Trabalho de Algoritmos Genéticos - Introdução a Ciência da Computação I

Nome: Vinícius de Moraes
Nome: Raul Ribeiro Teles

Primeiro caso: equação de uma função de 5° grau com raiz inteira em 1:

$$f: y = x^5 + 2x^4 + 3x^3 + 4x^2 + 5x - 15$$

```
Digite o tamanho da população: 5
Digite a taxa de mutação (0 a 100): 25
Digite o máximo de gerações: 250
Digite o valor de A: 1
Digite o valor de B: 2
Digite o valor de C: 3
Digite o valor de D: 4
Digite o valor de E: 5
Digite o valor de F: -15
Resultado encontrado na geracao 16
Melhor Individuo: 1
Resultado: 0
```

Resultado encontrado na décima sexta geração, sendo o indivíduo com valor 1 que possui distância 0 da raiz.

Segundo caso: equação de uma função de 5° grau com raiz real em aproximadamente -1.36:

$$f: y = 5x^5 + 4x^4 + 3x^3 + 2x^2 + x + 15$$

```
Digite o tamanho da população: 5
Digite a taxa de mutação (0 a 100): 25
Digite o máximo de gerações: 250
Digite o valor de A: 5
Digite o valor de B: 4
Digite o valor de C: 3
Digite o valor de D: 2
Digite o valor de E: 1
Digite o valor de F: 15
Melhor Individuo: -1
Resultado: 12
```

Como o algoritmo genético é baseado em números inteiros, a raiz mais próxima que ele conseguiu encontrar foi -1, o que é consideravelmente próximo do resultado real

Funcionamento do algoritmo: o algoritmo é dividido em 7 fases: geração da população inicial, avaliação da população, armazenamento do melhor indivíduo, verificar critério de parada, seleção dos melhores indivíduos, cruzamento e mutação. As 6 últimas fases são feitas dentro de um laço que se repete “número de gerações” vezes ou até o melhor resultado ser igual a distância alvo.

1. **Geração da população inicial:** nessa fase, o algoritmo escolhe uma quantidade de números aleatórios igual ao tamanho da população fornecido pelo usuário. Nós colocamos o limite arbitrário de 40 com o intuito de evitar que o valor ultrapasse o valor máximo de um inteiro na hora de calcular o resultado.

```
int populacaoAtual[tamanhoPopulacao];

for(int i = 0; i < tamanhoPopulacao; i++)
{
    populacaoAtual[i] = (rand() % LIMITE) - LIMITE/2; // Subtraindo limite/2 para incluir negativos
}
```

2. **Avaliação da população:** nessa fase, o algoritmo calcula o módulo do resultado da equação de 5 grau com os valores de a,b,c,d,e,f fornecidos pelo usuário durante a inicialização do programa. Após isso, o algoritmo coloca os valores em ordem crescente.

```
for(int j = 0; j < tamanhoPopulacao; j++)
{
    // Calculando a distância do resultado até 0
    resultado[j] = abs(a*pow(populacaoAtual[j],5)
    + b*pow(populacaoAtual[j],4) + c*pow(populacaoAtual[j],3)
    + d*pow(populacaoAtual[j],2) + e*populacaoAtual[j] + f);
}

// Ordenando o vetor resultado e o vetor populacao
for(int l = 0; l < tamanhoPopulacao; l++)
{
    int temp;
    int tempPop;
    for(int k = 0; k < tamanhoPopulacao-1; k++)
    {
        if(resultado[k] > resultado[k+1])
        {
            temp = resultado[k];
            resultado[k] = resultado[k+1];
            resultado[k+1] = temp;

            // Ordenando para que populacaoAtual[k] corresponda a resultado[k]
            tempPop = populacaoAtual[k];
            populacaoAtual[k] = populacaoAtual[k+1];
            populacaoAtual[k+1] = tempPop;
        }
    }
}
```

3. **Armazenamento do melhor indivíduo:** caso o melhor resultado da geração atual for melhor que todos os outros resultados anteriores, ele é armazenado na variável “melhor indivíduo” junto com o seu valor.

```
if(abs(resultado[0]) < resultadoMelhorInd)
{
    melhorIndividuo = populacaoAtual[0];
    resultadoMelhorInd = resultado[0];
}
```

4. **Verificar o critério de parada:** caso o melhor resultado seja igual a distância alvo (definida como 0 com o intuito de apresentar um valor mais próximo do valor real e potencialmente fazer com que o programa tenha mais gerações), o laço das gerações se interrompe e o resultado final é mostrado na tela.

```
if(abs(resultadoMelhorInd) <= distanciaAlvo)
{
    printf("Resultado encontrado na geracao %d\n", (geracao + 1));
    break;
}
```

5. **Seleção dos melhores indivíduos:** nessa fase, o algoritmo seleciona aleatoriamente 25% da população total, a probabilidade de ser escolhido é inversamente proporcional a sua classificação na fase 2. A equação escolhida foi $x = 99 - \left[\left(\frac{i}{(p-1)}\right) * 98\right]$, onde x é a probabilidade, i é posição do indivíduo no vetor ordenado com intervalo fechado $[0, p]$ e p é o tamanho da população total.

```
for(int i = 0; i < (int)(tamanhoPopulacao*porcentSelecionados);)
{
    moeda = rand()%100;
    if(moeda < (99 - ((float)i/(tamanhoPopulacao-1))*98))
    {
        individuosSelecionados[i] = populacaoAtual[i];
        i++;
    }
}
```

6. **Cruzamento:** nessa fase, o algoritmo seleciona dois indivíduos e realiza uma máscara de bits em ambos, após isso, ele soma o resultado e cria um novo indivíduo que carrega os “genes” daqueles que foram selecionados.

```
for (int i = (int)(tamanhoPopulacao*porcentSelecionados)+1; i<tamanhoPopulacao; i++)
{
    //Criando as mascaras
    int mascaraA = -1431655766; // 10101010101010101010101010101010
    int mascaraB = 1431655765;  // 01010101010101010101010101010101
    int genesA = populacaoAtual[i] & mascaraA;
    int genesB = populacaoAtual[rand()%(tamanhoPopulacao-1)] & mascaraB;
    populacaoAtual[i] = (genesA + genesB) % LIMITE;
}
```

7. **Mutação:** por último, o algoritmo aplica um NOT em cada bit aleatoriamente, cada bit possui a mesma chance de ser escolhido, sendo isso definido pelo usuário durante a inicialização do programa. O intuito da mutação é criar uma maior diversidade para os resultados, dando oportunidade deles encontrarem novas raízes para o problema.

```
for (int i = (tamanhoPopulacao*porcentSelecionados)+1; i<tamanhoPopulacao; i++)
{
    for (int j = 0; j<sizeof(int)*8; j++)
    {
        if (rand() % 100 < taxaMutacao)
        {
            populacaoAtual[i] = populacaoAtual[i] ^ (int)pow(2, j);
            populacaoAtual[i] %= LIMITE;
        }
    }
}
```