

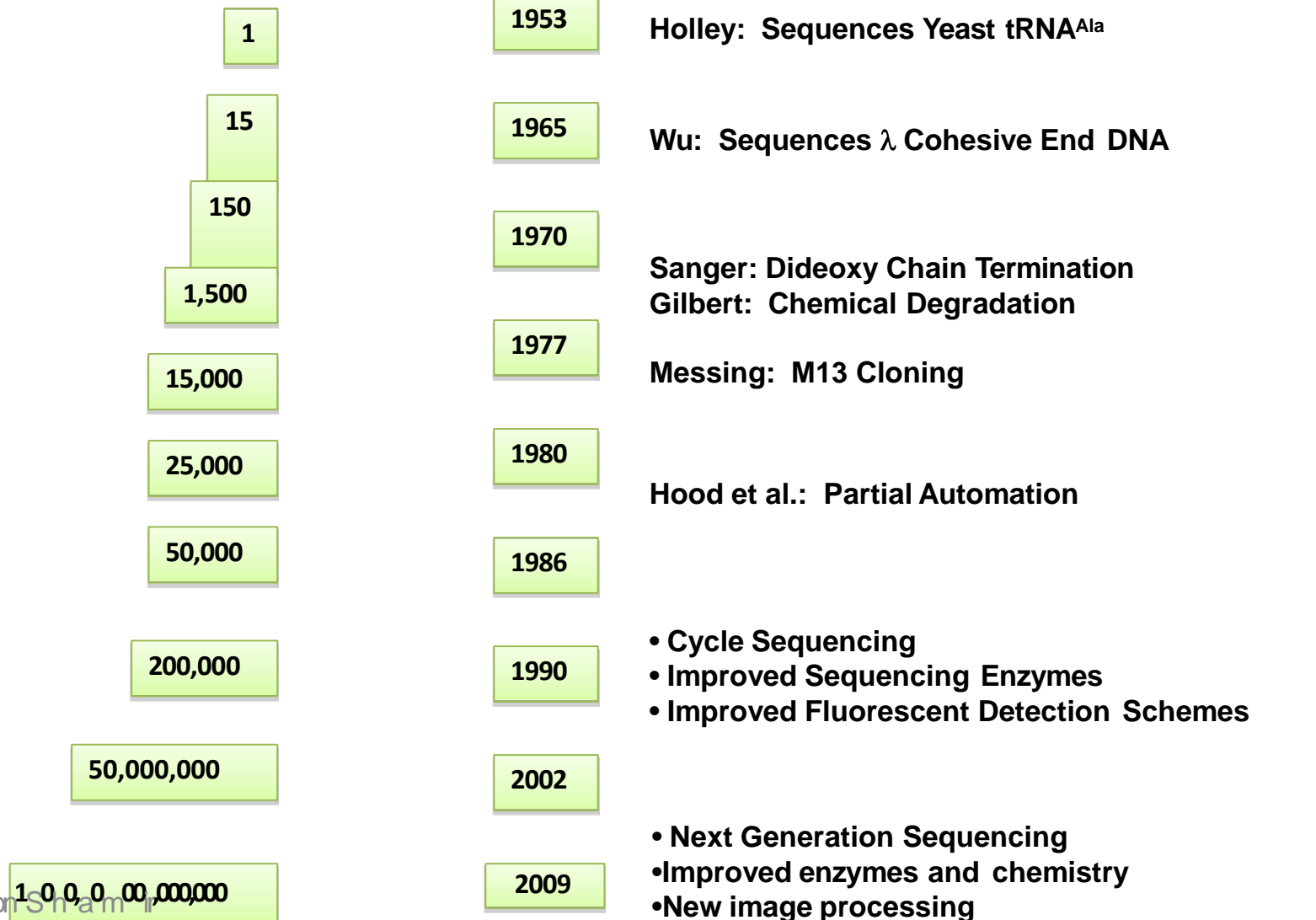
NGS algoritmai

# Planas

- Įvadas į gilaus sekvenavimo algoritmus
  - Skaičiai
  - Technologija
- Surinkimas
  - OLC: persidengimas (overlap), apjungimas(layout), consensus (consensus)
  - DGB: de Bruijn grafų algoritmas

# DNR sekoskaitos istorija

Efficiency  
(bp/person/year)



# Asmeninės genomikos era(2010 - ?)

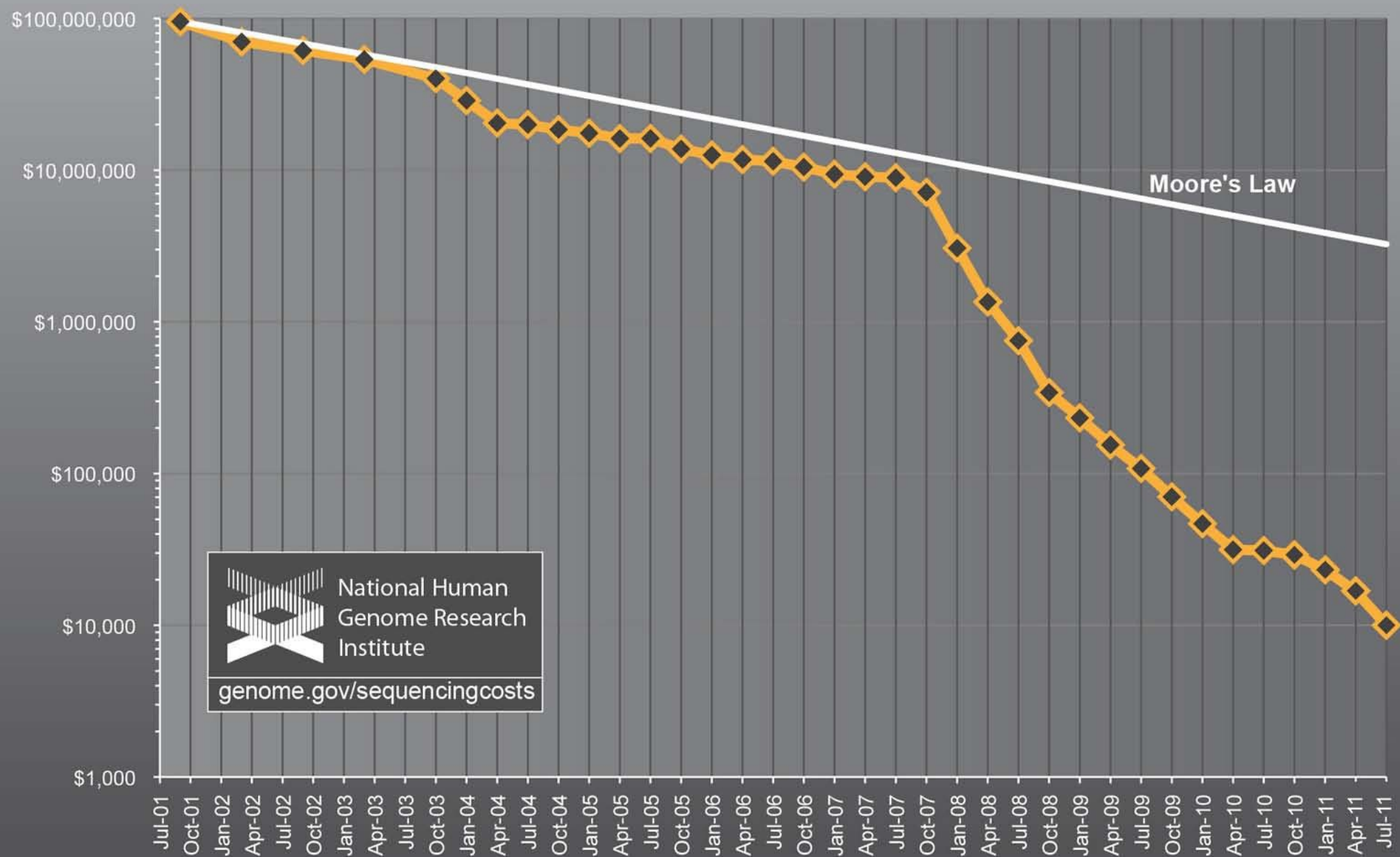
---

- 2003: Žmogaus genomo projektas baigtas – kaina **\$3,000,000,000**,
- 2003: J. Craig Venter fondas paskelbė \$500,000 prizą įmonei, kuri gebėtų nusekvenuoti žmogaus genomą už \$1000.
- 2005-2006: NIH paskelbia ~\$32M skirsianti grantams, sekvenavimo technologijų vystymui. X Prize fondas sukūrė Archon X Prizą genomikai: \$10M grupei, kuri gali sukurti aparatą galintį nusekvenuoti 100 žmonių genomų per 10 dienų su ne daugiau nei viena klaida per 100,000 bazių, už \$10,000 per per genomą..

**2010:** Illumina pasiūlo žmogaus genomo sekvenavimą už **\$10,000**.

**2014:** Illumina pasiūlo žmogaus genomo sekvenavimą už **\$1,000**. (30X perdengimas) naudojant HiSeq X sekvenatorių.

# Cost per Genome



# Kaip skaitoma DNR?

# Kaip skaitoma DNR?

- Padauginam

---

---

---

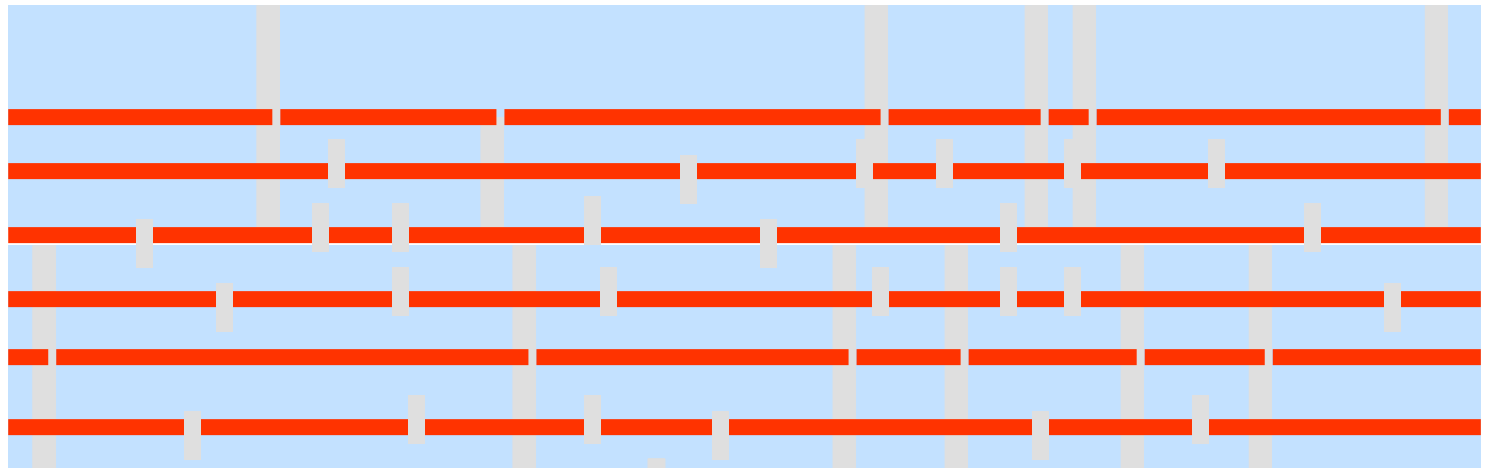
---

---

---

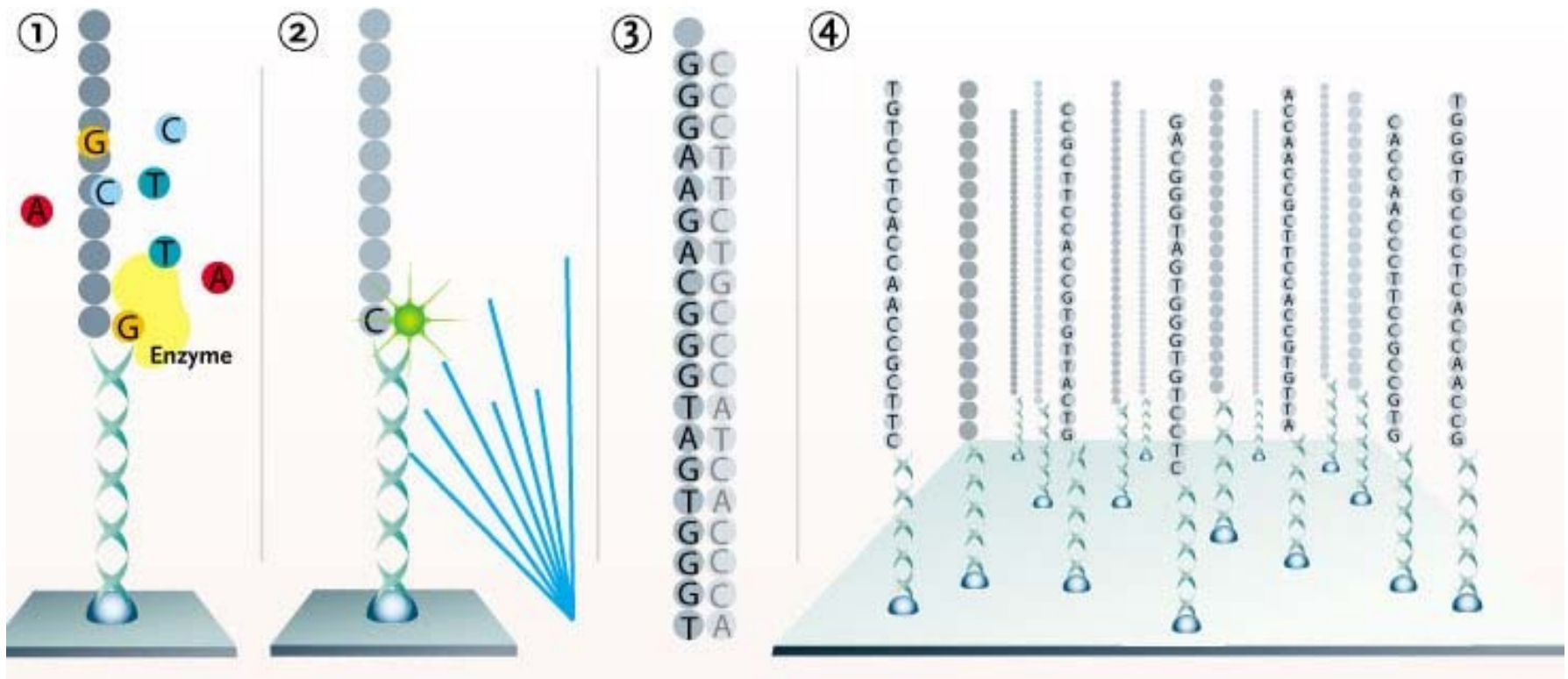
# Kaip skaitoma DNR?

- Pdauginam
- Fragmentuojam



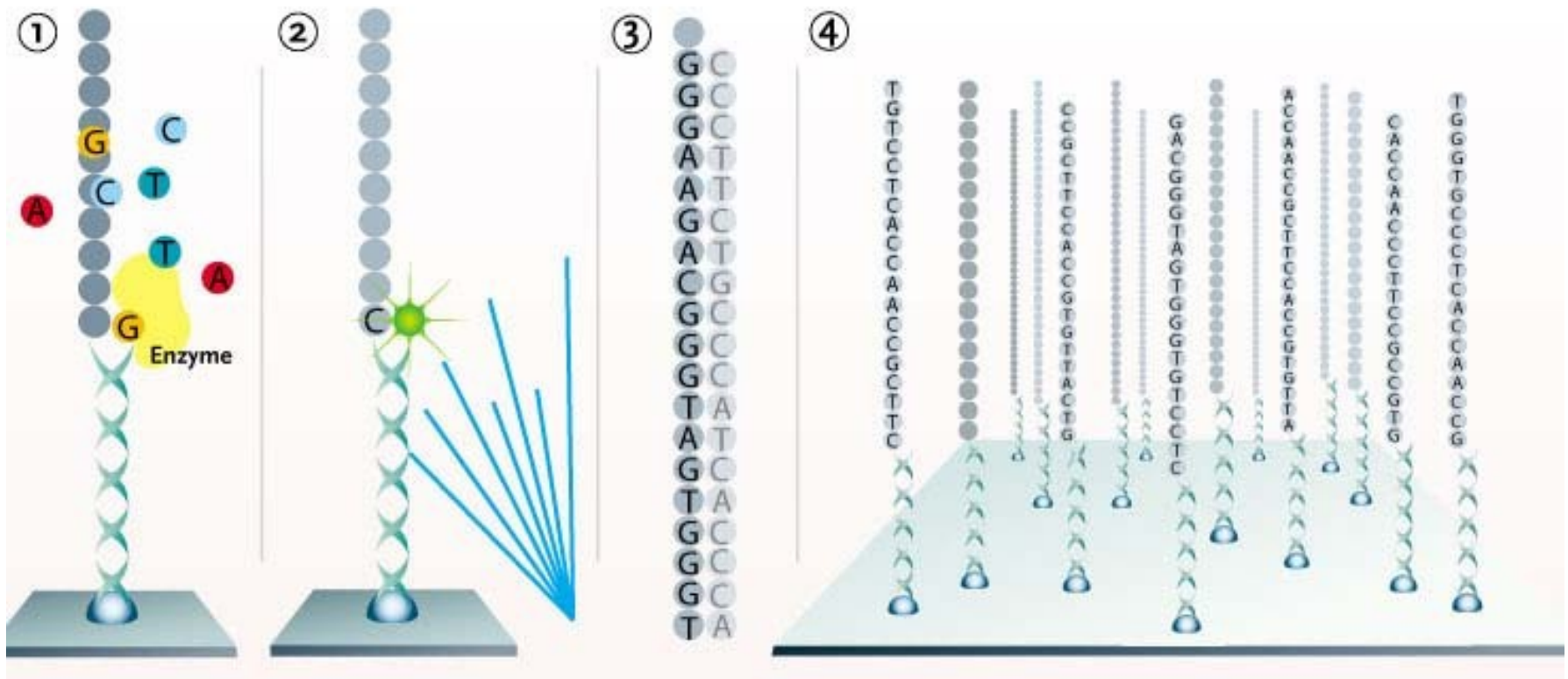


# Reading short DNA



- Use replication machinery with colored bases
- Take pictures of massively parallel reaction
- 10 million reads of 30 per day & \$1000

# Trumpas DNR sekvenavimas



- Naudojama sekvenavimo aparatas su spalvotai fluoresuojančiomis bazėmis.
- Gaunami trumpų fragmentų sekos.

# Pavyzdys

Genomas:

TTATGGTCGGTGAGTGTGACTGGTGTTGTCTAA

Nuskaitymai:

GGTCGGTGAG  
TGAGTGTGAC  
TGGTGTTGTC  
TGACTGGTTT  
AATGGTCGGT  
GAGTGTGACT  
AAAAAAAAAA



+



Seką iš fragmentų...nieko nežinant apie pradinę seką.



# Pavyzdys

Genomas:

TTATGGTCGGTGAGTGTGACTGGTGTTGTCTAA

|||||||

GGTCGGTGAG

Nuskaitymai:

GGTCGGTGAG

TGAGTGTGAC

TGGTGTTGTC

TGACTGGTTT

AATGGTCGGT

GAGTGTGACT

AAAAAAAAAA

# Pavyzdys

Genomas:

TTATGGTCGGTGAGTGTGACTGGTGTTGTCTAA

|||||||

TGAGTGTGAC

Nuskaitymai :

GGTCGGTGAG

TGAGTGTGAC

TGGTGTTGTC

TGACTGGTTT

AATGGTCGGT

GAGTGTGACT

AAAAAAAAAA



# Pavyzdys

Genomas:

TTATGGTCGGTGAGTGTGACTGGTGTTGTCCTAA

|||||||  
TGGTGTTGTC

Nuskaitymai :

GGTCGGTGAG

TGAGTGTGAC

TGGTGTTGTC

TGACTGGTTT

AATGGTCGGT

GAGTGTGACT

AAAAAAAAAA



# Pavyzdys

Genomas:

TTATGGTCGGTGAGTGTGACTGGTGTGTCTAA  
                          | | | | | | | |  
                          TGACTGGTTT

Nuskaitymai :

GGTCGGTGAG  
TGAGTGTGAC  
TGGTGTGTGTC  
TGACTGGTTT  
AATGGTCGGT  
GAGTGTGACT  
AAAAAAAAAA

# Pavyzdys

Genomas:

TTATGGTCGGTGAGTGTGACTGGTGTTGTCTAA

|||||||

AATGGTCGGT

Nuskaitymai :

GGTCGGTGAG

TGAGTGTGAC

TGGTGTTGTC

TGACTGGTTT

AATGGTCGGT

GAGTGTGACT

AAAAAAAAAA

# Pavyzdys

## Genomas:

TTATGGTCGGTGAGTGTGACTGGTGTTGTCTAA  
GGTCGGTGAG

TGAGTGTGAC

TGGTGTTGTC

TGACTGGTTT

AATGGTCGGT

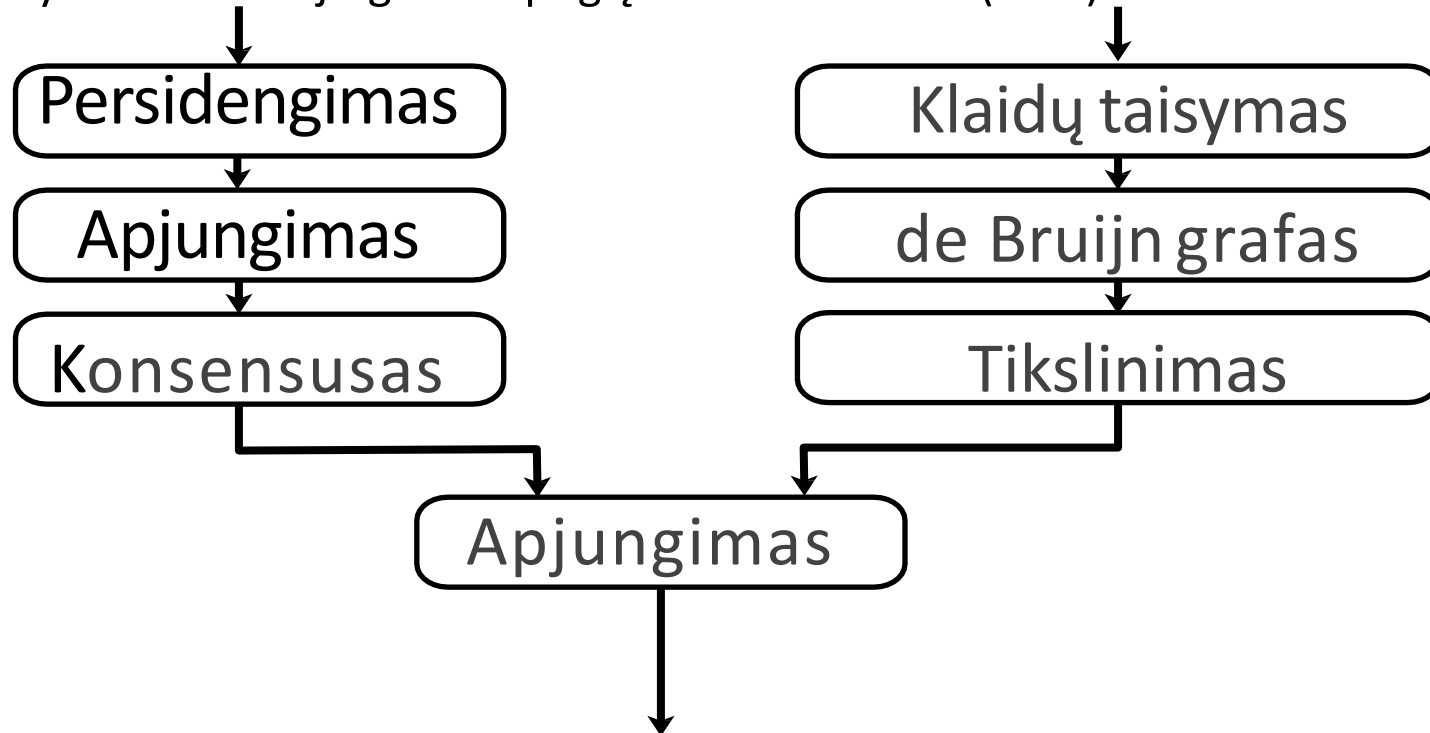
GAGTGTGACT

AAAAAAAAAA

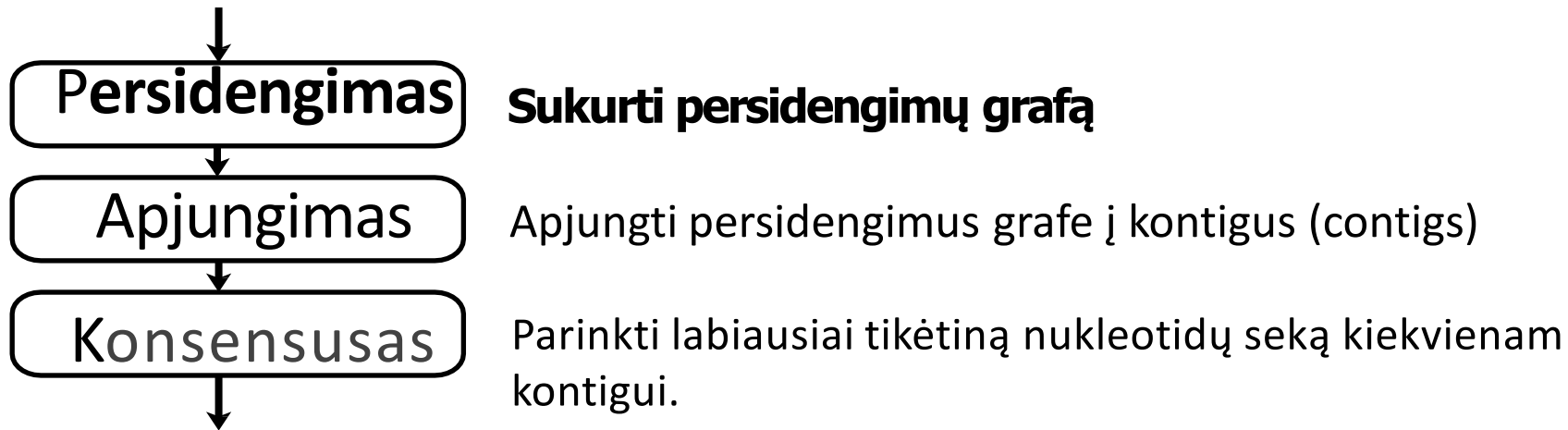
# Kaip iš fragmentų gauti vientisą seką?

Alternatyva 1: persidengimas-apjungimas-konsensusas surinkimas (Overlap-Layout-Consensus) (OLC).

Alternatyva 2: de Bruijn grafais pagrįstas surinkimas (DBG)



# Persidengimas-apjungimas-konsensusas



# Persidengimų radimas

Ar gali būti primityvesnis algoritmas?

Žodis  $l = 3$

ieškom jo  $Y$ , einant  
iš dešinės į kairę.

$X$ : CTCTAGGCC

$Y$ : TAGGCCCTC

$X$ : CTCTAGGCC

$Y$ : TAGGCCCTC

Rasta

Pratesiam į kairę. Šiuo atveju patvirtinam, kad 6-ių simbolių priešdėlis  $Y$  sekoje sutampa su priesaga  $X$  sekoje.

$X$ : CTCTAGGCC

$Y$ : TAGGCCCTC

Tai darom kiekvienai galimai nuskaitymų porai  
(nuskaitymų – milijonai)

# Persidengimų radimas

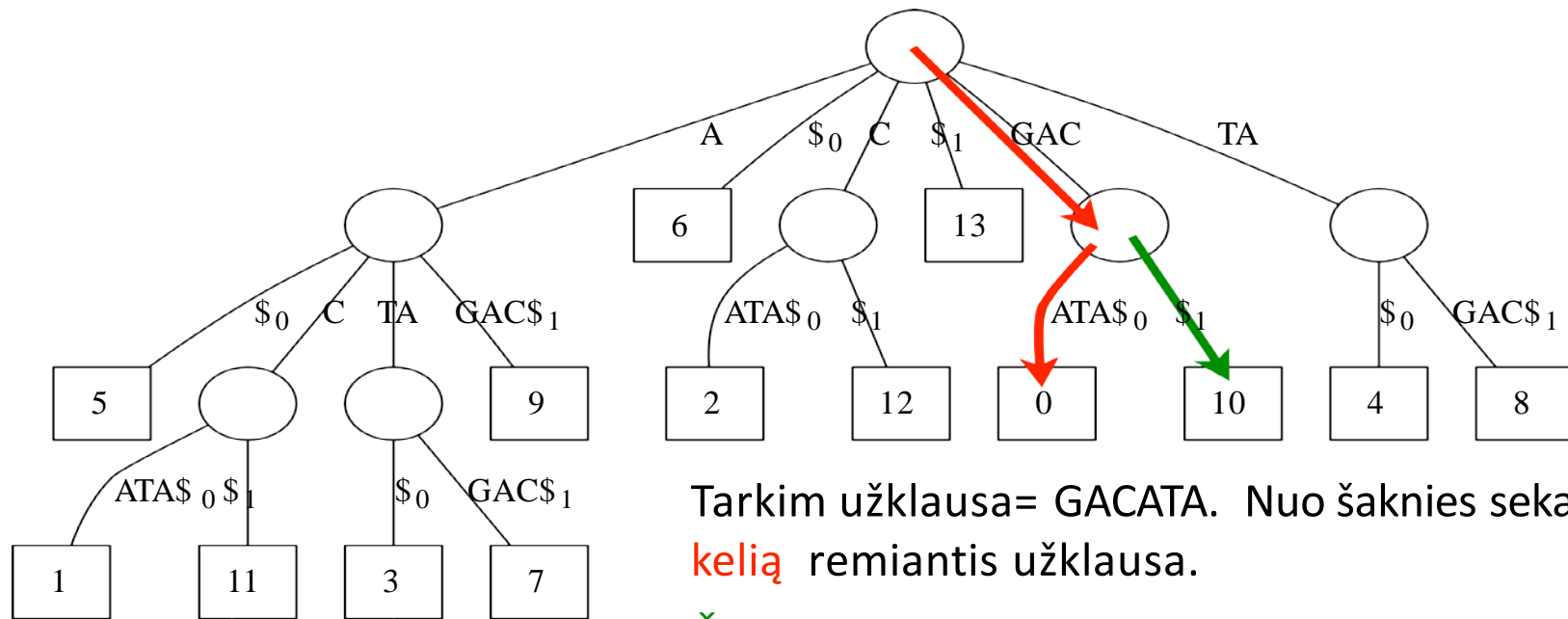
Peridengimams rasti galime panaudoti priesagų medį?

Problema: Duotas trumpų sekų rinkinys  $S$ . Kiekveinai sekai  $x$  esančiai  $S$  reikia rasti sutampančius fragmentus su kitomis sekomis, kurie turėtų priešdėlį iš  $x$  ir priesagą iš kitos sekos  $y$ .

Sprendimas: reikia sudaryti apibendrintą priesagų medį sekoms, esančioms  $S$ .

# Persidengimų radimas naudojant priesagų medį

Apibendrintas priesagų medis sekoms {“GACATA”, “ATAGAC”}    GACATA\$<sub>0</sub>ATAGAC\$<sub>1</sub>



ATAGAC  
|||  
GACATA

Tarkim užklausa= GACATA. Nuo šaknies sekame **kelią** remiantis užklausa.

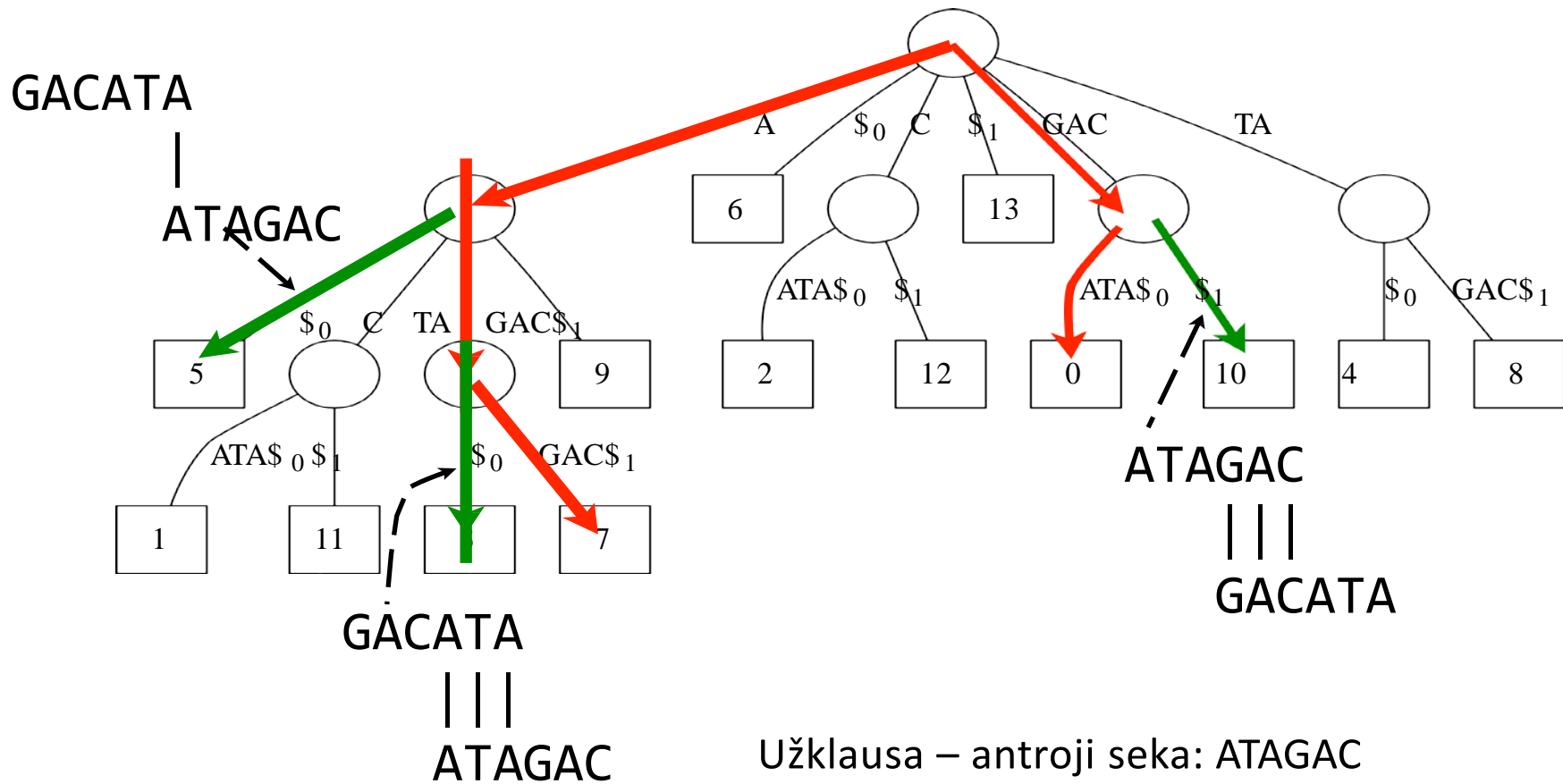
**Žalia atkarpa** rodo, kad trijų simbolių antros sekos priesaga sutampa su pirmos sekos trijų simbolių priešdėliu.

Persidengimų radimas  
naudojant priesagų medį

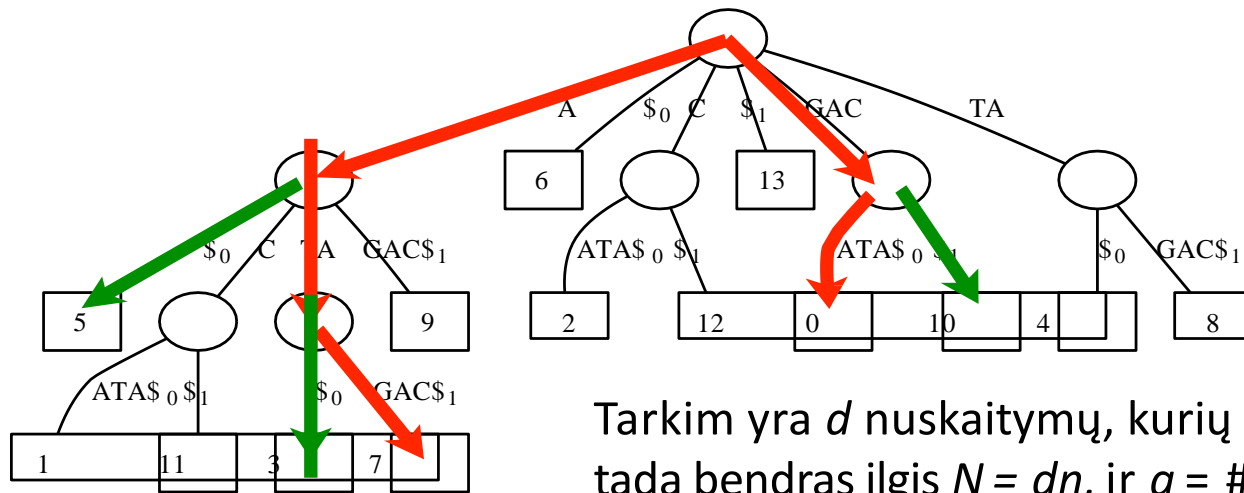


# Persidengimų radimas naudojant priesagų medį

Apibendrintas priesagų medis sekoms {“GACATA”, “ATAGAC”}    GACATA\$<sub>0</sub>ATAGAC\$<sub>1</sub>



# Persidengimų radimas naudojant priesagų medį



Tarkim yra  $d$  nuskaitymų, kurių kiekvienas yra  $n$  ilgio, tada bendras ilgis  $N = dn$ , ir  $a = \#$  persidengiančios nuskaitymų poros.

Tarkim vienai persidengimų porai ieškom tik ilgiausio persidengimo.

Laikas reikalingas gauti apibendrintą priesagų medį	$O(N)$	$d^2$ nėra žymimas, bet $a$ yra
... pereiti žemyn sekant nuskaitymo seka :	$O(N)$	$O(d^2)$ blogiausiu atveju.
... rasti ir persidengimus (žalia):	$O(a)$	
Bendrai:	$O(N + a)$	

# Persidengimų radimas

O kas jei noretume leisti persidengimuos e  
nepilnus sutapimus?

I.e. Kaip rastume geriausią palyginį tarp X priesagos  
ir Y priešdėlio?

Dinaminis programavimas...

X: CTCGGCCCTAGG

Y:     |||   |||  
     GGCTCTAGGCC

# Persidengimų radimas per dinaminį programavimą

Rasti geriausią priesagos iš X ir priešdėlio iš Y palyginį

X: CTCGGCCCTAGG

Y: GGCTCTAGGCC

Naudosime globalaus palyginio rekursiją ir sutapatinimo įverčių matricą  $s(a, b)$ .

$$D[i, j] = \min \begin{cases} D[i-1, j] + s(x[i-1], -) \\ D[i, j-1] + s(-, y[j-1]) \\ D[i-1, j-1] + s(x[i-1], y[j-1]) \end{cases}$$

	A	C	G	T	-
A	0	4	2	4	8
C	4	0	4	2	8
G	2	4	0	4	8
T	4	2	4	0	8
-	8	8	8	8	

Kaip “priversti” rasti prisegos/priešdėlio palyginimą, o ne per visą ilgį.

# Persidengimų radimas per dinaminį programavimą

Rasti geriausią priesagos iš X ir priešdėlio iš Y palyginį

$$D[i, j] = \min \begin{cases} D[i-1, j] + s(x[i-1], -) \\ D[i, j-1] + s(-, y[j-1]) \\ D[i-1, j-1] + s(x[i-1], y[j-1]) \end{cases}$$

$s(a, b)$

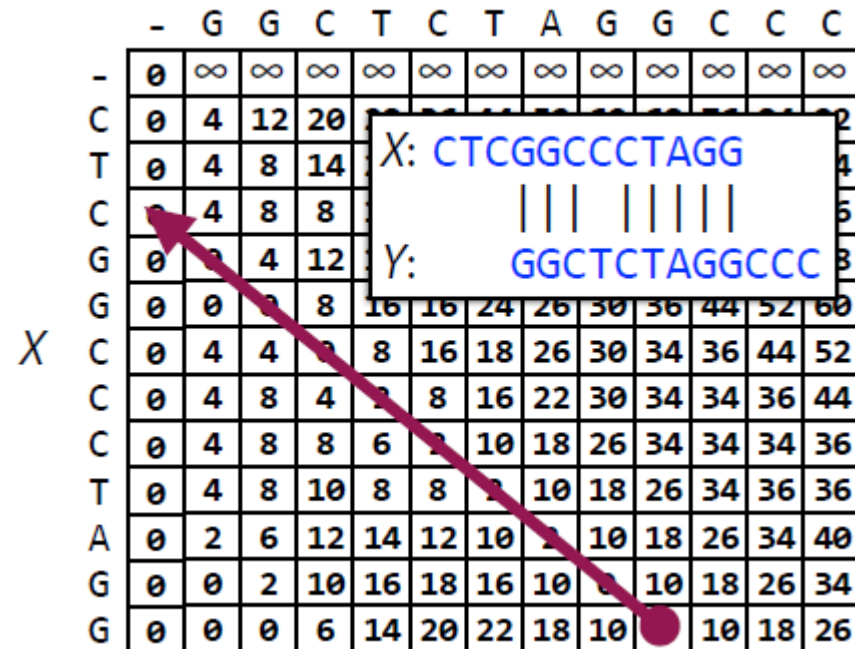
	A	C	G	T	-
A	0	4	2	4	8
C	4	0	4	2	8
G	2	4	0	4	8
T	4	2	4	0	8
-	8	8	8	8	8

Kaip inicijuoti pirmą stulpelį ir eilutę, kad X priesaga prisilygintų prie Y priešdėlio?

Pirmas stulpelis gauna 0-ius  
(bet kuri X priesaga *galima*)

Pirma eilutė gūna  $\infty$ -bes (turi būti Y  
priešdėlis)

Ieškom kelio nuo paskutinės eilutės.



# Persidengimų radimas per dinaminį programavimą

Rasti geriausią priesagos iš X ir priešdėlio iš Y palyginį

$$D[i, j] = \min \begin{cases} D[i-1, j] + s(x[i-1], -) \\ D[i, j-1] + s(-, y[j-1]) \end{cases}$$

$s(a, b)$

	A	C	G	T	-
A	0	4	2	4	8
C	4	0	4	2	8
G	2	4	0	4	8
T	4	2	4	0	8
-	8	8	8	8	8

Problema: **labai trumpi atitikmenys** gauna atsitiktinai didelis įverčius... ir gali trukdyti pastebėti **prasmingesnius sutapinimus**.

Tokiu atveju galim naudoti mažiausio persidengimo limitą (tarkim  $l = 5$ )

Y

	-	G	G	C	T	C	T	A	G	G	C	C	C
-	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
C	0	4	12	20	28	36	44	52	60	68	76	84	92
T	0	4	8	14	20	28	36	44	52	60	68	76	84
C	0	4	8	8	16	20	28	36	44	52	60	68	76
G	0	0	4	12	12	20	24	30	36	44	52	60	68
G	0	0	0	8	16	16	24	26	30	36	44	52	60
C	0	4	4	0	8	16	18	26	30	34	36	44	52
C	0	4	8	4	2	8	16	22	30	34	34	36	44
C	0	4	8	8	6	2	10	18	26	34	34	34	36
T	0	4	8	10	8	8	2	10	18	26	34	36	36
A	0	2	6	12	14	12	10	2	10	18	26	34	40
G	0	0	2	10	16	18	16	10	0	10	18	26	34
G	0	0	0	6	14	20	22	18	10	2	10	18	26

X

# Persidengimų radimas per dinaminį programavimą

Rasti geriausią priesagos iš X ir priešdėlio iš Y palyginį

$$D[i, j] = \min \begin{cases} D[i-1, j] + s(x[i-1], -) \\ D[i, j-1] + s(-, y[j-1]) \\ D[i-1, j-1] + s(x[i-1], y[j-1]) \end{cases}$$

$s(a, b)$

	A	C	G	T	-
A	0	4	2	4	8
C	4	0	4	2	8
G	2	4	0	4	8
T	4	2	4	0	8
-	8	8	8	8	8

Y

Sprendima – dar pildomas  
iniciacijos reksmes prilyginam  
 $\infty$ .

Langeliai, kurių vertės pakistų  
parodyti **raudonai**.

Dbar atitinkamas mažausio  
įverčio kelias link **mažiausios  
vertės** yra prasmingas.

	-	G	G	C	T	C	T	A	G	G	C	C	C
-	0	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
CT	0	4	12	20	28	36	44	52	60	68	76	84	92
CG	0	4	8	14	20	28	36	44	52	60	68	76	84
GC	0	4	8	8	16	20	28	36	44	52	60	68	76
CC	0	4	8	8	16	20	28	36	44	52	60	68	76
TA	0	0	4	12	12	20	24	30	36	44	52	60	68
GG	0	0	0	8	16	16	24	26	30	36	44	52	60
	0	4	4	0	8	16	18	26	30	34	36	44	52
	0	4	8	4	2	8	16	22	30	34	34	36	44
	0	4	8	8	6	2	10	18	26	34	34	34	36
	$\infty$	4	8	10	8	8	2	10	18	26	34	36	36
	$\infty$	12	6	12	14	12	10	2	10	18	26	34	40
	$\infty$	20	12	10	16	18	16	10	0	10	18	26	34
	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	20	22	18	10	2	10	18	26

# Persidengimų radimas per dinaminį programavimą

Tarkim yra  $d$  nuskaitymų, kurių kiekvienas yra  $n$  ilgio, bendras ilgis  $N = dn$ , ir  $a$  yra bendras skaičius porų su persidengimais

Persidengimų, kuriuos reikia įvertinti, skaičius  $O(d^2)$

Dinaminio programavimo matricos dydis:  $O(n^2)$

Bendrai:  $O(d^2n^2) = O(N^2)$

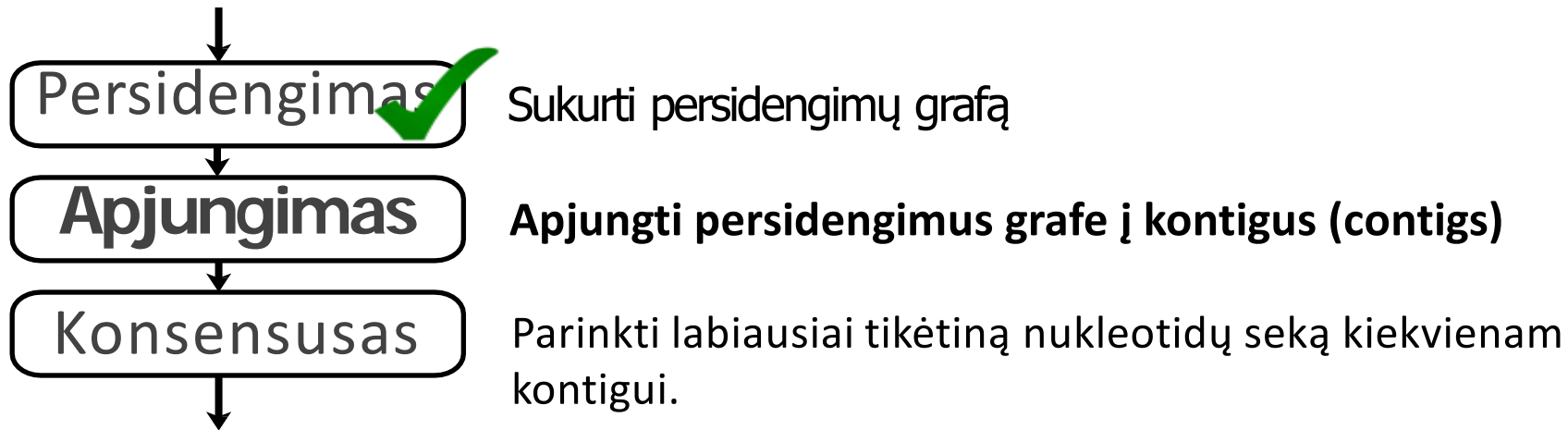
Palyginkit,  $O(N^2)$  su priesagų medžiu, kurio atveju:  $O(N + a)$ , ir kur  $a$ , blogiausiu atveju  $O(d^2)$

Bet dinaminis programavimas daug ankstesnis – leidžia nesutapimus ir tarpus.

Realiam gyvenime abu metodai ieškant persidengimų naudojami kartu. Iš pradžių atfiltruojami nepersidengiančios poros ir idealūs persidengimai. Dinaminis programavimas taikomas mažai daliai likusių duomenų.



# Persidengimas-apjungimas-konsensusas



# Apjungimas

Persidengimo grafas yra didelis ir komplikuotas. Kontigai akivaizdžiai nepasirodo...

Žemiau: dalis persidengimo grafo sekai:

to\_every\_thing\_turn\_turn\_turn\_there\_is\_a\_season

$l = 4, k = 7$

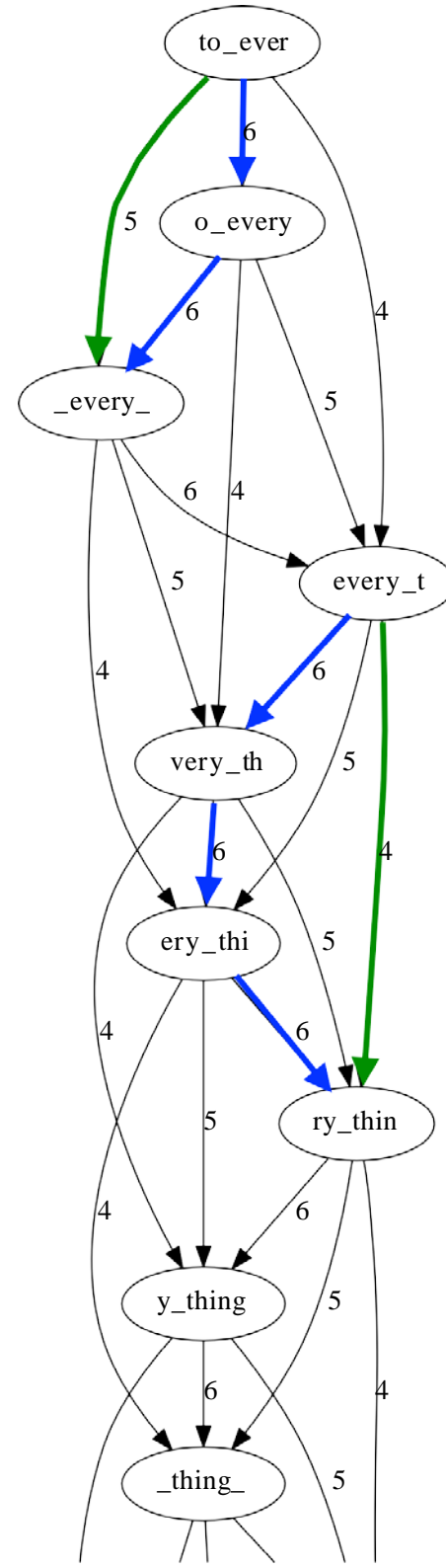


# Apjungimas

Kas yra perteklinio šio grafe?

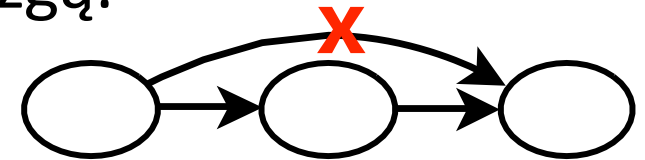
Kai kurie mazgai gali būti numatyti iš kitų mazgų.

Pvz. **žalias** mazgas gali būti numatytas iš **mėlyno**.

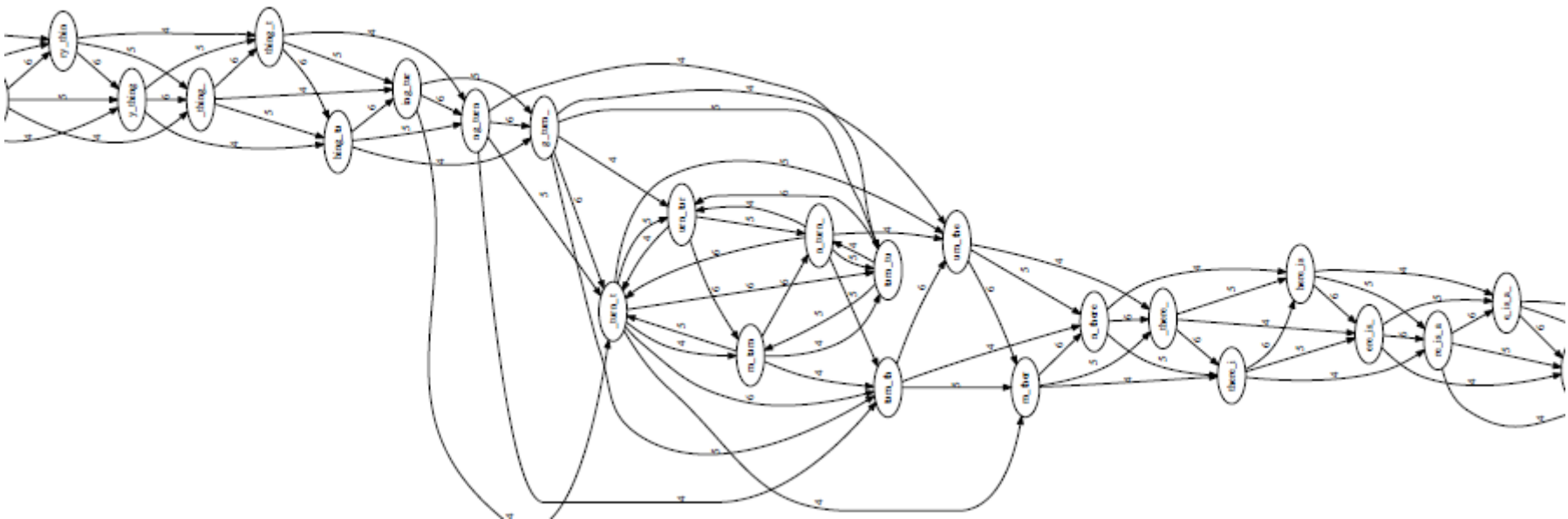


# Apjungimas

Pašalinamos jungtys, kurias galime numanyti iš kitų, pradedant nuo jungčių, kurios praleidžia vieną mazgą.

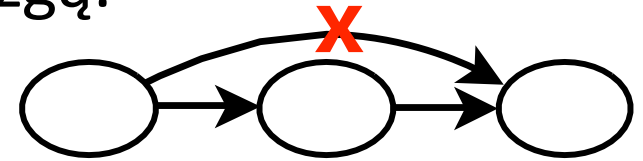


Prieš

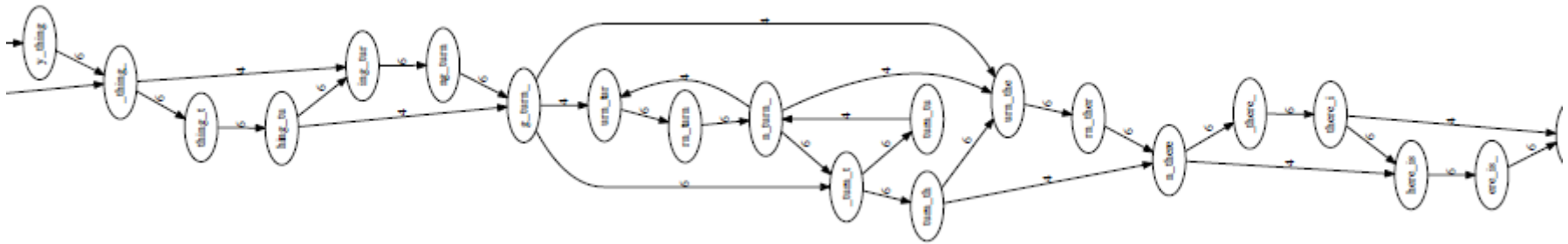


# Apjungimas

Pašalinamos jungtys, kurias galime numanyti iš kitų, pradedant nuo jungčių, kurios praleidžia vieną mazgą.

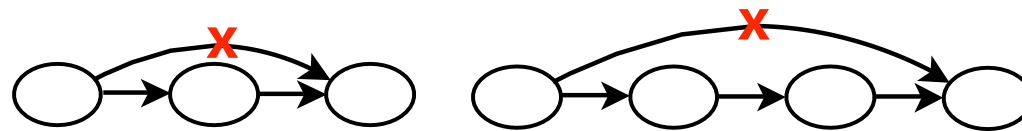


**Po**

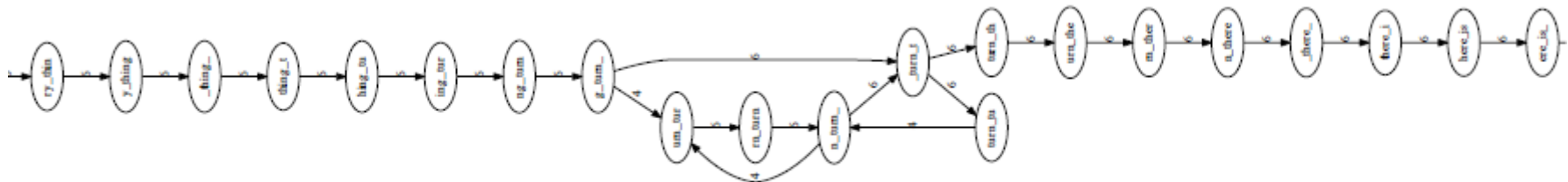


# Apjungimas

Pašalinamos jungtys, kurias galime numanyti iš kitų, pradedant nuo jungčių, kurios praleidžia vieną, **arba du mazgus**.



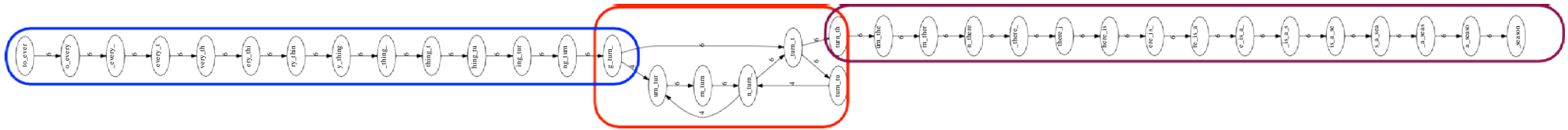
Po



Dar paprastesnis...

# Apjungimas

Išvedam kontigus, kurie atitinka nesišakojančius fragmentus



Kontigas 1

to\_every\_thing\_turn\_

Kontigas 2

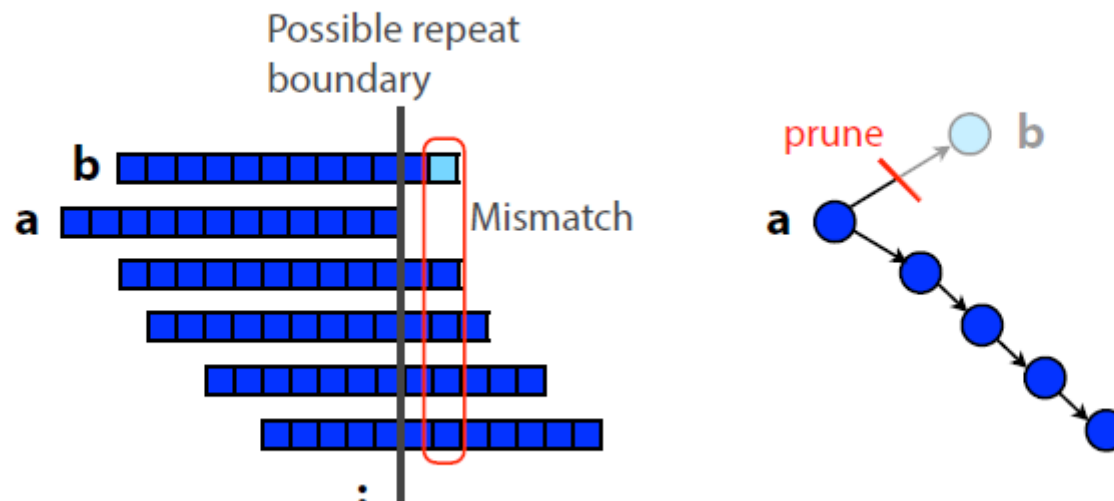
turn\_there\_is\_a\_season



Neišsprendžiami  
pasikartojimai

# Apjungimas

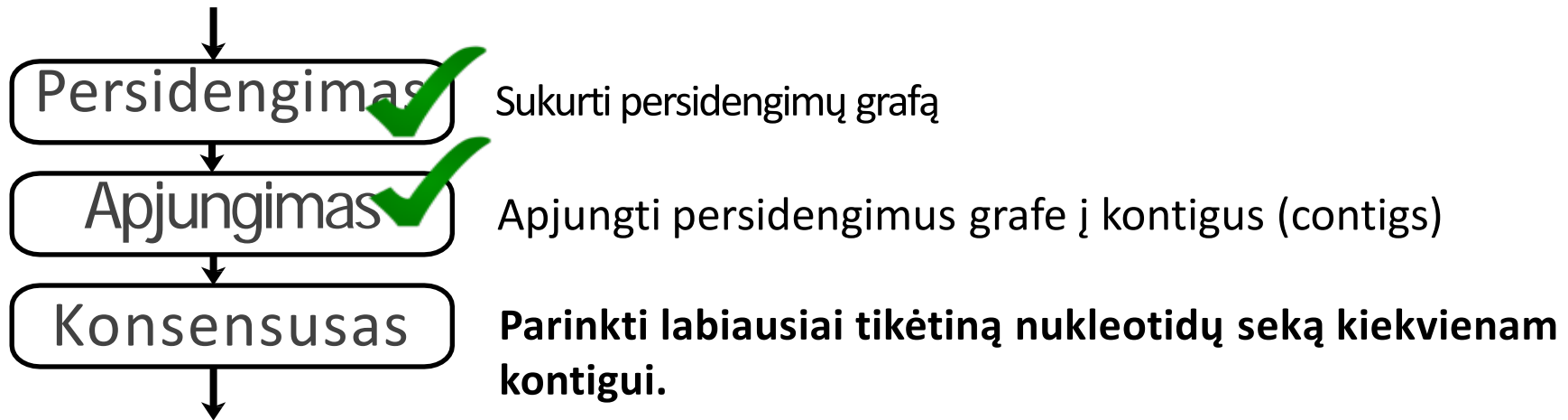
Realybėje reikia pašalinti ir tokius atsišakojimus, kurie kyla iš sekvenavimo klaidų



Nesutapimas gali būti dėl klaidos arba dėl pasikartojančios sekos (repeat). Kadangi kelias per b baigiasi netikėtai ir patenkam į “akligatvį”, laikom kad, tai dėl klaidos.

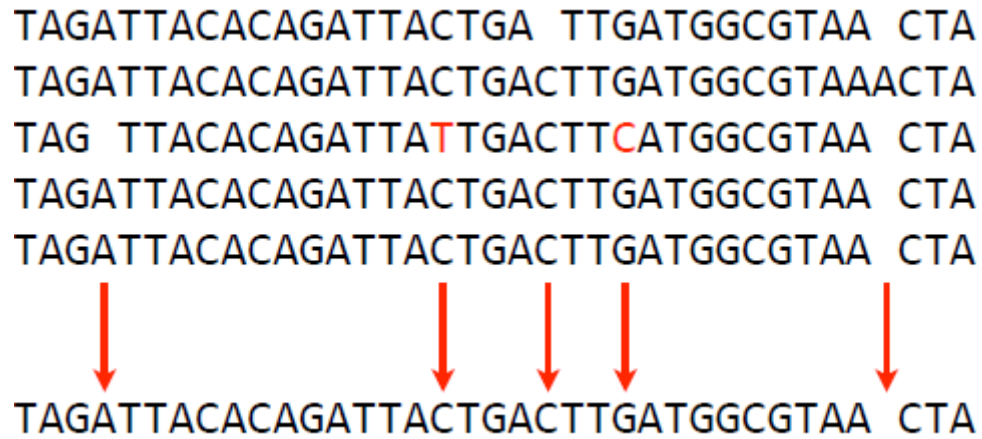


# Persidengimas-apjungimas-konsensusas



# Konsensusas

```
TAGATTACACAGATTACTGA TTGATGGCGTAA CTA  
TAGATTACACAGATTACTGACTTGATGGCGTAACTA  
TAG TTACACAGATTATTGACTTCATGGCGTAA CTA  
TAGATTACACAGATTACTGACTTGATGGCGTAA CTA  
TAGATTACACAGATTACTGACTTGATGGCGTAA CTA
```



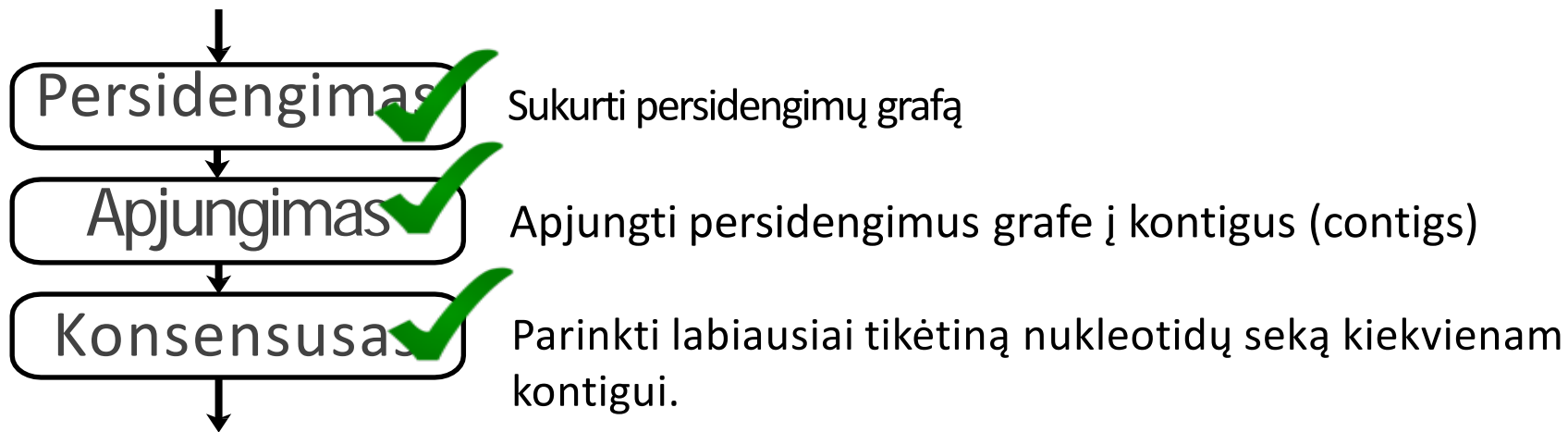
```
↓ ↓ ↓ ↓ ↓  
TAGATTACACAGATTACTGACTTGATGGCGTAA CTA
```

Sugretinam visus  
read'us, sudarančius  
kontigą.

Gaunam, galutinę seką pagal  
“Daugumos” balsą.

Galimos problemos dėl  
ploidiškumo...

# Persidengimas-apjungimas-konsensusas



## OLC trūkumai

Persidengimų grafo sudarymas lėtas.  $O(N + a)$ ,  $O(N^2)$

Persidengimų grafas didelis, vienas mazgas – vienas nuskaitymas...

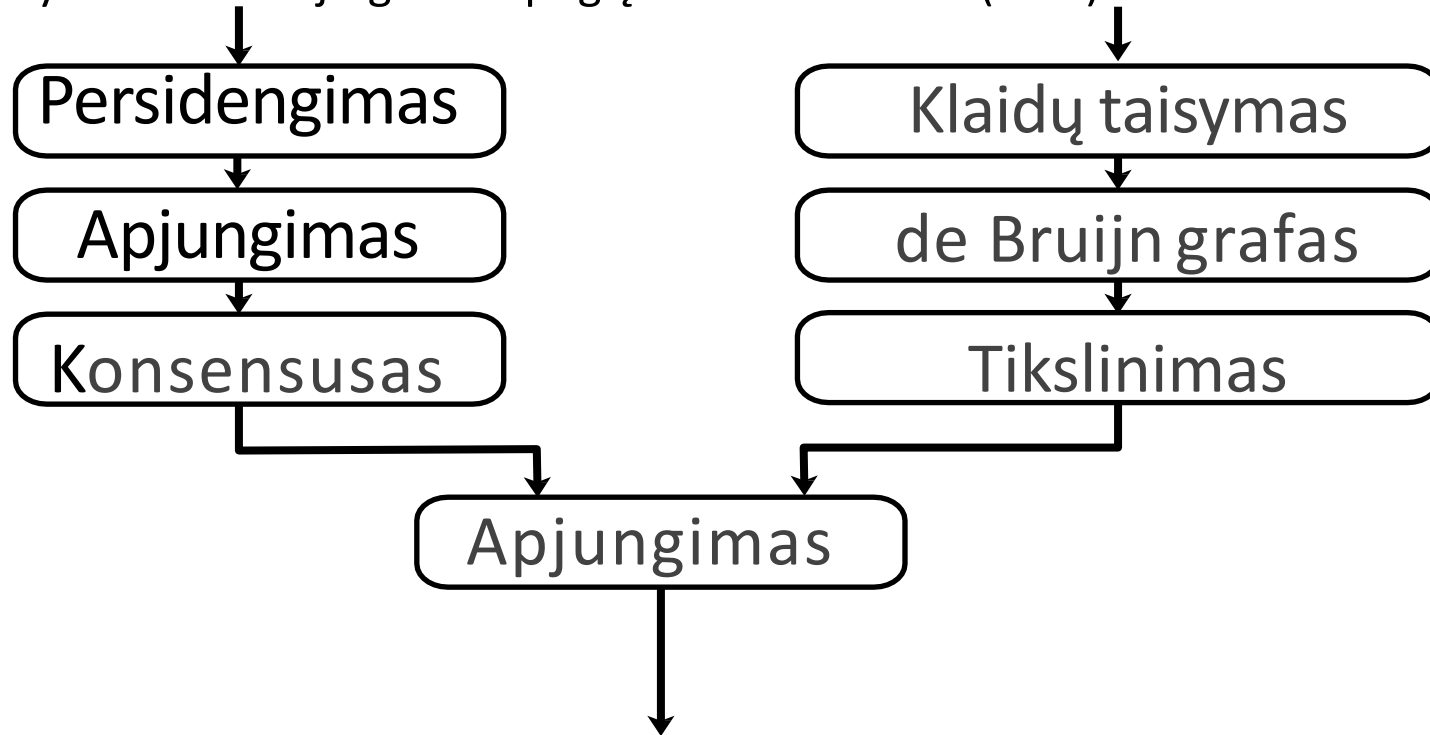
~ 100ai milijonų nuskaitymų, milijardai bazių porų...

Labai daug atminties..

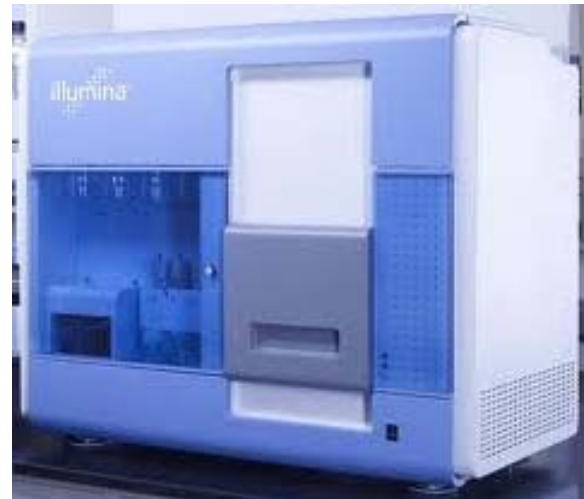
# Kaip iš fragmentų gauti vientisą seką?

Alternatyva 1: persidengimas-apjungimas-konsensusas surinkimas (Overlap-Layout-Consensus) (OLC).

Alternatyva 2: de Bruijn grafais pagrįstas surinkimas (DBG)

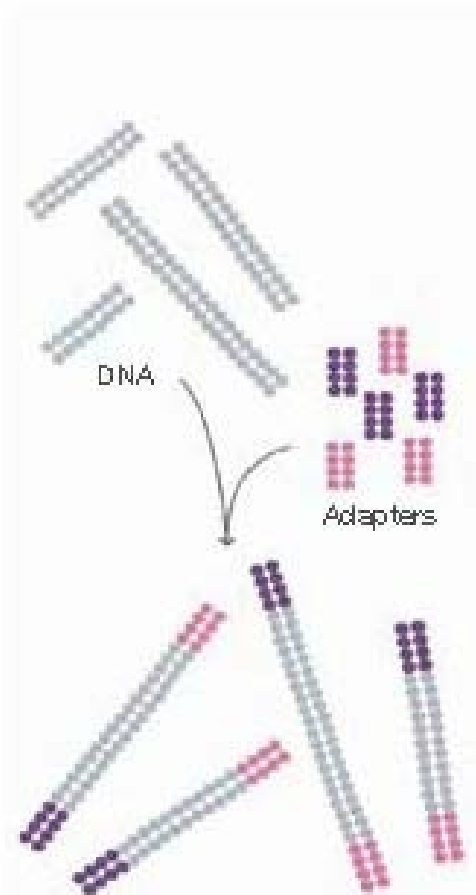


# Daugiau apie sekvenavimą



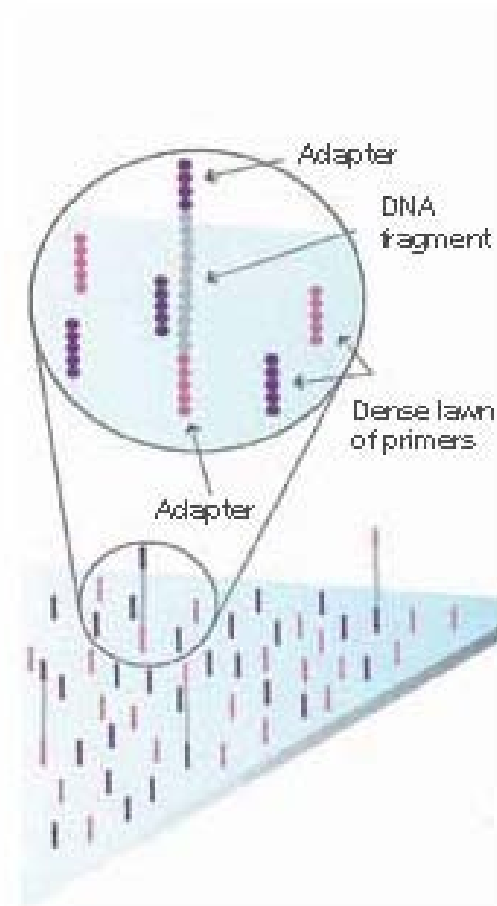
# Illumina

Figure 2: Prepare Genomic DNA Sample



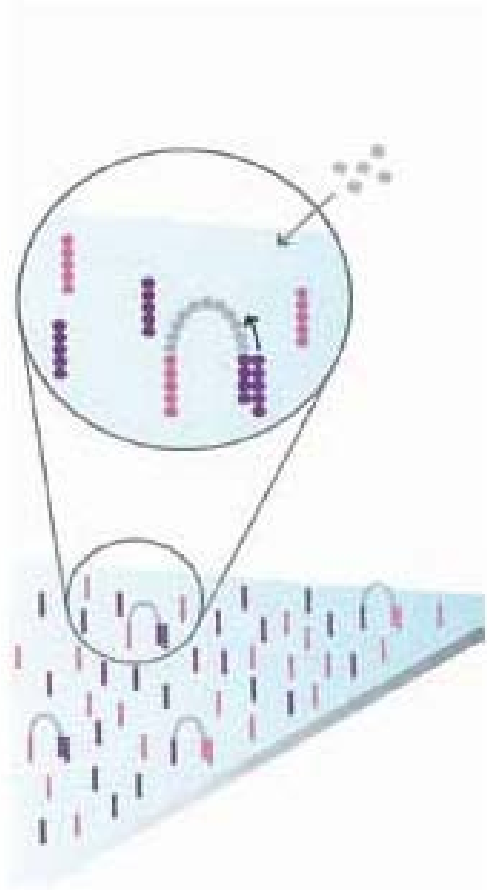
Randomly fragment genomic DNA and ligate adapters to both ends of the fragments.

Figure 3: Attach DNA to Surface



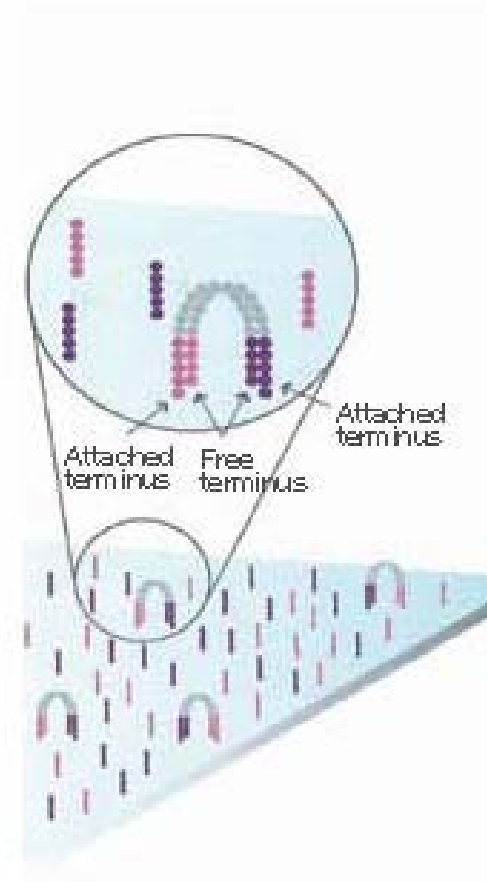
Bind single-stranded fragments randomly to the inside surface of the flow cell channels.

Figure 4: Bridge Amplification



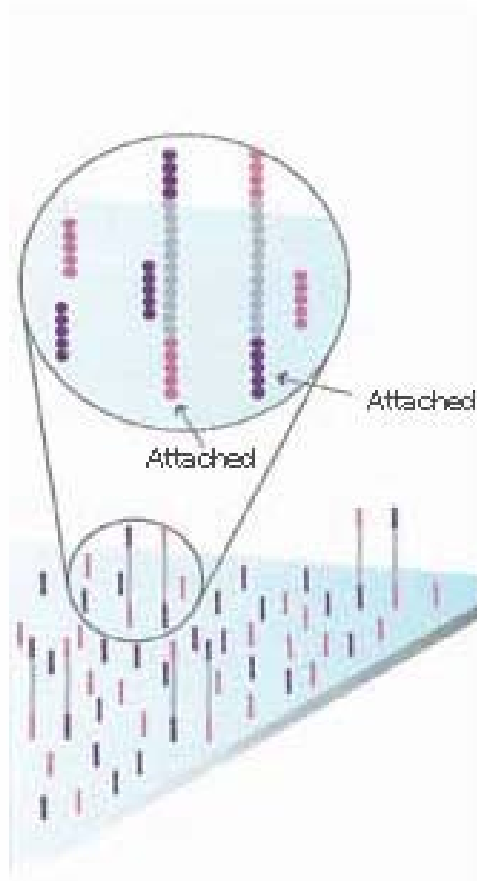
Add unlabeled nucleotides and enzyme to initiate solid-phase bridge amplification.

Figure 5: Fragments Become Double Stranded



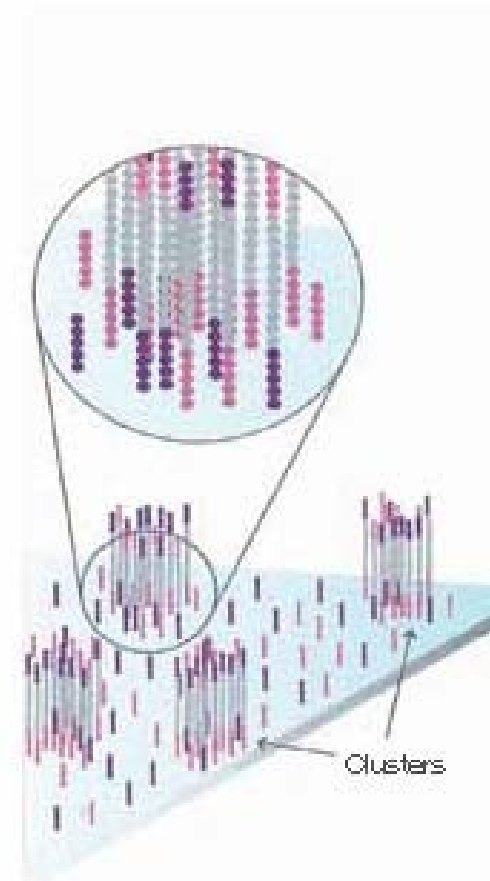
The enzyme incorporates nucleotides to build double-stranded bridges on the solid-phase substrate.

Figure 6: Denature the Double-Standed Molecules



Denaturation leaves single-stranded templates anchored to the substrate.

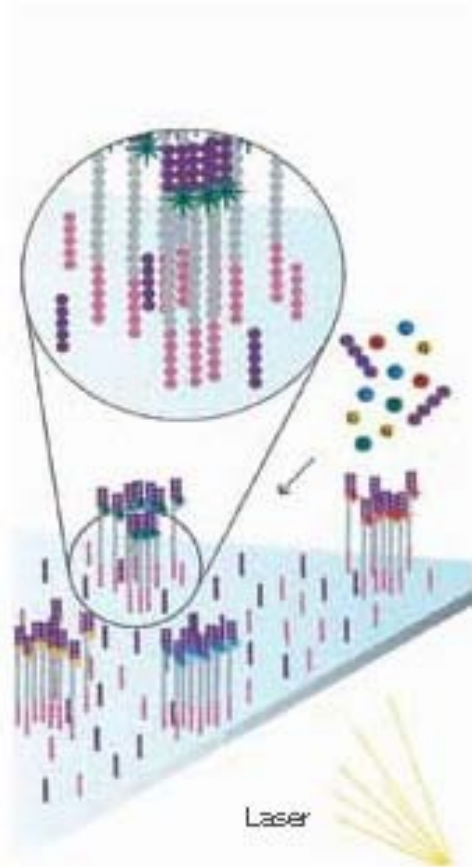
Figure 7: Complete Amplification



Several million dense clusters of double-stranded DNA are generated in each channel of the flow cell.



Figure 8: Determine First Base



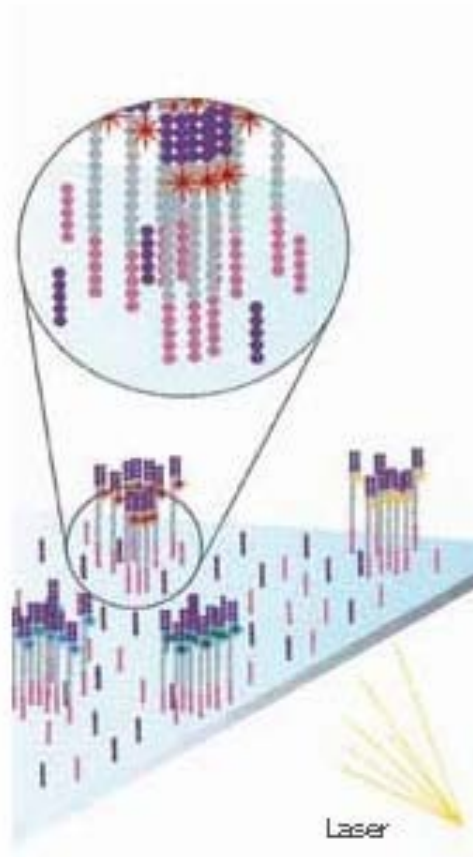
The first sequencing cycle begins by adding four labeled reversible terminators, primers, and DNA polymerase.

Figure 9: Image First Base



After laser excitation, the emitted fluorescence from each cluster is captured and the first base is identified.

Figure 10: Determine Second Base



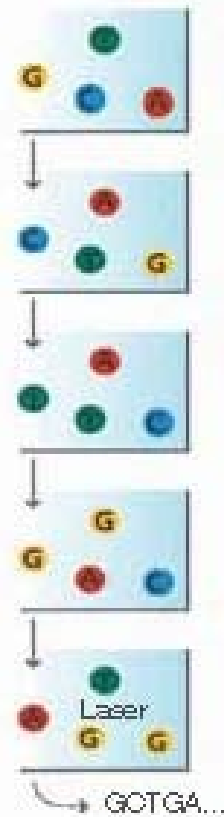
The next cycle repeats the incorporation of four labeled reversible terminators, primers, and DNA polymerase.

Figure 11: Image Second Chemistry Cycle



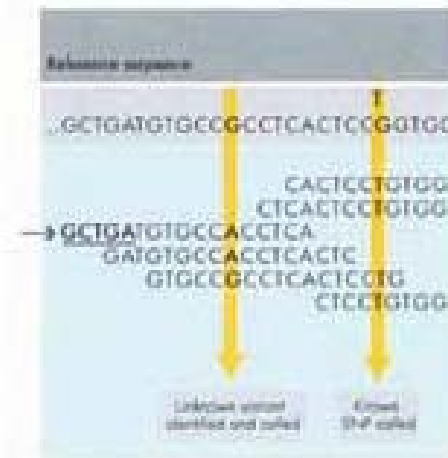
After laser excitation, the image is captured as before, and the identity of the second base is recorded.

Figure 12: Sequencing Over Multiple Chemistry Cycles



The sequencing cycles are repeated to determine the sequence of bases in a fragment, one base at a time.

Figure 13: Align Data



The data are aligned and compared to a reference, and sequencing differences are identified.

# De Bruijn grafo surinkimas

Konsepcija panaši į anksčiau aptartą metodą, bet turi privalumų...

# k-mer

“k-mer” subseka , kurios ilgis  $k$

S: GGCGATTCATCG

*mer*: Graikiškai “dalis”

Sekos S, 4-meras : ATTC

Visi sekos S 3-merai: GGC  
GCG  
CGA  
GAT  
ATT  
TTC  
TCA  
CAT  
ATC  
TCG

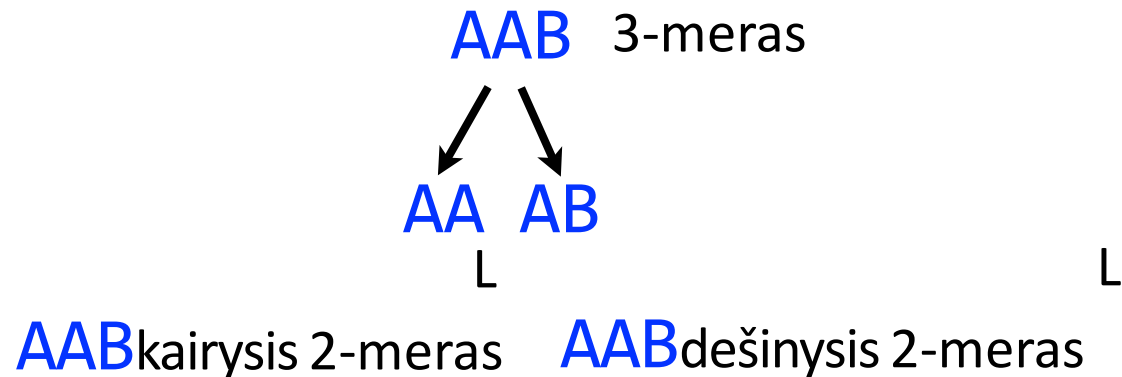
“k-1-mer” reikš subeką, kurios ilgis  $k - 1$

# De Bruijn grafas

Praedam su vienodo ilgio sekomis iš nusekvenuoto genomo

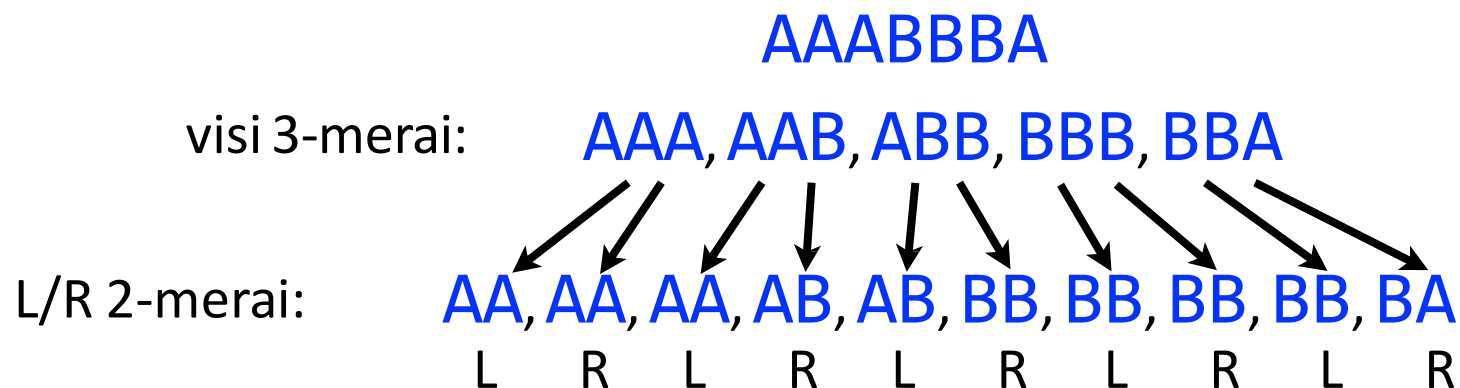
AAA, AAB, ABB, BBB, BBA

AAB yra  $k$ -meras ( $k = 3$ ). AA yra jo kairysis ( $L$ )  $k-1$ -meras, ir AB yra jo dešinysis ( $R$ )  $k-1$ -mer.

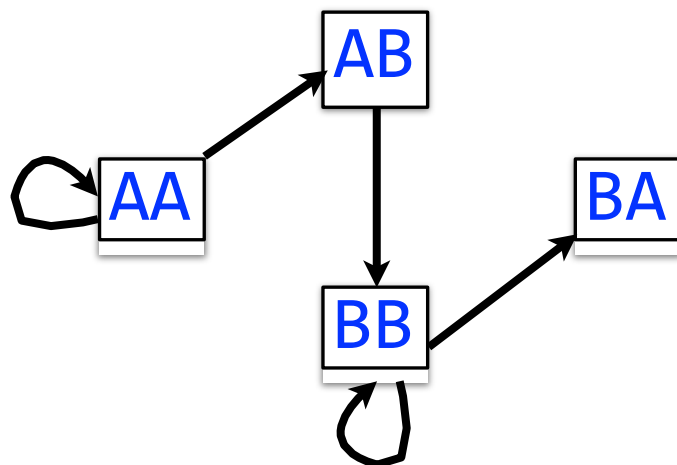


# De Bruijn grafas

Papimam ir suskaldom kiekvieną 3 simbolių ilgio fragmentus į kairiuosius ir dešiniuosius 2-merus.

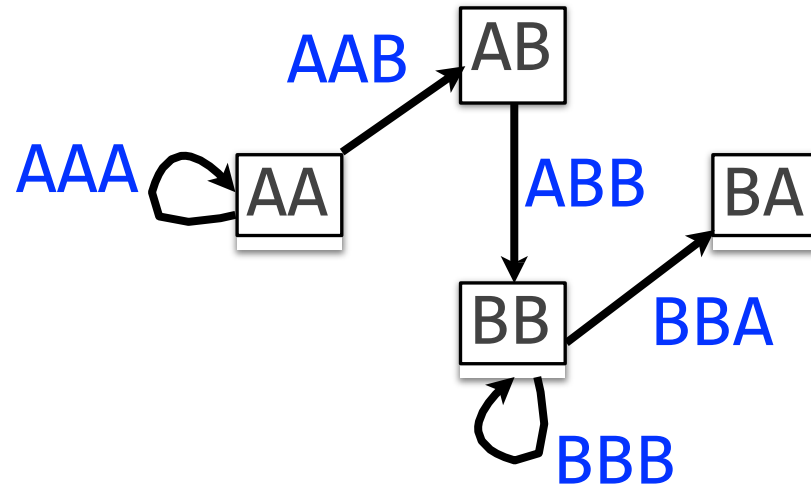


Tegul 2-merai būna mazgais naujame grafe. Nubrėžiame kryptines jungtis tarp atitinkamų L iki R 2-merų.



Kiekviena jungtis atitinka pradines sekas, kurių ilgis 3 simboliai.

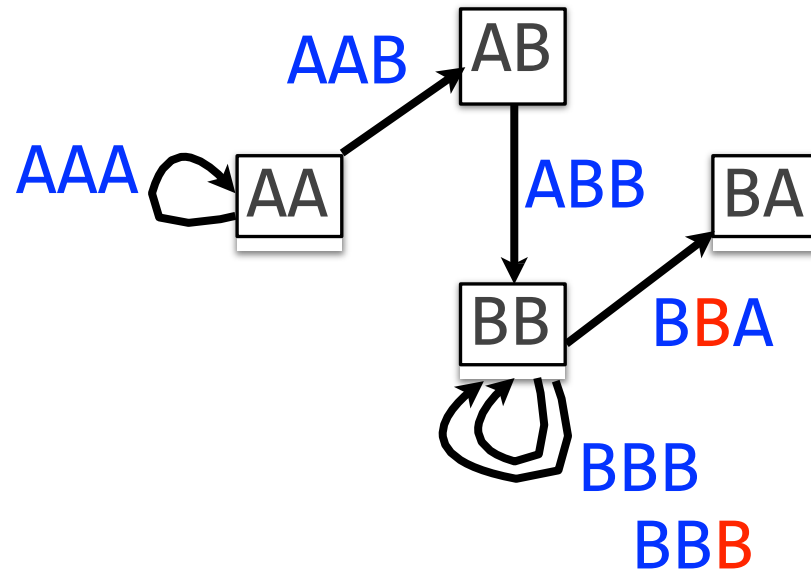
# De Bruijn grafas



Jungis atitinka persidengimą ( $k-1$  ilgio) tarp dviejų  $k-1$  mers. Atitinka  $k$ -mer ilgo įvesties simbolius.



# De Bruijn grafas



Jei pridėtume vieną ar daugiau B į pradinę seką: **AAABBBBA**, ir atnaujintume grafą, gautume *multijungtį*..

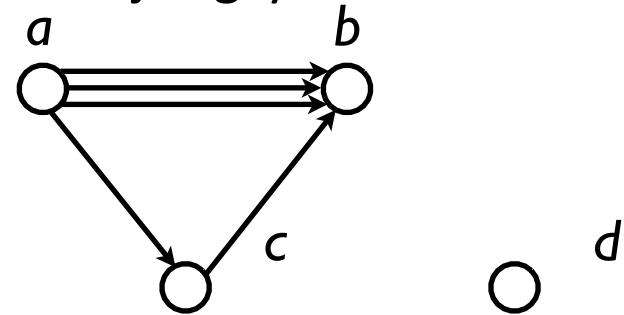
# Kryptinis multigrafas

Kryptinį multigrafą  $G(V, E)$  sudaro viršūnių (mazgų) rinkinys  $V$  ir jas jungiančios, kryptį turinčios jungtys  $E$ .  $E$  elementai gali kartotis.

Viršūnės įeities laipsnis (*indegree*) = # įeinančios jungtys.

Viršūnės išeities laipsnis (*outdegree*) = # išeinančios jungtys.

De Bruijn grafas yra kryptinis multigrafas



$$V = \{ a, b, c, d \}$$

$$E = \{ (a, b), (a, b), (a, b), (a, c), (c, b) \}$$

Pasikartoja

# Oilerio maršruto apibrėžimas ir teiginiai

Viršūnė yra subalansuota, jei sutampa įėjties ir išeities laipsnis sutampa

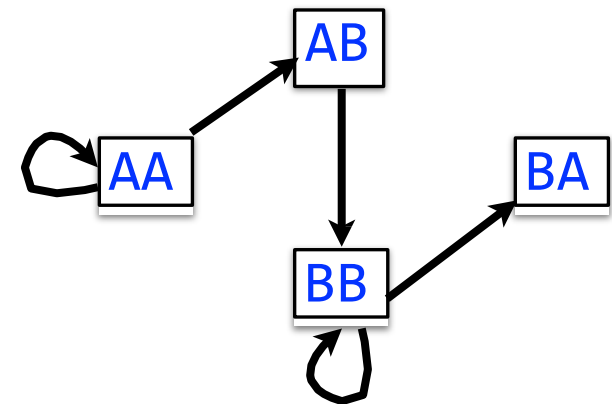
Viršūnė yra pusiau subalansuota jei išeities ir įėjties skiriasi 1-netu.

Grafas yra sujungtas jei kiekviena viršūnė yra sujungta bent su viena kita viršūne.

*Oilerio maršrutu, kiekviena **viršūnė?** aplankoma tik kartą.*

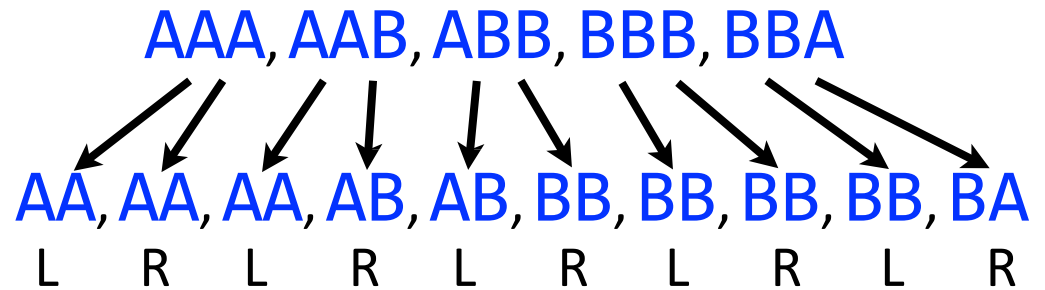
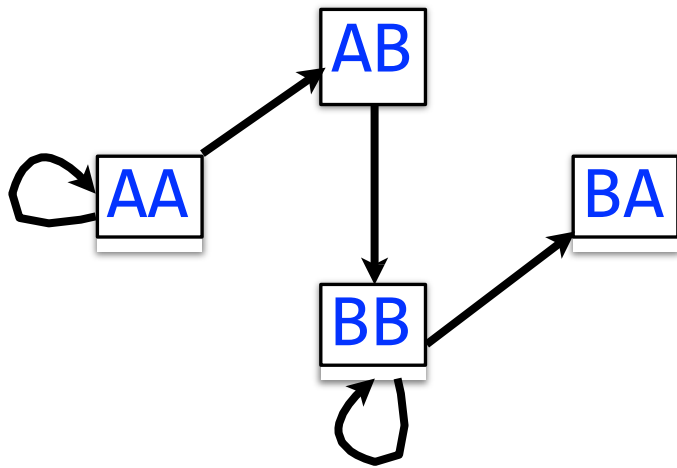
Ne viai grafai turi Oilerio maršrutą, tie kurie turi – vadinami Oilerio grafais

Kryptinis sujungtas grafas yra Oilerio grafas tada ir tik tada, jei jis turi ne daugiau nei 2-vi pusiau subalansuotas viršūnes, o kitos viršūnės yra subalansuotos.



# De Bruijn grafas

Vėl mūsų grafas



Ar jis Oilerio? Taip

Argumentas 1:  $AA \rightarrow AA \rightarrow AB \rightarrow BB \rightarrow BB \rightarrow BA$

Argumentas 2:  $AA$  ir  $BA$  yra pusiau subalansuoti,  $AB$  ir  $BB$  yra subalansuoti

# De Bruijn grafas

De Bruijn grafo sudarymas genomui

Laikom, kad buvo ideali sekoskaita, be klaidų ir gauti *k-ilgio* genomo subsekos yra nuskaitytos tik kartą.

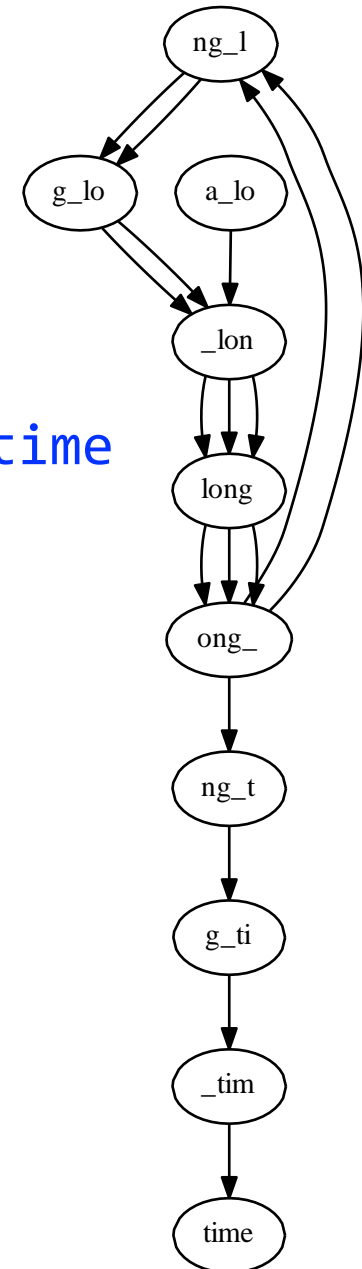
Pasirenkam subseką, kurios ilgis  $k=5$

Kiekvieną kmer'ą suskaldom į kairinį ir dešinįj  $k-1$  merus.

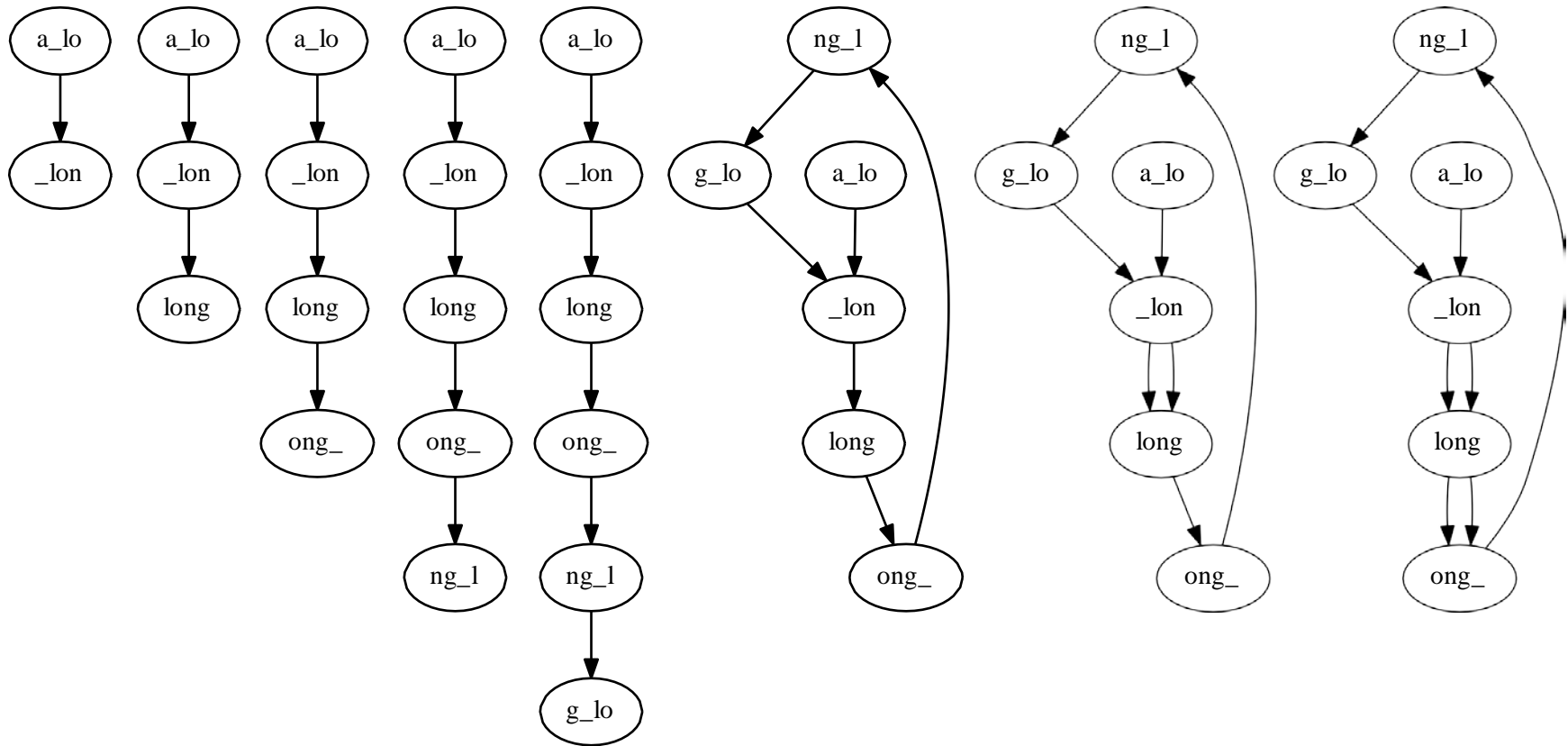
Pridedam  $k-1$  merus kaip viršūnes į de Bruijn grafą (jeigu jų ten nėra), pridedam jungtis iš kairiojo į dešinįj  $k-1$  merus.

a\_long\_long\_long\_time

long\_  
long ong\_



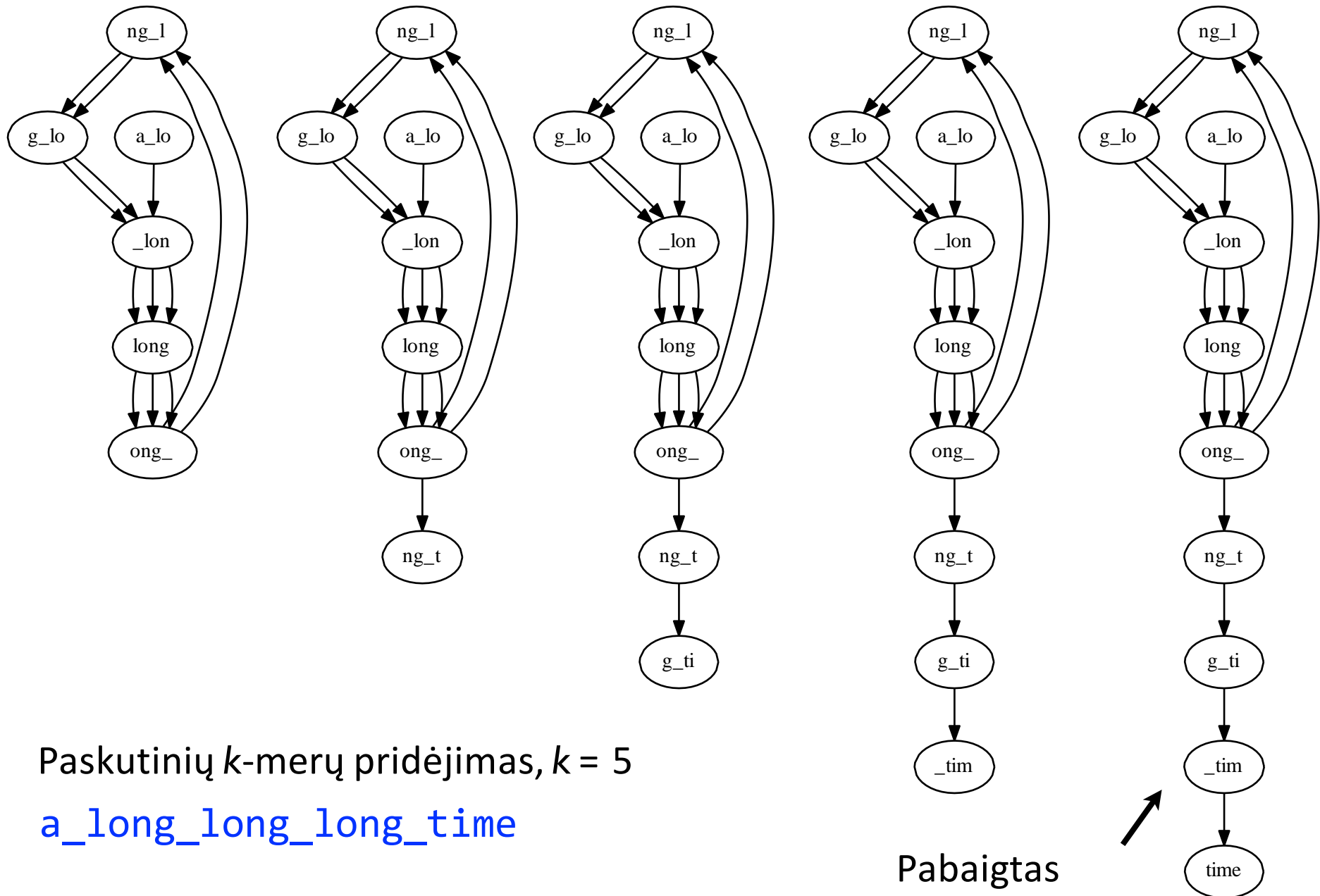
# De Bruijn grafas



Pirmų 8-i  $k$ -merų pridėjimas,  $k = 5$

`a_long_long_long_time`

# De Bruijn grafas



Paskutinių  $k$ -merų pridėjimas,  $k = 5$

`a_long_long_long_time`

Pabaigtas  
grafas

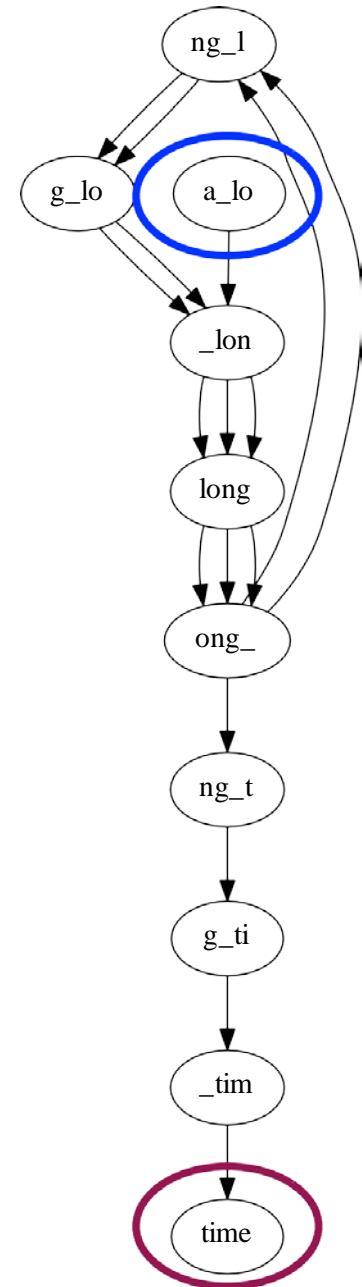
# De Bruijn grafas

Esant idealiam sekvenavimui visada gauname Oilerio grafą. Kodėl?

$k-1$ -mero iš kairiojo galo viršūnė yra pusiau subalansuota, nes yra viena daugiau išeinančių nei įeinančių mazgų.

$k-1$ -mero iš dešiniojo galo viršūnė yra pusiau subalansuota, nes yra viena daugiau įeinančių nei išeinančių mazgų.

Kitos viršūnės yra subalansuotos, nes kiekvienam nuskaitymui yra po du  $k-1$  merą: kairysis  $k-1$ -meras pasirodo tiek pat kartų kiek ir dešinysis.





# De Bruijn grafo realizacija

```
class DeBruijngrafas:
    """ A De Bruijn multigrafas built from a collection of strings.
        User supplies strings and k-mer length k. Nodes of the De
        Bruijn grafas are k-1-mers and edges join a left k-1-mer to
        a right k-1-mer. """

    @staticmethod
    def chop(st, k):
        """ Chop a string up into k mers of given length
        """
        for i in xrange(0, len(st)-(k-1)): yield
            st[i:i+k]

    class Node:
        """ Node in a De Bruijn grafas, representing a k-1mer
        """
        def __init__(self, km1mer):
            self.km1mer = km1mer

        def __hash__(self):
            return hash(self.km1mer)

    def __init__(self, strIter, k):
        """ Build De Bruijn multigrafas given strings and k-mer length k
        """
        self.G = {} # multimap from nodes to neighbors
        self.nodes = {} # maps k-1-mers to Node objects
        self.k = k
        for st in strIter:
            for kmer in self.chop(st, k):
                km1L, km1R = kmer[:-1], kmer[1:]
                nodeL, nodeR = None, None
                if km1L in self.nodes:
                    nodeL = self.nodes[km1L]
                else:
                    nodeL = self.nodes[km1L] = self.Node(km1L)
                if km1R in self.nodes:
                    nodeR = self.nodes[km1R]
                else:
                    nodeR = self.nodes[km1R] = self.Node(km1R)
                self.G.setdefault(nodeL, []).append(nodeR)
```

I Suskaldom pateiktas  
sekas į k-merus

I Kiekvienam  $k$ -merui,  
randame kairinį ir  
dešinį  $k-1$ -merą.  
Sukuriamė, jei reikia,  
viršūnes ir pridedam  
jungtis.

# De Bruijn grafas

Oilerio grafams Oilerio maršrtas, gali būti rastas per  $O(|E|)$  time.  $|E|$  yra viršūnių #.

Paverčiam grafą cikliniu Oilerio grafue (pridedam vieną viršūnę, kad visos viršūnės būtų subalansuotos).

Įžvalga: Jei  $C$  yra ciklas ir yra Oilerio grafas. Pašalinus  $C$  viršūnes, likusios sujungtos viršūnės taip pat sudaro Oilerio grafą.

```
# Make all nodes balanced, if not already
tour = []
# Pick arbitrary node
src =
    g.iterkeys().next()
def __visit(n):
    while len(g[n]) > 0:
        dst = g[n].pop()

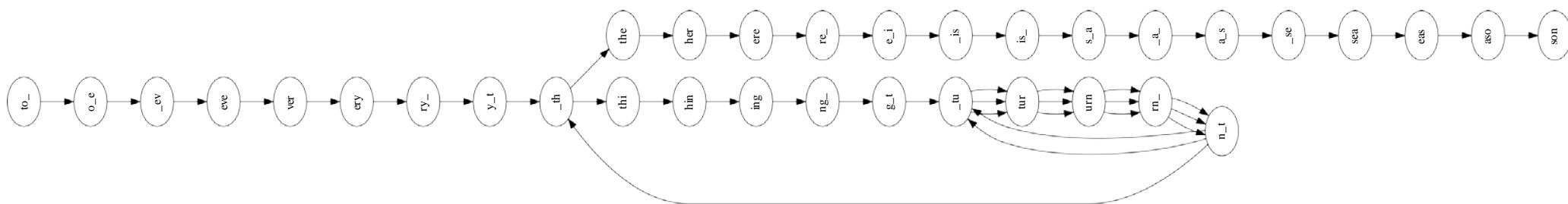
        __visit(dst)
    tour.append(n)
__visit(src)
# Reverse order, omit repeated node
tour = tour[::-1][:-1]
# Turn tour into walk, if necessary
```

# De Bruijn grafas

Iliustravūs Oilerio grafai ir pavyzdžiai::

<http://nbviewer.ipython.org/7237207>

```
>>> st = "to_everything_turn_turn_turn_there_is_a_season"  
>>> G = DeBruijngrafas([st], 4)  
>>> path = G.eulerianWalkOrCycle()  
>>> superstring = path[0] + ''.join(map(lambda x: x[---1], path[1:]))  
>>> print superstring  
to_everything_turn_turn_turn_there_is_a_season
```

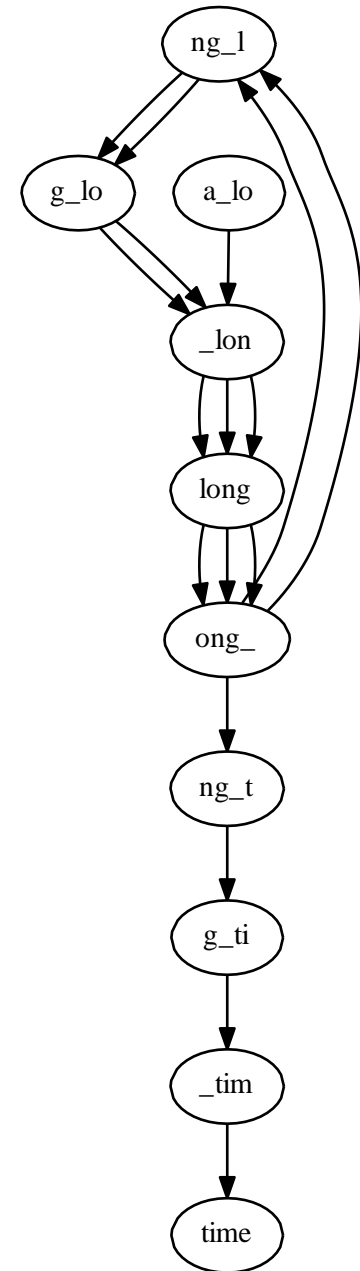


Šis atvejis persidengimo algoritmasms neįkandama...

# De Bruijn grafas

Laikant, kad sekvenavimas vyko idealiai ir klaidų nepadaryta, galima efektyviai rasti Oilerio maršrutą.

Ar visada Oilerio maršrutas atitinka ir originalią seką?



# De Bruijn grafas

**NE:** deja grafas gali turėti daugybinius Oilerio maršrutus, iš kurių tik vienas atitinka tikrąją seką.

Dešinėje: grafas sekai **ZABCDABEFABY**,  $k = 3$

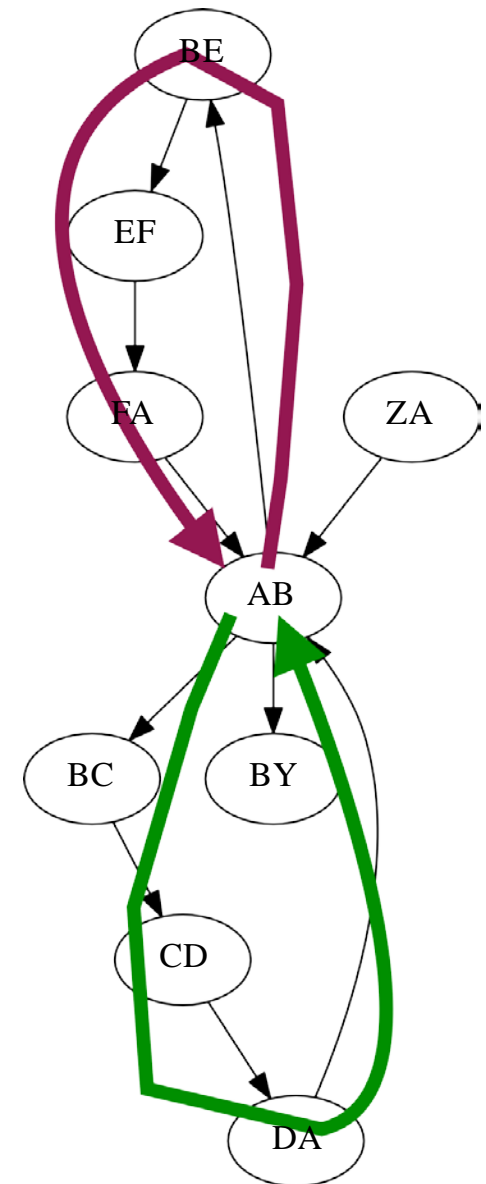
Du alternatyvūs Oilerio maršrutai:

**ZA** → **AB** → **BE** → **EF** → **FA** → **AB** → **BC** → **CD** → **DA** → **AB** → **BY**

**ZA** → **AB** → **BC** → **CD** → **DA** → **AB** → **BE** → **EF** → **FA** → **AB** → **BY**

Jie tinka dviejų viršūnių kryptinius ciklus sujungtus per AB viršūnę.

**AB** yra pasikartojimas: **ZABCDABEFABY**



# De Bruijn grafas

to\_every\_thing\_turn\_turn\_turn\_there\_is\_a\_season

Atvejis kai  $k = 4$  veikia:

```
>>> st = "to_every_thing_turn_turn_turn_there_is_a_season"
>>> G = DeBruijngrafas([st], 4)
>>> path = G.eulerianWalkOrCycle()
>>> superstring = path[0] + ''.join(map(lambda x: x[---1], path[1:]))
>>> print superstring
to_every_thing_turn_turn_turn_there_is_a_season
```

Bet, kai  $k = 3$  – neveikia,

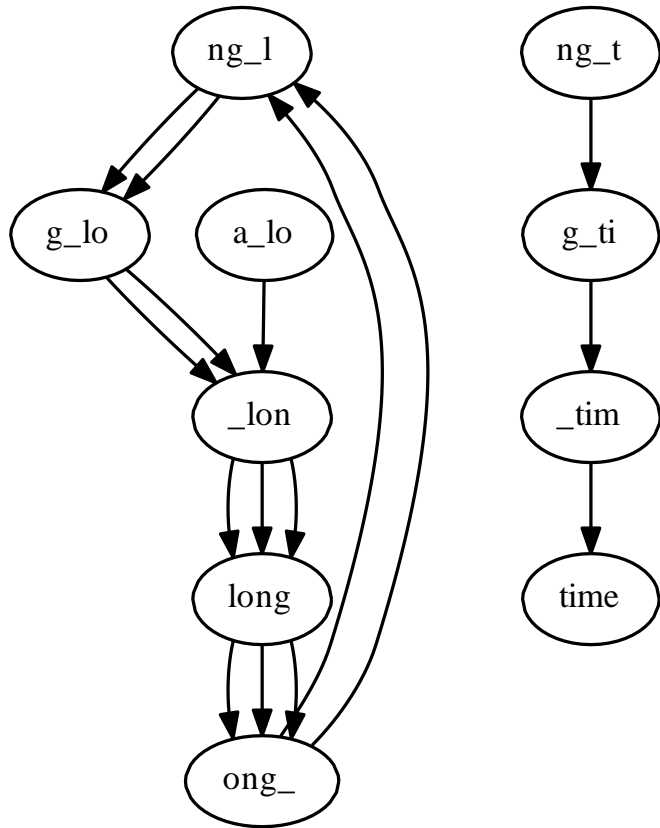
```
>>> st = "to_every_thing_turn_turn_turn_there_is_a_season"
>>> G = DeBruijngrafas([st], 3)
>>> path = G.eulerianWalkOrCycle()
>>> superstring = path[0] + ''.join(map(lambda x: x[---1], path[1:]))
>>> print superstring
to_every_turn_turn_thing_turn_there_is_a_season
```

dėl pasikartojimų, kurie neišsprendžiami, kai  $k = 3$

# De Bruijn grafas

Trūkiai perdengime gali pasireikšti kip nesujungtas grafas.

Grafas `a_long_long_time`,  $k = 5$  praleistas `ong_t`:



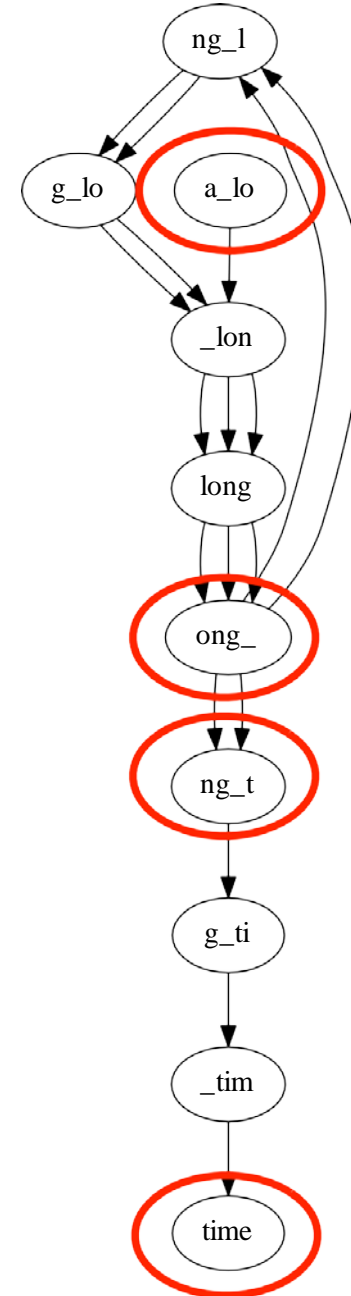
Sujungti komponentai sudaro Oilerio grafą, bet visi komponentai nėra Oilerio grafas.

# De Bruijn grafas

Skirtumai perdengime gali baigtis ne Oilerio grafu.

grafas `a_long_long_long_time`,  
 $k = 5$ , bet su papildoma `ong_t`, kopija:

grafas turi 4 **pusiau**  
**subalansuotas** viršūnes, taigi jis  
nėra Oilerio.





# De Bruijn grafas

Gauti surinkimą remiantis Oilerio maršrutu yra “elegantiška”, bet dažnai nepraktiška.

Netolygus perdengimas, sekvenavimo klaidos ir t.t. Paverčia grafą ne Oilerio tipo. Net jei grafas yra Oilerio, dėl pasikartojimų gali susidaryti keletas alternatyvių Oilerio maršrutų.

Kingsford, Carl, Michael C. Schatz, and Mihai Pop. "Assembly complexity of prokaryotic genomes using short reads." *BMC bioinformatics* 11.1 (2010): 21.

# De Bruijn grafas

**Praktikoje** De Bruijn grafupagrįsti metodai “pasiduoda” ties pasikartojančiomis sekomis, taip kaip ir persidengimais pagrįstos programos.

Kokie privalumai?...

# De Bruijn grafas

Grafas gali būti sukurtas  $O(N)$  tikėtinu laiku,  $N$  = bendras nuskaitymų ilgis. Idealiu atveju De Bruijn grafas užima  $O(\min(N, G))$  atminties;  $G$  = genomo ilgis. Atkreipkit dėmesį: esant aukštam perdengimui  $G \ll N$

Mažiau lyginant su persidengimo grafas

atminties kiekis  $O(N + a)$

persidengimų suradimas proporcingas laikui  $O(N + a)$

$a$  is  $O(n^2)$

PHASE : INTERPRETATION  
TWO :

SEIDMAN  
GOLDMAN

