Análise de Algoritmos

Parte destes slides são adaptações de slides

do Prof. Paulo Feofiloff e do Prof. José Coelho de Pina.

Busca de padrão

Dados

- uma palavra $P[1\,..\,m]$ e
- um texto $T[1\,..\,n]$,

uma ocorrência de P em T é um índice s tal que $T[s+j] = P[j]$ para $j = 1, \ldots, m$.

Busca de padrão

Dados

- uma palavra $P[1\,..\,m]$ e
- um texto $T[1\,..\,n]$,

uma ocorrência de P em T é um índice s tal que $T[s+j] = P[j]$ para $j = 1, \ldots, m$.

Exemplo:

	1	2	3
P	B	R	A

	1	2	3	4	5	6	7	8	9	10	11
T	A	B	R	A	C	A	D	A	B	R	A

Busca de padrão

Dados

- uma palavra $P[1\,..\,m]$ e
- um texto $T[1\,..\,n]$,

uma ocorrência de P em T é um índice s tal que $T[s+j] = P[j]$ para $j = 1, \ldots, m$.

Exemplo:

	1	2	3
P	B	R	A

	1	2	3	4	5	6	7	8	9	10	11
T	A	B	R	A	C	A	D	A	B	R	A

Problema: Dada uma palavra $P[1\,..\,m]$ e um texto $T[1\,..\,n]$, calcular o número de ocorrências de P em T.

Busca de padrão

Dados

- uma palavra $P[1\,..\,m]$ e
- um texto $T[1\,..\,n]$,

uma ocorrência de P em T é um índice s tal que $T[s+j] = P[j]$ para $j = 1, \ldots, m$.

Exemplo:

	1	2	3
P	B	R	A

	1	2	3	4	5	6	7	8	9	10	11
T	A	B	R	A	C	A	D	A	B	R	A

Problema: Dada uma palavra $P[1\,..\,m]$ e um texto $T[1\,..\,n]$, calcular o número de ocorrências de P em T.

No exemplo, P ocorre duas vezes em T: em 1 e em 8.

Algoritmo ingênuo

```
BUSCA-TRIVIAL (T, n, P, m)
 1   c ← 0
 2   para s ← 0 até n − m faça
 3         j ← 1
 4         enquanto j ≤ m e P[j] = T[s + j] faça
 5               j ← j + 1
 6         se j > m
 7               então c ← c + 1
 8   return c
```

Algoritmo ingênuo

```
BUSCA-TRIVIAL (T, n, P, m)
 1   c ← 0
 2   para s ← 0 até n − m faça
 3         j ← 1
 4         enquanto j ≤ m e P[j] = T[s + j] faça
 5               j ← j + 1
 6         se j > m
 7               então c ← c + 1
 8   return c
```

Consumo de tempo: $\mathrm{O}(mn)$

Algoritmo ingênuo

```
BUSCA-TRIVIAL (T, n, P, m)
 1   c ← 0
 2   para s ← 0 até n − m faça
 3         j ← 1
 4         enquanto j ≤ m e P[j] = T[s + j] faça
 5               j ← j + 1
 6         se j > m
 7               então c ← c + 1
 8   return c
```

Consumo de tempo: $\Theta(mn)$

Exemplo: $T = a^n$ e $P = a^m$, ou $P = a^{m-1}b$.

Algoritmo ingênuo

```
BUSCA-TRIVIAL (T, n, P, m)
 1   c ← 0
 2   para s ← 0 até n − m faça
 3         j ← 1
 4         enquanto j ≤ m e P[j] = T[s + j] faça
 5               j ← j + 1
 6         se j > m
 7               então c ← c + 1
 8   return c
```

Consumo de tempo: $\Theta(mn)$

Exemplo: $T = a^n$ e $P = a^m$, ou $P = a^{m-1}b$.

Nesta aula, vamos ver um algoritmo cujo consumo de tempo no pior caso é $\Theta(mn)$, porém na prática é $\mathrm{O}(n)$, podendo inclusive ter um comportamento sublinear.

Algoritmo BM – primeira versão

BM: Boyer-Moore.

Algoritmo BM – primeira versão

BM: Boyer-Moore.

$v_1[c]$: menor t em $0\,..\,m-1$ tal que $P[m-t]=c$
(se c não aparece em P, então $v_1[c]=m$).

Algoritmo BM – primeira versão

BM: Boyer-Moore.

$v_1[c]$: menor t em $0\,..\,m-1$ tal que $P[m-t] = c$
(se c não aparece em P, então $v_1[c] = m$).

$v_1[c]$ corresponde à última ocorrência de c em P.

Algoritmo BM – primeira versão

BM: Boyer-Moore.

$v_1[c]$: menor t em $0\,..\,m-1$ tal que $P[m-t]=c$
(se c não aparece em P, então $v_1[c]=m$).

$v_1[c]$ corresponde à última ocorrência de c em P.

Exemplo:

P

1	2	3	4	5	6	7	8	9	10	11
a	b	b	c	a	b	d	a	c	a	d

v_1

a	b	c	d
1	5	2	0

Algoritmo BM – primeira versão

BM: Boyer-Moore.

$v_1[c]$: menor t em $0\,..\,m-1$ tal que $P[m-t] = c$
(se c não aparece em P, então $v_1[c] = m$).

$v_1[c]$ corresponde à última ocorrência de c em P.

Exemplo:

	1	2	3	4	5	6	7	8	9	10	11
P	a	b	b	c	a	b	d	a	c	a	d

	a	b	c	d
v_1	1	5	2	0

Seja $k \geq m$. Suponha que $c = T[k+1]$ e que $v_1[c] = 4$. Então $P[1\,..\,m]$ não é um sufixo de $T[1\,..\,k+1], \ldots, T[1\,..\,k+4]$.
(Supondo que $k + 4 \leq n$.)

Algoritmo BM – primeira versão

BM: Boyer-Moore.

$v_1[c]$: menor t em $0\,..\,m-1$ tal que $P[m-t]=c$
(se c não aparece em P, então $v_1[c]=m$).

$v_1[c]$ corresponde à última ocorrência de c em P.

Exemplo:

	1	2	3	4	5	6	7	8	9	10	11
P	a	b	b	c	a	b	d	a	c	a	d

	a	b	c	d
v_1	1	5	2	0

Seja $k \geq m$. Suponha que $c = T[k+1]$. Então $P[1\,..\,m]$ não é um sufixo de $T[1\,..\,k+1], \ldots, T[1\,..\,k+\mu]$, onde $\mu = \min\{n-k, v_1[c]\}$.

Algoritmo Boyer-Moore 1

Supondo que o alfabeto é o conjunto $\Sigma = \{0, \ldots, 255\}$.

```
BM1 (P, m, T, n)
 1   para i ← 0 até 255 faça v1[i] ← m
 2   para i ← 1 até m faça v1[P[i]] ← m − i
 3   c ← 0        k ← m
 4   enquanto k ≤ n faça
 5         r ← 0
 6         enquanto m − r ≥ 1 e P[m − r] = T[k − r] faça
 7               r ← r + 1
 8         se m − r < 1 então c ← c + 1
 9         se k = n
10               então k ← k + 1
11               senão k ← k + v1[T[k + 1]] + 1
12   devolva c
```

Algoritmo Boyer-Moore 1

Supondo que o alfabeto é o conjunto $\Sigma = \{0, \ldots, 255\}$.

```
BM1 (P, m, T, n)
 1    para i ← 0 até 255 faça v1[i] ← m
 2    para i ← 1 até m faça v1[P[i]] ← m − i
 3    c ← 0        k ← m
 4    enquanto k ≤ n faça
 5         r ← 0
 6         enquanto m − r ≥ 1 e P[m − r] = T[k − r] faça
 7              r ← r + 1
 8         se m − r < 1 então c ← c + 1
 9         se k = n
10              então k ← k + 1
11              senão k ← k + v1[T[k + 1]] + 1
12    devolva c
```

Consumo de tempo: $\Theta(|\Sigma| + mn)$

Algoritmo BM – segunda versão

j em $1\,..\,m{-}1$ é bom para i se $P[m{-}\mu\,..\,m] = P[j{-}\mu\,..\,j]$,
onde $\mu = \min\{m{-}i, j{-}1\}$.

Algoritmo BM – segunda versão

j em $1\,..\,m{-}1$ é bom para i se $P[m{-}\mu\,..\,m] = P[j{-}\mu\,..\,j]$,
onde $\mu = \min\{m{-}i, j{-}1\}$.

j é bom para i se $P[i\,..\,m]$ é sufixo de $P[1\,..\,j]$
ou $P[1\,..\,j]$ é sufixo de $P[i\,..\,m]$

Algoritmo BM – segunda versão

j em $1\,..\,m{-}1$ é bom para i se $P[m{-}\mu\,..\,m] = P[j{-}\mu\,..\,j]$,
onde $\mu = \min\{m{-}i, j{-}1\}$.

j é bom para i se $P[i\,..\,m]$ é sufixo de $P[1\,..\,j]$
ou $P[1\,..\,j]$ é sufixo de $P[i\,..\,m]$

Exemplos:

	1	2	3	4	5	6
P	c	a	a	b	a	a

Algoritmo BM – segunda versão

j em $1\,..\,m{-}1$ é bom para i se $P[m{-}\mu\,..\,m] = P[j{-}\mu\,..\,j]$,
onde $\mu = \min\{m{-}i, j{-}1\}$.

j é bom para i se $P[i\,..\,m]$ é sufixo de $P[1\,..\,j]$
ou $P[1\,..\,j]$ é sufixo de $P[i\,..\,m]$

Exemplos:

	1	2	3	4	5	6
P	c	a	a	b	a	a

$j = 3$ é bom para $i = 5$, pois aa é sufixo de caa.

Algoritmo BM – segunda versão

j em $1\,..\,m{-}1$ é bom para i se $P[m{-}\mu\,..\,m] = P[j{-}\mu\,..\,j]$,
onde $\mu = \min\{m{-}i, j{-}1\}$.

j é bom para i se $P[i\,..\,m]$ é sufixo de $P[1\,..\,j]$
ou $P[1\,..\,j]$ é sufixo de $P[i\,..\,m]$

Exemplos:

	1	2	3	4	5	6	7	8	9
P	c	a	a	b	a	a	b	a	a

Algoritmo BM – segunda versão

j em $1\,..\,m{-}1$ é bom para i se $P[m{-}\mu\,..\,m] = P[j{-}\mu\,..\,j]$,
onde $\mu = \min\{m{-}i, j{-}1\}$.

j é bom para i se $P[i\,..\,m]$ é sufixo de $P[1\,..\,j]$
ou $P[1\,..\,j]$ é sufixo de $P[i\,..\,m]$

Exemplos:

	1	2	3	4	5	6	7	8	9
P	c	a	a	b	a	a	b	a	a

$j = 6$ é bom para $i = 8$, pois aa é sufixo de $caabaa$.

Algoritmo BM – segunda versão

j em $1\,..\,m{-}1$ é bom para i se $P[m{-}\mu\,..\,m] = P[j{-}\mu\,..\,j]$,
onde $\mu = \min\{m{-}i, j{-}1\}$.

j é bom para i se $P[i\,..\,m]$ é sufixo de $P[1\,..\,j]$
ou $P[1\,..\,j]$ é sufixo de $P[i\,..\,m]$

Exemplos:

	1	2	3	4	5	6	7	8	9
P	c	a	a	b	a	a	b	a	a

$j = 6$ é bom para $i = 8$, pois aa é sufixo de $caabaa$.

$j = 6$ é bom para $i = 5$, pois $aabaa$ é sufixo de $caabaa$.

Algoritmo BM – segunda versão

j em $1\,..\,m{-}1$ é bom para i se $P[m{-}\mu\,..\,m] = P[j{-}\mu\,..\,j]$,
onde $\mu = \min\{m{-}i, j{-}1\}$.

j é bom para i se $P[i\,..\,m]$ é sufixo de $P[1\,..\,j]$
ou $P[1\,..\,j]$ é sufixo de $P[i\,..\,m]$

Exemplos:

	1	2	3	4	5	6	7	8	9
P	c	a	a	b	a	a	b	a	a

$j = 6$ é bom para $i = 8$, pois aa é sufixo de $caabaa$.

$j = 6$ é bom para $i = 5$, pois $aabaa$ é sufixo de $caabaa$.

$j = 3$ é bom para $i = 8$, pois aa é sufixo de caa.

Algoritmo BM – segunda versão

j em $1\,..\,m-1$ é bom para i se $P[m-\mu\,..\,m] = P[j-\mu\,..\,j]$,
onde $\mu = \min\{m-i, j-1\}$.

j é bom para i se $P[i\,..\,m]$ é sufixo de $P[1\,..\,j]$
ou $P[1\,..\,j]$ é sufixo de $P[i\,..\,m]$

Exemplos:

	1	2	3	4	5	6	7	8	9
P	c	a	a	b	a	a	c	a	a

$j = 3$ é bom para $i = 6$, pois caa é sufixo de $acaa$.

Algoritmo BM – segunda versão

j em $1\,..\,m{-}1$ é bom para i se $P[m{-}\mu\,..\,m] = P[j{-}\mu\,..\,j]$,
onde $\mu = \min\{m{-}i, j{-}1\}$.

j é bom para i se $P[i\,..\,m]$ é sufixo de $P[1\,..\,j]$
ou $P[1\,..\,j]$ é sufixo de $P[i\,..\,m]$

Exemplos:

	1	2	3	4	5	6	7	8	9
P	c	a	a	b	a	a	c	a	a

$j = 3$ é bom para $i = 6$, pois caa é sufixo de $acaa$.

$j = 3$ é bom para $i = 1, \ldots, 7$, pois caa é sufixo de $caabaacaa$.

Algoritmo BM – segunda versão

j é bom para i se $P[i\,..\,m]$ é sufixo de $P[1\,..\,j]$
ou $P[1\,..\,j]$ é sufixo de $P[i\,..\,m]$

Algoritmo BM – segunda versão

j é bom para i se $P[i\,..\,m]$ é sufixo de $P[1\,..\,j]$
ou $P[1\,..\,j]$ é sufixo de $P[i\,..\,m]$

$v_2[i]$: menor t em $0\,..\,m-1$ tal que $m-t$ é bom para i.

Algoritmo BM – segunda versão

j é bom para i se $P[i\,..\,m]$ é sufixo de $P[1\,..\,j]$
ou $P[1\,..\,j]$ é sufixo de $P[i\,..\,m]$

$v_2[i]$: menor t em $0\,..\,m-1$ tal que $m-t$ é bom para i.

Exemplos:

1	2	3	4	5	6
c	a	a	b	a	a

i	6	5	4	3	2	1
v_2	1	3	6	6	6	6

Algoritmo BM – segunda versão

j é bom para i se $P[i\,..\,m]$ é sufixo de $P[1\,..\,j]$
ou $P[1\,..\,j]$ é sufixo de $P[i\,..\,m]$

$v_2[i]$: menor t em $0\,..\,m-1$ tal que $m-t$ é bom para i.

Exemplos:

1	2	3	4	5	6
c	a	a	b	a	a

i	6	5	4	3	2	1
v_2	1	3	6	6	6	6

1	2	3	4	5	6	7	8
b	a	-	b	a	.	b	a

i	8	7	6	5	4	3	2	1
v_2	3	3	6	6	6	6	6	6

Algoritmo BM – segunda versão

j é bom para i se $P[i\,..\,m]$ é sufixo de $P[1\,..\,j]$
ou $P[1\,..\,j]$ é sufixo de $P[i\,..\,m]$

$v_2[i]$: menor t em $0\,..\,m-1$ tal que $m-t$ é bom para i.

Exemplos:

1	2	3	4	5	6
c	a	a	b	a	a

i	6	5	4	3	2	1
v_2	1	3	6	6	6	6

1	2	3	4	5	6	7	8
b	a	-	b	a	.	b	a

i	8	7	6	5	4	3	2	1
v_2	3	3	6	6	6	6	6	6

1	2	3	4	5	6	7	8	9	10	11
b	a	-	b	a	*	b	a	*	b	a

i	11	10	9	8	7	6	5	4	3	2	1
v_2	3	3	3	3	3	9	9	9	9	9	9

Algoritmo Boyer-Moore 2

```
BM2 (P, m, T, n)
 1   para i ← m decrescendo até 1 faça
 2       j ← m − 1        r ← 0
 3       enquanto m − r ≥ i e j − r ≥ 1 faça
 4           se P[m − r] = P[j − r] então r ← r + 1
 5               senão j ← j − 1        r ← 0
 6       v2[i] ← m − j
```

Algoritmo Boyer-Moore 2

```
BM2 (P, m, T, n)
 1   para i ← m decrescendo até 1 faça
           ...
 6         v2[i] ← m − j
 7   c ← 0        k ← m
 8   enquanto k ≤ n faça
 9         r ← 0
10         enquanto m − r ≥ 1 e P[m − r] = T[k − r] faça
11               r ← r + 1
12         se m − r < 1 então c ← c + 1
13         se r = 0 então k ← k + 1
14               senão k ← k + v2[m − r + 1]
15   devolva c
```

Algoritmo Boyer-Moore 2

```
BM2 (P, m, T, n)
 1   para i ← m decrescendo até 1 faça
           ...
 6         v2[i] ← m − j
 7   c ← 0        k ← m
 8   enquanto k ≤ n faça
 9         r ← 0
10         enquanto m − r ≥ 1 e P[m − r] = T[k − r] faça
11               r ← r + 1
12         se m − r < 1 então c ← c + 1
13         se r = 0 então k ← k + 1
14               senão k ← k + v2[m − r + 1]
15   devolva c
```

Consumo de tempo: $\Theta(mn)$

Algoritmo Boyer-Moore

Calcula v_1 e v_2, e
atualiza fazendo sempre o maior dos dois deslocamentos.

Algoritmo Boyer-Moore

Calcula v_1 e v_2, e
atualiza fazendo sempre o maior dos dois deslocamentos.

Consumo de tempo: $\Theta(mn)$

Algoritmo Boyer-Moore

Calcula v_1 e v_2, e
atualiza fazendo sempre o maior dos dois deslocamentos.

Consumo de tempo: $\Theta(mn)$

No caso médio entretanto, ele consome apenas $\mathrm{O}(n)$.

É o mais rápido algoritmo de busca de padrão na prática.

Algoritmo Boyer-Moore

Calcula v_1 e v_2, e
atualiza fazendo sempre o maior dos dois deslocamentos.

Consumo de tempo: $\Theta(mn)$

No caso médio entretanto, ele consome apenas $\mathrm{O}(n)$.

É o mais rápido algoritmo de busca de padrão na prática.

É possível aperfeiçoar a tabela v_2 para que
este algoritmo consuma apenas $\mathrm{O}(m+n)$ no pior caso.

Algoritmo Boyer-Moore

Calcula v_1 e v_2, e
atualiza fazendo sempre o maior dos dois deslocamentos.

Consumo de tempo: $\Theta(mn)$

No caso médio entretanto, ele consome apenas $\mathrm{O}(n)$.

É o mais rápido algoritmo de busca de padrão na prática.

É possível aperfeiçoar a tabela v_2 para que
este algoritmo consuma apenas $\mathrm{O}(m+n)$ no pior caso.

Veja os exercícios nas notas de aula indicadas para leitura.