Algoritmos probabilísticos

Projeto e Análise de Algoritmos

Bruno Prado

Departamento de Computação / UFS

Introdução

- Teoria da probabilidade
  - É utilizada para calcular as chances de ocorrência de um determinado evento possível em um espaço S

Introdução

- Teoria da probabilidade
  - É utilizada para calcular as chances de ocorrência de um determinado evento possível em um espaço S
    - A probabilidade de ocorrência de um evento A é definida por $0 \leq P[A] \leq P[S] = 1$, onde $P[S]$ é a probabilidade de qualquer evento possível acontecer

Introdução

- Teoria da probabilidade
  - É utilizada para calcular as chances de ocorrência de um determinado evento possível em um espaço S
    - A probabilidade de ocorrência de um evento A é definida por $0 \leq P[A] \leq P[S] = 1$, onde $P[S]$ é a probabilidade de qualquer evento possível acontecer
    - As chances de um evento qualquer que não seja A acontecer é $P[\neg A] = P[S] - P[A] = 1 - P[A]$

Introdução

- Teoria da probabilidade
  - É utilizada para calcular as chances de ocorrência de um determinado evento possível em um espaço S
    - A probabilidade de ocorrência de um evento A é definida por $0 \leq P[A] \leq P[S] = 1$, onde $P[S]$ é a probabilidade de qualquer evento possível acontecer
    - As chances de um evento qualquer que não seja A acontecer é $P[\neg A] = P[S] - P[A] = 1 - P[A]$
    - A ocorrência dos eventos independentes A <u>e</u> B tem probabilidade $P[A \cap B] = P[A] \times P[B]$

Introdução

- Teoria da probabilidade
  - É utilizada para calcular as chances de ocorrência de um determinado evento possível em um espaço S
    - A probabilidade de ocorrência de um evento A é definida por $0 \leq P[A] \leq P[S] = 1$, onde $P[S]$ é a probabilidade de qualquer evento possível acontecer
    - As chances de um evento qualquer que não seja A acontecer é $P[\neg A] = P[S] - P[A] = 1 - P[A]$
    - A ocorrência dos eventos independentes A <u>e</u> B tem probabilidade $P[A \cap B] = P[A] \times P[B]$
    - Para que os eventos A <u>ou</u> B aconteçam, as chances são $P[A \cup B] = P[A] + P[B] - P[A \cap B]$

Introdução

- Teoria da probabilidade
  - É utilizada para calcular as chances de ocorrência de um determinado evento possível em um espaço S
    - A probabilidade de ocorrência de um evento A é definida por $0 \leq P[A] \leq P[S] = 1$, onde $P[S]$ é a probabilidade de qualquer evento possível acontecer
    - As chances de um evento qualquer que não seja A acontecer é $P[\neg A] = P[S] - P[A] = 1 - P[A]$
    - A ocorrência dos eventos independentes A <u>e</u> B tem probabilidade $P[A \cap B] = P[A] \times P[B]$
    - Para que os eventos A <u>ou</u> B aconteçam, as chances são $P[A \cup B] = P[A] + P[B] - P[A \cap B]$
    - Um evento impossível ϕ tem probabilidade $P[\phi] = 0$

Introdução

- Teoria da probabilidade
  - Considerando dados honestos com 6 faces, existe um espaço de eventos possíveis $S = \{1, 2, 3, 4, 5, 6\}$

Introdução

- Teoria da probabilidade
  - Considerando dados honestos com 6 faces, existe um espaço de eventos possíveis $S = \{1, 2, 3, 4, 5, 6\}$
  - Qual é a probabilidade dos eventos independentes A (exceto o número 1) e B (qualquer número par)?

Introdução

- Teoria da probabilidade
  - Considerando dados honestos com 6 faces, existe um espaço de eventos possíveis $S = \{1, 2, 3, 4, 5, 6\}$
  - Qual é a probabilidade dos eventos independentes A (exceto o número 1) e B (qualquer número par)?

$$
\begin{aligned}
P[A] &= 1 - P[1] \\
&= 1 - \frac{1}{6} \\
&= \frac{5}{6} \\
&\approx 83\%
\end{aligned}
$$

Introdução

- Teoria da probabilidade
  - Considerando dados honestos com 6 faces, existe um espaço de eventos possíveis $S = \{1, 2, 3, 4, 5, 6\}$
  - Qual é a probabilidade dos eventos independentes A (exceto o número 1) e B (qualquer número par)?

$$\begin{aligned} P[A] &= 1 - P[1] \\ &= 1 - \frac{1}{6} \\ &= \frac{5}{6} \\ &\approx 83\% \end{aligned} \qquad \begin{aligned} P[B] &= P[2] + P[4] + P[6] \\ &= \frac{1}{6} + \frac{1}{6} + \frac{1}{6} \\ &= \frac{1}{2} \\ &= 50\% \end{aligned}$$

Introdução

- Teoria da probabilidade
  - Considerando dados honestos com 6 faces, existe um espaço de eventos possíveis $S = \{1, 2, 3, 4, 5, 6\}$
  - Qual é a probabilidade dos eventos independentes C (evento A ou B) e D (três eventos A seguidos)

Introdução

- Teoria da probabilidade
  - Considerando dados honestos com 6 faces, existe um espaço de eventos possíveis $S = \{1, 2, 3, 4, 5, 6\}$
  - Qual é a probabilidade dos eventos independentes C (evento A ou B) e D (três eventos A seguidos)

$$
\begin{aligned}
P[C] &= P[A \cup B] \\
&= P[A] + P[B] - P[A \cap B] \\
&= \frac{5}{6} + \frac{1}{2} - \left(\frac{5}{6} \times \frac{1}{2}\right) \\
&= \frac{11}{12} \\
&\approx 92\%
\end{aligned}
$$

Introdução

- Teoria da probabilidade
  - Considerando dados honestos com 6 faces, existe um espaço de eventos possíveis $S = \{1, 2, 3, 4, 5, 6\}$
  - Qual é a probabilidade dos eventos independentes C (evento A ou B) e D (três eventos A seguidos)

$$\begin{aligned} P[C] &= P[A \cup B] \\ &= P[A] + P[B] - P[A \cap B] \\ &= \frac{5}{6} + \frac{1}{2} - \left(\frac{5}{6} \times \frac{1}{2}\right) \\ &= \frac{11}{12} \\ &\approx 92\% \end{aligned}$$

$$\begin{aligned} P[D] &= P[A \cap A \cap A] \\ &= P[A] \times P[A] \times P[A] \\ &= \frac{5}{6} \times \frac{5}{6} \times \frac{5}{6} \\ &= \frac{125}{216} \\ &\approx 58\% \end{aligned}$$

Introdução

- Teoria da probabilidade
  - Caso os eventos sejam dependentes, ou seja, a ocorrência de um determinado evento *B* influenciar as chances de outro evento *A* acontecer, é utilizada a probabilidade condicional ou teorema de Bayes

	B	$\neg B$	
A	r	s	$r + s$
$\neg A$	u	v	$u + v$
	$r + u$	$s + v$	$r + s + u + v$

Introdução

- Teoria da probabilidade
  - Caso os eventos sejam dependentes, ou seja, a ocorrência de um determinado evento *B* influenciar as chances de outro evento *A* acontecer, é utilizada a probabilidade condicional ou teorema de Bayes

	B	$\neg B$	
A	r	s	$r+s$
$\neg A$	u	v	$u+v$
	$r+u$	$s+v$	$r+s+u+v$

$$P[A] = \frac{r+s}{r+s+u+v}$$

Introdução

- Teoria da probabilidade
  - Caso os eventos sejam dependentes, ou seja, a ocorrência de um determinado evento *B* influenciar as chances de outro evento *A* acontecer, é utilizada a probabilidade condicional ou teorema de Bayes

	B	¬B	
A	r	s	r + s
¬A	u	v	u + v
	r + u	s + v	r + s + u + v

$$P\left[B\right] = \frac{r+u}{r+s+u+v}$$

Introdução

- Teoria da probabilidade
  - Caso os eventos sejam dependentes, ou seja, a ocorrência de um determinado evento B influenciar as chances de outro evento A acontecer, é utilizada a probabilidade condicional ou teorema de Bayes

	B	¬B	
A	r	s	r + s
¬A	u	v	u + v
	r + u	s + v	r + s + u + v

$$P\,[A|B] = \frac{r}{r+u} = \frac{P[B|A] \times P[A]}{P[B]}$$

Introdução

- Teoria da probabilidade
  - Caso os eventos sejam dependentes, ou seja, a ocorrência de um determinado evento B influenciar as chances de outro evento A acontecer, é utilizada a probabilidade condicional ou teorema de Bayes

	B	$\neg B$	
A	r	s	$r+s$
$\neg A$	u	v	$u+v$
	$r+u$	$s+v$	$r+s+u+v$

$$P\left[B|A\right] = \frac{r}{r+s} = \frac{P[A|B] \times P[B]}{P[A]}$$

Introdução

- Teoria da probabilidade
  - Considerando os dados abaixo sobre doenças respiratórias, com os eventos de complicações graves (A) e de infecções assintomáticas (B), qual é a probabilidade $P[A|B]$ de desenvolver a forma grave da doença estando assintomático?

	B	$\neg B$	
A	2	48	50
$\neg A$	24	33	57
	26	81	107

Introdução

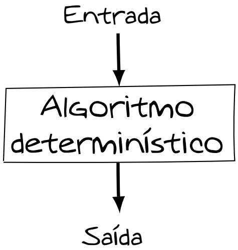
- Teoria da probabilidade
  - Considerando os dados abaixo sobre doenças respiratórias, com os eventos de complicações graves (A) e de infecções assintomáticas (B), qual é a probabilidade $P[A|B]$ de desenvolver a forma grave da doença estando assintomático?

	B	$\neg B$	
A	2	48	50
$\neg A$	24	33	57
	26	81	107

$$
\begin{aligned}
P[A|B] &= \frac{P[B|A] \times P[A]}{P[B]} \\
&= \frac{\frac{2}{50} \times \frac{50}{107}}{\frac{26}{107}} \\
&= \frac{2}{26} \\
&\approx 7,7\%
\end{aligned}
$$

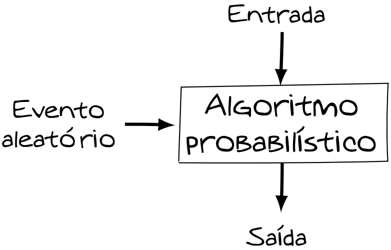
Introdução

- Algoritmos determinísticos $\times$ probabilísticos
  - Em um algoritmo determinístico, cada passo de sua execução é predeterminado, sempre gerando a mesma saída para uma entrada específica



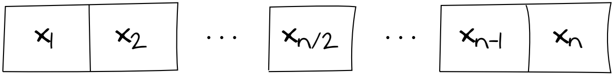
Introdução

- Algoritmos determinísticos $\times$ probabilísticos
  - Em um algoritmo determinístico, cada passo de sua execução é predeterminado, sempre gerando a mesma saída para uma entrada específica
  - Já os algoritmos probabilísticos não dependem somente da entrada fornecida, mas também de eventos aleatórios na execução de suas operações



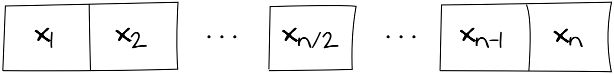
Introdução

- Algoritmos determinísticos $\times$ probabilísticos
  - Considerando um conjunto de números não ordenados $x_1, x_2, \ldots, x_n$, selecione um número que pertença a metade superior do conjunto, ou seja, um número que é maior ou igual do que a mediana



Introdução

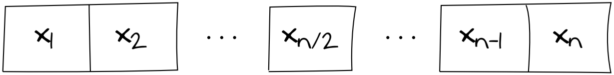
- Algoritmos determinísticos × probabilísticos
  - Considerando um conjunto de números não ordenados $x_1, x_2, \ldots, x_n$, selecione um número que pertença a metade superior do conjunto, ou seja, um número que é maior ou igual do que a mediana



Determinístico: escolhe o maior valor de $n/2$ comparações entre $n/2 + 1$ números do vetor

Introdução

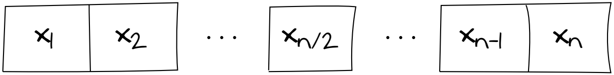
- Algoritmos determinísticos $\times$ probabilísticos
  - Considerando um conjunto de números não ordenados $x_1, x_2, \ldots, x_n$, selecione um número que pertença a metade superior do conjunto, ou seja, um número que é maior ou igual do que a mediana



Probabilístico: são escolhidos k números que possuem pelo menos 50% de chance de estarem na metade superior do vetor

Introdução

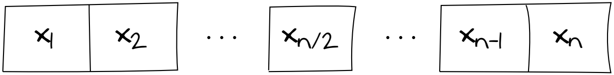
- Algoritmos determinísticos $\times$ probabilísticos
  - Considerando um conjunto de números não ordenados $x_1, x_2, \ldots, x_n$, selecione um número que pertença a metade superior do conjunto, ou seja, um número que é maior ou igual do que a mediana



A probabilidade de nenhum destes k números estarem na metade superior do conjunto é de $0{,}5^{-k}$

Introdução

- Algoritmos determinísticos × probabilísticos
  - Considerando um conjunto de números não ordenados $x_1, x_2, \ldots, x_n$, selecione um número que pertença a metade superior do conjunto, ou seja, um número que é maior ou igual do que a mediana



Com $k - 1$ comparações, a probabilidade deste algoritmo escolher corretamente o número da metade superior do conjunto é de $1 - 0{,}5^{-k}$

Introdução

- Abordagens Monte Carlo e Las Vegas
  - Monte Carlo: pode gerar um resultado incorreto com baixa probabilidade, mas apresenta um tempo de execução que pode ser melhor do que seu respectivo algoritmo determinístico mais eficiente

Introdução

- Abordagens Monte Carlo e Las Vegas
  - Monte Carlo: pode gerar um resultado incorreto com baixa probabilidade, mas apresenta um tempo de execução que pode ser melhor do que seu respectivo algoritmo determinístico mais eficiente
  - Las Vegas: a solução gerada sempre é correta, entretanto, a quantidade de passos executados não é garantida, devendo ser utilizado quando o tempo de execução esperado é baixo

Introdução

- Abordagens Monte Carlo e Las Vegas
  - Monte Carlo: pode gerar um resultado incorreto com baixa probabilidade, mas apresenta um tempo de execução que pode ser melhor do que seu respectivo algoritmo determinístico mais eficiente
  - Las Vegas: a solução gerada sempre é correta, entretanto, a quantidade de passos executados não é garantida, devendo ser utilizado quando o tempo de execução esperado é baixo

	Monte Carlo	Las Vegas
Solução	Probabilística	Determinística
Tempo	Determinístico	Probabilístico

Algoritmos probabilísticos

- Método da congruência linear
  - É essencial para a implementação eficiente dos algoritmos probabilísticos, uma vez que as mesmas sequências de números pseudoaleatórios são criadas deterministicamente a partir de um valor inicial (*seed*)

$$R(i) = [a \times R(i-1) + b] \bmod n$$

Algoritmos probabilísticos

- Método da congruência linear
  - É essencial para a implementação eficiente dos algoritmos probabilísticos, uma vez que as mesmas sequências de números pseudoaleatórios são criadas deterministicamente a partir de um valor inicial (*seed*)

$$R(i) = [a \times R(i-1) + b] \bmod n$$

A semente é definida por $R(0)$

Algoritmos probabilísticos

- Método da congruência linear
  - É essencial para a implementação eficiente dos algoritmos probabilísticos, uma vez que as mesmas sequências de números pseudoaleatórios são criadas deterministicamente a partir de um valor inicial (*seed*)

$$R(i) = [a \times R(i-1) + b] \bmod n$$

Os parâmetros a e b devem ser escolhidos adequadamente para evitar a repetição dos números gerados na sequência

Algoritmos probabilísticos

- Método da congruência linear
  - É essencial para a implementação eficiente dos algoritmos probabilísticos, uma vez que as mesmas sequências de números pseudoaleatórios são criadas deterministicamente a partir de um valor inicial (*seed*)

$$R(i) = [a \times R(i-1) + b] \bmod n$$

Não é adequado para aplicações criptográficas!

Algoritmos probabilísticos

- Método da congruência linear
  - Função *rand*() da biblioteca *stdlib* de C/C++: retorna os 15 bits mais significativos $R(i)_{30:16}$
    $R(i) = [1103515245 \times R(i-1) + 12345] \bmod 2^{31}$

Algoritmos probabilísticos

- Método da congruência linear
  - Função *rand*() da biblioteca *stdlib* de C/C++: retorna os 15 bits mais significativos $R(i)_{30:16}$

    $$R(i) = [1103515245 \times R(i-1) + 12345] \bmod 2^{31}$$

  - Classe *java.util.Random*: pode gerar um número inteiro de 32 bits de $R(i)_{47:16}$

    $$R(i) = [25214903917 \times R(i-1) + 11] \bmod 2^{48}$$

Algoritmos probabilísticos

- Método da congruência linear

```c
// Variável global do próximo número R(i)
static unsigned long next = 1;
// Procedimento da semente
void srand(unsigned seed) {
    // R(0) = semente
    next = seed;
}
// Função pseudoaleatória
int rand() {
    // Calculando R(i) a partir de R(i - 1)
    next = (1103515245 * next)  + 12345;
    // Retornando R(i)[30:16]
    return ((next >> 16) & 0x7FFF);
}
```

Algoritmos probabilísticos

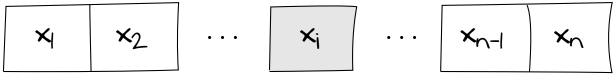
- Método da congruência linear

```c
// Variável global do próximo número R(i)
static unsigned long next = 1;
// Procedimento da semente
void srand(unsigned seed) {
    // R(0) = semente
    next = seed;
}
// Função pseudoaleatória
int rand() {
    // Calculando R(i) a partir de R(i - 1)
    next = (1103515245 * next)  + 12345;
    // Retornando R(i)[30:16]
    return ((next >> 16) & 0x7FFF);
}
```

$$srand(123) \quad \rightarrow \quad \begin{aligned} rand() &= 6727 \\ rand() &= 22524 \\ rand() &= 25453 \end{aligned}$$

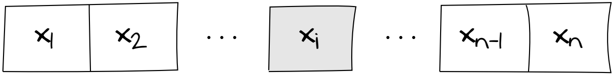
Algoritmos probabilísticos

- Abordagem Monte Carlo
  - Considere a busca de um caractere x em uma cadeia com n elementos, executando no máximo k iterações para encontrar o símbolo procurado



Algoritmos probabilísticos

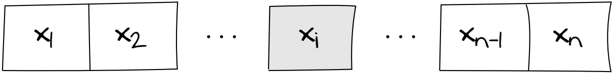
- Abordagem Monte Carlo
  - Considere a busca de um caractere x em uma cadeia com n elementos, executando no máximo k iterações para encontrar o símbolo procurado



$$P[\text{não ser } x] = \frac{2^8 - 1}{2^8} = 0{,}9961$$

Algoritmos probabilísticos

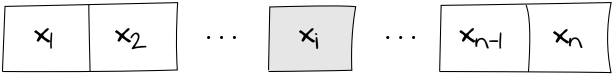
- Abordagem Monte Carlo
  - Considere a busca de um caractere *x* em uma cadeia com *n* elementos, executando no máximo *k* iterações para encontrar o símbolo procurado



$$P[\text{não ser } x \text{ por } k \text{ vezes}] = P[\text{não ser } x]^k = 0{,}9961^k$$

Algoritmos probabilísticos

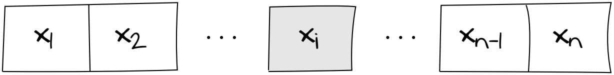
- Abordagem Monte Carlo
  - Considere a busca de um caractere *x* em uma cadeia com *n* elementos, executando no máximo *k* iterações para encontrar o símbolo procurado



$$P[\text{não ser } x \text{ por } k \text{ vezes}] = 0{,}961^k$$

Algoritmos probabilísticos

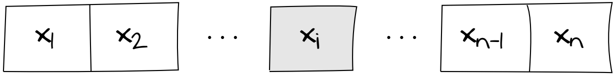
- Abordagem Monte Carlo
  - Considere a busca de um caractere x em uma cadeia com n elementos, executando no máximo k iterações para encontrar o símbolo procurado



P[encontrar x] = 1 - P[não ser x por k vezes]

Algoritmos probabilísticos

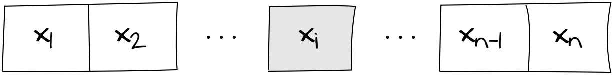
- Abordagem Monte Carlo
  - Considere a busca de um caractere x em uma cadeia com n elementos, executando no máximo k iterações para encontrar o símbolo procurado



$$P[\text{encontrar } x] = 1 - P[\text{não ser } x]^k$$

Algoritmos probabilísticos

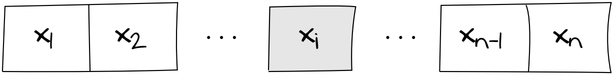
- Abordagem Monte Carlo
  - Considere a busca de um caractere *x* em uma cadeia com *n* elementos, executando no máximo *k* iterações para encontrar o símbolo procurado



$$P[\text{encontrar } x] = 1 - 0{,}9961^k$$

Algoritmos probabilísticos

- Abordagem Monte Carlo
  - Considere a busca de um caractere *x* em uma cadeia com *n* elementos, executando no máximo *k* iterações para encontrar o símbolo procurado



$$P[\text{encontrar } x] = 1 - 0{,}9961^k$$

$$\uparrow k \longrightarrow \uparrow P[\text{encontrar } x]$$

Algoritmos probabilísticos

- Abordagem Monte Carlo
  - Considere a busca de um caractere *x* em uma cadeia com *n* elementos, executando no máximo *k* iterações para encontrar o símbolo procurado

```c
// Função busca por caractere (Monte Carlo)
int32_t bmc(char* s, uint32_t n, char x, uint32_t k) {
    // Índice do resultado
    int32_t r = -1;
    // k tentativas
    for(uint32_t i = 0; i < k && r == -1; i++) {
        // Calculando posição pseudoaleatória
        uint32_t pos = rand() % n;
        // Checando se caractere é igual a x
        if(s[pos] == x) r = pos;
    }
    // Retornando resultado
    return r;
}
```

Algoritmos probabilísticos

- Abordagem Monte Carlo
  - Considere a busca de um caractere x em uma cadeia com n elementos, executando no máximo k iterações para encontrar o símbolo procurado

```c
// Função busca por caractere (Monte Carlo)
int32_t bmc(char* s, uint32_t n, char x, uint32_t k) {
    // Índice do resultado
    int32_t r = -1;
    // k tentativas
    for(uint32_t i = 0; i < k && r == -1; i++) {
        // Calculando posição pseudoaleatória
        uint32_t pos = rand() % n;
        // Checando se caractere é igual a x
        if(s[pos] == x) r = pos;
    }
    // Retornando resultado
    return r;
}
```

Espaço $\Theta(n)$ e tempo entre $\Omega(1)$ e $O(k)$

Algoritmos probabilísticos

- Abordagem Monte Carlo
  - No cenário de busca do caractere x em uma sequência de dados com 500 MB (500×10^6 caracteres), a busca probabilística é ajustada para realizar até 1.000 comparações (k)

$$
\begin{aligned}
P[encontrar\ x] &= 1 - (0,9961)^{10^3} \\
&= 1 - 0,02 \\
&= 0,98
\end{aligned}
$$

Algoritmos probabilísticos

- Abordagem Monte Carlo
  - No cenário de busca do caractere x em uma sequência de dados com 500 MB (500×10^6 caracteres), a busca probabilística é ajustada para realizar até 1.000 comparações (k)

$$
\begin{aligned}
P[encontrar\ x] &= 1 - (0,9961)^{10^3} \\
&= 1 - 0,02 \\
&= 0,98
\end{aligned}
$$

Com até 1.000 tentativas, existe uma probabilidade de sucesso de 98% para encontrar x

Algoritmos probabilísticos

- Abordagem Monte Carlo
  - Pequeno teorema de Fermat

$$primo(p) \land 1 \leq a < p \rightarrow a^{p-1} \equiv 1 \bmod p$$

Algoritmos probabilísticos

- Abordagem Monte Carlo
  - Pequeno teorema de Fermat

$$primo(p) \wedge 1 \leq a < p \rightarrow a^{p-1} \equiv 1 \bmod p$$

  - Teste de primalidade de Fermat

$$primo\_fermat(n) = \begin{cases} a^{n-1} \equiv 1 \bmod n & , \ provável\ primo \\ a^{n-1} \not\equiv 1 \bmod n & , \ composto \end{cases}$$

Algoritmos probabilísticos

- Abordagem Monte Carlo
  - Pequeno teorema de Fermat

$$primo(p) \wedge 1 \leq a < p \rightarrow a^{p-1} \equiv 1 \bmod p$$

  - Teste de primalidade de Fermat

$$primo\_fermat(n) = \begin{cases} a^{n-1} \equiv 1 \bmod n & , \textit{provável primo} \\ a^{n-1} \not\equiv 1 \bmod n & , \textit{composto} \end{cases}$$

Para $n < 10.000$ existem apenas 22 valores que podem gerar um resultado incorreto (0,22%)

Algoritmos probabilísticos

- Abordagem Monte Carlo
  - O teste de primalidade de Fermat executa por k iterações para determinar se n é um provável primo

```c
// Teste de primalidade de Fermat
uint8_t primo_fermat(uint64_t n, uint32_t k) {
    // 1 = provável primo
    uint8_t r = 1;
    // k tentativas
    for(uint32_t i = 0; i < k && r == 1; i++) {
        // Obtendo a (1 <= a < n)
        uint64_t a = 1 + rand64() % (n - 1);
        // r = a ^ (p - 1)
        r = expmod(a, n - 1, n);
    }
    // Retornando resultado
    return r;
}
```

Algoritmos probabilísticos

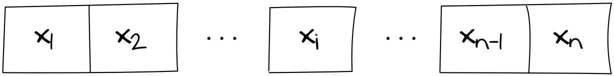
- Abordagem Monte Carlo
  - O teste de primalidade de Fermat executa por k iterações para determinar se n é um provável primo

```c
// Teste de primalidade de Fermat
uint8_t primo_fermat(uint64_t n, uint32_t k) {
    // 1 = provável primo
    uint8_t r = 1;
    // k tentativas
    for(uint32_t i = 0; i < k && r == 1; i++) {
        // Obtendo a (1 <= a < n)
        uint64_t a = 1 + rand64() % (n - 1);
        // r = a ^ (p - 1)
        r = expmod(a, n - 1, n);
    }
    // Retornando resultado
    return r;
}
```

Espaço $\Theta(1)$ e tempo entre $\Omega(\log_2 n)$ e $O(k \log_2 n)$

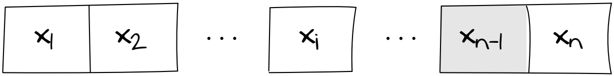
Algoritmos probabilísticos

- Abordagem Las Vegas
  - Considere a busca de um caractere x em uma cadeia com n elementos, executando comparações em posições pseudoaleatórias até encontrar x



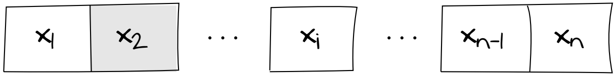
Algoritmos probabilísticos

- Abordagem Las Vegas
  - Considere a busca de um caractere x em uma cadeia com n elementos, executando comparações em posições pseudoaleatórias até encontrar x



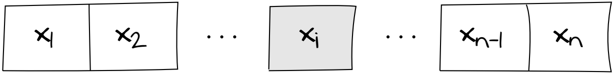
Algoritmos probabilísticos

- Abordagem Las Vegas
  - Considere a busca de um caractere x em uma cadeia com n elementos, executando comparações em posições pseudoaleatórias até encontrar x



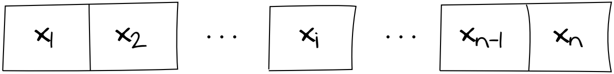
Algoritmos probabilísticos

- Abordagem Las Vegas
  - Considere a busca de um caractere x em uma cadeia com n elementos, executando comparações em posições pseudoaleatórias até encontrar x



Algoritmos probabilísticos

- Abordagem Las Vegas
  - Considere a busca de um caractere x em uma cadeia com n elementos, executando comparações em posições pseudoaleatórias até encontrar x



Se estiver na cadeia, o símbolo sempre é encontrado

Algoritmos probabilísticos

- Abordagem Las Vegas
  - Considere a busca de um caractere *x* em uma cadeia com *n* elementos, executando comparações em posições pseudoaleatórias até encontrar *x*

```
// Função busca por caractere (Las Vegas)
int32_t blv(char* s, uint32_t n, char x) {
    // Índice do resultado
    int32_t r = -1;
    // Enquanto x não foi encontrado
    while(r == -1 && pendentes(n)) {
        // Calculando posição pseudoaleatória
        uint32_t pos = rand_pendentes();
        // Comparando e marcando posição
        if(comparar(s, pos, x)) r = pos;
    }
    // Retornando resultado
    return r;
}
```

Algoritmos probabilísticos

- Abordagem Las Vegas
  - Considere a busca de um caractere x em uma cadeia com n elementos, executando comparações em posições pseudoaleatórias até encontrar x

```
// Função busca por caractere (Las Vegas)
int32_t blv(char* s, uint32_t n, char x) {
    // Índice do resultado
    int32_t r = -1;
    // Enquanto x não foi encontrado
    while(r == -1 && pendentes(n)) {
        // Calculando posição pseudoaleatória
        uint32_t pos = rand_pendentes();
        // Comparando e marcando posição
        if(comparar(s, pos, x)) r = pos;
    }
    // Retornando resultado
    return r;
}
```

Espaço $\Theta(n)$ e tempo entre $\Omega(1)$ e $O(n)$

Algoritmos probabilísticos

- Abordagem Las Vegas
  - Particionamento do Quicksort

```
// Particionamento randômico do Hoare
void hoare_lv(int32_t* V, int32_t i, int32_t j) {
    // Troca do pivô por aleatório
    trocar(&V[i], &V[i + (rand() % (j - i + 1))]);
    // Chamada do particionamento
    return hoare(V, i, j);
}
// Particionamento randômico do Lomuto
void lomuto_lv(int32_t* V, int32_t i, int32_t j) {
    // Troca do pivô por aleatório
    trocar(&V[j], &V[i + (rand() % (j - i + 1))]);
    // Chamada do particionamento
    return lomuto(V, i, j);
}
```

Algoritmos probabilísticos

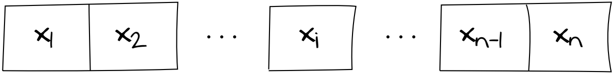
- Abordagem Las Vegas
  - Particionamento do Quicksort

```c
// Particionamento randômico do Hoare
void hoare_lv(int32_t* V, int32_t i, int32_t j) {
    // Troca do pivô por aleatório
    trocar(&V[i], &V[i + (rand() % (j - i + 1))]);
    // Chamada do particionamento
    return hoare(V, i, j);
}
// Particionamento randômico do Lomuto
void lomuto_lv(int32_t* V, int32_t i, int32_t j) {
    // Troca do pivô por aleatório
    trocar(&V[j], &V[i + (rand() % (j - i + 1))]);
    // Chamada do particionamento
    return lomuto(V, i, j);
}
```

A escolha pseudoaleatória do pivô busca reduzir as chances de escolha do maior ou menor elemento da partição (pior caso)

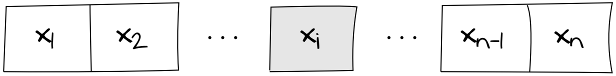
Algoritmos probabilísticos

- Abordagem Las Vegas
  - Particionamento do Quicksort



Algoritmos probabilísticos

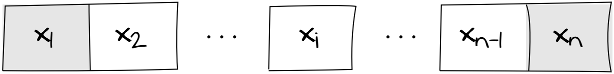
- Abordagem Las Vegas
  - Particionamento do Quicksort



$$P[x_i \text{ ser a mediana}] = \frac{1}{n}$$

Algoritmos probabilísticos

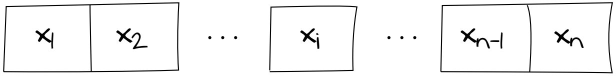
- Abordagem Las Vegas
  - Particionamento do Quicksort



$$P[x_i \text{ ser a mediana e } i = 1/n] = \frac{1}{n} \times \frac{1}{n}$$

Algoritmos probabilísticos

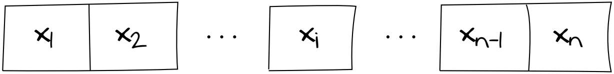
- Abordagem Las Vegas
  - Particionamento do Quicksort



$$P[x_i \text{ ser a mediana}] > P[x_i \text{ ser a mediana e } i = 1/n]$$

Algoritmos probabilísticos

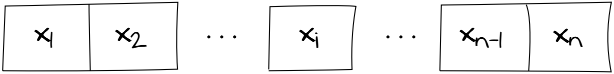
- Abordagem Las Vegas
  - Particionamento do Quicksort



Qualquer que seja o pivô escolhido,
a ordenação será realizada corretamente

Algoritmos probabilísticos

- Abordagem Las Vegas
  - Particionamento do Quicksort



Qualquer que seja o pivô escolhido,
a ordenação será realizada corretamente,
com um tempo de execução pseudoaleatório