементы очереди только в светлых ячейках. (а) очередь из 5 элементов в позициях Q[6..10]; (б) очередь после вызовов Enqueue(Q,17), Enqueue(Q,3), Enqueue(Q,5); (в) очередь после вызова Dequeue(Q) – головным элементом является ключ 6.

Все операции со стеками и очередями, функции которых приведены, выполняются за **время** *O(1)*.

Задачи

1. Исправьте все функции так, чтобы корректно обрабатывались ошибки переполнения и опустошения.
2. Покажите результат воздействия на изначально пустой стек S, хранящийся в массиве S[1..7], каждой из операций Push(S,15), Push(S,9), Pop(S), Push(S,7), Push(S,14), Push(S,1), Pop(S), Pop(S), Push(S,3), Push(S,18).
3. Покажите результат воздействия на изначально пустую очередь Q, хранящуюся в массиве Q[1..7], каждой из операций Enqueue(Q,21), Enqueue(Q,8), Enqueue(Q,15), Dequeue(Q), Enqueue(Q,6), Enqueue(Q,1), Dequeue(Q), Dequeue(Q), Enqueue(Q,27), Enqueue(Q,10).

Пособия и инструменты

1. MS Visual Studio 2010 … 2017.
2. Data Structure Visualizations. [Электронный ресурс] – Режим доступа: http://www.cs.usfca.edu/~galles/visualization/java/download.html.

Вопросы для защиты лабораторной работы

1. Дайте асимптотическую оценку выполнения каждой операции.

Литература

1. Кормен Т.Х. Алгоритмы: построение и анализ, 3-е изд. : Пер. с англ. / Т.Х. Кормен, Ч.И. Лейзерсон, Р.Л. Ривест, К. Штайн. – М.: Издательский дом «Вильямс», 2013. – 1328 с.
2. Algorithms and Data Structures with implementations in Java and C++ [Electronic Resource]. – URL: http://www.algolist.net/.
3. Data Structure Visualizations / David Galles, Department of Computer Science // University of San Francisco [Electronic Resource]. – URL: http://www.cs.usfca.edu/~galles/visualization/Algorithms.html.
4. Анимированные визуализации структур данных / VISUALGO [Electronic Resource]. – URL: http://ru.visualgo.net/.
5. Platform Algomation [Electronic Resource]. – URL: http://www.algomation.com/.
6. A computer science portal for geeks GeeksforGeeks [Electronic Resource]. – URL: http://www.geeksforgeeks.org/fundamentals-of-algorithms/.