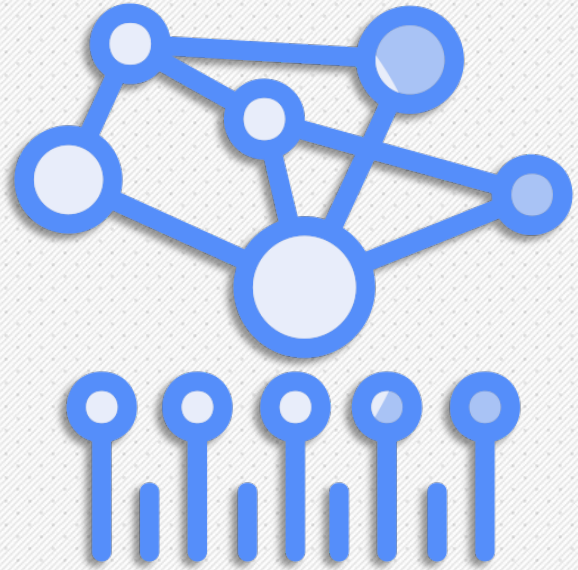


Alignment-Free Sequence to Graph

Davide Grandesso

Bioinformatic

2023/2024

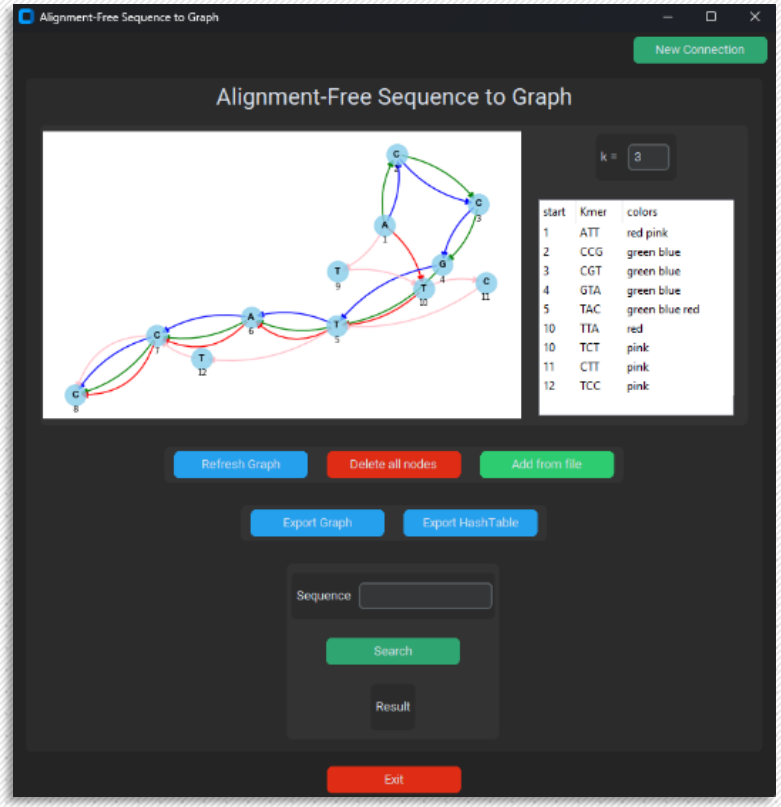
Outline

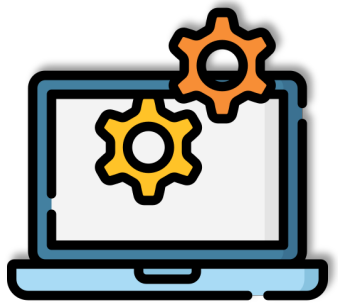
1. **Introduction**
2. **Technologies**
    1. DB & Language
    2. Data Structures
    3. Data Import
3. **Algorithmic Solutions**
    1. Class Design
    2. HashTable building
    3. Sequence Analysis

Biological context

High-throughput sequencing technologies generate massive data, allowing detailed genomic explorations.

- Challenges:
  - Deciphering intricate **patterns** and relationships within nucleotide sequences.
  - **Limitations of alignment-based methods**, computationally expensive and inefficient with large datasets.
- Advantages of graph-based analysis:
  - Graphs offer a **versatile representation** to capture complex **relationships** between sequences.
  - More **efficient analysis** and interpretation of genomic data compared to alignments.
  - Exploration of structural and functional properties of the genome through the graph's network.



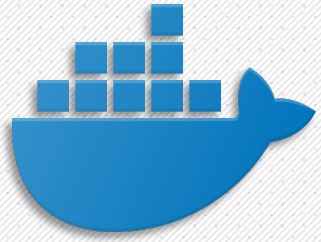


Technologies

- Database
- Programming language
- Container platform
- Graph Design
- HashTable Design
- Data Import

Neo4J & Python

- **Neo4J**: Open-source graph database
  - high performance and flexibility
  - optimized for **graph queries**
  - capability to handle **graph iterations efficiently**.
- **Python**: Open-source programming language
  - **Intuitive** syntax
  - Extensive third-party **libraries**.
- **Docker**: Platform for application deployment.
  - **Encapsulates** applications in portable drives.
  - Improves **maintainability** and **consistency** of deployment.



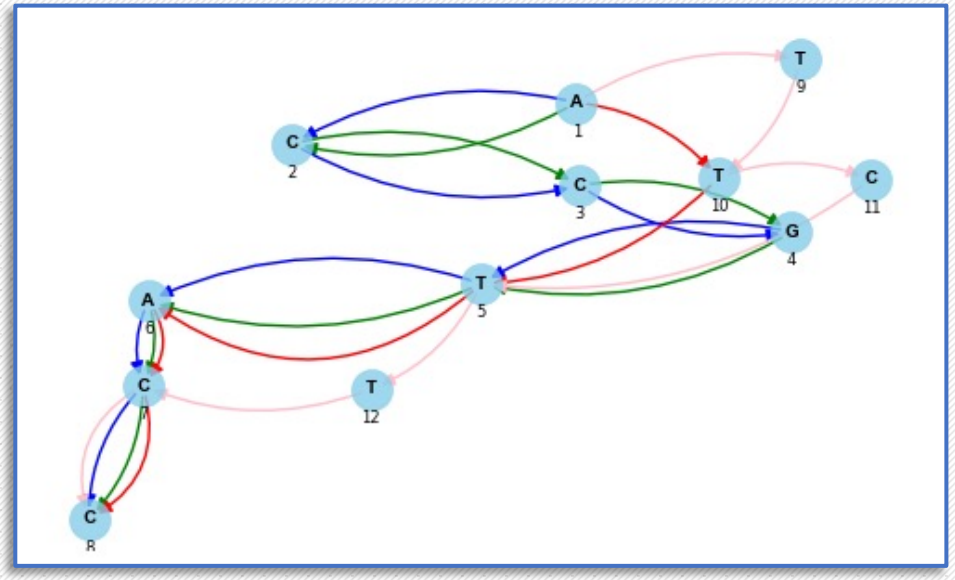
Data Structures – Graph design

Nodes are labeled with **":base"**

- **name**: Represents the nitrogen base character
- **id**: Uniquely identifies the node as an integer

```
[:base {id:1, name:"A"}]-[:blue]->[:base {id:2, name:"C"}]
```

Management of large graphs with multiple colored sequences passing through shared nodes



Data Structures – HashTable

Python dictionary chosen as surrogate for HashTable.

Unique **k-mers** as keys and tuples as values.

- **First element** of tuple: ID of initiating k-mer node.
- **Second element**: Array containing colors attributed to the k-mer.

```
{
"ATT": (1, ["red", "pink"]),
"CCG": (2, ["green", "blue"]),
"CGT": (3, ["green", "blue"]),
"GTA": (4, ["green", "blue"]),
"TAC": (5, ["green", "blue", "red"]),
"TTA": (10, ["red"]),
"TCT": (10, ["pink"]),
"TCC": (12, ["pink"]),
"CTT": (11, ["pink"])
}
```

Data Import

Import from **JSON**

- Utilizes specialized formatting tailored for the application.
- **Nodes**: ID, character representation, optional label
- **Relationships**: Starting node, ending node, relationship label

Import from **GFA**

- Parsing information from GFA (Graphical Fragment Assembly)
- **Nodes** represented by segment
- **Relationships** established based on defined paths

Algorithmic Solutions

- Class Design
- HashTable Building
- Sequence Analysis

Class Design

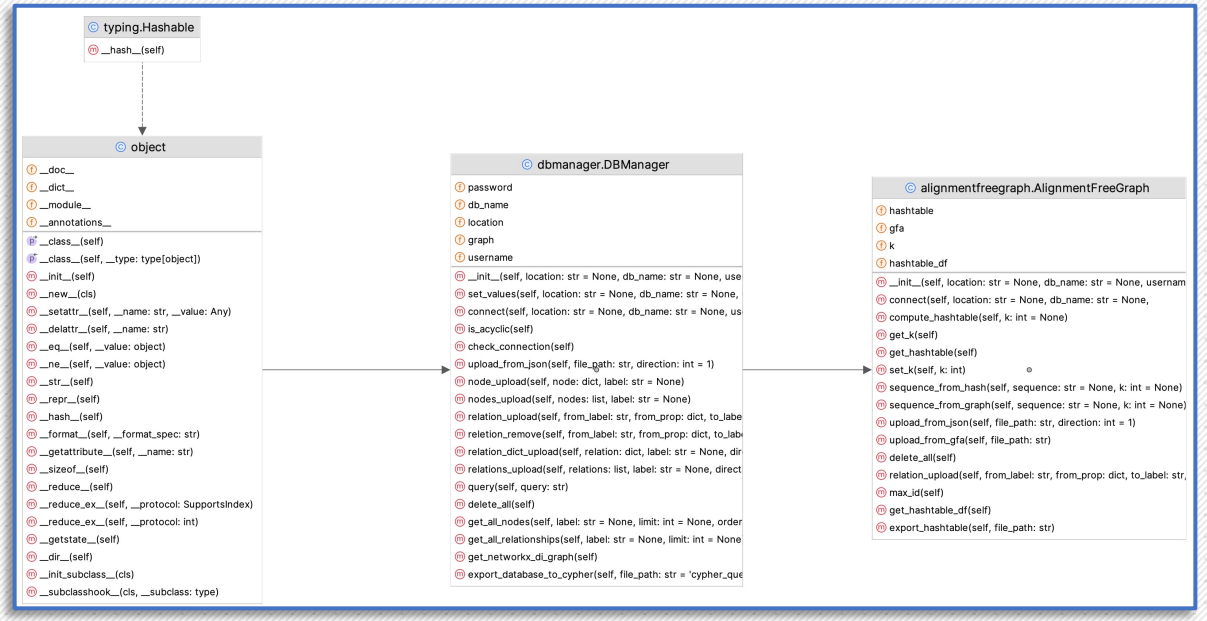
Prioritize the principles of modularity and extensibility

DBManager Class (dbmanager.py):

- Manages interactions and queries with Neo4j
- Responsible for establishing and executing database operations.

AlignmentFreeGraph Class (alignmentfreegraph.py)

- Core component of the project.
- Handles graph import, HashTable construction, and sequence analysis.
- Validates graph as a DAG (Direct Acyclic Graph)



HashTable building

1) **Initialization**: HashTable initialized as a dictionary with k-mers as keys and corresponding nodes as values.
2) **Construction from Graph**: Query made on the database to extract all k-character sequences.
3) **Handling Collisions**: retaining only unique k-mers in the graph, even if shared by multiple sequences.

Start	Kmer	Colors
1	ATT	['red', 'pink']
2	CCG	['green', 'blue']
3	CGT	['green', 'blue']
4	GTA	['green', 'blue']
5	TAC	['green', 'blue', 'red']
10	TTA	['red']
10	TCT	['pink']
11	CTT	['pink']
12	TCC	['pink']

$$O(NK) + O(NK) = O(NK)$$

N → Number of nodes K → Length of k-mers

Sequence Analysis

1) **Sequence Parsing**: extracting k-mers from the sequence.
2) **HashTable Lookup**: find k-mers in the HashTable
3) **Vertex Identification**: identify the starting node of the k-mers
4) **Handling Collisions**: checking whether the k-mers analyzed have no sequence in common with each other

"CGTTCC"

(3, 12)

$$O(L)$$

L → Number of k-mers in the sequence to be analyzed

Thank you for your attention

Davide Grandesso