

8 Поиск элемента в списке

8.1 Линейный поиск элемента в списке

Пусть a_i ($i = 0, \dots, n-1$) — элементы целочисленного списка; T — некоторое целое число. Выяснить, входит ли данное число T в список $a_0, \dots, a_{n-1}$ и, если входит, то каково значение p , для которого $a_p = T$.

Линейный поиск-I заключается в переборе всех элементов списка и поочередном их сравнении с искомым элементом (в условии цикла).

Условие окончания поиска:

<элемент найден> или <достигнут конец списка>

Результат поиска: индекс искомого элемента (если он есть) или информация об отсутствии искомого элемента.

Поиск элемента в списке

```
T = int(input('Введите искомый элемент: '))
# ----- поиск -----
i = 0
while (i < len(a) and a[i] != T):
    i += 1
# ----- анализ результата поиска -----
if i == len(a):
    print('Числа в списке нет')
else:
    print('Номер первого вхождения', i)
```

Поиск элемента в списке с использование метода index()

```
T = int(input('Введите искомый элемент: '))

if T not in a:
    print('числа в списке нет')
else:
    print('Номер первого вхождения', a.index(T))
```

Линейный поиск-II заключается в добавлении в конец массива фиктивного элемента, хранящего искомое значение T . Элемент будет найден всегда. В конце поиска следует проверить, что найденный элемент не фиктивный.

Условие окончания поиска: <элемент найден>

Поиск элемента в массиве (метод фиктивного элемента)

```
T = int(input('Введите искомый элемент: '))
a.append(T)      # фиктивный элемент
# ----- поиск -----
i = 0
while a[i] != T:
    i += 1
# ----- анализ результата поиска -----
if i == len(a) - 1:
    print('числа в списке нет')
else:
    print('номер первого вхождения', i)
```

8.2 Задачи-I

При решении задач из этого раздела используйте списки.

8.1. Элементы последовательности — буквы и цифры. Определить, верно ли, что последовательность содержит хотя бы одну цифру 5.

8.2. Элементы последовательности — натуральные числа. Определить, верно ли, что последовательность содержит хотя бы одно чётное число.

8.3. Элементы последовательности — символы латинского алфавита. Определить, верно ли, что последовательность содержит хотя бы одну гласную букву.

8.4. Элементы последовательности натуральные числа. Определить, все ли они различны.

8.5. Элементы последовательностей A и B — натуральные числа. Определить, верно ли, что каждый элемент последовательности A содержится в последовательности B .

 **8.6.** Элементы последовательности натуральные числа. Определить, есть ли среди элементов последовательностей числа с одинаковой последней цифрой.

 **8.7.** Элементы последовательностей натуральные числа. Определить, верно ли, что каждое число встречается не более двух раз.

 **8.8.** Определить, есть ли в последовательности простые числа.

 **8.9 (★).** Дана последовательность натуральных чисел. Определить количество вхождений в неё каждого числа.

Результат работы программы

```
Дано:  8  21  23  10  2  8  23  23
      2 - 1 шт.
      8 - 2 шт.
     21 - 1 шт.
     23 - 3 шт.
```

Указания. 1) Инициализировать список в программе. 2) Описать и вызвать функцию для печати элементов списка.

Ограничение. В программе использовать только один список.

8.3 Двоичный поиск элемента в массиве

Пусть a_i ($i = 0, \dots, n-1$) — элементы упорядоченного списка (пусть элементы упорядочены по возрастанию $a_0 < \dots < a_{n-1}$); T — некоторое число. Выяснить, входит ли данное число T в список $a_0, \dots, a_{n-1}$, и если входит, то каково значение p , для которого $a_p = T$.

Алгоритм деления пополам. Возьмем в качестве границ поиска индексы элементов $p = 0$ и $q = n - 1$.

$$\begin{array}{ccc} | & & | \\ p = 0 & & q = n - 1 \\ & s = \left\lfloor \frac{p+q}{2} \right\rfloor & \end{array}$$

До тех пор пока границы не совпадут, будем делить отрезок индексов пополам ($s = (p+q) // 2$) и шаг за шагом сдвигать границы индексов следующим образом:

$a_s < T$	p	q
да	$s+1$	прежнее
нет	прежнее	s

Поиск закончится, когда $p = q$. Искомый элемент будет найден, если выполняется условие

$$a[p] = T$$

Пример 8.1 (двоичный поиск элемента в упорядоченном списке). Продемонстрируем метод двоичного поиска для последовательности целых чисел A , упорядоченной по возрастанию

$$a_0 = 3 \quad a_1 = 7 \quad a_2 = 8 \quad a_3 = 10 \quad a_4 = 13 \quad a_5 = 15 \quad a_6 = 16 \quad a_7 = 18$$

$$a_8 = 21 \quad a_9 = 23 \quad a_{10} = 37 \quad a_{11} = 39 \quad a_{12} = 40 \quad a_{13} = 44 \quad a_{14} = 53.$$

Разыскиваемый элемент — 9

s	a[s]	a[s] < T	p	q	p ≠ q
			0	14	True
7	18	False	0	7	True
3	10	False	0	3	True
1	7	True	2	3	True
2	8	True	3	3	False

$a[3] = 9$?: нет $\Rightarrow$

элемента нет

Разыскиваемый элемент — 44

s	a[s]	a[s] < T	p	q	p ≠ q
			0	14	True
7	18	True	8	14	True
11	39	True	12	14	True
13	44	False	12	13	True
12	40	True	13	13	False

$a[13] = 44$?: да $\Rightarrow$

элемент найден, его номер 13

8.4 Задачи-II

8.10. Дан список целых чисел A , упорядоченный по возрастанию. Инициализировать список A в программе следующими значениями:

$$a_0 = 3 \quad a_1 = 7 \quad a_2 = 8 \quad a_3 = 10 \quad a_4 = 13 \quad a_5 = 15 \quad a_6 = 16 \quad a_7 = 18$$

$$a_8 = 21 \quad a_9 = 23 \quad a_{10} = 37 \quad a_{11} = 39 \quad a_{12} = 40 \quad a_{13} = 44 \quad a_{14} = 53.$$

Написать программу, реализующую алгоритм двоичного поиска элемента в списке.

Указание 1. В программе должны присутствовать операторы для построения таблицы, демонстрирующей ход поиска (здесь T — разыскиваемое значение):

Разыскиваемый элемент — ____

s	a[s]	a[s] < T	p	q	p ≠ q
...	...	...	...	...	...

a[____] = ____?: _____ (да/нет) ⇒
Результат поиска: ...

Указание 2. Программа должна быть написана в двух вариантах: простом (делим отрезок до совпадения границ) и оптимальном (делим отрезок до совпадения границ или до нахождения элемента).

8.11. Дан список целых чисел A , упорядоченный по убыванию. Инициализировать список A в программе следующими значениями:

$$a_0 = 53 \quad a_1 = 44 \quad a_2 = 40 \quad a_3 = 39 \quad a_4 = 37 \quad a_5 = 23 \quad a_6 = 21 \quad a_7 = 18$$

$$a_8 = 16 \quad a_9 = 15 \quad a_{10} = 13 \quad a_{11} = 10 \quad a_{12} = 8 \quad a_{13} = 7 \quad a_{14} = 3.$$

Написать программу, реализующую алгоритм двоичного поиска элемента в массиве.

Указание 1. В программе должны присутствовать операторы для построения таблицы, демонстрирующей ход поиска (здесь T — разыскиваемое значение):

Разыскиваемый элемент — ____

s	a[s]	a[s] < T	p	q	p ≠ q
...	...	...	...	...	...

a[____] = ____?: _____ (да/нет) ⇒
Результат поиска: ...

Указание 2. Программа должна быть написана в двух вариантах: простом (делим отрезок до совпадения границ) и оптимальном (делим отрезок до совпадения границ или до нахождения элемента).

9 Сортировка списков

Постановка задачи сортировки списка по возрастанию: дан числовой список $x_1, x_2, \dots, x_n$, элементы которого попарно различны; требуется переставить элементы списка так, чтобы после перестановки они были упорядочены в порядке возрастания: $x_1 < \dots < x_n$.

9.1 Сортировка выбором

Алгоритм сортировки выбором (I вариант):

- 1) найти элемент с **наибольшим значением**;
- 2) поменять значениями **найденный элемент** и **последний** в рассматриваемом подмассиве;
- 3) уменьшить на единицу количество просматриваемых элементов;
- 4) если (количество элементов для следующего просмотра больше единицы), то (повторить пункты, начиная с 1-го).

3	7	5	2	6	1
3	1	5	2	6	7
3	1	5	2	6	7
3	1	2	5	6	7
2	1	3	5	6	7
1	2	3	5	6	7

Сортировка выбором по возрастанию, I вариант

```

1 for i in range(len(a)-1, 0, -1):
2     # пусть макс. значение у I эл-нта в подмассиве
3     imax = 0
4     for j in range(1, i+1):
5         if a[j] > a[imax]:
6             imax = j
7     # обмен значений
8     a[imax], a[i] = a[i], a[imax]
```

Алгоритм сортировки выбором (II вариант).

- 1) найти элемент с наименьшим значением;
- 2) поменять значениями найденный элемент и первый в рассматриваемом подмассиве;
- 3) сдвинуть левую границу просматриваемых элементов вправо (будет на один элемент меньше);
- 4) если $\langle$ количество элементов для следующего просмотра больше единицы $\rangle$, то $\langle$ повторить пункты, начиная с 1-го $\rangle$.

Сортировка выбором по возрастанию, II вариант

```
1 for i in range(0, len(a)-1):
2     # пусть мин. значение у I эл-нта в подмассиве
3     imin = i
4     for j in range(i+1, len(a)):
5         if a[j] < a[imin]:
6             imin = j
7     # перестановка элементов
8     a[imin], a[i] = a[i], a[imin]
```

9.2 Сортировка обменом

Алгоритм сортировки обменом (метод пузырька). Последовательно сравниваются соседние элементы, и если выполняется неравенство

$$x_k > x_{k+1}$$

(сортировка по возрастанию), то поменять элементы x_k и x_{k+1} местами; в результате элемент с наибольшим значением окажется в конце массива.

Следующие просмотры списка применяются ко всем элементам, кроме последнего, и т. д.

Исходный список	3 7 5 2 6 1	Обозначения: сортируемый интервал списка отсортированный фрагмент списка.
Список после I шага	3 5 2 6 1 7	
Список после II шага	3 2 5 1 6 7	
Список после III шага	2 3 1 5 6 7	
Список после IV шага	2 1 3 5 6 7	
Список после V шага	1 2 3 5 6 7	

Сортировка обменом, I вариант

```

1 for i in range(0, len(a)-1):
2     for j in range(0, len(a)-i-1):
3         # если соседи не упорядочены
4         if a[j] > a[j+1]:
5             # перестановка соседей
6             a[j], a[j+1] = a[j+1], a[j]
```

II вариант реализации метода сортировки обменом — экономичный. Заведем переменную `sort` типа `boolean`. Если `sort = True`, значит перестановка на данном шаге была произведена, в противном случае `sort` остается равным `False`. Если на некотором шаге не было произведено ни одной перестановки, то сортировка прекращается.

Обычный вариант

1	3	2	5	6	7
1	2	3	5	6	7
1	2	3	5	6	7
1	2	3	5	6	7
1	2	3	5	6	7
1	2	3	5	6	7
1	2	3	5	6	7

Экономичный вариант

1	3	2	5	6	7
1	2	3	5	6	7
1	2	3	5	6	7
1	2	3	5	6	7

Экономичный вариант метода сортировки обменом выгоден для частично упорядоченных списков. Иначе, он будет выполняться даже дольше обычного варианта сортировки обменом.

Сортировка обменом, II вариант

```

1 sort = True    # список надо сортировать
2 i = 0
3 while sort:
4     sort = False
5     for j in range(0, len(a)-i-1):
6         # если соседи не упорядочены
7         if a[j] > a[j+1]:
8             sort = True # есть перестановка
9             # перестановка соседних элементов
10            a[j], a[j+1] = a[j+1], a[j]
11    i += 1

```

9.3 Сортировка включением

Алгоритм сортировки включением (простыми вставками): просматривать последовательно $a_1, \dots, a_{n-1}$ и каждый **новый элемент** a_i вставлять на подходящее место в уже **упорядоченную последовательность** $a_0, \dots, a_{i-1}$. Это место определяется последовательным сравнением a_i с упорядоченными элементами $a_0, \dots, a_{i-1}$.

14	7	3	1	11
7	14	3	1	11
3	7	14	1	11
1	3	7	14	11
1	3	7	11	14

Схематично каждая строка таблицы может быть представлена в виде

упорядоченные $a_0, \dots, a_{i-1}$ **элемент a_i** **неупорядоченные $a_{i+1}, \dots, a_{n-1}$**

Сортировка включением, I вариант

```

1 for i in range(1, len(a)):
2     y = a[i]
3     j = i - 1
4     while (j >= 0) and (y < a[j]):
5         a[j+1] = a[j]
6         j = j - 1
7     a[j+1] = y

```

9.4 Методы сортировки в Python

<code>A.sort([key=функция])</code>	Сортирует список на основе функции
------------------------------------	------------------------------------

Пример 9.1 (сортировка средствами языка Python). Приведем несколько примеров сортировок списков встроенными средствами языка Python.

Сортировка средствами Python

```
A = [13, 7, 8, 17, 13, 15, 16, 8, 21, 3, 37, 39, 40, 4, 13]
A.sort()
print(A)
A.sort(reverse=True)
print(A)
```

Результат работы программы

```
[3, 4, 7, 8, 8, 13, 13, 13, 15, 16, 17, 21, 37, 39, 40]
[40, 39, 37, 21, 17, 16, 15, 13, 13, 13, 8, 8, 7, 4, 3]
```

Python (сортировка по длине строки)

```
A = ['a', 'kotik', 'cccc', 'bbb']
A.sort(key=len)
print(A)
```

Результат работы программы

```
['a', 'bbb', 'cccc', 'kotik']
```

Средства Python (сортировка по алфавиту I символа)

```
A = ['ar', 'car', 'as', 'bar', 'abt', 'abg']
print('Исх. список: ', A)

# функция для сортировки списка в алф. порядке по I символу:
def sortByAlphabet1(inputStr):
    return inputStr[0] # ключ - I символ в каждой строке

A.sort(key = sortByAlphabet1) #
print('по I букве: ', A)
```

Результат работы программы

```
Исх. список:  ['ar', 'car', 'as', 'bar', 'abt', 'abg']
по I букве:   ['ar', 'as', 'abt', 'abg', 'bar', 'car']
```


Средства Python (сортировки по алфавиту)

```
A = ['ar', 'car', 'as', 'bar', 'abt', 'abg']
print('Исх. список: ', A)

# функция для сортировки списка в алф. порядке:
def sortByAlphabet(inputStr):
    return inputStr # Ключ - вся строка

A.sort(key = sortByAlphabet)
print('Алф. порядок: ', A)
```

Результат работы программы

```
Исх. список:  ['ar', 'car', 'as', 'bar', 'abt', 'abg']
Алф. порядок: ['abg', 'abt', 'ar', 'as', 'bar', 'car']
```

Пример 9.2 (случайная сортировка). Часто требуется элементы исходного списка наоборот перемешать, т.е. установить случайный порядок элементов.

В этом случае требуется использовать в качестве ключа функцию `random()`, которая выдает числа в случайном порядке от 0 до 1.

Средства Python (случайная сортировка)

```
from random import random

def randomOrder_key(elem):
    return random()
```

Примеры сортировки в случайном порядке

```
a = [1, 2, 3, 4, 5, 6]
print('Исх. список: ', a)

b = sorted(a, key = randomOrder_key)
print(b) # [5, 3, 2, 4, 6, 1]

c = sorted(a, key = randomOrder_key)
print(c) # [2, 1, 5, 4, 3, 6]
```

9.5 Задачи

 **9.1.** Сортировать по возрастанию элементы списка с индексами от i_1 до i_2 включительно. Метод сортировки: 1) выбором; 2) обменом; 3) включением.

Указание. В программе должны присутствовать операторы для построения таблицы, демонстрирующей ход сортировки.

 **9.2.** Используя любой алгоритм сортировки, расположить числа в списке в порядке убывания модулей.

Указание. В программе должны присутствовать операторы для построения таблицы, демонстрирующей ход сортировки.
