Модуль bisect в Python, вставка в отсортированный список

Вставка значений в отсортированный список

Модуль bisect обеспечивает поддержку вставки значений в отсортированный список, без необходимости сортировать этот список после каждой вставки. Для длинных списков элементов с дорогостоящими операциями сравнения это может быть улучшением по сравнению с более распространенным подходом. Модуль называется "*bisect*", потому что он использует базовый алгоритм деления пополам для выполнения своей работы. Исходный код может быть наиболее полезным в качестве рабочего примера алгоритма.

Примечания к производительности.

При написании чувствительного ко времени кода с использованием bisect.bisect() и bisect.insort() имейте в виду следующие мысли:

- Биссекция эффективна для поиска **диапазонов** значений. Для поиска определенных значений, более производительны словари.
- Функции insort() равны O(n), потому что на этапе логарифмического поиска преобладает этап вставки с линейным временем.
- Функции поиска не сохраняют состояния и отбрасывают результаты ключевых функций после их использования. Следовательно, если функции поиска используются в цикле, то ключевая функция может вызываться снова и снова для одних и тех же элементов массива. Если ключевая функция быстро не работает, то нужно подумать о том, чтобы обернуть ее функцией functools.cache(), чтобы избежать дублирования вычислений. В качестве альтернативы рассмотрите возможность поиска в массиве предварительно вычисленных ключей, чтобы найти точку вставки (как показано в разделе примеров ниже).

Поиск в отсортированных списках.

Вышеупомянутые функции bisect () полезны для поиска точек вставки, но могут быть сложными или неудобными в использовании для общих задач поиска. Следующие пять функций показывают, как преобразовать их в стандартные поисковые запросы для отсортированных списков:

```
def index(a, x):
    'Находит крайнее левое значение, точно равное to x'
    i = bisect_left(a, x)
    if i != len(a) and a[i] == x:
        return i
    raise ValueError

def find_lt(a, x):
    'Находит крайнее правое значение меньше, чем x'
    i = bisect_left(a, x)
    if i:
        return a[i-1]
    raise ValueError

def find_le(a, x):
    'Находит крайнее правое значение меньше или равно x'
    i = bisect_right(a, x)
    if i:
        return a[i-1]
    raise ValueError
```

```
def find_gt(a, x):
    'Находит крайнее левое значение больше, чем x'
    i = bisect_right(a, x)
    if i != len(a):
        return a[i]
    raise ValueError

def find_ge(a, x):
    'Находит крайний левый элемент больше или равен x'
    i = bisect_left(a, x)
    if i != len(a):
        return a[i]
    raise ValueError
```


Примеры использования модуля bisect:

Функция [bisect.bisect()](#) может быть полезна для поиска в числовой таблице. В этом примере используется bisect() для поиска буквенной оценки за экзамен (скажем) на основе набора упорядоченных числовых точек останова: 90 и выше - это "A", от 80 до 89 - "B" и так далее:

```
>>> def grade(score, breakpoints=[60, 70, 80, 90], grades='FDCBA'):
...     i = bisect(breakpoints, score)
...     return grades[i]

>>> [grade(score) for score in [33, 99, 77, 70, 89, 90, 100]]
# ['F', 'A', 'C', 'C', 'B', 'A', 'A']
```

Один из способов избежать повторного вызова функций для аргумента key - это поиск по списку предварительно вычисленных ключей, чтобы найти индекс записи:

```
>>> data = [('red', 5), ('blue', 1), ('yellow', 8), ('black', 0)]
# Или `key=operator.itemgetter(1)`
>>> data.sort(key=lambda r: r[1])
# Предварительно вычислим список ключей.
>>> keys = [r[1] for r in data]
>>> data[bisect_left(keys, 0)]
# ('black', 0)
>>> data[bisect_left(keys, 1)]
# ('blue', 1)
>>> data[bisect_left(keys, 5)]
# ('red', 5)
>>> data[bisect_left(keys, 8)]
# ('yellow', 8)
```

Содержание раздела:

Вверх