Алгоритмы

31.10.2023

План-банан:

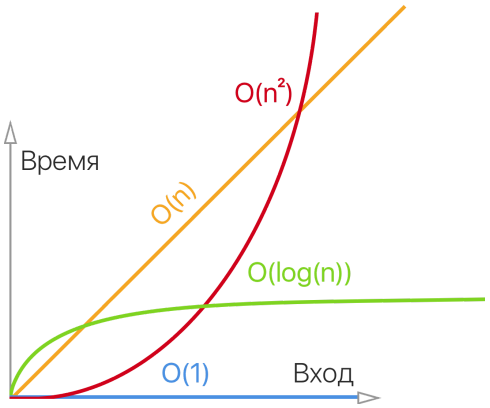
- Алгоритмическая сложность
- Бинарный поиск, вариации задач
- Дорешать задачки из раздела 1

Скорость работы алгоритмов

- Асимптотическая оценка - позволяет оценить с какой скоростью растёт число операций в конкретном алгоритме с ростом входных данных. Позволяет понять какой алгоритм лучше остальных при достаточно больших входных данных.
- Математическое определение

$$f(n) \in O(g(n))$$
$$\exists C \exists N_o : \forall n > N_o \quad f(n) \leq C * g(n)$$

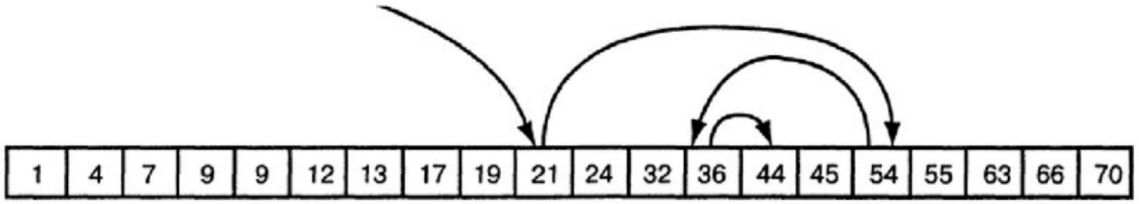
- Графички



- https://habr.com/ru/articles/188010/
- Важные моменты:
  - Анализ в худшем случае
  - До фиксированного множителя, несущественные слагаемые при стремлении `n → inf` могут быть отброшены:

        - `O(3n^2 + 2n + 1) → O(n^2)`
        - `O(nlogn + n) → O(nlogn)`
    - Оценка снизу, снизу и сверху, точная оценка сверху (`100x^3 + x^2 + 1000000` можно оценить и `O(n^4)`, но следует выбирать оптимальную оценку)
- Скрытая мультипликативная и аддитивная константа и не только константа (пример с сортировкой вставками и heap sort, важен размер входных данных)
- Оценка может быть для нескольких параметров - входов. Пример (действия с матрицами, строками и тд)
- Амортизационная сложность и сложность в среднем
    - Амортизационная сложность
        - усредняем по операциям
        - пример с вектором
        - лучше забыть про неё, не для всех случаев подходит
        - в лучшем, в худшем, в среднем, в худшем амортизационная - более сильная оценка, чем в среднем
    - Сложность в среднем
        - усредняем по входам
        - пример с хэш-таблицей, O(1) и O(n) в худшем
- Сложность по памяти:
    - пример с бинарным поиском (рекурсия `O(logN)` и без рекурсии (`O(1)`))

Бинарный (двоичный) поиск



Двоичный поиск элемента со значением 44

- Массив чисел должен быть отсортирован
- Хорош для случая, когда на одном массиве большое число запросов поиска
- TODO: оценить сложность алгоритма временную и по памяти в лоб и бинарного
- Переполнение в C:
  - `(l + r) / 2 → l + (r - l) / 2`
- TODO: зарешать

▼ Реализация из книжки

```
def binary_search(a, x):
    left = 0
    right = len(a) - 1
    while left < right:
        mid = (left + right) // 2
        if a[mid] < x:
            left = mid + 1
        else:
            right = mid
    return (left < len(a) and a[left] == x)
```

- TODO: зарешать upper/lower bound
- Библиотечные реализации
  - `bisect()`: позиция наименьшего элемента $ai > x$
  - `bisect_left()`: позиция наименьшего элемента $ai \geq x$
  - Примеры:

```
from bisect import bisect, bisect_left, bisect_right
a = [1, 1, 3, 5, 6, 7]
bisect(a, 6)
5
bisect_left(a, 10)
6
bisect_right(a, 10)
6
bisect_left(a, 0)
0
bisect_left(a, 2)
2
bisect_left(a, 4)
3
```

```
bisect_right(a, 1)
2
```

- TODO: домашка - рассмотреть примеры задачек из книжки

Задачки на перестановки

- Определение перестановки: перестановка длины n - набор чисел $1, 2, \ldots, n$, выписанный в некоторой последовательности
- Определение цикла для перестановки: пусть дана некоторая перестановка длины n. Выберем стартовый элемент - целое число от 1 до n - и будем двигаться, каждый раз переходя от числа i к числу, записанному в перестановке на i-м месте. Утверждается, что рано или поздно мы вернёмся к стартовому элементу; назовем циклом пройденный путь от стартовой точки вплоть до возврата в неё.
- Задача 1: Проверка перестановки
  - Дана последовательность натуральных чисел. Требуется вывести OK, если она является перестановкой, в противном случае BAD.
  - 1 3 2 - OK
  - 2 3 4 - BAD

  ▼ Реализация из книжки:

```
from collections import *
a = list(map(int, input().split()))
cnt = Counter(a)
ans = (max(a) == len(a) and
       max(cnt.values()) == 1)
print("OK" if ans else "BAD")
```

- Задача 2: Циклы перестановки
  - Дана перестановка. Вывести число циклов в ней
  - 1 3 2 - 2
  - 2 3 4 1 - 1

  ▼ Реализация из книжки

```
perm = map(int, input().split())
perm = [x - 1 for x in perm]
```

```
n = len(perm)
ans = 0
visited = [False] * n
for start in range(n):
  if visited[start]:
    continue
  ans += 1
  current = start
  while True:
    visited[current] = True
    current = perm[current]
    if current == start:
      break
print(ans)
```