Eulerian paths and k-mers assembly

**Breaking down a read into a k-mers list**

A list of $k$-mers is created from a list of reads, implement a list that breaks down a read into its fragments of size $k$

**Example:**

```
buildkmers("ACGUACGACU", k=4) → ["ACGU", "CGUA" "GUA" "UAOG" "ACGA", "CGAC", "GACU"]
```

**Computing overlaps**

Given a list of $k$-mers, when need to compute the (directed) pairs of $k$ mers ($w, w'$) that overlap on ($k-1$) letters (the ($k-1$) first letters of $w'$ are the same as the ($k-1$) last ones of $w$)

**Example:**

```
overlaps(["ACGU", "CGUU", "ACGG"]) → ["ACGU", "CGUU"], ["ACGU", "CGUG"]
```

It is possible to use classic Python data structures, like `dict`, or use an optimized prefix tree.

**Building the de Bruijn graph**

Build the de Bruijn graph as shown in the lecture.

To represent the graph $G$, we will use the `networkx` which includes many graph algorithms.

**Example:**

```
deBruijn(reads, k) → networkx.Graph object
```

Using the `draw` method of `networkx` (+ `matplotlib`), check on the example given in the lecture that your implementation produces the same graph.

**Checking the input**

Check that the graph in Eulerian is amenable to an assembly:

- indegree=outdegree for all nodes, except maybe for start/end nodes;
- single connected component.

**Example:**

```
check_deBruijn(["AGGG", "CCCC"], 2) → False
```
Towards Eulerian path

Implement the Eulerian path reconstruction algorithm to generate a possible assembly of $k$-mers

**Example:**

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>eulerian(reads,k)</code></td>
<td>&quot;XXX&quot;</td>
</tr>
</tbody>
</table>

Beyond the single genome

Generate the list of all possible ordering for a given initial path. Is that all there is?

**Example:**

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>all_eulerian(reads,k)</code></td>
<td>[&quot;XXX&quot;,&quot;YYY&quot;...]</td>
</tr>
</tbody>
</table>