Рекурсия. Алгоритмы сортировки

@pvavilin

13 августа 2022 г.

Outline

Что такое рекурсия?

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)
```

Правильная рекурсия

```
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)
```

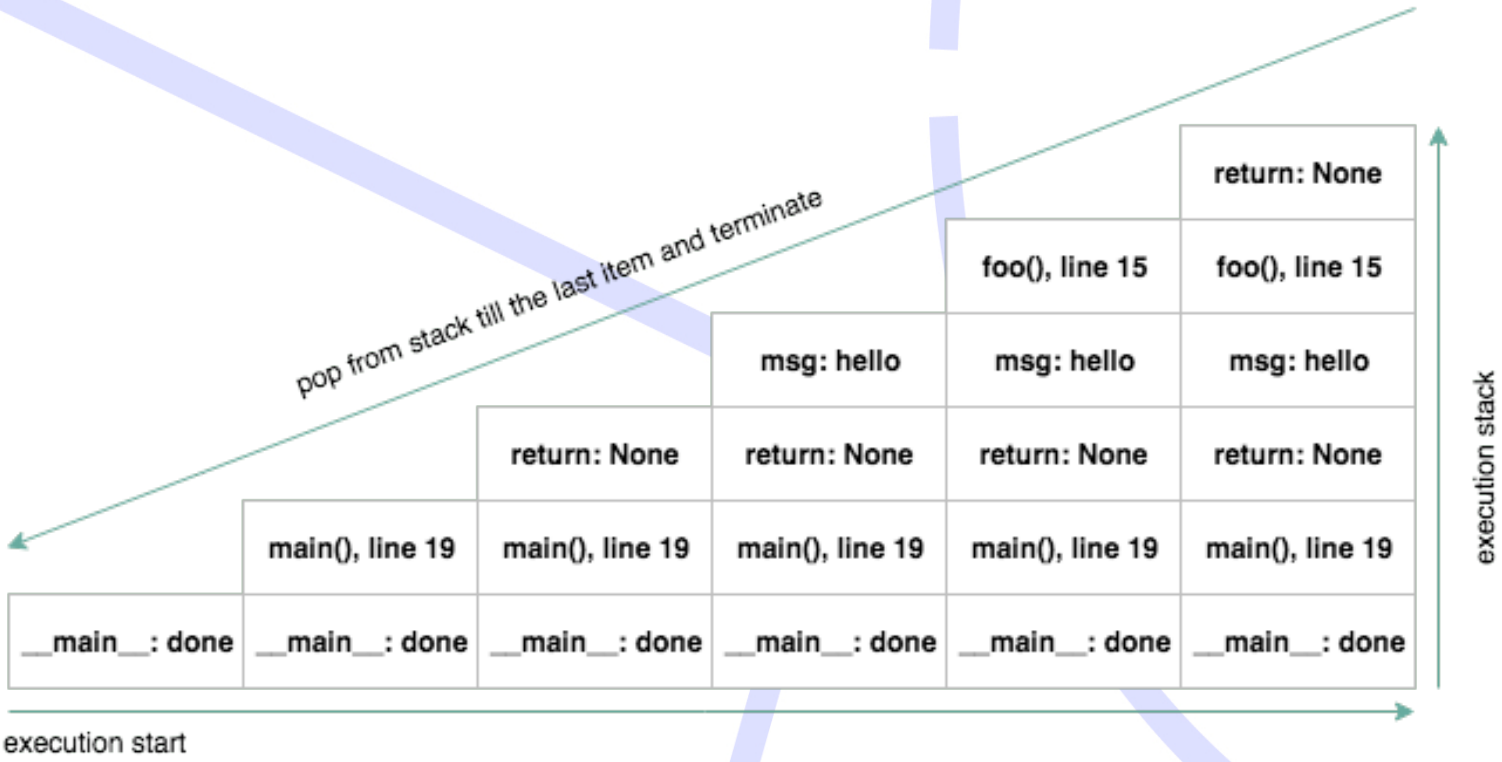
Что такое стек вызовов?

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

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

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

Что такое стек вызовов?



Почему рекурсия это плохо

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

Recursion depth

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

Глубина рекурсии

```
import sys

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

```
1000
1234
```

Почему рекурсия это хорошо

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
```

Вариант задачи для рекурсии

Попробуйте реализовать решение этой задачи без использования рекурсии 

Хвостовая рекурсия

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

```
def factorial(n, collected=1):
    if n == 0:
        return collected
    return factorial(n-1, collected*n)

print(factorial(5))
```

```
120
```

Оптимизация хвостовой рекурсии и почему её нет в Python

- 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

Пример когда рекурсия помогает

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

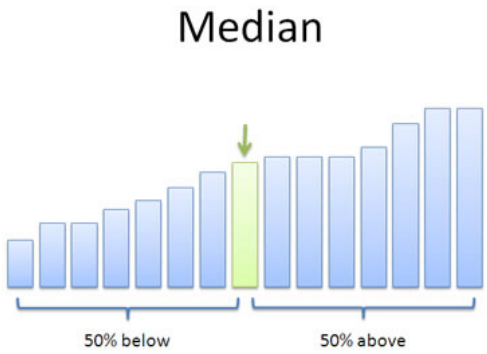
Решение задачи `https://gist.github.com/pimiento/bc4d5800f66541cb59ea388c1c3c263c`

Дополнительная литература

- SICP

Зачем нужна сортировка данных

- Можем получить медианное значение



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

Глупая сортировка / сортировка дурака

```
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: $\mathcal{O}(n^3)$

Пузырьковая сортировка

```
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: $\mathcal{O}(n^2)$

Сортировка вставками

```
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]
```

Сортировка вставками

- 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

Сортировка Шелла

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

Быстрая сортировка

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

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

```
[0, 1, 2, 3]
```

Быстрая сортировка без рекурсии

Сортировка слиянием (Merge Sort)

- Код
- мультик

Сортировка слиянием позволяет нам распараллелить процесс сортировки. Это очень эффективно на больших данных и широко используется в алгоритмах map/reduce.

Сравнение алгоритмов сортировки

Comparison sorts

Name	Best	Average	Worst	Memory	Stable	Method	Other notes
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.

Устойчивость сортировки

```
records = [
    (("A", "X"), ("B", 1)),
    (("A", "Y"), ("B", 1)),
    (("A", "X"), ("B", 2)),
]
records.sort(key=lambda x: x[0][1])
for r in records:
    print(f"{r[0][1]}, {r[1][1]}")
```

```
X, 1
X, 2
Y, 1
```

Экзотические сортировки

- Тыц
- Тыц

Дополнительная литература

- Пузырьковая сортировка и её улучшения
- Сравнение алгоритмов
- Т.Кормен, Ч.Лейзерсон, Р.Ривест, К.Штайн «Алгоритмы. Построение и анализ.»

Вопросы-ответы