# Projeto Final : Grafos do Recife + Comparação de Algoritmos

**Python 3.11+**. Proibido usar libs que já implementem os algoritmos (ex.: `networkx`, `igraph`, `graph-tool`) para **BFS/DFS/Dijkstra/Bellman-Ford**. Pode usar `pandas` (IO), `argparse`, `heapq`, `dataclasses`, `typing`, `matplotlib/plotly/pyvis/streamlit` **apenas** para visualização/UX.

## Entrega (GitHub obrigatório)

- Subam todo o projeto em um **repositório GitHub** e entreguem **apenas o link** no Classroom.
- Incluam: código, datasets, **README** com instruções e **PDF** (manual + técnica).

## Estrutura de pastas (obrigatória)

```
projeto-grafos/
├─ README.md
├─ requirements.txt  (ou pyproject.toml)
├─ data/
│   ├─ bairros_recife.csv             # (arquivo enviado)
│   ├─ adjacencias_bairros.csv        # vocês constroem (ver Parte 1)
│   ├─ enderecos.csv                  # vocês constroem (Parte 1)
│   └─ dataset_parte2/                # dataset maior (Parte 2)
├─ out/                                # saídas (.json/.html/.png)
│   └─ .gitkeep
├─ src/
│   ├─ cli.py
│   ├─ solve.py
│   ├─ graphs/
│   │  ├─ io.py           # carregar/validar e "derreter" o CSV
fornecido
│   │  ├─ graph.py        # estrutura: lista de adjacência
│   │  └─ algorithms.py   # BFS, DFS, Dijkstra, Bellman-Ford
(implementação própria)
│   └─ viz.py             # (bônus) visualizações/UX
└─ tests/                 # (obrigatórios, mínimos)
   ├─ test_bfs.py
   ├─ test_dfs.py
   ├─ test_dijkstra.py
   └─ test_bellman_ford.py
```

# PARTE 1

# Grafo dos Bairros do Recife

## O que há no CSV

O arquivo tem **colunas com rótulos "1.1" a "6.3"** contendo nomes de bairros agrupados por **microrregiões** (grupos de bairros). Seu primeiro passo é "**derreter**" (unpivot/melt) todas as colunas para obter uma **lista única de bairros** (nós do grafo) e um **mapeamento bairro → microrregião**.

Resultado esperado deste passo:

- `bairros_unique.csv`: `bairro, microrregiao`
- **Lista de nós** normalizada (sem duplicatas, com acentuação padronizada).

## 1) NÓS: bairros do Recife

- Crie um **grafo rotulado** onde **cada nó é um bairro**.
- O **rótulo** do nó é o nome do bairro (ex.: "Boa Viagem", "Nova Descoberta", …).
- **Observação**: "Setúbal" **não** aparece como bairro no CSV (é sub-bairro de Boa Viagem). Para todas as tarefas que citam **Setúbal**, **tratem como "Boa Viagem (Setúbal)"** e considerem o nó **Boa Viagem**.

## 2) ARESTAS (interconexões): <u>vocês constroem</u>

O CSV **não** traz as conexões explícitas. Cada grupo deve criar **um arquivo** `data/adjacencias_bairros.csv` com **as arestas entre bairros**, baseado em **interconexões reais por logradouros/limites**.

**Formato obrigatório de `adjacencias_bairros.csv`:**

```
bairro_origem,bairro_destino,logradouro,observacao,peso
Boa Viagem,Ipsep,Av. Boa Viagem,"acesso via viaduto X",1.0
Boa Viagem,Imbiribeira,Av. Domingos Ferreira,,1.0
Boa Viagem,Pina,Ponte Y,,1.0
Boa Viagem,Setúbal,Rua Baltazar Passos,"sub-bairro de BV",1.0
...
```

- Grafo **não-direcionado** (salvem apenas uma linha por par; o sistema espelha).
- **`logradouro`** e **`observacao`** são livres, mas **valorizam** a entrega.
- **`peso`**: ver Seção 5 (definam e **documentem** sua régua de pesos).

## 3) Métricas globais e por grupo

Calculem (sempre com base no **grafo dos bairros**):

**Definições:**

- **Ordem** = |V| (número de nós/bairros)
- **Tamanho** = |E| (número de arestas/interconexões)
- **Densidade** (grafo **não-direcionado**):

$$densidade = \frac{2 \mid E \mid}{\mid V \mid (\mid V \mid -1)}$$

  Se |V| < 2, densidade = 0.

**Peçam e entreguem:**

1. **Cidade do Recife (grafo completo):** ordem, tamanho, densidade.
2. **Microrregiões (subgrafos induzidos):** para **cada microrregião** (ex.: colunas 1.*, 2.*, …, 6.*), calculem ordem, tamanho e densidade **no subgrafo** apenas com seus bairros e as arestas entre eles.
3. **Ego-subrede por bairro ("ruas dos bairros"):** para cada bairro `v`, considerem a **ego-network** $v \cup N(v)$ e calculem **ordem/tamanho/densidade**. Isso aproxima a ideia de "ruas do bairro" conectando aos vizinhos. Entreguem uma **tabela** com: `bairro, grau, ordem_ego, tamanho_ego, densidade_ego`.

Arquivos de saída (obrigatórios):

- `out/recife_global.json` (ordem, tamanho, densidade)
- `out/microrregioes.json` (lista com métricas por microrregião)
- `out/ego_bairro.csv` (tabela completa por bairro)

## 4) Graus e rankings

- **Lista de graus**: `out/graus.csv` → `bairro, grau` (grau = nº de interconexões).
- **Bairro mais denso**: o **maior** **`densidade_ego`** na tabela da Seção 3.3.
- **Bairro com maior grau**: `argmax grau`.

## 5) Pesos das arestas (definição criativa e consistente)

Para calcular **distância** (Seção 6) com **Dijkstra**, definam **pesos** para as arestas do grafo. Exemplos (escolham, combinem e **documentem** no PDF):

- **Peso=1** para toda interconexão (distância topológica).
- **Categoria de via** *(menor é melhor)*:

    - avenida: 1.0
    - coletora: 1.5
    - local: 2.0
- **Penalidades**: travessia de ponte/túnel (+0.5), semáforos (+0.2 por conjunto), horário de pico (+α).
- **Fórmula composta**: `peso = α·categoria + β·penalidades`, com $\alpha,\beta > 0$.

Gravem esses pesos em `adjacencias_bairros.csv` (coluna `peso`). **Não usem pesos negativos** aqui; BF fica para a Parte 2.

## 6) Distância entre endereços X e Y

1. Criem `data/enderecos.csv` com **pelo menos 5 pares** de endereços reais do Recife (X,Y) e a classificação manual do **bairro** correspondente (`bairro_X`, `bairro_Y`).
2. Para cada par (X,Y), calculem **custo** e **percurso** no grafo de bairros usando **Dijkstra** (pesos da Seção 5).
3. Gerem:
    - `out/distancias_enderecos.csv`: `X,Y,bairro_X,bairro_Y,custo,caminho`
    - Para **um par obrigatório**: **"Nova Descoberta → Setúbal"** (*usem o nó* **Boa Viagem (Setúbal)**) e salvem também `out/percurso_nova_descoberta_setubal.json` com o caminho.
    -

## 7) Transforme o percurso em árvore e mostre

- A partir do caminho "**Nova Descoberta → Boa Viagem (Setúbal)**", construam a **árvore de caminho** (um subgrafo com as arestas do percurso) e **exportem** uma visualização:
    - `out/arvore_percurso.html` (**interativa**, ex.: `pyvis`/`plotly`) **ou**
    - `out/arvore_percurso.png` (**estática**, ex.: `matplotlib`).

**Requisito**: destacar o caminho (cor, espessura) e mostrar **rótulos dos bairros**.

## 8) Explorações e visualizações analíticas

Usem os **conceitos de aula** para criar **no mínimo 3** visualizações/insights adicionais (salvem em `out/`), por exemplo:

- **Mapa de cores** por **grau** do bairro (mais conexões = cor mais intensa).
- **Ranking** de densidade de ego-subrede por microrregião (barra).
- **Subgrafo** dos 10 bairros com maior grau (graph view).
- **Distribuição** dos graus (histograma).
- **Árvore BFS** a partir de um polo (ex.: "Boa Vista") para visualizar camadas (níveis).

**Entreguem as imagens/HTML + uma nota analítica curta justificando o que cada visualização revela.**

### 9) Apresentação interativa do grafo

- Entreguem um **HTML interativo** (ex.: `pyvis`) com:
  - **Tooltip** por bairro (grau, microrregião, densidade_ego),
  - Caixa de busca por bairro,
  - Botão/legenda para **realçar** o caminho "Nova Descoberta → Boa Viagem (Setúbal)".

Arquivo: `out/grafo_interativo.html`.

# PARTE 2

# Dataset Maior e Comparação de Algoritmos

Escolham um **dataset maior** de grafos (rede de transporte, coautoria, dependências, etc.). Construam o grafo **sem** libs de algoritmos prontos e comparem **BFS, DFS, Dijkstra e Bellman–Ford** em **corretude e desempenho**. Dataset Parte 2 preferencial: **até ~200k arestas**. Se maior, justifiquem e mostrem amostragens/estratégias

**Obrigatório:**

1. Descrever o dataset (|V|, |E|, tipo: dirigido/ponderado, distribuição de graus).
2. Implementar e **rodar**:
   - **BFS/DFS** a partir de ≥ 3 fontes distintas (relatar ordem/camadas/ciclos).
   - **Dijkstra** com **pesos ≥ 0** (≥ 5 pares origem-destino).
   - **Bellman–Ford** com ao menos **um caso** com **peso negativo** (e **sem** ciclo negativo) e **um** com **ciclo negativo** (detectado).
3. **Métricas de desempenho**: tempo e (opcional) memória por algoritmo/tarefa (tabela `out/parte2_report.json`).
4. Visualização: pelo menos **uma** (ex.: amostra do grafo, distribuição de graus, heatmap de distâncias).
5. Discussão crítica: **quando/por que** cada algoritmo é mais adequado; limites do seu design de pesos.

## Como executar

Exemplos (vocês implementam a orquestração em `src/cli.py`/`src/solve.py`):

```
# PARTE 1 (grafo dos bairros)
python -m src.cli --dataset ./data/bairros_recife.csv --alg BFS --
source "Boa Viagem" --out ./out/
python -m src.cli --dataset ./data/bairros_recife.csv --alg DIJKSTRA -
-source "Nova Descoberta" --target "Boa Viagem" --out ./out/
python -m src.cli --dataset ./data/bairros_recife.csv --interactive --
out ./out/
```

```
# PARTE 2 (dataset maior)
python -m src.cli --dataset ./data/dataset_parte2/ --alg DIJKSTRA --
source A --target Z --out ./out/
```

# Saídas obrigatórias (resumo)

- `out/recife_global.json`, `out/microrregioes.json`, `out/ego_bairro.csv`, `out/graus.csv`
- `out/distancias_enderecos.csv`, `out/percurso_nova_descoberta_setubal.json`, `out/arvore_percurso.html|png`
- `out/grafo_interativo.html`
- `out/parte2_report.json` (tabela com tempos/pares/algoritmos)
- Visualizações adicionais em `out/` (+ notas analíticas no PDF)

# Testes mínimos (pytest)

- **BFS**: níveis corretos em grafo pequeno.
- **DFS**: detecção de ciclo e classificação de arestas.
- **Dijkstra**: caminhos corretos com pesos ≥ 0; **recusar** dado com peso negativo.
- **Bellman–Ford**: (i) com pesos negativos **sem** ciclo negativo → distâncias corretas; (ii) com **ciclo negativo** → flag.

# O que vai ser avaliado (10,0 pts + bônus)

1. **Parte 1 : Dados do Recife** (qualidade técnica e completude) : **3,0 PONTOS**
   - Nós/arestas, métricas (global, microrregiões, ego), graus e rankings.
   - Distâncias (endereços), percurso Nova Descoberta → Boa Viagem (Setúbal), árvore do percurso.
   - Visualizações analíticas + grafo interativo.
2. **Parte 2 : Dataset maior e comparação** : **3,0 PONTOS**
   - Execução correta dos 4 algoritmos; casos cobrindo pesos negativos e ciclo negativo (BF).
   - Métricas de desempenho + discussão crítica.
3. **Apresentação:** Participação nas reuniões de acompanhamento, apresentação e comprometimento com o projeto : **2,0 PONTOS**
4. **Qualidade do código, organização, testes, README e PDF - 2,0 PONTOS**

**Bônus Visual/UX**: até **+1,0** (sem ultrapassar 10) por experiência interativa caprichada (filtros, busca, destaque de caminhos, camadas por microrregião, etc.).

# Observações e dicas importantes

- **Padronização de nomes**: cuidem de acentos/variações (“Setúbal” como sub-bairro de **Boa Viagem**).
- **Pesos**: escolham **uma régua clara** e mantenham **consistência**; evitem pesos negativos na Parte 1.
- **Documentem** tudo no PDF: como obtiveram as interconexões (fontes/justificativas), fórmula de peso, limitações.
- **Interatividade**: `pyvis` é simples para HTML; `streamlit` é ótimo para appzinho (carregar dataset, escolher algoritmo, ver resultados).
- **Sem libs de algoritmo**: toda lógica de BFS/DFS/Dijkstra/BF deve ser **implementação própria** (ok usar `heapq` em Dijkstra).
- **Setúbal**: tratem como **nó “Boa Viagem (Setúbal)”** nas saídas e visualizações.