Dawkins' Weasel Program

- Bruno França do Prado
- Fernanda Pinto Lopes
- Vanessa de Souza Câmara

O que é Teoria da Evolução?

- Charles Darwin
- “A Origem das Espécies” (1859)
- Expedição na América do Sul (Semelhanças)
- Criação da Teoria
- Publicação do Estudo

Algoritmo Dawkins' Weasel Program

- Objetivo: demonstrar que o processo que impulsiona sistemas evolutivos - combinação de mudanças aleatórias - é diferente do acaso.

  Iniciamos uma sequência aleatória de 28 caracteres (geração 1)

  Fazemos 100 cópias da sequência do alvo

  Em cada uma das cópias, para cada char há 5% de probabilidade de mutar e 95% de não mutar para um novo caracter aleatório.

  Comparar cada cópia com o alvo "METHINKS IT IS LIKE A WEASEL", e quanto mais próximo do alvo (letras iguais em posições iguais), melhor é aquela string

  Se alguma das novas sequências de caracteres tiver uma pontuação perfeita, alvo == cópia o programa deve parar. Caso contrário, pegar a cópia de maior pontuação e voltar para a etapa 2

Aplicação do problema em C

```c
#include <stdlib.h>
#include <string.h>
#include<climits>
char alvo[] = "METHINKS IT IS LIKE A WEASEL", letras[] = "ABCDEFGH
#define QTE_LETRAS (sizeof(letras) - 1)
#define MUTAR 20
#define COPIAS 100
// retorna inteiro aleatório de 0 a n-1
int int_rand(int n){
    int r = rand(), rand_max = RAND_MAX-(RAND_MAX % n);
    while(r >= rand_max){
        r = rand();
    }
    r = r / (rand_max/n);
    return r;
}
// quantidade de caracters diferentes entre a string a e b
int comparar(char *a, char *b){
    int i, sum = 0;
    for (i = 0; a[i]; i++)
        sum += (a[i] != b[i]);
    return sum;
}
// cada char de b tem 5% de chance de mutar
// e 95% de chance de não mutar
void mutar(char *a, char *b){
    int i;
    for(i = 0; i < 28; i++){
        if(int_rand(MUTAR) > 0){
            b[i] = a[i];
        }else{
            b[i] = letras[int_rand(QTE_LETRAS)];
        }
    }
    // indica o término da string
    b[i] = '\0';
}
```

```c
int main(){
    int i, dif_letras, iteracao = 0, melhor_resp = INT_MAX, melhor
    char copia_string[100][29];

    // crio uma string inicial
    for (i = 0; i < 28; i++){
        copia_string[0][i] = letras[int_rand(QTE_LETRAS)];
    }
    copia_string[0][i] = '\0';

    while(melhor_resp != 0){
        // faço cópias da string inicial e, para cada char,
        // eu dou uma chance de 5% desse char mutar
        for (i = 1; i < COPIAS; i++)
            mutar(copia_string[0], copia_string[i]);

        // índice da melhor resposta
        melhor_id = 0;
        melhor_resp = INT_MAX;
        for (i = 0; i < COPIAS; i++) {
            dif_letras = comparar(alvo, copia_string[i]);  // quan
            // atualizo qual a menor diferença entre as strings at
            if(dif_letras < melhor_resp) {
                melhor_resp = dif_letras;
                melhor_id = i;
            }
        }

        // a primeira cópia recebe sempre a string que mais se par
        // o alvo (ou seja, que tem a menor diferença de chars em
        strcpy(copia_string[0], copia_string[melhor_id]);

        printf("geração %d, pontos %d: %s\n", iteracao, (28 - melh
        iteracao++;
    }

    return 0;
}
```

Aplicação do problema em Python

```python
import random


def main():
    objetivo = "METHINKS IT IS LIKE A WEASEL"
    caracteres = "ABCDEFGHIJKLMNOPQRSTUVWXYZ "
    geracoes = 0
    sequencia_atual = gerar_sequencia_aleatoria(objetivo, caracteres)
    print("Geração:", geracoes)
    print(sequencia_atual)
    while sequencia_atual != objetivo:
        sequencia_atual = pegar_sequencia_mutada(
            sequencia_atual, objetivo, caracteres)
        geracoes += 1
        print("Geração:", geracoes)
        print(sequencia_atual)


def gerar_sequencia_aleatoria(objetivo, caracteres):
    sequencia = ""
    for i in range(len(objetivo)):
        sequencia += caracteres[random.randint(0, len(caracteres)-1)]
    return sequencia


def pegar_sequencia_mutada(sequencia, objetivo, caracteres):
    lista_sequencia = pegar_mutacoes_sequencia(sequencia, caracteres)
    melhor_sequencia = lista_sequencia[0]
    melhor_similaridade = pegar_posicao(melhor_sequencia, objetivo)
    for seq in lista_sequencia:
        similaridade = pegar_posicao(seq, objetivo)
        if similaridade > melhor_similaridade:
            melhor_similaridade = similaridade
            melhor_sequencia = seq
    return melhor_sequencia


def pegar_mutacoes_sequencia(sequencia, caracteres):
    lista_sequencia = []
    for i in range(100):
        lista_sequencia.append(sequencia_mutada(sequencia, caracteres))
    return lista_sequencia


def sequencia_mutada(sequencia, caracteres):
    resultado = ""
    for i in range(len(sequencia)):
        if random.randint(0, 100) <= 5:
            resultado += caracteres[random.randint(0, len(caracteres)-1)]
        else:
            resultado += sequencia[i]
    return resultado


def pegar_posicao(sequencia, objetivo):
    posicao = 0
    for i in range(len(objetivo)):
        if objetivo[i] == sequencia[i]:
            posicao += 1
    return posicao


main()
```

Aplicação do problema em Java

```java
import java.util.Random;

public class Evolucao {
    static final String alvo = "METHINKS IT IS LIKE A WEASEL";
    static final char[] possibilidades = "ABCDEFGHIJKLMNOPQRSTUVWXYZ ".toCharArray();
    static double minimoMutacoes = 0.05;
    static int apitidaoPerfeito = alvo.length();
    private static String mutante;
    static Random rand = new Random();

    public String getAlvo() { return alvo; }

    public String getMutante() { return mutante; }

    public void setMutante(String parent) { Evolucao.mutante = parent; }

    /**
     * Cálculo da diferença de caracteres entre a string alvo e a string mutadora.
     *
     * Quanto menor o valor da taxa (sua porcentagem) de mutação da string mutante,
     * mais proximo estou da string alvo.
     * */
    public int comparacaoStrings(String trial){
        int valorComparacao = 0;
        for(int i = 0;i < trial.length(); i++){
            if (trial.charAt(i) == alvo.charAt(i)) valorComparacao++;
        }
        return valorComparacao;
    }

    /**
     * Calculo da taxa de mutação.
     *
     *Quanto menor for a taxa de mutação mais próxima
     * */
    public double taxaDeMutacoes(){
        return (((double) apitidaoPerfeito - comparacaoStrings(mutante)) / apitidaoPerfeito * (1 - minimoMutacoes));
    }

    /**
     * Realização da mutação da string.
     *
     * Para cada char da string, caso o valor double gerado for menor ou igual a taxa de mutação obtida,
     * a string de retorno recebe um caracter aleatório entre as minhas
     * possibilidades. Caso não ela recebe o caractere naquele indice
     * */
    public String mutacao(String mutante, double taxa){
        String copiadora = "";
        for(int i = 0; i < mutante.length(); i++){
            copiadora += (rand.nextDouble() <= taxa) ?
                    possibilidades[rand.nextInt(possibilidades.length)]:
                    mutante.charAt(i);
        }
        return copiadora;
    }
}
```

```java
public class Main {
    public static void main(String[] args){
        Evolucao evolucao = new Evolucao();

        //Criação de uma string inicial aleatoria (28 caracteres)
        evolucao.setMutante(evolucao.mutacao(evolucao.getAlvo(),  1));
        int iteracao = 0;
        int melhorPontuacao = 0;

        //Enquanto a melhor pontuação não for atingida
        while(melhorPontuacao != evolucao.getAlvo().length()){
            double taxaMutacoes = evolucao.taxaDeMutacoes();
            System.out.println(taxaMutacoes);

            System.out.println(iteracao +": " + evolucao.getMutante() + ", pontos: " +
                    evolucao.comparacaoStrings(evolucao.getMutante()) + ", taxaMutacoes: " + taxaMutacoes);

            String melhorString = null;

            //Fazendo copias de mutações
            for(int i = 0; i < 100; i++){
                String copia = evolucao.mutacao(evolucao.getMutante(), taxaMutacoes);

                //Resultado obtido da comparação entre as strings (quantidade de letras diferentes)
                int pontos = evolucao.comparacaoStrings(copia);

                //Melhor sentença é atualizada com base na pontuação obtida
                if(pontos > melhorPontuacao){
                    melhorString = copia;
                    melhorPontuacao = pontos;
                }
            }

            /*
             caso a melhor pontuação obtida ate o momento seja maior que resultado das comparações,
              a string mutadora recebe a string mais proxima do alvo e, caso não, ela permanece a mesma
              */
            if (melhorPontuacao > evolucao.comparacaoStrings(evolucao.getMutante())){
                evolucao.setMutante(melhorString);
            } else {
                evolucao.setMutante(evolucao.getMutante());
            }
            iteracao++;
        }
        System.out.println(evolucao.getMutante() + ", " + iteracao);
    }
}
```

Comparação dos Códigos

- Python, Java, C

- Nível de dificuldade:

1° C -> feito com matrizes, verificação do final da srt ‘\0’, conhecimento de funções

2° Java -> trabalha direto com strings

3° Python -> não tipada, alto nível

- Legibilidade do código:

1° Python -> simples de modularizar, não precisa de tantas contas matemáticas

2° Java -> trabalha com objetos

3° C -> defines e includes, trabalho com funções, ‘\0’, [29], ponteiros

- Tamanho dos programas:

1° 82 linhas Java

2° 76 linhas C

3° 64 linhas Python