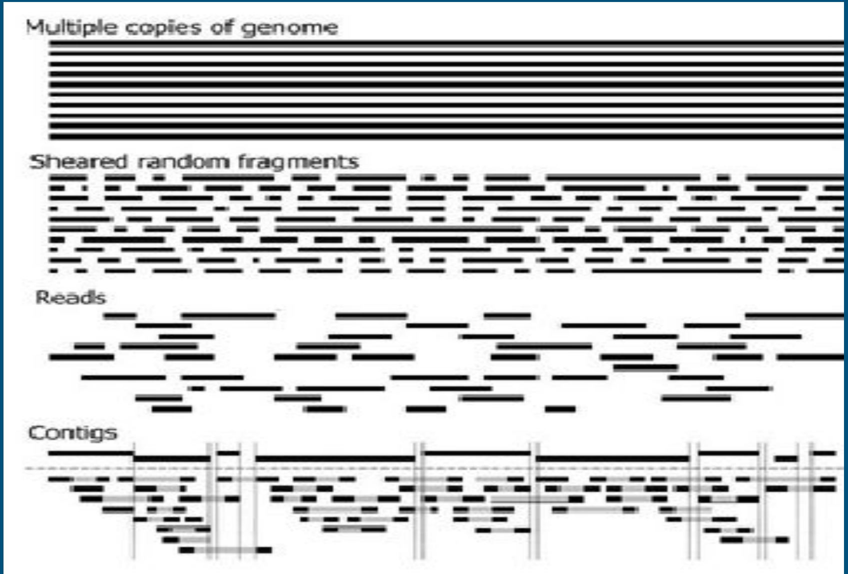
Genome Assembly and Comparison Using De-Bruijn Graph

Sarwar Ali Siddiqui (BIM2015007)
Shivani Gupta (MBI2018006)
Ankit Mishra (MBI2018007)
Tanya Gohit (MBI2018008)

Why do we need genome assembly??

- Cannot read the complete genome with the sequencer from one end to another.
- DNA isolated from a cell is amplified.
- Broken into fragments (shearing).
- Fragments are "read" from the sequencer.
- Use the fragments - read to reconstruct the genome from sequencing reads.
- Assembly - hierarchical process to reconstruct genome from reads.
- Assemble the puzzle of the genome from reads. Overlap connect the pieces.
- Oversample the genome so that read overlaps.
- Key approach- data structure representing overlaps and algorithm operating on that data structure.



redit: Masahiro Kasahara, Large-Scale Genome Sequence Processing, Imprerial College Press

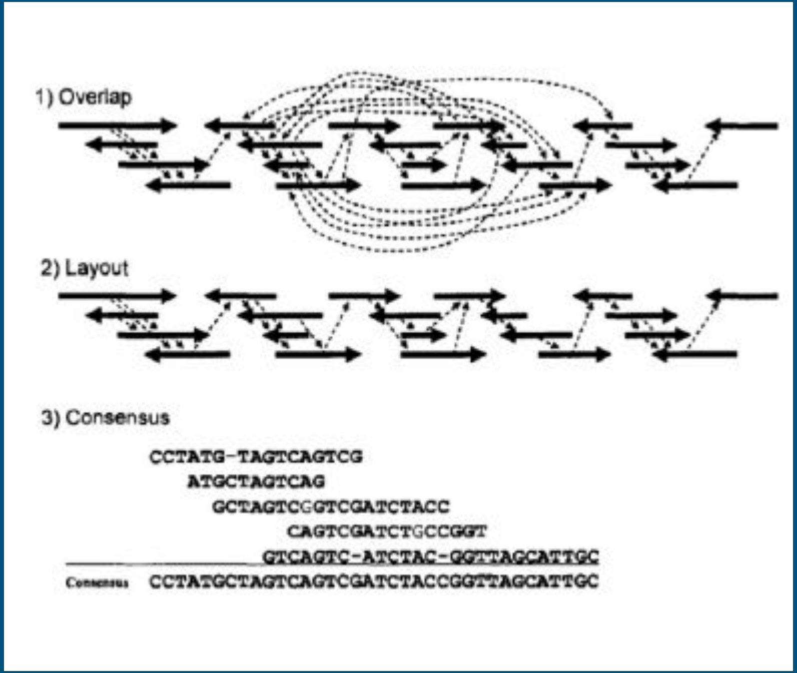
Two major algorithmic paradigms for genome assembly

- Overlap - Layout - Consensus (OLC): well established, more powerful method, but more difficult to implement.

- deBruijn - Euler: newer, easier to implement, problematic in complex genomes.

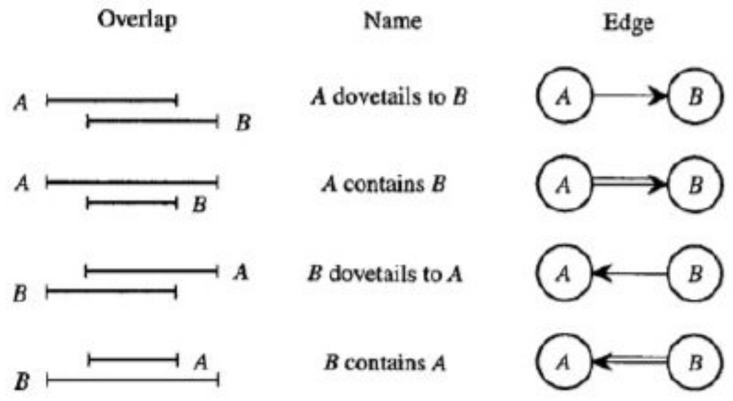
OLC

- Finds overlaps by aligning the sequence of reads.
- Layout the reads based on which aligns to which
- Get consensus by joining all read sequences, merging overlaps
- Sequencer reads in random direction left to right or right to left.

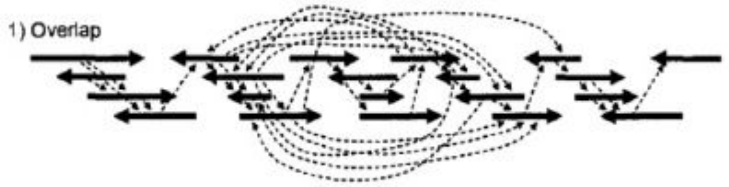


OVERLAP

- Sequence alignment, all-against-all-reads.
- Computationally intensive but easily parallelizable.
- Represent read overlap by connected with directed link.
- First step in creating the genome assembly graph.

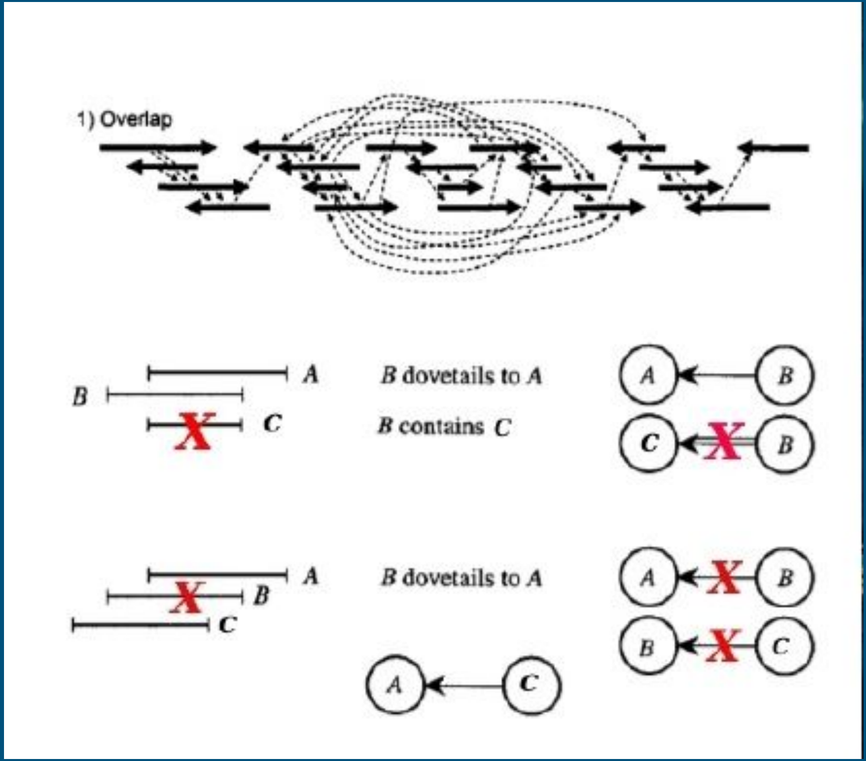


Credit: Kececioglu and Myers 1995, Algorithmica 13:7-51



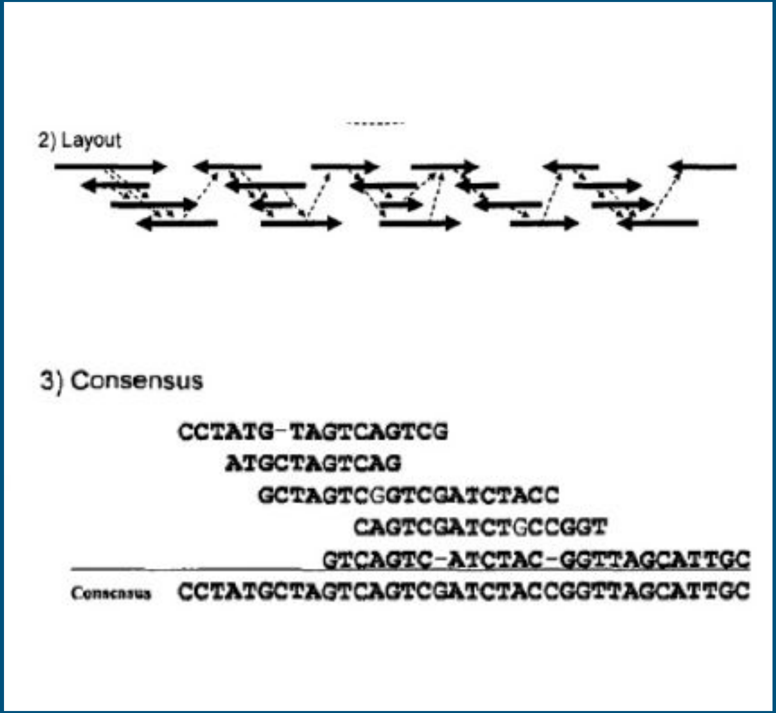
LAYOUT

- Create a consistent linear ordering of the reads.
- Remove redundancy, so no two dovetails have the same edge.
- No containment edge is followed by a dovetail edge.
- Remove cycles, one link in, one out.



CONSENSUS

- Multiple sequence alignment (CLUSTALW)algorithms.
- Vote for the most abundant nucleotide for each position.
- Incorporate read quality data.
- Create pre consensus from high quality reads, and align remaining reads to it.



De Bruijn and overlap graphs

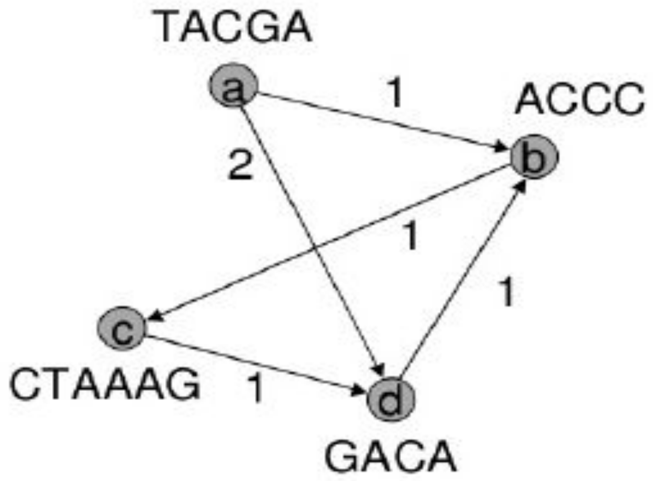
- The ”consensus-layout-overlap” assembly methods use the overlap graph structure which is quite similar to the de Bruijn graph.
- However, the fundamental difference lies in the scale of resolution of these different structures.
- A de Bruijn graph is inherently k-mer centric,meaning that its topology is unaffected by the fragmentation of the read.
- Secondly, because of the one-to-one relationship between paths andsequences, overlapping sequences necessarily follow the same path. Thissimplifies the search for consistently overlapping sets of reads.

- The de Bruijn graph does not integrate all the calculations needed to produce the overlap graph.

- Especially with longreads, global pairwise alignments can be extremely useful to determine re-liably if two reads really come from the same genomic locus.

- Constrained by the k-mer length, the de Bruijn graph can accumulate false-positive overlaps which could have easily been detected by pairwise alignment.

Graph and genome assembly

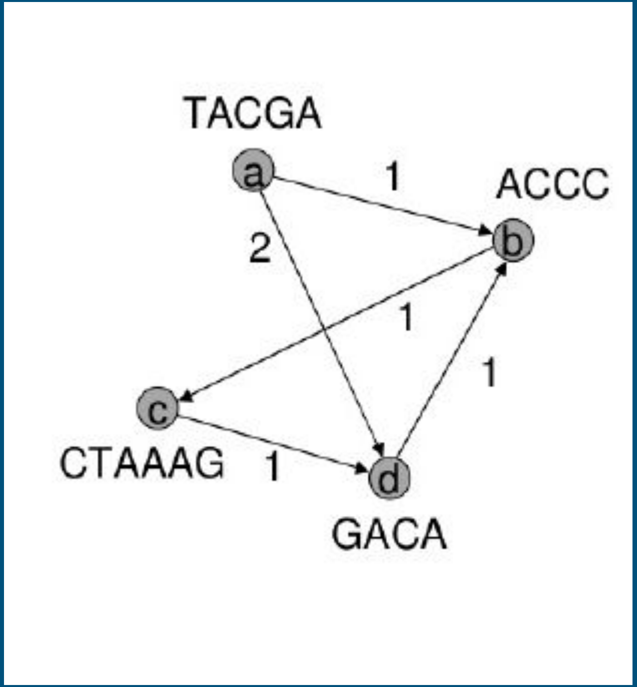


Shortest Common Superstring (SCS)

```
TACGA
   GACA
     ACCC
        CTAAAG
------------------
TACGACACCCTAAAG
```

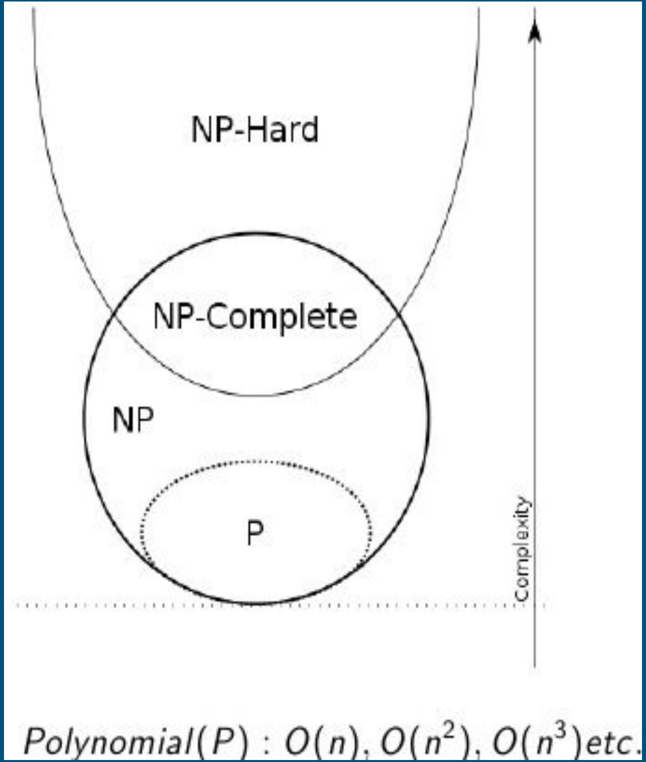
find the shortest string that contains all the sequences as substrings

- Represent sequence overlaps as a graph with weighted edges.
- SCS Solution: find path (visit all edges and vertices once) that maximizes weight sum.
- Hamiltonian Cycle or travelling salesman problem.



Computational Complexity

- SCS solution by searching for a Hamiltonian cycle on a graph is a difficult algorithmic problem (NP-hard).
- Using approximation or greedy algorithm can yield 2 to 4-approximation solutions (twice or four times the length of the optimal-shortest string).
- Transformation of Overlap Graph to String Graph to Polynomial time solution.



ATGCG

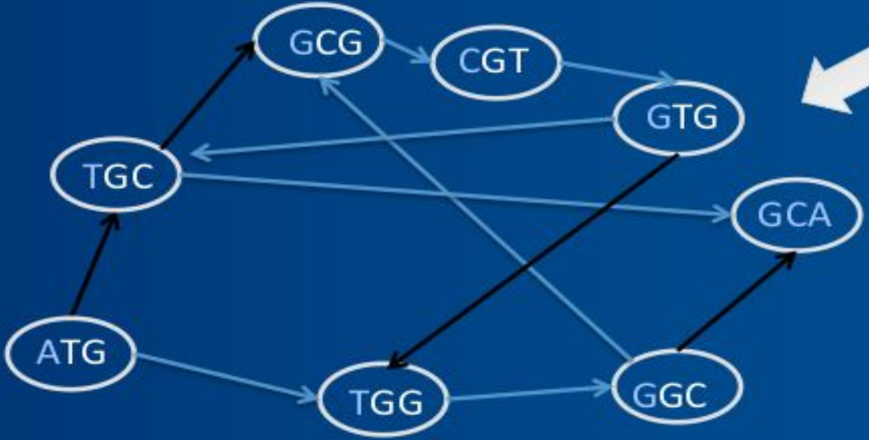
GCGTG

GTGGC

TGGCA

$\xrightarrow{\text{3-mers}}$

ATG, TGC, GCG,
~~GCG~~, CGT, GTG
~~GTG~~, TGG GGC
~~TGG~~, ~~GGC~~, GCA



Find **Hamiltonian path**, that is, a path that visits every <u>vertex</u> exactly once

Record the First letter of each vertex + All letters of last vertex

ATGCGTGGCA

ATGGCGTGCA

- Thus, people generally use *<u>k-mers of certain length</u>*

← Here we use *3*-mers by cutting the original reads into reads of length 3

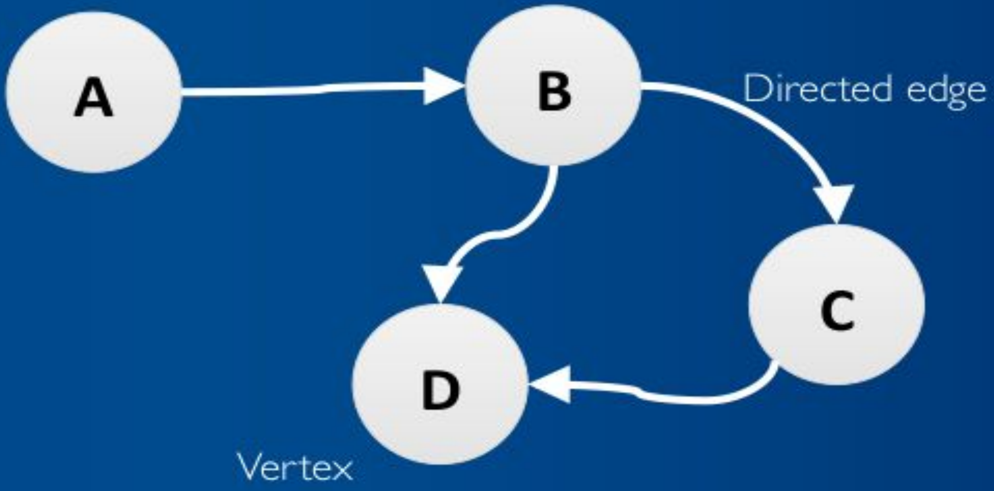
← **UNFORTUNATELY**: The Hamiltonian path problem is <u>very difficult to solve</u> (np-complete)

Seven bridges of Königsberg

- In 1735 Leonhard Euler was presented with the following problem:
  - find a walk through the city that would cross each bridge once and only once
  - He proved that a connected graph with undirected edges contains an Eulerian cycle exactly when every node in the graph has an **even** number of edges touching it.
  - For the Königsberg Bridge graph, this is not the case because each of the four nodes has an odd number of edges touching it and so the desired stroll through the city does not exist.

Assembly as a graph theoretical problem

- The degree of a vertex: # of edges connected to it
- outdegree: # of outgoing edges
- indegree: # of ingoing edges
- degree(B)?
- outdegree(B)?
- indegree(D)?



Seven bridges of Königsberg II

- The case of directed graphs is similar:
  - A graph in which indegrees are equal to outdegrees for all nodes is called 'balanced'.
  - Euler's theorem states that a connected directed graph has an Eulerian cycle if and only if it is balanced.

- Mathematically/computationally finding Eulerian path is much easier than Hamiltonian

  → we need to reformulate our assembly problem

We construct a ***de Bruijn graph***:

- edges represent *k*-mers
- vertices correspond to (*k*-*1*)-mers

1. Form a node for every distinct prefix or suffix of a *k*-mer
2. Connect vertex *x* to vertex *y* with a directed edge if some *k*-mer (e.g., ATG) has prefix *x* (e.g., AT) and suffix *y* (e.g., TG), and label the edge with this *k*-mer.

k-mers: ATG, TGG, TGC, GTG, GGC, GCA, GCG, CGT

Distinct (k-1)-mers:

Can we find a DNA sequence containing all *k*-mers?

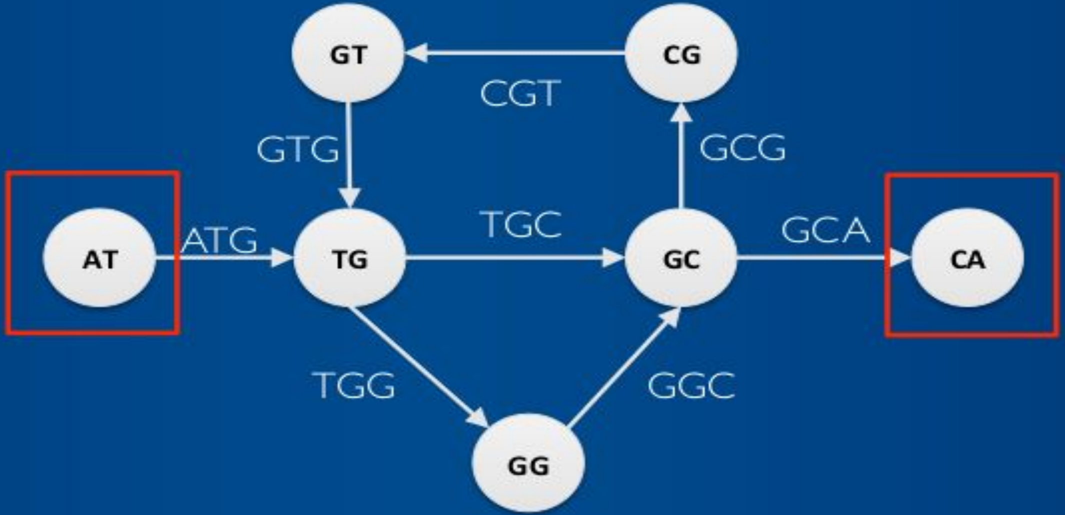
→ **In a *de Bruijn* graph, can we find a path that visits every edge of the graph exactly once?**

→ **Eulerian path**

- a vertex *v* is semibalanced if |indegree(v) – outdegree(v)| = 1
- a connected graph has an Eulerian path if and only if it contains at most two semibalanced vertices

k-mers: ATG, TGG, TGC, GTG, GGC, GCA, GCG, CGT

Distinct (k-1)-mers:



Underlying assumptions of genome assemblies

- Four hidden assumptions that do not hold for next-generation sequencing
  We took for granted that:
  1. we can generate all *k*-mers present in the genome
  2. all *k*-mers are error free

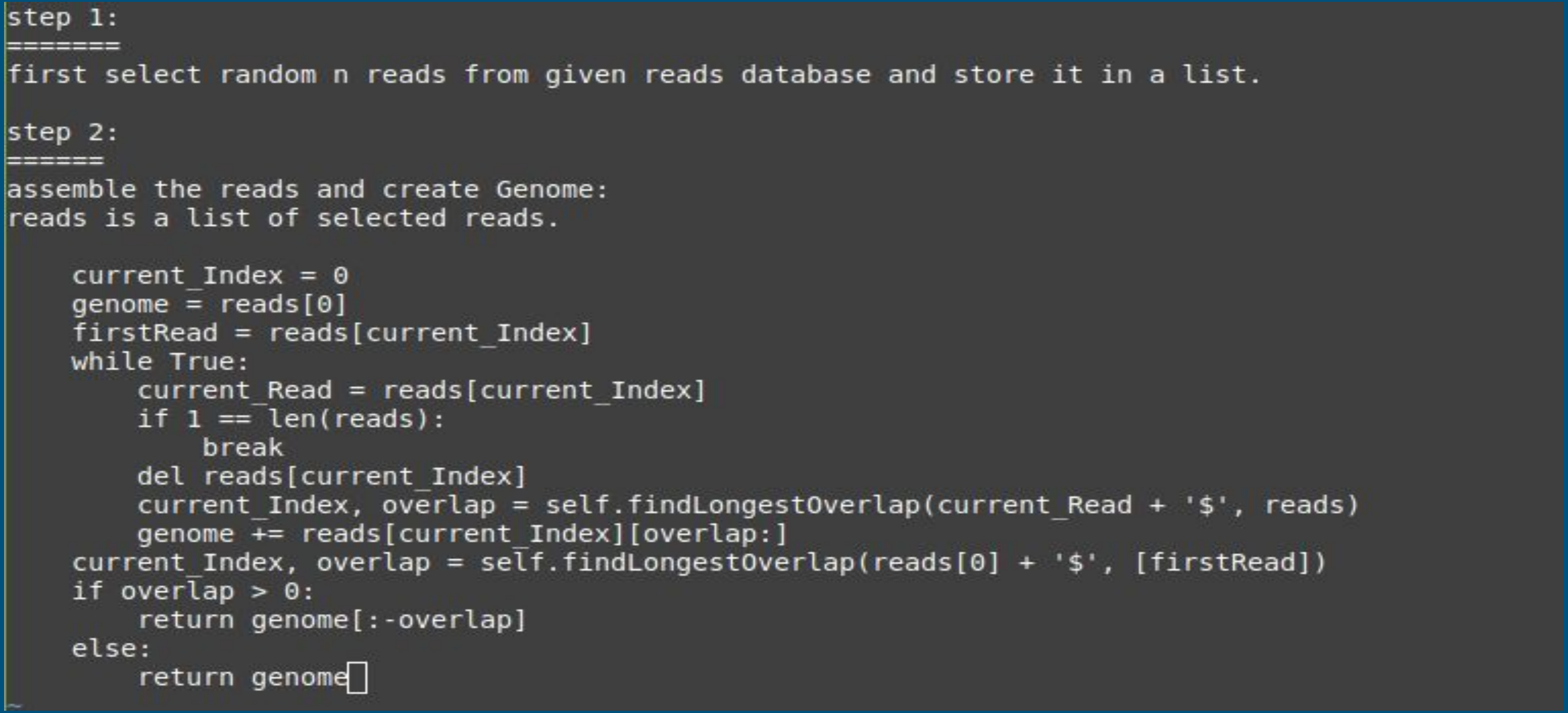
- Due to these reasons we do ***NOT*** choose the longest possible *k*-mer
- The ***smaller*** the *k*-mer the higher the possibility that we see ***all*** *k*-mers
- Errors:

$$\begin{tikzcd}
 & \text{ATG TGG GGC GCG CGT GTG TGC GCA} \\
\text{ATGGCGTGCA} \arrow[ur, "K=3"] \arrow[dr, "K=10"'] & \\
 & \text{ATGGCGTGCA}
\end{tikzcd}$$

Mostly unaffected *k*-mers

100% affected *k*-mers

Assembling genome from error-free reads.



```
step 1:
========
first select random n reads from given reads database and store it in a list.

step 2:
=======
assemble the reads and create Genome:
reads is a list of selected reads.

    current_Index = 0
    genome = reads[0]
    firstRead = reads[current_Index]
    while True:
        current_Read = reads[current_Index]
        if 1 == len(reads):
            break
        del reads[current_Index]
        current_Index, overlap = self.findLongestOverlap(current_Read + '$', reads)
        genome += reads[current_Index][overlap:]
    current_Index, overlap = self.findLongestOverlap(reads[0] + '$', [firstRead])
    if overlap > 0:
        return genome[:-overlap]
    else:
        return genome
```