ALGORITMOS PROBABILÍSTICOS

DEFINIÇÃO DE ALGORITMOS PROBABILÍSTICOS

ALGORITMOS PROBABILÍSTICOS

- A grande maioria dos algoritmos estudados no inicio do curso de Engenharia são algoritmos determinísticos.
- Um **algoritmo probabilístico** é um programa que toma decisões aleatórias durante sua execução. Decisões aleatórias independem da entrada e podem ser diferentes quando se executa o algoritmo para uma mesma entrada.
- Ao usar decisões aleatórias, o algoritmo probabilístico, pode dar uma resposta errada.
- De que adianta uma resposta errada, para nossas decisões totalmente determinística.

ALGORITMOS PROBABILÍSTICOS

- Vamos imaginar um problema numérico difícil, como por exemplo a trajetória para fazer o trajeto de ida e volta da terra a marte usando a gravidade da lua como catapulta, afim de ter o menor gasto de combustível.
- Temos a disposição dois tipos de algoritmos. Um determinístico que dura mais de 600 minutos para dar a resposta correta. E temos um outro algoritmo que fornece uma solução em 6 minutos, só que 0,005% das vezes temos uma resposta errada.
- Qual o Algoritmo tem a resposta mais confiável para ser implementado?

ALGORITMOS PROBABILÍSTICOS

- Apesar do algoritmo ser determinístico, roda em um computador que é falível. Estatisticamente todos os comutadores tem um erro indetectável a cada 10 mil horas de trabalho. (inversão de bit).
- Sendo assim, o algoritmo determinístico em 10 horas de trabalho me dará um resposta errada indetectável em 0,01% das vezes.
- No algoritmo probabilístico que tem seu erro intrínseco de 0,005%, que roda no mesmo hardware, então temos 0,0051% das vezes uma resposta errada a nossa execução.
- Então dependendo do custo computacional é mais vantagem usar um algoritmo probabilístico a um algoritmo determinístico.

ALGORITMOS PROBABILÍSTICOS

- Números Pseudoaleatórios (PRNs):
- São gerados por um algoritmo determinístico.
- O algoritmo começa com um valor inicial (a ***semente*** ou ***seed***), e a partir daí, gera uma sequência que parece aleatória.
- No entanto, se você souber a semente, pode reproduzir a sequência exata.
- Se a semente for a mesma, a sequência gerada será sempre a mesma.
- É por isso que, muitas vezes, é possível "reproduzir" um bug ou uma situação em um programa que usa pseudoaleatoriedade, desde que se saiba a semente inicial

ALGORITMOS PROBABILÍSTICOS

- Números Aleatórios de Verdade (TRNs)
- São gerados por processos físicos e imprevisíveis do mundo real.
- Pense em coisas como o ruído térmico de um resistor, o tempo entre cliques do mouse de um usuário, ou o decaimento radioativo.
- Esses eventos não podem ser previstos e, portanto, geram uma sequência de números que é genuinamente imprevisível.
- O exemplo do Diodo Zener polarizado Reversamente com uma tensão bem próxima a tensão de Zener, utilizando assim a aleatoriedade do rompimento da barreira de potencial.

ALGORITMOS PROBABILÍSTICOS

Por que a Aleatoriedade de Verdade Importa?

A importância de gerar números verdadeiramente aleatórios vai além da simples imprevisibilidade.

Ela é crítica para a segurança e a robustez de sistemas.

Criptografia e Segurança Essa é, provavelmente, a aplicação mais fácil e impactante para demonstrar. A segurança de protocolos como o ***HTTPS*** depende da geração de chaves criptográficas fortes.

Se um hacker souber que seu sistema está usando um gerador pseudoaleatório com uma semente fraca ou previsível (por exemplo, a hora do sistema), ele pode adivinhar a chave criptográfica e quebrar a segurança.

ALGORITMOS PROBABILÍSTICOS

Solução: Chaves criptográficas devem ser geradas usando números verdadeiramente aleatórios.

Isso garante que a chave seja imprevisível e, portanto, impossível de ser adivinhada ou reproduzida.

Os sistemas operacionais modernos usam fontes de entropia

movimento do mouse,

o tempo entre as entradas do teclado,

para criar sementes verdadeiramente aleatórias.

ALGORITMOS PROBABILÍSTICOS

Simuladores de Eventos e Monte Carlo

Em áreas como finanças, física e engenharia, a aleatoriedade é usada para simular o comportamento de sistemas complexos.

O método de Monte Carlo é um exemplo clássico.

- ***Problema:*** Se as simulações usarem um gerador pseudoaleatório com um ciclo de repetição muito curto, os resultados podem ser enviesados, não refletindo a verdadeira aleatoriedade do sistema que está sendo modelado.
- ***Solução:*** Usar números aleatórios de alta qualidade permite que as simulações sejam mais precisas e reflitam melhor a realidade, o que é crucial para tomar decisões informadas em áreas de alto risco.

ALGORITMOS PROBABILÍSTICOS

Jogos e Sorteios

Embora muitos jogos possam usar pseudoaleatoriedade, a aleatoriedade de verdade é fundamental quando há dinheiro ou resultados importantes envolvidos, como em cassinos online e loterias.

Problema: Se um jogo de poker online usar uma semente previsível para embaralhar as cartas, um jogador mal-intencionado poderia "hackear" o algoritmo e prever a ordem das cartas.

Solução: As casas de aposta e loterias usam geradores de números aleatórios de hardware (TRNGs) para garantir que os resultados dos sorteios e jogos sejam justos e imprevisíveis, preservando a integridade do sistema.

ALGORITMOS PROBABILÍSTICOS
IMPLEMENTE ESTE CÓDIGO E VERIFIQUE O RESULTADO

```
import random

def gerar_numeros_aleatorios_sem_semente():

print("Gerando 10 números aleatórios sem semente:")

  numeros = [random.randint(1, 100) for _ in range(10)]

  print(numeros)

  print("Se você executar este programa novamente, a sequência de números será diferente.")

gerar_numeros_aleatorios_sem_semente()
```

ALGORITMOS PROBABILÍSTICOS
IMPLEMENTE ESTE CÓDIGO E VERIFIQUE O RESULTADO

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
void gerar_numeros_com_semente_fixa(int semente) {
    printf("Gerando 10 numeros com semente fixa (%d):\n", semente);
    srand(semente); // Define a semente para o gerador de numeros aleatorios
    for (int i = 0; i < 10; i++) {
        // Gera um numero pseudoaleatorio entre 1 e 100
        int numero_aleatorio = rand() % 100 + 1;
        printf("%d ", numero_aleatorio);
    }
    printf("\n");
}
int main() {
    gerar_numeros_com_semente_fixa(1);
    printf("Execute o programa novamente para ver que esta sequencia de numeros sera a mesma.\n");
    return 0;
}
```

ALGORITMOS PROBABILÍSTICOS

- O primeiro Algoritmo Probabilístico existente foi executado pela máquina de Touring Probabilística. Apesar do Hardware (famoso em o "Jogo da Imitação") tem acesso a uma fonte de dados aleatórios "0" ou "1", e o momento de leitura desta fonte, também ocorre aleatoriamente.
- Um dos problemas mais antigos da computação é a geração de números aleatórios. Diversos métodos estatísticos, fazem uso de valores aleatórios sem seus processos.
- O primeiro registro que temos de uma sequência de números aleatórios são as tabelas de números aleatórios com cerca de 41 mil dígitos aleatórios.

ALGORITMOS PROBABILÍSTICOS

- A primeira ideia para criar condições aleatórias no PC era carregar as tabelas em local de memoria para ser acessada quando necessário.
- Para solucionar esta deficiência os computadores geram sequencias de números pseudoaleatórios. Números pseudoaleatórios são sequencias que possuem características de sequencias aleatórias (só que geradas de forma determinística).
- Isso é feito por meio de uma função que gera um novo numero a partir de número de entrada de forma que a sequência resultante alimenta uma função com a saída anterior com características de sequências aleatórias.

ALGORITMOS PROBABILÍSTICOS

```
VAR
    f_anterior : INTEIRO
    a : CONSTANTE INTEIRO = 3
    b : CONSTANTE INTEIRO = 5
    m : CONSTANTE INTEIRO = 31
    n : INTEIRO
INÍCIO
  // Define o valor inicial (semente)
  f_anterior <- 2
  // Loop para gerar a sequência
  PARA n DE 1 ATÉ 10 (ou qualquer outro limite) FAÇA
   // Calcula o próximo número da sequência
   f_atual <- (a * f_anterior + b) MOD m
  // Imprime o número gerado
  ESCREVA("f(", n, ") = ", f_atual)
  // Atualiza f_anterior para a próxima iteração
  f_anterior <- f_atual
 FIM PARA
FIM
```

ALGORITMOS PROBABILÍSTICOS

- Por exemplo f(n) = (a*f(n-1)+b) mode m,
- Quando a = 3, b=5, m =31 e f(0) = 2
- Rodar este algoritmo até que sequencia começar o processo repetitivo