# Graphs & genome assembly

## 1. Introduction

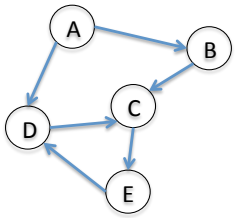
DNA sequencing machines cannot read a genome from beginning to end. Instead, they can only read short fragments of DNA (100-250 nt in case of Illumina technology). Genome assembly refers to the technique to reconstruct the original genome by putting all the pieces together.

The intuitive approach to genome assembly is to compare all DNA pieces to all others to find overlap, and then to join overlapping pieces. Where this was working for relatively few and long Sanger reads, the all-vs-all comparison is not computationally feasible anymore for millions/billions of short reads. So they had to come up with something else, avoiding the all-vs-all comparison. The mathematical concept of “de Bruijn graphs” came to the rescue...

This topic will provide an introduction to graphs and graph theory. A graph is a representation of a set of objects (called nodes) and relationships between these nodes (called edges). There are many graph algorithms that have applications in bioinformatics. Genome assembly is one of the topics where graphs are very useful, but we will see them back later in the course as well.

In this assignment you will implement an algorithm to find a Eulerian cycle or path in a graph. The algorithm is described in Jones and Pevzner section 8.8.

Directed graphs are stored as a dictionary with the nodes as keys and a list of connected nodes as values. For example:




```
{'A': ['B','D'],
 'B': ['C'],
 'C': ['E'],
 'D': ['C'],
 'E': ['D']}
```

See also: https://www.python.org/doc/essays/graphs/

### Reading material

Chapter 8 of Jones and Pevzner introduces graphs and genome assembly. Focus on section 8.6-8.8 for the assignment. The paper “How to apply de Bruijn graphs to genome assembly” also gives a nice introduction to the topic.

## 2. Assignment

Use the code skeleton in **`assignment3_skeleton.py`** to:

- Implement a function (e.g. **`is_eulerian(graph)`**) to determine whether a graph is Eulerian (theorem 8.1)
- Implement a function (e.g. **`has_eulerian_path(graph)`**) to determine whether a graph has an Eulerian path (theorem 8.2)

- Implement a function (e.g. **`find_eulerian_cycle(graph)`**) to find an Eulerian cycle in a graph. The algorithm is described in section 8.8. Use the graph from Fig 8.22 for development. You may create as many supporting functions as necessary.
- Implement a function (e.g. **`find_eulerian_path(graph)`**) to find an Eulerian path in a graph. The algorithm is described in section 8.8. Use the example from Fig 8.20 for development. You may create as many supporting functions as necessary.
- Implement a function to create a graph from a spectrum (Fig 8.20).

**Questions:**

1. Is the graph from Fig 8.22 Eulerian (in other words: what's the result of your function **`is_eulerian`**, given this graph as input)?
2. Does the graph from Fig 8.22 contain a Eulerian path?
3. Print the Eulerian path that your code can find in graph_822.
4. If you run it 3 times (print your results below), do you always find the same path? Why or why not?
5. Print the graph that you constructed from the spectrum s (Fig 8.20). Use:
```
for k, v in graph.items():
    print(k, v)
```
6. Is this graph Eulerian? Why or why not? And does it contain a Eulerian path? Why or why not?
7. Print the Eulerian path that your algorithm finds. To which DNA sequence does this path correspond? Print the sequence.
8. Which Eulerian cycle or path (if any) do you find in the **`bigger_graph`** (provided in the skeleton)?
9. Is the algorithm exact or approximate?
10. Is the running time of Euler's algorithm proportional to the number of nodes or the number of edges in the graph? Can you explain why?

**Hints**

- The nodes in Fig 8.22 are labeled A through J, and the graph is provided in the skeleton. The nodes of the bigger_graph are numbered 1 to 12. A sketch of the graphs is provided on BlackBoard (figures).

**Please, hand in your Python script, and a PDF file containing the answers to the questions above by e-mail, to:**
**algorithms@bioinformatics.nl no later than 9.00 on the day of the lecture.**