

Разбор задачи 10. Regular Expression Matching

Данилов Кирилл Алексеевич
Группа Б9124-02.03.01МО

Постановка задачи

Даны строка s и шаблон p . Необходимо определить, соответствует ли строка s шаблону p , где:

- Символ '.' соответствует любому одному символу
- Символ '*' соответствует нулю или более вхождениям предыдущего элемента

Примеры:

- $s = "aa", p = "a" \rightarrow \text{false}$
- $s = "aa", p = "a*" \rightarrow \text{true}$
- $s = "ab", p = ".*" \rightarrow \text{true}$
- $s = "aab", p = "c * a * b" \rightarrow \text{true}$

Подробное описание алгоритма

1. Подготовка данных

Пусть:

- n — длина строки s
- m — длина шаблона p
- Создаем два массива: $pred$ (предыдущая строка) и tek (текущая строка) размером $m + 1$
- Инициализируем оба массива значениями **false**

2. Обработка базового случая — пустой строки

Для пустой строки соответствие возможно только если:

1. $pred[0] = \text{true}$ — пустая строка соответствует пустому шаблону
2. Для каждого символа шаблона $p[j - 1]$:
 - Если $p[j - 1] = '*'$, то $pred[j] = pred[j - 2]$
 - Это позволяет пропускать конструкции вида "*" для пустой строки

3. Основной цикл по символам строки

Для каждого символа строки $s[i - 1]$ (где i от 1 до n):

1. Устанавливаем $tek[0] = \text{false}$, так как непустая строка не может соответствовать пустому шаблону
2. Для каждого символа шаблона $p[j - 1]$ (где j от 1 до m):

Случай А: Если текущий символ шаблона — звездочка ($p[j - 1] = '*'$):

Рассматриваем два варианта:

- *Вариант 1 (ноль повторений)*: Пропускаем символ и звездочку.
 - Берем значение из $tek[j - 2]$ — соответствие без учета конструкции $"*"$
- *Вариант 2 (одно или более повторений)*: Используем символ перед звездочкой.
 - Проверяем, совпадает ли символ перед звездочкой ($p[j - 2]$) с текущим символом строки ($s[i - 1]$) или является точкой ($'.'$)
 - Если совпадает, то берем значение из $pred[j]$ — соответствие с предыдущим шагом
- Итоговое значение: $tek[j] = tek[j - 2]$ ИЛИ (совпадение И $pred[j]$)

Случай Б: Если текущий символ шаблона — точка или совпадает с символом строки:

- Просто переносим значение из предыдущего состояния: $tek[j] = pred[j - 1]$

Случай В: Если символы не совпадают:

- $tek[j] = \text{false}$

4. Обновление состояний

После обработки всех символов шаблона для текущего символа строки:

1. Массив $pred$ заменяем на массив tek (текущее состояние становится предыдущим)
2. Создаем новый массив tek , заполненный **false**, для следующей итерации

5. Возврат результата

После обработки всех символов строки:

- Результат находится в $pred[m]$ (последний элемент предыдущего массива)
- Возвращаем $pred[m]$ как итоговый ответ

Реализация на Python

```
1 class Solution:
2     def isMatch(self, s: str, p: str) -> bool:
3         pred = [False] * (len(p) + 1)
4         tek = [False] * (len(p) + 1)
5
6         pred[0] = True
7
8         for j in range(1, len(p) + 1):
9             if p[j - 1] == '*':
10                 pred[j] = pred[j - 2]
11
12         for i in range(1, len(s) + 1):
13             tek[0] = False
14
15             for j in range(1, len(p) + 1):
16                 if p[j - 1] == '*':
17                     if p[j - 2] == s[i - 1] or p[j - 2] == '.':
18                         tek[j] = tek[j - 2] or pred[j]
19                     else:
20                         tek[j] = tek[j - 2]
21                 elif p[j - 1] == '.' or p[j - 1] == s[i - 1]:
22                     tek[j] = pred[j - 1]
23                 else:
24                     tek[j] = False
25
26             pred, tek = tek, [False] * (len(p) + 1)
27
28         return pred[len(p)]
```

Листинг 1: Решение задачи 10

Пример пошагового выполнения

Рассмотрим пример: $s = "aab"$, $p = "c * a * b"$

Шаг 0: Инициализация

- $pred = [T, F, T, F, T, F]$ (T=true, F=false)
- Объяснение значений:
 - $pred[0] = T$: пустая строка — пустой шаблон
 - $pred[2] = T$: пропускаем "c"
 - $pred[4] = T$: пропускаем "c" и "a"

Шаг 1: Обработка первого символа 'a' ($i = 1$)

- Создаем $tek = [F, F, F, F, F, F]$
- Устанавливаем $tek[0] = F$ (непустая строка — пустой шаблон)
- При $j = 4$ ($p[3] = ' *', p[2] = ' a'$):
 - $p[2] = s[0]$ ('a' = 'a')
 - $tek[4] = tek[2]$ ИЛИ $pred[4] = F$ ИЛИ $T = T$
- Результат: $tek = [F, F, T, F, T, F]$
- Обновляем: $pred = tek, tek = [F, F, F, F, F, F]$

Шаг 2: Обработка второго символа 'a' ($i = 2$)

- При $j = 4$:
 - $p[2] = s[1]$ ('a' = 'a')
 - $tek[4] = tek[2]$ ИЛИ $pred[4] = F$ ИЛИ $T = T$
- Результат: $tek = [F, F, T, F, T, F]$
- Обновляем: $pred = tek, tek = [F, F, F, F, F, F]$

Шаг 3: Обработка третьего символа 'b' ($i = 3$)

- При $j = 5$ ($p[4] = ' b'$):
 - $p[4] = s[2]$ ('b' = 'b')
 - $tek[5] = pred[4] = T$
- Результат: $tek = [F, F, T, F, T, T]$
- Обновляем: $pred = tek$

Шаг 4: Возврат результата

- $pred[5] = T$
- Возвращаем **true** — строка соответствует шаблону

Временная сложность — асимптотика

- Подготовка массивов: $O(m)$
- Инициализация базовых случаев: $O(m)$
- Внешний цикл по символам строки: $O(n)$
- Внутренний цикл по символам шаблона: $O(m)$
- Общая временная сложность: $O(n \times m)$

Затраты памяти — асимптотика

- Два массива размера $m + 1$: $2 \times (m + 1) \approx O(m)$
- Вспомогательные переменные: $O(1)$
- Общие затраты памяти: $O(m)$

Ключевые особенности алгоритма

1. **Динамическое программирование:** Используем предыдущие вычисления для новых
2. **Оптимизация памяти:** Храним только две строки вместо всей таблицы
3. **Обработка звездочки:** Рассматриваем два варианта — ноль и одно или более повторений
4. **Выходящее программирование:** Каждое новое состояние вычисляется на основе предыдущих
5. **Корректность:** Охватывает все возможные варианты соответствия

Выполнено на латехе в оверлифе

Исправление ошибок, добавление недостающих частей отчета

Общая идея решения

Основная идея решения заключается в использовании динамического программирования для проверки всех возможных соответствий между строкой и шаблоном. Мы рассматриваем задачу как построение таблицы, где каждая ячейка $dp[i][j]$ отвечает на вопрос: "Можно ли сопоставить первые i символов строки с первыми j символами шаблона?".

Ключевая сложность в обработке символа '*', который может либо обнулять предыдущий символ, либо повторять его один или более раз. Для эффективного решения используем восходящее ДП, начиная с простых случаев и постепенно переходя к более сложным.

Описание используемых переменных и структур

1. Входные параметры:

- s — исходная строка
- p — шаблон

2. Переменные для размеров:

- $n = |s|$ — длина строки s
- $m = |p|$ — длина шаблона p

3. Массивы:

- $pred$ — массив размера $m + 1$, хранит значения для предыдущего символа строки
- tek — массив размера $m + 1$, хранит значения для текущего символа строки

4. Индексы:

- i — сколько символов строки взяли (от 1 до n)
- j — сколько символов шаблона взяли (от 1 до m)

5. Символы:

- $s[i - 1]$ — текущий символ строки
- $p[j - 1]$ — текущий символ шаблона
- $p[j - 2]$ — символ перед звездочкой

Как это работает

1. Массив *pred*:

- Сначала хранит базовые случаи для пустой строки
- Потом хранит результаты для предыдущего символа строки

2. Массив *tek*:

- Хранит результаты для текущего символа строки
- Потом становится новым *pred*

3. Индексы:

- Сдвиг на 1 (i и j от 1) нужен чтобы удобно работать с пустыми случаями
- $j - 2$ — чтобы взять символ перед звездочкой
- $i - 1$ и $j - 1$ — чтобы взять текущие символы

Почему так сделали

1. **Быстро:** Не делаем лишних вычислений
 2. **Экономно:** Два массива вместо огромной таблицы
 3. **Понятно:** Легко следить за логикой
 4. **Работает везде:** Подходит для любых строк и шаблонов
-