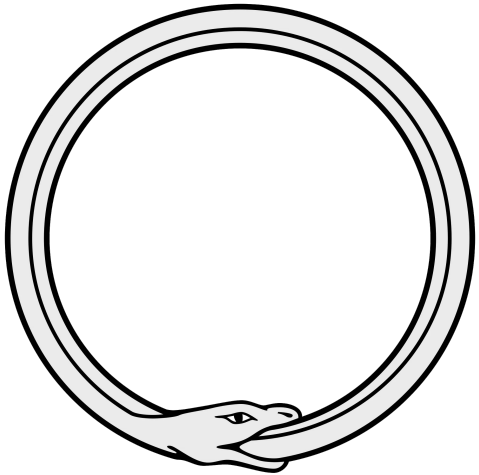
Genome assembly

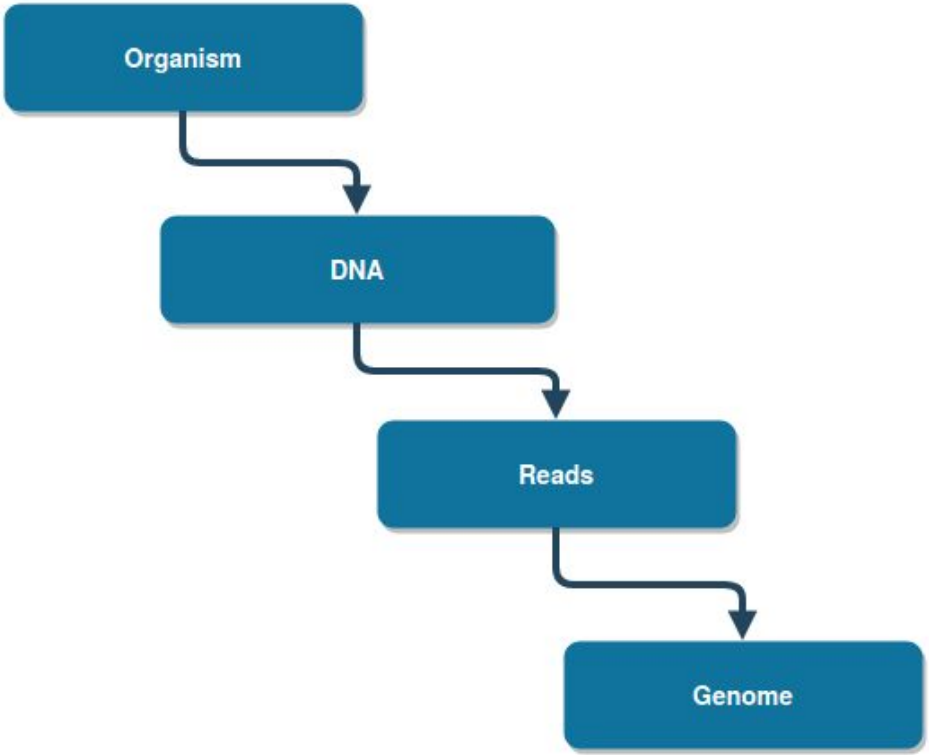


Why do we need it?

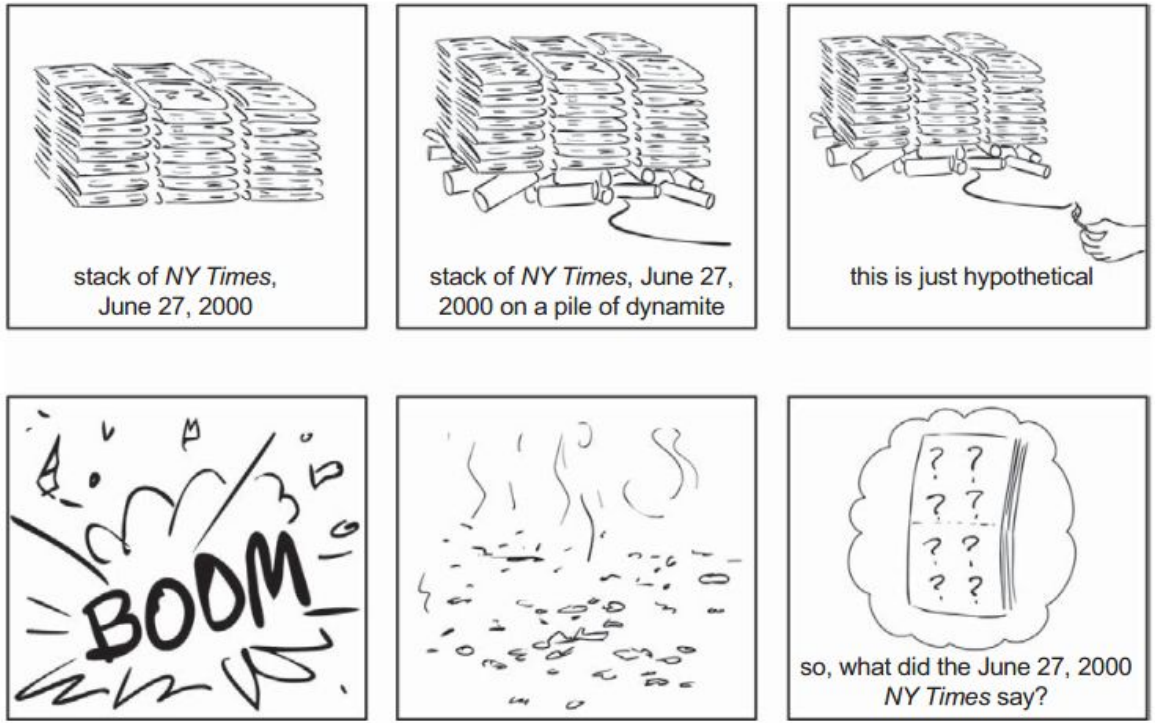
No need to ask this question - genome sequence knowledge is crucial or beneficial for many applications including

- Gene searching
- Transcriptome analysis
- Primer construction
- Mobile elements investigation
- Genome stability exploration
- Phylogeny

Typical information for genome assembly



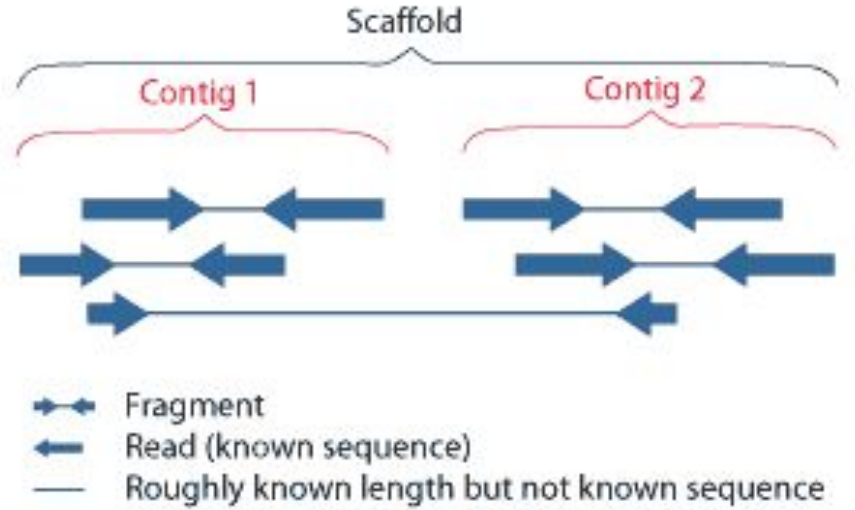
So, something like this...



Levels of genome reconstruction

Just some definitions to be honest

1. read - stretch of DNA from the sequenator
2. contig - overlapped reads which are merged together
3. scaffold - contigs and gaps between them



Assembly strategies

Depend on read type (for example, their length)

1. Naive
2. Greedy
3. Overlap Layout Consensus (OLC)
4. de Bruijn graph (DBG)
5. Super-reads approach
6. TF-IDF approach

Naive approach

Well, looks pretty simple

1. Align locally each read vs another
2. Take alignments with scores higher than some threshold and unite pair of reads
3. Repeat previous steps

Sequencing example

Let's allocate reads near their origin. Read length is equal to 5 and genome is БИОИНФОРМАТИКА

```
                БИОИНФОРМАТИКА

1               БИОИН
2                ИОИНФ
3                 ОИНФО
4                  ИНФОР
5                   НФОРМ
6                    ФОРМА
7                     ОРМАТ
8                      РМАТИ
9                       МАТИК
10                       АТИКА
```

So we just align each read vs all others, take best alignments and merge them into contigs. After this we obtain scaffold, in our simple case we have good result

```
БИОИН                  ОИНФО
  +      БИОИНФ          +      ОИНФОР      ...
 ИОИНФ                  ИНФОР


БИОИНФ
  +      БИОИНФОР...   БИОИНФОРМАТИКА
  ОИНФОР
```

OLC

Improved version of previous algo

1. Align reads with each other
2. Create a graph (map) of reads connections
3. Fix inconsistencies of overlaps via multiple alignment

So in our simple case it looks like

read1 -> read2 -> read3 -> read4 -> read5 -> read6 -> read7 -> read8 -> read9
|
v
read10

Ambiguos overlaps are resolved with alignment of reads and taking consensus
We don't have such problems here (we have precise bioinformatics here =))

Imagine we have several reads from 1 genome position

```
            reads
            БИОИН
             ИОИНФ
             ИАИНФ
             ИОИНФ
consensus   БИОИНФ
```

De Bruijn Graph

Very popular nowadays, was introduced to genome assembly by Pevzner

1. Split reads into k-mers with fixed k, usually k is odd
2. Build de Bruijn graph from them - (k - 1)-mers are the vertices and edges are k-mers (overlap sequence between edges)
3. Process this graph

Let's look at this strategy on our example with БИОИНФОРМАТИКА genome
read_length = 5, k = 4

Reads	k-mers	(k - 1)-mers
БИОИН	БИОН, ИОИН	БИО, ИОН, ИОИ, ОИН
ИОИНФ	ИОИН, ОИНФ	ИОИ, ОИН, ОИН, ИНФ
ОИНФО	ОИНФ, ИНФО	ОИН, ИНФ, ИНФ, НФО
ИНФОР	ИНФО, НФОР	ИНФ, НФО, НФО, ФОР
НФОРМ	НФОР, ФОРМ	НФО, ФОР, ФОР, ОРМ
ФОРМА	ФОРМ, ОРМА	ФОР, ОРМ, ОРМ, РМА
ОРМАТ	ОРМА, РМАТ	ОРМ, РМА, РМА, МАТ
РМАТИ	РМАТ, МАТИ	РМА, МАТ, МАТ, АТИ
МАТИК	МАТИ, АТИК	МАТ, АТИ, АТИ, ТИК
АТИКА	АТИК, ТИКА	АТИ, ТИК, ТИК, ИКА

Graph


БИО
БИОИ
ИОИ
ИОИН
ОИН
ОИНФ
ФОР
НФОР
НФО
ИНФО
ИНФ
ФОРМ
ОРМ
ОРМА
РМА
РМАТ
МАТ
МАТИ
ИКА
АТИК
ТИК
АТИК
АТИ

About coverage

Let's compute the coverage of our genome and think how it is affected by k-mering

```
                БИОИНФОРМАТИКА
          1     БИОИН
          2      ИОИНФ
          3       ОИНФО
          4        ИНФОР
          5         НФОРМ
          6          ФОРМА
          7           ОРМАТ
          8            РМАТИ
          9             МАТИК
          10             АТИКА

coverage        1 3 5 5 5 4 2

                     2 4 5 5 5 3 1
```

Coverage after creating k-mers

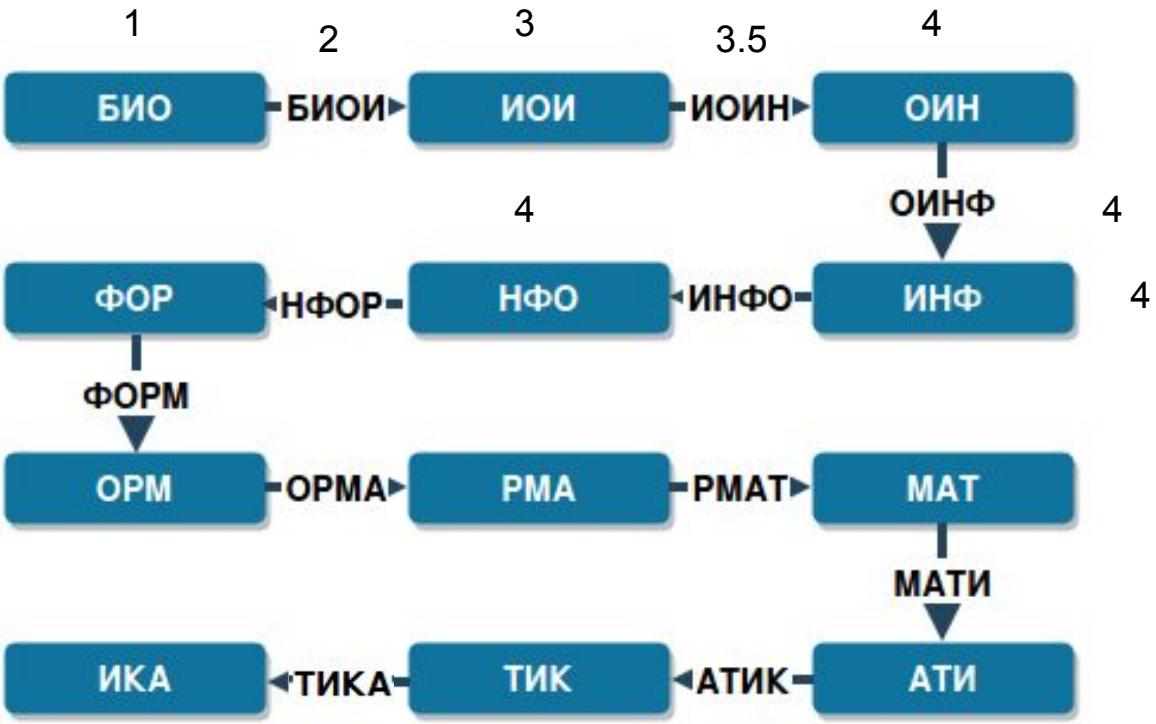
Let's compute the coverage of our genome and think how it is affected by k-mering

```
                БИОИНФОРМАТИКА
          1     БИОИ
                 ИОИН
          2      ИОИН
                  ОИНФ
          3       ОИНФ
                   ИНФО
          4        ИНФО
                    НФОР
          5         НФОР
                     ФОРМ
                ...

coverage      1 6 8

                  3 7 8
```

Graph with coverage



Coverages

- vertex coverage - number of such sequences (k - 1) in data
- edge coverage - computed from vertex coverage with such formula

$$E_c = \frac{V_{c-in} + V_{c-out}}{parts}$$

E - coverage of the edge
V - coverage of the vertex
parts - number of vertex which built this edge - 2 by default (each edge's vertex contribute 1)

Condense graph

We can condense all straight parts (without branching) of our graphs

$$E_c^* = \frac{E_c \cdot parts_1 + V_{new}}{parts_1 + parts_2 - 1}$$

$$\frac{2 \cdot 2 + 4}{2 + 2 - 1} = 2.66$$

Condensed graph

During this process we reduce number of vertices and edges, elongate our edges and recompute their coverage. After that we have condensed graph possibly with some conflicts

Possible problems

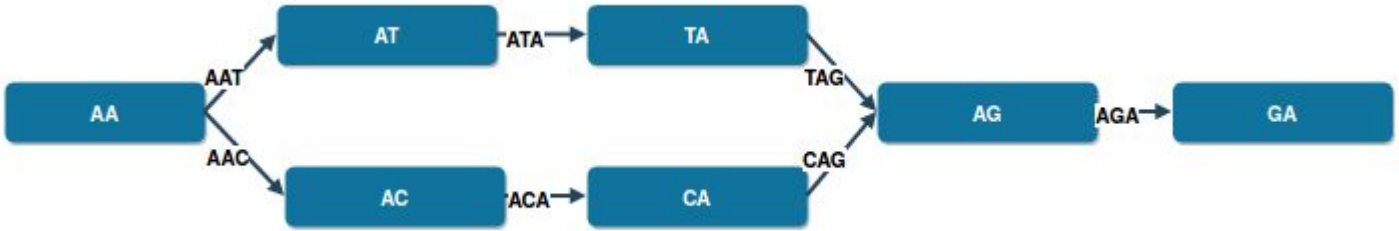
Possible origins - sequencing errors, polymorphism (especially in polyploid organisms), different repeats, contamination

- tips
- bubbles
- chimerism

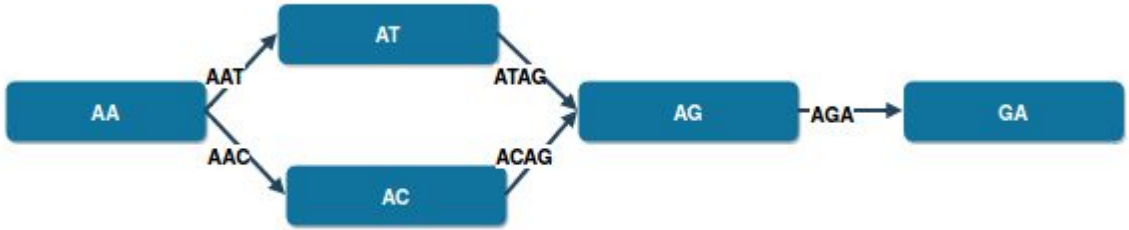
Bubble

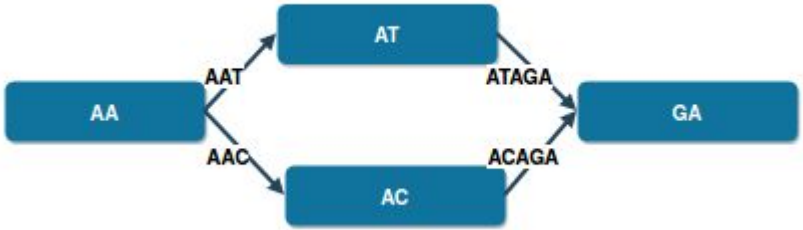
Let's illustrate it with more DNAic example

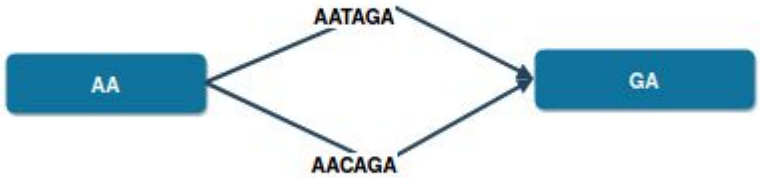
Genome	AATAGA
Reads	AATA
	AACA and so on



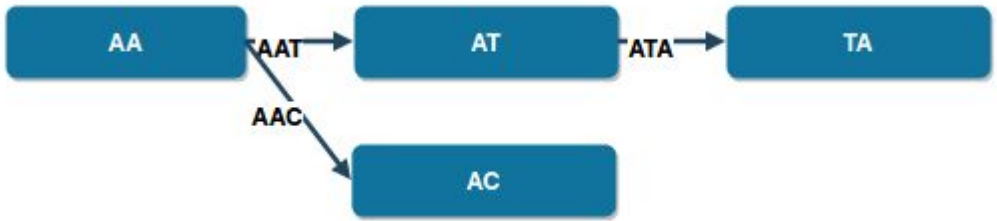
After condensing







Tip



Possible solutions

Prune tips if their coverage is low (less than some threshold)

Choose way in the bubble with the highest coverage

Possible additional steps

Assemblers would like to get ideal reads. So we can do the following before the assembly

- clean dataset from contamination
- deduplicate reads
- correct reads

Contigs

Just edges of collapsed graph

БИОИНФОРМАТИКА

AATAGA

AACAGA

Super-reads

Implemented in masurca

1. Build hash table where we store frequency of each k-mer in our dataset (like a dictionary with k-mers keys and their occurence as values)
2. Elongate reads to super-reads
3. Assemble super-reads

Super-reads construction

Look at k-mers in the end of the reads

read	kmer	Hash table	kmer	elongated read
AATA	ATA		ATAT	AATAT

```
{
    'TAG': 0,
    'TAC': 0,
    'TAA': 0,
    'TAT': 10,
    'ATG': 10, ...
}
```

Only 1 k-mer from 4 possible is present in dataset TAT (no TAA, TAC, TAG) - this means that after ATA nothing can be placed except T to obtain TAT k-mer

This process is repeated until we can find unique following k-mer for 3' and 5' ends of reads

After that you obtain set of super-reads - their number less than read number and they are longer (~2000 from short reads)