Алгоритмы и структуры данных

ТЕМА #2: АТД - СЛОВАРЬ.

КЛАСС: ОДНОСВЯЗНЫЙ УПОРЯДОЧЕННЫЙ СПИСОК

Динамические множества.

Словарь – динамическое множество, поддерживающее операции:

добавить элемент;

удалить элемент;

проверить, принадлежит ли (найти)…

Словарь *не может иметь записи с одинаковыми ключами* (словами).

Способ реализации множества зависит от операций, которые должны поддерживаться.

Один и тот же АТД может быть реализован с использованием разных структур данных.

Словарь – список, дерево поиска, красно-черное дерево, хеш-таблица…

Словарь, реализованный с использованием связного списка.

Будем формировать список, элементы которого упорядочены по возрастанию (нет записей с одинаковыми ключами).

Операции в динамических множествах

SEARCH(S, k) – Запрос, возвращает указатель на элемент x заданного множества S для которого ***key[x] = k***, или значение NIL, если в множестве S такой элемент отсутствует.

INSERT(S, x) – Модифицирующая операция, которая добавляет в заданное множество S один элемент, на который указывает x. Предполагается, что выполнена инициализация всех полей x.

DELETE (S, x) - Модифицирующая операция, которая удаляет из заданного множества S один элемент, на который указывает x. (x – указатель на элемент)

MINIMUM(S) – Запрос к упорядоченному множеству, возвращает указатель на минимальный элемент.

MAXIMUM(S) – …

SUCCESSOR (S, x) - Ближайший, превышающий x. (Ключ)

PREDECESSOR (S, x) - Ближайший меньший x.

Словарные операции:

	Операции	Реализация	Примечание. Возвращаемыек значения.
1	Создать пустой список	Конструктор	
2	Добавить слово	**Метод insert**	**public:** true, если добавлено; false, если слово уже есть в словаре
3	Найти слово	**Метод search**	**private:** адрес узла с искомым словом; nullptr, если слово не найдено. **public: ?** true, если слово найдено; false, если слово не найдено в слове.
4	Удалить слово	**Метод delete**	public: true, если слово найдено и удалено; false, если слово не найдено в словаре.
5	Удалить словарь	Деструктор	
6			
7			

search, insert

? Private / protected

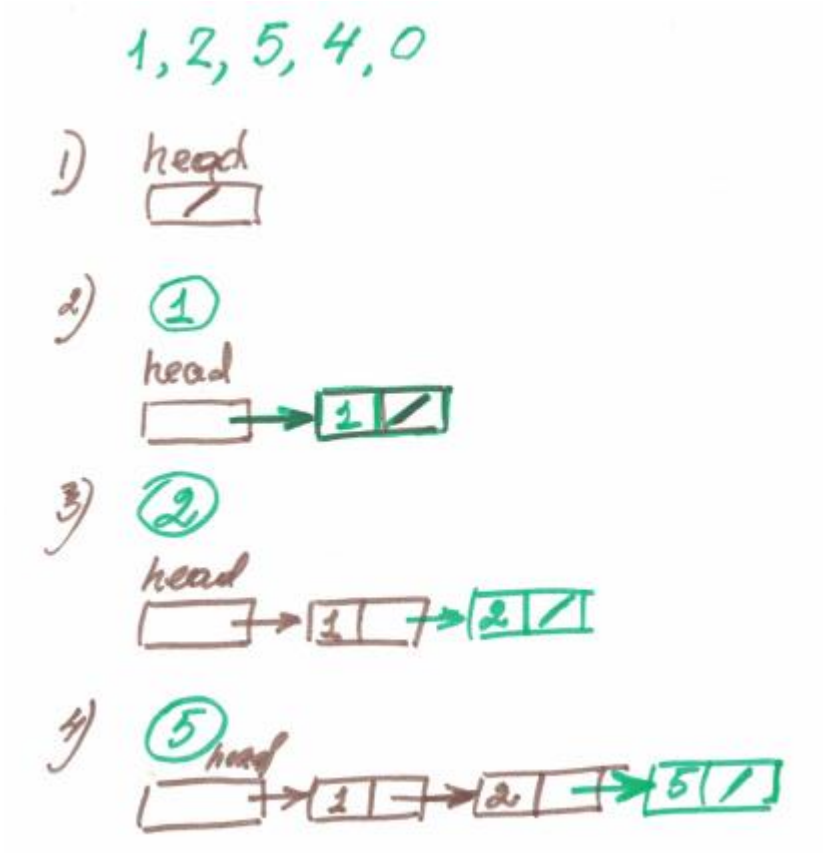
? Параметры? Возвращаемые значения

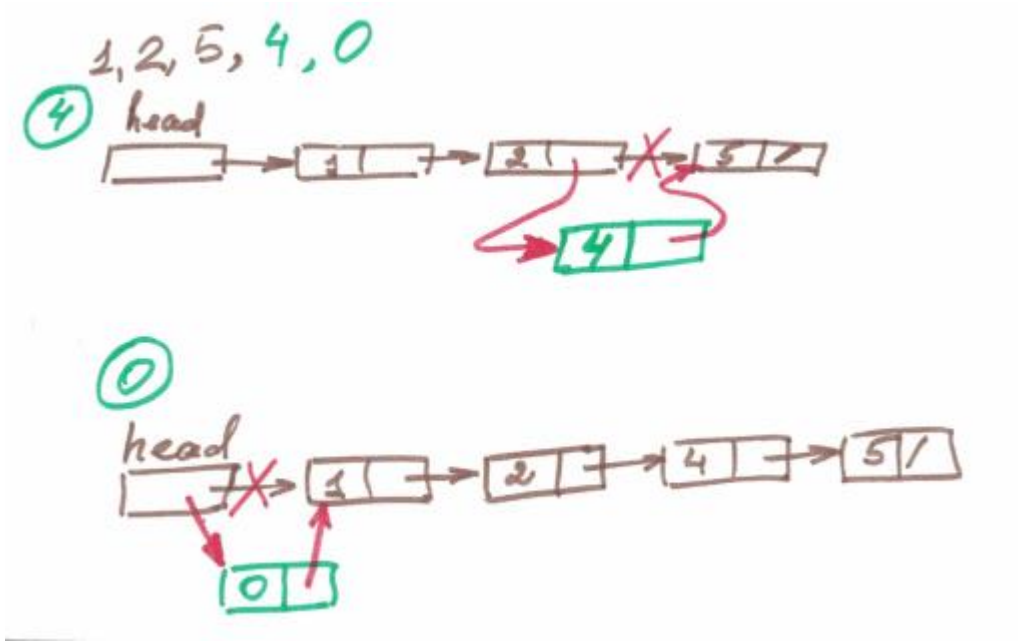
Словарные операции:

1) Создать «пустой» словарь
2) Добавить запись в словарь, обеспечивая лексикографическую упорядоченность и уникальность ключей (слов).
3) Найти слово в словаре.
4) Удалить слово из словаря.

Словарные операции:

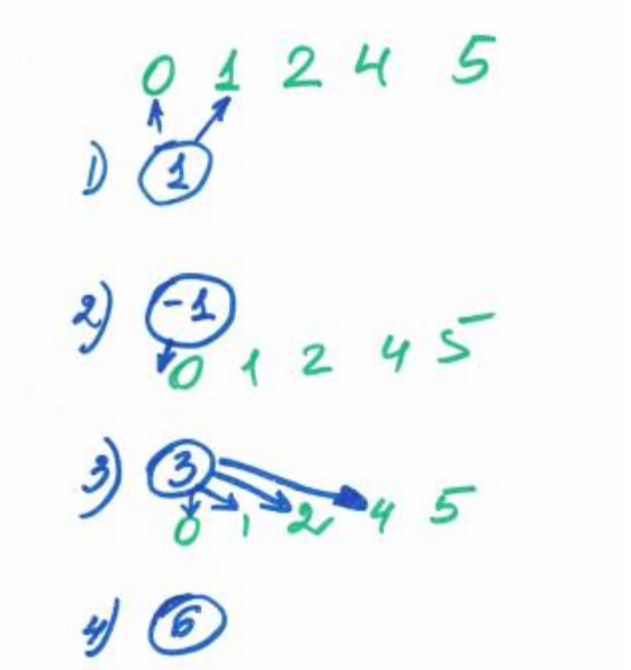
Добавить запись в словарь, обеспечивая лексико-графическую упорядоченность.





Словарные операции:

Найти слово в словаре.



Действия со словарями:

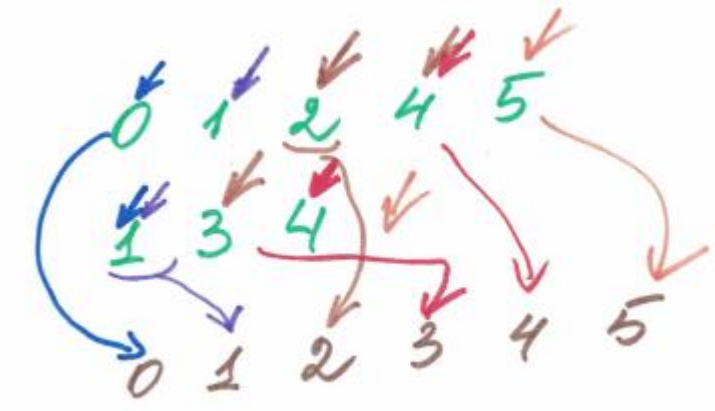
При выполнении действий необходимо учитывать лексикографическую упорядоченность и уникальность ключей (слов).

Первый словарь – объект, к которому применяется метод, второй – параметр метода.

1) *Объединение словарей*. Метод класса для добавления в первый словарь слов, содержащихся во втором. Элементы второго словаря, уже присутствующие в первом не добавлять; после выполнения операции второй словарь должен быть пустым. При выполнении задания нельзя копировать списки, копировать узлы списков. Для вставки узлов нужно корректировать ссылки.
2) *Вычитание словарей*. *Метод класса для* удаления из первого словаря слов, встречающихся во втором. *В процессе выполнения метода второй словарь не меняется.*
3) *Пересечение словарей*. Дружественная функция, формирующая новый словарь, содержащий слова, присутствующие одновременно в двух словарях. Исходные словари остаются без изменения.

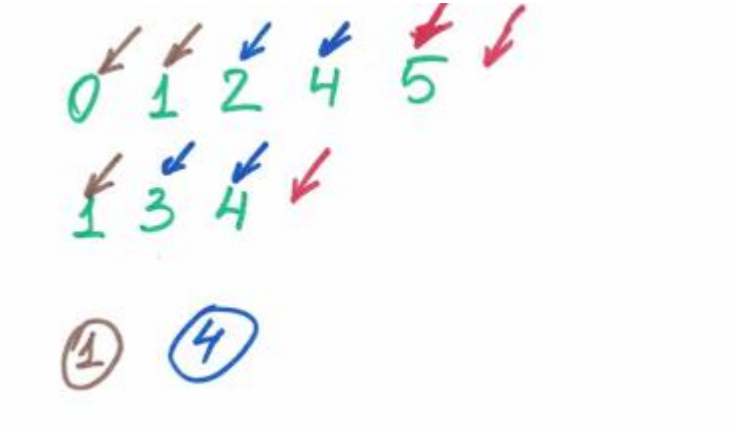
Действия со словарями:

Объединение словарей. Метод класса для добавления в первый словарь слов, содержащихся во втором. Элементы второго словаря, уже присутствующие в первом не добавлять; после выполнения операции второй словарь должен быть пустым. При выполнении задания нельзя копировать списки, копировать узлы списков. Для вставки узлов нужно корректировать ссылки.



Действия со словарями:

Пересечение словарей. Дружественная функция, формирующая новый словарь, содержащий слова, присутствующие одновременно в двух словарях. Исходные словари остаются без изменения.



Класс `SinglyOrderedList`:

односвязный список (упорядоченный)

1. Элемент списка (узел) - тип Node, в котором значение элемента списка item_ (key_), а для связи между элементами используется поле next__.

2. Тип Node может использоваться только в классе `SinglyOrderedList`

Класс SinglyOrderedList

```
// Тип  Node используется для описания элемента списка, связанного со
// следующим с помощью поля next_

  struct Node    // может использоваться только в классе SinglyOrderedList
  {
    int    item_;       // значение элемента списка (key_)
    Node *next_;        // указатель на следующий элемент списка
  };
```

Класс SinglyOrderedList t

```
 private:
 // Тип  Node используется для описания элемента списка, связанного со
 // следующим с помощью поля next_
 struct Node   // может использоваться только в классе SinglyOrderedList
 {
     int   item_;       // значение элемента списка
     Node *next_;       // указатель на следующий элемент списка
  // Конструктор для создания нового элемента списка.
  Node (int item, Node *next = nullptr ):
                item_(item) , next_(next))
  { }
 };
```

Класс SinglyOrderedList

```
class SinglyOrderedList
{
private:
struct Node            // только в классе SinglyOrderedList
{. . .
};

   Node* head_;        // первый элемент списка
   Node* tail_;        // последний элемент списка (! не обязательно)
   int count_;         // счетчик числа элементов (! не обязательно)
```

Класс SinglyOrderedList

```
public:

// Конструктор "по умолчанию" - создание пустого списка
SinglyOrderedList():
    head_( nullptr )
{  }
  . . .
}
```

```
SinglyOrderedList   testList1;  // создание пустого списка
```

Два набора методов:

1. private для работы с узлами – для разработчика класса
2. public работы со значениями (ключами) – для пользователя

Методы могут быть перегруженными, т.е. можно использовать одно и то же имя для private и public методов.

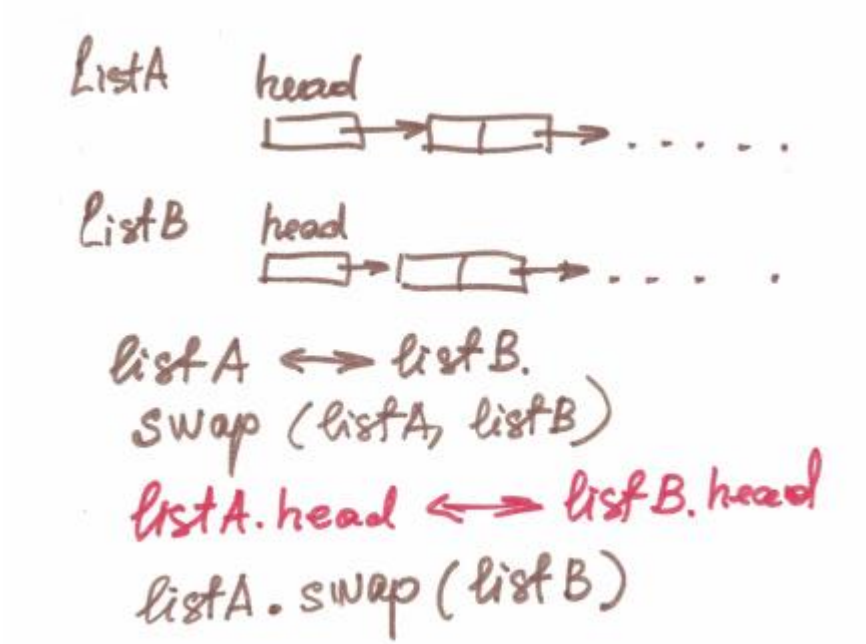
Класс SinglyOrderedList

```
SinglyOrderedList   ListA;
SinglyOrderedList   ListB;
. . .
```

Поменять местами ListA и ListB

Оставляем элементы на месте,

меняем только указатели head.



Примеры использования однонаправленного упорядоченного списка SinglyOrderedList

```
 SinglyOrderedList list;         // Создание пустого списка
   list.insert (2);          // Добавление элементов
   list.insert (3);
   list.insert (1);
   cout << list;                // Печать элементов
   cout << ( (list.search (1)) ?  "1 find " :  "1 not find ")<< endl;
   cout << ( (list.search (8)) ? " 8 find " : " 8 not find ")<< endl;
   list.delete (1);
   cout << list;
   cout << ( (list.search (1)) ?  "1 find " :  "1 not find ")<< endl;
   list.insert (0);
 // . . .
```

Задание размещено в файле ***Практика_#2_Задание***