Corretude

Definição: Dadas as strings **text** e **pattern** com tamanhos T e P, tal que t_i e p_j são elementos dada as posições, onde

- $0 \leq i < T$
- $0 \leq j < P$
- **text** é formada por caracteres {a..z}
- **pattern** é formada por caracteres (a..z, *, .).
  - . define um caractere qualquer em (a..z)
  - * define 0 ou n repetições do caractere antecessor, este pertencendo a .

Temos que isMatch(text,pattern) que retorna Match(i,j), para $i = 0$ e $j = 0$, e este retorna se **text** é formada por **pattern** através do algoritmo.

```
def isMatch(text: str, pattern: str)-> bool:
  def Match(i: int,j: int)->bool:
    if j == len(pattern):
      return i == len(text)
    else:
        first_match = i < len(text) and pattern[j] in {text[i], '.'}
        if (j+1 < len(pattern) and pattern[j+1] == '*'):
            return Match(i, j+2) or first_match and Match(i+1, j)
        else:
            return first_match and Match(i+1, j+1)
     ###
  return Match(0,0)
```

- **Base:** Se no processo atual **pattern** encerrou, $j = P$, e **text** encerrou, $i = T$, então **text** é formada por **pattern**. ou seja,

$$\text{Match(T,P)} = \text{True}$$

$$\text{Match(i,P)} = \text{False}$$

- **Hipotese**:
  - Se **pattern** não encerrou
    1. Verifico o first_match, se **text** não encerrou **E** se $p_j = t_i \ \vee p_j =$ '.'
    2. Se existir um proximo caractere em **pattern E** ele for *:
       1. Decido entre usar ou não p_j (p_j*? Repete p_j : não),
          - Pois se der match sem p_j então ignoro ele, pulando para p_{j+2}
          - Se não der match, então através do first_match verifico se ou encerrou ou continuo repetindo.
       2. Ou Não havia * e posso seguir verificando se o first_match é verdadeiro e Match(t_{i+1},p_{j+1}) também é.
- **Passo:**
  - i, j sempre são incrementados, logo sempre vão se aproximar do caso base ($j = P$? [$i = T$? True : False] : chamada da recursão)

- **Caso Base:**
  - Se **pattern** encerrou ***And*** **Text** encerrou, e contando que os passos anteriores operaram de forma efetiva, Então **Text** é formada por **pattern** (**TRUE**).
  - Se **pattern** encerrou ***And*** **Text** não encerrou, e contando que os passos anteriores operaram de forma efetiva, Então **Text** não formada por **pattern** (FALSE).
- **Chamada da Recursão:** (Caso contrário)
  - Obtém o `first_match`:
    - Verifica se **Text** não encerrou ***And*** ($p_j = t_i$ ***Or*** $p_j =$ '.')
    - Ele vai servir para cancelar o processo quando **text** encerrar e o **pattern** não.
    - Ou para cancelar o processo quando não houver match entre t_i e p_j.
  - **Caso de haver Repetição**: (Se p_{j+1} existir ***&*** $p_{j+1} = *$)
    - Verifica `Match(i, j+2)` ***Or*** (`first_match` ***And*** `Match(i+1, j)`)
    - (Lembrando que o ***Or*** exige que exista pelo menos um dos valores seja verdadeira, ou seja, na primeira aparição de um valor verdadeiro poda as outras verificações)
    - `Match(i, j+2)`: Verifica se sem a repetição atual o o match ainda ocorre, logo ele não é necessário para formar **text**. Ele trás a necessidade de uma veficação (`first_match` ***And*** `Match(i+1, j)`) quando falso.
    - `first_match`: Ele encerra a necessidade de uma verificação de `Match(i+1, j)`, repetições de p_j, em caso de valor falso.
    - `Match(i+1, j)`: Verifica o match do loop de p_j em t_i.
  - **Caso não houver repetição:**
    - Verifica `first_match` ***And*** `Match(i+1, j+1)`
    - (lembrando que o ***And*** exige que todos as suas verificações sejam verdadeiras podando na primeira aparição de uma valor Falso)
    - `first_match`: exclui a necessidade da verificação `Match(i+1, j+1)`, se o mesmo for falso, ou seja, se tratarmos de um **text** que encerrou ou se não houve match p_j e t_i.
    - `Match(i+1, j+1)`: Verifica o match p_j e t_i termo a termo para os casos onde não há 'loop'(*).

80 (0,0){ 84 (0,2){} 91 (1,2){}

}