Computational Biology

DNA Sequencing

Department of Mathematics
Stockholm University

In a nutshell: Breakthrough Technologies

- Copying DNA:
  - Polymerase chain reaction (PCR)
- Sequencing DNA:
  - Sanger Sequencing [AKA 1st generation sequencing]
  - Next/2nd-generation sequencing (NGS) [AKA Massive parallel sequencing]
  - 3rd-generation [AKA long-read sequencing]

Polymerase chain reaction (PCR)

- used to copy DNA
- Invented by Kary Mullis (Nobel prize 1993)
- Input: a DNA "template" t to copy, primers, polymerase, bases A, C, G, T,

  Process: n "cycles" (see right)

  Output: roughly 2^n copies of t

Per cycle there are 3 phases:

1. Denaturate: 94-98 °C for 20–30 s
2. Anneal: 50-65 °C for 20–40 s
3. Extension: 75-80 °C

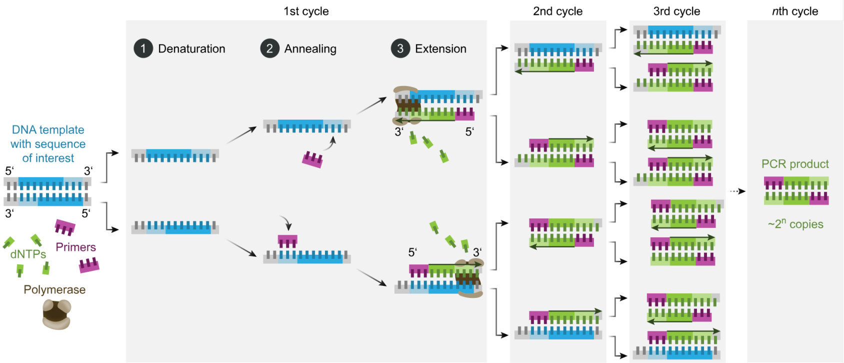


fig taken from `https://en.wikipedia.org/wiki/Polymerase_chain_reaction`

Sanger Sequencing

- used to read "small (500bp)" DNA sequences
- Invented by Fredrick Sanger and coworkers, 1977 (Nobel prize 1980)
- Input: copies of DNA split into 4 test tubes that contains primers, polmerase, bases, "modified bases A, C, T, G"

  Each tube contains all bases and ONE "modified base" $I \in \{A, C, G, T\}$

  Process (Basic Idea): "modified base" I ensures that when added during reading process of one DNA-copy, the reading process stops.

  Having multiple copies and the four tubes, this ensures, that (with high probability) the tupe I contains all single strands that end with I.

  gel electrophoresis: reads are negative charged and small reads get "closer" to positive pol (proportional to their length)

  Output: the read of the input DNA

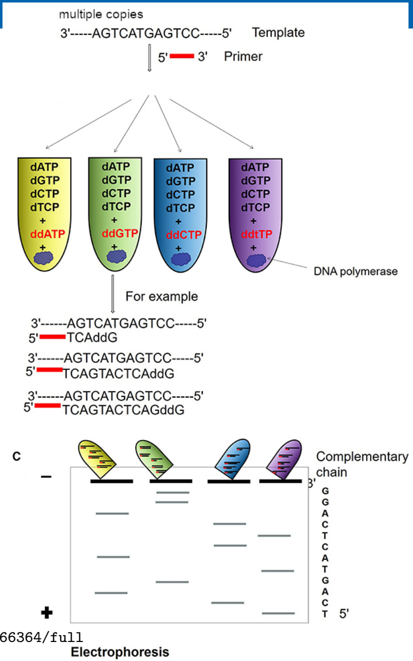


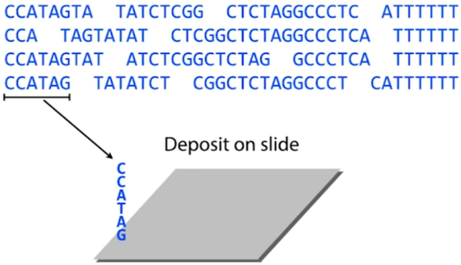
fig taken from `https://www.frontiersin.org/articles/10.3389/fmicb.2021.766364/full`

In a nutshell: Breakthrough Technologies

Next-generation sequencing (NGS)

- used to read **multiple** "small (500bp)" DNA sequences
- Several methods exits, one is the "Illumina sequencing process":

  Input: copies of **multiple** DNA (fragments) placed on a slide, bases, terminators, polymerase, ..


CCATAGTA TATCTCGG CTCTAGGCCCTC ATTTTTT
CCA TAGTATAT CTCGGCTCTAGGCCCTCA TTTTTT
CCATAGTAT ATCTCGGCTCTAG GCCCTCA TTTTTT
CCATAG TATATCT CGGCTCTAGGCCCT CATTTTTT
Deposit on slide
C C A T A G

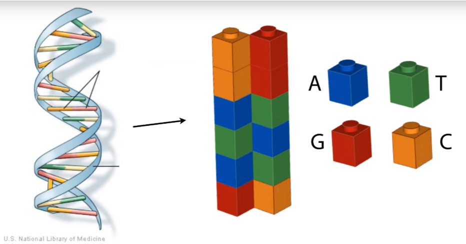
figs taken from Ben Langmead "ADS1: Sequencing by Synthesis" (youtube)

In a nutshell: Breakthrough Technologies

Next-generation sequencing (NGS)

- used to read **multiple** "small (500bp)" DNA sequences
- Several methods exits, one is the "Illumina sequencing process":

  Input: copies of **multiple** DNA (fragments) placed on a slide, bases, terminators, polymerase, ..



Double stranded DNA (double helix)

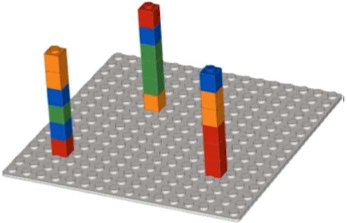
Double stranded DNA (lego version)

figs taken from Ben Langmead "ADS1: Sequencing by Synthesis" (youtube)

Next-generation sequencing (NGS)

- used to read **multiple** "small (500bp)" DNA sequences
- Several methods exits, one is the "Illumina sequencing process":

  Input: copies of **multiple** DNA (fragments) placed on a slide, bases, terminators, polymerase, ..



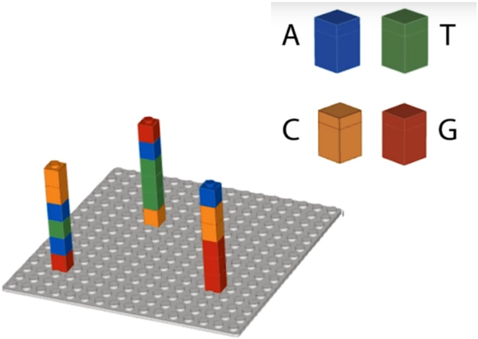
figs taken from Ben Langmead "ADS1: Sequencing by Synthesis" (youtube)

In a nutshell: Breakthrough Technologies

Next-generation sequencing (NGS)

- used to read **multiple** "small (500bp)" DNA sequences
- Several methods exits, one is the "Illumina sequencing process":

  Input: copies of **multiple** DNA (fragments) placed on a slide, bases, terminators, polymerase, ..



figs taken from Ben Langmead "ADS1: Sequencing by Synthesis" (youtube)

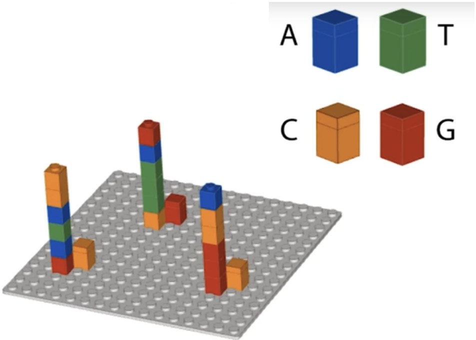
In a nutshell: Breakthrough Technologies

Next-generation sequencing (NGS)

- used to read **multiple** "small (500bp)" DNA sequences
- Several methods exits, one is the "Illumina sequencing process":

  Input: copies of **multiple** DNA (fragments) placed on a slide, bases, terminators, polymerase, ..

  Process (Basic Idea): when bases with terminators bind, no further base can be added.



figs taken from Ben Langmead "ADS1: Sequencing by Synthesis" (youtube)

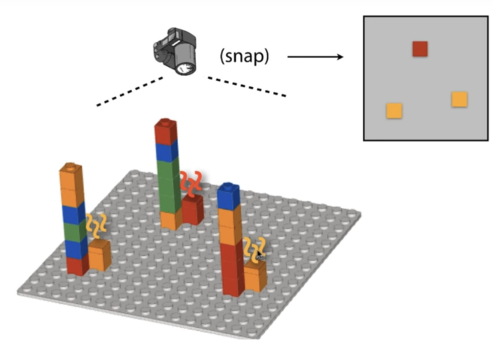
Next-generation sequencing (NGS)

- used to read **multiple** "small (500bp)" DNA sequences
- Several methods exits, one is the "Illumina sequencing process":

  Input: copies of **multiple** DNA (fragments) placed on a slide, bases, terminators, polymerase, ..

  Process (Basic Idea): when bases with terminators bind, no further base can be added.

  terminators are engineered to glow a particular color (A, C, G, T)
  → take photo



figs taken from Ben Langmead "ADS1: Sequencing by Synthesis" (youtube)

Next-generation sequencing (NGS)

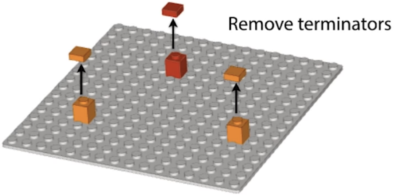
- used to read **multiple** "small (500bp)" DNA sequences
- Several methods exits, one is the "Illumina sequencing process":

  Input: copies of **multiple** DNA (fragments) placed on a slide, bases, terminators, polymerase, ..

  Process (Basic Idea): when bases with terminators bind, no further base can be added.

  terminators are engineered to glow a particular color (A, C, G, T)
  → take photo

  after taking photo, terminators are removed and process is repeated.



figs taken from Ben Langmead "ADS1: Sequencing by Synthesis" (youtube)

In a nutshell: Breakthrough Technologies

Next-generation sequencing (NGS)

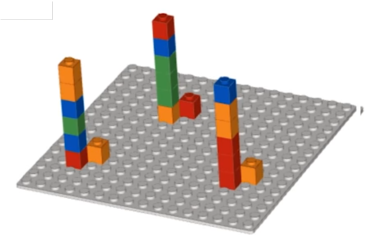
- used to read **multiple** "small (500bp)" DNA sequences
- Several methods exits, one is the "Illumina sequencing process":

  Input: copies of **multiple** DNA (fragments) placed on a slide, bases, terminators, polymerase, ..

  Process (Basic Idea): when bases with terminators bind, no further base can be added.

  terminators are engineered to glow a particular color (A, C, G, T)
  $\rightarrow$ take photo

  after taking photo, terminators are removed and process is repeated.



figs taken from Ben Langmead "ADS1: Sequencing by Synthesis" (youtube)

Next-generation sequencing (NGS)

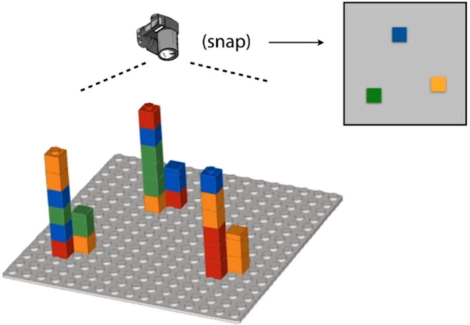
- used to read **multiple** "small (500bp)" DNA sequences
- Several methods exits, one is the "Illumina sequencing process":

  Input: copies of **multiple** DNA (fragments) placed on a slide, bases, terminators, polymerase, ..

  Process (Basic Idea): when bases with terminators bind, no further base can be added.

  terminators are engineered to glow a particular color (*A*, *C*, *G*, *T*)
  → take photo

  after taking photo, terminators are removed and process is repeated.



figs taken from Ben Langmead "ADS1: Sequencing by Synthesis" (youtube)

Next-generation sequencing (NGS)

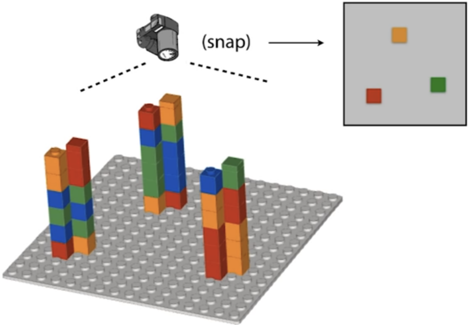
- used to read **multiple** "small (500bp)" DNA sequences
- Several methods exits, one is the "Illumina sequencing process":

  Input: copies of **multiple** DNA (fragments) placed on a slide, bases, terminators, polymerase, ..

  Process (Basic Idea): when bases with terminators bind, no further base can be added.

  terminators are engineered to glow a particular color (*A*, *C*, *G*, *T*)
  → take photo

  after taking photo, terminators are removed and process is repeated.



figs taken from Ben Langmead "ADS1: Sequencing by Synthesis" (youtube)

In a nutshell: Breakthrough Technologies

Next-generation sequencing (NGS)

- used to read **multiple** "small (500bp)" DNA sequences
- Several methods exits, one is the "Illumina sequencing process":

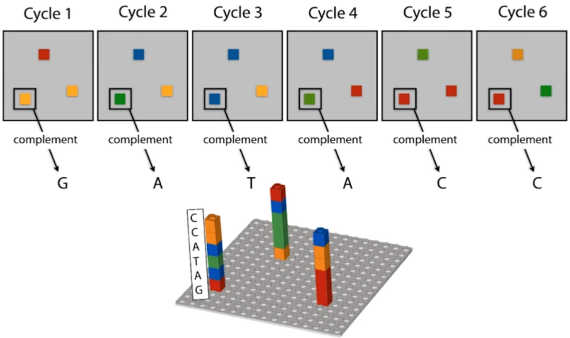
  Input: copies of **multiple** DNA (fragments) placed on a slide, bases, terminators, polymerase, ..

  Process (Basic Idea): when bases with terminators bind, no further base can be added.

  terminators are engineered to glow a particular color (A, C, G, T)
  $\rightarrow$ take photo

  after taking photo, terminators are removed and process is repeated.

  Output: the read of the **multiple** input DNAs (photos of each cycle)



figs taken from Ben Langmead "ADS1: Sequencing by Synthesis" (youtube)

In a nutshell: Breakthrough Technologies

Next-generation sequencing (NGS)

- used to read **multiple** "small (500bp)" DNA sequences
- Several methods exits, one is the "Illumina sequencing process":

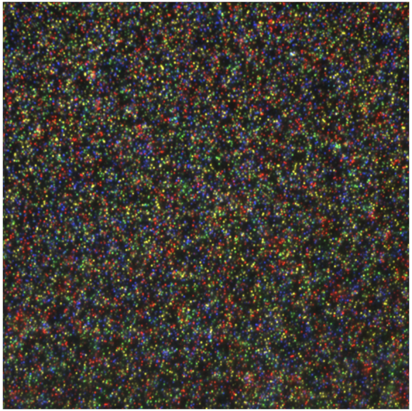
  Input: copies of **multiple** DNA (fragments) placed on a slide, bases, terminators, polymerase, ..

  Process (Basic Idea): when bases with terminators bind, no further base can be added.

  terminators are engineered to glow a particular color (A, C, G, T)
  $\rightarrow$ take photo

  after taking photo, terminators are removed and process is repeated.

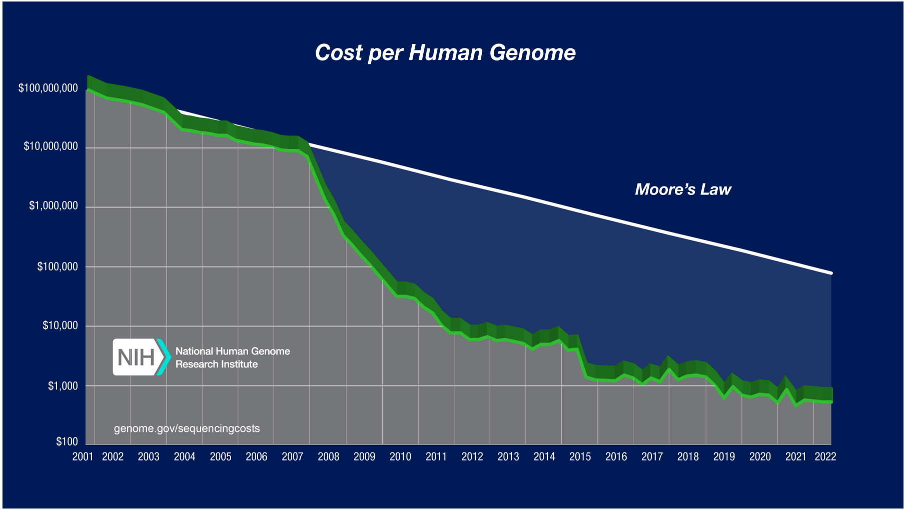
  Output: the read of the **multiple** input DNAs (photos of each cycle)



Key feature:
massively parallel, photograph captures all templates simultaneously (billions of DNA templates on a slide)

fig taken from `https://theory.labster.com/ngs-experiment/`

In a nutshell: Breakthrough Technologies



In a nutshell: Breakthrough Technologies

- Copying DNA:
  - Polymerase chain reaction (PCR)
- Sequencing DNA:
  - Sanger Sequencing [AKA 1st generation sequencing]
  - Next/2nd-generation sequencing (NGS) [AKA Massive parallel sequencing]
  - 3rd-generation [AKA long-read sequencing]
    (currently under active development*, can read more than 10000 bp)

To recall, humanDNA 3.2×10^9bp, Carsonella ruddii DNA 159 662bp

Observation: Whole genomes cannot be read at once.

*Marx, V. Method of the year: long-read sequencing. Nat Methods 20, 6–11 (2023).

Shotgun Sequencing

Observation:

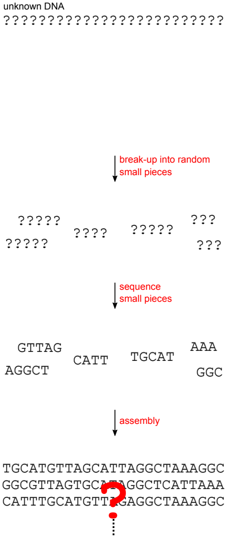
Sequencers cannot read whole genomes at once.

Idea_1: randomly break-up long DNA into multiple pieces

(e.g. with ultrasound)

and sequence them

However: if we just use a single DNA strand, well ..



unknown DNA

?????????????????????????

break-up into random small pieces

????? ???? ????? ???

????? ???

sequence small pieces

GTTAG CATT TGCAT AAA

AGGCT GGC

assembly

TGCATGTTAGCATTAGGCTAAAGGC

GGCGTTAGTGCATAGGCTCATTAAA

CATTTGCATGTTAGAGGCTAAAGGC

Shotgun Sequencing

Observation:

Sequencers cannot read whole genomes at once.

Idea_1: randomly break-up long DNA into multiple pieces

(e.g. with ultrasound)

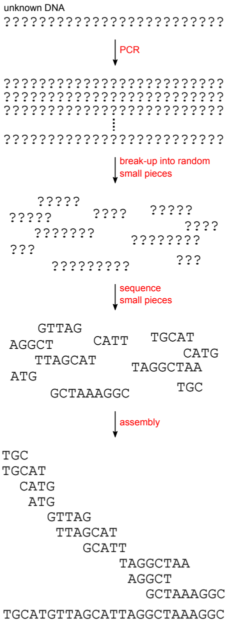
and sequence them

However: if we just use a single DNA strand, well ..

Idea_2: Produce multiple copies of DNA first and then apply Idea_1

⟹ results in overlapping reads

⟹ assemply (here smart computational methods are needed!)



unknown DNA

```
????????????????????????
```

PCR

```
????????????????????????
????????????????????????
????????????????????????
⋮
????????????????????????
```

break-up into random small pieces

```
   ?????
?????      ????   ?????
                      ????
   ???????      ????????
???
     ?????????       ???
```

sequence small pieces

```
   GTTAG
AGGCT      CATT   TGCAT
                      CATG
   TTAGCAT      TAGGCTAA
ATG
     GCTAAAGGC       TGC
```

assembly

```
TGC
TGCAT
  CATG
   ATG
     GTTAG
      TTAGCAT
         GCATT
             TAGGCTAA
              AGGCT
                GCTAAAGGC
TGCATGTTAGCATTAGGCTAAAGGC
```

Sequence Assembly

For a given set $\zeta = \{S_1, \dots S_N\}$ of strings (=reads of fragments of DNA D), a superstring is a string S that contains all S_i as substrings.

Trivially, we could concatenate all strings in ζ to get superstring S. However, having say $\sim 10^6$ copies of DNA D fragmented and sequenced, we get then a string S of length $|S| \sim |D| \times 10^6$ $\implies$ far away from D.

In the assembly problem, we want to find a superstring that "best represents" D.

There are several ways on how to define "best represents" !!

We start with considering following problem:

Shortest Common Superstring Problem (SCS):

For a given $\zeta = \{S_1, \dots S_N\}$ find a superstring S of shortest length.

SCS is NP-hard. So we focus ways to approximate solutions

$\implies$ overlap graphs and `Greedy_SCS` (board)

Basics:

string $S = x_1 \dots\dots x_m$, $|S| = m$

$S[i..j] = x_i x_{i+1} .. x_j$

$S(i) = x_i$

$S[1..j] \hat{=}$ prefix of S ending at j

$S[j..m] \hat{=}$ suffix of S starting at j

1 2 3 4 5 6 7 8

S: $\underbrace{\text{ATTC}}_{S[1..4]}\text{GT}\underbrace{\text{AC}}_{S[7..8]}$

$S[1..4]$ prefix

$S[7..8]$ suffix

$S(5) = L$

S' substring of $S = x_1 \dots x_m$, in symbols $S' \sqsubseteq S$

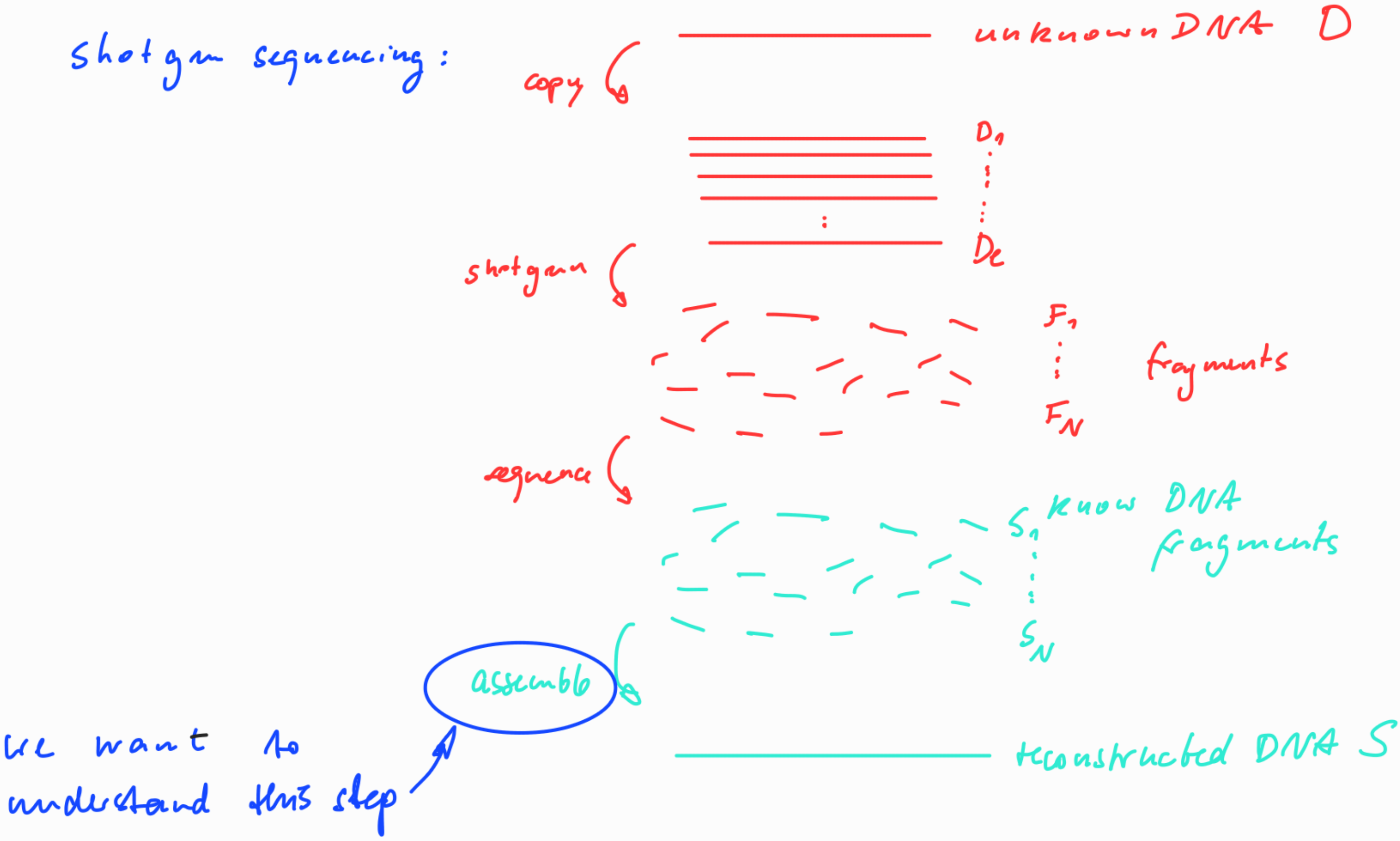
if $S' = x_i x_{i+1} \dots x_j$, $1 \leq i < j \leq m$

Shotgun-assembly

Formally: given DNA $D \rightarrow$ copy, i.e. we get set $\{D_1 \dots D_e\}$ of copies of D

→

shotgun sequencing:



Assembly Problem:

For given set $\mathcal{S} = \{s_1 \dots s_N\}$ of strings (substrings of D) find superstring S st $s_i \sqsubseteq S$, $1 \le i \le N$ and S "best represents" D.

There are several ways to define "best represents" here we focus on:

Shortest common superstring problem (SCS):

Given $\mathcal{S} = \{s_1 .. s_N\}$. Find S of min-length st $s_i \sqsubseteq S$, $1 \le i \le N$.

EXMPL: $\mathcal{S} = \{$ ATAT, TATA, TATT, TAAT, TTAT, AATA $\}$

// in this example, s_i of same length but this is not necessary for alg. //

→ shortest superstring S is

T A A T A T T A T A

Theorem (Gallant et al 1980): SCS is NP-hard

⇒ no hope for polynomial-time exact algorithm (P ≠ NP)

Q: Can we approximate a solution?

"substring-free"

In what follows: $\mathcal{S} = \{S_1 .. S_N\}$

st $S_i \neq S_j$ & $S_i \not\sqsubseteq S_j$ $\forall i \neq j$.

[If $S_i = S_j$ or $S_i \sqsubseteq S_j$, we can remove S_i in preprocess step]

has no impact on SCS!

we will see later,

this can be done in $O(N \cdot \|\mathcal{S}\|)$ time

$\|\mathcal{S}\| = \sum_{S \in \mathcal{S}} |S|$ via Suffix-trees

[Lit: Algorithmic Aspects of Bioinformatics, Böckenhauer & Bongartz, 2007, Springer]

DEF:

For all $S_i, S_j \in \mathcal{S} = \{S_1 .. S_N\}$ there is a longest substring v_{ij} st v_{ij} is suffix of S_i & prefix of S_j

that is
$$S_i = u\, v_{ij}$$
$$S_j = v_{ij}\, w$$
($v_{ij} = \varepsilon$ empty string is possible!)

v_{ij} called overlap of S_i & S_j

(! $v_{ij} \neq v_{ji}$ possible !)

$$ov(i,j) = |v_{ij}|$$
$$p(i,j) = |u| = |S_i| - ov(i,j)$$
$$merge(i,j) = u\, v_{ij}\, w$$
$$pref(i,j) = u$$

EXMPL

$S_1 = \underbrace{AT}_{u}\underline{AT}$, $S_2 = \underline{AT}\underbrace{TT}_{w}$

$v_{12} = AT$

$ov(1,2) = 2$

$p(1,2) = 2$

$merge(1,2) = u\, v_{12}\, w$
$= ATATTT$

$S_2 = \underbrace{ATTT}_{u}$ $S_1 = \underbrace{ATAT}_{w}$ $v_{21} = \varepsilon$ (empty)

$ov(2,1) = 0$

$p(2,1) = 4$ [$S_2 = ATTT = u\, v_{21}$]

$merge(2,1) = ATTTATAT$

DEF: graph $G_{\mathcal{S}} = (V, E)$ over $\mathcal{S} = \{S_1 \ldots S_N\}$:

- $V = \mathcal{S}$
- $E = (\mathcal{S} \times \mathcal{S}) \setminus \{(S,S) : S \in \mathcal{S}\}$

"no loops"

"prefix graph": arc $S_i \longrightarrow S_j$ gets weight $p(i,j)$

"overlap graph": —— " —— $ov(i,j)$.

[can be constructed in $O(n\,\|\mathcal{S}\|)$ time]
(↗ suffix trees later)

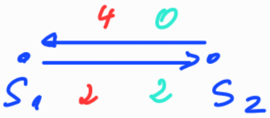
A path in $G_{\mathcal{S}}$ is a subgraph of the "form"

$$v_1 \to v_2 \to \ldots \to v_\ell$$

$$v_i \neq v_j \quad \forall\, i \neq j,$$

EXMPL: (1) $\mathcal{S} = \{S_1 = ATAT,\ S_2 = ATTT\}$

$G_{\mathcal{S}}$:



weight: p, ov

! $p(i,j) = |S_i| - ov(i,j)$!

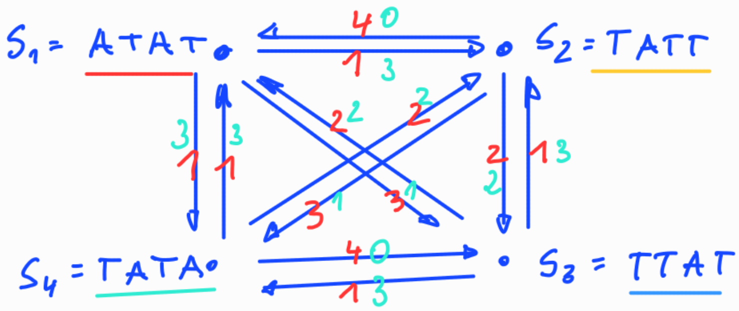
Observe min-length superstring S for S_1, S_2 is

$$S = \mathrlap{\underbrace{ATAT}_{S_1}}AT\underbrace{ATTT}_{S_2} \quad \hat{=} \quad S_1 \xrightarrow{2} S_2$$

i.e. arc where prefix-length min./
overlap max.

(2) $\mathcal{S} = \{$ ATAT , TATA , TATT , TTAT $\}$

$G_\mathcal{S}$: p (overlap : $ov(i,j) = |S_i| - p(i,j)$)



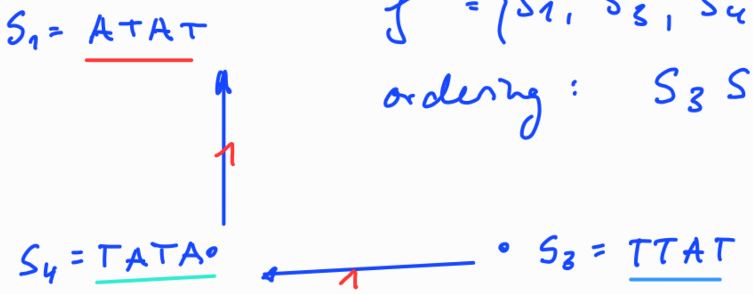
A T A T T A T A

1:1 correspondence between orderings of elements in $\mathcal{S}' \subseteq \mathcal{S}$ & paths in $G_\mathcal{S}$.

Exmpl:

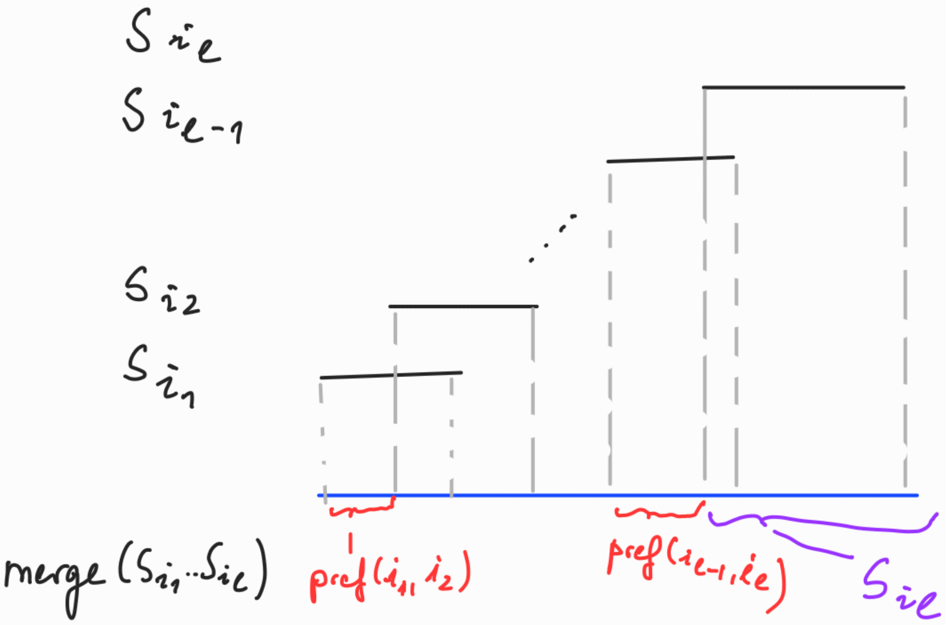
$\mathcal{S}' = \{S_1, S_3, S_4\}$

ordering : $S_3 \, S_4 \, S_1$



DEF given order $S_{i_1} .. S_{i_\ell}$ (path in G_S) define:

$$\text{merge}(S_{i_1} \dots S_{i_\ell}) := \text{pref}(i_1, i_2)\ \text{pref}(i_2 i_3) \dots \text{pref}(i_{\ell-1}, i_\ell)\ S_{i_\ell}$$



EXMPL (above) order S_3, S_4, S_1

$S_1 =$ T A T A

$S_4 =$ A T A T

$S_3 =$ T T A T

$$\text{merge}(S_3 S_4 S_1): \underbrace{T\ T}_{\text{pref}(3,2)}\ \underbrace{A}_{\text{pref}(4,1)}\ \underbrace{T\ A\ T\ A}_{S_1}$$

By def: $|merge(s_{i_1} .. s_{i_\ell})| = \sum_{i=1}^{\ell-1} p(i, i+1) + |s_{i_\ell}|$

Basic Idea:

If for s_i, s_j, Overlap $ov(s_i, s_j)$ is „large"
then s_i, s_j might also overlap in genom

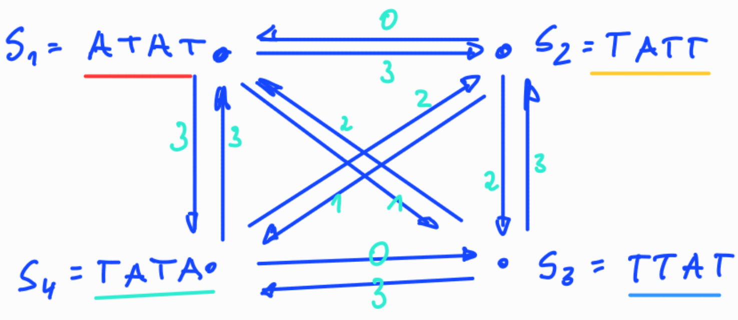
⇒ successively "merge" strings with large overlap.

```
GREEDY-SCS ($\mathcal{S}$)
    WHILE ($|\mathcal{S}| > 1$)
        s, t ← 2 strings in $\mathcal{S}$ with max ov(s,t)
                                                // s=t possible
        remove s,t from $\mathcal{S}$ & add uvw to $\mathcal{S}$
                                                // s=uv, t=vw
    3. return the single string $s \in \mathcal{S}$.
```

EXMPL:

$\mathcal{S} = \{ ATAT, TATA, TATT, TTAT \}$

$G_{\mathcal{S}}$: ov(i, j)



max overlap is 3 eg (S_1, S_4), $(S_4 S_1)$, $(S_2 S_4)$, $(S_3 S_2)$ ...

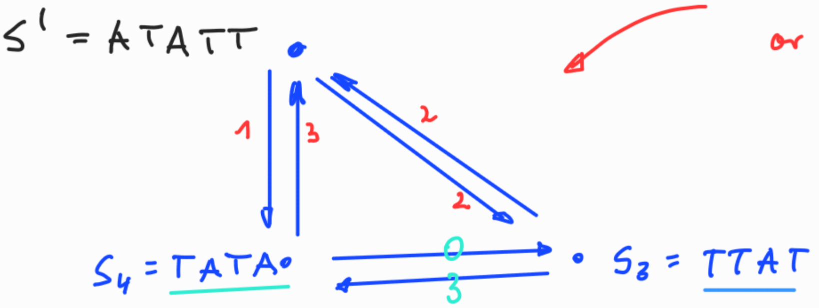
take one of them & merge.

choose here (S_1, S_2): $S_1 \neq S_2$

$$\Rightarrow \mathcal{S} = \{ \underset{\substack{merge(S_1 S_2) \\ \| \\ S'}}{ATATT}, \underset{S_3}{TTAT}, \underset{S_4}{TATA} \}$$

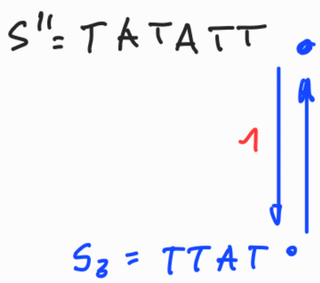
ov.(S', S_i) = ov(S_2, S_i)

ov(S_i, S') = ov(S_i, S_1)



max overlap: (S_4, S') → merge(S_4, S')

get $\mathcal{S} = (\underbrace{TATATT}_{S''}, \underset{S_3}{TTAT})$



ov(S'', S_3) = ov$(S_1 S_2)$

ov$(S_3 S'')$ = ov(S_3, S_4)

merge(S_3, S''): TTATATT

Lemma: GREEDY_SCS has runtime $O(|\mathcal{S}| \cdot \|\mathcal{S}\|)$

$\|\mathcal{S}\| = \sum_{s \in \mathcal{S}} |s|$

(Exercise, see also:
[Lit: Algorithmic Aspects of Bioinformatics, Böckenhauer & Bongartz 2007, Springer])

! important for the latter result is the following result, that shows that overlaps in the intermediate steps don't need to be recomputed!

Lemma IF in a step of Greedy-SCS we merge S' & S''

& $S' = merge(S_{i_1} .. S_{i_k})$ $k \geq 1$
$S'' = merge(S_{j_1} .. S_{j_l})$ $l \geq 1$
(prior to this step)

THEN

$ov(S', S'') = ov(i_k, j_1)$

Proof: we show first that after merging S' & S'' remains "substring-free" ie $\tilde{S} \subseteq S$ will never occur.

By assumption, before any merging, $\mathcal{S}$ is substring-free

Assume, for contradiction, that after some merging we have $\tilde{S} \subseteq S$.

Let S be the first element constructed st for some $\tilde{S} \in \mathcal{S}$: $\tilde{S} \subseteq S$

S is either (a) the original S (not part of any merging process so-far)

or (b) the result of merging S_1, S_2

(S_1, S_2 could also have been merged before)

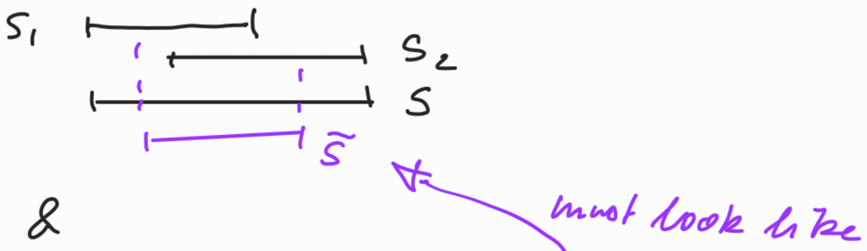
[Note $S \sqsubseteq \tilde{S}$ not possible in case (a) since then I not substring-free

resp. S not first element

& in case (b): $S_1, S_2 \sqsubseteq \tilde{S}$ & S not first element]

it looks as follows:

$\Rightarrow$ $S = \text{merge}(S_1\ S_2)$



Since $\tilde{S} \sqsubseteq S$ &

Since S "first" $\Rightarrow$ $\tilde{S} \not\sqsubseteq S_1, S_2$

But then
$$\left.\begin{array}{l} ov(S_1, \tilde{S}) \geq \\ ov(\tilde{S}, S_2) \geq \end{array}\right\} ov(S_1, S_2)$$

& at least for one „$\geq$" we have "$>$"

& so greedy would choose $\tilde{S}$ & S_i, $i \in \{1,2\}$

↯ greedy choice.

$\Rightarrow$ after each mergestep, I remains substring free.

Assume now, for contradiction, that after some step $N \geq 1$, $\mathcal{S}$ does not satisfy condition in the Lemma.

Put $\mathcal{S}_{N-1} = \mathcal{S}$ after applying $N-1$ steps of greedy.

By assumption in $\mathcal{S}_{N-1}$ everything ok!

W.l.o.g let $S', S'' \in \mathcal{S}_{N-1}$ st $ov(S', S'')$ = max overlap over all elements in $\mathcal{S}$

& greedy chooses S' & S'' to be merged.

$$\Rightarrow \quad \mathcal{S}_N = \mathcal{S}_{N-1} \setminus \{S', S''\} \cup \{\underbrace{merge(S', S'')}_{= S}\}$$

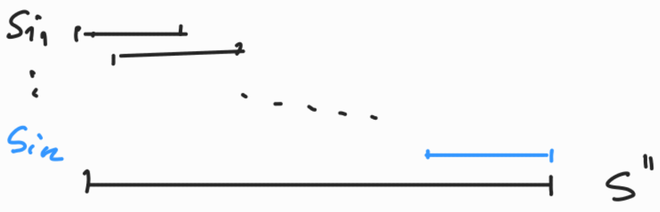
Note, $\mathcal{S}_N$ violates condition in lemma
& in $\mathcal{S}_N \setminus \{S\} \subseteq \mathcal{S}_{N-1}$ everything is ok

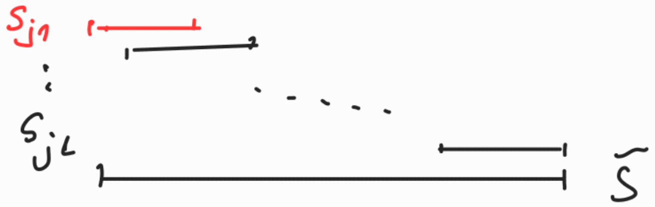
$\Rightarrow$ $\exists\, \tilde{S} \in \mathcal{S}_N \setminus S$ st S & $\tilde{S}$ violate condition in lemma.

Situation is as follows:

$$S = merge(S', S'')$$
$$S'' = merge(S_{i_1} .. S_{i_k}) \quad k \geq 1, \; S_{i_r} \in S_0$$
$$\tilde{S} = merge(S_{j_1} .. S_{j_\ell}) \quad \ell \geq 1, \; S_{j_r} \in S_0$$

& since condition in lemma violated we have
$$ov(S, \tilde{S}) \neq ov(S_{i_k}, S_{j_1})$$
[analog arguments for case $ov(\tilde{S}, S) \neq \dots$]





By definition : $s_{i_k} \sqsubseteq s = \text{merge}(s', s'')$

& $s_{j_1} \sqsubseteq \tilde{s}$

$\Rightarrow \text{ov}(s, \tilde{s}) \geq \text{ov}(s_{i_k}, s_{j_1})$

& since $\nearrow$ „≠"

$\Rightarrow \text{ov}(s, \tilde{s}) > \text{ov}(s_{i_k}, s_{j_1})$

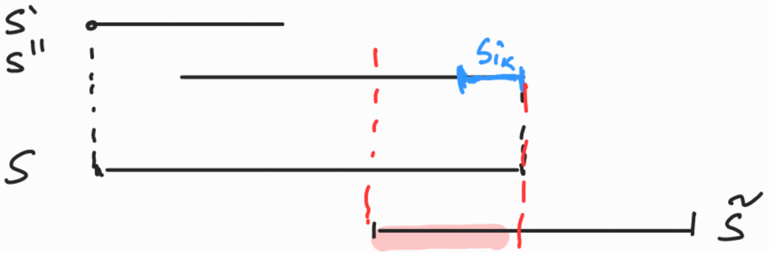
By Ind-assumption (all true in S_{N-1} & $\tilde{s}, s'' \in S_{N-1}$) we have:

$$\text{ov}(s'', \tilde{s}) = \text{ov}(s_{i_k}, s_{j_1}) < \text{ov}(s, \tilde{s})$$

looks like:

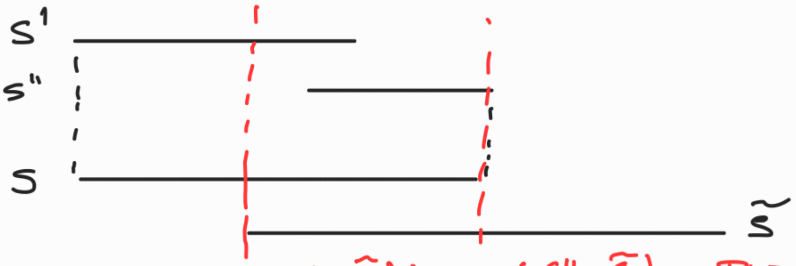
Not Possible!

$s = \text{merge}(s', s'')$:



$\text{ov}(s, \tilde{s}) = \text{ov}(s'', \tilde{s})$ ↯

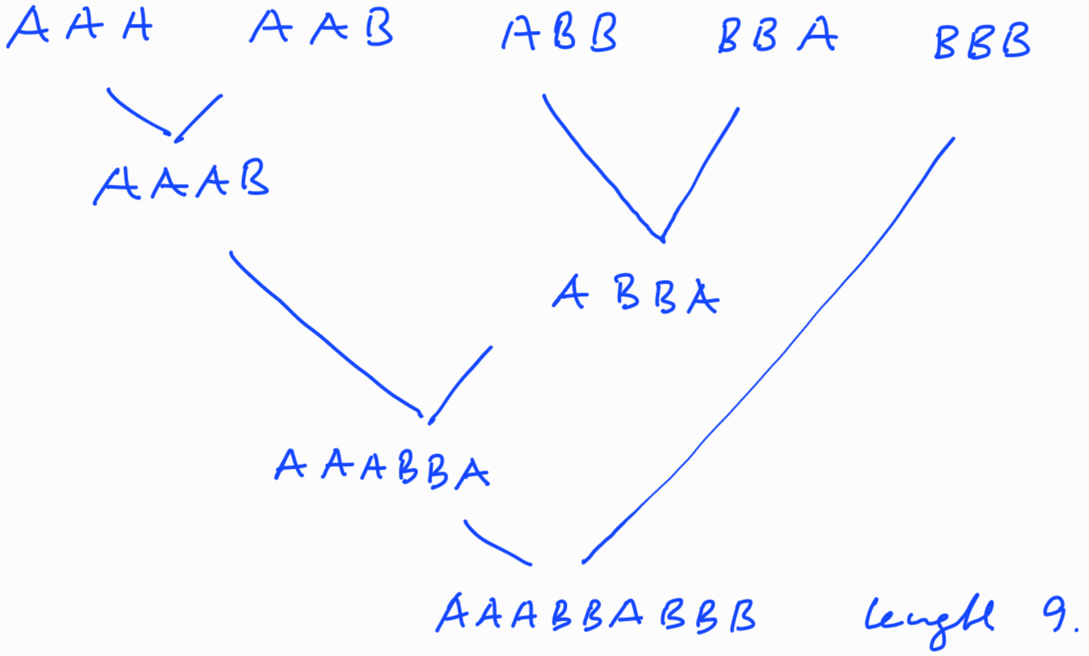
$\Rightarrow$ must look like:



$\text{ov}(s, \tilde{s}) > \text{ov}(s'', \tilde{s})$, BUT THEN: $s'' \sqsubseteq \tilde{s}$ ↯

□

In general, Greedy-SCS not optimal:



opt: AAABBBA length 7.

But we are never "too far" away from opt. solution:

__Theorem:__ String S returned by Greedy-SCS satisfies:

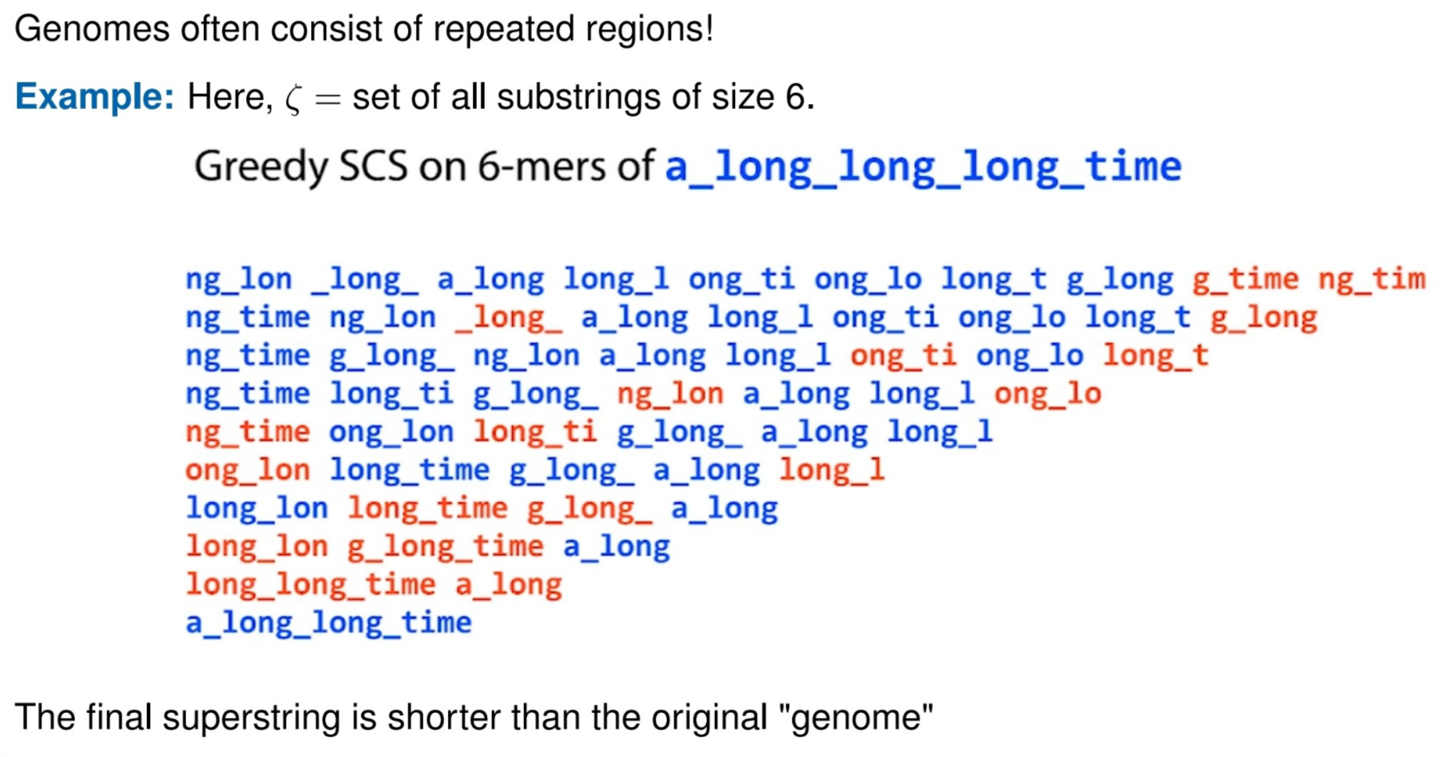
$$|S| \le 4\,opt(\mathcal{S})$$

ie, never worse than 4 times opt. solution.

[without proof]

One particular problem in practice: repeats.

In genomes often repeated regions.



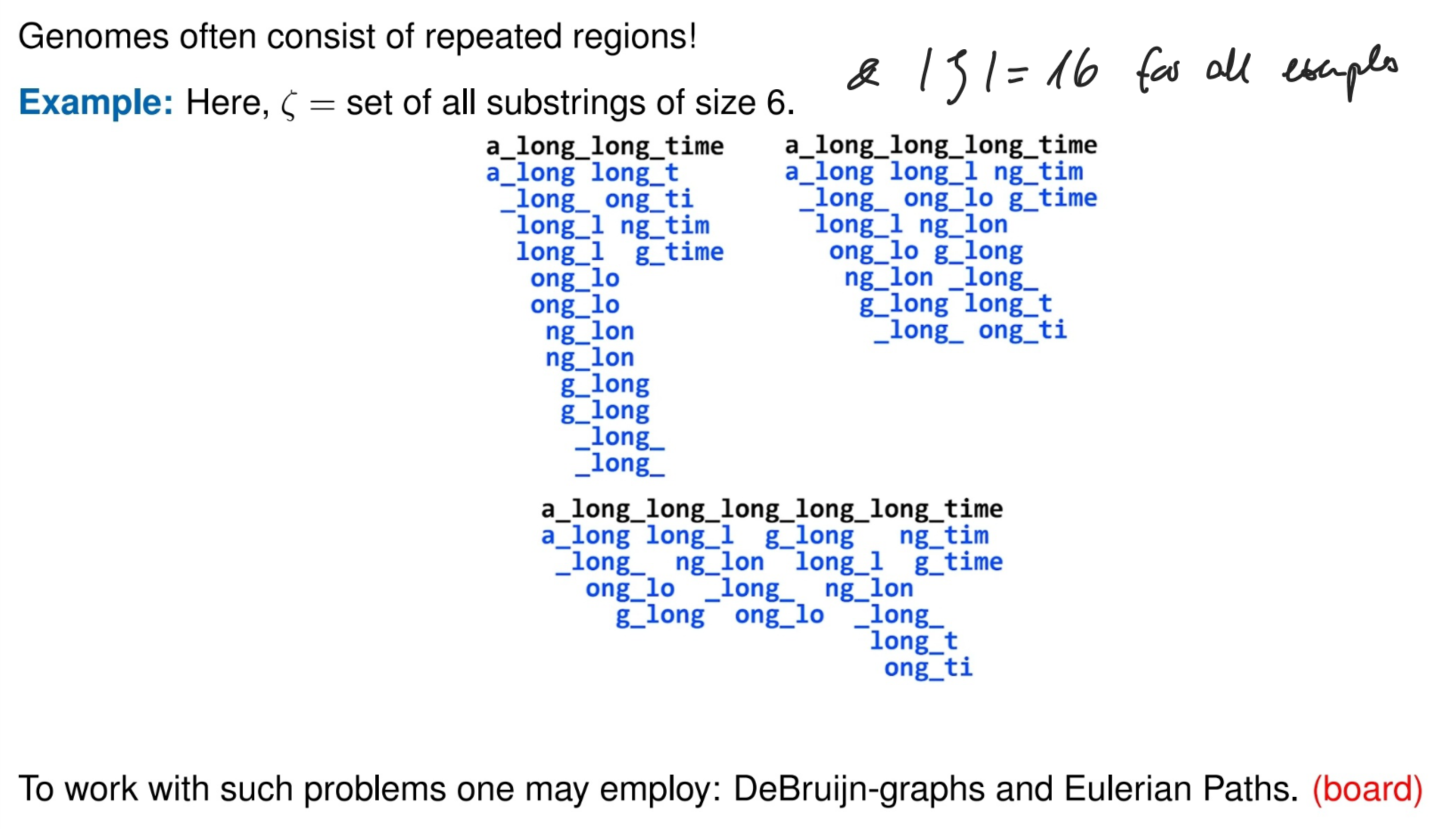
Genomes often consist of repeated regions!

Example: Here, $\zeta =$ set of all substrings of size 6.

Greedy SCS on 6-mers of **a_long_long_long_time**

```
ng_lon _long_ a_long long_l ong_ti ong_lo long_t g_long g_time ng_tim
ng_time ng_lon _long_ a_long long_l ong_ti ong_lo long_t g_long
ng_time g_long_ ng_lon a_long long_l ong_ti ong_lo long_t
ng_time long_ti g_long_ ng_lon a_long long_l ong_lo
ng_time ong_lon long_ti g_long_ a_long long_l
ong_lon long_time g_long_ a_long long_l
long_lon long_time g_long_ a_long
long_lon g_long_time a_long
long_long_time a_long
a_long_long_time
```

The final superstring is shorter than the original "genome"



Genomes often consist of repeated regions!

Example: Here, $\zeta =$ set of all substrings of size 6.

& $|\zeta| = 16$ for all examples

```
a_long_long_time
a_long long_t
 _long_ ong_ti
  long_l ng_tim
  long_l  g_time
   ong_lo
   ong_lo
    ng_lon
    ng_lon
     g_long
     g_long
      _long_
      _long_
```

```
a_long_long_long_time
a_long long_l ng_tim
 _long_ ong_lo g_time
  long_l ng_lon
   ong_lo g_long
    ng_lon _long_
     g_long long_t
      _long_ ong_ti
```

```
a_long_long_long_long_long_time
a_long long_l  g_long   ng_tim
 _long_  ng_lon  long_l  g_time
   ong_lo  _long_  ng_lon
     g_long  ong_lo  _long_
                      long_t
                       ong_ti
```

To work with such problems one may employ: DeBruijn-graphs and Eulerian Paths. (board) k-mer = substring of size k.

Given $\mathcal{S}$ = {sequenced fragments}

↓

$\mathcal{S}_k$ = all k-mers of sequences in $\mathcal{S}$.

Exmpl: (unknown) DNA AAABBBBA

→ $\mathcal{S}$ = {AAAA, AABB, BBA, ABBB}

→ $\mathcal{S}_3$ = {AAA, AAB, ABB, BBB, BBA}

DeBruijn-graph $\mathcal{G}_k$ = multigraph (ie. multiple edges allowed)

subdivide each k-mer $s_1 \dots s_k$ into left k-1 mer $s_1 \dots s_{k-1}$
right k-1-mer $s_2 \dots s_k$

these become the vertices in DeBruijn graph $\mathcal{G}_k$

arc (v, w) in $\mathcal{G}_k$ if v is L-k-1mer
w is R-k-1 mer
of some kmer
in $\mathcal{S}_k$.

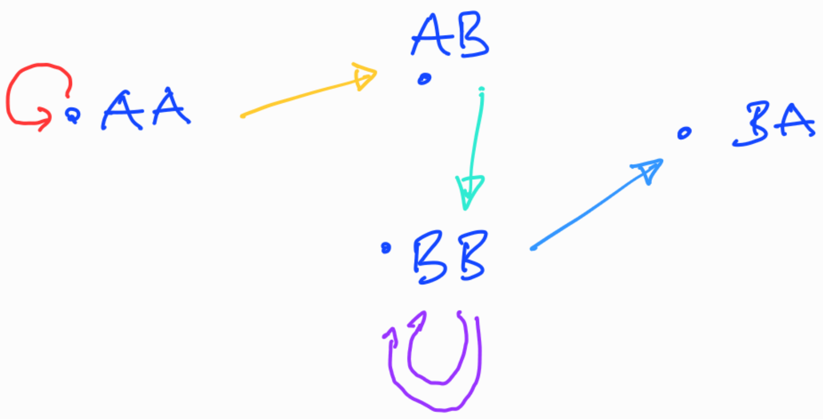
Exmpl: (unknown) DNA AAABBBBA

all 3mers: $S_3 = \{AAA, AAB, ABB, BBB, BBB, BBA\}$

can eg be obtained from reads of fragments.

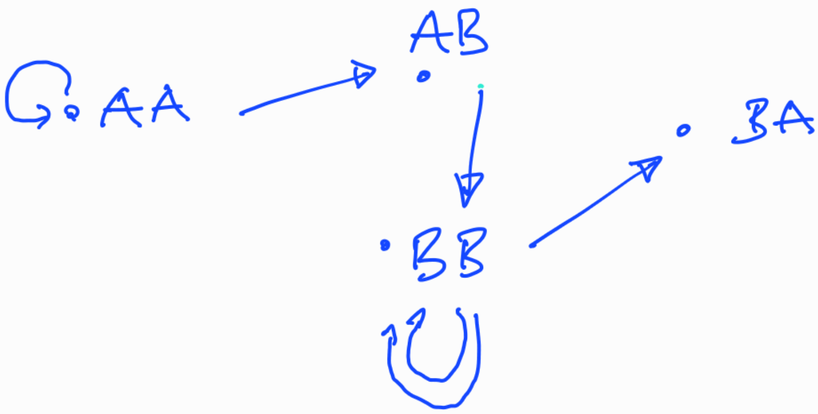
3-mers	AAA	→	L	k-1 mer	AA
			R	k-1 mer	AA
	AAB		L	-u-	AA
			R	-u-	AB
	ABB		L		AB
			R		BB
(2x)	BBB		L		BB
			R		BB
	BBA		L		BB
			R		BA

Vertices in S_3:

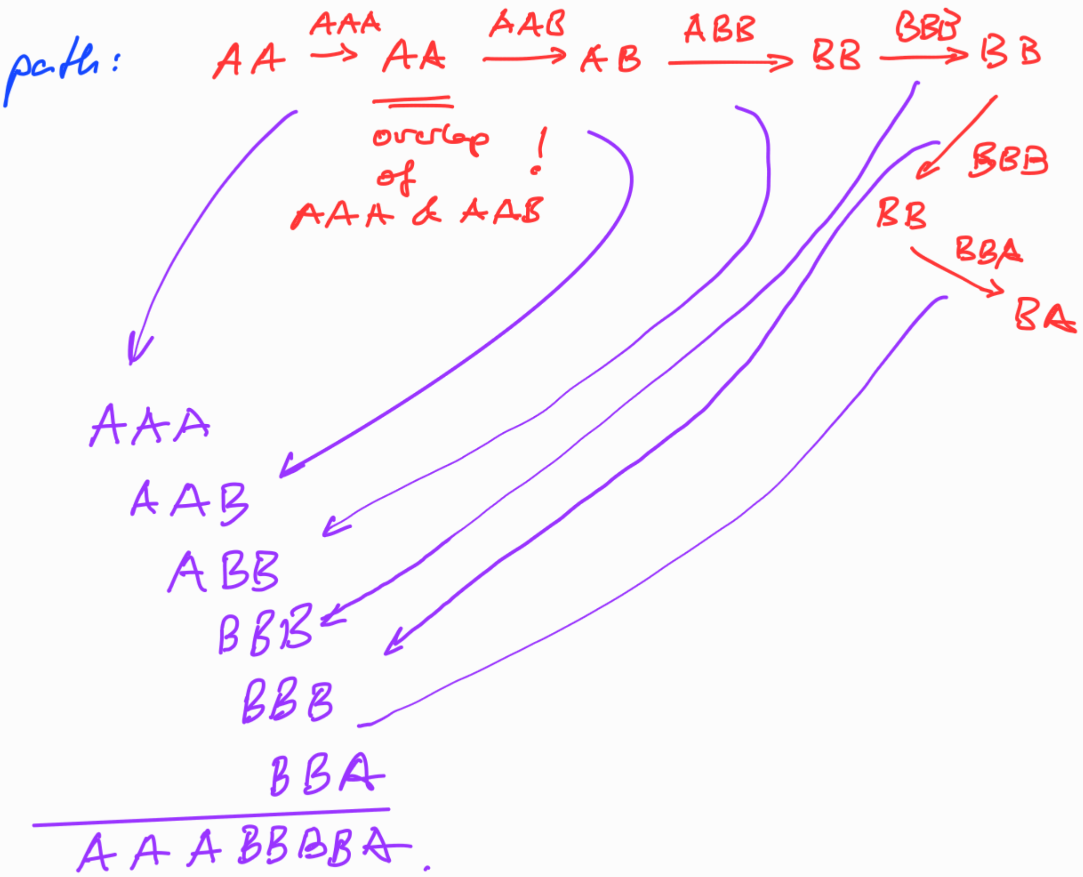


edges (implicitly) represent 3-mers $AB \xrightarrow{ABB} BB$

Eulerian path in a directed graph is a path that "visits" each edge exactly once.



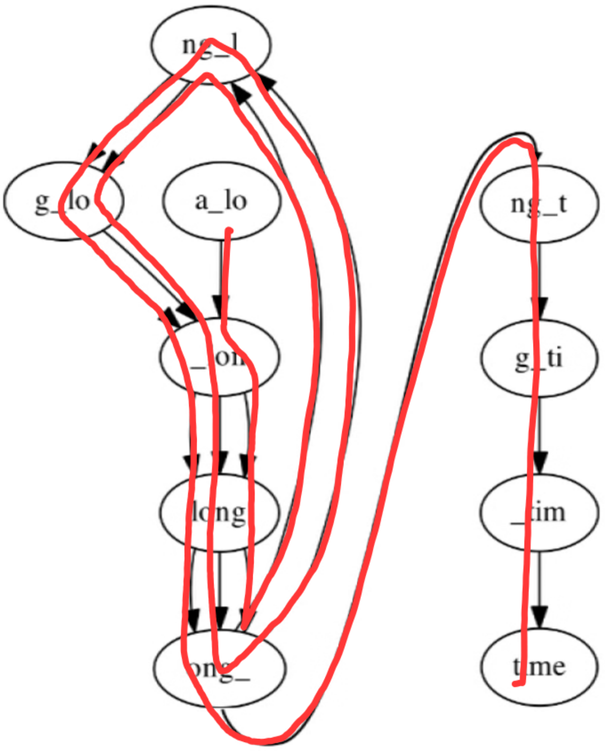
Eulerian path:



De Bruijn graph (*k*=5) for:

a_long_long_long_time

Eulerian walk gives original genome!



Eulerian walk.

(typical k ~ 30-50)

k = 8 Genome: a_long_long_long_time

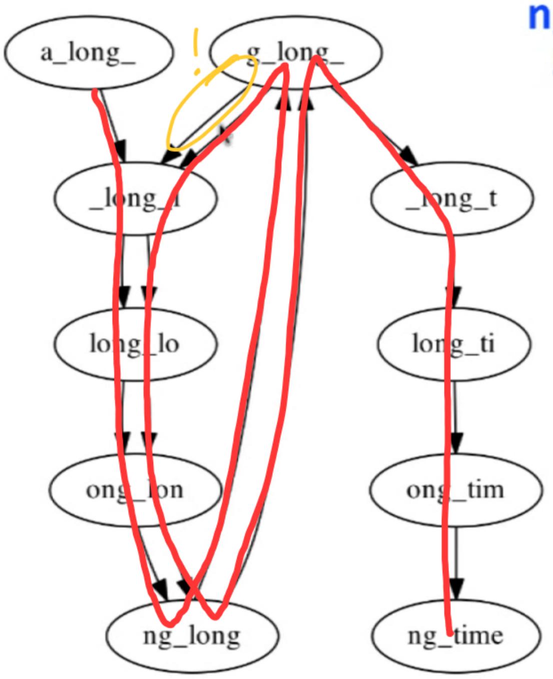
Reads: a_long_long_long, ng_long_l, g_long_time

k-mers:

a_long_l
_long_lo
long_lon
ong_long
ng_long_
g_long_l
_long_lo
long_lon
ong_long

ng_long_
g_long_l

g_long_t
_long_ti
long_tim
ong_time



No Eulerian walk!

Overlap graphs and DeBruijn graphs can be used to represent "relationships" between substrings.

The provided algorithms can, in general, not solve the assembly problem in an "optimal way" but serve as useful heuristics.

There are more sophisticated methods out there that are often based on these type pf algorithms.

[*] Medvedev & Pop *What do Eulerian and Hamiltonian cycles have to do with genome assembly?* PLoS Comput Biol. 2021

Classical problems in practice:

- sequencing errors
- overlapping regions of fragments that are located on "far away" positions on DNA
- incomplete data (DNA not covered by resulting sequenced fragments)
- orientation of reads usually unknown
- repeats