Documento do Projeto – Recomendação Musical com Grafos e Busca

1. Introdução

Este projeto consiste em gerar sequências de músicas com transições naturais. Cada música é representada como um nó em um grafo, e as arestas representam a diferença entre suas características musicais. O sistema deve ser capaz de, dada uma música inicial e uma música alvo, encontrar o caminho mais suave entre elas usando algoritmos de busca. O projeto será desenvolvido seguindo boas práticas exigidas pela disciplina, incluindo TDD, testes estáticos, métricas e relatório Think Aloud.

2. Qual o projeto a ser desenvolvido?

O projeto consiste no desenvolvimento de um sistema capaz de gerar **sequências de músicas com transições suaves**, utilizando **grafos** e **algoritmos de busca**. Cada música é representada como um **nó**, e a similaridade entre duas faixas é representada por **arestas ponderadas**, cujos pesos são calculados a partir de características musicais extraídas automaticamente, como *tempo (BPM)*, *energy*, *danceability*, *valence* e *acousticness*.

A partir de uma música inicial e uma música alvo, o sistema deverá encontrar o **caminho mais natural** entre elas, passando por músicas com características semelhantes. Para isso, algoritmos como **Dijkstra** ou **A*** serão aplicados para determinar o caminho de menor custo no grafo de similaridade musical.

O objetivo final é permitir a criação de playlists coerentes, onde cada transição reproduz uma evolução musical fluida, simulando um processo semelhante ao de recomendações inteligentes observadas em plataformas musicais modernas.

3. Equipe

- Allan de Albuquerque Monteiro
- Denis Almeida Ferreira
- Emerson Henrique Sulpino de Araújo
- Everton Daniel de Lima Romualdo
- Pedro Arthur de Oliveira Barreto
- Vinicius Candido Firmino

4. Tarefas do Projeto

A seguir, estão listadas **todas as tarefas necessárias** para desenvolver o sistema, suas descrições detalhadas e os responsáveis.

4.1. Coleta de Informações e Definição das Características Musicais

Responsáveis:

- Everton Daniel
- Allan de Albuquerque

Descrição da tarefa:

- Estudar como funciona a Spotify Web API.
- Escolher quais atributos musicais melhor representam similaridade (tempo, energy, danceability, valence, acousticness, etc.).
- Justificar por que esses atributos foram escolhidos.
- Definir como será estruturado o dataset final (CSV).
- Criar o script inicial para coleta de características usando *spotipy*.
- Montar o arquivo `songs.csv` com um conjunto de músicas inicial para testes.

Produtos gerados:

- Script `features.py`
- Dataset inicial `dataset/songs.csv`
- Documento curto explicando a escolha dos atributos

4.2. Modelagem do Grafo

Responsáveis:

- Emerson
- Everton Daniel

Descrição da tarefa:

- Definir como o grafo será representado (NetworkX).
- Criar uma função para transformar o dataset em um grafo.
- Definir a métrica de distância (euclidiana, cosseno, etc.).

- Construir todas as arestas calculando similaridade entre músicas.
- Armazenar corretamente pesos nas arestas.
- Criar testes unitários para esta etapa usando Pytest.

Produtos gerados:

- Arquivo `graph.py`
- Grafo inicial funcional
- Testes: `test_graph.py`

4.3. Implementação dos Algoritmos de Busca (Dijkstra e/ou A)*

Responsáveis:

- Emerson
- Allan de Albuquerque

Descrição da tarefa:

- Implementar o algoritmo de busca mais adequado:
    - Dijkstra (caso não seja definida heurística)
    - A* (se grupo decidir criar uma heurística musical baseada no vetor do destino)
- Criar uma função que recebe música inicial e final e retorna a sequência.
- Criar testes automatizados para cada algoritmo.
- Garantir que o algoritmo esteja retornando o caminho mais suave de acordo com a métrica definida.

Produtos gerados:

- Arquivo `search.py`
- Testes: `test_search.py`

4.4. Visualização

Responsável:

- Allan de Albuquerque

Descrição:

- Criar função simples usando NetworkX + Matplotlib para visualizar o grafo.
- Visualização pode ser apenas para debugging interno.

Produto gerado:

- Opcional: `visualization.py`

4.5. Testes e Métricas

Responsáveis:

- Denis Almeida, Vinicius Firmino

Descrição da tarefa:

Inclui todas as métricas exigidas pelo PDF da disciplina:

● Testes Unitários (TDD)

- Cobertura usando `pytest-cov`.
- Testar individualmente:
    - Extração de características (quando possível)
    - Construção do grafo
    - Cálculo de distâncias
    - Algoritmo de busca

● Testes Estáticos – Pylint

- Executar Pylint

- Avaliar problemas e escrever relatório curto sobre os resultados

● Métricas Estáticas do Algoritmo

- Medir tempo de execução da busca com grafos pequenos/médios
- Medir impacto da quantidade de nós
- Medir complexidade geral
- Criar relatório curto (~1 página)

Produtos gerados:

- Testes completos em `/tests`
- Relatório Pylint
- Relatório de métricas

4.6. Think Aloud

Responsável:

- Pedro Arthur de Oliveira Barreto (Entrevistador)

Para este relatório, a equipe selecionou o subconjunto considerado o mais complexo durante o desenvolvimento do projeto:

- A criação, limpeza e balanceamento do grafo de músicas, que envolveu coleta de dados, definição de pesos, tentativas repetidas e ajustes sucessivos.

Esse subconjunto foi trabalhado por Everton(entrevistado) e foi explorado em profundidade em uma sessão de Think Aloud guiada pelo entrevistador, seguindo as perguntas orientadoras propostas pelo professor.

4.6.1 Think Aloud - Construção e Balanceamento do Grafo

Compreensão do Problema

P: Qual problema estamos tentando resolver?
R: Montar um grafo grande o suficiente, com transições musicais coerentes, para que o Dijkstra conseguisse gerar recomendações naturais. A ideia era que músicas próximas fizessem sentido juntas e que fosse possível navegar pelo grafo sem saltos bruscos entre estilos.

P: Quais são as entradas e saídas esperadas?
R: Entrada: dataset com metadados das músicas. Saída: grafo em que cada música está conectada aos vizinhos mais próximos, considerando atributos musicais.

P: Existem restrições ou limitações?
R: A maior foi a API do Spotify ter descontinuado o endpoint que coletava os metadados que precisávamos. Isso destruiu o plano original. Depois, ao trabalhar com o dataset do

Kaggle, a limitação passou a ser o tamanho real dos dados e a inviabilidade de gerar um grafo completo (custo $O(n^2)$).

Abordagem da Solução

P: Como devemos abordar este problema?
R: Depois que a API deixou de servir, comecei procurando o maior dataset possível e encontrei o “Spotify Tracks”, com cerca de 120 mil músicas. No início, tentei gerar um **grafo completo**, porque achava que isso daria resultados melhores. Mas depois de várias tentativas que nunca terminavam por ser $O(n^2)$, percebi que era inviável e precisei mudar de estratégia. A partir daí passei a trabalhar somente com os vizinhos mais próximos para reduzir drasticamente o tempo de geração.

P: Quais estruturas de dados ou algoritmos podem ser adequados?
R: Usei distância euclidiana para medir proximidade entre músicas e, depois de abandonar a tentativa de grafo completo, passei a gerar apenas os vizinhos mais próximos para cada música. Isso tornou o processo viável de rodar.

P: Quais são algumas abordagens alternativas?
R: A principal alternativa foi insistir em tentar gerar um grafo completo, mas isso se mostrou inviável. Outra tentativa foi dividir o dataset por gênero, criando subgrafos com amostras mais equilibradas.

Explicação do Código

P: O que o código faz?
R: Primeiro limpa e prepara a base: remove valores faltantes, mantém apenas os atributos relevantes e garante consistência. Depois calcula a distância euclidiana entre as músicas para encontrar os vizinhos mais próximos e monta o grafo parcial.

P: Por que escolhemos esta implementação específica?
R: A distância euclidiana foi a que eu entendi melhor matematicamente. Além disso, ela permitiu implementar rapidamente e ajustar com facilidade.

P: Quais são os potenciais efeitos colaterais deste código?
R: Lentidão extrema ao tentar gerar o grafo completo. Possibilidade de conexões incoerentes caso não houvesse divisão por gêneros. Risco de desconexão ou perda de cobertura ao reduzir a quantidade de vizinhos.

Depuração

P: Onde você acha que pode estar o erro?
R: Em acreditar que um grafo completo seria possível. Também percebi que a divisão inicial por gênero não era suficiente e deixava transições musicais ruins..

P: Como podemos testar este código para encontrar o bug?
R: Medindo tempo de execução, inspecionando amostras do grafo e testando manualmente grupos de gêneros para verificar se as transições faziam sentido.

P: O que estamos tentando alcançar com este bloco de código específico?
R: Garantir que o grafo fosse, ao mesmo tempo, coerente musicalmente e viável computacionalmente.

Decisões de Design

P: Por que escolhemos este padrão de design específico?
R: Quando tentei dividir o grafo por gênero, percebi que muitos gêneros não tinham músicas suficientes para formar subconjuntos grandes. Então selecionei apenas os principais gêneros disponíveis no dataset e criei “pontes” entre eles, com base em gêneros que apareciam frequentemente relacionados. Isso evitou transições bruscas, como pular de ópera para rock pesado.

P: Como este design impacta a arquitetura geral?
R: Isso deixou o grafo mais coerente e reduziu o tempo de processamento, além de proporcionar transições musicais mais naturais.

P: Existem potenciais compensações com este design?
R: Sim. Ao focar só nos principais gêneros e trabalhar com vizinhanças menores, parte do dataset fica de fora. Isso reduz a cobertura geral do grafo.

Esclarecimento e Reflexão

P: Este código atende aos requisitos?
R: Atende, mas com limitações. O grafo funciona, mas tem menos músicas do que seria ideal.

P: Há algo obscuro ou ambíguo neste código?
R: A parte do balanceamento por gênero ainda depende muito de tentativa e erro. Não existe uma fórmula única que garanta resultados perfeitos.

P: O que estamos aprendendo com esse processo?
R: Que tarefas pesadas realmente são pesadas. Que planejamento evita retrabalho. E que heurísticas simples funcionam melhor quando há pouco tempo.

4.6.1 Principais 10 itens observados

1. Dificuldade inicial em lidar com o tamanho real do dataset.
2. Subestimação do custo computacional para gerar o grafo.
3. Impacto grande da API descontinuada no andamento do projeto.
4. Importância da limpeza e filtragem dos dados antes de qualquer cálculo.
5. Uso frequente de tentativa e erro para ajustar parâmetros.
6. Necessidade de reduzir vizinhos para tornar o grafo viável.
7. Preocupação constante com a coerência musical entre gêneros.
8. Dependência de heurísticas simples para resolver problemas práticos.
9. Observação recorrente de lentidão extrema nos primeiros testes.
10. Aprendizado claro sobre limites de processamento e importância de planejar tarefas pesadas.

5. Resumo das Tarefas e Responsáveis

Tarefa	Responsáveis
Coleta de características musicais	Everton Daniel, Allan de Albuquerque
Modelagem do grafo	Emerson, Everton Daniel
Algoritmos de busca	Allan de Albuquerque, Emerson
Visualização	Allan de Albuquerque
Testes e métricas	Denis Almeida, Vinicius Firmino
Think Aloud	Pedro Arthur