Planejador de Rotas para Viajante

Reidner dos Santos Medeiros
Universidade de Rio Verde
reidner.medeiros@academico.unirv.edu.br
Rio Verde, GO, Brasil.

1 INTRODUÇÃO

Este planejador de rotas permite encontrar a rota mais curta entre dois pontos, no caso duas cidades para usuários que desejam planejar viagens, oferecendo um roteiro de cidades pelas quais o usuário deverá seguir para otimizar seu caminho.
Calcular a rota mais curta entre dois pontos é desafiador ainda mais quando há muitas conexões possíveis, sendo necessário um algoritmo que explore todas as rotas e determine a mais curta com base nas distâncias pré-definidas em sua base de dados.
O projeto visa fornecer uma interface amigável onde o usuário pode selecionar as cidades de origem e destino e receber a melhor rota e a distância total. Combinando o processamento lógico do Prolog com a interface gráfica do Tkinter em Python.

2 TECNOLOGIAS UTILIZADAS

Visual Studio Code(Para escrita do código)
Copilot Designer(Para gerar a imagem do plano de fundo)
ChatGpt(Para consultas gerais)
Linguagem Python Versão 3.12.3(Usada para interface e integração com prolog)
Swi-Prolog Versão 8.4.3(Usada como banco de dados e regras para consulta)
Tkinter(Biblioteca usada para criação da interface gráfica)
Pyswip(Biblioteca utilizada para integrar o Prolog ao Python)

3 IMPLEMENTAÇÃO

1. Interface Gráfica com Tkinter

A interface gráfica é criada usando a biblioteca Tkinter em Python. Ela inclui elementos como labels, comboboxes para seleção de cidades, um botão para encontrar a rota e uma label para exibir o resultado.

Lembrando de utilizar no início do código os seguintes imports:

import tkinter as tk:

tkinter é uma biblioteca para criar janelas e elementos gráficos em Python.

“as tk” faz com que a gente possa usar um nome curto (tk) para acessar os recursos dessa biblioteca.
Exemplo: tk.Tk() cria uma nova janela; tk.Label() cria um rótulo de texto.

from tkinter import messagebox:

Importa messagebox, que é usado para mostrar mensagens pop-up para o usuário, como avisos e erros.
Exemplo: messagebox.showerror("Erro", "Algo deu errado") mostra uma mensagem de erro.

from tkinter import ttk:

Importa ttk, que oferece uma versão mais moderna e estilizada dos elementos gráficos do tkinter.
Exemplo: ttk.Combobox() cria uma lista suspensa estilizada.

```python
# Criação da interface gráfica com Tkinter
root = tk.Tk()
root.title("Planejador de Rotas")

# Ajuste do tamanho da janela principal
root.geometry("600x400")
root.resizable(False, False) # Serve para não deixar o tamanho da janela ser alterado

# Carregar imagem de fundo
background_image = tk.PhotoImage(file="background_image.png")

# Adicionar imagem de fundo
background_label = tk.Label(root, image=background_image)
background_label.place(relwidth=1, relheight=1)

# Lista de cidades
cidades = ["goiania", "anapolis", "rio_verde", "brasilia", "luziania", "trindade", "jatai", "planaltina", "goianesia"]

# Obter a altura da janela
window_height = root.winfo_height()

# Calcular o deslocamento vertical para centralizar os widgets
vertical_offset = (window_height - 180) // 2  # Ajuste conforme necessário

# Tamanho da fonte
font_size = 11

# Labels e Comboboxes centralizados
tk.Label(root, text="Cidade de Origem:", bg="white", font=("Courier New", font_size)).place(relx=0.25, rely=0.3 - vertical_offset/400)
origem_var = tk.StringVar(root)
origem_combobox = ttk.Combobox(root, textvariable=origem_var, values=cidades, font=("Courier New", font_size))
origem_combobox.place(relx=0.55, rely=0.3 - vertical_offset/400)

tk.Label(root, text="Cidade de Destino:", bg="white", font=("Courier New", font_size)).place(relx=0.25, rely=0.4 - vertical_offset/400)
destino_var = tk.StringVar(root)
destino_combobox = ttk.Combobox(root, textvariable=destino_var, values=cidades, font=("Courier New", font_size))
destino_combobox.place(relx=0.55, rely=0.4 - vertical_offset/400)

# Botão "Encontrar rota" centralizado
tk.Button(root, text="Encontrar Rota", command=on_click, font=("Courier New", font_size)).place(relx=0.35, rely=0.5 - vertical_offset/400, relwidth=0.3)

# Label "Planejador de Rotas" centralizada no topo
tk.Label(root, text="Planejador de Rotas", bg="#e74c3c", fg="black", font=("Courier New", 14)).place(relx=0.5, rely=0.05, anchor="center")

resultado_var = tk.StringVar()
tk.Label(root, textvariable=resultado_var, bg="white").place(relx=0.025, rely=0.6 - vertical_offset/400, relwidth=0.95)

root.mainloop()
```

2. Integração com Prolog usando pyswip
A biblioteca pyswip é usada para integrar Python com Prolog. O código consulta o arquivo Prolog para encontrar a melhor rota entre as cidades selecionadas.

Lembrando de utilizar no início do código o seguinte import:

import pyswip:

Permite integrar o Python com o Prolog. Isso significa que você pode usar funcionalidades do Prolog, como consultas a bancos de dados Prolog, diretamente em seu código Python, assim garantindo o funcionamento correto deste trecho de código.

```python
# Função para chamar o Prolog e obter a melhor rota
def encontrar_rota(origem, destino):
    prolog = pyswip.Prolog()
    prolog.consult('rotas.pl') # Carrega o arquivo Prolog com as definições de rotas
    consulta = f"melhor_rota({origem}, {destino}, Rota, Distancia)"

    try:
        resultado = list(prolog.query(consulta))
        if resultado:
            rota = resultado[0]['Rota']
            distancia = resultado[0]['Distancia']
            return rota, distancia
        else:
            return None, None
    except Exception as e:
        messagebox.showerror("Erro", str(e))
        return None, None

```

3. Função de Callback do Botão

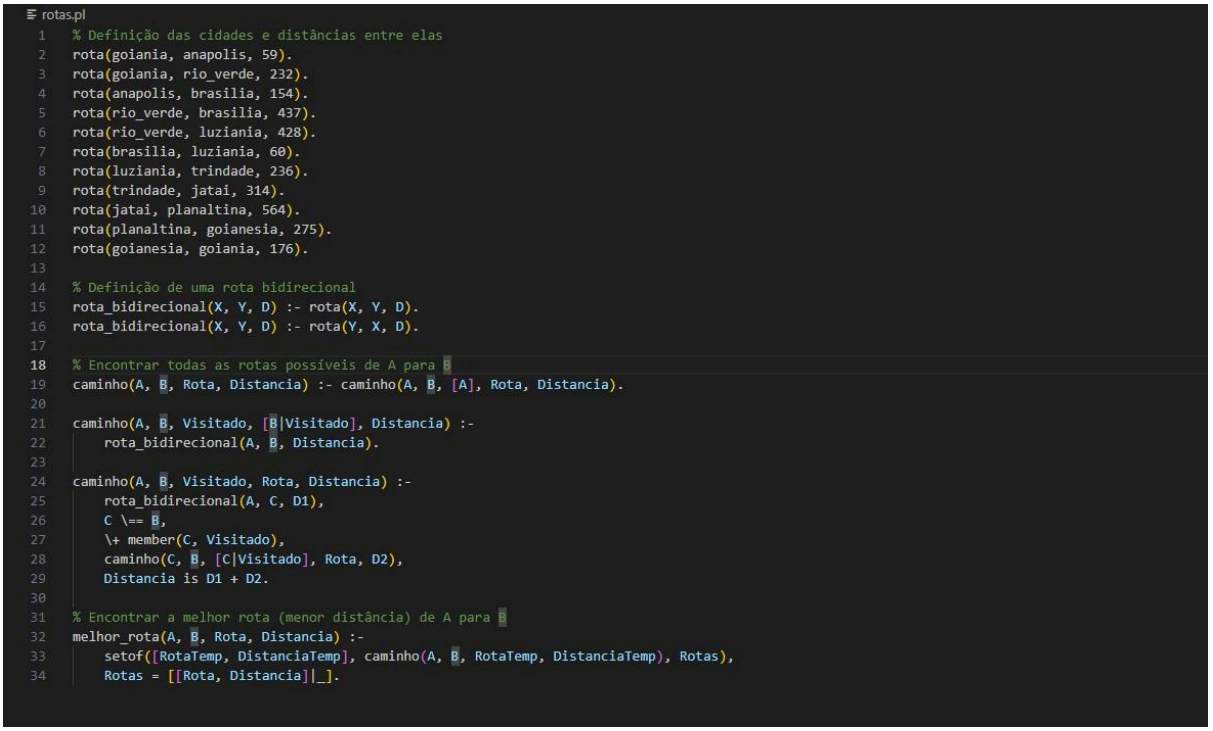
A função on_click é chamada quando o usuário clica no botão "Encontrar Rota". Ela obtém as cidades selecionadas, consulta o Prolog e exibe a rota e a distância.

```python
# Função de callback do botão
def on_click():
    origem = origem_var.get()
    destino = destino_var.get()

    if not origem or not destino:
        messagebox.showwarning("Aviso", "Por favor, selecione tanto a cidade de origem quanto a de destino.")
        return

    rota, distancia = encontrar_rota(origem, destino)

    if rota:
      rota_invertida = rota[::-1]
      resultado_var.set(f"Rota: {' -> '.join(map(str, rota_invertida))}\nDistância: {distancia} km")
    else:
      resultado_var.set("Nenhuma rota encontrada entre as cidades especificadas.")

```

4. Lógica Prolog para Cálculo das Rotas (rotas.pl)

O arquivo rotas.pl define as cidades e as distâncias entre elas, e implementa a lógica para encontrar todas as rotas possíveis e selecionar a melhor (menor distância).

O arquivo “rotas.pl” deve estar salvo na mesma pasta do programa em Python possibilitando sua interação.



rotas.pl

```
% Definição das cidades e distâncias entre elas
rota(goiania, anapolis, 59).
rota(goiania, rio_verde, 232).
rota(anapolis, brasilia, 154).
rota(rio_verde, brasilia, 437).
rota(rio_verde, luziania, 428).
rota(brasilia, luziania, 60).
rota(luziania, trindade, 236).
rota(trindade, jatai, 314).
rota(jatai, planaltina, 564).
rota(planaltina, goianesia, 275).
rota(goianesia, goiania, 176).

% Definição de uma rota bidirecional
rota_bidirecional(X, Y, D) :- rota(X, Y, D).
rota_bidirecional(X, Y, D) :- rota(Y, X, D).

% Encontrar todas as rotas possíveis de A para B
caminho(A, B, Rota, Distancia) :- caminho(A, B, [A], Rota, Distancia).

caminho(A, B, Visitado, [B|Visitado], Distancia) :-
    rota_bidirecional(A, B, Distancia).

caminho(A, B, Visitado, Rota, Distancia) :-
    rota_bidirecional(A, C, D1),
    C \== B,
    \+ member(C, Visitado),
    caminho(C, B, [C|Visitado], Rota, D2),
    Distancia is D1 + D2.

% Encontrar a melhor rota (menor distância) de A para B
melhor_rota(A, B, Rota, Distancia) :-
    setof([RotaTemp, DistanciaTemp], caminho(A, B, RotaTemp, DistanciaTemp), Rotas),
    Rotas = [[Rota, Distancia]|_].
```

4 DESAFIOS

1-Integração entre Python e Prolog: Para integrar estas duas linguagens só foi possível utilizando o Swi-Prolog Versão 8.4.3, as versões mais recentes do Swi-Prolog Versão resultaram em diversos erros durante a integração.

2-Nome das cidades: Cidades com nome composto não eram passadas para o código prolog durante a consulta a forma encontrada para resolver foi o uso do underline (_) entre os espaços.

3-Tratamento de erros do usuário: Para evitar buscas por cidades que não estão disponíveis para consulta foi adicionado uma lista no campo onde é preenchido as cidades “Origem” e “Destino”:

4-Escalabilidade:Pensando em possíveis adições de cidades no futuro o armazenamento de dados permite a adição de novos fatos/cidades para consulta desde que ela seja representada da seguinte forma “rota((cidade adicionada), (cidade já existente), distância)”.

5 REFERÊNCIAS

Python Tkinter Documentation:

https://docs.python.org/3/library/tkinter.htmlDocumentação
oficial fornecendo informações detalhadas sobre como usar a biblioteca Tkinter para criar interfaces gráficas em Python.
pyswip Documentation:

https://pypi.org/project/pyswip/
Documentação oficial do pyswip, fornecendo orientações sobre como integrar o Python com o Prolog usando a biblioteca pyswip.

Tutoriais de Programação
You tube:
https://www.youtube.com/
Videos sobre Python e Tkinter, também vídeos sobre SWI-Prolog.
Tutoriais sobre como usar o Python e o Tkinter e fazer sua integração com SWI-Prolog.
ChatGpt:
https://chatgpt.com/
Explicações e consultas para melhorias do código.

Créditos de Imagens
Background Image:
Imagem de fundo usada na interface gráfica: Copilot Designer.
Créditos para a imagem de fundo utilizada na interface gráfica, gerada no site https://copilot.microsoft.com/images/create

SWI-Prolog
SWI-Prolog Documentation:
https://www.swi-prolog.org/pldoc/index.html
Documentação oficial do SWI-Prolog, fornecendo informações sobre a linguagem Prolog e a implementação SWI-Prolog.