Projeto I de Estrutura de dados: Rede social simplificada

Universidade Politécnica de Pernambuco

Prof. Dr. Cleyton Mario Oliveira da Silva

Aluno: Diego Augusto Soares Amaral

11/12/2025

Resumo: Neste relatório é apresentado o desenvolvimento de um projeto de knowledge graphs (grafos de conhecimento), no qual foram implementadas funções para adicionar nós e relacionamentos, bem como operações de busca na base de conhecimento. A base de dados escolhida foi um recorte das finais da NBA entre os anos de 2015 e 2018. Por fim, são discutidas possíveis aplicações desse grafo de conhecimento e as limitações encontradas durante o projeto.
Palavras - chave: Estrutura de dados, Grafos de conhecimento, Base de dados.

Abstract: This report presents the development of a knowledge graph project, in which functions were implemented to add nodes and relationships, as well as query operations over the knowledge base. The chosen domain is a subset of the NBA Finals between 2015 and 2018. Finally, we discuss possible applications of this knowledge graph and the limitations found during the project..
KEYWORDS: Data structure. Knowledge graphs. Database.

Conteúdo

1 Introdução

1.1 Apresentação do problema

Grafos de conhecimento (knowledge graphs) são estruturas que representam um domínio por meio de entidades (nós) e relacionamentos (arestas). Em vez de apenas armazenar dados em tabelas, um grafo de conhecimento torna explícito "quem se relaciona com quem" e "de que maneira", facilitando análises mais profundas e complexas. Esse tipo de grafo vem sendo usado cada vez mais em esportes, onde empresas o utilizam para buscar perfis específicos de jogadores, medir o desempenho de cada atleta com diferentes colegas de equipe e identificar formações em que o time pode ser melhor aproveitado.

1.2 Mini-mundo escolhido

A base de conhecimento construída neste projeto é um recorte das finais da NBA entre 2015 e 2018. Esse período foi escolhido por ter sido muito marcante, com as finais disputadas sempre entre os mesmos dois times, diversos recordes quebrados e várias reviravoltas, o que torna o grafo resultante interessante de ser analisado. Os nós representam, por exemplo, jogadores, times, técnicos, arenas, séries finais e eventos históricos, enquanto as relações incluem tipos como *JOGOU_POR*, *MVP_DE*, *SAIU_DE*, entre outras.

2 Objetivos

Os objetivos do projeto são:

1. Modelar o mini-mundo das finais da NBA em termos de entidades (nós), relacionamentos (arestas) e propriedades;
2. Implementar, em Python, uma estrutura de dados (*NBAKnowledgeGraph*) com operações de inserção, remoção e consulta de nós e relacionamentos
3. Permitir a visualização do grafo completo e de subgrafos correspondentes a cada temporada;
4. Analisar possíveis aplicações do grafo de conhecimento no contexto esportivo e discutir as limitações da solução implementada;

3 Desenvolvimento

3.1 Modelagem do grafo

Neste projeto foram definidos os seguintes tipos de nós: *Team*, *Arena*, *Finals*, *Player*, *Coach* e *Evento_historico*. Os nós do tipo *Team* representam as franquias da NBA envolvidas no recorte escolhido (por exemplo, Cavaliers e Warriors), enquanto nós do tipo *Player* e *Coach* representam, respectivamente, jogadores e técnicos associados a esses times. Nós do tipo *Finals* modelam cada série de finais entre 2015 e 2018, e nós do tipo *Arena* representam os ginásios onde os jogos foram disputados. Por fim, nós do tipo *Evento_historico* registram momentos marcantes das séries, como lesões, ajustes táticos e jogos históricos.

Ainda na modelagem, foram definidos

diferentes tipos de relacionamentos entre os nós, como *JOGOU_POR*, *TREINOU*, *MANDA_JOGOS_EM*, *DISPUTOU* e *MVP_DE*. O relacionamento *JOGOU_POR* liga jogadores aos times em que atuaram em determinadas temporadas, enquanto *TREINOU* conecta técnicos às equipes que comandaram. O relacionamento *MANDA_JOGOS_EM* associa cada time à sua arena principal, e *DISPUTOU* liga os times às finais em que participaram. Já *MVP_DE* conecta o jogador eleito MVP à série de finais correspondente. Além desses, foram modelados relacionamentos específicos para eventos históricos, como lesões, ajustes táticos e transferências.

3.2 Implementação em python

3.2.1 Estrutura principal

A implementação do grafo de conhecimento foi feita na linguagem Python, por meio da classe `NBAKnowledgeGraph`. Internamente, os nós são armazenados em um dicionário `nodes`, em que a chave é o identificador do nó (`id_no`) e o valor é um dicionário de propriedades, incluindo o tipo do nó. As arestas são armazenadas em um dicionário `edges`, que associa cada nó de origem a uma lista de relacionamentos saindo dele. Cada relacionamento é representado por um dicionário contendo o tipo de relação, o nó de destino e, opcionalmente, um conjunto de propriedades adicionais.

3.2.2 Método adicionar nó

O método `adicionar_no` verifica se o identificador `id_no` já existe no grafo e, caso não exista, cria uma cópia do dicionário de propriedades recebido, adiciona a ele o campo `tipo` e armazena esse dicionário em `self.nodes`. Logo depois, inicializa em `self.edges[id_no]` uma lista vazia, que será usada para guardar os relacionamentos que partem desse nó.

3.2.3 Método adicionar relacionamento)

O método `adicionar_relacionamento` recebe um nó de origem, um nó de destino, o tipo de relação e, opcionalmente, um dicionário de propriedades da aresta. Primeiro, ele verifica se tanto a origem quanto o destino existem no grafo. Caso existam, o método cria um dicionário representando o relacionamento (com a relação, o destino e as propriedades) e adiciona ele a lista de arestas associadas à origem em `self.edges`. Caso algum dos nós não exista, o relacionamento não é criado.

3.2.4 Funções de busca

Foram implementadas diferentes operações de busca na base de conhecimento. A função `buscar_nos_por_tipo` percorre o dicionário `nodes` e retorna uma lista com os identificadores de todos os nós que possuem o tipo desejado. Já a função `buscar_nos_por_id` consulta diretamente o dicionário `nodes` para recuperar as informações associadas a um identificador específico. Além disso, a função `buscar_arestas_de_origem` permite obter todas as relações que saem de um determinado nó, facilitando a inspeção das conexões daquele elemento no grafo.

3.2.5 Funções de remoção

Também foram implementadas operações de remoção. O método `remover_no` verifica se o identificador existe em `nodes` e, em caso positivo, remove o nó, apaga a lista de arestas que saem dele em `edges` e ainda filtra, em todos os outros nós, quaisquer relacionamentos que tenham esse nó como destino. Dessa forma, o grafo não fica com arestas "soltas". Já o método `remover_relacionamento` atua de forma mais localizada: ele recebe origem, tipo de relação e destino, e remove da lista `edges[origem]` apenas a aresta que corresponde exatamente a esse trio.

3.2.6 Visualização dos grafos

Por fim, foram implementados recursos de visualização do grafo utilizando as bibliotecas NetworkX e Matplotlib. O método `para_networkx` converte a estrutura interna (`nodes` e `edges`) em um grafo direcionado do NetworkX, preservando as propriedades dos nós

e o tipo de cada relacionamento. A partir disso, o método `desenhar` calcula uma disposição automática dos nós e exibe o grafo, incluindo rótulos nos nós e nas arestas com o tipo de relação. Além da visualização do grafo completo, o método `subgrafo_por_nos` permite criar um novo grafo de conhecimento contendo apenas um subconjunto de nós e das arestas entre eles, o que foi usado para gerar subgrafos específicos para cada final entre 2015 e 2018.

3.3 Análise de complexidade

Do ponto de vista de eficiência, as operações de inserção e busca direta por identificador (`adicionar_no`, `adicionar_relacionamento` e `buscar_nos_por_id`) possuem complexidade $O(1)$, pois se baseiam em acessos diretos a dicionários ou inserções em listas. Por outro lado, funções que percorrem toda a estrutura, como `buscar_nos_por_tipo`, `imprimir_arestas` e `remover_no`, têm custo proporcional ao número de nós ($O(N)$) ou ao número de arestas ($O(E)$), o que pode se tornar mais caro em grafos muito grandes.

4 Aplicações

Mesmo com um recorte pequeno, já é possível verificar várias informações sobre o mini-mundo. Por exemplo, é possível ver rapidamente que as quatro finais foram disputadas pelos mesmos times, listar todos os jogadores que participaram das finais, identificar quem ganhou os prêmios de MVP em cada ano e consultar os eventos históricos associados a cada série.

O grafo também permite visualizar informações sobre cada um dos jogadores, como em quais anos foram campeões dentro do recorte e em quais finais receberam o troféu de MVP. Em um cenário real de análise esportiva, uma estrutura desse tipo poderia ser estendida para incluir estatísticas de desempenho, informações contratuais e dados de diferentes temporadas, ajudando em tarefas como busca de perfis de jogadores, estudo de rivalidades históricas e avaliação do impacto de transferências na competitividade dos times.

5 Limitações

O projeto apresenta algumas limitações importantes. A base de conhecimento foi construída manualmente e cobre apenas as finais entre 2015 e 2018, com um número reduzido de times e jogadores. Além disso, os dados não são persistidos em arquivo ou banco de dados, ou seja, o grafo é perdido ao finalizar a execução do programa. As operações de consulta também são relativamente simples e não utilizam linguagens de consulta específicas para grafos, como SPARQL, nem mecanismos de inferência.

Como trabalhos futuros, seria interessante ampliar o mini-mundo para incluir mais temporadas e estatísticas detalhadas de desempenho, além de integrar o grafo a fontes de dados externas (como APIs da NBA). Tambem seria possível adicionar consultas mais complexas e métricas de análise de grafos (como centralidade), aproximando o projeto dos exemplos apresentados na literatura de grafos de conhecimento.

6 Referências

1. JOSHI, Prateek. Knowledge Graph — A Powerful Data Science Technique to Mine Information from Text (with Python code). Medium, 14 out. 2019. Disponível em: ¡https://prateekjoshi.medium.com/knowledge-graph-a-powerful-data-science-technique-to-mine-information-from-text-with-python-f8bfd217accc¿. Acesso em: 10 dez. 2025.

2. LOPEZ YSE, Diego. Knowledge Graphs from scratch with Python. Medium, 17 ago. 2023. Disponível em: ¡https://lopezyse.medium.com/knowledge-graphs-from-scratch-with-python-f3c2a05914cc¿. Acesso em: 10 dez. 2025.