Procura de padrões em sequências

Objetivo

Dada uma sequência num dado alfabeto, procurar **todas as ocorrências** de um padrão nessa sequência

Vamos assumir que o padrão é fixo.

Algoritmo “naive”:
procura em todas as posições

```
seq = input("Sequence:")
pattern = input("Pattern:")


res = [ ]
for i in range(len(seq)-len(pattern)+1):
        j = 0
        while j < len(pattern) and pattern[j]==seq[i+j]:
                j = j + 1
        if j == len(pattern):
                res.append(i)

print ("Pattern occurs in positions:" , res)
```

Complexidade do algoritmo naive

No seu pior caso o algoritmo naive anterior tem complexidade: ***O(m(n-m))***

- ***m*** – comprimento do padrão
- ***n*** – comprimento da sequência

Percorrem-se todas as posições possíveis (*n-m*) e, em cada caso fazem-se no pior caso *m* comparações.

Como melhorar a eficiência destes algoritmos ?

Pré-processamento do padrão

Uma das alternativas para melhorar o desempenho destes algoritmos é realizar o **pré-processamento do padrão** de uma forma inteligente.

Objectivo:

- Realizar a procura de vários padrões com uma **única passagem** pela sequência, e/ou
- Optimizar a procura do(s) **mesmo(s) padrão(ões) em várias sequências** distintas

Iremos abordar 3 alternativas:

- Algoritmos heurísticos
- **Autómatos finitos**
- Árvore com os padrões **(tries) – próxima aula !!**

Algoritmo de Boyer-Moore

O algoritmo de Boyer-Moore é um algoritmo **heurístico** que permite tornar mais eficiente a procura de padrões em sequências

- o pior caso tem complexidade igual ao algoritmo naive
- na maior parte dos casos permite ganhos significativos

Algoritmo baseia-se num pré-processamento do padrão segundo duas regras; estas permitem, quando há uma falha no match entre o padrão e a sequência avançar o máximo de posições possível

Será implementado na classe *BoyerMoore* (material suplementar à aula)

Algoritmo de Boyer Moore - descrição

Percorre-se a sequência tal como no algoritmo naive

Texto é comparado com padrão da direita para a esquerda (do final para o início);

Quando há uma falha no match entre o padrão e a sequência, existem duas regras que permitem avançar a pesquisa:

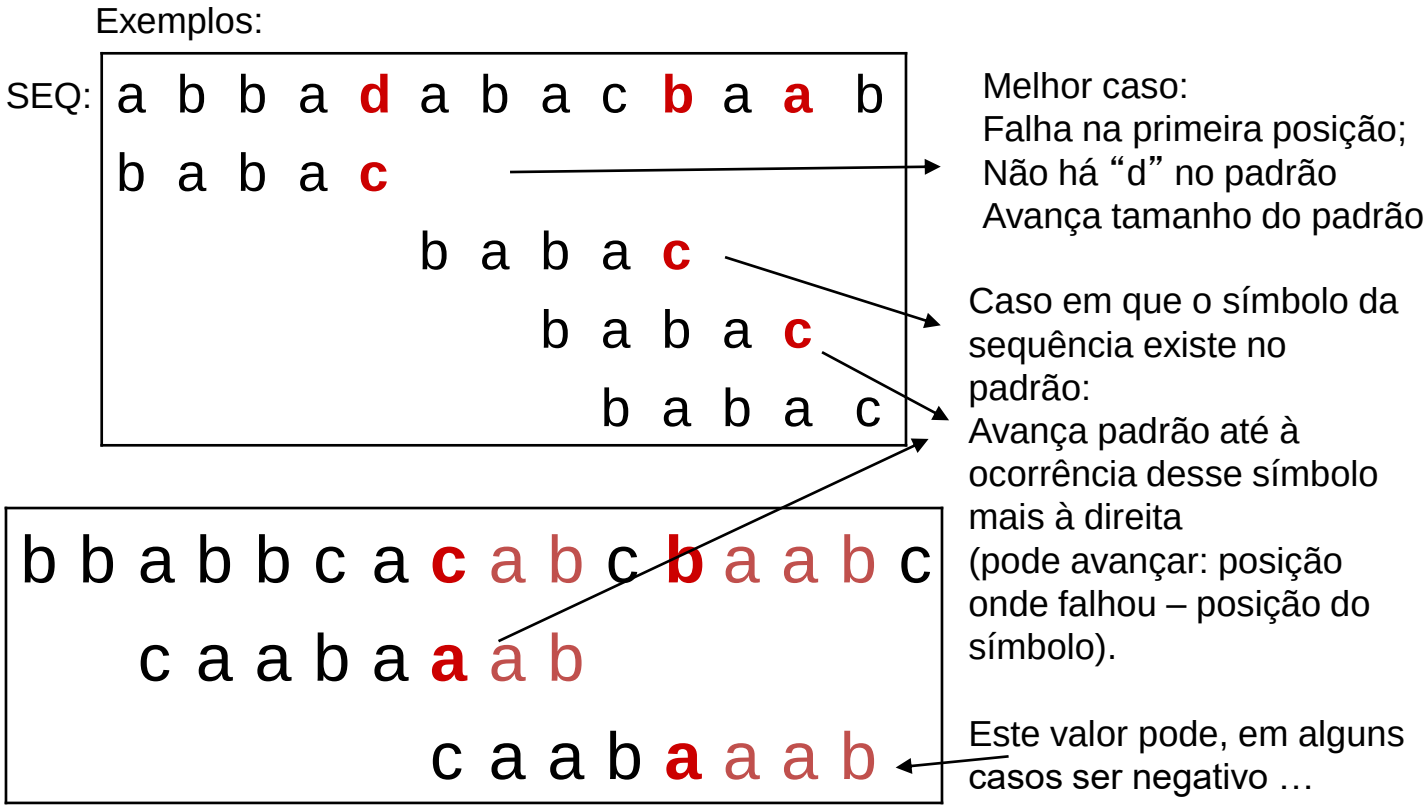
- **Bad-character rule**

  Avançar para a próxima ocorrência no padrão do símbolo que falhou (ou se não existir avançar o máximo possível)

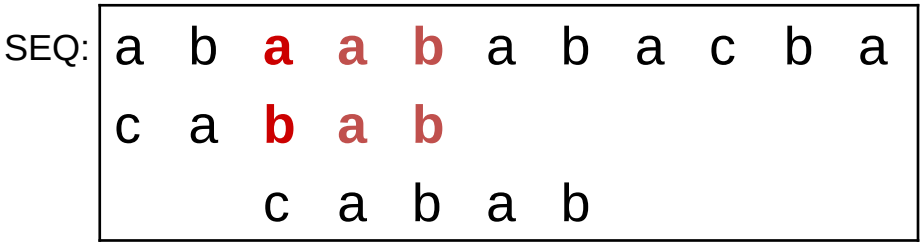
- **Good suffix rule**

  Avançar para a próxima ocorrência no padrão da parte que fez match antes de falhar

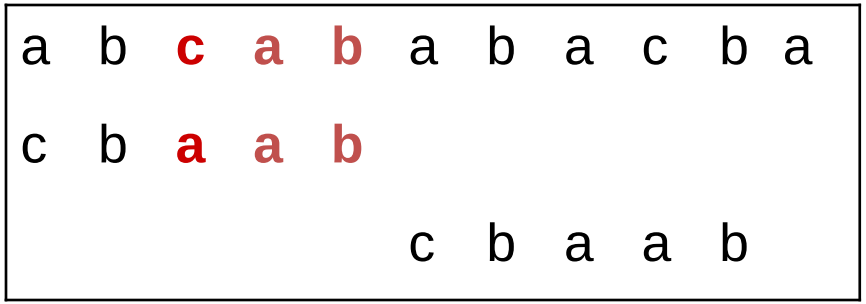
Boyer Moore: bad character rule



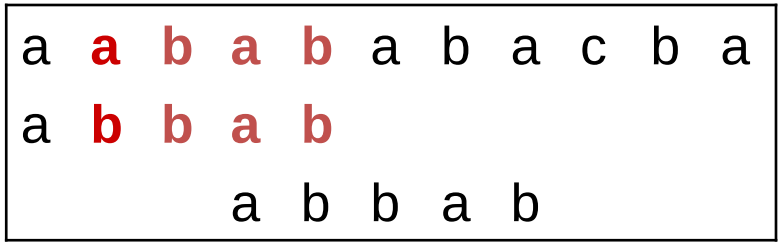
Boyer Moore: good suffix rule



Avança para a próxima ocorrência do sufixo



Sufixo não ocorre de novo: Pode avançar tamanho do padrão



Sufixo não ocorre mas um prefixo do padrão faz match com um sufixo do match anterior (no exemplo “ab”)

Autómatos finitos

Autómatos finitos (AF) podem ser descritos como “máquinas” que processam uma sequência de símbolos da esquerda para a direita

Os AFs alteram o seu estado interno à medida que processam os carateres; novo estado depende apenas do anterior e do último caractere lido

Escolhendo adequadamente as transições pode determinar-se se o padrão está contido até à posição lida

Um AF criado a partir de um padrão *p* pode determinar todas as posições de *p* numa sequência com uma única passagem

Autómatos finitos

Definição: um **autómato** é uma estrutura

$$M = (Q, A, q_0, \delta, F)$$

Q – conjunto de estados

A – alfabeto (conjunto de possíveis símbolos)

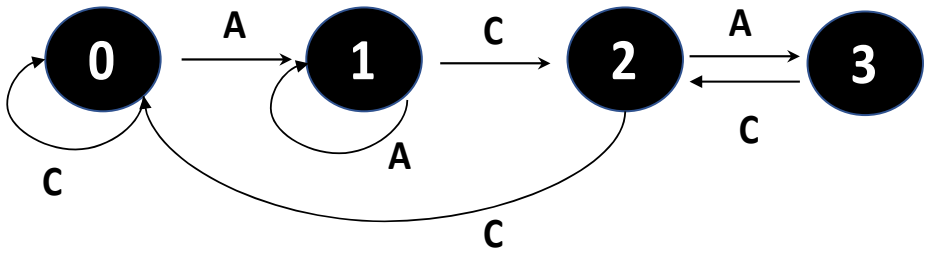
q_0 Є Q – estado inicial

δ: Q, A -> Q - função de transição

F (sub-conjunto de Q) – conjunto de estados finais

Exemplo de um AF para um padrão

State	Symbol	Next State
0	A	1
0	C	0
1	A	1
1	C	2
2	A	3
2	C	0
3	A	1
3	C	2



Alphabet: A, C
Pattern: A C A

Sequence		C	A	C	A	A	C	A	A
State	0	0	1	2	3	1	2	3	1
Occurrence					1			4	

Construindo um AF para um padrão

Dado um padrão *p* de tamanho *m* definido num alfabeto *Ap*, um autómato capaz de detectar p em qualquer sequência tem a seguinte estrutura:

Q = { 0, ..., m }

A = Ap

$q_0 = 0$

F = { m }

Função de transição:

$$\delta (q, a) = \text{max\_overlap}(p_0 \ldots p_{q-1}a , p)$$

onde max_overlap(s1,s2) é definido como uma função que dá o comprimento do overlap máximo entre s1 e s2

Procura de padrões com um AF

Tendo um autómato construído, este pode ser usado para eficientemente procurar ocorrências do padrão usado para o construir

A sequência é percorrida de forma linear (uma única vez), sendo o estado atual do padrão atualizado de acordo com a tabela de transições dado o símbolo na sequência (e o estado anterior)

Sempre que o estado atual é igual a ***m***, o padrão foi encontrado

Complexidade dos AFs

Quanto à complexidade:

- A tarefa de construir o AF pode ser realizada por algoritmos ***O(|A|.m)***
- A tarefa de percorrer a sequência é O(n)
- No total teremos: ***O(n + |A|.m)***

O uso de AFs é especialmente eficiente quando queremos procurar o mesmo padrão num conjunto de sequências – uma tarefa comum em Bioinformática

Exercício

- Implementar AFs definindo uma classe Automata
- Classe deverá ter variáveis para guardar o alfabeto, o número de estados, a tabela de transições
- Métodos:
  - Construtor (a partir do alfabeto e de um padrão)
  - Método que constrói a tabela de transições a partir do padrão (usado no construtor)
  - Método que aplica o AF a uma sequência (dada como entrada), dando como resultado a lista de estados
  - Método que identifica as posições de match do padrão representado pelo AF numa sequência (dada como entrada)