Палиндромы

Палиндром

Палиндром - число, буквосочетание, слово или текст, одинаково читающееся в обоих направлениях.

Число палиндромов: медленное решение

Зададимся вопросом, сколько палиндромов содержится в строке s длины n.

Наивный алгоритм решает задачу за $O(n^3)$.

```
def count_palindromes(s: str) -> int:
    n = len(s)
    cnt = 0
    for i in range(n):
        for j in range(i, n):
            if s[i:j+1] == s[i:j+1][::-1]:
                cnt += 1
    return cnt
```

Число палиндромов: решение побыстрее

Если перебирать не границы палиндрома, а его центр, то можно сократить время работы до $O(n^2)$.

```
def count_palindromes(s: str) -> int:
    n = len(s)
    cnt = 0
    for i in range(n):
        for j in range(2): # четная и нечетная длина
            l, r = i, i + j
            while l >= 0 and r < n and s[l] == s[r]:
                cnt += 1
                l -= 1
                r += 1
    return cnt
```

Число палиндромов: быстрое решение

Мы что, зря учились считать хеши?

Построим полиномиальный хеш для строки s, а также для $reversed(s)$.

Тогда определение, является ли подстрока палиндромом займет $O(1)$.

```
def count_palindromes(s: str) -> int:
    n = len(s)
    cnt = 0
    for i in range(n):
        for j in range(2): # четная и нечетная длина
            d = binary_search(i, j, s)  # максимальная длина палиндрома
            cnt += d
    return cnt
```

Алгоритм Манакера

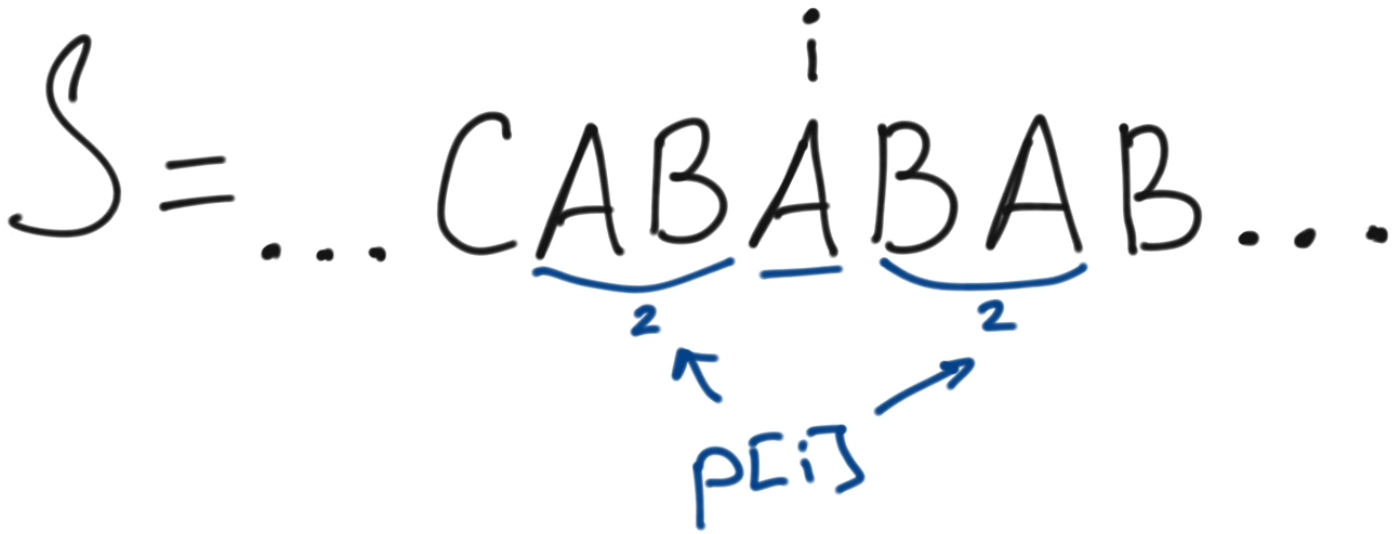
(очень быстрое решение)

Радиус палиндрома

Для простоты будем рассматривать палиндромы нечетной длины.
Рассуждения для палиндромов четной длины аналогичны.

Пусть $p[i]$ - радиус самого длинного палиндрома с центром в i.

То есть $p[i] = \arg\max_k S[i-k : i+k+1] == S[i+k : i-k-1 : -1]$



Алгоритм Манакера: идея

Будем последовательно строить значения $p[0]$, $p[1]$, $p[2]$, ...

- Назовем p-блоком пару $(i, j) : p[i] = j - i$.
- Пусть $right$ - самая правая граница j среди всех построенных p-блоков.
- Также $left$ - левая граница соответствующая $right$.

...погодите

Алгоритм Манакера: идея

Будем последовательно строить значения $p[0], p[1], p[2], ...$

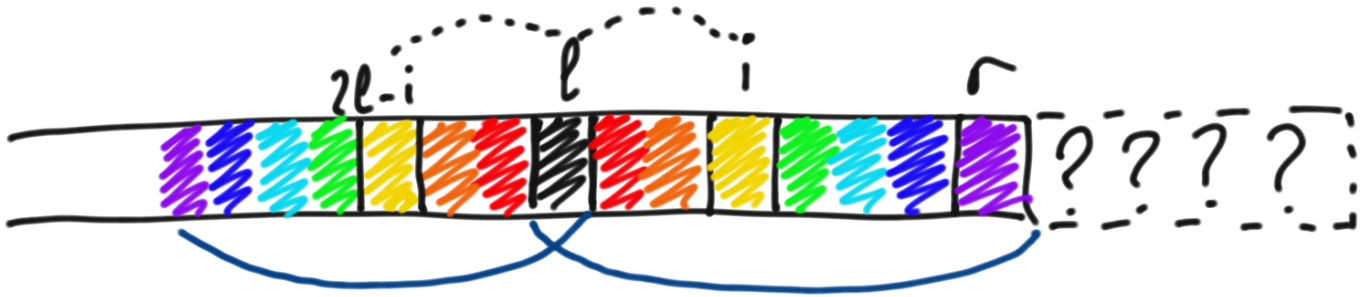
- Назовем p-блоком пару $(i, j) : p[i] = j - i$.
- Пусть $right$ - самая правая граница j среди всех построенных p-блоков.
- Также $left$ - левая граница соответствующая $right$.

Идея алгоритма совпадает с идеей построения z-функции!

Алгоритм Манакера

Будем последовательно строить значения $p[0], p[1], p[2], ...$

- Назовем p-блоком пару $(i, j) : p[i] = j - i.$
- Пусть $right$ - самая правая граница j среди всех построенных p-блоков.
- Также $left$ - левая граница соответствующая $right.$
- Если $i < right$, то $p[i] \geq \min(p[2l - i], right - i + 1).$
  Далее уточняем перебором.
- Иначе ($i \geq right$) придется считать $p[i]$ честно.



Алгоритм Манакера

Подсчет $p[i]$ для палиндромов **нечетной** длины:

```
def manaker(s: str) -> List[int]:
    n = len(s)
    p = [0] * n
    right = 0
    left = 0
    for i in range(n):
        if i < right:
            p[i] = min(p[2 * left - i], right - i + 1)
        else:
            p[i] = 1
        while i - p[i] >= 0 and i + p[i] < n and s[i - p[i]] == s[i + p[i]]:
            p[i] += 1
        if i + p[i] - 1 > right:
            right = i + p[i] - 1
            left = i
    return p
```

Алгоритм Манакера: анализ

Аналоично алгоритму построения z-функции, алгоритм Манакера работает за $O(n)$.

Построение массива p для палиндромов четной длины выполняется аналогично. ~~ну, ок, чуть больше возни с индексами~~

Таким образом, задача поиска всех палиндромов в строке s решается за $O(n)$:

```
def count_palindromes(s: str) -> int:
    n = len(s)
    p_even, p_odd = manaker(s)
    return sum(p_even) + sum(p_odd) + n
```