

Рекурсия. Алгоритмы сортировки

@pvavilin

13 августа 2022 г.

Что такое рекурсия?

Правильная рекурсия

Что такое стек вызовов?

Что такое стек вызовов?

Почему рекурсия это плохо

Recursion depth

Глубина рекурсии

Почему рекурсия это хорошо

Вариант задачи для рекурсии

Хвостовая рекурсия

Оптимизация хвостовой рекурсии и почему её нет в Python

Пример когда рекурсия помогает

Дополнительная литература

Зачем нужна сортировка данных

Глупая сортировка / сортировка дурака

Пузырьковая сортировка

Сортировка вставками

Сортировка вставками

Сортировка Шелла

Быстрая сортировка

Быстрая сортировка без рекурсии

Сортировка слиянием (Merge Sort)

Параллельный MergeSort

Сравнение алгоритмов сортировки

Устойчивость сортировки

Экзотические сортировки

Дополнительная литература

Вопросы-ответы

Приём в программировании, когда задача может быть разделена на несколько таких же, но проще, задач.

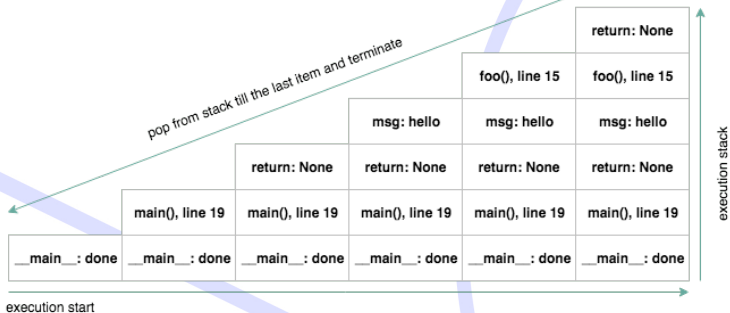
```
def pow(x, n):  
    # возведение числа в степень это  
    # умножение числа на число  
    # в степени n-1  
    if n == 0:  
        return 1  
    return x * pow(x, n-1)
```

```
def pow(x, n):  
    # хорошо бы проверить,  
    # что база достижима  
    assert n >= 0  
    # base case / база рекурсии  
    if n == 0:  
        return 1  
    # recursive case / шаг рекурсии  
    return x * pow(x, n-1)
```

```
def foo(msg):  
    print '{} foo'.format(msg)
```

```
def main():  
    msg = 'hello'  
    foo(msg)
```

```
if __name__ == '__main__':  
    main()
```



- ▶ стек вызовов растёт вместе с ростом глубины рекурсии
- ▶ можно попасть в бесконечную рекурсию и истратить всю память на стек вызовов

```
def inf_counter(x):  
    print(x)  
    return inf_counter(x+1)  
inf_counter(0)
```

```
import sys

print(sys.getrecursionlimit())
sys.setrecursionlimit(
    sys.getrecursionlimit() + 234
)
print(sys.getrecursionlimit())
1000
1234
```

Помогает описать решение задачи понятным языком

```
#  $n! = n * (n-1)$ 
def factorial(n):
    if n == 0:
        return 1
    return n * factorial(n-1)

print(factorial(5))
```

120

Рекурсия, не требующая действий с возвращённым результатом из шага рекурсии.

```
print(factorial(5))
```

- ▶ Интерпретаторы/компиляторы могут оптимизировать хвостовую рекурсию (Tail Call Optimization) и не делать записей в стек вызовов, а подменять переменные в стеке вызовов, таким образом код получится равнозначным обычному циклу

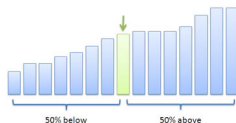
► Почему TCO нет и не будет в Python

Задача У вас есть вложенная структура данных и вы хотите просуммировать значения поля X во всех объектах этой структуры.

Решение задачи <https://gist.github.com/pimiento/bc4d5800f66541cb59ea388c1c3c263c>

- ▶ SICP
- ▶ СИКП
- ▶ Можем получить медианное значение

Median



- ▶ Можем использовать бинарный поиск
- ▶ Проще найти минимум/максимум
- ▶ Множество других применений


```

def sort_alg(l):
    while True:
        c = 0
        for i in range(len(l)-1):
            if l[i] > l[i+1]:
                l[i+1], l[i] = l[i], l[i+1]
            else:
                c += 1
        if c == (len(l) - 1): return l

print(sort_alg([1, 3, 2, 0]))
[0, 1, 2, 3]

```

► Эффективность **глупой сортировки**: $O(n^3)$

```

def sort_alg(l):
    for i in range(len(l)):
        for j in range(len(l[i+1:])):
            if l[j] > l[j+1]:
                l[j], l[j+1] = (
                    l[j+1], l[j]
                )

```

```
    return l
```

```
print(sort_alg([1, 3, 2, 0]))  
[0, 1, 2, 3]
```

► Эффективность **пузырьковой сортировки**: $O(n^2)$

```
def sort_alg(l):  
    for i in range(1, len(l)):  
        k = l[i]  
        j = i-1  
        print(f"i: {i}; k: {k}")  
        while j >= 0 and k < l[j]:  
            l[j+1] = l[j]  
            print(f"l: {l}")  
            j -= 1  
        l[j+1] = k  
        print(f"j: {j}; l: {l}")
```

```
d = [12, 11, 13, 5, 6]
```

- ▶ Эффективность **сортировки вставками**: $\mathcal{O}(n^2)$

Но! Эта сортировка эффективна если у вас уже частично отсортированные данные, так как пропускается этап перестановки данных.

- ▶ Дополнительно почитать
- ▶ Код
- ▶ на практике получается скорость работы быстрее $\mathcal{O}(n^2)$ но нет математических описаний как выбор последовательности дистанций влияет на алгоритмическую сложность.

```
def sort_alg(L):  
    if L:  
        return (  
            sort_alg(  
                [e for e in L[1:] if e < L[0]]  
            ) +  
            L[0:1] +  
            sort_alg(  

```


Comparison sorts						Other notes	
Name	Best	Average	Worst	Memory	Stable	Method	
Quicksort	$n \log n$	$n \log n$	n^2	$\log n$	No	Partitioning	Quicksort is usually done in-place with $O(\log n)$ stack space. ^{[5][6]}
Merge sort	$n \log n$	$n \log n$	$n \log n$	n	Yes	Merging	Highly parallelizable (up to $O(\log n)$ using the Three Hungarians' Algorithm). ^[7]
In-place merge sort	—	—	$n \log^2 n$	1	Yes	Merging	Can be implemented as a stable sort based on stable in-place merging. ^[8]
Introsort	$n \log n$	$n \log n$	$n \log n$	$\log n$	No	Partitioning & Selection	Used in several STL implementations.
Heapsort	$n \log n$	$n \log n$	$n \log n$	1	No	Selection	
Insertion sort	n	n^2	n^2	1	Yes	Insertion	$O(n + d)$, in the worst case over sequences that have d inversions.
Block sort	n	$n \log n$	$n \log n$	1	Yes	Insertion & Merging	Combine a block-based $O(n)$ in-place merge algorithm ^[9] with a bottom-up merge sort.
Timsort	n	$n \log n$	$n \log n$	n	Yes	Insertion & Merging	Makes $n-1$ comparisons when the data is already sorted or reverse sorted.
Selection sort	n^2	n^2	n^2	1	No	Selection	Stable with $O(n)$ extra space, when using linked lists, or when made as a variant of Insertion Sort instead of swapping the two items. ^[10]
Cubesort	n	$n \log n$	$n \log n$	n	Yes	Insertion	Makes $n-1$ comparisons when the data is already sorted or reverse sorted.
Shellsort	$n \log n$	$n^{4/3}$	$n^{3/2}$	1	No	Insertion	Small code size.
Bubble sort	n	n^2	n^2	1	Yes	Exchanging	Tiny code size.
Exchange sort	n^2	n^2	n^2	1	Yes	Exchanging	Tiny code size.
Tree sort	$n \log n$	$n \log n$	$n \log n$ (balanced)	n	Yes	Insertion	When using a self-balancing binary search tree.
Cycle sort	n^2	n^2	n^2	1	No	Selection	In-place with theoretically optimal number of writes.
Library sort	$n \log n$	$n \log n$	n^2	n	No	Insertion	Similar to a gapped insertion sort. It requires randomly permuting the input to warrant with-high-probability time bounds, which makes it not stable.
Patience sorting	n	$n \log n$	$n \log n$	n	No	Insertion & Selection	Finds all the longest increasing subsequences in $O(n \log n)$.
Smoothsort	n	$n \log n$	$n \log n$	1	No	Selection	An adaptive variant of heapsort based upon the Leonardo sequence rather than a traditional binary heap.
Strand sort	n	n^2	n^2	n	Yes	Selection	
Tournament sort	$n \log n$	$n \log n$	$n \log n$	$n^{[11]}$	No	Selection	Variation of Heapsort.
Cocktail shaker sort	n	n^2	n^2	1	Yes	Exchanging	A variant of Bubblesort which deals well with small values at end of list
Comb sort	$n \log n$	n^2	n^2	1	No	Exchanging	Faster than bubble sort on average.
Gnome sort	n	n^2	n^2	1	Yes	Exchanging	Tiny code size.
Odd-even sort	n	n^2	n^2	1	Yes	Exchanging	Can be run on parallel processors easily.

```
records = [
    (("A", "X"), ("B", 1)),
    (("A", "Y"), ("B", 1)),
    (("A", "X"), ("B", 2)),
```

```
]
records.sort(key=lambda x: x[0][1])
```

X,	1
X,	2
Y,	1

- Яндекс Практикум



designed by  freepik.com