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 $j = 1, \ldots, m$.

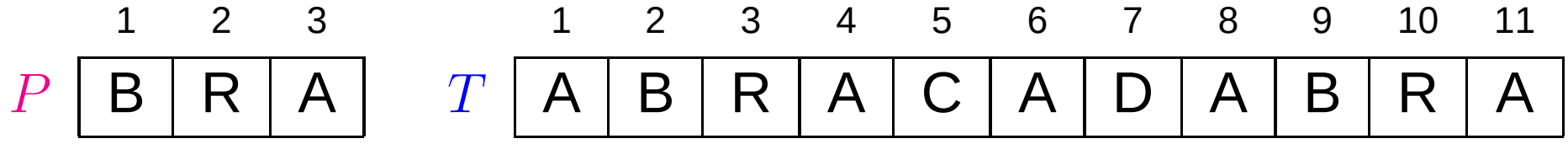
Busca de padrão

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:



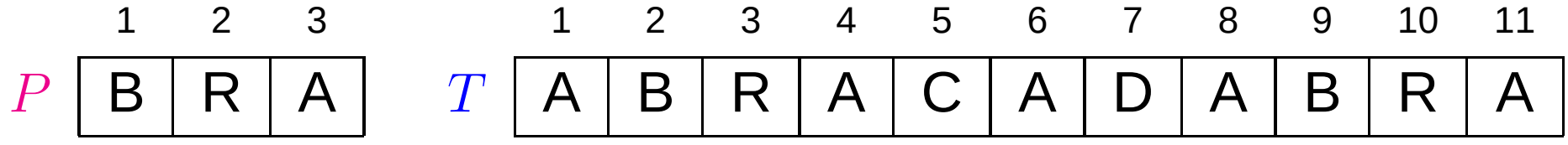
Busca de padrão

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:



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

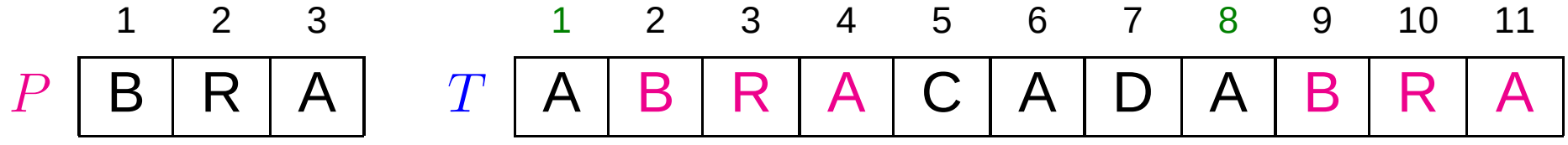
Busca de padrão

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:



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: $\Theta(mn)$ (justo)

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: $\Theta(mn)$ (justo)

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

Nesta aula: algoritmo KMP

Algoritmo KMP

KMP: Knuth, Morris e Pratt.

Algoritmo KMP

KMP: Knuth, Morris e Pratt.

P_q: prefixo de P de comprimento q

Algoritmo KMP

KMP: Knuth, Morris e Pratt.

P_q: prefixo de P de comprimento q

$P_k \sqsupseteq P_q$: P_k é sufixo de P_q

Algoritmo KMP

KMP: Knuth, Morris e Pratt.

P_q: prefixo de P de comprimento q

$P_k \sqsupseteq P_q$: P_k é sufixo de P_q

Função prefixo: $\Pi[q] = \max\{k : k < q \text{ e } P_k \sqsupseteq P_q\}$

Algoritmo KMP

KMP: Knuth, Morris e Pratt.

P_q: prefixo de P de comprimento q

$P_k \sqsupseteq P_q$: P_k é sufixo de P_q

Função prefixo: $\Pi[q] = \max\{k : k < q \text{ e } P_k \sqsupseteq P_q\}$

Em palavras, $\Pi[q]$ é o comprimento do
maior prefixo próprio de P_q que é sufixo de P_q.

Algoritmo KMP

KMP: Knuth, Morris e Pratt.

P_q: prefixo de P de comprimento q

$P_k \sqsupseteq P_q$: P_k é sufixo de P_q

Função prefixo: $\Pi[q] = \max\{k : k < q \text{ e } P_k \sqsupseteq P_q\}$

Em palavras, $\Pi[q]$ é o comprimento do
maior prefixo próprio de P_q que é sufixo de P_q.

Exemplo:

	1	2	3	4	5	6	7	8	9	10	11
P	a	b	a	b	a	a	b	a	b	c	a
Π	0	0	1	2	3	1	2	3	4	0	1

Algoritmo KMP

```
CALCULA-PREFIXO (P, m)
 1  Π[1] ← 0
 2  k ← 0
 3  para q ← 2 até m faça
 4      enquanto k > 0 e P[k + 1] ≠ P[q] faça
 5          k ← Π[k]
 6      se P[k + 1] = P[q]
 7          então k ← k + 1
 8      Π[q] ← k
 9  devolva Π
```

Algoritmo KMP

```
CALCULA-PREFIXO (P, m)
 1  Π[1] ← 0
 2  k ← 0
 3  para q ← 2 até m faça
 4     enquanto k > 0 e P[k + 1] ≠ P[q] faça
 5        k ← Π[k]
 6     se P[k + 1] = P[q]
 7        então k ← k + 1
 8     Π[q] ← k
 9  devolva Π
```

Consumo de tempo: $O(m^2)$

Algoritmo KMP

```
CALCULA-PREFIXO (P, m)
 1  Π[1] ← 0
 2  k ← 0
 3  para q ← 2 até m faça
 4     enquanto k > 0 e P[k + 1] ≠ P[q] faça
 5        k ← Π[k]
 6     se P[k + 1] = P[q]
 7        então k ← k + 1
 8     Π[q] ← k
 9  devolva Π
```

Consumo de tempo: $\Theta(m^2)$??

Invariantes do Algoritmo KMP

```
CALCULA-PREFIXO (P, m)
 1  Π[1] ← 0    k ← 0
 2  para q ← 2 até m faça
 3     enquanto k > 0 e P[k + 1] ≠ P[q] faça
 4        k ← Π[k]
 5     se P[k + 1] = P[q]
 6        então k ← k + 1
 7     Π[q] ← k
 8  devolva Π
```

Invariantes do Algoritmo KMP

```
CALCULA-PREFIXO (P, m)
 1  Π[1] ← 0    k ← 0
 2  para q ← 2 até m faça
 3     enquanto k > 0 e P[k + 1] ≠ P[q] faça
 4        k ← Π[k]
 5     se P[k + 1] = P[q]
 6        então k ← k + 1
 7     Π[q] ← k
 8  devolva Π
```

(1) $k < q - 1$

(2) $P_k \sqsupseteq P_{q-1}$

(3) k é o maior possível tal que valem (1) e (2)

(4) $k = \Pi[q - 1]$

(5) $\Pi[1 \,..\, q - 1]$ está calculado corretamente

Consumo de tempo do KMP

```
CALCULA-PREFIXO (P, m)
 1  Π[1] ← 0    k ← 0
 2  para q ← 2 até m faça
 3     enquanto k > 0 e P[k + 1] ≠ P[q] faça
 4        k ← Π[k]
 5     se P[k + 1] = P[q]
 6        então k ← k + 1
 7     Π[q] ← k
 8  devolva Π
```

Método do potencial: Tome $\Phi_q = k$ no fim da iteração q.

Consumo de tempo do KMP

```
CALCULA-PREFIXO (P, m)
 1  Π[1] ← 0    k ← 0
 2  para q ← 2 até m faça
 3     enquanto k > 0 e P[k + 1] ≠ P[q] faça
 4        k ← Π[k]
 5     se P[k + 1] = P[q]
 6        então k ← k + 1
 7     Π[q] ← k
 8  devolva Π
```

Método do potencial: Tome $\Phi_q = k$ no fim da iteração q.

Valor inicial: $\Phi_1 = 0$

Consumo de tempo do KMP

```
CALCULA-PREFIXO (P, m)
 1  Π[1] ← 0    k ← 0
 2  para q ← 2 até m faça
 3     enquanto k > 0 e P[k + 1] ≠ P[q] faça
 4        k ← Π[k]
 5     se P[k + 1] = P[q]
 6        então k ← k + 1
 7     Π[q] ← k
 8  devolva Π
```

Método do potencial: Tome $\Phi_q = k$ no fim da iteração q.

$\Phi_1 = 0$ e $\Phi_q \geq 0$ para $q = 1, \ldots, m$.

Consumo de tempo do KMP

```
CALCULA-PREFIXO (P, m)
 1  Π[1] ← 0    k ← 0
 2  para q ← 2 até m faça
 3     enquanto k > 0 e P[k + 1] ≠ P[q] faça
 4        k ← Π[k]
 5     se P[k + 1] = P[q]
 6        então k ← k + 1
 7     Π[q] ← k
 8  devolva Π
```

Método do potencial: Tome $\Phi_q = k$ no fim da iteração q.

$\Phi_1 = 0$ e $\Phi_q \geq 0$ para $q = 1, \ldots, m$.

c_q: $1+$ número de execuções da linha 4 na iteração q.

Consumo de tempo do KMP

```
CALCULA-PREFIXO (P, m)
 1  Π[1] ← 0    k ← 0
 2  para q ← 2 até m faça
 3     enquanto k > 0 e P[k + 1] ≠ P[q] faça
 4        k ← Π[k]
 5     se P[k + 1] = P[q]
 6        então k ← k + 1
 7     Π[q] ← k
 8  devolva Π
```

Método do potencial: Tome $\Phi_q = k$ no fim da iteração q.

$\Phi_1 = 0$ e $\Phi_q \geq 0$ para $q = 1, \ldots, m$.

c_q: $1+$ número de execuções da linha 4 na iteração q.

Custo amortizado $\hat{c}_q = c_q + \Phi_q - \Phi_{q-1}$.

Consumo de tempo do KMP

```
CALCULA-PREFIXO (P, m)
 1  Π[1] ← 0    k ← 0
 2  para q ← 2 até m faça
 3     enquanto k > 0 e P[k + 1] ≠ P[q] faça
 4        k ← Π[k]
 5     se P[k + 1] = P[q]
 6        então k ← k + 1
 7     Π[q] ← k
 8  devolva Π
```

$\Phi_q = k$ no fim da iteração q e $\Phi_1 = 0$.

c_q: $1+$ número de execuções da linha 4 na iteração q.

Consumo de tempo do KMP

```
CALCULA-PREFIXO (P, m)
 1  Π[1] ← 0    k ← 0
 2  para q ← 2 até m faça
 3     enquanto k > 0 e P[k + 1] ≠ P[q] faça
 4        k ← Π[k]
 5     se P[k + 1] = P[q]
 6        então k ← k + 1
 7     Π[q] ← k
 8  devolva Π
```

$\Phi_q = k$ no fim da iteração q e $\Phi_1 = 0$.

c_q: $1+$ número de execuções da linha 4 na iteração q.

Cada execução da linha 4 faz k diminuir de pelo menos 1.

Logo $\Phi_q \leq \Phi_{q-1} - (c_q - 1) + 1$.

Consumo de tempo do KMP

```
CALCULA-PREFIXO (P, m)
 1  Π[1] ← 0    k ← 0
 2  para q ← 2 até m faça
 3     enquanto k > 0 e P[k + 1] ≠ P[q] faça
 4        k ← Π[k]
 5     se P[k + 1] = P[q]
 6        então k ← k + 1
 7     Π[q] ← k
 8  devolva Π
```

$\Phi_q = k$ no fim da iteração q e $\Phi_1 = 0$.

c_q: $1+$ número de execuções da linha 4 na iteração q.

$\Phi_q \leq \Phi_{q-1} - (c_q - 1) + 1 = \Phi_{q-1} - c_q + 2$.

Consumo de tempo do KMP

```
CALCULA-PREFIXO (P, m)
 1  Π[1] ← 0    k ← 0
 2  para q ← 2 até m faça
 3     enquanto k > 0 e P[k + 1] ≠ P[q] faça
 4        k ← Π[k]
 5     se P[k + 1] = P[q]
 6        então k ← k + 1
 7     Π[q] ← k
 8  devolva Π
```

$\Phi_q = k$ no fim da iteração q e $\Phi_1 = 0$.

c_q: $1+$ número de execuções da linha 4 na iteração q.

$\Phi_q \leq \Phi_{q-1} - (c_q - 1) + 1 = \Phi_{q-1} - c_q + 2$.

Então $\hat{c}_q = c_q + \Phi_q - \Phi_{q-1} \leq c_q - c_q + 2 = 2$.

Consumo de tempo do KMP

```
CALCULA-PREFIXO (P, m)
 1  Π[1] ← 0    k ← 0
 2  para q ← 2 até m faça
 3     enquanto k > 0 e P[k + 1] ≠ P[q] faça
 4        k ← Π[k]
 5     se P[k + 1] = P[q]
 6        então k ← k + 1
 7     Π[q] ← k
 8  devolva Π
```

$\Phi_q = k$ no fim da iteração q e $\Phi_1 = 0$.

c_q: $1+$ número de execuções da linha 4 na iteração q.

$\Phi_q \leq \Phi_{q-1} - (c_q - 1) + 1 = \Phi_{q-1} - c_q + 2$.

Então $\hat{c}_q = c_q + \Phi_q - \Phi_{q-1} \leq c_q - c_q + 2 = 2$.

Custo amortizado por iteração: 2

Consumo de tempo do KMP

```
CALCULA-PREFIXO (P, m)
 1  Π[1] ← 0    k ← 0
 2  para q ← 2 até m faça
 3     enquanto k > 0 e P[k + 1] ≠ P[q] faça
 4        k ← Π[k]
 5     se P[k + 1] = P[q]
 6        então k ← k + 1
 7     Π[q] ← k
 8  devolva Π
```

$\Phi_q = k$ no fim da iteração q e $\Phi_1 = 0$.

c_q: $1+$ número de execuções da linha 4 na iteração q.

$\Phi_q \leq \Phi_{q-1} - (c_q - 1) + 1 = \Phi_{q-1} - c_q + 2$.

Então $\hat{c}_q = c_q + \Phi_q - \Phi_{q-1} \leq c_q - c_q + 2 = 2$.

Consumo de tempo total: $\sum_q c_q \leq \sum_q \hat{c}_q \leq 2m = \Theta(m)$

Algoritmo KMP

```
KMP (T, n, P, m)
 1   Π ← CALCULA-PREFIXO (P, m)
 2   q ← 0
 3   para i ← 1 até n faça
 4       enquanto q > 0 e P[q + 1] ≠ T[i] faça
 5           q ← Π[q]
 6       se P[q + 1] = T[i]
 7           então q ← q + 1
 8       se q = m
 9           então imprima i − q
10                   q ← Π[q]
```

Algoritmo KMP

Invariantes:

(1) $q < m$

(2) $P_q \sqsupseteq T_{i-1}$

(3) q é o maior possível tal que valem (1) e (2)

Algoritmo KMP

Invariantes:

(1) $q < m$

(2) $P_q \sqsupseteq T_{i-1}$

(3) q é o maior possível tal que valem (1) e (2)

Método do potencial: Tome $\Phi_i = q$ no fim da iteração i.

Algoritmo KMP

Invariantes:

(1) $q < m$

(2) $P_q \sqsupseteq T_{i-1}$

(3) q é o maior possível tal que valem (1) e (2)

Método do potencial: Tome $\Phi_i = q$ no fim da iteração i.

Exercício: Complete a análise de correção e a análise do consumo de tempo.

Conjuntos dinâmicos

Conjunto dinâmico, que sofre as seguintes operações:

- inserções,
- remoções,
- buscas.

Como armazenar tal conjunto?

Conjuntos dinâmicos

Conjunto dinâmico, que sofre as seguintes operações:

- inserções,
- remoções,
- buscas.

Como armazenar tal conjunto?

Tempo de pior caso:

	inserção	busca	remoção
LL	$\mathrm{O}(1)$	$\mathrm{O}(n)$	$\mathrm{O}(1)^*$
ABB	$\mathrm{O}(\lg n)$	$\mathrm{O}(\lg n)$	$\mathrm{O}(\lg n)$

* depois da busca

Conjuntos dinâmicos

Conjunto dinâmico, que sofre as seguintes operações:

- inserções,
- remoções,
- buscas.

Como armazenar tal conjunto?

Tempo de pior caso:

	inserção	busca	remoção
LL	$\mathrm{O}(1)$	$\mathrm{O}(n)$	$\mathrm{O}(1)^*$
ABB	$\mathrm{O}(\lg n)$	$\mathrm{O}(\lg n)$	$\mathrm{O}(\lg n)$

Tempo esperado:

hashing	$\mathrm{O}(1)$	$\mathrm{O}(1)$	$\mathrm{O}(1)$

Tabelas de espalhamento

U: universo de chaves possíveis

função de hashing: $h : U \to \{0, \ldots, m-1\}$

m: tamanho da tabela de hashing

Tabelas de espalhamento

U: universo de chaves possíveis

função de hashing: $h : U \to \{0, \ldots, m-1\}$

m: tamanho da tabela de hashing

Colisão: duas chaves distintas x e y tq $h(x) = h(y)$.

Tabelas de espalhamento

U: universo de chaves possíveis

função de hashing: $h : U \to \{0, \ldots, m-1\}$

m: tamanho da tabela de hashing

Colisão: duas chaves distintas x e y tq $h(x) = h(y)$.

Geralmente se escolhe m de modo que
o número de elementos na tabela seja $\Theta(m)$.

Tabelas de espalhamento

U: universo de chaves possíveis

função de hashing: $h : U \to \{0, \ldots, m-1\}$

m: tamanho da tabela de hashing

Colisão: duas chaves distintas x e y tq $h(x) = h(y)$.

Geralmente se escolhe m de modo que
o número de elementos na tabela seja $\Theta(m)$.

Operações: inserções, remoções, e buscas.

Tabelas de espalhamento

U: universo de chaves possíveis

função de hashing: $h : U \to \{0, \ldots, m-1\}$

m: tamanho da tabela de hashing

Colisão: duas chaves distintas x e y tq $h(x) = h(y)$.

Geralmente se escolhe m de modo que
o número de elementos na tabela seja $\Theta(m)$.

Operações: inserções, remoções, e buscas.

Resolução de colisões: usando listas ligadas, por exemplo.

Tabelas de espalhamento

U: universo de chaves possíveis

função de hashing: $h : U \to \{0, \ldots, m-1\}$

m: tamanho da tabela de hashing

Colisão: duas chaves distintas x e y tq $h(x) = h(y)$.

Geralmente se escolhe m de modo que
o número de elementos na tabela seja $\Theta(m)$.

Operações: inserções, remoções, e buscas.

Resolução de colisões: usando listas ligadas, por exemplo.

Aplicações: tabela de símbolos de um compilador.

Boas funções de hashing

- $h(k) = k \bmod m$,
  onde sugere-se usar escolher
  um primo distante de potências de 2 como m.

Boas funções de hashing

- $h(k) = k \bmod m$,
  onde sugere-se usar escolher
  um primo distante de potências de 2 como m.

- $h(k) = m(kA \bmod 1)$,
  onde A é uma constante em $(0, 1)$ e
  $kA \bmod 1$ é a parte fracionária de kA.
  A escolha de m aqui é livre neste caso.
  Knuth sugere que $A = (\sqrt{5} - 1)/2 = 0.6180339\ldots$
  é uma boa escolha em geral.

Boas funções de hashing

- $h(k) = k \bmod m$,
  onde sugere-se usar escolher
  um primo distante de potências de 2 como m.

- $h(k) = m(kA \bmod 1)$,
  onde A é uma constante em $(0, 1)$ e
  $kA \bmod 1$ é a parte fracionária de kA.
  A escolha de m aqui é livre neste caso.
  Knuth sugere que $A = (\sqrt{5} - 1)/2 = 0.6180339\ldots$
  é uma boa escolha em geral.

- hashing universal

Hashing universal

U: conjunto universo (contém todas as possíveis chaves).

m: um número muito menor que $|U|$.

$\mathcal{H}$: conjunto de funções de U em $\{0, \ldots, m-1\}$.

Hashing universal

U: conjunto universo (contém todas as possíveis chaves).

m: um número muito menor que $|U|$.

$\mathcal{H}$: conjunto de funções de U em $\{0, \ldots, m-1\}$.

Se, para cada par de chaves k, ℓ em U, o número de funções h em $\mathcal{H}$ tais que $h(k) = h(\ell)$ é no máximo $|\mathcal{H}|/m$, então $\mathcal{H}$ é uma coleção universal de hashing (para U e m).

Hashing universal

U: conjunto universo (contém todas as possíveis chaves).

m: um número muito menor que $|U|$.

$\mathcal{H}$: conjunto de funções de U em $\{0, \ldots, m-1\}$.

Se, para cada par de chaves k, ℓ em U, o número de funções h em $\mathcal{H}$ tais que $h(k) = h(\ell)$ é no máximo $|\mathcal{H}|/m$, então $\mathcal{H}$ é uma coleção universal de hashing (para U e m).

Teorema: Seja $\mathcal{H}$ uma coleção universal de hashing para U e m, seja $S \subseteq U$ tal que $|S| = m$ e $u \in U$. Se h é escolhida aleatoriamente de $\mathcal{H}$ e X é o número de elementos s em S tais que $h(s) = h(u)$, então $\mathrm{E}[X] \leq 1$.

Prova na disciplina de análise de algoritmos da pós.

Exemplo de coleção universal de hashing

Seja p um primo tal que $U \subseteq \{0, \ldots, p-1\}$.

$\mathbb{Z}_p$: conjunto $\{0, \ldots, p-1\}$.
$\mathbb{Z}_p^*$: conjunto $\{1, \ldots, p-1\}$.

Exemplo de coleção universal de hashing

Seja p um primo tal que $U \subseteq \{0, \ldots, p-1\}$.

$\mathbb{Z}_p$: conjunto $\{0, \ldots, p-1\}$.
$\mathbb{Z}_p^*$: conjunto $\{1, \ldots, p-1\}$.

Para todo a em $\mathbb{Z}_p^*$ e b em $\mathbb{Z}_p$, seja

$$h_{a,b}(k) \;=\; ((ak + b) \bmod p) \bmod m,$$

para todo k em U.

Exemplo de coleção universal de hashing

Seja p um primo tal que $U \subseteq \{0, \ldots, p-1\}$.

$\mathbb{Z}_p$: conjunto $\{0, \ldots, p-1\}$.
$\mathbb{Z}_p^*$: conjunto $\{1, \ldots, p-1\}$.

Para todo a em $\mathbb{Z}_p^*$ e b em $\mathbb{Z}_p$, seja

$$h_{a,b}(k) \;=\; ((ak + b) \bmod p) \bmod m,$$

para todo k em U.

A coleção $\mathcal{H} \;=\; \{h_{a,b} : a \in \mathbb{Z}_p^* \text{ e } \; b \in \mathbb{Z}_p\}$ é universal.

Prova na disciplina de análise de algoritmos da pós.