UFRPE

Projeto Interdidciplinar para Sistemas de Informação 2

Fly Food

1 VA

INTEGRANTES

FELIPE CAVALCANTE

GUSTAVO SANTOS

PEDRO AILTON

Sumário

- Introdução ao problema e solução
- Fundamentos e Conceitos
- Metodologia
- Implementação
- Testes e Resultados
- Melhorias Futuras
- Conclusão

O Problema

- O ano é 2030;
- **Entregas estão impossibilitadas para entregadores terrestres;**
- **Drones da empresa Flyfood precisam otimizar o uso de sua bateria**;
- **Drones precisam otimizar o tempo de entrega**.

Solução Proposta

- **Programa em Python** que **calcula o melhor percurso (de menor custo) automaticamente** a partir de uma entrada do mapa da região de entrega em forma de matriz no formato de arquivo .txt.

Objetivos

- Implementar algoritmo completo de:
  - Leitura de uma matriz de entrada em .txt.
  - Cálculo da rota mais eficiente dentre todas possíveis.
- Analisar resultados e desempenho.
- Aplicar fundamentos matemáticos e computacionais.

Fundamentos do projeto

- Abordagem de força bruta e Complexidade algorítmica;
- Métrica de distância Manhattan;
- Teste de desempenho algorítmico;
- Manipulação de diferentes estruturas de dados.

Tecnologias, bibliotecas e arquitetura

- Python 3
- Bibliotecas: itertools, e time
- Arquitetura modular
  - parser (analisador)
  - otimizador
  - main

Algoritmo

01

Modo rápido (fast)

- Calcula todas as rotas e armazena apenas a melhor
- Imprime apenas a melhor rota

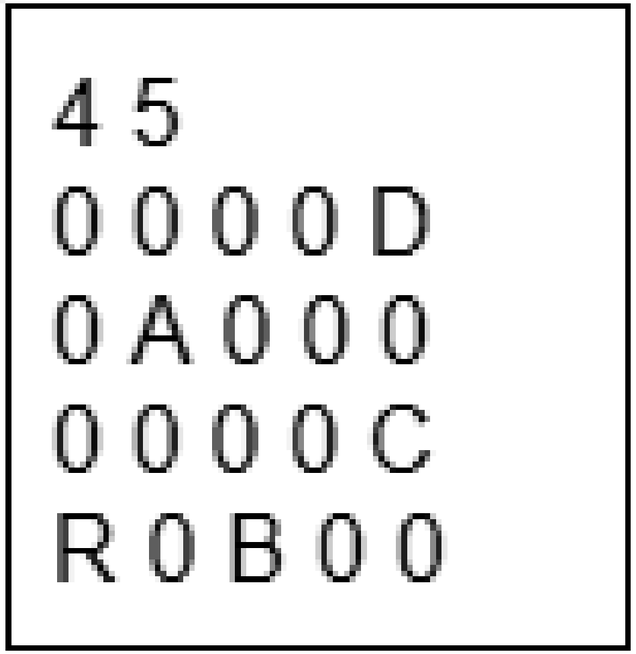
02

Modo detalhado (plus)

- Armazena todos os caminhos e seus custos;
- Imprime todas as rotas, destacando a melhor, o custo médio e a pior delas.

Explicando o Algoritmo por meio de um Exemplo

Entrada exemplo:



entrada.txt

Algoritmo

```python
def parseArquivo(caminho_arquivo):
    linhas = lerArquivo(caminho_arquivo)
    num_linhas, num_colunas = map(int, linhas[0].split())
    matriz = [linha.split() for linha in linhas[1:]]
    pontos = {}
    for i in range(num_linhas):
        for j in range(num_colunas):
            valor = matriz[i][j]
            if valor != "0":
                pontos[valor.upper()] = (i, j)
```

Interpretação da matriz de entrada com a função parseArquivo().

```
Analisando o arquivo e mapeando os pontos...

Informações do arquivo:
  Número de linhas: 4
  Número de colunas: 5

Matriz:
  0 0 0 0 D
  0 A 0 0 0
  0 0 0 0 C
  R 0 B 0 0

Pontos mapeados:
  - D: (0, 4)
  - A: (1, 1)
  - C: (2, 4)
  - R: (3, 0)
  - B: (3, 2)
```

Primeiro trecho da saída no terminal

Algoritmo

```
pontos_de_entrega = extrairPontos(pontos)

# Se não houver pontos de entrega, retorna uma mensagem
if not pontos_de_entrega:
    return "Nenhum ponto de entrega foi especificado."

resultados= [] # lista com (rota,custo) de todas as permutações
total = 0 # variável que vai guardar o total das distâncias para cálcul

# Calcula o custo de cada rota possível
for rota_atual in permutations(pontos_de_entrega):
    custo_da_rota_atual = calcularCustoTotalDaRota(rota_atual, pontos)
    resultados.append((rota_atual,custo_da_rota_atual))
    total += custo_da_rota_atual
# Ordena da menor para a maior distância (custo)
resultados.sort(key=lambda x: x[1])
media = total/len(resultados) # Cálculo da média dos resultados
```

trecho de otimizarRotaPlus()

Algoritmo

```python
pontos_de_entrega = extrairPontos(pontos)

# Se não houver pontos de entrega, retorna uma mensagem
if not pontos_de_entrega:
    return "Nenhum ponto de entrega foi especificado."

melhor_rota = None
melhor_custo = float('inf')
contador = 0

# Calcula o custo de cada rota possível
for rota_atual in permutations(pontos_de_entrega):
    # Calcula o custo da rota que está sendo verificada
    custo_da_rota_atual = calcularCustoTotalDaRota(rota_atual, pontos)

    if custo_da_rota_atual < melhor_custo:
        melhor_custo = custo_da_rota_atual
        melhor_rota = rota_atual
        contador += 1
        if mostrar_atualizacoes:
            print(f"{contador}ª atualização: {' -> '.join(rota_atual)} | Custo:
            {custo_da_rota_atual}")
```

trecho de otimizarRota()

Algoritmo

```python
def calcularCustoTotalDaRota(rota, pontos):

    # Ponto de origem e retorno
    ponto_r = pontos['R']
    custo_total = 0

    # 1. Calcula o custo da origem 'R' até o primeiro ponto da rota
    primeiro_ponto = pontos[rota[0]]
    custo_total += calcularDistancia(ponto_r, primeiro_ponto)

    # 2. Calcula o custo entre os pontos intermediários da rota
    # O loop vai do primeiro ponto até o penúltimo
    for i in range(len(rota) - 1):
        ponto_atual = pontos[rota[i]]
        proximo_ponto = pontos[rota[i+1]]
        custo_total += calcularDistancia(ponto_atual, proximo_ponto)

    # 3. Calcula o custo do último ponto da rota de volta para a origem 'R'
    ultimo_ponto = pontos[rota[-1]]
    custo_total += calcularDistancia(ultimo_ponto, ponto_r)

    return custo_total
```

função calcularCustoTotalDaRota()

Algoritmo

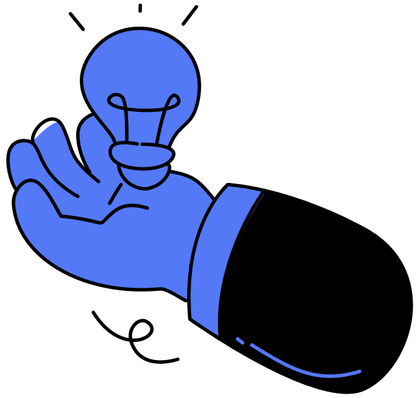
```
19. D -> A -> B -> C | Custo total: 22.
20. D -> B -> C -> A | Custo total: 22.
21. A -> C -> B -> D | Custo total: 22.
22. C -> B -> A -> D | Custo total: 22.
23. D -> B -> A -> C | Custo total: 24.
24. C -> A -> B -> D | Custo total: 24.
Custo médio: 19.0 dronômetros


Cálculo finalizado. Apresentando resultados...

------------------------------------
Melhor rota encontrada: A -> D -> C -> B
Custo total: 14 dronômetros

Pior rota encontrada: C -> A -> B -> D
Custo total: 24 dronômetros

Tempo de execução do algoritmo: 0.0069 segundos
------------------------------------
```

Saída final do terminal no modo plus

```
Iniciando o cálculo da rota ótima...

Buscando a melhor rota (modo rápido)...
1ª atualização: D -> A -> C -> B | Custo: 20
2ª atualização: D -> C -> A -> B | Custo: 18
3ª atualização: A -> D -> C -> B | Custo: 14

Cálculo finalizado. Apresentando resultados...

------------------------------------
Melhor rota encontrada: A -> D -> C -> B
Custo total: 14 dronômetros

Tempo de execução do ALGORITMO: 0.0015 segundos
------------------------------------
```

Saída final do terminal no modo fast

Método de teste



Passo 1

Testes de leitura e tratamento de dados, com suporte a diferentes formatos de arquivo (.txt e .csv).

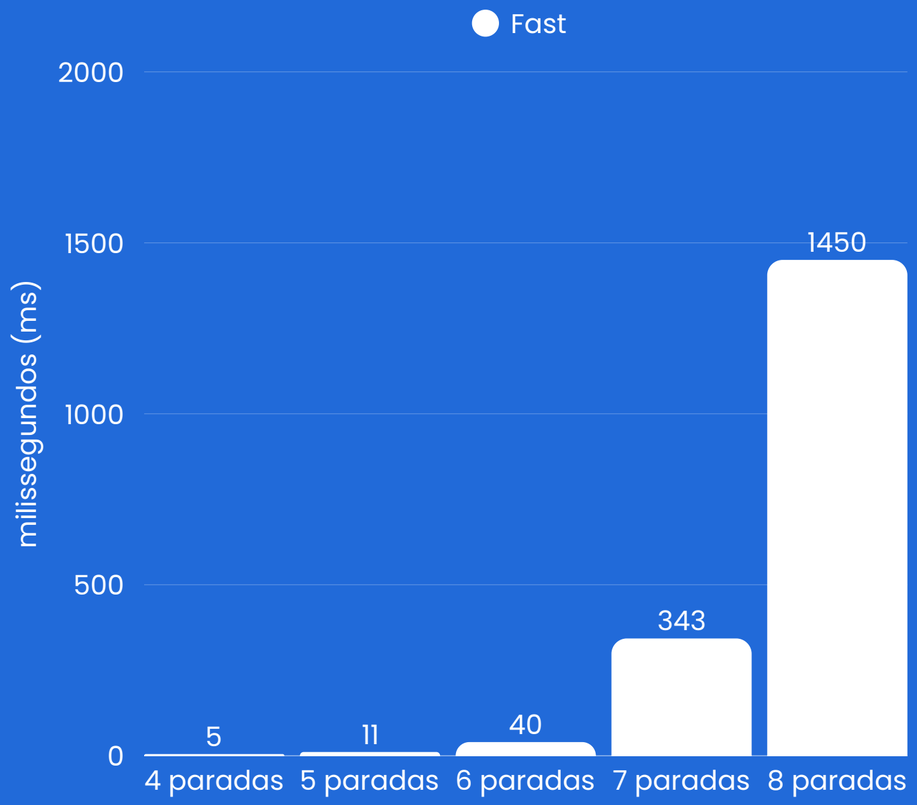
Passo 2

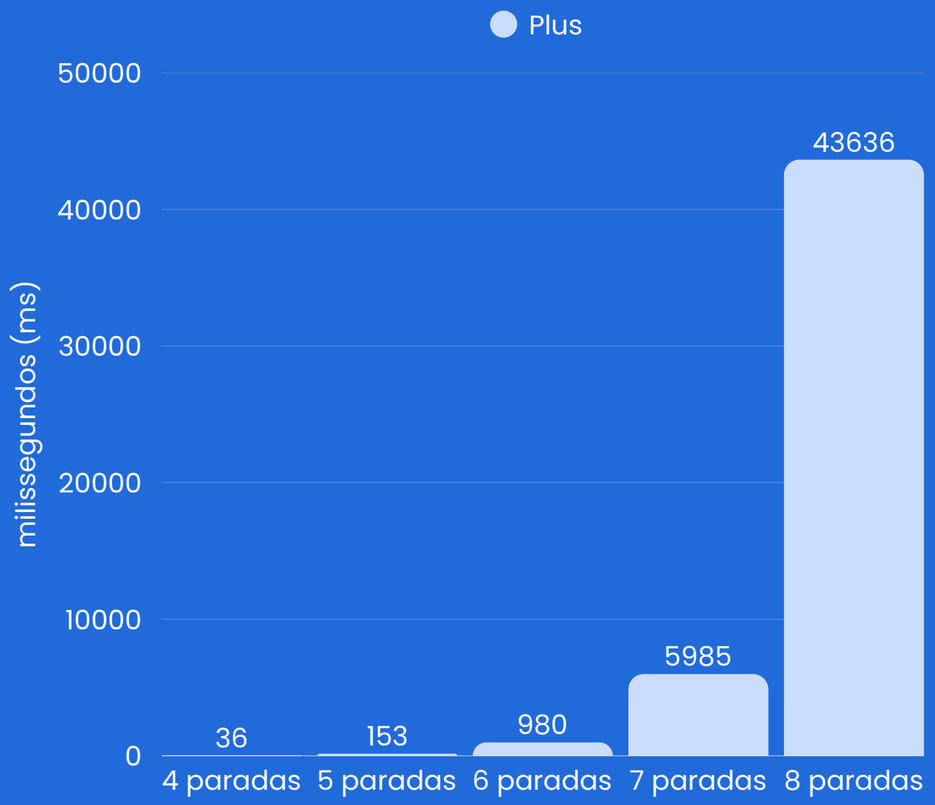
Execução completa do algoritmo, medindo o tempo de execução como métrica de desempenho.

Passo 3

Análise dos resultados e ajustes entre as versões Fast e Plus do algoritmo.

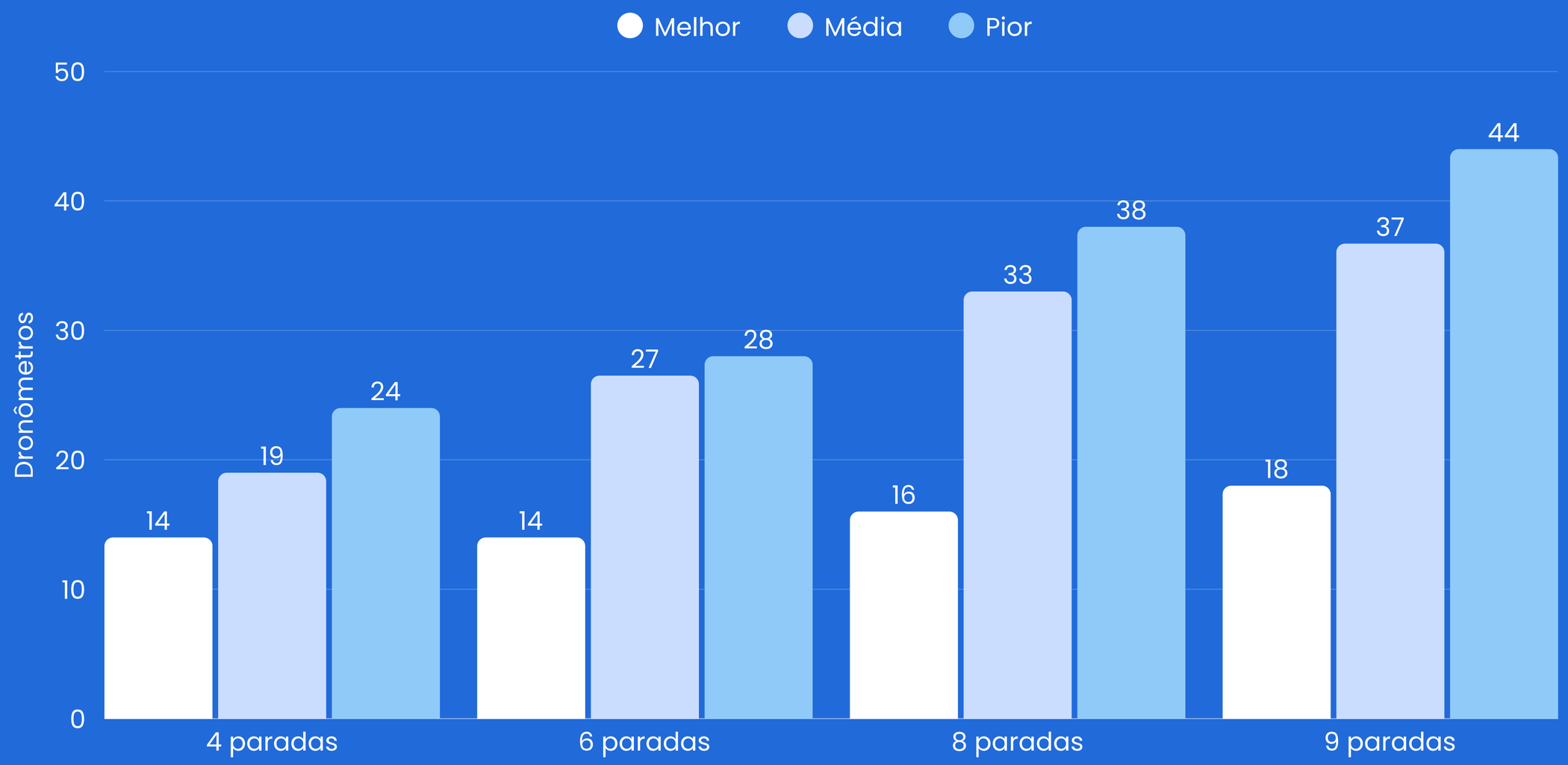
Análise dos Dados de Desempenho





- **Testes realizados com diferentes matrizes (de 4 a 8 pontos).**
- **Desempenho satisfatório até cerca de 6–7 pontos.**
- **Crescimento exponencial do tempo de execução a partir desse ponto.**

Custos das Rotas – Melhor, Média e Pior



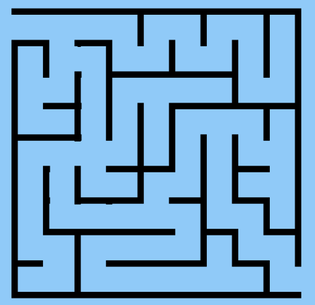
- *Custo total das rotas em dronômetros, calculado pela distância Manhattan.*

Limitações e Descobertas

Tempo de Execução

- O tempo cresce de forma fatorial com o número de pontos (O(n!)).
- Aumentos pequenos no número de entregas geram saltos enormes no tempo.

VS



Complexidade das Rotas

- Cada novo ponto cria múltiplas novas combinações.
- O algoritmo precisa testar todas as permutações possíveis (força bruta).

- **Mapa simplificado, sem obstáculos reais.**
- **Otimização baseada apenas em distância total (Manhattan).**
- **Base para futuras melhorias com algoritmos heurísticos.**

Melhorias Futuras

- Implementar algoritmos mais eficientes para grande número de pontos (ex: Dijkstra, A*).
- Adicionar interface gráfica para uso mais intuitivo.
- Converter o sistema em uma aplicação web (HTML, CSS e JavaScript).
- Salvar automaticamente os resultados em arquivos .csv.
- Suporte a novos formatos de entrada (.json, .xlsx etc.).

OBRIGADO!

Nosso Repositório no Github:

https://github.com/pedroailton/flyfood-pisi-2

ou por esse QR-Code