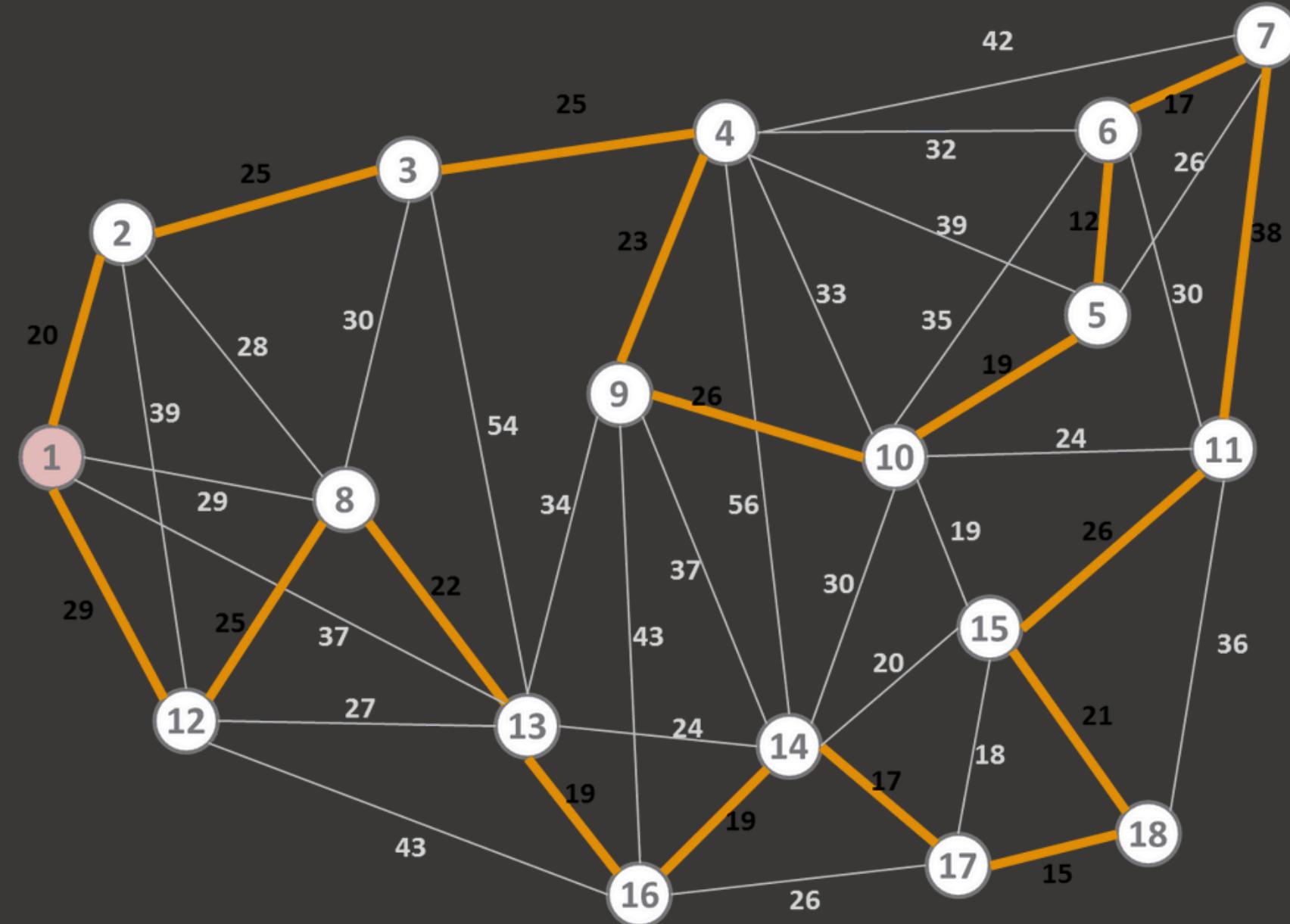


ALGORITMOS GENÉTICOS



PROBLEMA DO CAIXEIRO VIAJANTE (PCV)

PARTICIPANTES



LEONARDO VINÍCIUS



LUCCAS HENRIQUE



MATHEUS GARCIA



RANIER SALES

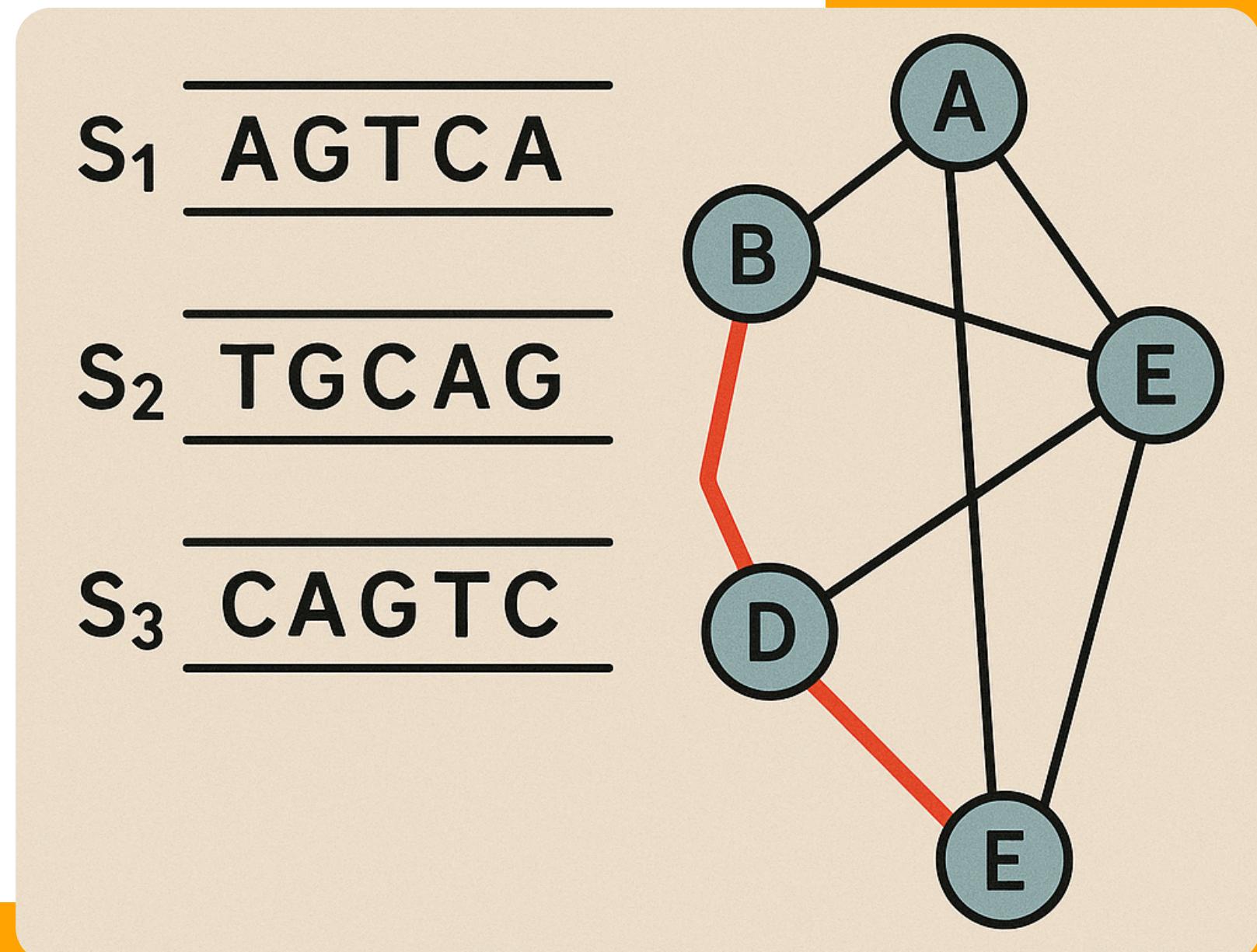


Problema do Caixeiro Viajante

É um problema clássico de otimização combinatória que busca a rota mais curta para visitar um conjunto de cidades, retornando à cidade de origem. É um problema NP-difícil, o que significa que não há um algoritmo eficiente conhecido para encontrar a solução ótima para grandes instâncias.

Resumo:

O problema do caixeiro viajante consiste em encontrar a menor rota possível que visita todas as cidades (ou pontos) e retorna à cidade inicial, também conhecida como cidade de origem



O PCV tem [aplicações](#) em diversas áreas, como logística, roteamento e sequenciamento de DNA.



Algoritmos Genéticos

```
printf("Digite o numero de cidades desejadas.\n");
scanf("%d", &numeroCidades);
getchar();

char nome[numeroCidades][100]; // nomes das cidades
char buffer[100];
int pesos[100][100]; // matriz de pesos das arestas

//Entrada dos nomes das cidades com validação
for (int i = 0; i < numeroCidades; i++) {
    printf("Digite o nome da %d cidade:\n", i + 1);
    fgets(buffer, 100, stdin);
    buffer[strcspn(buffer, "\n")] = '\0';

    int isNumber = 1;

    for(int j = 0; buffer[j] != '\0'; j++) {
        if (!isdigit(buffer[j]))
        {
            isNumber = 0;
            break;
        }
    }
}
```

Um algoritmo genético (AG) é uma técnica de otimização inspirada na evolução biológica. Ele trabalha com populações de soluções possíveis, e melhora essas soluções ao longo do tempo com operadores como seleção, cruzamento (crossover) e mutação.

Como usar ele pra resolver o PCV:

O algoritmo genético para resolver o problema do caixeiro viajante (PCV) funciona simulando a evolução de rotas ao longo de várias gerações, com o objetivo de encontrar a menor rota possível. Ele é inspirado na seleção natural — sobrevivem e se reproduzem as rotas mais "aptas", ou seja, as mais curtas.

FUNCIONAMENTO

1. REPRESENTAÇÃO DE UMA ROTA

Cada rota será representada como um vetor de inteiros, onde cada número é o índice de uma cidade.

2. GERAÇÃO DA POPULAÇÃO INICIAL

A população é um vetor de rotas aleatórias.

3. AVALIAÇÃO (FITNESS FUNCTION)

Calcula-se a distância total da rota. Suponha uma matriz de distâncias:

4. SELEÇÃO (EX: TORNEIO)

Seleciona os melhores indivíduos com menor distância.

5. CRUZAMENTO (CROSSOVER)

O que é: combinação de duas rotas (pais) para gerar uma nova rota (filho), misturando partes das rotas dos pais.

6. MUTAÇÃO

Alteração aleatória em uma rota, geralmente trocando duas cidades de posição.



7. NOVA GERAÇÃO

A criação de uma nova população com base nos pais selecionados, filhos cruzados e rotas mutadas.



8. CRITÉRIO DE PARADA

A condição para encerrar o algoritmo.



Nossa Implementação



```

int isNumber = 1;

for(int j = 0; buffer[j] != '\0'; j++) {
    if (!isdigit(buffer[j]))
    {
        isNumber = 0;
        break;
    }
}

if(isNumber) {
    system("cls");
    printf("Error: O nome da cidade nao pode ser um numero!\n");
    sleep(1.5);
    system("cls");
}

```

Dificuldades Encontradas

- Fazer o algoritmo genético e entender como ele funciona
- Adaptar nosso código pro contexto da genética(lembrando que não é só DNA)
- Rodar o código de forma que ele rodasse partindo de onde o usuário queria
- Manter a organização, já que você só usar as funções padrões do caixeiro viajante(como adicionar aresta, criar NO) não é suficiente e por ultimo, fazer com que não criasse duas arestas entre dois pontos
- Erros.

```

typedef struct Grafo
{
    int numVertices;
    int matrizAdjacencia[100][100];
    No* listaAdjacencia[100];
} Grafo;

// grafo.c
Grafo* criarGrafo(int vertices);
No* criarNo(int destino, int peso);
void adicionarAresta(Grafo* grafo, int origem, int destino, int peso);
void imprimirGrafo(Grafo* grafo, char nomeBairro[][100]);
void liberarGrafo(Grafo* grafo);
void menu(Grafo* grafo, int pesos[100][100], int numeroCidades, char nomeBairro[][100]);

// algoritmo_genetico.c
void algoritmoGenetico(Grafo* grafo, int matrizPesos[100][100], int numCidades, char nomes[][100], int cidadeInicial);
Individuo* gerarPopulacaoInicial(int numCidades, int cidadeInicial);
int comparar(const void* a, const void* b);
int calcularCusto(int* caminho, int numCidades, int matriz[100][100]);
void embaralhar(int* vetor, int tamanho);
void avaliarPopulacao(Individuo* populacao, int numCidades, int matriz[100][100]);
void cruzar(Individuo pai1, Individuo pai2, Individuo* filho, int numCidades);
void mutar(Individuo* individuo, int numCidades);

#endif

```

LEONARDO AGRADA A PRESENÇA DE VOCÊS

