ЛАБОРАТОРНАЯ РАБОТА №9

По дисциплине: Алгоритмы и структуры данных
Тема работы: **Связанные списки**
Цель работы: выполнить программную реализацию двусвязного списка.
Количество часов: 2

Содержание работы:

1. Описание структуры данных
2. Описание главной функции
3. Добавление функций основных операций
4. Выводы

Методические указания по выполнению

Описание структуры данных

Связанные списки (linked list) – это структура данных, в которой объекты расположены в линейном порядке. Однако, в отличие от массива, в котором этот порядок определяется индексами, порядок в связанном списке определяется указателями на каждый объект.

Списки могут быть разных видов:

- однократно и дважды (многократно) связанными,
- отсортированными и неотсортированными,
- кольцевыми и некольцевыми.

Каждый элемент ***дважды связанного списка*** (*double linked list*) L – это объект с одним полем ключа *key* и двумя полями-указателями: *next* (следующий) и *prev* (предыдущий). Для каждого элемента списка *x* указатель *next* указывает на следующий элемент связанного списка, а указатель *prev* – на предыдущий.

Если у элемента нет предшественника, он является ***головным*** в списке и, следовательно, *prev = nil.* Если у элемента нет последующего, он является последним или ***хвостовым*** в списке и, следовательно, *next = nil*. И, наконец, *head* указывает на первый элемент списка. Если *head = nil,* то список пуст.

Если ***список однократно связанный (однонаправленный***) (*singly linked*), то указатель *prev* в его элементах отсутствует.

Если ***список отсортирован*** (*sorted*), то его линейный порядок соответствует линейному порядку его ключей; в этом случае минимальный элемент находится в голове списка, а максимальный – в его хвосте. Если список не отсортирован, то его элементы могут располагаться в произвольном порядке.

Если ***список кольцевой*** (*circular list*), то указатель *prev* его головного элемента указывает на хвост, а указатель *next* хвостового на головной элемент.

В дальнейшем рассматриваются *неотсортированные и дважды связанные списки.*

Поиск в связанном списке

Функция *List_Search(L, k)* позволяет найти в списке *L* первый элемент с ключом *k* путем линейного поиска, и возвращает указатель на найденный элемент. Если элемент с заданным ключом *k*, отсутствует, возвращается значение *nil.*

```
List_Search(L, k)
1. x = head
2. while x ≠ nil и x.key ≠ k
3.     x = x.next
4. return x
```

Поиск с помощью рассмотренной функции в списке, состоящем из *n* элементов, в наихудшем случае выполняется в течение **времени** Θ(n), т.к. может понадобиться просмотр всего списка.

Вставка в связанный список

Пусть имеется ключ *k*, которому предварительно присвоено значение. Рассмотрим функцию *List_Insert()*, которая создает элемент списка *x*, инициализируя его поле *key* данным ключом *k*. Затем элемент *x* вставляется перед первым элементом существующего списка и становится головным.

```
List_Insert(L,k)
1. создать элемент x
2. x.key = k
3. x.next = head
4. if head ≠ nil
5.     head.prev = x
6. head = x
7. x.prev = nil
```

Время работы функции *List_Insert()* равно *O(1)*.

Удаление из связанного списка

Пусть имеется ключ *k*, которому предварительно присвоено значение. Рассмотрим функцию *List_Delete()*, которая создает элемент списка *x*. Затем осуществляет поиск элемента в списке с заданным ключом *k* с помощью функции *List_Search()*. Результаты поиска присваиваются элементу *x*. Если элемент был найден, то *x* является указателем на удаляемый элемент. Иначе элемент с ключом *k* отсутствует, и возвращается значение *nil*.

```
List_Delete(L,k)
1. создать элемент x
2. x = List_Search(L,k)
3. if (x ≠ nil)
4.     if x.prev ≠ nil
5.         x.prev.next = x.next
6.     else head = x.next
7.     if x.next ≠ nil
8.         x.next.prev = x.prev
```

Время работы функции *List_Delete()* равно *O(1)*. Однако если требуется поиск элемента с заданным ключом, время работы станет равным (в худшем случае) $\Theta(n)$.

Задачи

1. Реализуйте функцию для вывода элементов списка на экран.
2. Добавьте в список фиктивный элемент *nil*. Измените все операции с учетом данного элемента.

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 издание. : 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/.