

# Fragment Assembly

Determinazione dei *contigs*:

- ✓ Overlap Layout Consensus (OLC) → Overlap
- ✓ Grafo di de Bruijn (dBG) → k-mer

goldrush

uso overlap per  
assemblare

scompongo  
le reads per  
assemblare

Entrambi gli approcci sono graph-based

# [Fragment Assembly come TSP]

$$G = (V, E)$$

$$V = \{r_1, r_2, \dots, r_n\}$$

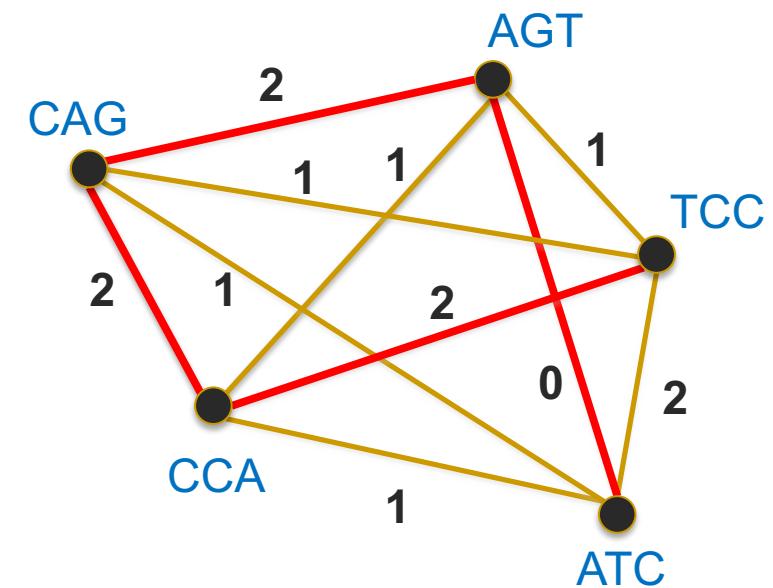
$$E = \{(r_i, r_j) \text{ t.c. } r_i, r_j \in V, r_i \text{ e } r_j \text{ hanno overlap}\}$$

CH DI PESO MASSIMO:

cammino che tocca tutti i vertici del grafo  
una e una sola volta tale che la somma  
dei pesi degli archi coinvolti sia massima

→ Assemblaggio = TCCAGTC

NP-completo



# [Cammino Euleriano (CE)]

$$G = (V, E)$$

Cammino Euleriano → cammino che tocca tutti gli archi del grafo una e una sola volta

Polinomiale

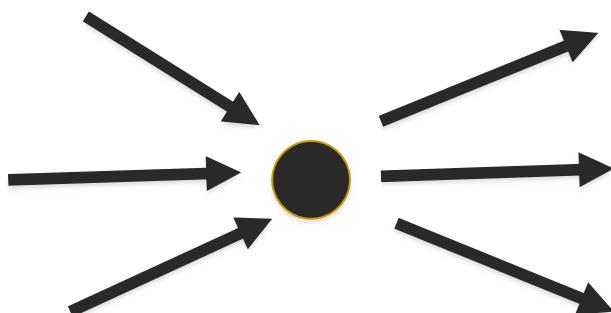
# Ciclo Euleriano

$G = (V, A)$  (grafo orientato)

Quando un grafo ammette un Ciclo Euleriano?

TH: il grafo ammette un Ciclo Euleriano se e solo se è bilanciato

cioè,  $\text{IN}(v) = \text{OUT}(v)$  per ogni  $v \in V$



# Ciclo Euleriano

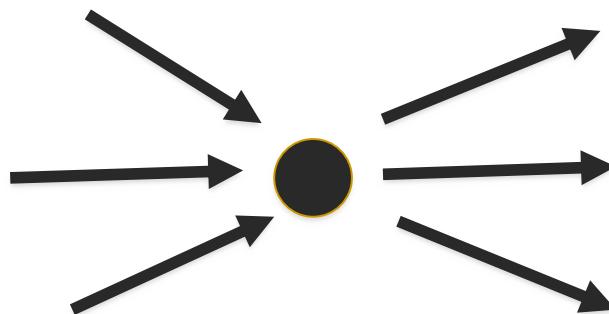
$G = (V, A)$  (grafo orientato)

Quando un grafo ammette un Ciclo Euleriano?

TH: il grafo ammette un Ciclo Euleriano se e solo se è bilanciato

PROOF:

(1) esiste il Ciclo Euleriano  $\rightarrow$  il grafo è bilanciato



# Ciclo Euleriano

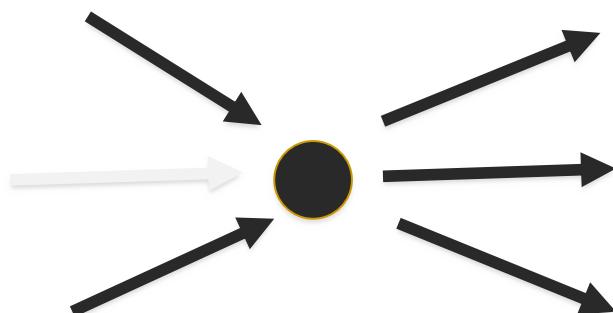
$G = (V, A)$  (grafo orientato)

Quando un grafo ammette un Ciclo Euleriano?

**TH:** il grafo ammette un Ciclo Euleriano se e solo se è bilanciato

**PROOF:**

(1) esiste il Ciclo Euleriano  $\rightarrow$  il grafo è bilanciato



Se per assurdo esistesse un nodo non bilanciato allora esisterebbe un arco che non potrebbe essere percorso

# [Ciclo Euleriano]

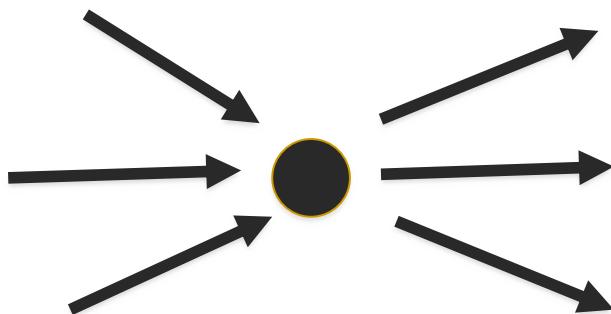
$G = (V, A)$  (grafo orientato)

Quando un grafo ammette un Ciclo Euleriano?

TH: il grafo ammette un Ciclo Euleriano se e solo se è bilanciato

PROOF:

(2) il grafo è bilanciato → esiste il Ciclo Euleriano



# Ciclo Euleriano

## PROOF:

(2) il grafo è bilanciato → esiste il Ciclo Euleriano

## Dimostrazione tramite algoritmo di costruzione

### STEP1: scomposizione in cicli

Si parte da un arco e si percorrono archi successivi fino a formare un ciclo. Due casi:

- (a) tutti gli archi sono stati percorsi
- (b) esistono archi non ancora percorsi

Caso (a): ci si ferma

Caso (b): si ripete a partire da un arco non ancora percorso per trovare un nuovo ciclo (verificando di nuovo i due casi)

# Ciclo Euleriano

## PROOF:

(2) il grafo è bilanciato → esiste il Ciclo Euleriano

Dimostrazione tramite algoritmo di costruzione

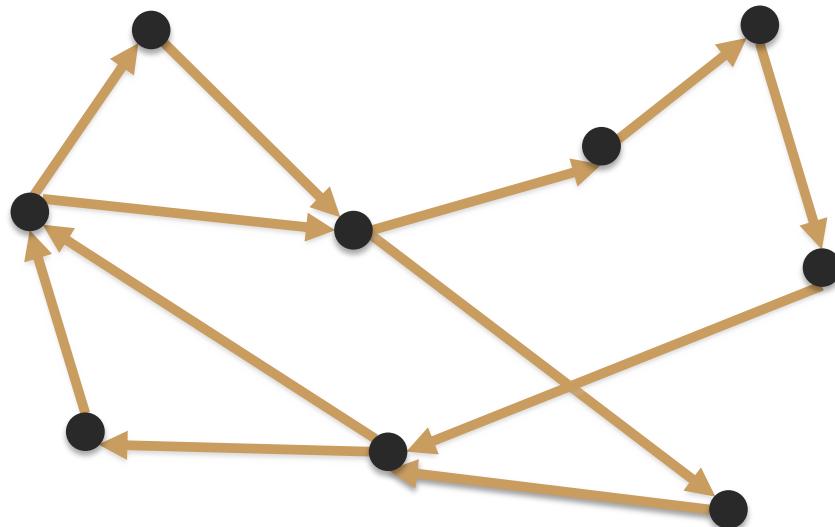
## STEP2: composizione dei cicli trovati

Si parte da un arco (di un determinato ciclo) e si percorrono gli archi del grafo uscendo se possibile su un ciclo diverso.

# Ciclo Euleriano

PROOF:

(2) il grafo è bilanciato → esiste il Ciclo Euleriano

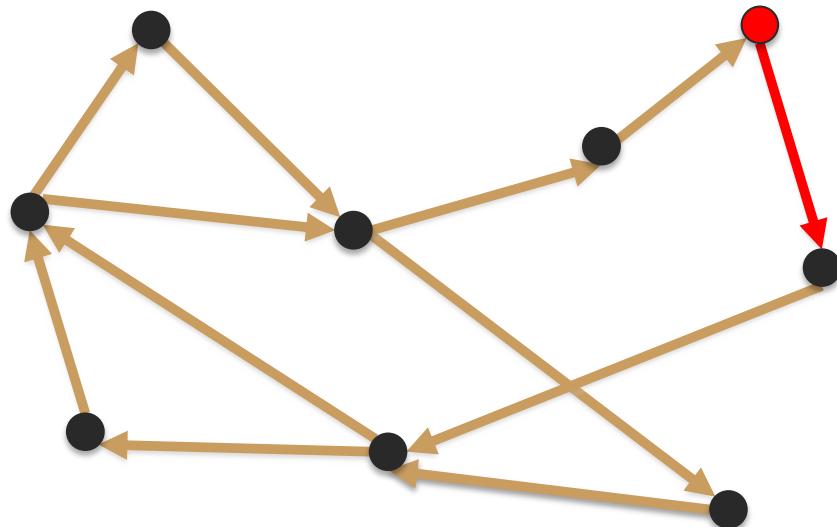


Algoritmo di costruzione del Ciclo Euleriano:  
STEP1: scomposizione del grafo in cicli

# Ciclo Euleriano

PROOF:

(2) il grafo è bilanciato → esiste il Ciclo Euleriano

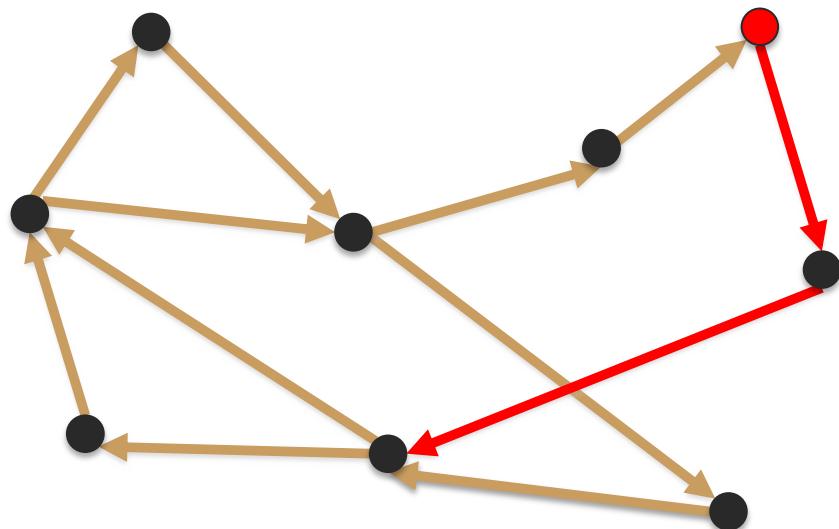


Algoritmo di costruzione del Ciclo Euleriano:  
STEP1: scomposizione del grafo in cicli

# Ciclo Euleriano

PROOF:

(2) il grafo è bilanciato → esiste il Ciclo Euleriano

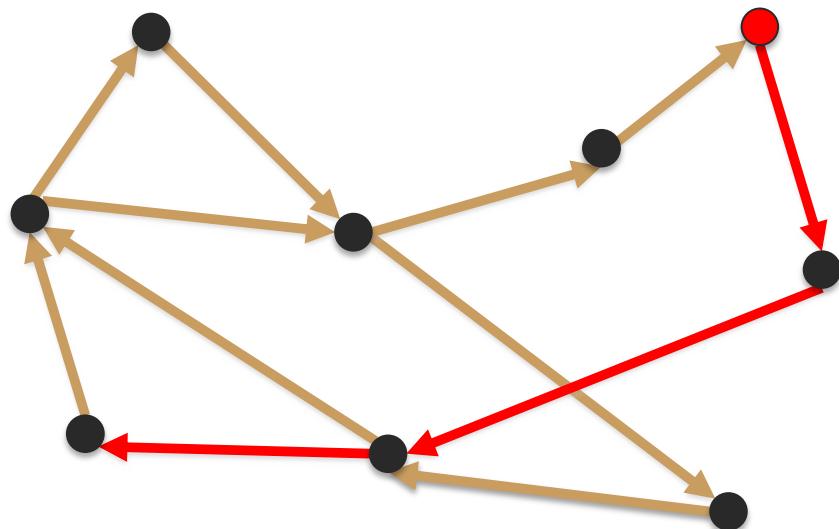


Algoritmo di costruzione del Ciclo Euleriano:  
STEP1: scomposizione del grafo in cicli

# Ciclo Euleriano

PROOF:

(2) il grafo è bilanciato → esiste il Ciclo Euleriano

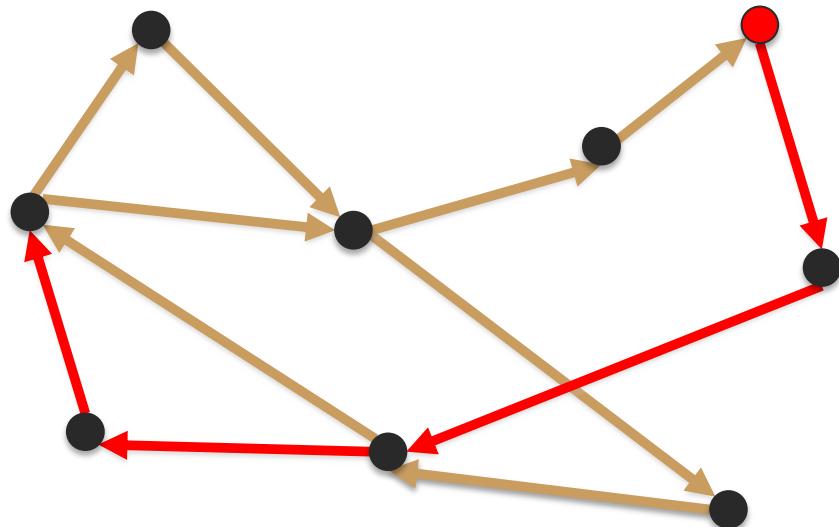


Algoritmo di costruzione del Ciclo Euleriano:  
STEP1: scomposizione del grafo in cicli

# Ciclo Euleriano

## PROOF:

(2) il grafo è bilanciato → esiste il Ciclo Euleriano

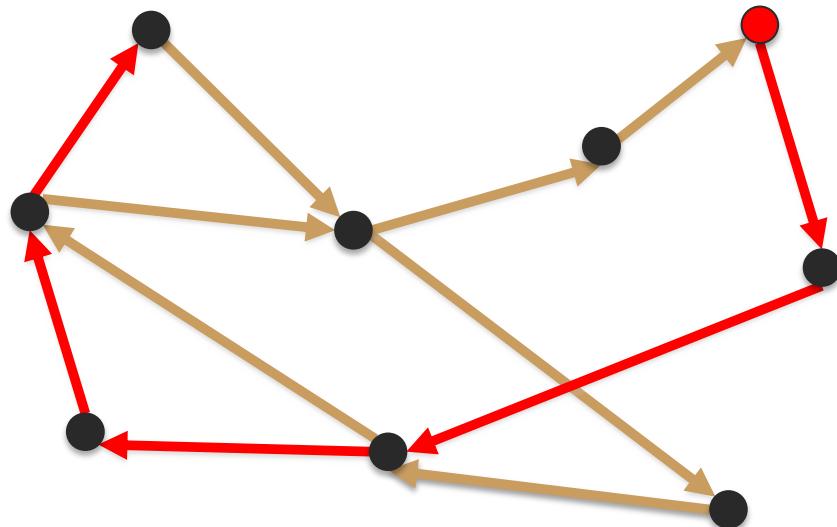


Algoritmo di costruzione del Ciclo Euleriano:  
STEP1: scomposizione del grafo in cicli

# Ciclo Euleriano

## PROOF:

(2) il grafo è bilanciato → esiste il Ciclo Euleriano

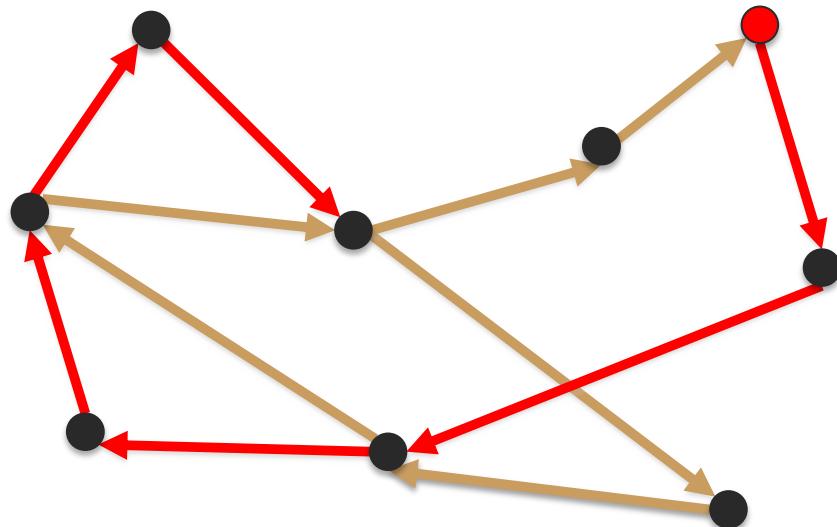


Algoritmo di costruzione del Ciclo Euleriano:  
STEP1: scomposizione del grafo in cicli

# Ciclo Euleriano

PROOF:

(2) il grafo è bilanciato → esiste il Ciclo Euleriano

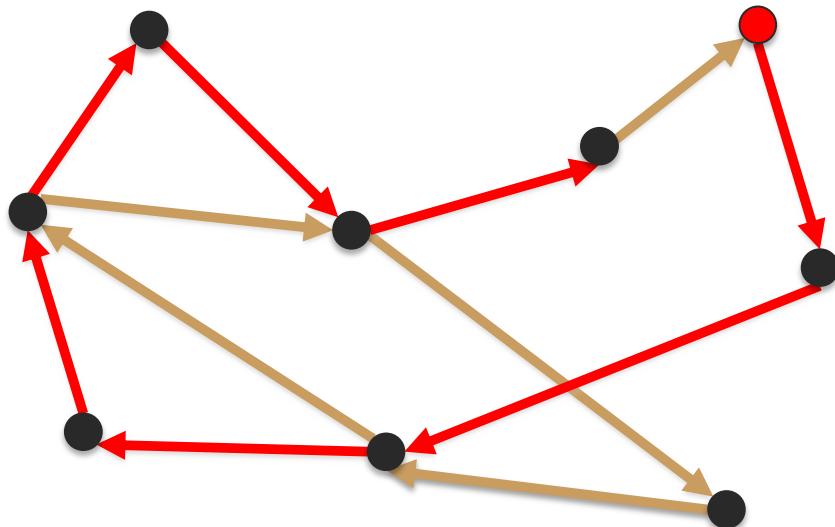


Algoritmo di costruzione del Ciclo Euleriano:  
STEP1: scomposizione del grafo in cicli

# Ciclo Euleriano

## PROOF:

(2) il grafo è bilanciato → esiste il Ciclo Euleriano

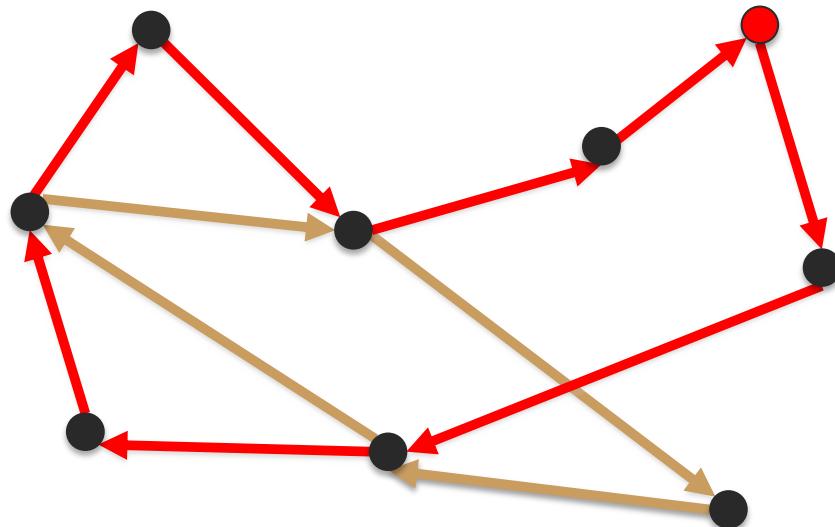


Algoritmo di costruzione del Ciclo Euleriano:  
STEP1: scomposizione del grafo in cicli

# Ciclo Euleriano

PROOF:

(2) il grafo è bilanciato → esiste il Ciclo Euleriano

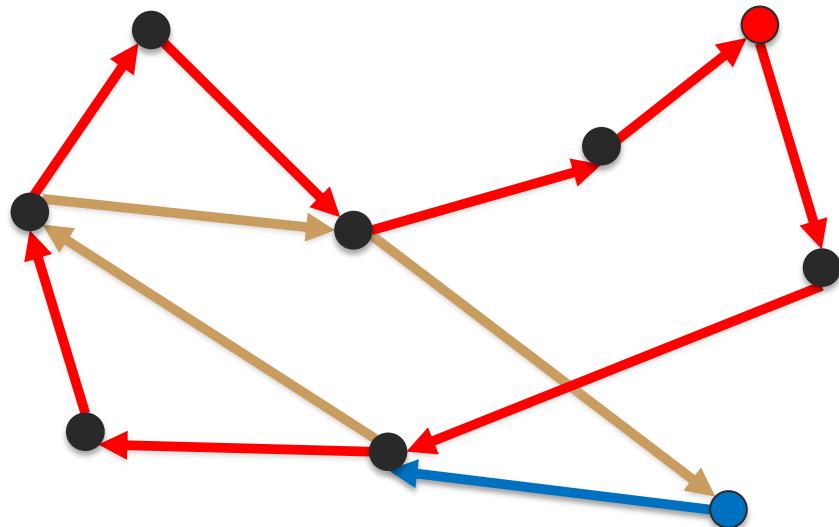


Algoritmo di costruzione del Ciclo Euleriano:  
STEP1: scomposizione del grafo in cicli

# Ciclo Euleriano

## PROOF:

(2) il grafo è bilanciato → esiste il Ciclo Euleriano

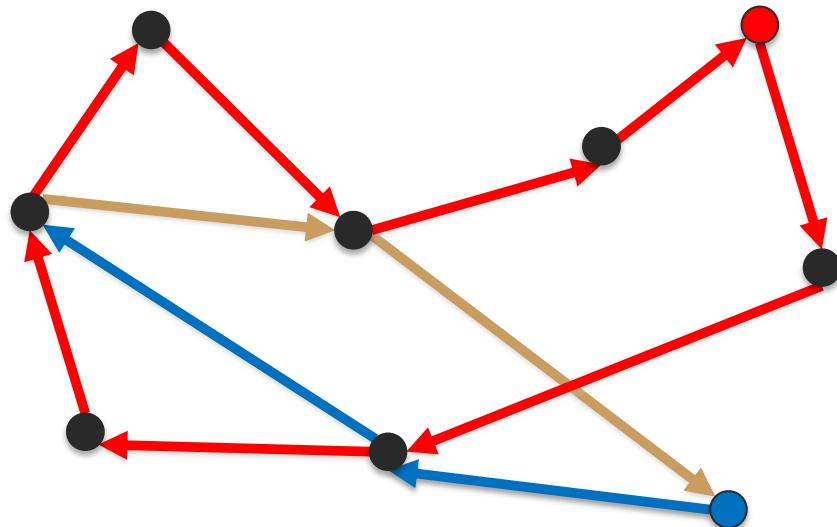


Algoritmo di costruzione del Ciclo Euleriano:  
STEP1: scomposizione del grafo in cicli

# Ciclo Euleriano

## PROOF:

(2) il grafo è bilanciato → esiste il Ciclo Euleriano

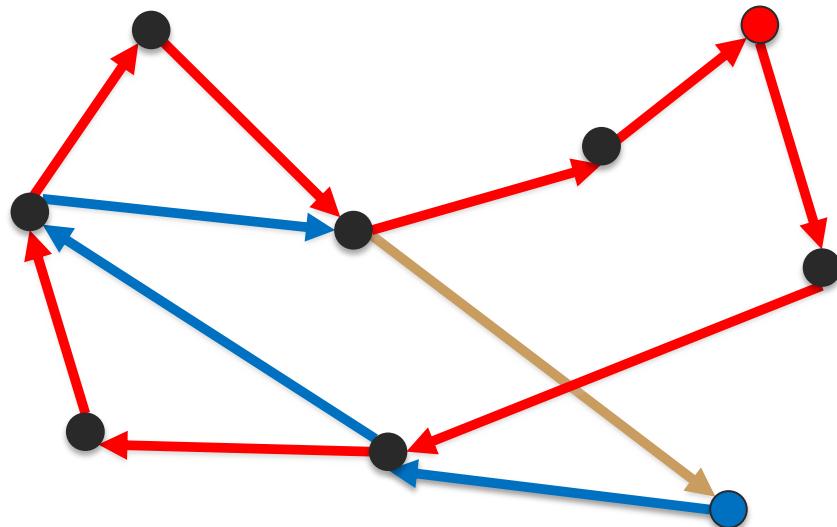


Algoritmo di costruzione del Ciclo Euleriano:  
STEP1: scomposizione del grafo in cicli

# Ciclo Euleriano

## PROOF:

(2) il grafo è bilanciato → esiste il Ciclo Euleriano

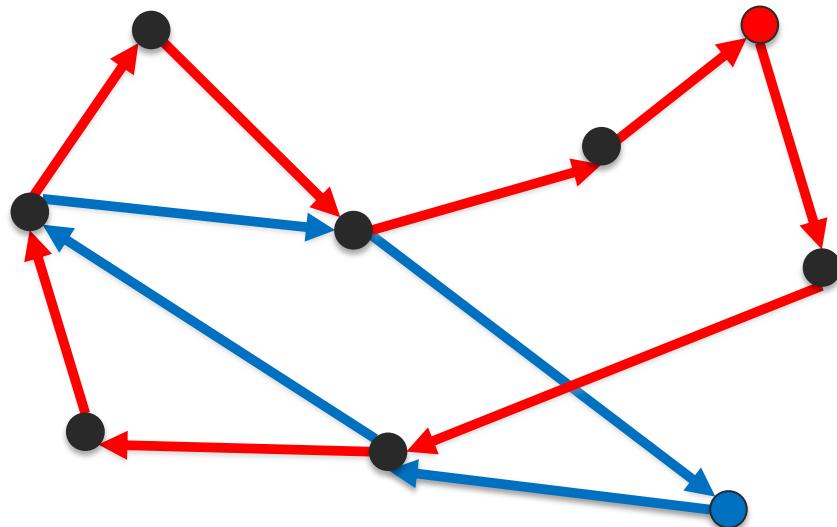


Algoritmo di costruzione del Ciclo Euleriano:  
STEP1: scomposizione del grafo in cicli

# Ciclo Euleriano

## PROOF:

(2) il grafo è bilanciato → esiste il Ciclo Euleriano

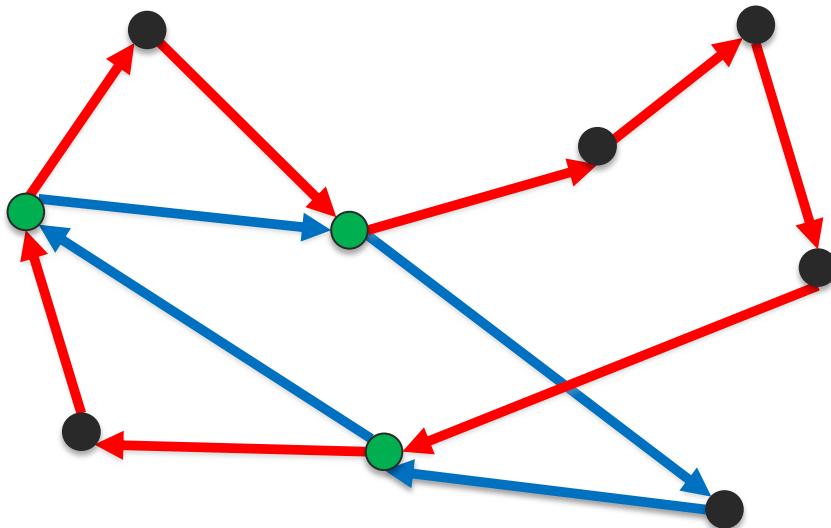


Algoritmo di costruzione del Ciclo Euleriano:  
STEP1: scomposizione del grafo in cicli

# Ciclo Euleriano

## PROOF:

(2) il grafo è bilanciato → esiste il Ciclo Euleriano

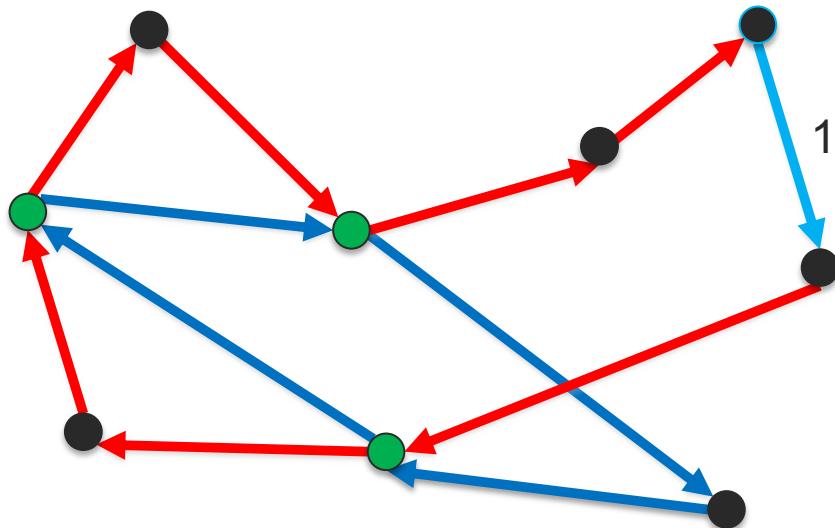


Algoritmo di costruzione del Ciclo Euleriano:  
STEP2: composizione dei cicli nel Ciclo Euleriano

# Ciclo Euleriano

## PROOF:

(2) il grafo è bilanciato → esiste il Ciclo Euleriano

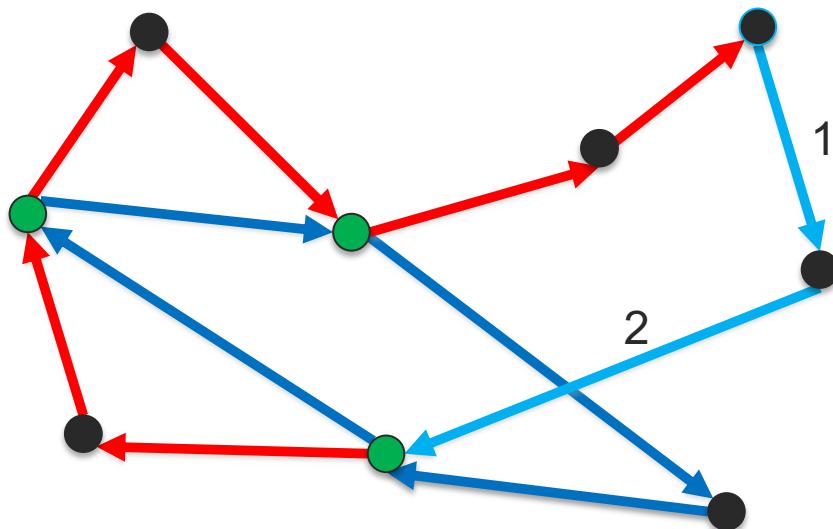


Algoritmo di costruzione del Ciclo Euleriano:  
STEP2: composizione dei cicli nel Ciclo Euleriano

# Ciclo Euleriano

## PROOF:

(2) il grafo è bilanciato → esiste il Ciclo Euleriano

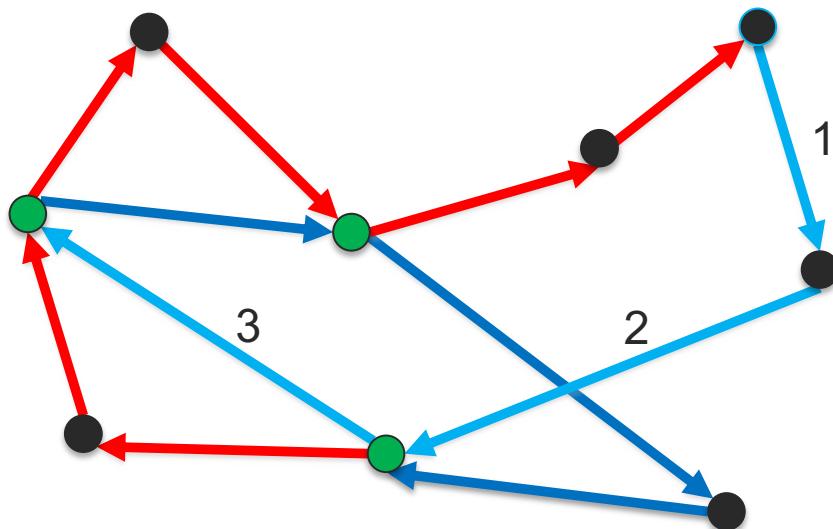


Algoritmo di costruzione del Ciclo Euleriano:  
STEP2: composizione dei cicli nel Ciclo Euleriano

# Ciclo Euleriano

## PROOF:

(2) il grafo è bilanciato → esiste il Ciclo Euleriano

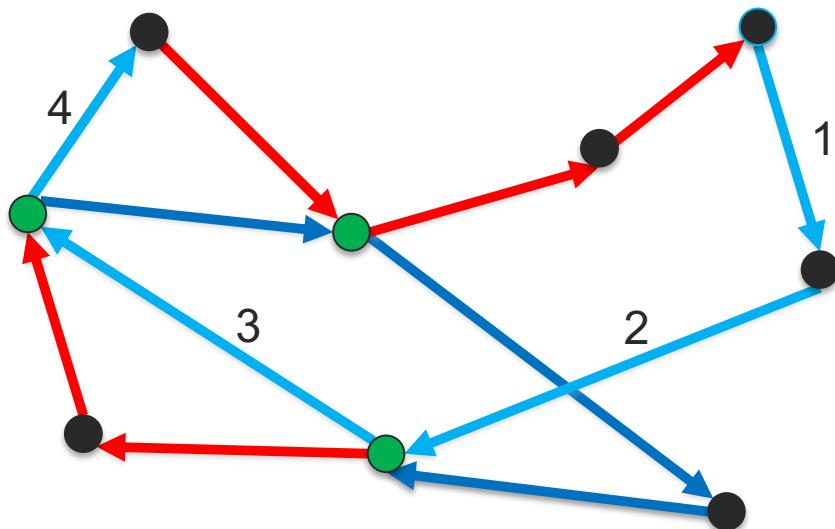


Algoritmo di costruzione del Ciclo Euleriano:  
STEP2: composizione dei cicli nel Ciclo Euleriano

# Ciclo Euleriano

PROOF:

(2) il grafo è bilanciato → esiste il Ciclo Euleriano

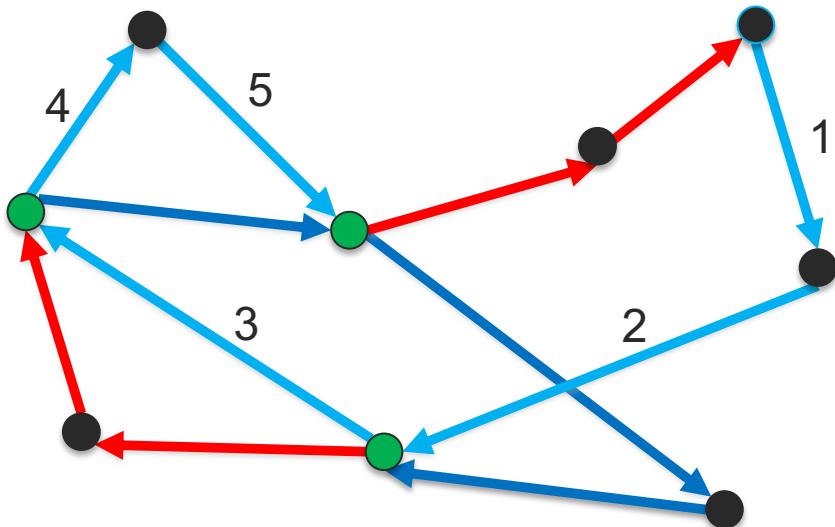


Algoritmo di costruzione del Ciclo Euleriano:  
STEP2: composizione dei cicli nel Ciclo Euleriano

# Ciclo Euleriano

## PROOF:

(2) il grafo è bilanciato → esiste il Ciclo Euleriano

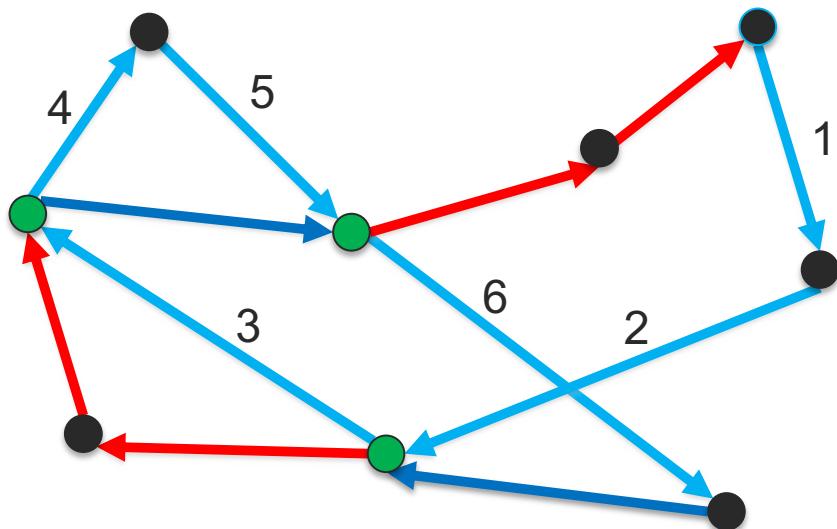


Algoritmo di costruzione del Ciclo Euleriano:  
STEP2: composizione dei cicli nel Ciclo Euleriano

# Ciclo Euleriano

## PROOF:

(2) il grafo è bilanciato → esiste il Ciclo Euleriano

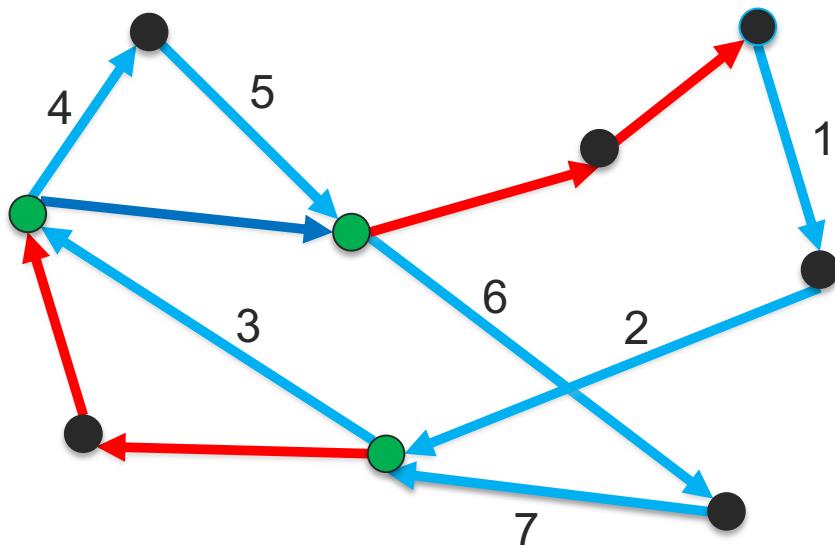


Algoritmo di costruzione del Ciclo Euleriano:  
STEP2: composizione dei cicli nel Ciclo Euleriano

# Ciclo Euleriano

PROOF:

(2) il grafo è bilanciato → esiste il Ciclo Euleriano

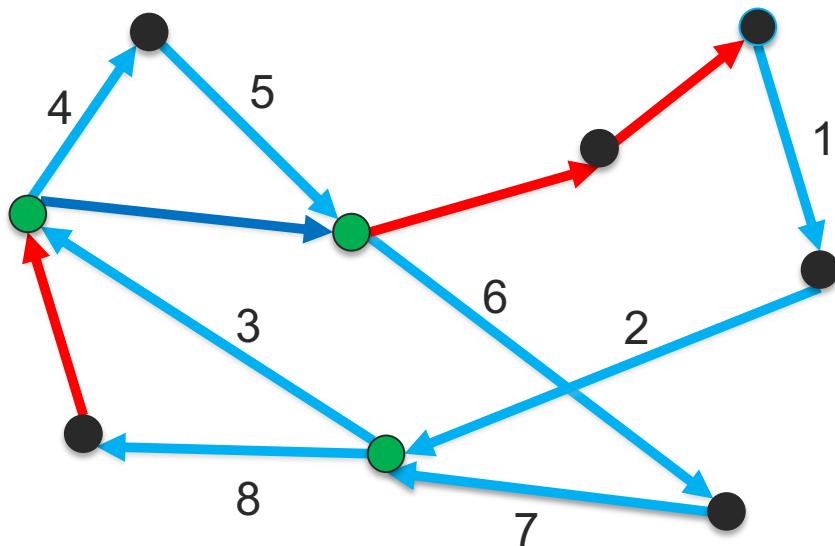


Algoritmo di costruzione del Ciclo Euleriano:  
STEP2: composizione dei cicli nel Ciclo Euleriano

# Ciclo Euleriano

PROOF:

(2) il grafo è bilanciato → esiste il Ciclo Euleriano

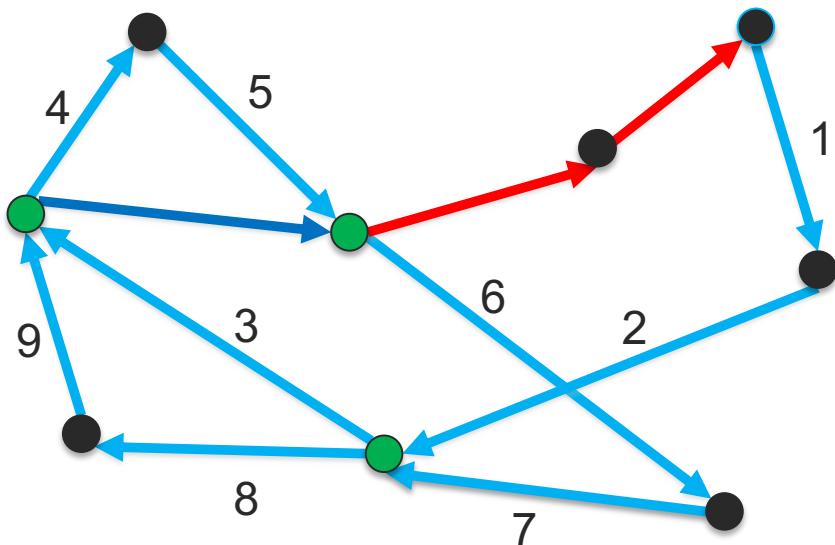


Algoritmo di costruzione del Ciclo Euleriano:  
STEP2: composizione dei cicli nel Ciclo Euleriano

# Ciclo Euleriano

## PROOF:

(2) il grafo è bilanciato → esiste il Ciclo Euleriano

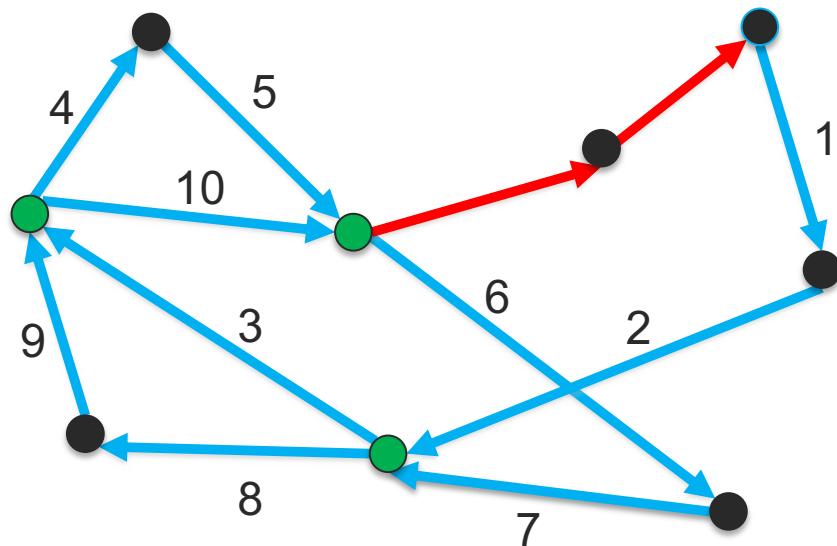


Algoritmo di costruzione del Ciclo Euleriano:  
STEP2: composizione dei cicli nel Ciclo Euleriano

# Ciclo Euleriano

PROOF:

(2) il grafo è bilanciato → esiste il Ciclo Euleriano

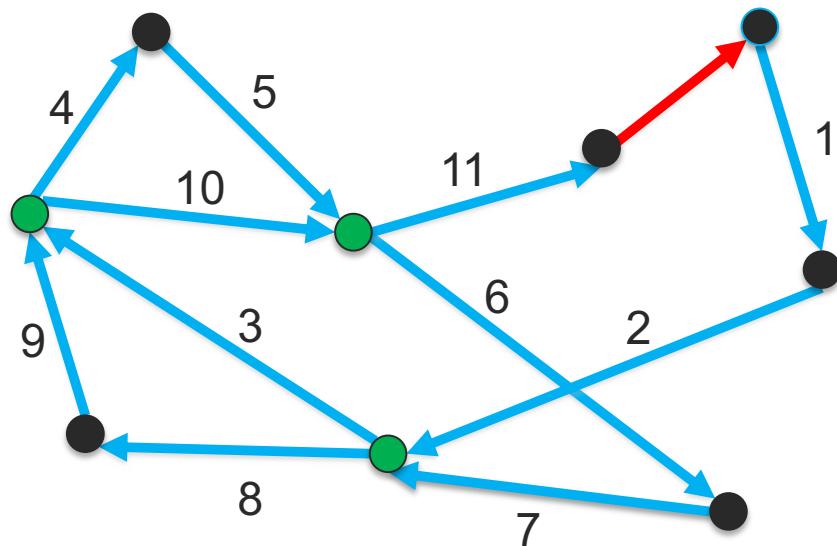


Algoritmo di costruzione del Ciclo Euleriano:  
STEP2: composizione dei cicli nel Ciclo Euleriano

# Ciclo Euleriano

PROOF:

(2) il grafo è bilanciato → esiste il Ciclo Euleriano

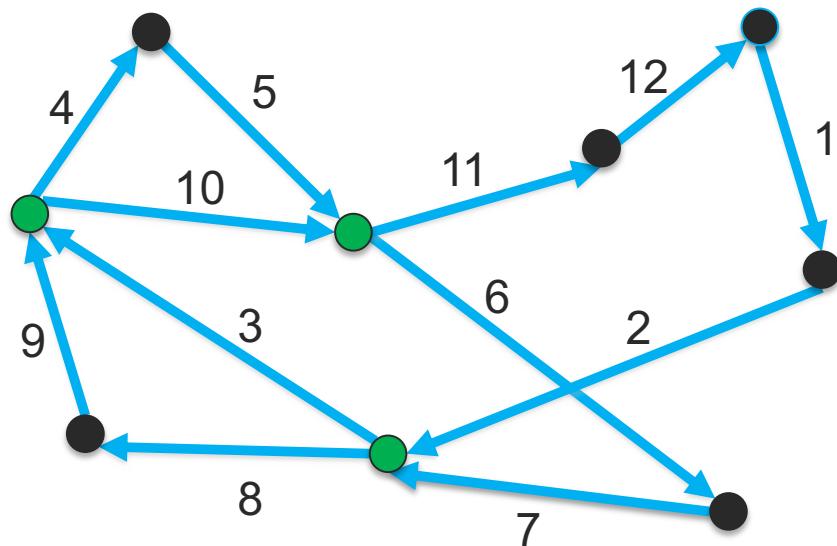


Algoritmo di costruzione del Ciclo Euleriano:  
STEP2: composizione dei cicli nel Ciclo Euleriano

# Ciclo Euleriano

PROOF:

(2) il grafo è bilanciato → esiste il Ciclo Euleriano



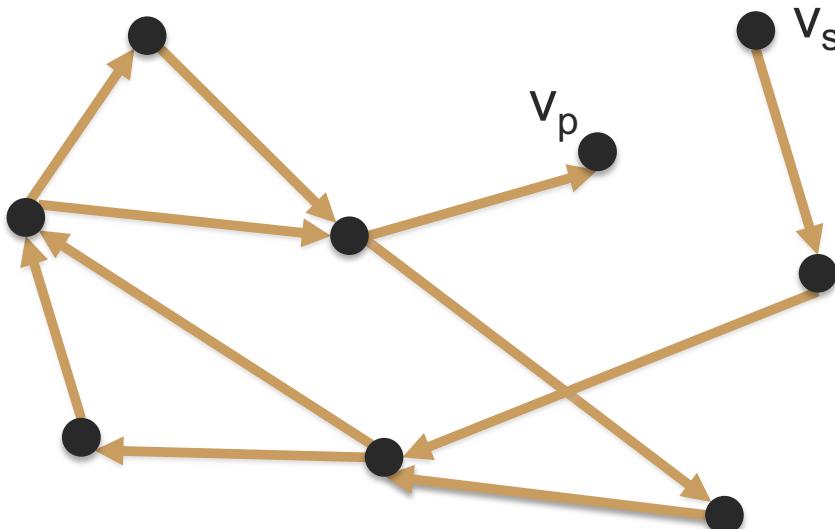
Algoritmo di costruzione del Ciclo Euleriano:  
STEP2: composizione dei cicli nel Ciclo Euleriano

# Cammino Euleriano

Cammino e non ciclo  
perchè il genoma è  
lineare...

## Grafo semi-bilanciato

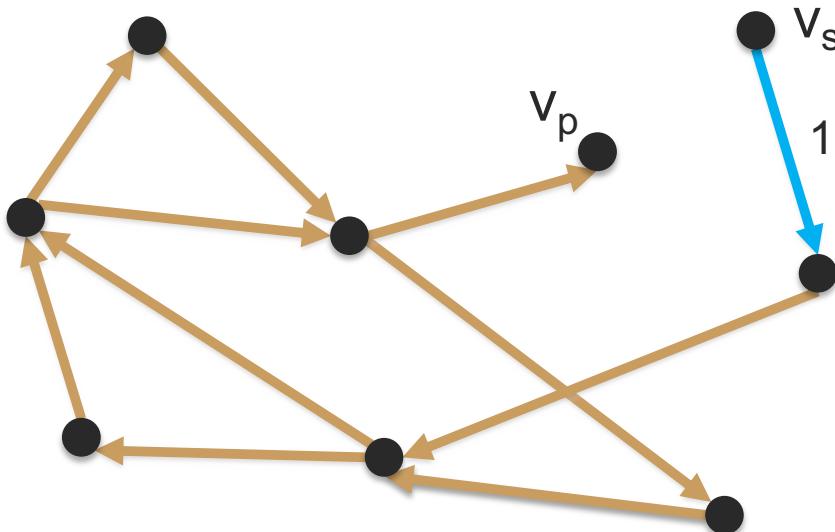
- ✓ esiste un solo nodo  $v_s$  per cui  $\text{OUT}(v_s) - \text{IN}(v_s) = 1 \rightarrow$  sorgente
- ✓ esiste un solo nodo  $v_p$  per cui  $\text{IN}(v_p) - \text{OUT}(v_p) = 1 \rightarrow$  pozzo



# Cammino Euleriano

## Grafo semi-bilanciato

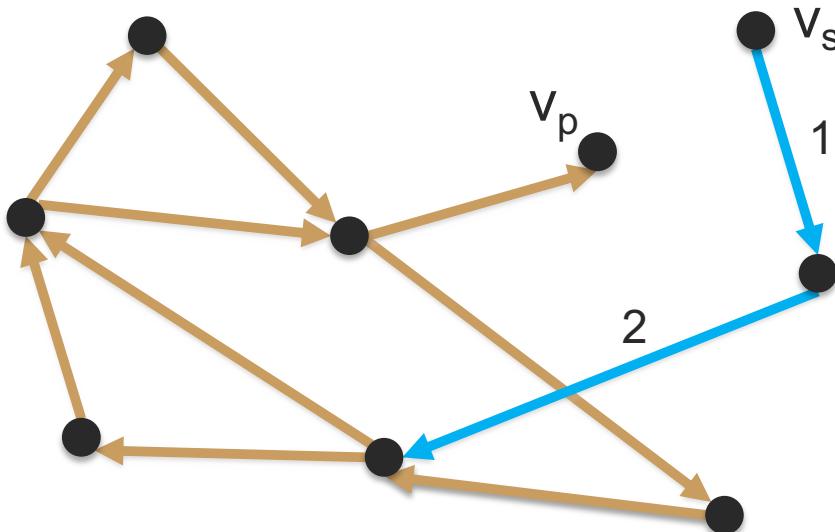
- ✓ esiste un solo nodo  $v_s$  per cui  $\text{OUT}(v_s) - \text{IN}(v_s) = 1 \rightarrow$  sorgente
- ✓ esiste un solo nodo  $v_p$  per cui  $\text{IN}(v_p) - \text{OUT}(v_p) = 1 \rightarrow$  pozzo



# Cammino Euleriano

## Grafo semi-bilanciato

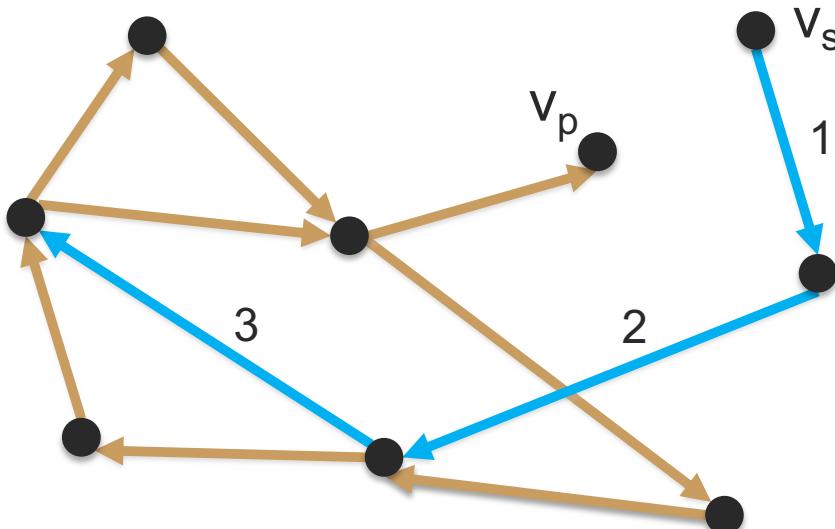
- ✓ esiste un solo nodo  $v_s$  per cui  $\text{OUT}(v_s) - \text{IN}(v_s) = 1 \rightarrow$  sorgente
- ✓ esiste un solo nodo  $v_p$  per cui  $\text{IN}(v_p) - \text{OUT}(v_p) = 1 \rightarrow$  pozzo



# Cammino Euleriano

## Grafo semi-bilanciato

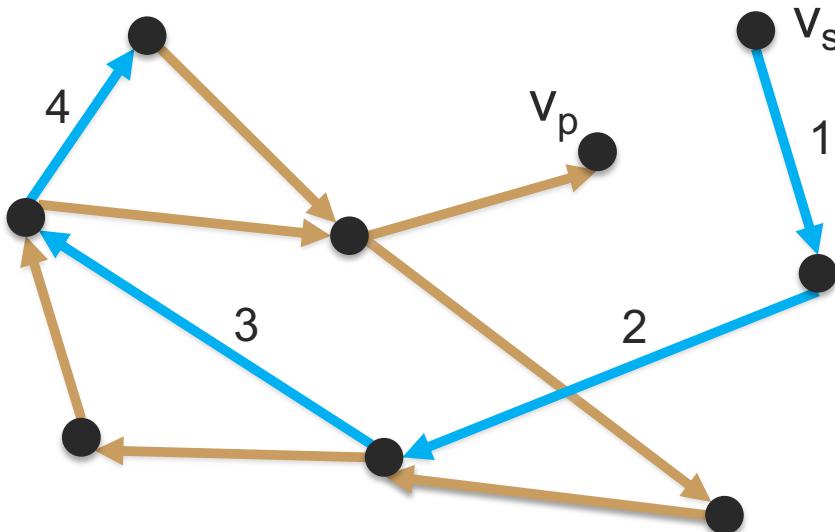
- ✓ esiste un solo nodo  $v_s$  per cui  $\text{OUT}(v_s) - \text{IN}(v_s) = 1 \rightarrow$  sorgente
- ✓ esiste un solo nodo  $v_p$  per cui  $\text{IN}(v_p) - \text{OUT}(v_p) = 1 \rightarrow$  pozzo



# [Cammino Euleriano]

## Grafo semi-bilanciato

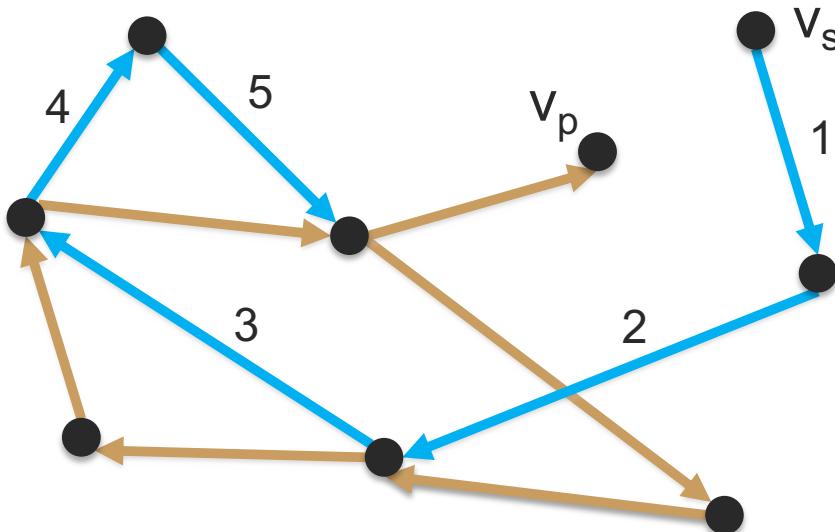
- ✓ esiste un solo nodo  $v_s$  per cui  $\text{OUT}(v_s) - \text{IN}(v_s) = 1 \rightarrow$  sorgente
- ✓ esiste un solo nodo  $v_p$  per cui  $\text{IN}(v_p) - \text{OUT}(v_p) = 1 \rightarrow$  pozzo



# Cammino Euleriano

## Grafo semi-bilanciato

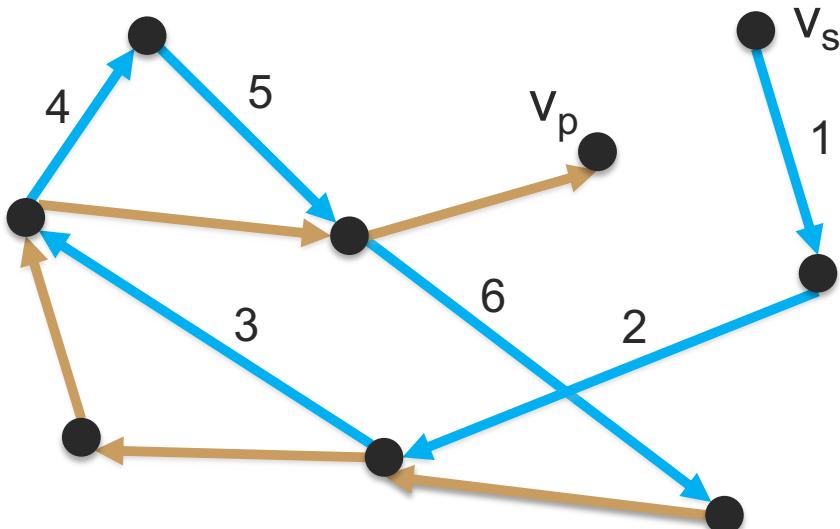
- ✓ esiste un solo nodo  $v_s$  per cui  $\text{OUT}(v_s) - \text{IN}(v_s) = 1 \rightarrow$  sorgente
- ✓ esiste un solo nodo  $v_p$  per cui  $\text{IN}(v_p) - \text{OUT}(v_p) = 1 \rightarrow$  pozzo



# Cammino Euleriano

## Grafo semi-bilanciato

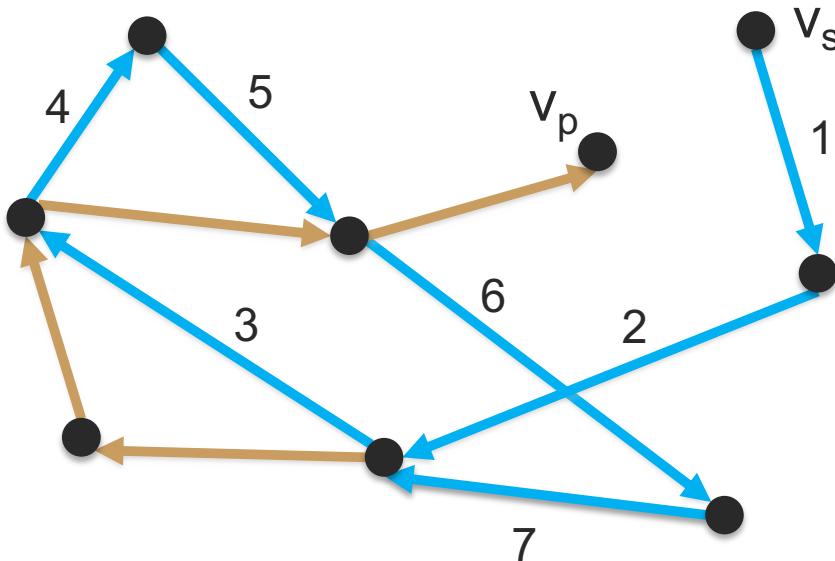
- ✓ esiste un solo nodo  $v_s$  per cui  $\text{OUT}(v_s) - \text{IN}(v_s) = 1 \rightarrow$  sorgente
- ✓ esiste un solo nodo  $v_p$  per cui  $\text{IN}(v_p) - \text{OUT}(v_p) = 1 \rightarrow$  pozzo



# Cammino Euleriano

## Grafo semi-bilanciato

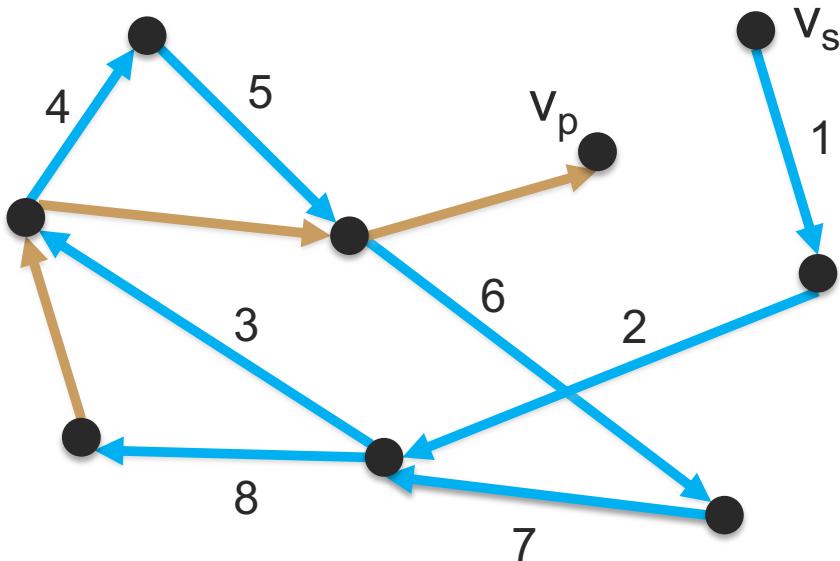
- ✓ esiste un solo nodo  $v_s$  per cui  $\text{OUT}(v_s) - \text{IN}(v_s) = 1 \rightarrow$  sorgente
- ✓ esiste un solo nodo  $v_p$  per cui  $\text{IN}(v_p) - \text{OUT}(v_p) = 1 \rightarrow$  pozzo



# Cammino Euleriano

## Grafo semi-bilanciato

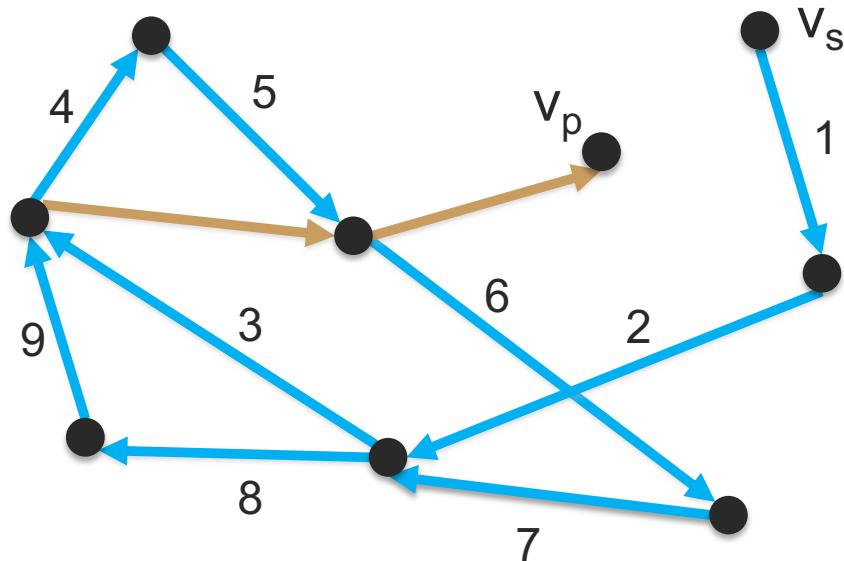
- ✓ esiste un solo nodo  $v_s$  per cui  $\text{OUT}(v_s) - \text{IN}(v_s) = 1 \rightarrow$  sorgente
- ✓ esiste un solo nodo  $v_p$  per cui  $\text{IN}(v_p) - \text{OUT}(v_p) = 1 \rightarrow$  pozzo



# Cammino Euleriano

## Grafo semi-bilanciato

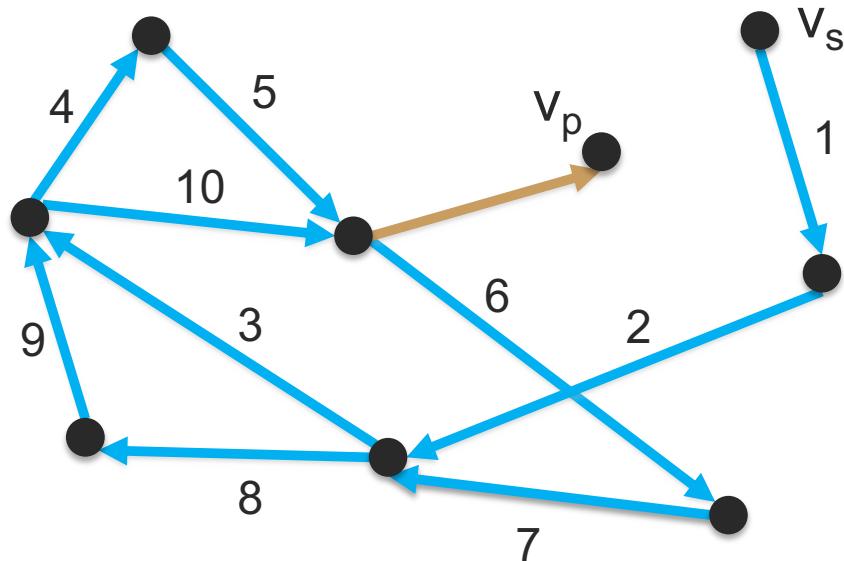
- ✓ esiste un solo nodo  $v_s$  per cui  $\text{OUT}(v_s) - \text{IN}(v_s) = 1 \rightarrow$  sorgente
- ✓ esiste un solo nodo  $v_p$  per cui  $\text{IN}(v_p) - \text{OUT}(v_p) = 1 \rightarrow$  pozzo



# Cammino Euleriano

## Grafo semi-bilanciato

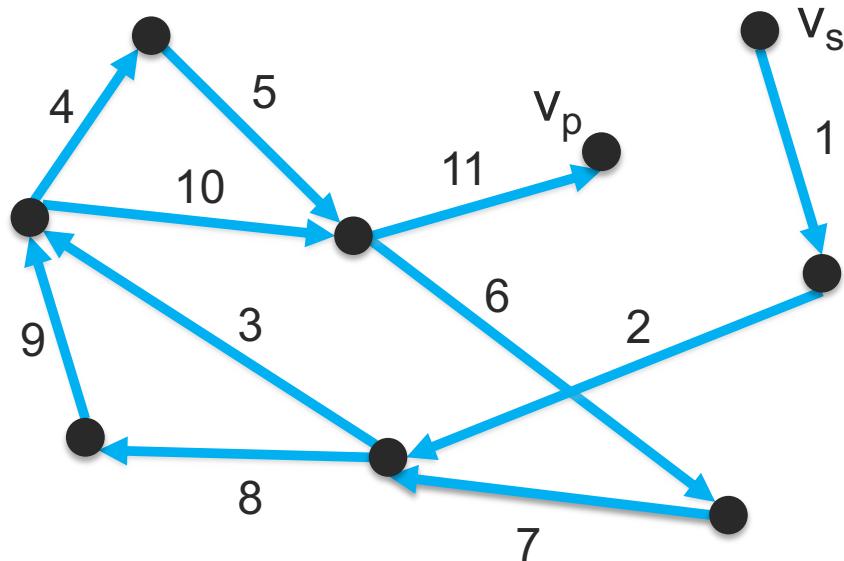
- ✓ esiste un solo nodo  $v_s$  per cui  $\text{OUT}(v_s) - \text{IN}(v_s) = 1 \rightarrow$  sorgente
- ✓ esiste un solo nodo  $v_p$  per cui  $\text{IN}(v_p) - \text{OUT}(v_p) = 1 \rightarrow$  pozzo



# Cammino Euleriano

## Grafo semi-bilanciato

- ✓ esiste un solo nodo  $v_s$  per cui  $\text{OUT}(v_s) - \text{IN}(v_s) = 1 \rightarrow$  sorgente
- ✓ esiste un solo nodo  $v_p$  per cui  $\text{IN}(v_p) - \text{OUT}(v_p) = 1 \rightarrow$  pozzo



# [de Bruijn Graph (dBG)]

(1946)

**Goal:** assemblare una sequenza di DNA attraverso la determinazione del cammino Euleriano in un grafo semi-bilanciato (sequenziamo in modo da averlo semi-bilanciato).

**DEF (k-mer):** il k-mer di una stringa  $r$  su alfabeto  $\Sigma$  è un elemento di  $\Sigma^k$  che occorre in  $r$ , cioè è una sottostringa di  $r$  di lunghezza  $k$

GTGC è un k-mer per  $k=4$  nella stringa ACGTGCCGTG

data una sequenza di lunghezza  $N$ , avrò  $N-k+1$  possibili k-mer.

Facile “overlap”...

# [de Bruijn Graph (dBG)]

**Goal:** assemblare una sequenza di DNA attraverso la determinazione del cammino Euleriano in un grafo semi-bilanciato

L'assemblaggio di due k-mers  $m_1$  e  $m_2$  che hanno overlap lungo  $k-1$  è il  $(k+1)$ -mer dato dalla concatenazione tra  $m_1$  e l'estensione di  $m_2$  rispetto all'overlap.

GTGC  
TGCA

GTGCA

# [de Bruijn Graph (dBG)]

**Goal:** assemblare una sequenza di DNA attraverso la determinazione del cammino Euleriano in un grafo semi-bilanciato

**DEF (spettro di ordine k):** lo spettro di ordine k di una stringa è il *multiset* di tutti i suoi k-mers

ACGTGCCGTG

Spettro di ordine 4:

{ACGT, CGTG, GTGC, TGCC, GCCG, CCGT, CGTG}

# [de Bruijn Graph (dBG)]

**Goal:** assemblare una sequenza di DNA attraverso la determinazione del cammino Euleriano in un grafo semi-bilanciato

**DEF (spettro di ordine k):** lo spettro di ordine k di una stringa è il *multiset* di tutti i suoi k-mers

ACGTGCCGTG

Spettro di ordine 4:

{ACGT, CGTG, GTGC, TGCC, GCCG, CCGT, CGTG}

# [de Bruijn Graph (dBG)]

**Goal:** assemblare una sequenza di DNA attraverso la determinazione del cammino Euleriano in un grafo semi-bilanciato

**DEF (spettro di ordine k):** lo spettro di ordine k di una collezione di stringhe è il *multiset* di tutti i k-mers delle stringhe della collezione

ACGTGCCGTG, CCGTCGGTAA

Spettro di ordine 4:

{ACGT, CGTG, GTGC, TGCC, GCCG, CCGT, CGTG  
CCGT, CGTC, GTCG, TCGG, CGGT, GGTA, GTAA}

# [de Bruijn Graph (dBG)]

**Goal:** assemblare una sequenza di DNA attraverso la determinazione del cammino Euleriano in un grafo semi-bilanciato

**DEF (spettro di ordine k):** lo spettro di ordine k di una collezione di stringhe è il *multiset* di tutti i k-mers delle stringhe della collezione

ACGTGCCGTG, CCGTCGGTAA

Spettro di ordine 4:

{ACGT, CGTG, TGCG, TGCC, GCCG, CCGT, CGTG, CCGT, CGTC, GTCG, TCGG, CGGT, GGTA, GTAA}

# [de Bruijn Graph (dBG)]

**Goal:** assemblare una sequenza di DNA attraverso la determinazione del cammino Euleriano in un grafo semi-bilanciato

**DEF (spettro di ordine k):** lo spettro di ordine k di una collezione di stringhe è il *multiset* di tutti i k-mers delle stringhe della collezione

ACGTGCCGTG, CCGTCGGTAA

Spettro di ordine 4:

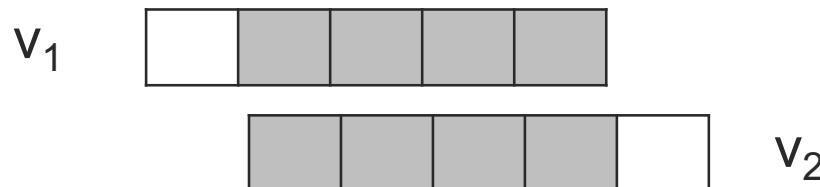
{ACGT, CGTG, GTGC, TGCC, GCCG, CCGT, CGTG  
CCGT, CGTC, GTCG, TCGG, CGGT, GGTA, GTAA}

# [ de Bruijn Graph (dBG) ]

## de Bruijn Graph di ordine k (*node-centric*)

Il de Bruijn Graph di una collezione di reads  $\{r_1, r_2, \dots, r_n\}$  è il grafo  $G = (V, A)$  dove:

- ✓  $V$  è l'insieme degli elementi dello spettro di ordine  $k$
- ✓ esiste un arco dal nodo  $v_1$  al nodo  $v_2$  se e solo se:
  1. il suffisso di  $v_1$  lungo  $k-1$  è uguale a prefisso di  $v_2$  lungo  $k-1$  (overlap)

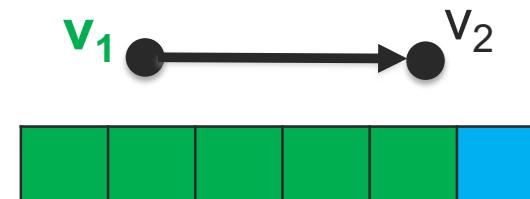
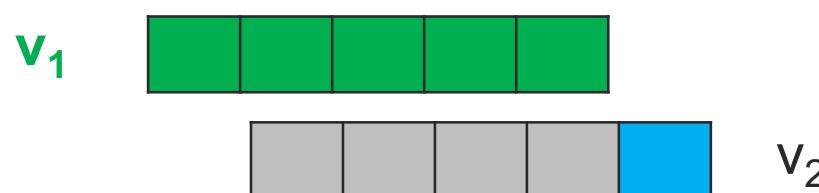


# [de Bruijn Graph (dBG)]

## de Bruijn Graph di ordine k (*node-centric*)

Il de Bruijn Graph di una collezione di reads  $\{r_1, r_2, \dots, r_n\}$  è il grafo  $G = (V, A)$  dove:

- ✓  $V$  è l'insieme degli elementi dello spettro di ordine  $k$
- ✓ esiste un arco dal nodo  $v_1$  al nodo  $v_2$  se e solo se:
  1. il suffisso di  $v_1$  lungo  $k-1$  è uguale a prefisso di  $v_2$  lungo  $k-1$  (overlap)
  2. il  $(k+1)$ -mer ottenuto assemblando  $v_1$  e  $v_2$  è sottostringa in almeno un read della collezione

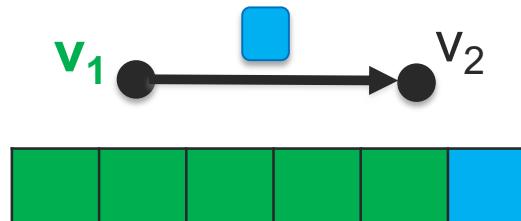
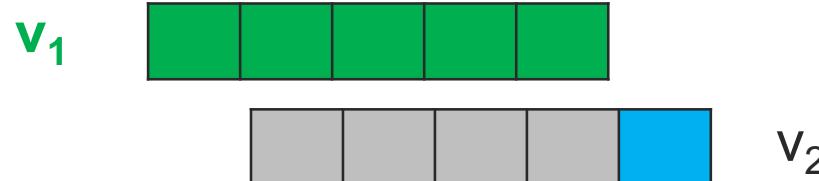


# [de Bruijn Graph (dBG)]

## de Bruijn Graph di ordine k (*node-centric*)

Il de Bruijn Graph di una collezione di reads  $\{r_1, r_2, \dots, r_n\}$  è il grafo  $G = (V, A)$  dove:

- ✓  $V$  è l'insieme degli elementi dello spettro di ordine  $k$
- ✓ esiste un arco dal nodo  $v_1$  al nodo  $v_2$  se e solo se:
  1. il suffisso di  $v_1$  lungo  $k-1$  è uguale a prefisso di  $v_2$  lungo  $k-1$  (overlap)
  2. il  $(k+1)$ -mer ottenuto assemblando  $v_1$  e  $v_2$  è sottostringa in almeno un read della collezione

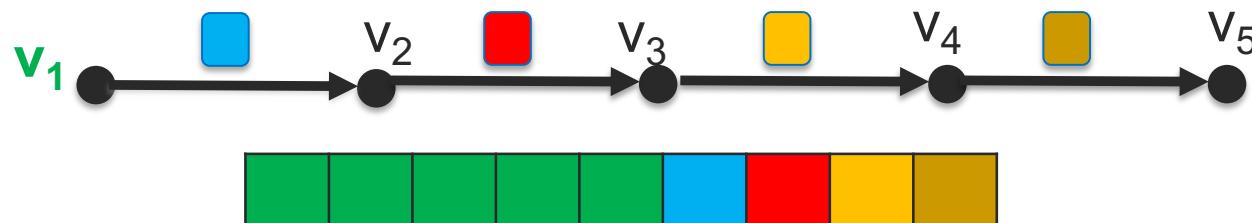


# [de Bruijn Graph (dBG)]

## de Bruijn Graph di ordine k (*node-centric*)

Il de Bruijn Graph di una collezione di reads  $\{r_1, r_2, \dots, r_n\}$  è il grafo  $G = (V, A)$  dove:

- ✓  $V$  è l'insieme degli elementi dello spettro di ordine  $k$
- ✓ esiste un arco dal nodo  $v_1$  al nodo  $v_2$  se e solo se:
  1. il suffisso di  $v_1$  lungo  $k-1$  è uguale a prefisso di  $v_2$  lungo  $k-1$  (overlap)
  2. il  $(k+1)$ -mer ottenuto assemblando  $v_1$  e  $v_2$  è sottostringa in almeno un read della collezione



# [ de Bruijn Graph (dBG) ]

de Bruijn Graph di ordine k (*node-centric*)

ATCGATAGATGG                            k = 5  
    AGATGGAAAG

Spettro di ordine 5:

{ATCGA, TCGAT, CGATA, GATAG, ATAGA, TAGAT, AGATG,  
GATGG,  
AGATG, GATGG, ATGGA, TGGAA, GGAAG}

# [ de Bruijn Graph (dBG) ]

de Bruijn Graph di ordine k (*node-centric*)

ATCGATAGATGG                    k = 5  
                                    AGATGGAAAG

# [ de Bruijn Graph (dBG) ]

de Bruijn Graph di ordine k (*node-centric*)

ATCGATAGATGG                            k = 5  
    AGATGGAAAG

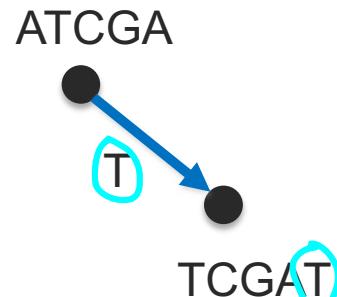
ATCGA



# [de Bruijn Graph (dBG)]

de Bruijn Graph di ordine k (*node-centric*)

ATCGATAGATGG                            k = 5  
    AGATGGAAAG

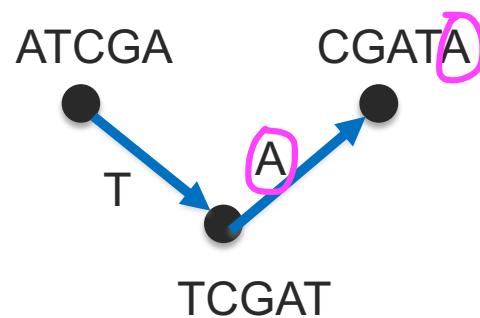


ATCGAT → 6-mer

# [de Bruijn Graph (dBG)]

de Bruijn Graph di ordine k (*node-centric*)

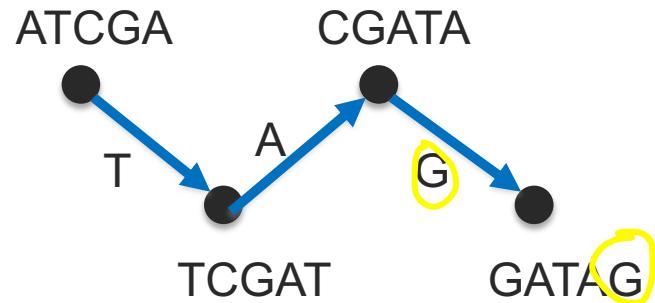
ATCGATAGATGG                    k = 5  
                                        AGATGGAAAG



# [de Bruijn Graph (dBG)]

de Bruijn Graph di ordine k (*node-centric*)

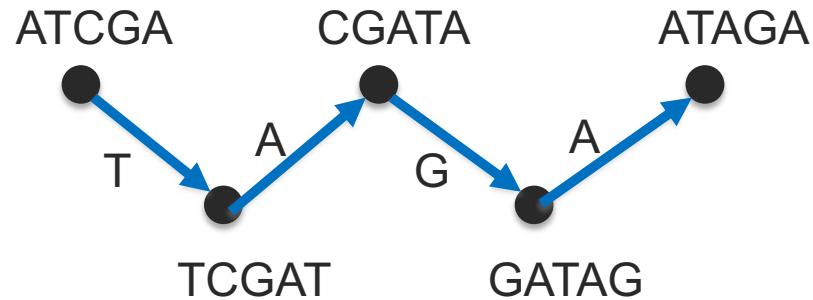
ATCGGATAGATGG                    k = 5  
    AGATGGAAAG



# [de Bruijn Graph (dBG)]

de Bruijn Graph di ordine k (*node-centric*)

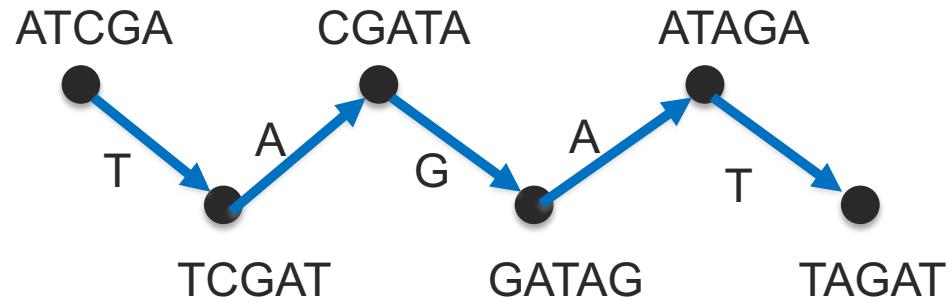
ATCGGATAGATGG                    k = 5  
    AGATGGAAAG



# [de Bruijn Graph (dBG)]

de Bruijn Graph di ordine k (*node-centric*)

ATCGATAGATGG                            k = 5  
    AGATGGAAAG



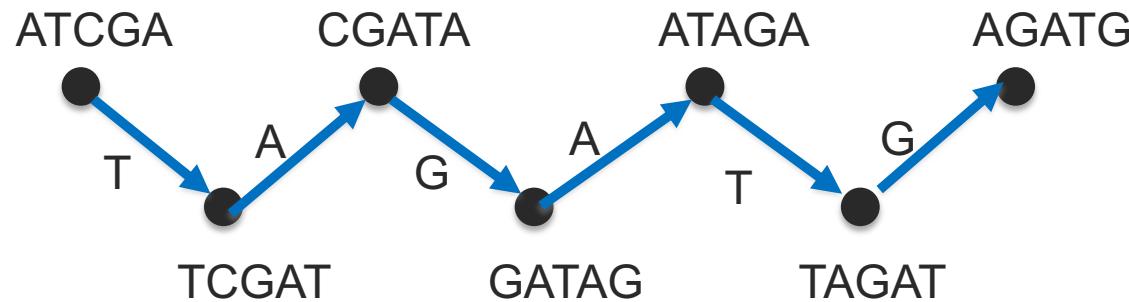
# [de Bruijn Graph (dBG)]

de Bruijn Graph di ordine k (*node-centric*)

ATCGGATAGATGG

k = 5

AGATGGAAAG



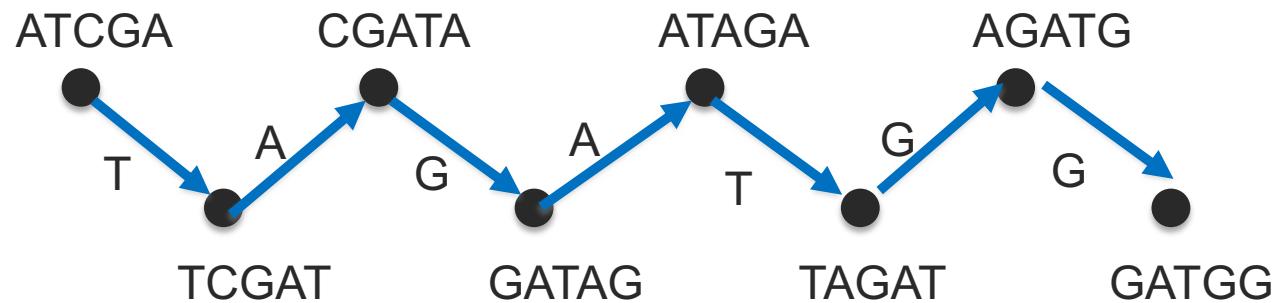
# [de Bruijn Graph (dBG)]

de Bruijn Graph di ordine k (*node-centric*)

ATCGATAGATGG

k = 5

AGATGGAAAG



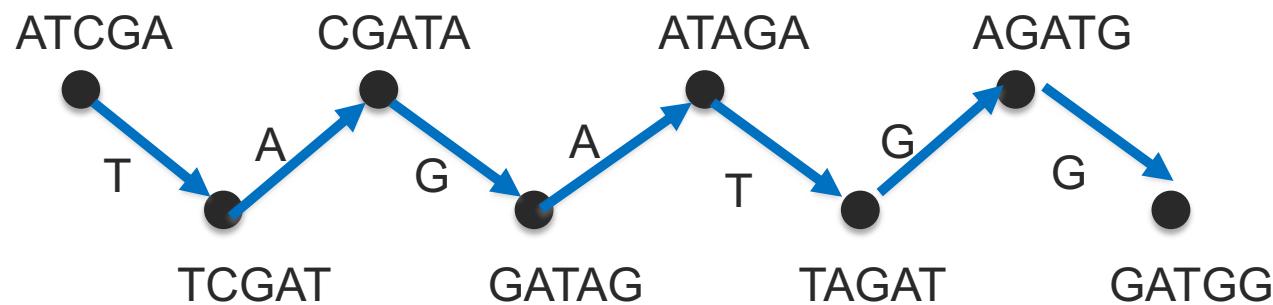
# [de Bruijn Graph (dBG)]

de Bruijn Graph di ordine k (*node-centric*)

ATCGATAGATGG

k = 5

AGATGGAAAG



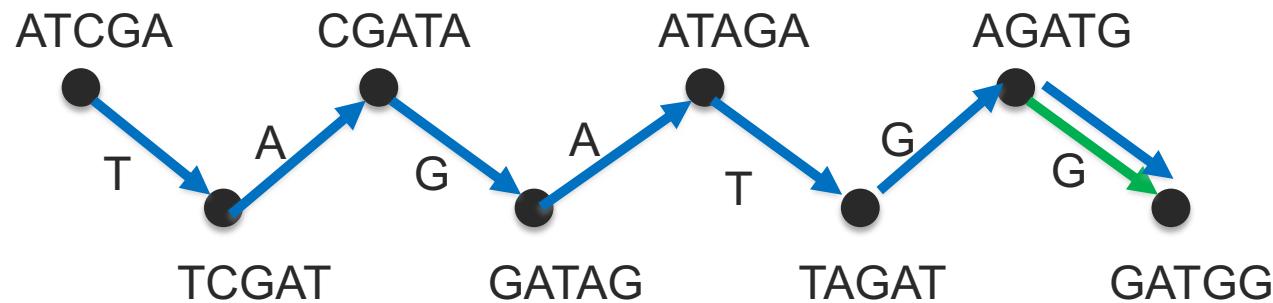
# [de Bruijn Graph (dBG)]

de Bruijn Graph di ordine k (*node-centric*)

ATCGATAGATGG

k = 5

AGATGGAAAG



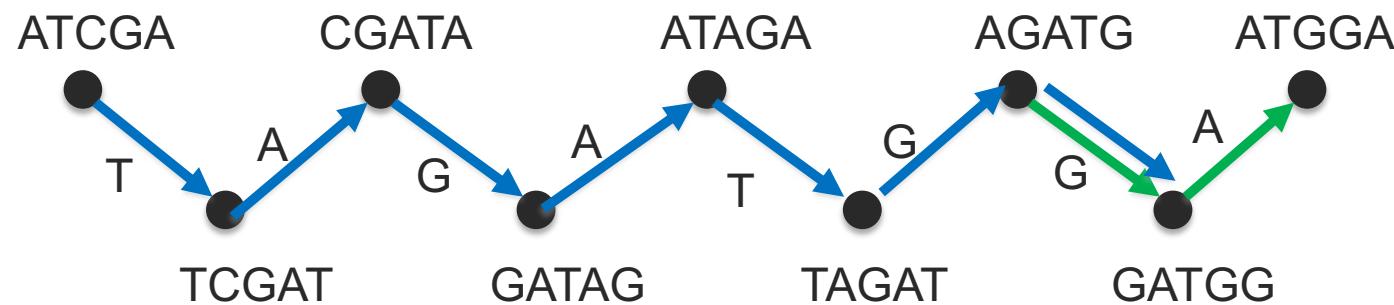
# [de Bruijn Graph (dBG)]

de Bruijn Graph di ordine k (*node-centric*)

ATCGATAGATGG

k = 5

AGATGGAAAG



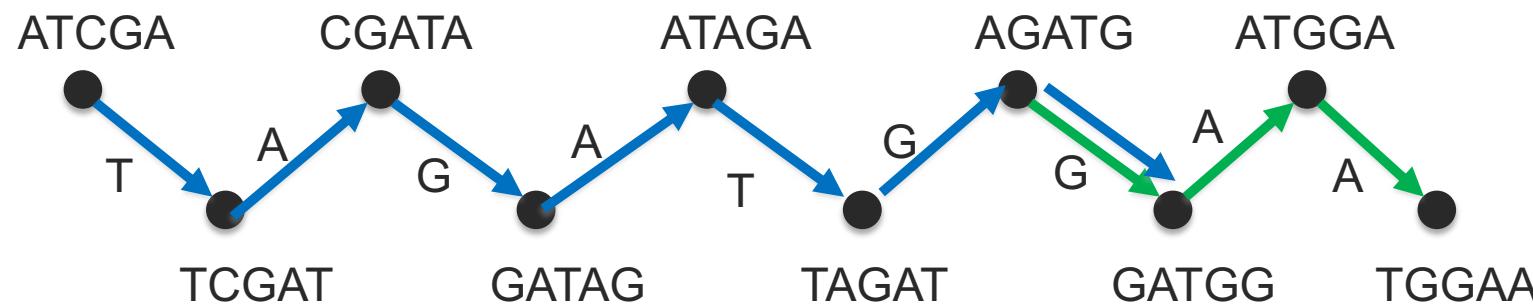
# [de Bruijn Graph (dBG)]

de Bruijn Graph di ordine k (*node-centric*)

ATCGATAGATGG

k = 5

AGATGGAAAG



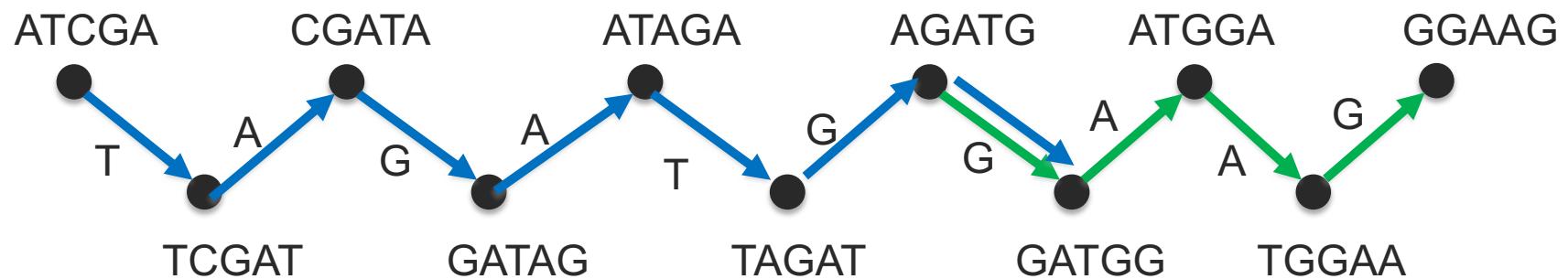
# [de Bruijn Graph (dBG)]

de Bruijn Graph di ordine k (*node-centric*)

ATCGATAGATGG

k = 5

AGATGGAAAG



idea opposta a OLG: qui spezziamo per poi ricomporre (è più veloce  
assemblare dopo...)

# [de Bruijn Graph (dBG)]

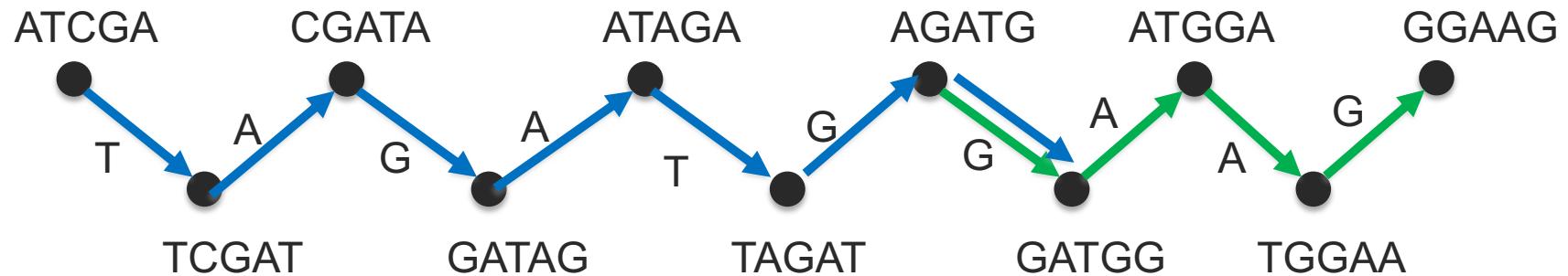
de Bruijn Graph di ordine k (*node-centric*)

ATCGATAGATGG

k = 5

AGATGGAAAG

NB: multigrafo



# [de Bruijn Graph (dBG)]

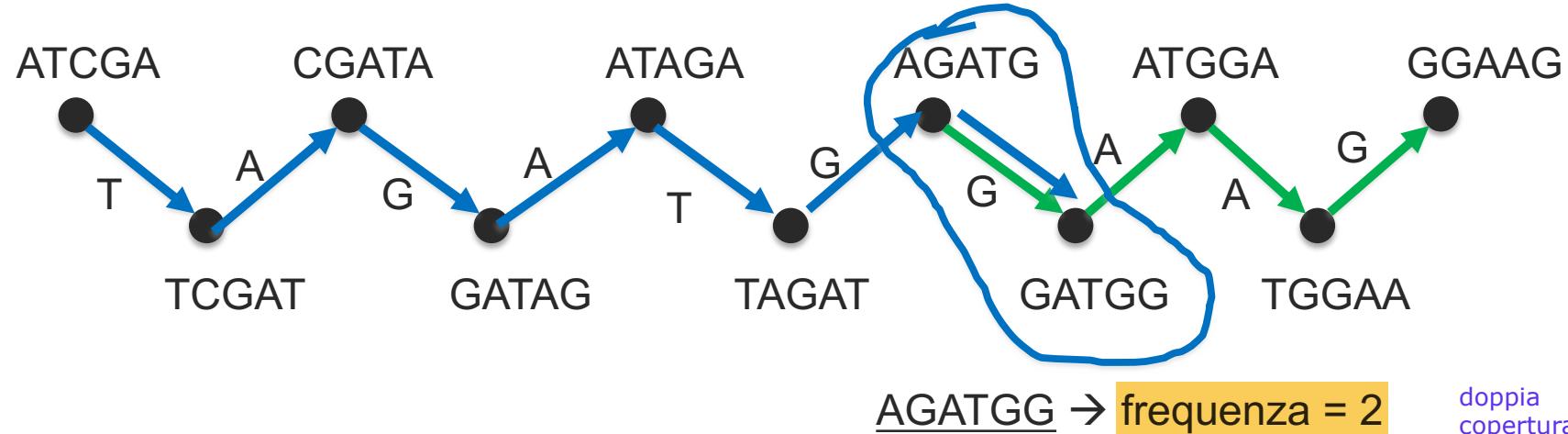
de Bruijn Graph di ordine k (*node-centric*)

ATCGGATAGATGG

k = 5

AGATGGAAG

NB: multigrafo



ma potrebbe rappresentare solo una ripetizione “altrove”... scelta del k opportuno (tipicamente 30-40) se troppo piccolo ho ambiguità, se troppo grande ho G sconnesso

# [de Bruijn Graph (dBG)]

de Bruijn Graph di ordine k (*node-centric*)

ATCGATAGATGG

k = 5

AGATGGAAAG

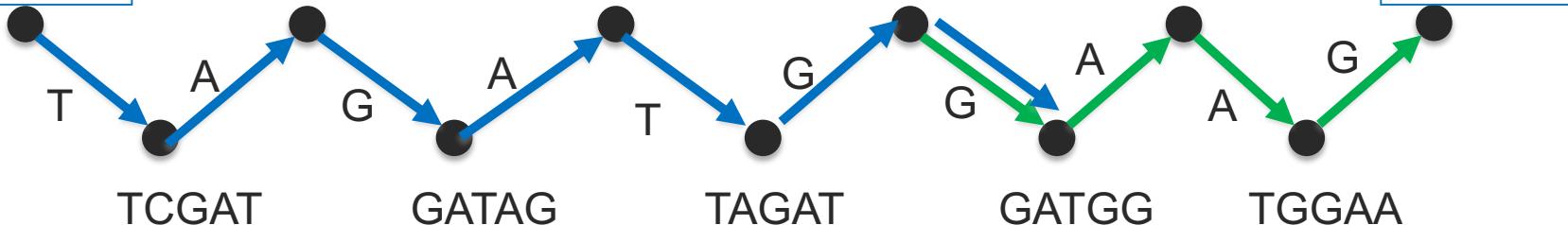
semi-bilanciato

Bilanciato

gli archi multipli  
non contano  
nel degree

sorgente

ATCGA



# [de Bruijn Graph (dBG)]

de Bruijn Graph di ordine k (*node-centric*)

ATCGATAGATGG

k = 5

AGATGGAAAG

ATCGA

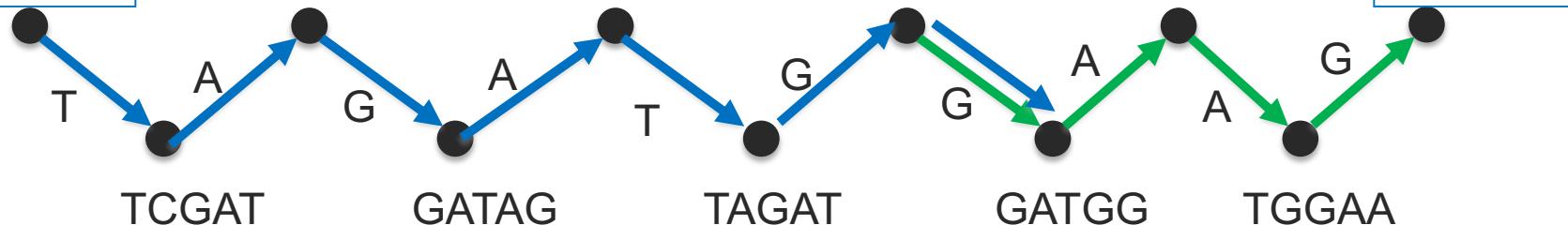
Bilanciato

sorgente

ATCGA

pozzo

GGAAG



# [de Bruijn Graph (dBG)]

de Bruijn Graph di ordine k (*node-centric*)

ATCGATAGATGG

k = 5

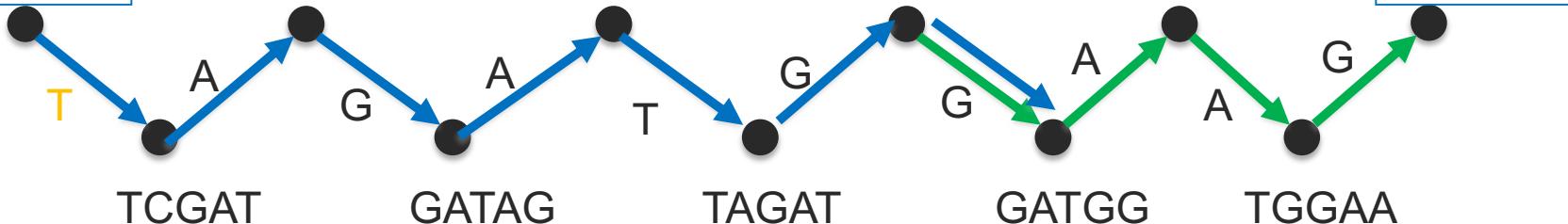
AGATGGAAAG

ATCGAT

Bilanciato

sorgente

ATCGA



# [de Bruijn Graph (dBG)]

de Bruijn Graph di ordine k (*node-centric*)

ATCGATAGATGG

k = 5

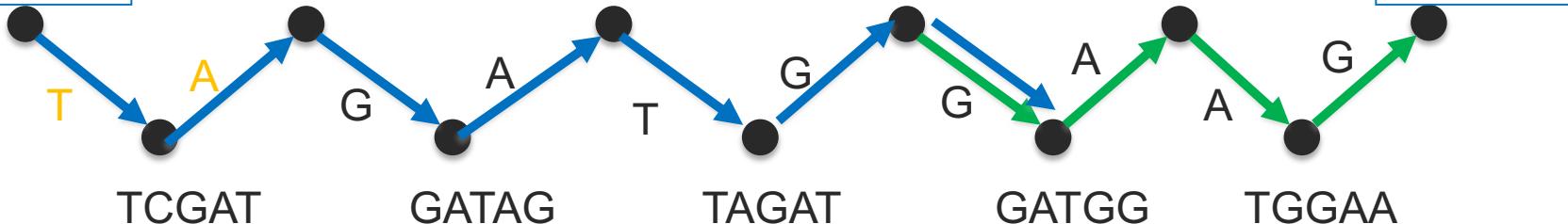
AGATGGAAAG

ATCGATA

Bilanciato

sorgente

ATCGA



# [de Bruijn Graph (dBG)]

de Bruijn Graph di ordine k (*node-centric*)

ATCGATAGATGG

k = 5

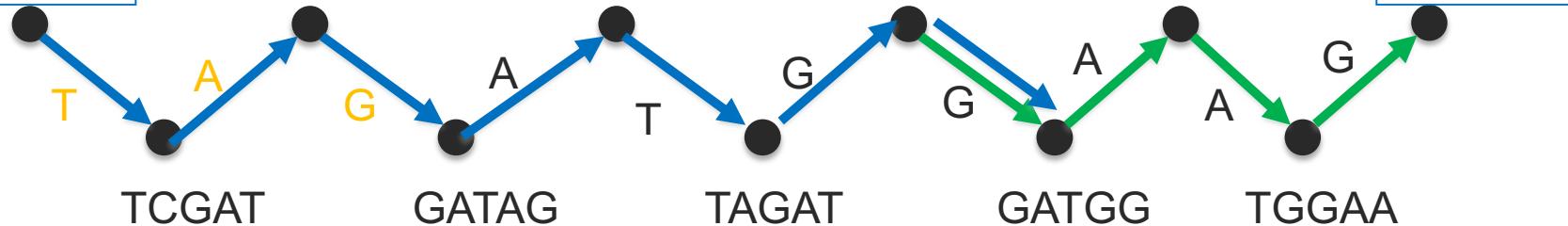
AGATGGAAAG

ATCGA**TAG**

Bilanciato

sorgente

ATCGA



pozzo

GGAAG

# [de Bruijn Graph (dBG)]

de Bruijn Graph di ordine k (*node-centric*)

ATCGATAGATGG

k = 5

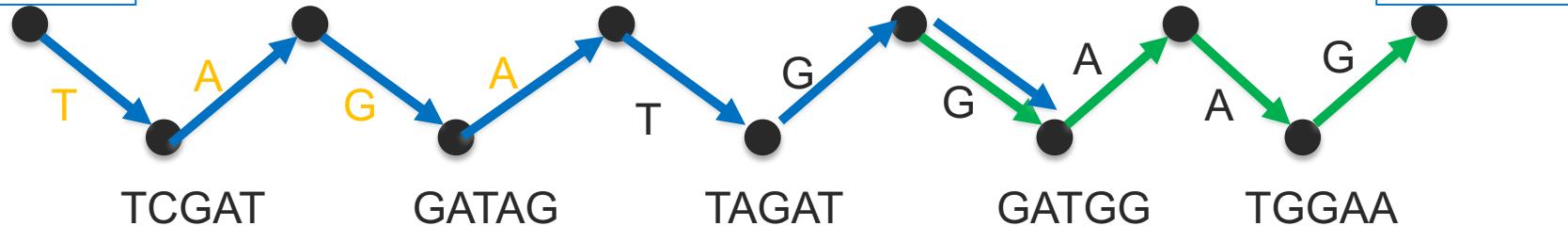
AGATGGAAAG

ATCGATAGA

Bilanciato

sorgente

ATCGA



# [de Bruijn Graph (dBG)]

de Bruijn Graph di ordine k (*node-centric*)

ATCGATAGATGG

k = 5

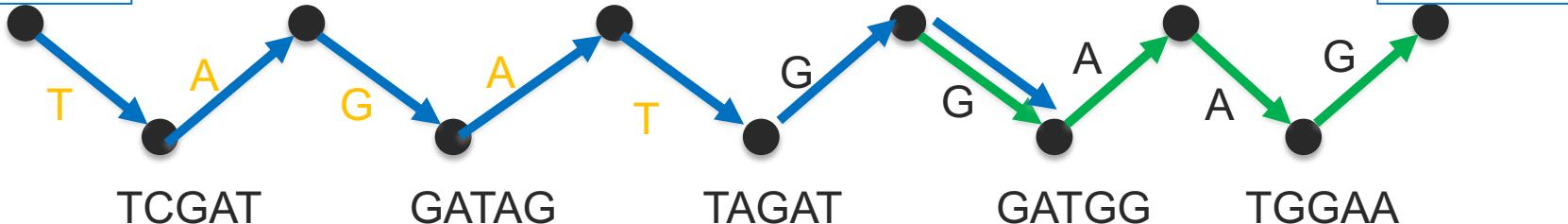
AGATGGAAAG

ATCGA**TAGAT**

Bilanciato

sorgente

ATCGA



pozzo

GGAAG

# [de Bruijn Graph (dBG)]

de Bruijn Graph di ordine k (*node-centric*)

ATCGATAGATGG

k = 5

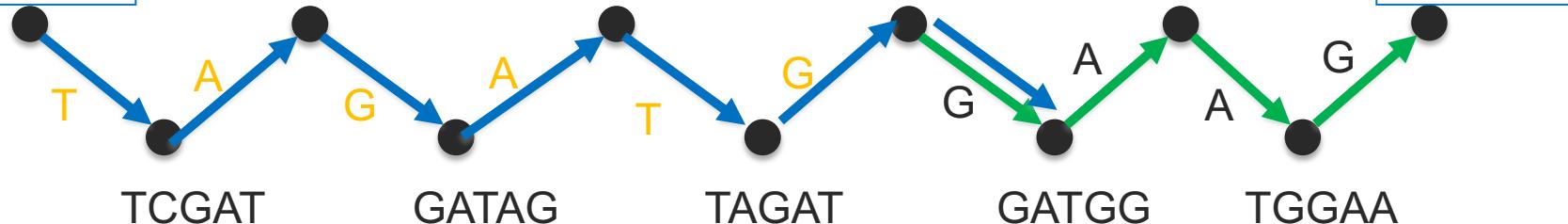
AGATGGAAAG

ATCGATAGATG

Bilanciato

sorgente

ATCGA



# [de Bruijn Graph (dBG)]

de Bruijn Graph di ordine k (*node-centric*)

ATCGATAGATGG

k = 5

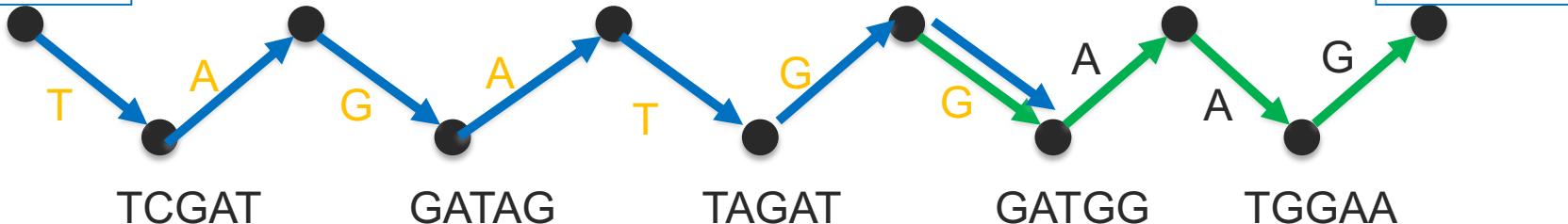
AGATGGAAAG

ATCGATAGATGG

Bilanciato

sorgente

ATCGA



# [de Bruijn Graph (dBG)]

de Bruijn Graph di ordine k (*node-centric*)

ATCGATAGATGG

k = 5

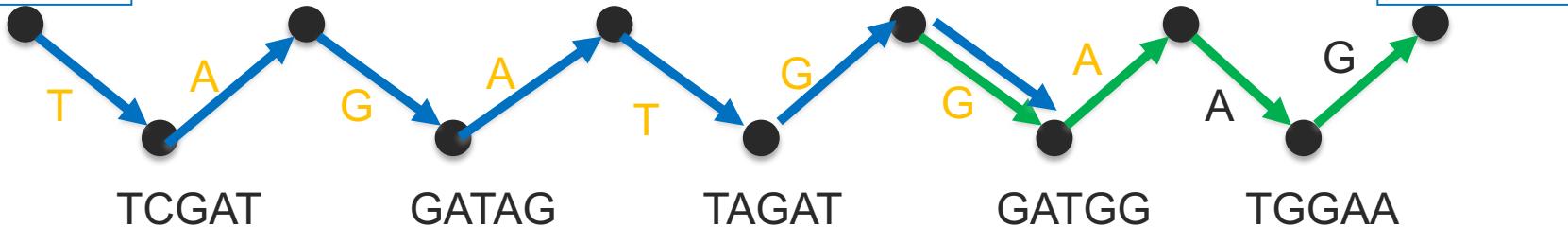
AGATGGAAAG

ATCGATAGATGGA

Bilanciato

sorgente

ATCGA



# [de Bruijn Graph (dBG)]

de Bruijn Graph di ordine k (*node-centric*)

ATCGATAGATGG

k = 5

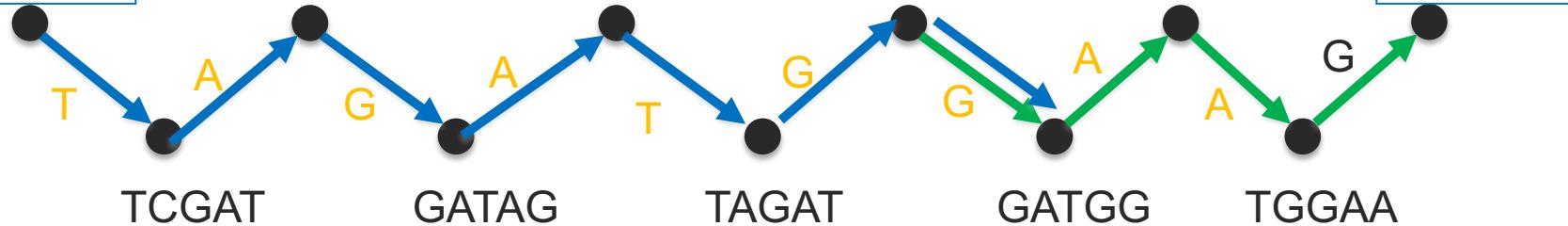
AGATGGAAAG

ATCGATAGATGGAA

Bilanciato

sorgente

ATCGA



# [de Bruijn Graph (dBG)]

de Bruijn Graph di ordine k (*node-centric*)

ATCGATAGATGG

k = 5

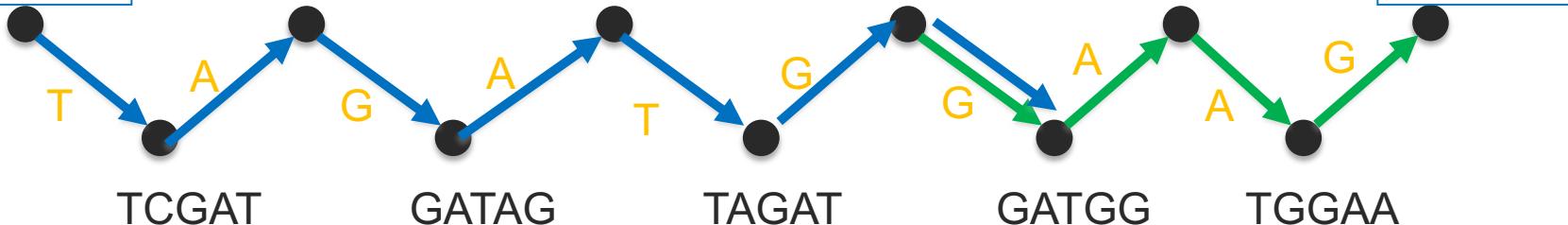
AGATGGAAAG

ATCGATAGATGGAAAG

Bilanciato

sorgente

ATCGA



# [ de Bruijn Graph (dBG) ]



## de Bruijn Graph di ordine k (*arc-centric*)

Il de Bruijn Graph di una collezione di reads  $\{r_1, r_2, \dots, r_n\}$  è il grafo  $G = (V, A)$  dove:

- ✓  $A$  rappresenta l'insieme degli elementi dello spettro di ordine  $k$ ; cioè, l'arco dal nodo  $v_1$  al nodo  $v_2$  è tale per cui:
  1. il suffisso di  $v_1$  lungo  $k-2$  è uguale a prefisso di  $v_2$  lungo  $k-2$  (overlap)
  2. il  $k$ -mer ottenuto assemblando  $v_1$  e  $v_2$  è sottostringa in almeno un read della collezione
- ✓ L'arco è etichettato dall'estensione (lunga un carattere) di  $v_2$  che eccede l'overlap con  $v_1$

# [ de Bruijn Graph (dBG) ]

## de Bruijn Graph di ordine k (*arc-centric*)

Il de Bruijn Graph di una collezione di reads  $\{r_1, r_2, \dots, r_n\}$  è il grafo  $G = (V, A)$  dove:

- ✓  $A$  rappresenta l'insieme degli elementi dello spettro di ordine  $k$ ; cioè, l'arco dal nodo  $v_1$  al nodo  $v_2$  è tale per cui:
  1. il suffisso di  $v_1$  lungo  $k-2$  è uguale a prefisso di  $v_2$  lungo  $k-2$  (overlap)
  2. il  $k$ -mer ottenuto assemblando  $v_1$  e  $v_2$  è sottostringa in almeno un read della collezione
- ✓ l'arco è etichettato dall'estensione (lunga un carattere) di  $v_2$  che eccede l'overlap con  $v_1$   
**in  $V$  ho  $(k-1)$ -mer**

Il dBG *arc-centric* di ordine  $k$  è uguale al dBG *node-centric* di ordine  $k-1$

# [ de Bruijn Graph (dBG) ]

de Bruijn Graph di ordine k (*arc-centric*)

ATCGATAGATGG                            k = 5  
    AGATGGAAAG

Spettro di ordine 5:

{ATCGA, TCGAT, CGATA, GATAG, ATAGA, TAGAT, AGATG,  
GATGG,  
AGATG, GATGG, ATGGA, TGGAA, GGAAG}

# [ de Bruijn Graph (dBG) ]

de Bruijn Graph di ordine k (*arc-centric*)

ATCGATAGATGG                            k = 5  
    AGATGGAAAG

devo ottenere un k-mer assemblando...  
quindi nei vertici avrò (k-1)-mer...

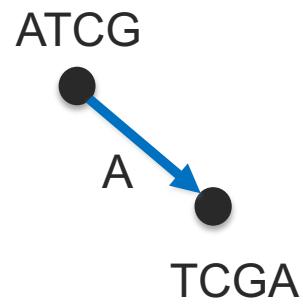
# [de Bruijn Graph (dBG)]

de Bruijn Graph di ordine k (*arc-centric*)

ATCGATAGATGG

k = 5

AGATGGAAAG



ATCGA → 5-mer

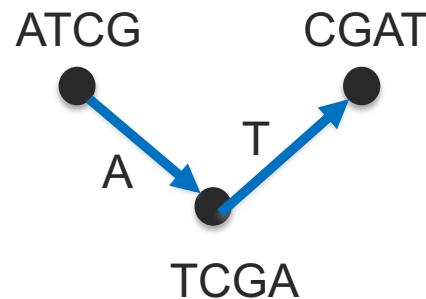
# [de Bruijn Graph (dBG)]

de Bruijn Graph di ordine k (*arc-centric*)

ATCGATAGATGG

k = 5

AGATGGAAAG



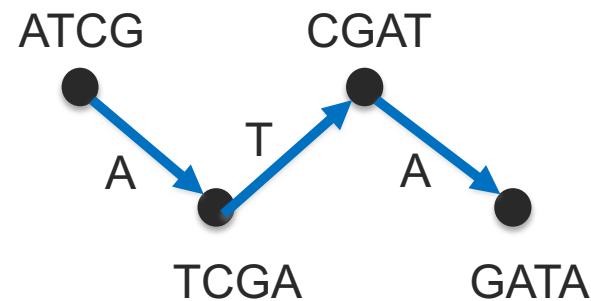
# [de Bruijn Graph (dBG)]

de Bruijn Graph di ordine k (*arc-centric*)

ATCGATAGATGG

k = 5

AGATGGAAAG



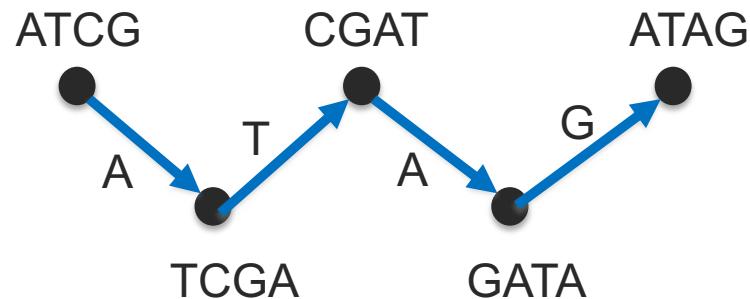
# [de Bruijn Graph (dBG)]

de Bruijn Graph di ordine k (*arc-centric*)

ATCGGATAGATGG

k = 5

AGATGGAAAG



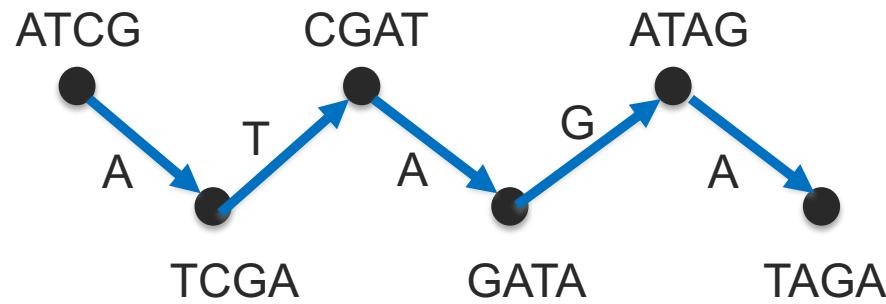
# [de Bruijn Graph (dBG)]

de Bruijn Graph di ordine k (*arc-centric*)

ATCGGATAGATGG

k = 5

AGATGGAAAG



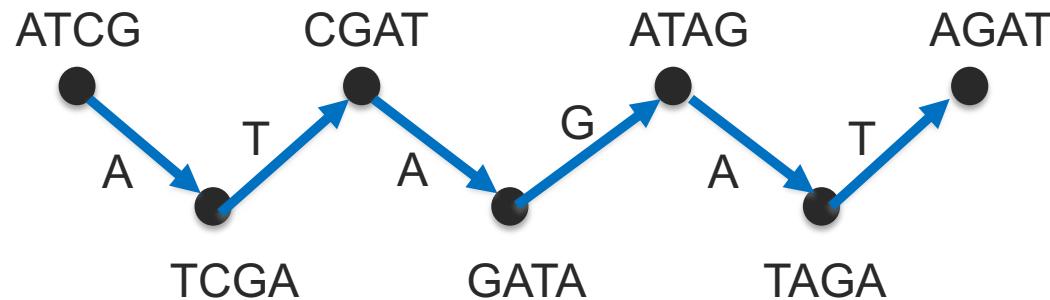
# [de Bruijn Graph (dBG)]

de Bruijn Graph di ordine k (*arc-centric*)

ATCGGATAGATGG

k = 5

AGATGGAAAG



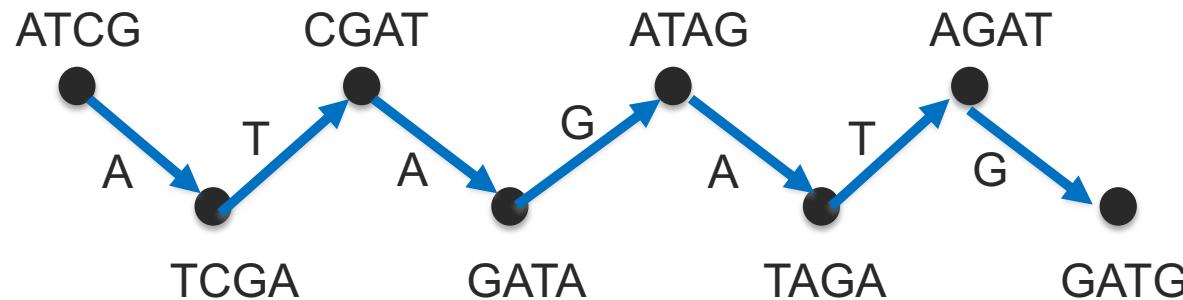
# [de Bruijn Graph (dBG)]

de Bruijn Graph di ordine k (*arc-centric*)

ATCGGATAGATGG

k = 5

AGATGGAAAG



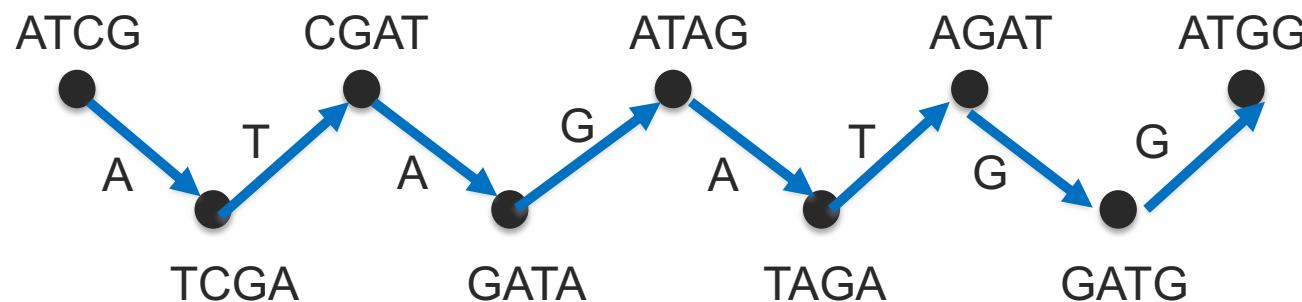
# [de Bruijn Graph (dBG)]

de Bruijn Graph di ordine k (*arc-centric*)

ATCGGATAGATGG

k = 5

AGATGGGAAG



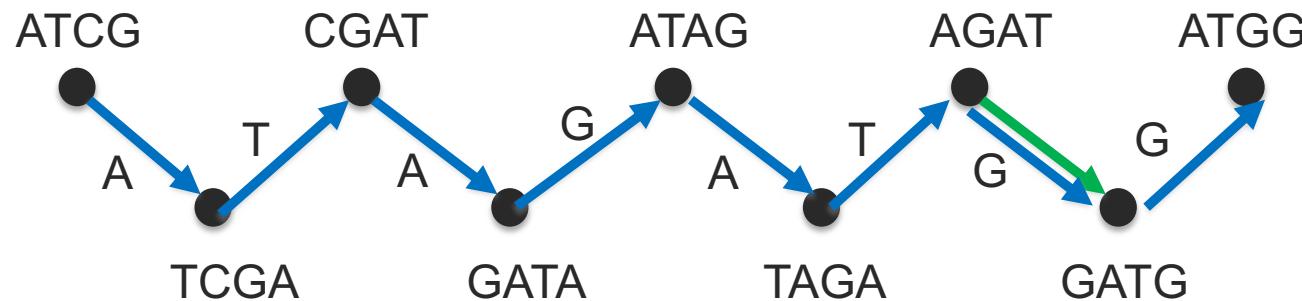
# [de Bruijn Graph (dBG)]

de Bruijn Graph di ordine k (*arc-centric*)

ATCGATAGATGG

k = 5

AGATGGAAAG



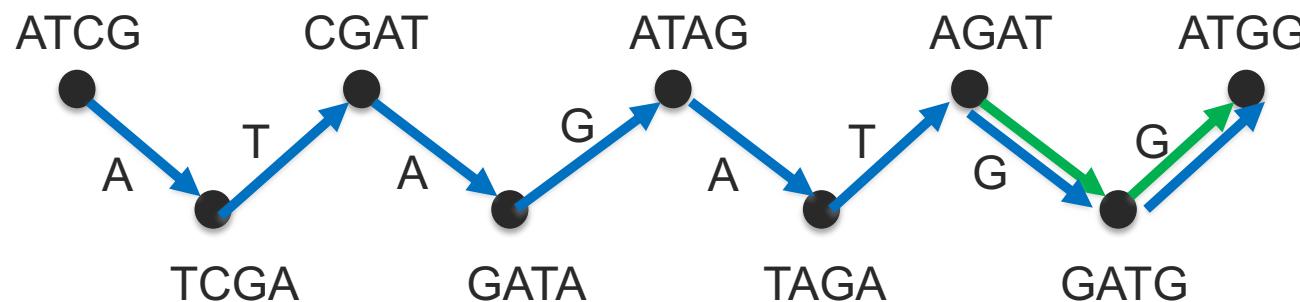
# [de Bruijn Graph (dBG)]

de Bruijn Graph di ordine k (*arc-centric*)

ATCGATAGATGG

k = 5

AGATGGAAAG



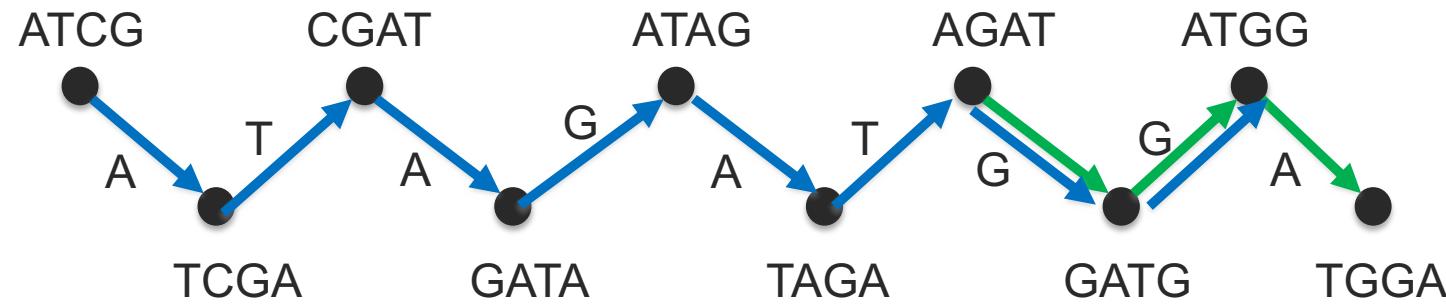
# [de Bruijn Graph (dBG)]

de Bruijn Graph di ordine k (*arc-centric*)

ATCGATAGATGG

k = 5

AGATGGAAAG



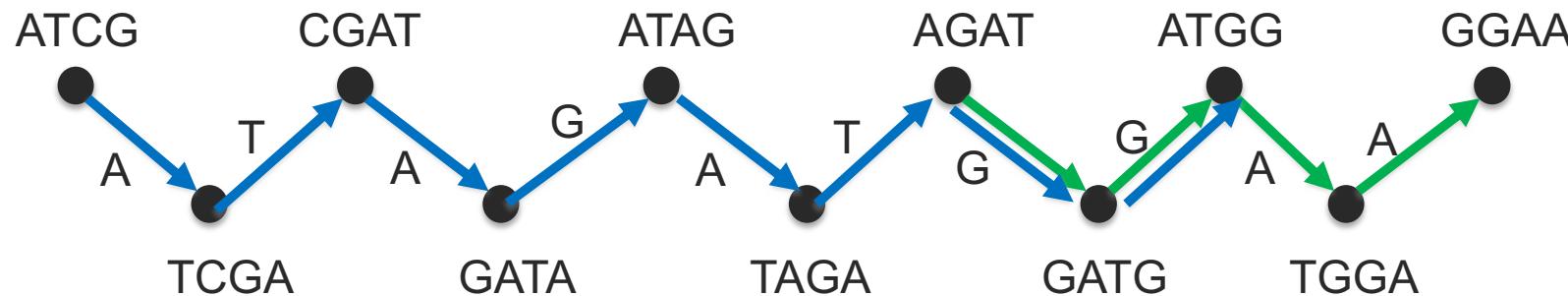
# [de Bruijn Graph (dBG)]

de Bruijn Graph di ordine k (*arc-centric*)

ATCGATAGATGG

k = 5

AGATGGAAAG



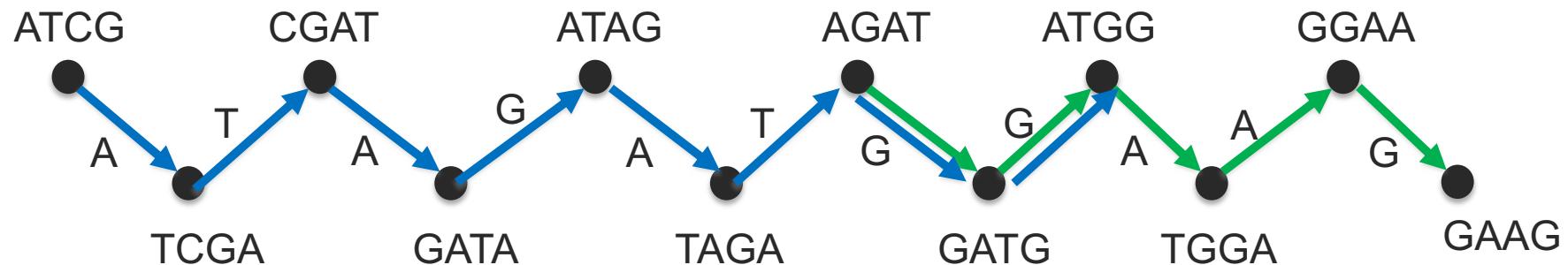
# [de Bruijn Graph (dBG)]

de Bruijn Graph di ordine k (*arc-centric*)

ATCGATAGATGG

k = 5

AGATGGAAAG



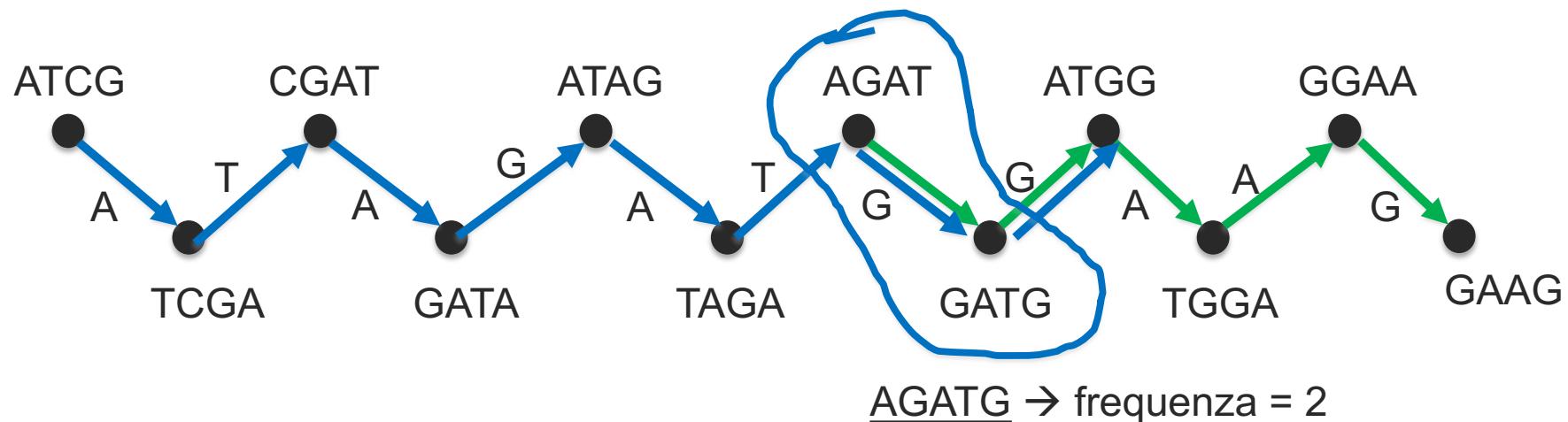
# [de Bruijn Graph (dBG)]

de Bruijn Graph di ordine k (*arc-centric*)

ATCGGATAGATGG

k = 5

AGATGGAAG

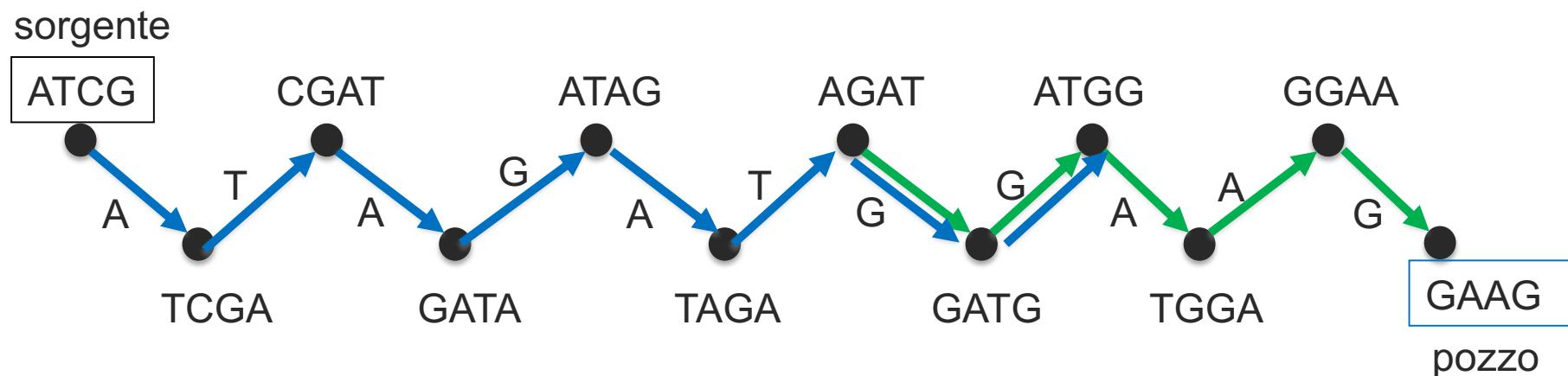


# [de Bruijn Graph (dBG)]

de Bruijn Graph di ordine k (*arc-centric*)

ATCGATAGATGG  
AGATGGAAAG

k = 5



# [de Bruijn Graph (dBG)]

de Bruijn Graph di ordine k (*arc-centric*)

ATCGATAGATGG

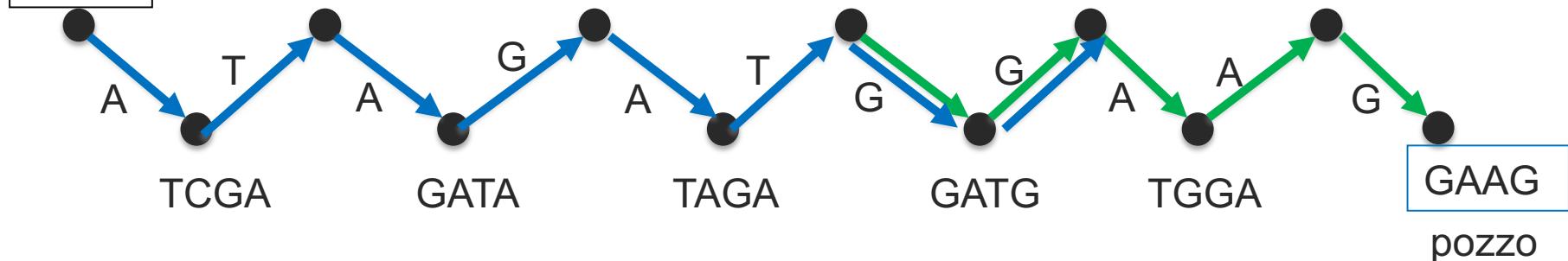
k = 5

AGATGGAAAG

ATCG

sorgente

ATCG



# [de Bruijn Graph (dBG)]

de Bruijn Graph di ordine k (*arc-centric*)

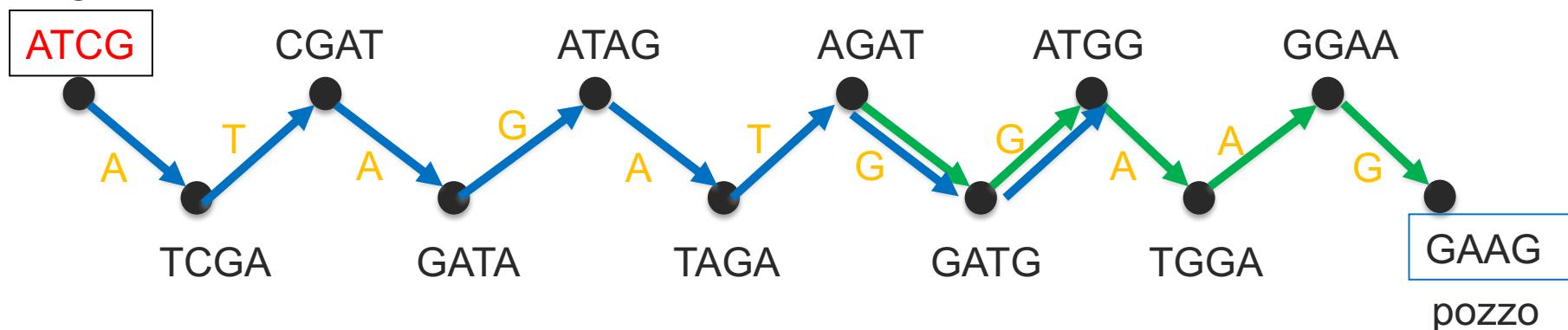
ATCGATAGATGG

k = 5

AGATGGAAAG

ATCGATAGATGGAAAG

sorgente



# [ de Bruijn Graph ]

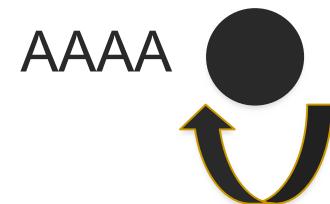
Prestare attenzione a:

✓ *loop*

AAAAAA

k=4

→ AAAA, AAAA, AAAA, AAAA, AAAA, AAAA



4 è basso..  
scelgo k>9

# [ de Bruijn Graph ]

Prestare attenzione a:

- ✓ *loop*
- ✓ ripetizioni

(per assemblaggio del genoma  $k \sim 32\text{nt}$ )

se  $k$  è minore della lunghezza delle stringhe ripetute, non si riesce a creare un assemblamento.  
Ho multiedge, non per copertura ma per ripetizione.

Se so che esistono stringhe ripetute di una certa lunghezza, scelgo  $k >$  di quella lunghezza  
(Bloom Filter)

---

# Sparse and Skew Hashing of K-Mers

Giulio Ermanno Pibiri<sup>1</sup>

<sup>1</sup>ISTI-CNR, Pisa, Italy

## Abstract

**Motivation:** A dictionary of  $k$ -mers is a data structure that stores a set of  $n$  distinct  $k$ -mers and supports membership queries. This data structure is at the heart of many important tasks in computational biology. High-throughput sequencing of DNA can produce very large  $k$ -mer sets, in the size of billions of strings – in such cases, the memory consumption and query efficiency of the data structure is a concrete challenge.

**Results:** To tackle this problem, we describe a *compressed* and *associative* dictionary for  $k$ -mers, that is: a data structure where strings are represented in compact form and each of them is associated to a unique integer identifier in the range  $[0, n]$ . We show that some statistical properties of  $k$ -mer *minimizers* can be exploited by minimal perfect hashing to substantially improve the space/time trade-off of the dictionary compared to the best-known solutions.

**Availability:** The C++ implementation of the dictionary is available at <https://github.com/jermp/sshash>.

**Contact:** [giulio.ermanno.pibiri@isti.cnr.it](mailto:giulio.ermanno.pibiri@isti.cnr.it)

---

# Ripetizioni (esempio 1)

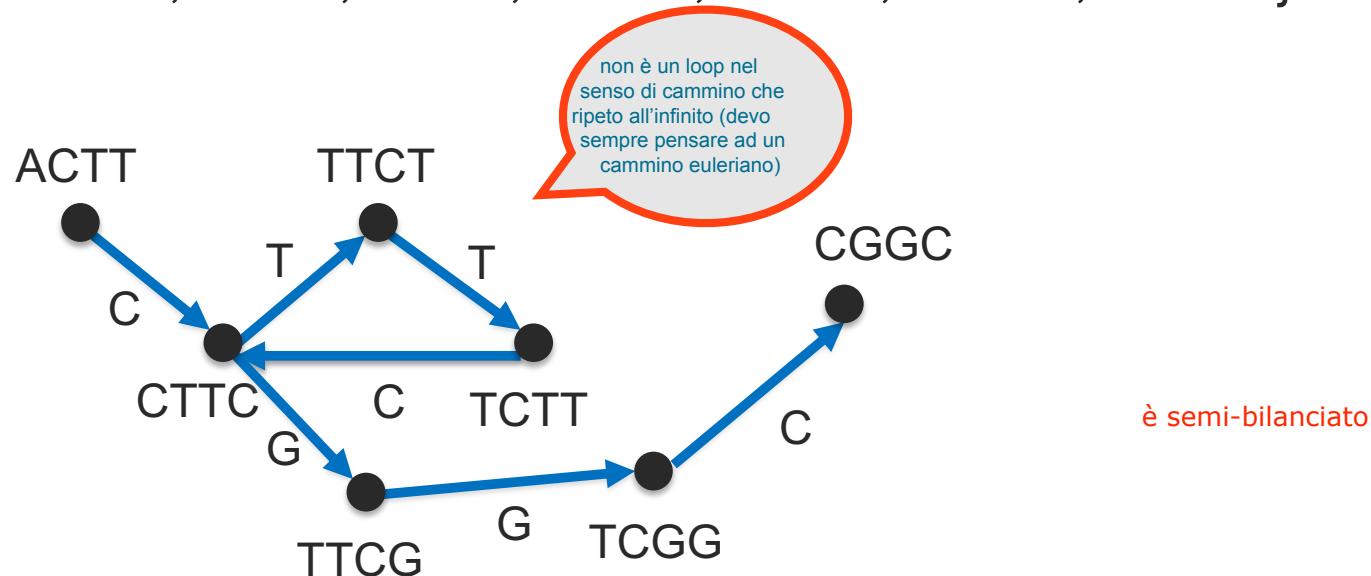
ACTTCTTGGC

$k = 5$

(per costruire il dBG arco-centrico di ordine 5)

Spettro di ordine 4:

{ACTT, CTTC, TTCT, TCTT, CTTC, TTCT, TCGG, CGGC}



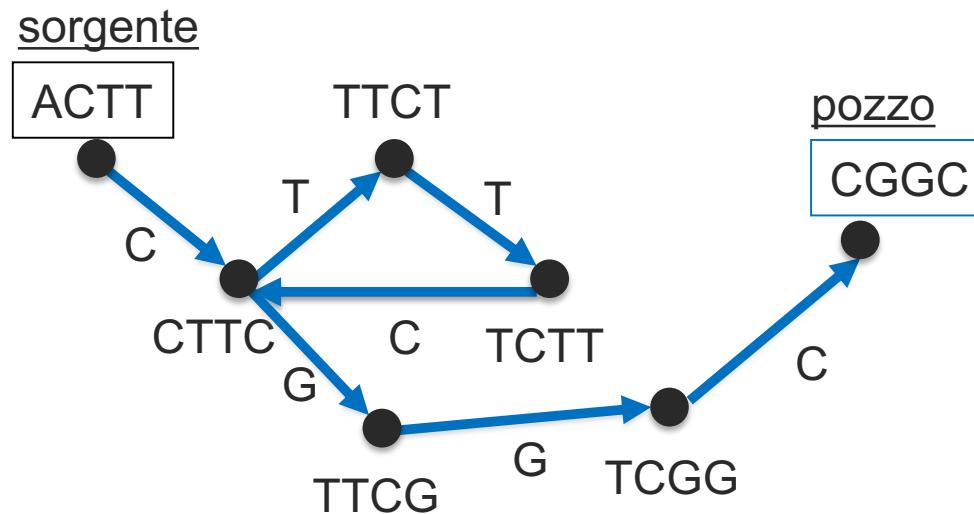
# Ripetizioni (esempio 1)

ACTTCTTGGC

$k = 5$

Spettro di ordine 4:

{ACTT, CTTC, TTCT, TCTT, CTTC, TTGCG, TCGG, CGGC}



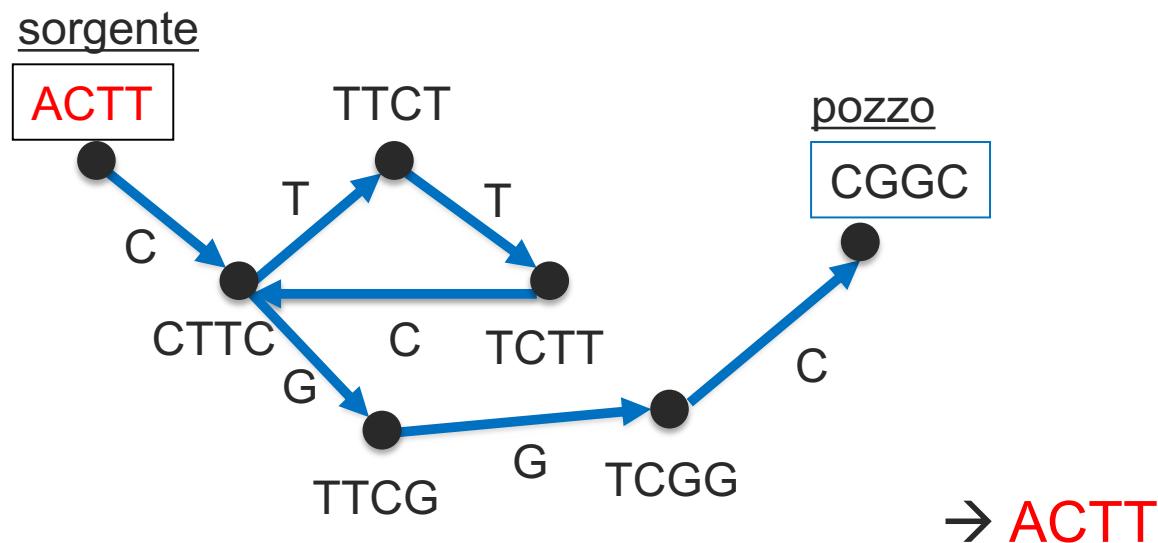
# Ripetizioni (esempio 1)

ACTTCTTGGC

$k = 5$

Spettro di ordine 4:

{ACTT, CTTC, TTCT, TCTT, CTTC, TTG, TCGG, CGGC}



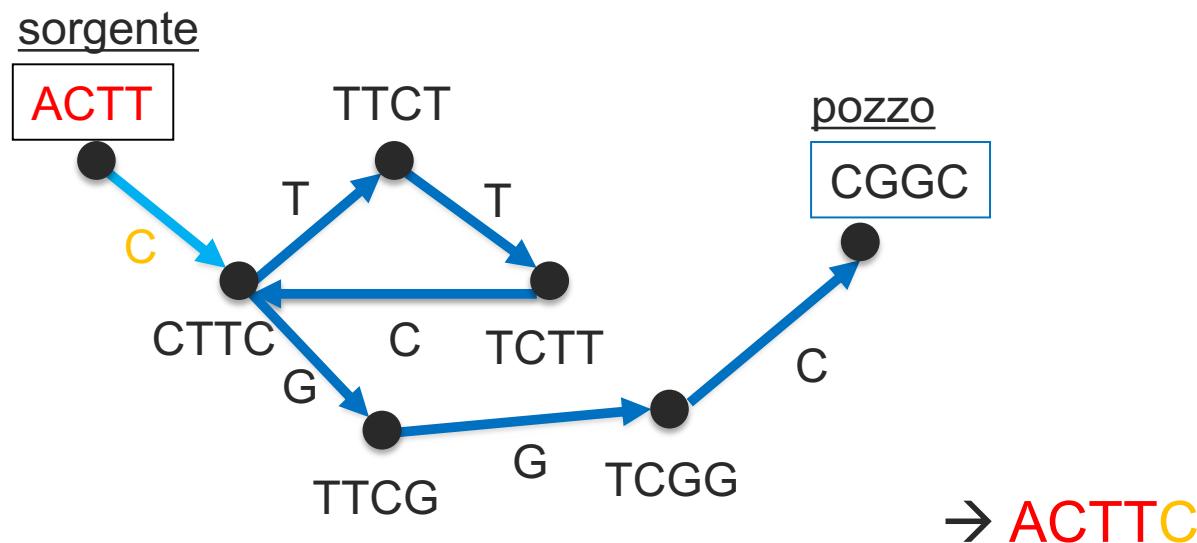
# Ripetizioni (esempio 1)

ACTTCTTGGC

$k = 5$

Spettro di ordine 4:

{ACTT, CTTC, TTCT, TCTT, CTTC, TTG, TCGG, CGGC}



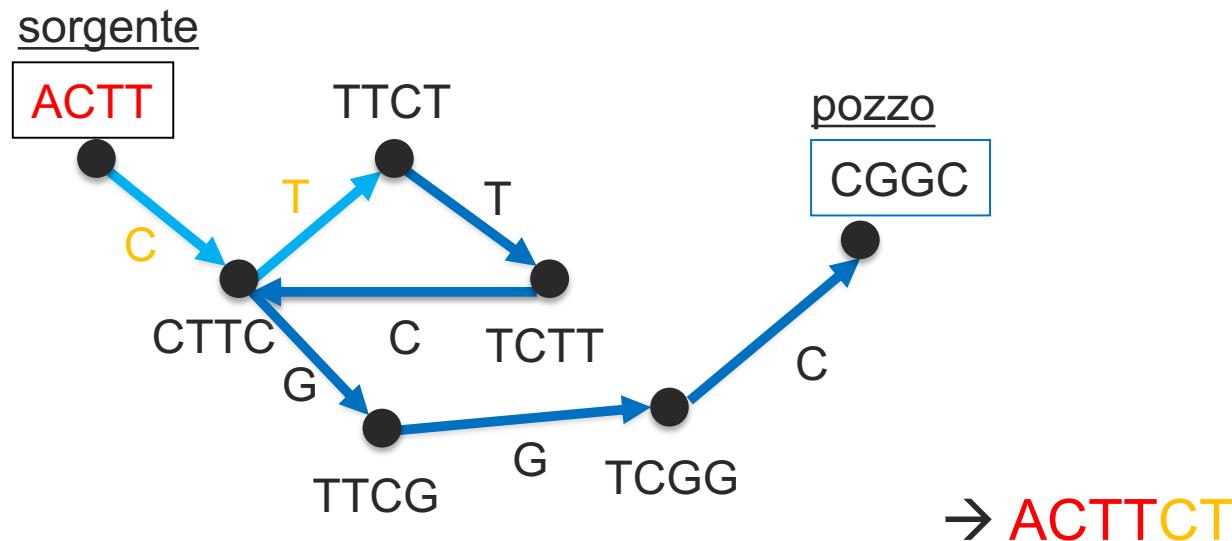
# Ripetizioni (esempio 1)

ACTTCTT CGGC

$k = 5$

Spettro di ordine 4:

{ACTT, CTTC, TTCT, TCTT, CTTC, TTGCG, TCGG, CGGC}



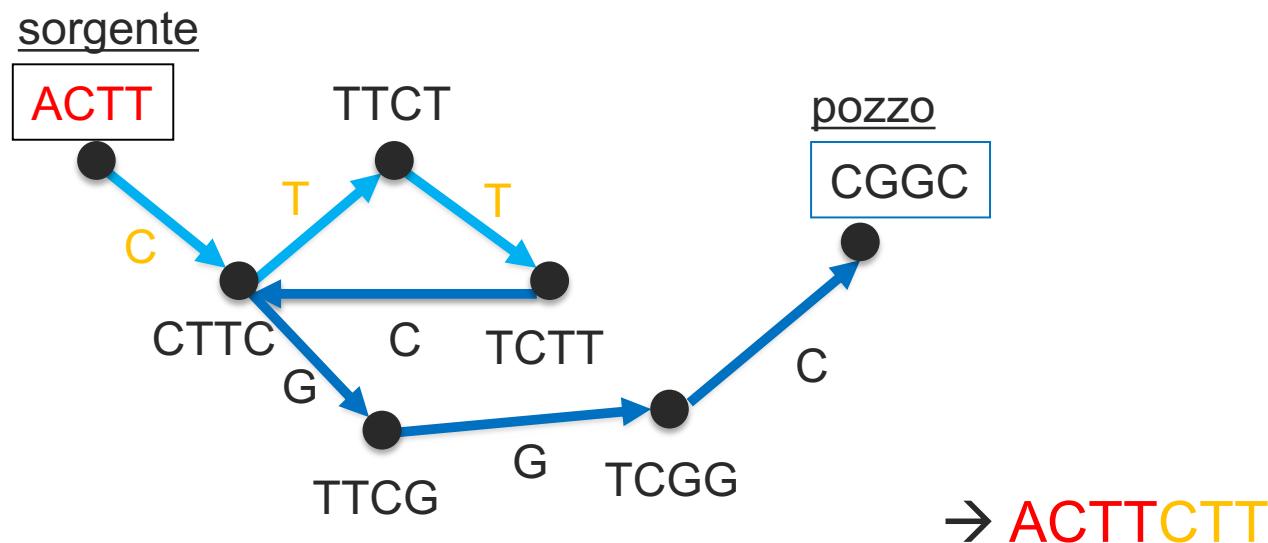
# Ripetizioni (esempio 1)

ACTTCTT CGGC

$k = 5$

Spettro di ordine 4:

{ACTT, CTTC, TTCT, TCTT, CTTC, TTCTG, TCGG, CGGC}



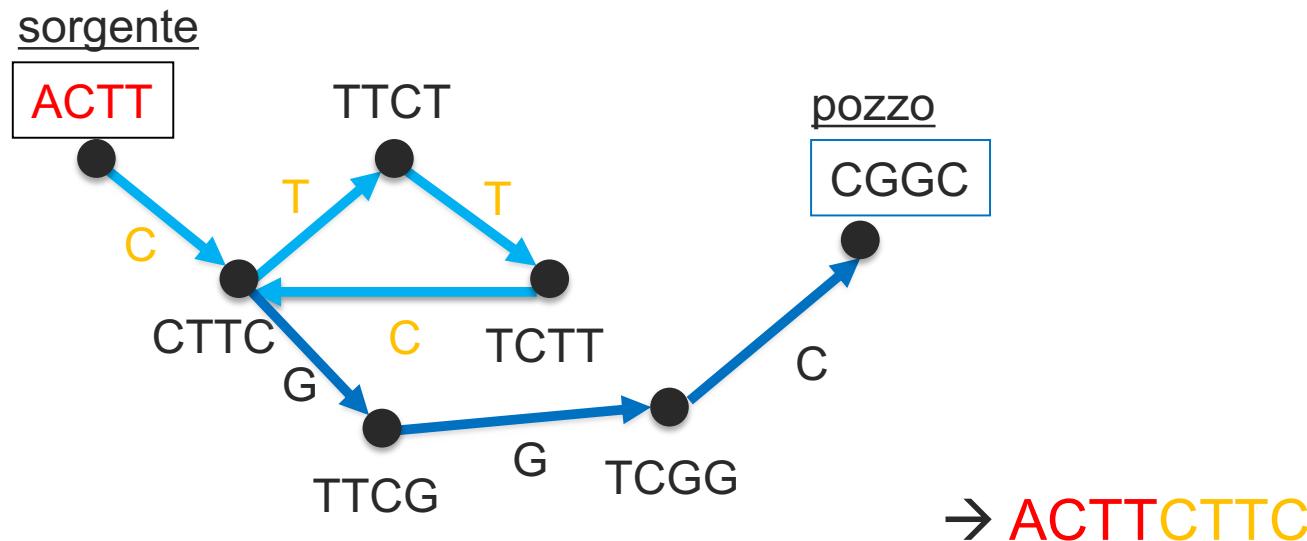
# Ripetizioni (esempio 1)

ACTTCTT CGGC

$k = 5$

Spettro di ordine 4:

{ACTT, CTTC, TTCT, TCTT, CTTC, TTCTG, TCGG, CGGC}



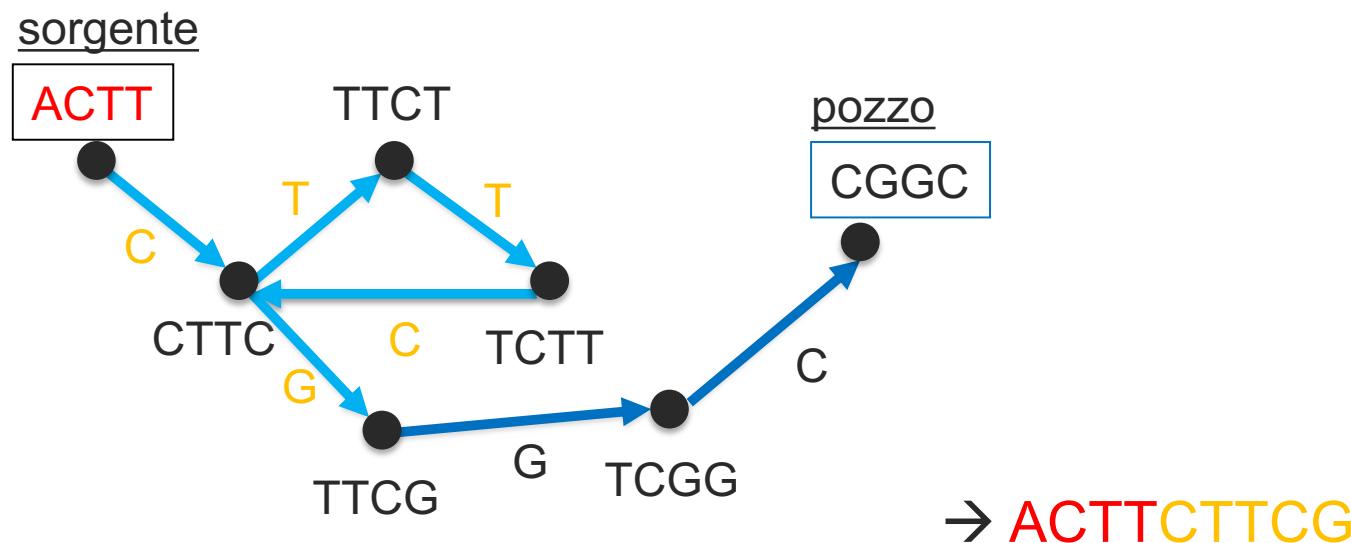
# Ripetizioni (esempio 1)

ACTTCTT CGGC

$k = 5$

Spettro di ordine 4:

{ACTT, CTTC, TTCT, TCTT, CTTC, TTCTG, TCGG, CGGC}



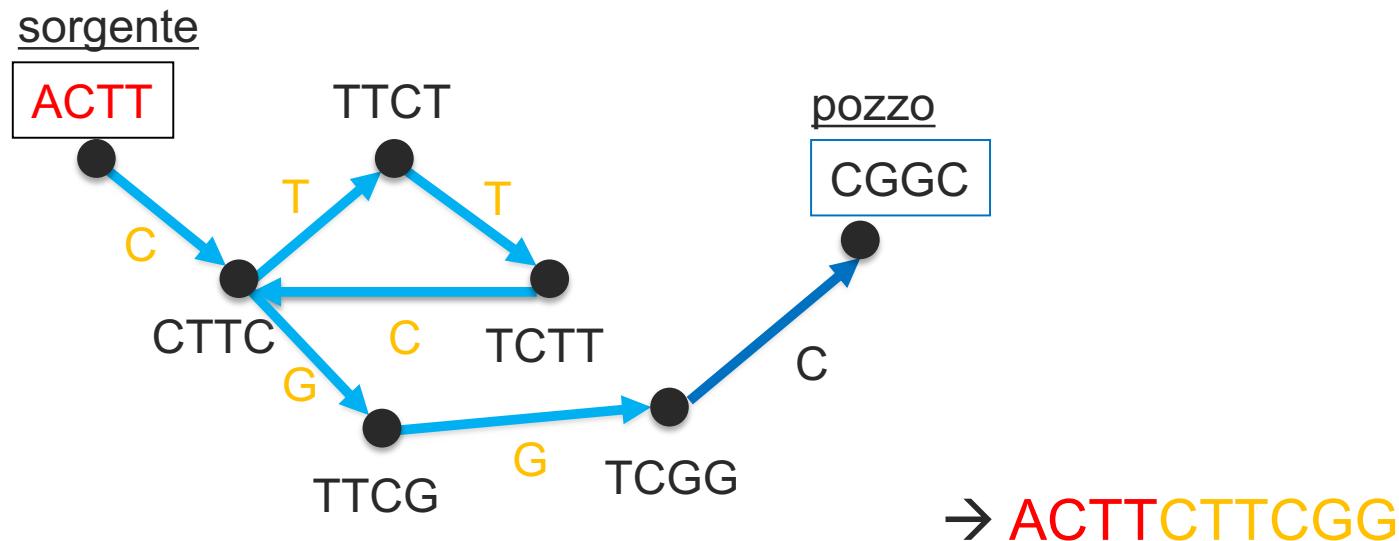
# Ripetizioni (esempio 1)

ACTTCTTGGC

$k = 5$

Spettro di ordine 4:

{ACTT, CTTC, TTCT, TCTT, CTTC, TTGCG, TCGG, CGGC}



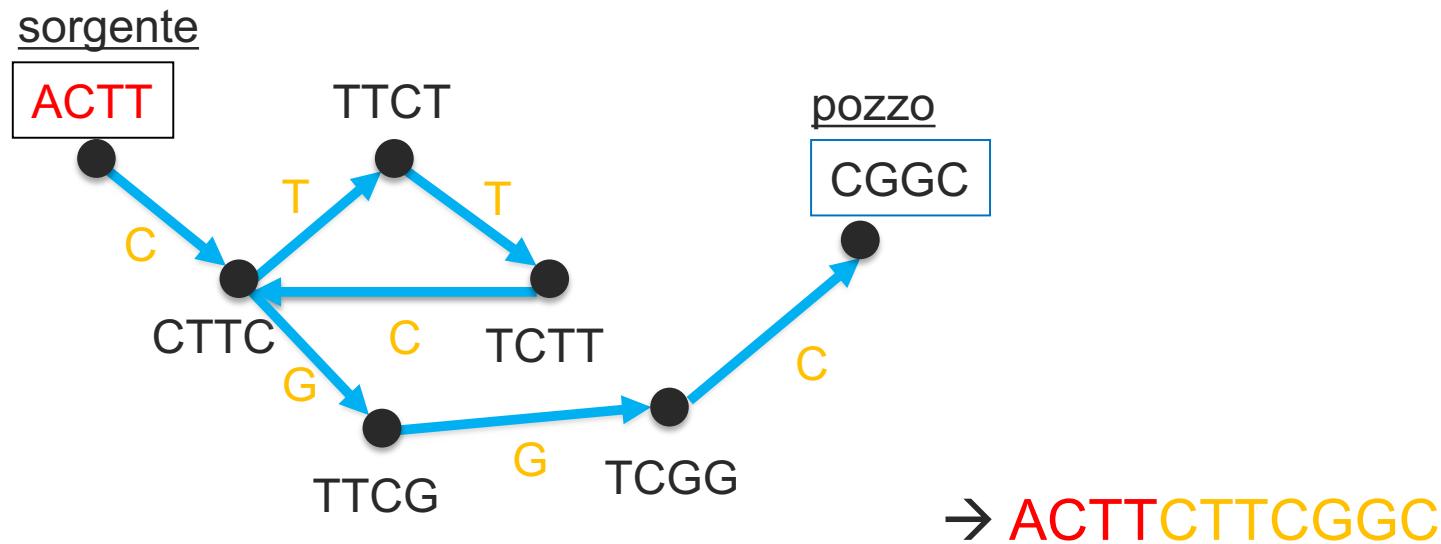
# Ripetizioni (esempio1)

ACTTCTTGGC

k = 5

Spettro di ordine 4:

{ACTT, CTTC, TTCT, TCTT, CTTC, TTGCG, TCGG, CGGC}



OK! Questo k ha consentito l'assemblaggio  
(CTTC è il fattore che si ripete)

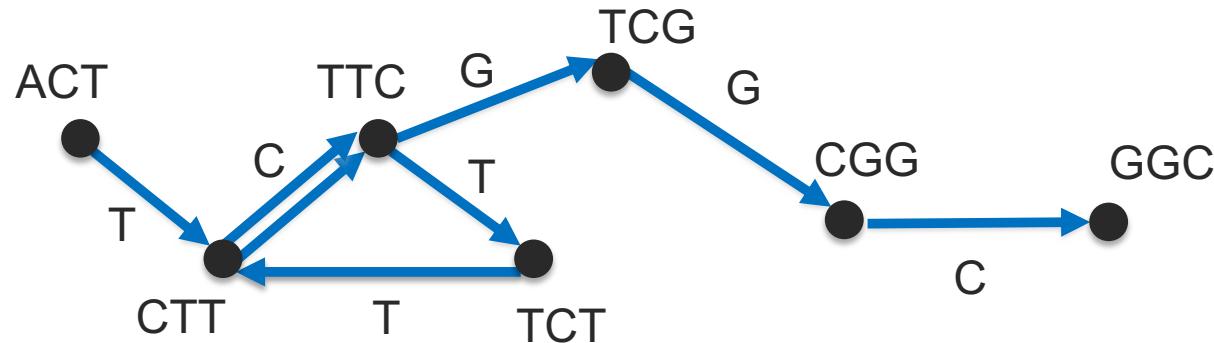
# Ripetizioni (esempio1)

ACTTCTTCGGC

$k = 4$

Spettro di ordine 3:

{ACT, CTT, TTC, TCT, CTT, TTC, TCG, CGG, GGC}



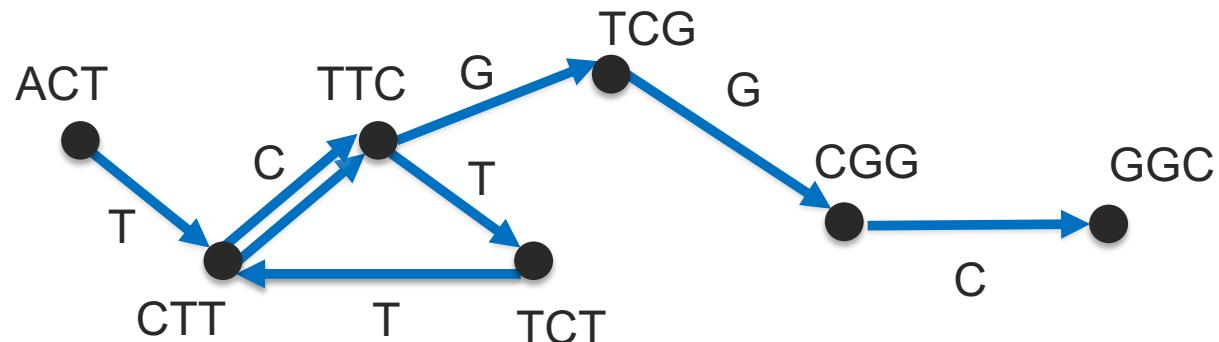
# Ripetizioni (esempio 1)

ACTTCTTCGGC

$k = 4$

Spettro di ordine 3:

{ACT, CTT, TTC, TCT, CTT, TTC, TCG, CGG, GGC}



doppio arco non dovuto alla coverage  
ma alle ripetizioni.

il multiedge è  
sulla stessa  
sequenza  
(stesso  
colore)

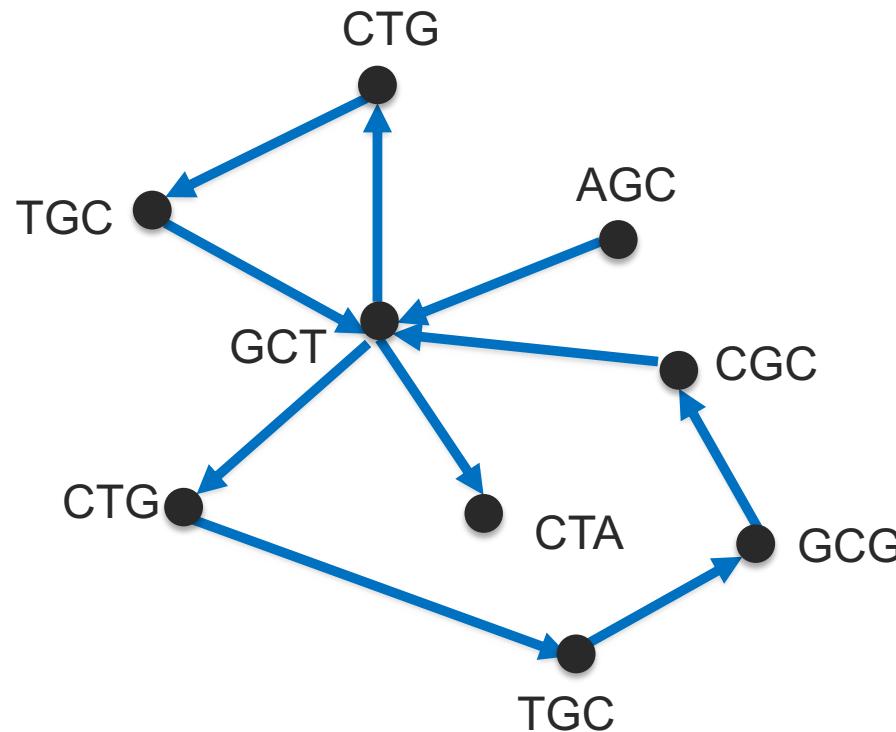
Non bilanciato!

non esiste cammino euleriano,  
cioè non posso assemblare!

# [Ripetizioni (esempio2)]

(non scritte le etichette  
degli archi, per semplicità)

$k = 4$

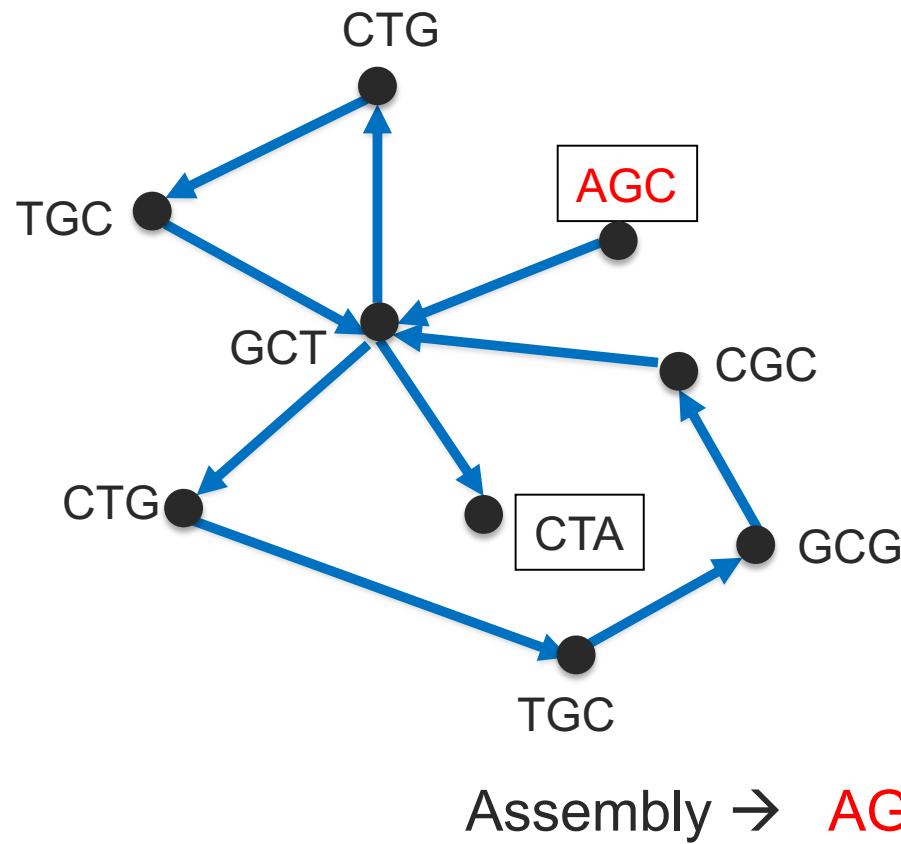


è semi-bilanciato

Esiste il cammino euleriano...  
ma non è unico!

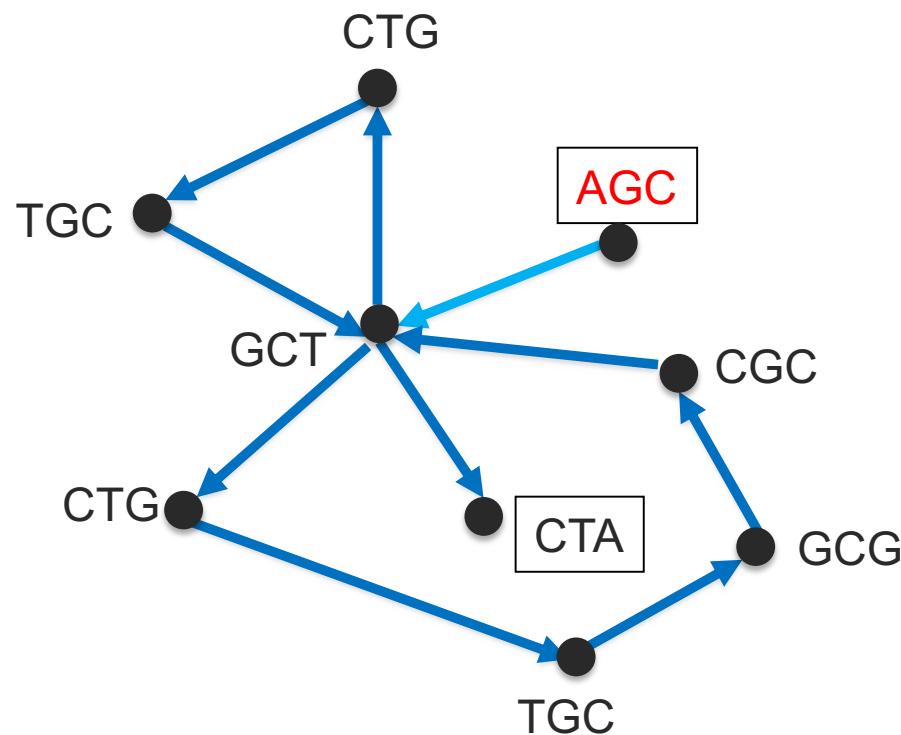
# Ripetizioni (esempio2)

$k = 4$



# Ripetizioni (esempio2)

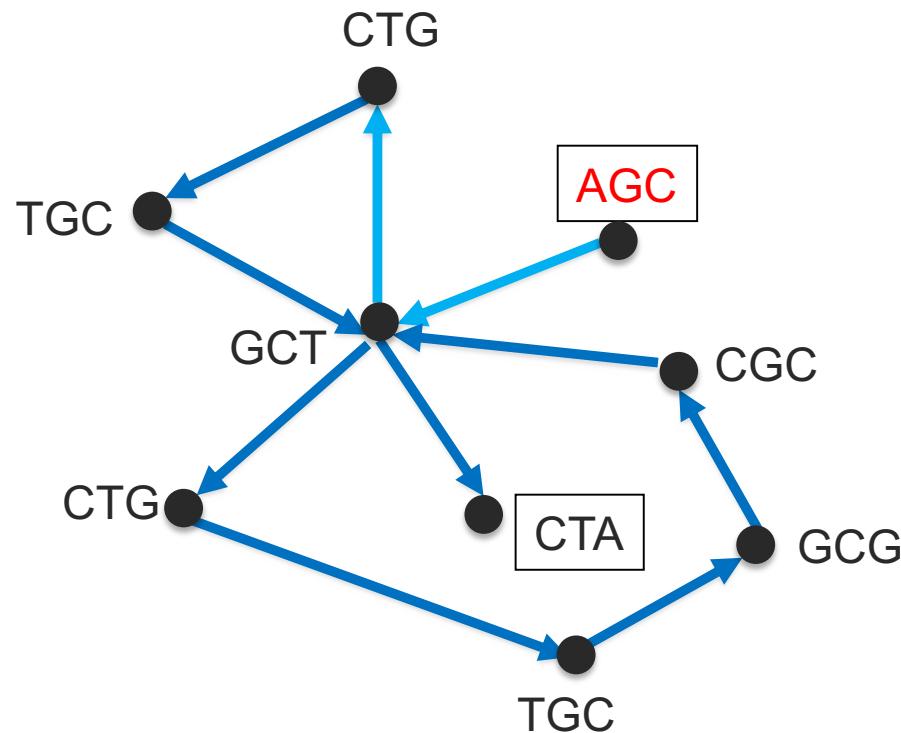
$k = 4$



Assembly → **AGCT**

# Ripetizioni (esempio2)

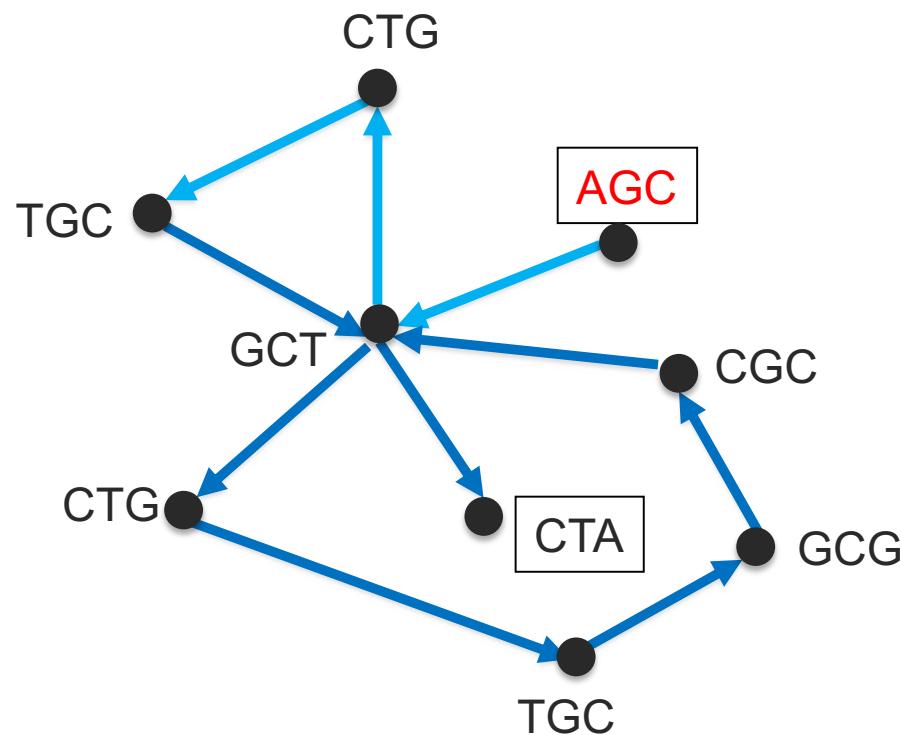
$k = 4$



Assembly → **AGCTG**

# Ripetizioni (esempio2)

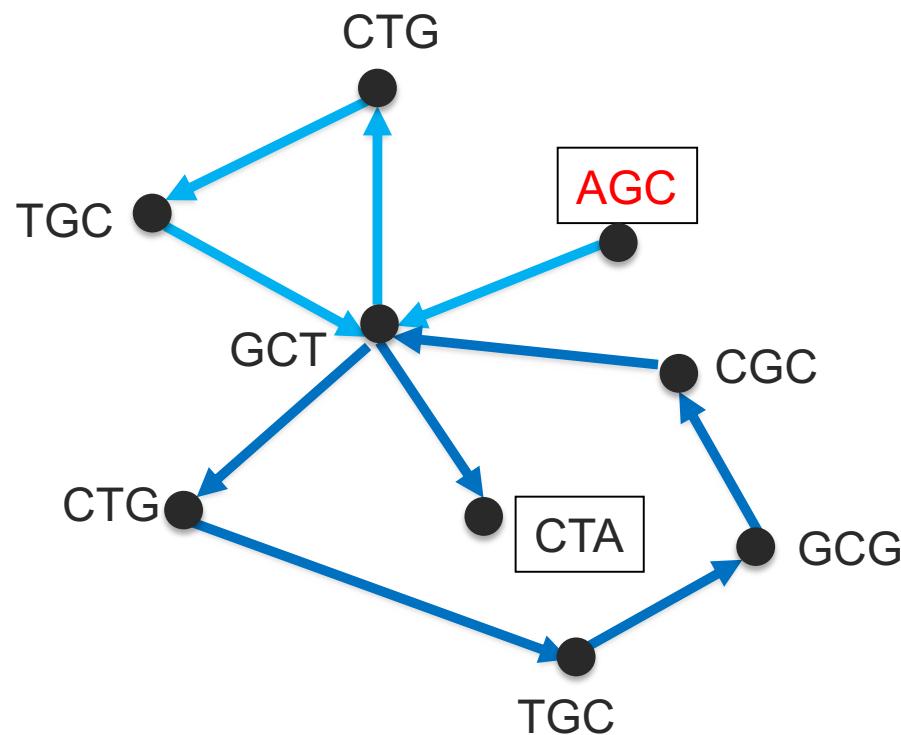
$k = 4$



Assembly → **AGCTGC**

# Ripetizioni (esempio2)

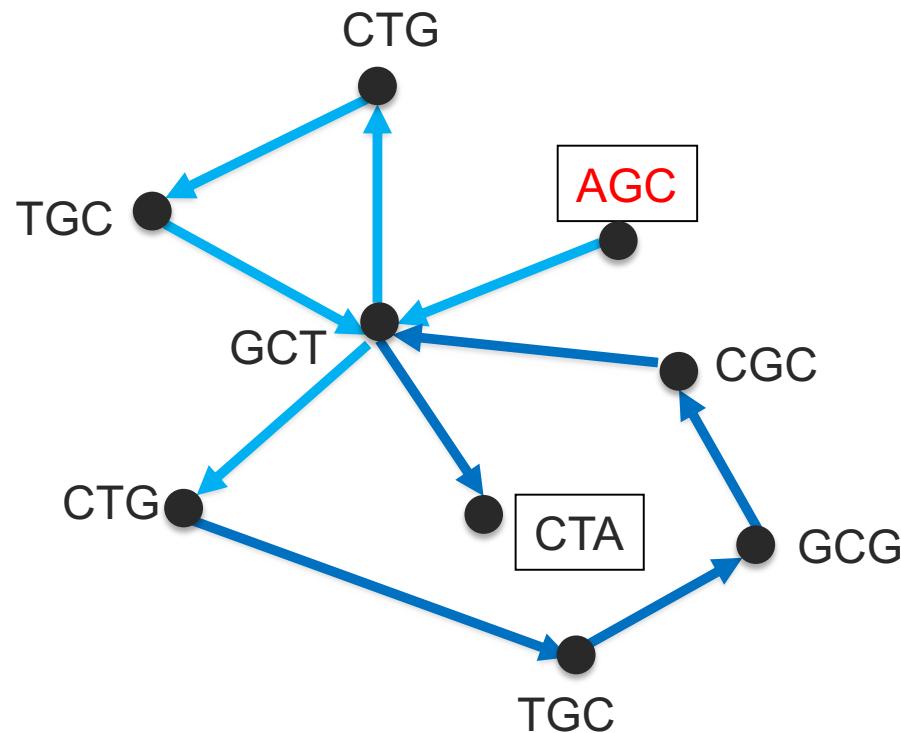
$k = 4$



Assembly → **AGC**TGCT

# Ripetizioni (esempio2)

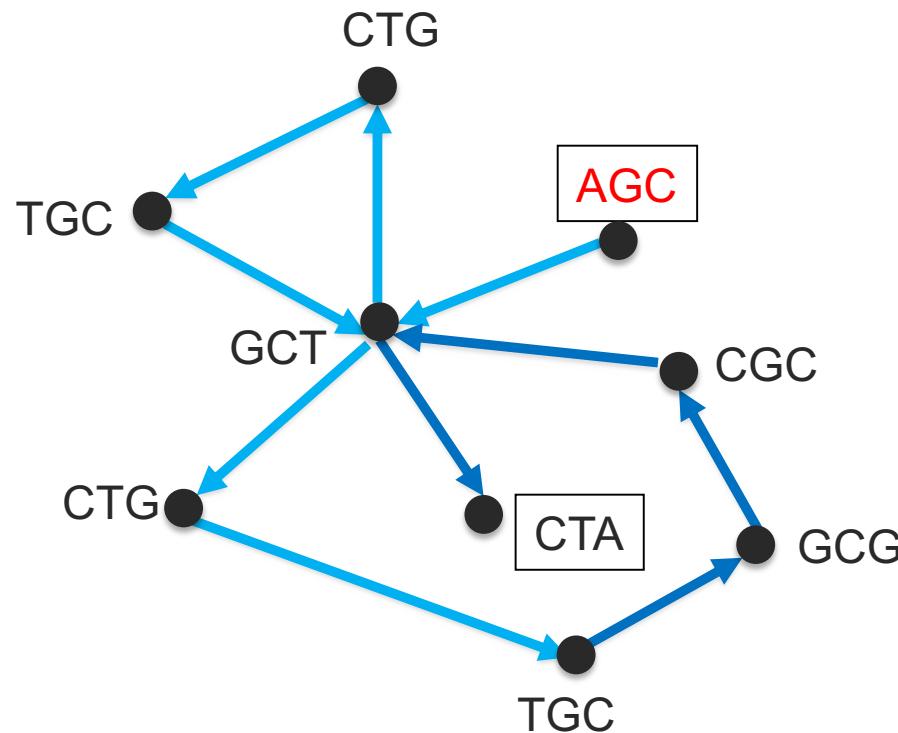
$k = 4$



Assembly → **AGC**TGCTG

# Ripetizioni (esempio2)

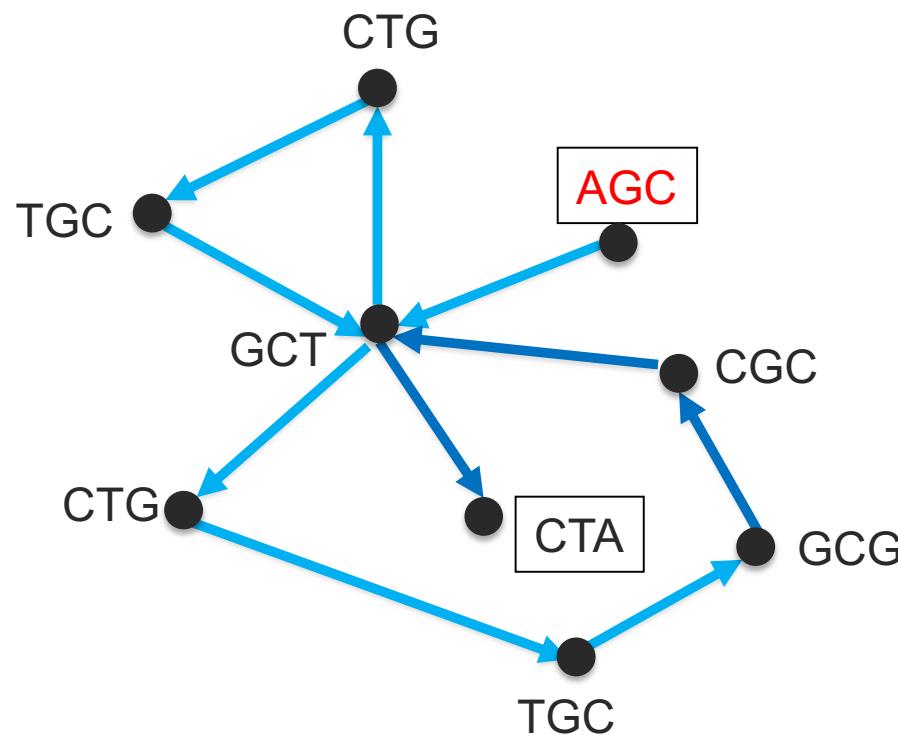
$k = 4$



Assembly → **AGC**TGCTGC

# Ripetizioni (esempio2)

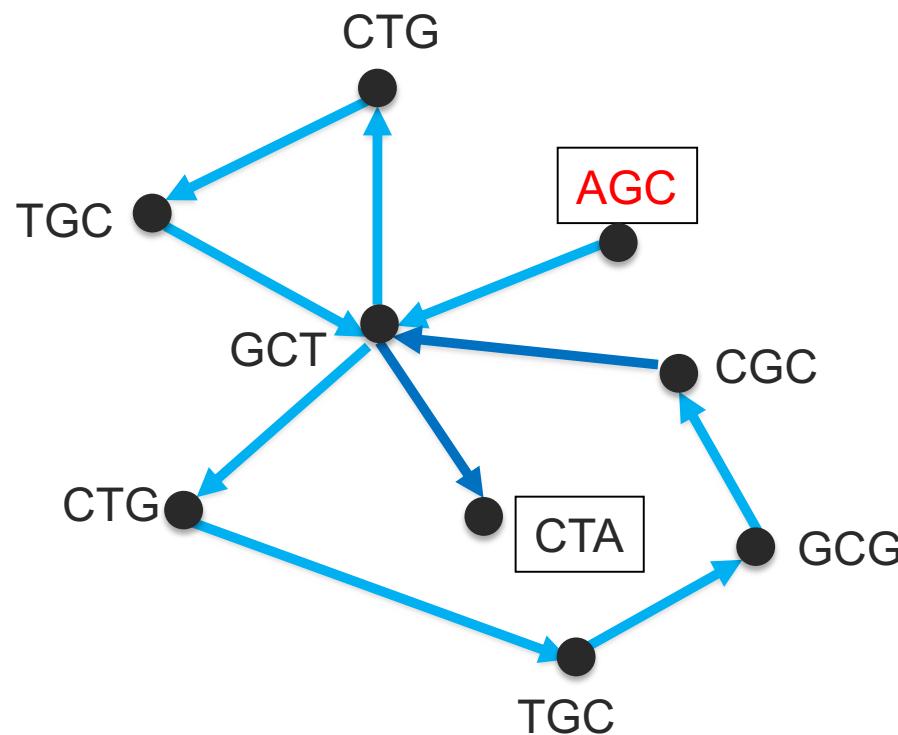
$k = 4$



Assembly → **AGC**TGCTGCG

# Ripetizioni (esempio2)

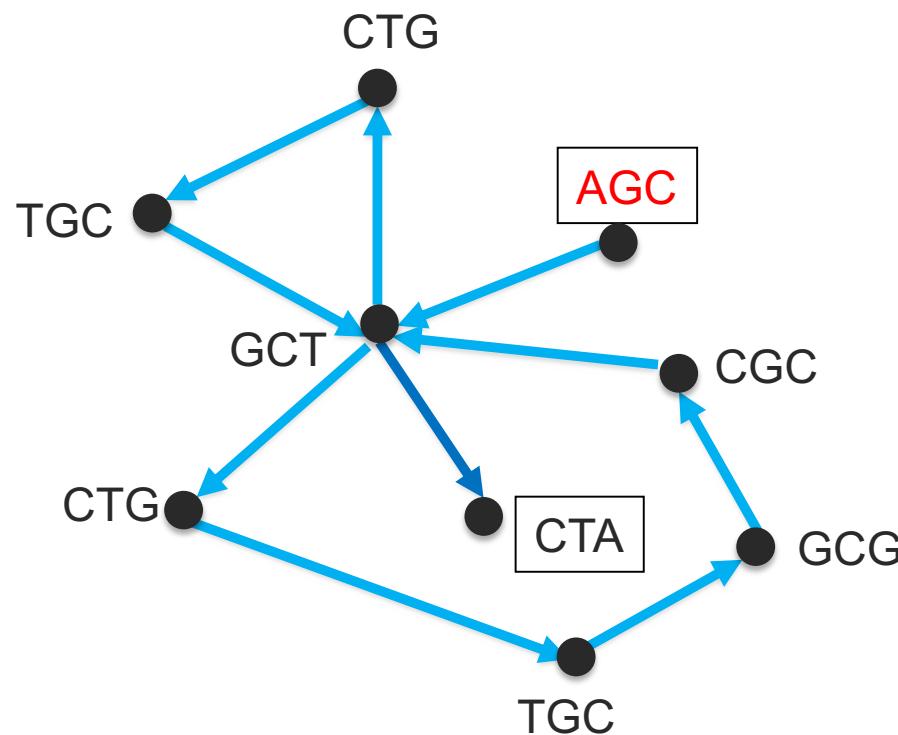
$k = 4$



Assembly → **AGC**TGCTGCGC

# Ripetizioni (esempio2)

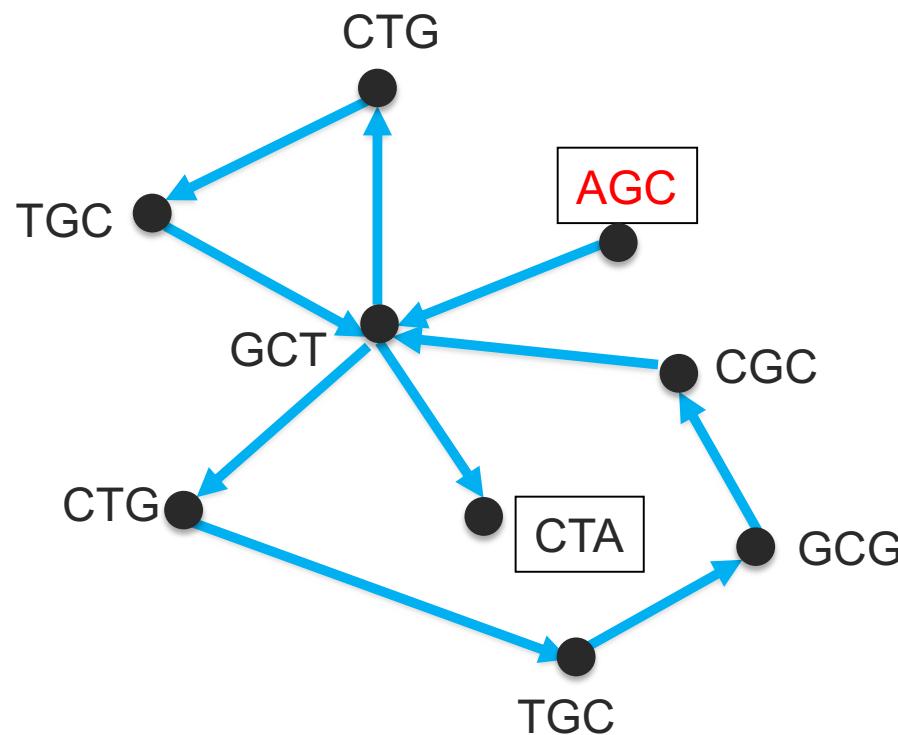
$k = 4$



Assembly → **AGC**TGCTGCGCT

# Ripetizioni (esempio2)

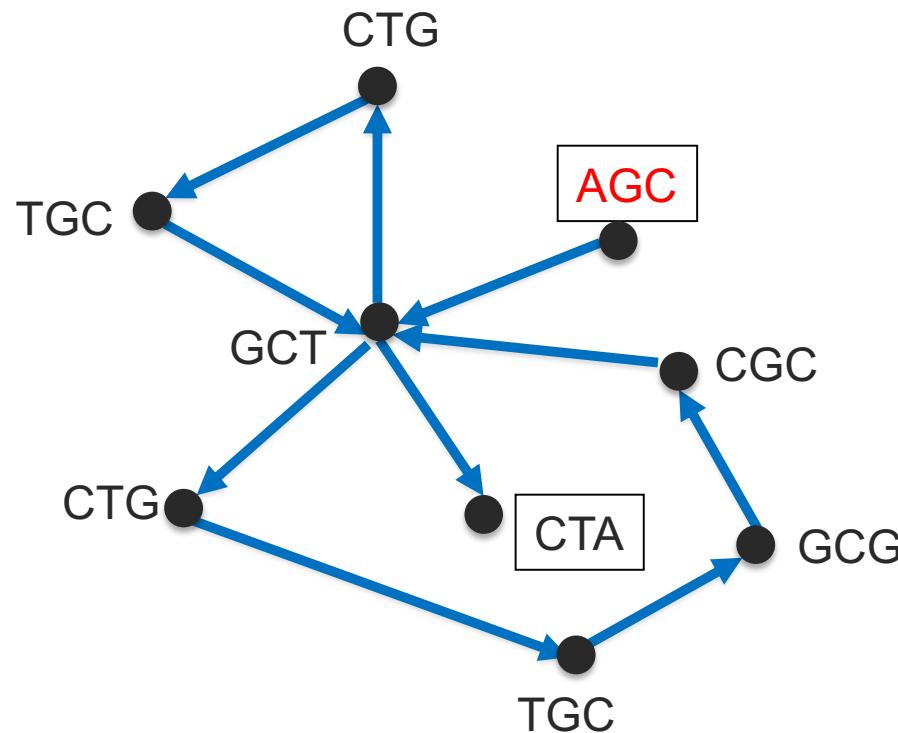
$k = 4$



Assembly → **AGC**TGCTGCGCTA

# Ripetizioni (esempio2)

$k = 4$

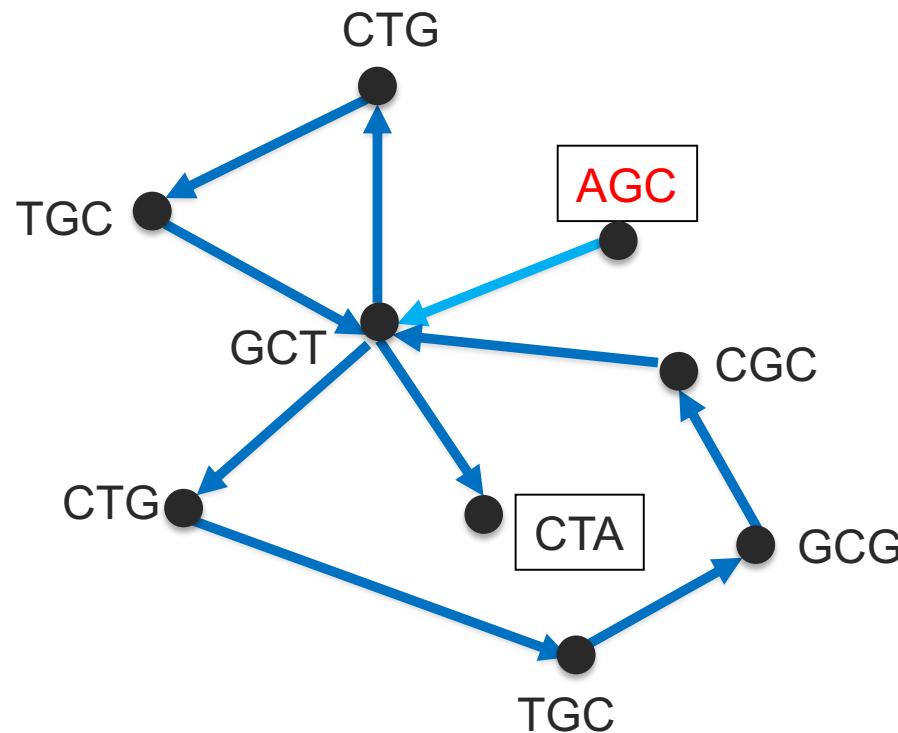


Assembly → AGCTGCTGCGCTA

Assembly2 → AGC

# Ripetizioni (esempio2)

$k = 4$

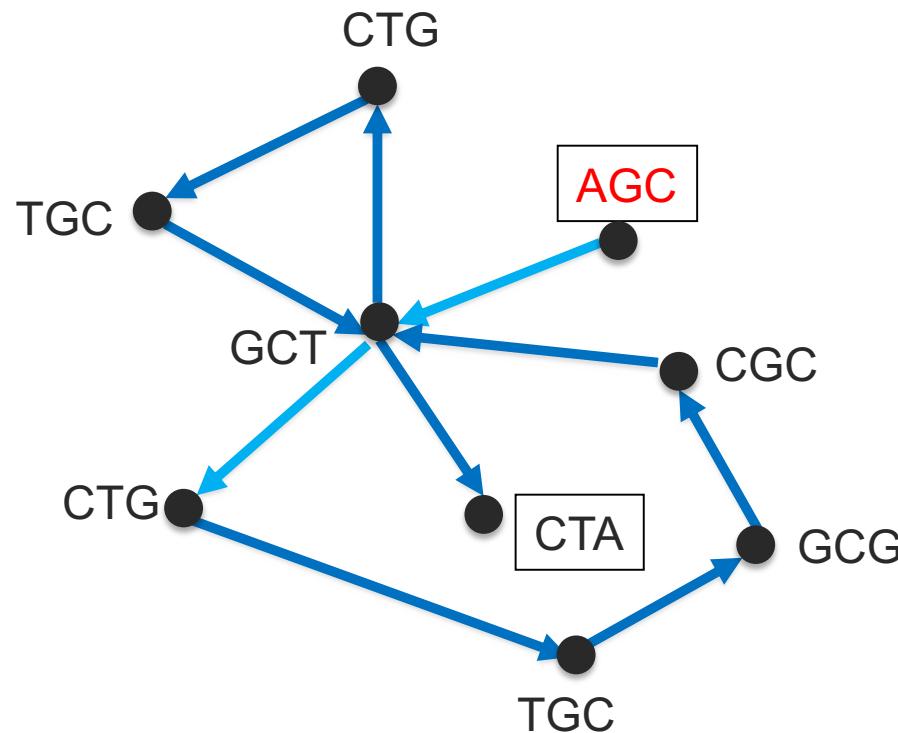


Assembly → AGCTTGCTGCGCTA

Assembly2 → AGCT

# Ripetizioni (esempio2)

$k = 4$

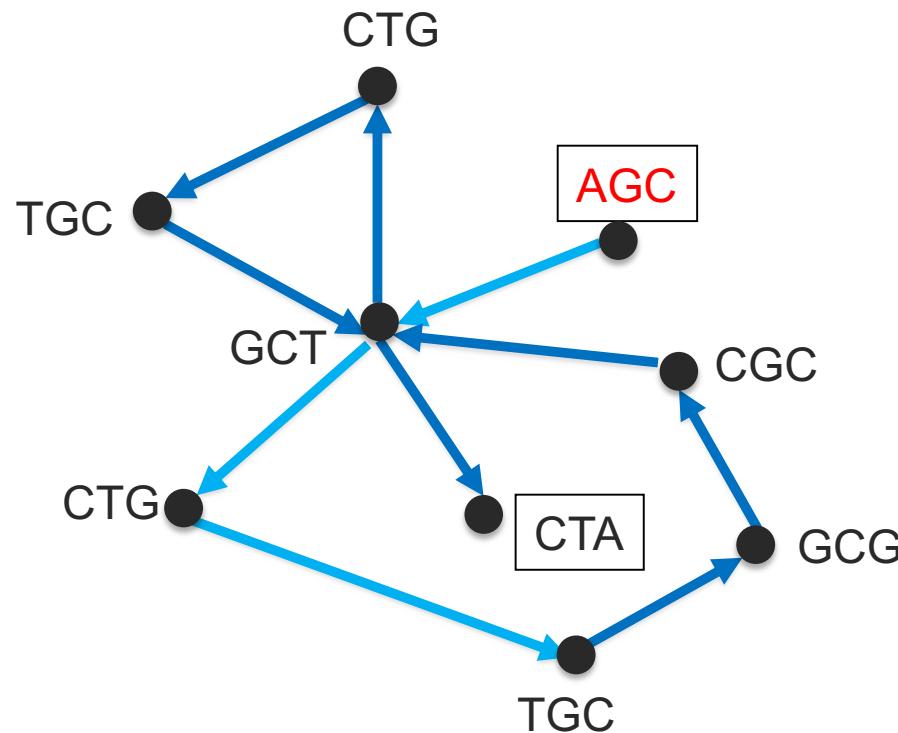


Assembly → AGCTGCTGCGCTA

Assembly2 → AGCTG

# Ripetizioni (esempio2)

$k = 4$

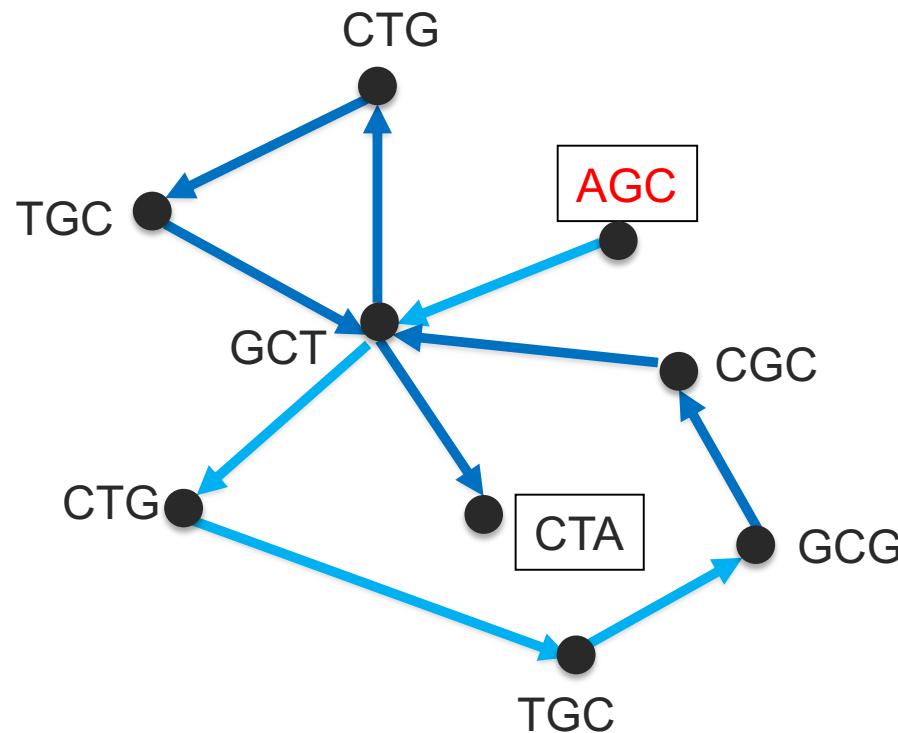


Assembly → AGCTGCTGCGCTA

Assembly2 → AGCTGC

# Ripetizioni (esempio2)

$k = 4$

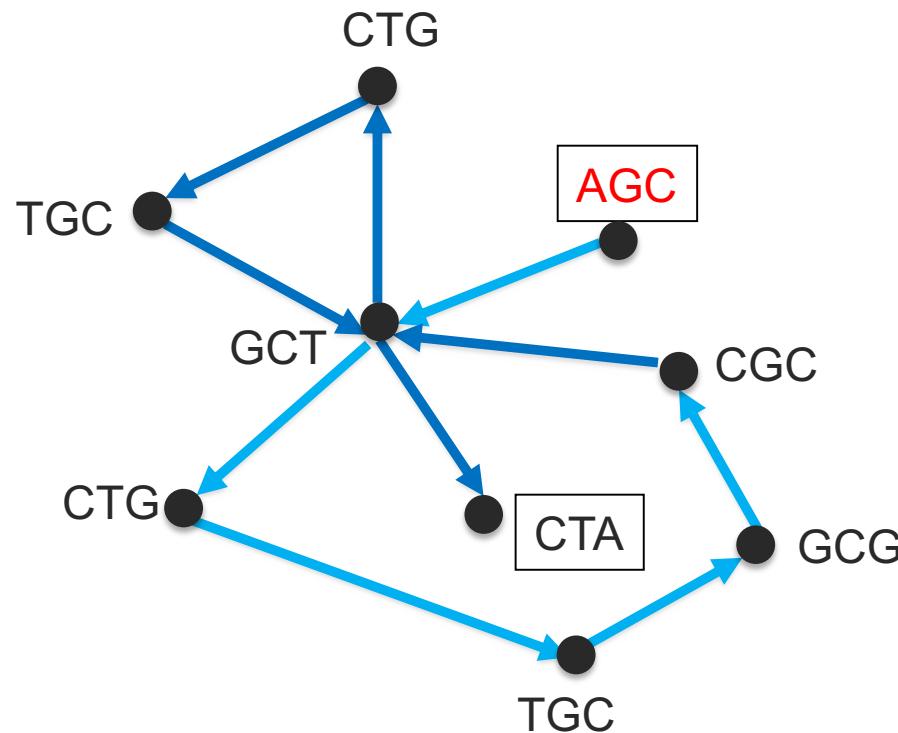


Assembly → AGCTGCTGCGCTA

Assembly2 → AGCTGCG

# Ripetizioni (esempio2)

$k = 4$

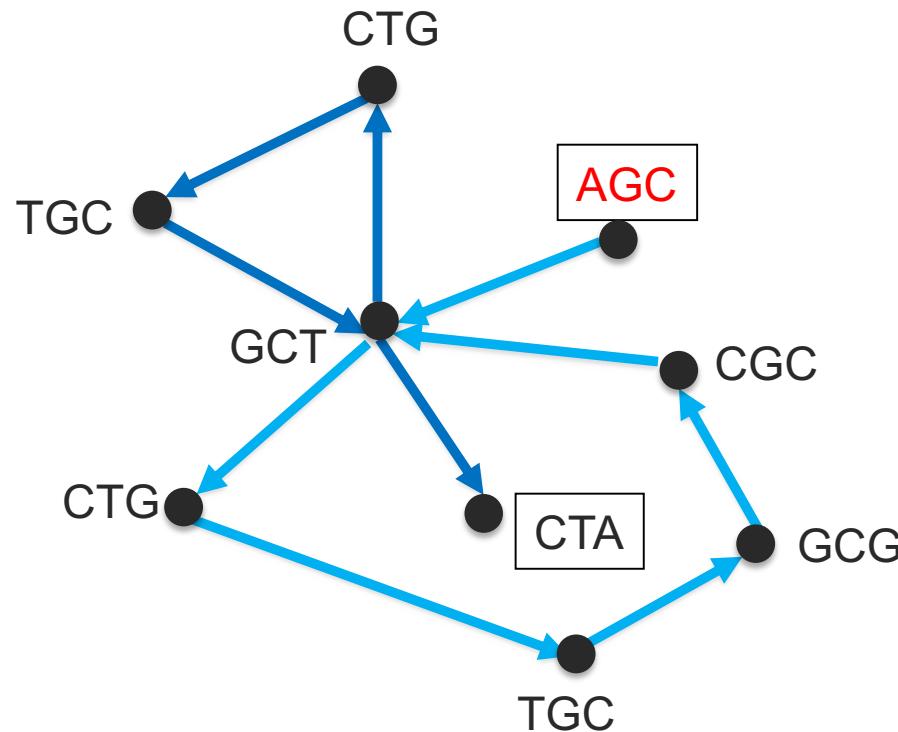


Assembly → AGCTGCTGCGCTA

Assembly2 → AGCTGCGC

# Ripetizioni (esempio2)

$k = 4$

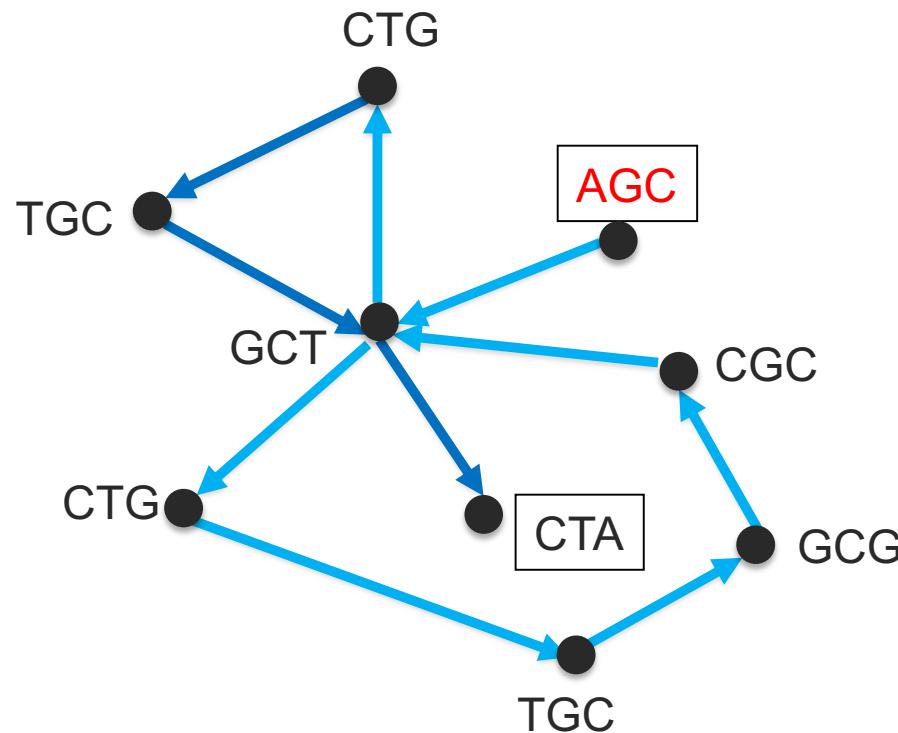


Assembly → AGCTGCTGCGCTA

Assembly2 → AGCTTGCGCT

## Ripetizioni (esempio2)

$k = 4$

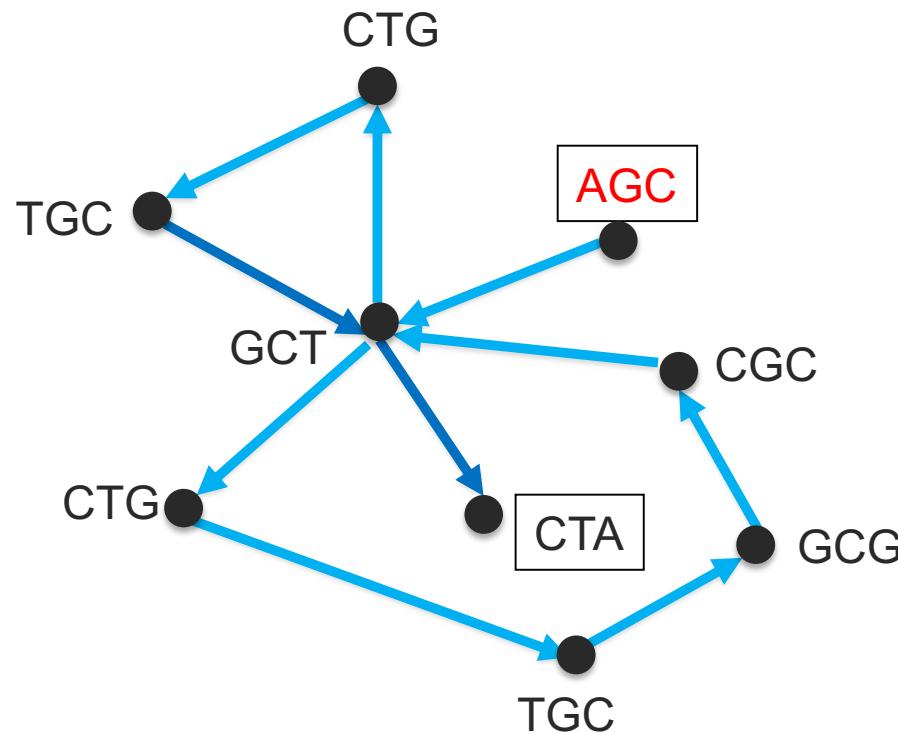


Assembly → **AGC**TGCTGCGCTA

Assembly2 → **AGC**TGCGCTG

# Ripetizioni (esempio2)

$k = 4$

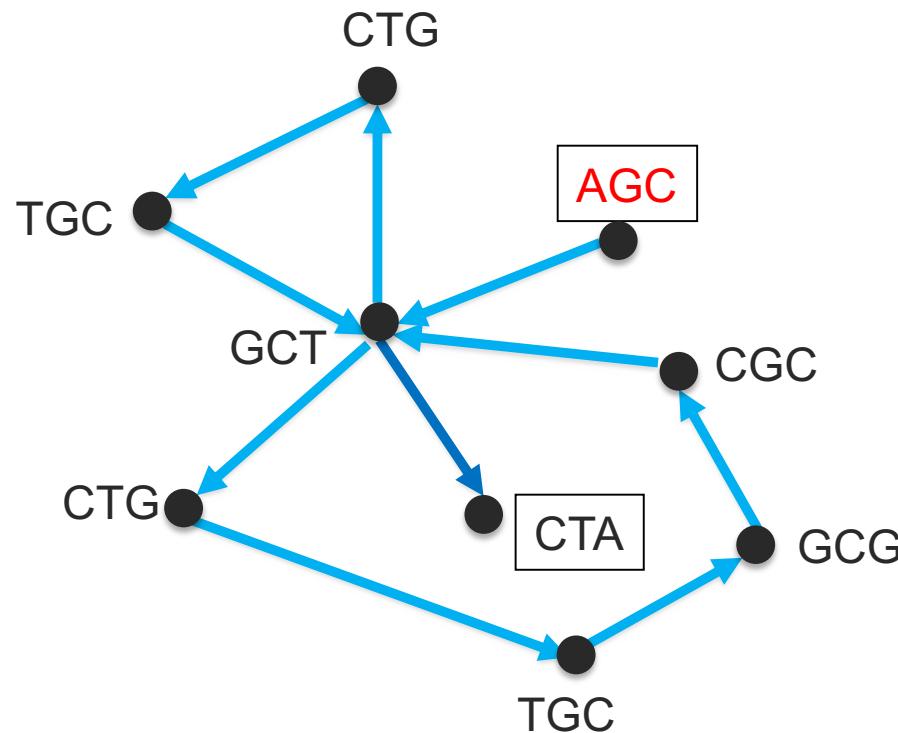


Assembly → **AGC**TGCTGCGCTA

Assembly2 → **AGC**TGCGCTGC

# Ripetizioni (esempio2)

$k = 4$

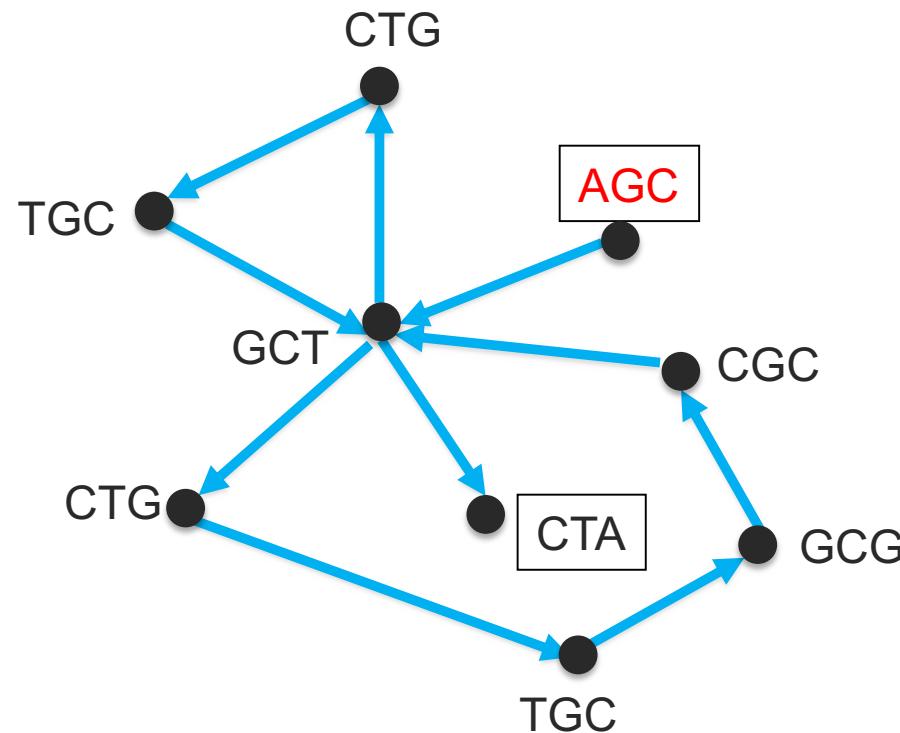


Assembly → **AGC**TGCTGCGCTA

Assembly2 → **AGC**TGCGCTGCT

## Ripetizioni (esempio2)

$k = 4$

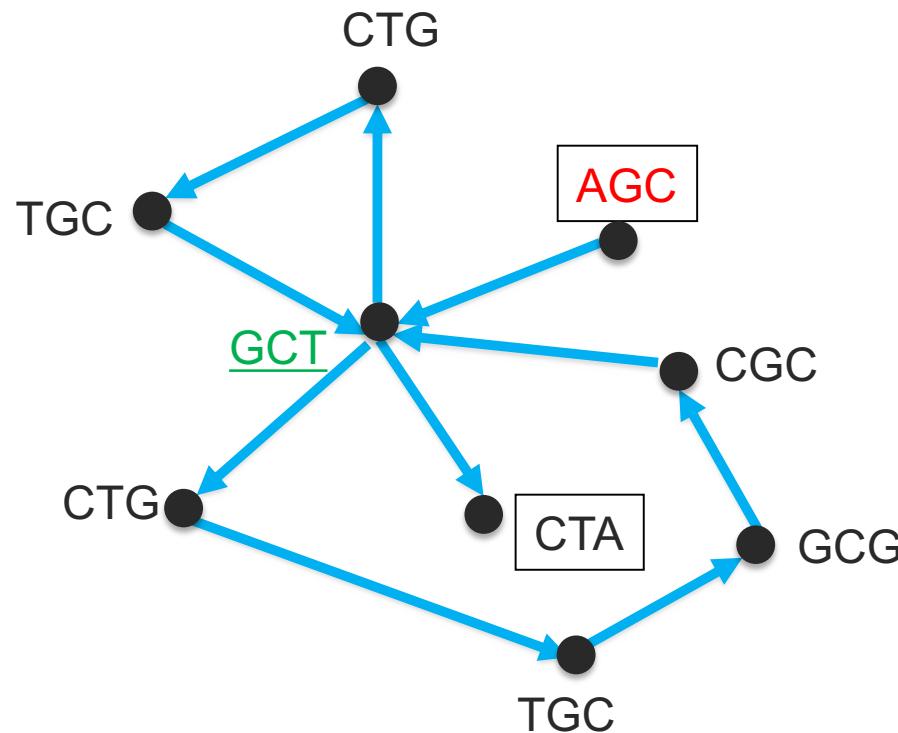


Assembly → **AGC**TGCTGCGCTA

Assembly2 → **AGC**TGCGCTGCTA

# Ripetizioni (esempio2)

$k = 4$



Assembly → AGCTGCTGCGCTA

Assembly2 → AGCTTGCGCTGCTA

# [de Bruijn Graph (assemblaggio)]

Prestare attenzione a:

- ✓ *loop*
- ✓ ripetizioni
- ✓ errori (di sequenziamento)

- ✓ *tips*
- ✓ *bubbles*

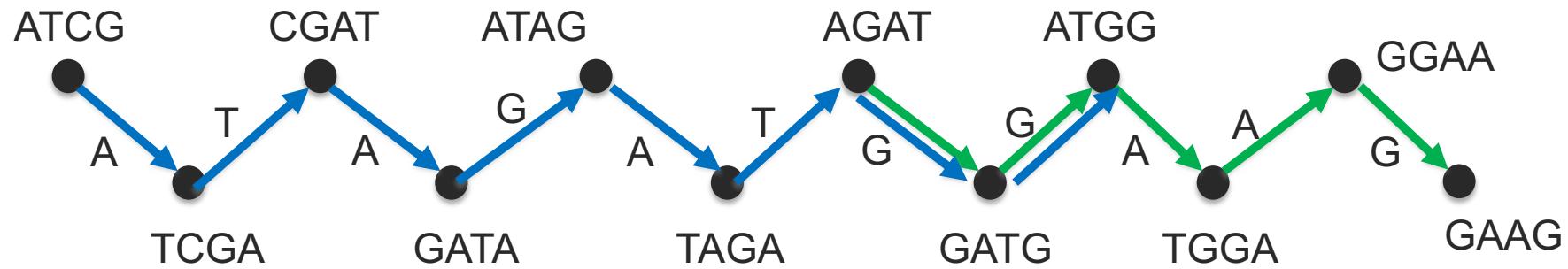
prima di assemblare,  
il dBG viene ripulito...

## [Tips]

ATCGATAGATGG

AGATGGAAAG

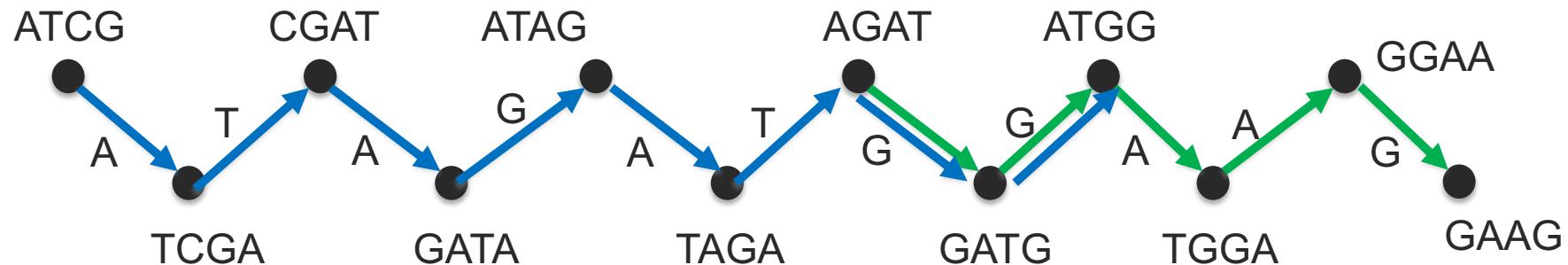
$k = 5$



## [Tips]

ATCGATAGATGG  
AGATGGAAAG  
ACCGGA

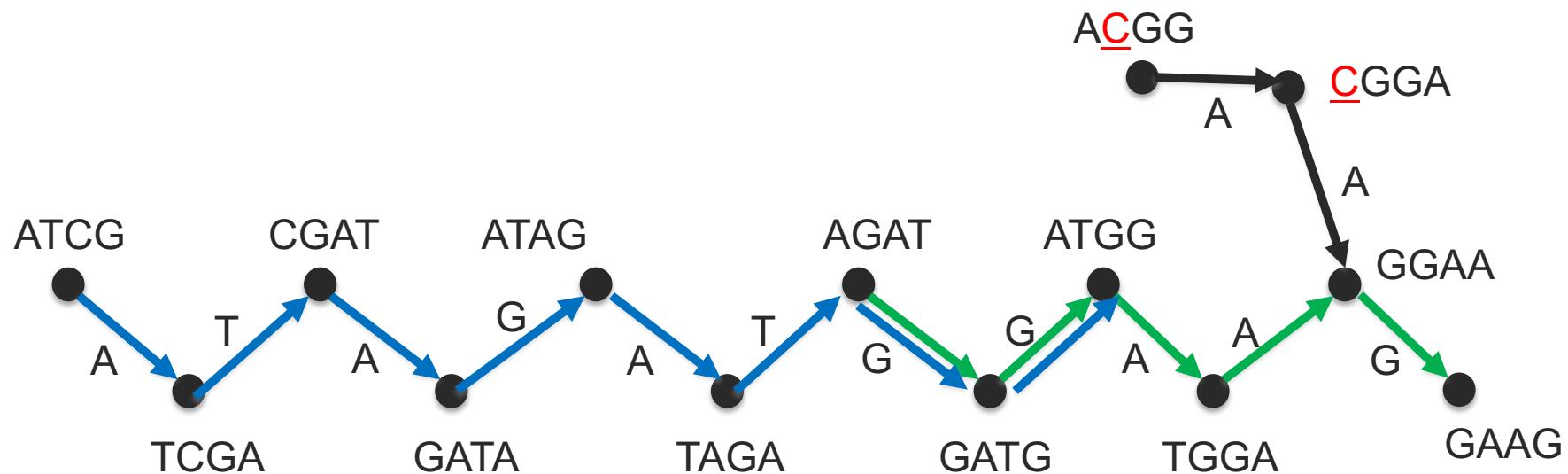
$k = 5$



## [Tips]

ATCGATAGATGG  
AGATGGAAAG  
ACCGGAA

$k = 5$



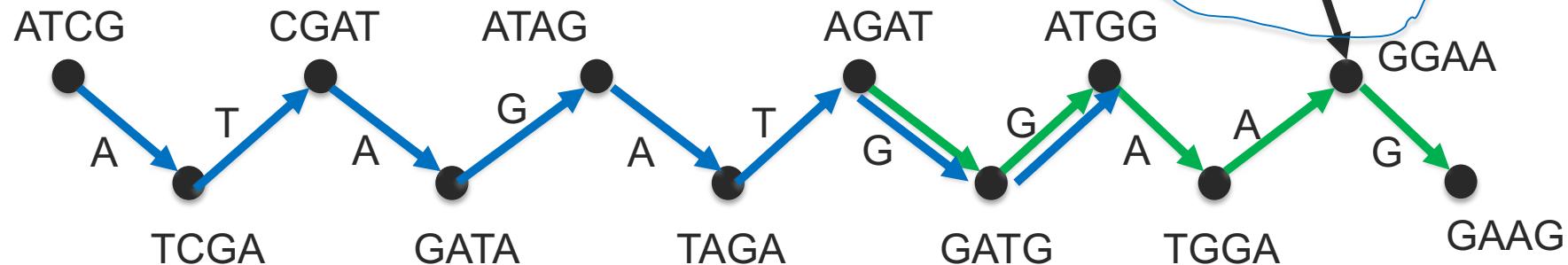
# Tips

]

ATCGATAGATGG

AGATGGAAAG  
ACGGAA

$k = 5$



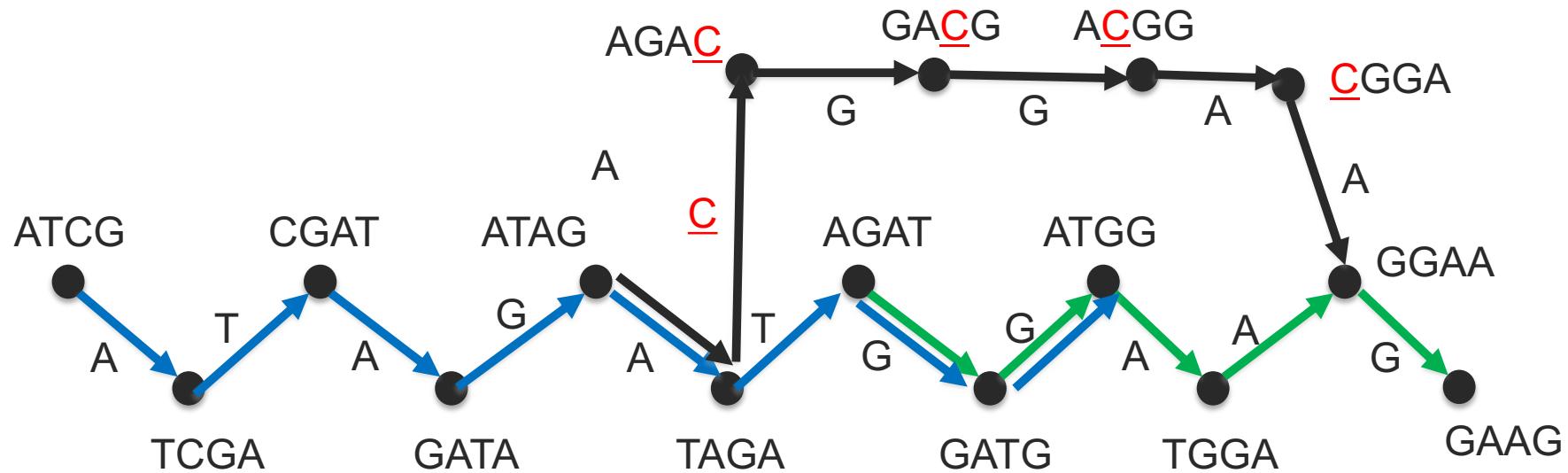
“percorsi” lineari  
che entrano o  
escono dal grafo

# Bubbles

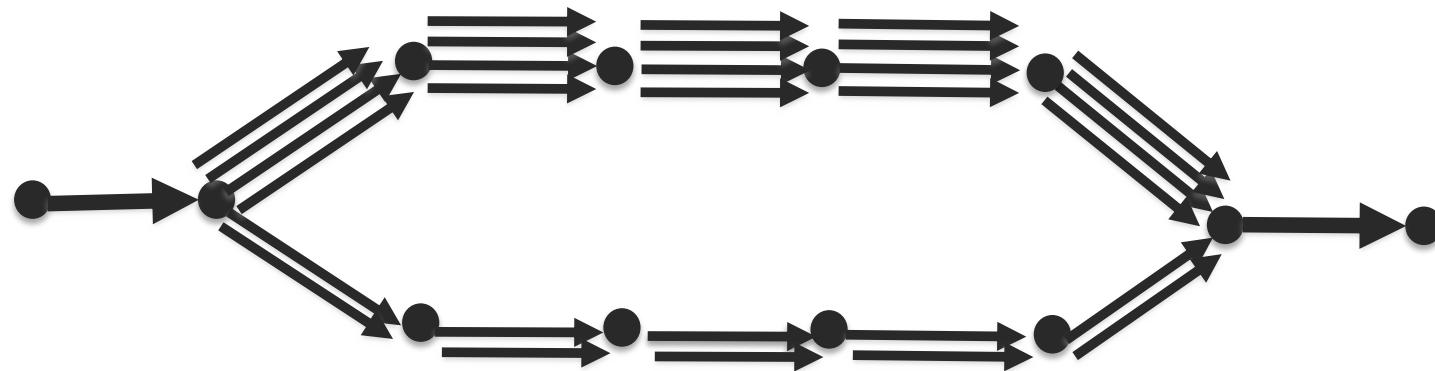
]

ATCGATAGATGG  
AGATGGAAAG  
ATAGACGGAA

$k = 5$



# Bubbles



## Causa delle bubbles:

1. errori di sequenziamento
2. Single Nuclotide Polymorphisms (SNPs)

(posizioni note)

Normalmente si fa un k-mer counting, cioè istogramma delle frequenze nel frammento:

- se pochi, allora sono errori;
- se troppi, sono ripetizioni
- gli intermedi, sono quelli dovuti alla coverage.

Nel caso di errori, usando la coverage scelgo uno dei due percorsi

Nel caso di SNP, sono annotati e quindi scelgo entrambi

## [Overlap Graph **VS** de Bruijn Graph ]

- ✓ lento **VS** veloce
- ✓ reads lunghi (Sanger) **VS** short reads (NGS)
- ✓ poco sensibile alle ripetizioni **VS** sensibile alle ripetizioni (per k basso)
- ✓ overlap variabile **VS** overlap fisso
- ✓ con dBG viene persa a volte la coerenza con i reads

# [Overlap Graph VS de Bruijn Graph ]

## De Bruijn Graph:

- ✓ Tempo di costruzione →  $O(N)$  se riesco ad aggiornare in tempo costante...
- ✓ Spazio occupato →  $O(\min(N, G))$

## Overlap Graph:

- ✓ Tempo di costruzione →  $O(N + a)$
- ✓ Spazio occupato →  $O(N + a)$

N: lunghezza totale dei reads

a: numero di overlaps  $O(n^2)$ , con n numero di reads

pesante

# **Comparison of the two major classes of assembly algorithms: overlap–layout–consensus and de-bruijn-graph**

*Zhenyu Li\*, Yanxiang Chen\*, Desheng Mu\*, Jianying Yuan, Yujian Shi, Hao Zhang, Jun Gan, Nan Li, Xuesong Hu,  
Binghang Liu, Bicheng Yang and Wei Fan*

> *Bioinformatics*. 2013 Apr 15;29(8):1072-5. doi: 10.1093/bioinformatics/btt086.  
Epub 2013 Feb 19.

# QUAST: quality assessment tool for genome assemblies

Alexey Gurevich<sup>1</sup>, Vladislav Saveliev, Nikolay Vyahhi, Glenn Tesler

Affiliations + expand

PMID: 23422339 PMCID: [PMC3624806](#) DOI: [10.1093/bioinformatics/btt086](https://doi.org/10.1093/bioinformatics/btt086)

[Free PMC article](#)

## Abstract

**Summary:** Limitations of genome sequencing techniques have led to dozens of assembly algorithms, none of which is perfect. A number of methods for comparing assemblers have been developed, but none is yet a recognized benchmark. Further, most existing methods for comparing assemblies are only applicable to new assemblies of finished genomes; the problem of evaluating assemblies of previously unsequenced species has not been adequately considered. Here, we present QUAST—a quality assessment tool for evaluating and comparing genome assemblies. This tool improves on leading assembly comparison software with new ideas and quality metrics. QUAST can evaluate assemblies both with a reference genome, as well as without a reference. QUAST produces many reports, summary tables and plots to help scientists in their research and in their publications. In this study, we used QUAST to compare several genome assemblers on three datasets. QUAST tables and plots for all of them are available in the Supplementary Material, and interactive versions of these reports are on the QUAST website.

**Availability:** <http://bioinf.spbau.ru/quast>.

**Supplementary information:** Supplementary data are available at Bioinformatics online.

# [Assemblatori dBG]

Check  
upda

## 2001 An Eulerian path approach to DNA fragment assembly

Pavel A. Pevzner<sup>\*</sup>, Haixu Tang<sup>†</sup>, and Michael S. Waterman<sup>†‡§</sup>

\*Department of Computer Science and Engineering, University of California, San Diego, La Jolla, CA; and Departments of <sup>†</sup>Mathematics and

<sup>‡</sup>Biological Sciences, University of Southern California, Los Angeles, CA

Contributed by Michael S. Waterman, June 7, 2001

For the last 20 years, fragment assembly in DNA sequencing followed the “overlap–layout–consensus” paradigm that is used in all currently available assembly tools. Although this approach proved useful in assembling clones, it faces difficulties in genomic shotgun assembly. We abandon the classical “overlap–layout–consensus” approach in favor of a new EULER algorithm that, for the first time, resolves the 20-year-old “repeat problem” in fragment assembly. Our main result is the reduction of the fragment assembly to a variation of the classical Eulerian path problem that allows one to generate accurate solutions of large-scale sequencing problems. EULER, in contrast to the CELERA assembler, does not mask such repeats but uses them instead as a powerful fragment assembly tool.

Because the Eulerian path approach transforms a once difficult layout problem into a simple one, a natural question is: “Could the Eulerian path approach be applied to fragment assembly?” Idury and Waterman, mimicked fragment assembly as an SBH problem (11) by representing every read of length  $n$  as a collection of  $n - l + 1$  overlapping  $l$ -tuples (continuous short strings of fixed length  $l$ ). At first glance, this transformation of every read into a collection of  $l$ -tuples (breaking the puzzle into smaller pieces) is a very short-sighted procedure, because information about the sequencing reads is lost. However, the loss of information is minimal for large  $l$  and is well paid for by the computational advantages of the Eulerian path approach. In addition, lost information can be restored at later stages.

## EULER e EULER-SR



short read

# Assemblatori dBG

2008

## Velvet: Algorithms for de novo short read assembly using de Bruijn graphs

Daniel R. Zerbino and Ewan Birney<sup>1</sup>

*EMBL-European Bioinformatics Institute, Wellcome Trust Genome Campus, Hinxton, Cambridge CB10 1SD, United Kingdom*

We have developed a new set of algorithms, collectively called “Velvet,” to manipulate de Bruijn graphs for genomic sequence assembly. A de Bruijn graph is a compact representation based on short words ( $k$ -mers) that is ideal for high coverage, very short read [25–50 bp] data sets. Applying Velvet to very short reads and paired-ends information only, one can produce contigs of significant length, up to 50-kb N50 length in simulations of prokaryotic data and 3-kb N50 on simulated mammalian BACs. When applied to real Solexa data sets without read pairs, Velvet generated contigs of ~8 kb in a prokaryote and 2 kb in a mammalian BAC, in close agreement with our simulated results without read-pair information. Velvet represents a new approach to assembly that can leverage very short reads in combination with read pairs to produce useful assemblies.

[Supplemental material is available online at [www.genome.org](http://www.genome.org). The code for Velvet is freely available, under the GNU Public License, at <http://www.ebi.ac.uk/~zerbino/velvet>.]

N50= the length of the shortest contig for which longer and equal length contigs cover at least 50 % of the assembly.

NG50 resembles N50 except the metric relates to the genome size rather than the assembly size.

# Assemblatori dBG

2008

## ALLPATHS: De novo assembly of whole-genome shotgun microreads

Jonathan Butler,<sup>1</sup> Iain MacCallum,<sup>1</sup> Michael Kleber,<sup>1,6</sup> Ilya A. Shlyakhter,<sup>1</sup> Matthew K. Belmonte,<sup>1,2</sup> Eric S. Lander,<sup>1,3,4,5</sup> Chad Nusbaum,<sup>1</sup> and David B. Jaffe<sup>1,7</sup>

<sup>1</sup>Broad Institute of MIT and Harvard, Cambridge, Massachusetts 02141, USA; <sup>2</sup>Department of Human Development, Cornell University, Ithaca, New York 14853, USA; <sup>3</sup>Whitehead Institute for Biomedical Research, Cambridge, Massachusetts 02139, USA;

<sup>4</sup>Department of Biology, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139, USA; <sup>5</sup>Department of Systems Biology, Harvard Medical School, Boston, Massachusetts 02115, USA

New DNA sequencing technologies deliver data at dramatically lower costs but demand new analytical methods to take full advantage of the very short reads that they produce. We provide an initial, theoretical solution to the challenge of de novo assembly from whole-genome shotgun “microreads.” For 11 genomes of sizes up to 39 Mb, we generated high-quality assemblies from 80x coverage by paired 30-base simulated reads modeled after real Illumina-Solexa reads. The bacterial genomes of *Campylobacter jejuni* and *Escherichia coli* assemble optimally, yielding single perfect contigs, and larger genomes yield assemblies that are highly connected and accurate. Assemblies are presented in a graph form that retains intrinsic ambiguities such as those arising from polymorphism, thereby providing information that has been absent from previous genome assemblies. For both *C. jejuni* and *E. coli*, this assembly graph is a single edge encompassing the entire genome. Larger genomes produce more complicated graphs, but the vast majority of the bases in their assemblies are present in long edges that are nearly always perfect. We describe a general method for genome assembly that can be applied to all types of DNA sequence data, not only short read data, but also conventional sequence reads.

[Supplemental material is available online at [www.genome.org](http://www.genome.org).]

# Assemblatori dBG

2009 ABySS: A parallel assembler for **short** read sequence data

Jared T. Simpson,<sup>1</sup> Kim Wong, Shaun D. Jackman, Jacqueline E. Schein,  
Steven J.M. Jones, and İnanç Birol<sup>2</sup>

*Genome Sciences Centre, British Columbia Cancer Agency, Vancouver, British Columbia V5Z 4E6, Canada*

Widespread adoption of massively parallel deoxyribonucleic acid (DNA) sequencing instruments has prompted the recent development of de novo short read assembly algorithms. A common shortcoming of the available tools is their inability to efficiently assemble vast amounts of data generated from large-scale sequencing projects, such as the sequencing of individual human genomes to catalog natural genetic variation. To address this limitation, we developed ABySS (Assembly By Short Sequences), a parallelized sequence assembler. As a demonstration of the capability of our software, we assembled 3.5 billion paired-end reads from the genome of an African male publicly released by Illumina, Inc. Approximately 2.76 million contigs  $\geq$ 100 base pairs (bp) in length were created with an N50 size of 1499 bp, representing 68% of the reference human genome. Analysis of these contigs identified polymorphic and novel sequences not present in the human reference assembly, which were validated by alignment to alternate human assemblies and to other primate genomes.

[Supplemental material is available online at [www.genome.org](http://www.genome.org). Software binaries and instructions are available at <http://www.bcgsc.ca/platform/bioinfo/software/abyss>.]

vediamo il pdf a  
fine lezione (usa  
Bloom Filters)

2012

# Assemblatori dBG

## SPAdes: A New Genome Assembly Algorithm and Its Applications to Single-Cell Sequencing

ANTON BANKEVICH,<sup>1,2</sup> SERGEY NURK,<sup>1,2</sup> DMITRY ANTIPOV,<sup>1</sup> ALEXEY A. GUREVICH,<sup>1</sup>  
MIKHAIL DVORKIN,<sup>1</sup> ALEXANDER S. KULIKOV,<sup>1,3</sup> VALERY M. LESIN,<sup>1</sup>  
SERGEY I. NIKOLENKO,<sup>1,3</sup> SON PHAM,<sup>4</sup> ANDREY D. PRJIBELSKI,<sup>1</sup> ALEXEY V. PYSHKIN,<sup>1</sup>  
ALEXANDER V. SIROTKIN,<sup>1</sup> NIKOLAY VYAHHI,<sup>1</sup> GLENN TESLER,<sup>5</sup>  
MAX A. ALEKSEYEV,<sup>1,6</sup> and PAVEL A. PEVZNER<sup>1,4</sup>

### ABSTRACT

The lion's share of bacteria in various environments cannot be cloned in the laboratory and thus cannot be sequenced using existing technologies. A major goal of single-cell genomics is to complement gene-centric metagenomic data with whole-genome assemblies of uncultivated organisms. Assembly of single-cell data is challenging because of highly non-uniform read coverage as well as elevated levels of sequencing errors and chimeric reads. We describe SPAdes, a new assembler for both single-cell and standard (multicell) assembly, and demonstrate that it improves on the recently released E + V – SC assembler (specialized for single-cell data) and on popular assemblers Velvet and SoapDeNovo (for multicell data). SPAdes generates single-cell assemblies, providing information about genomes of uncultivable bacteria that vastly exceeds what may be obtained via traditional metagenomics studies. SPAdes is available online (<http://bioinf.spbau.ru/spades>). It is distributed as open source software.

**Key words:** assembly, de Bruijn graph, single cell, sequencing, bacteria



RESEARCH

Open Access

# Space-efficient and exact de Bruijn graph representation based on a Bloom filter

Rayan Chikhi<sup>1\*</sup> and Guillaume Rizk<sup>2</sup>

## Abstract

**Background:** The de Bruijn graph data structure is widely used in next-generation sequencing (NGS). Many programs, e.g. *de novo* assemblers, rely on in-memory representation of this graph. However, current techniques for representing the de Bruijn graph of a human genome require a large amount of memory ( $\geq 30$  GB).

**Results:** We propose a new encoding of the de Bruijn graph, which occupies an order of magnitude less space than current representations. The encoding is based on a Bloom filter, with an additional structure to remove critical false positives.

**Conclusions:** An assembly software implementing this structure, Minia, performed a complete *de novo* assembly of human genome short reads using 5.7 GB of memory in 23 hours.

**Keywords:** *de novo* assembly, de Bruijn graph, Bloom filter

Abyss 2.0

vediamo il pdf a  
fine lezione

Received December 28, 2021, accepted January 11, 2022, date of publication January 18, 2022, date of current version January 27, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3144113

# Graph Theoretical Strategies in De Novo Assembly

**KIMIA BEHIZADI<sup>✉</sup>, NAFISEH JAFARZADEH<sup>✉</sup>, AND ALI IRANMANESH**

Department of Mathematics, Faculty of Mathematical Sciences, Tarbiat Modares University, Tehran 14115-111, Iran

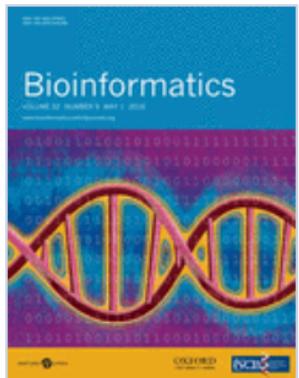
Corresponding author: Ali Iranmanesh (iranmanesh@modares.ac.ir)

This work was supported by the Research Core Bio-Mathematics with Computational Approach of Tarbiat Modares University under Grant IG-39706.

**ABSTRACT** De novo genome assemblers assume the reference genome is unavailable, incomplete, highly fragmented, or significantly altered as in cancer tissues. Algorithms for de novo assembly have been developed to deal with and assemble a large number of short sequence reads from genome sequencing. In this review paper, we have provided an overview of the graph-theoretical side of de novo genome assembly algorithms. We have investigated the construction of fourteen graph data structures related to OLC-based and DBG-based algorithms in order to compare and discuss their application in different assemblers. In addition, the most significant and recent genome de novo assemblers are classified according to the extensive variety of original, generalized, and specialized versions of graph data structures.



**FIGURE 2.** Diagram of graph data structures in DBG method. Overview of how to construct a de-Brujin graph in DBG-based algorithm of de novo genome assemblers and how to obtain each of eleven generalized versions of de-Brujin graph.



Volume 32, Issue 9  
May 2016

[Article Contents](#)

JOURNAL ARTICLE

# Integration of string and de Bruijn graphs for genome assembly FREE

Yao-Ting Huang Chen-Fu Liao [Author Notes](#)

*Bioinformatics*, Volume 32, Issue 9, May 2016, Pages 1301–1307,

<https://doi.org/10.1093/bioinformatics/btw011>

**Published:** 10 January 2016 **Article history ▾**

PDF Split View Cite Permissions Share ▾

# Graph Neural Network Meets de Bruijn Genome Assembly

Publisher: IEEE

Cite This



Mario Simunovic ; Lovro Vrcek ; Mile Sikic [All Authors](#)

115

Full

Text Views



## Abstract

### Document Sections

I. Introduction

II. Related Work

III. Problem Setup

IV. Results

V. Conclusion

Show Full Outline ▾

### Authors

## Abstract:

The genome assembly problem, which involves reconstructing the sequence of nucleotides in a DNA molecule from its short, error-prone substrings called reads, is one of the most extensively studied computational problems in biology. The majority of existing solutions for this problem rely on representing input reads in the form of an assembly graph, where each vertex and edge represents a sequence of nucleotides, and the genome corresponds to a path in this graph. However, assembly graphs often contain millions of erroneous edges due to read errors, which significantly complicates the graph analysis. Traditionally, these erroneous edges are detected and removed using a variety of heuristics. However, these heuristics often require manual parameter tuning. In this paper, we propose a novel approach to detect erroneous edges in a specific type of assembly graph called the de Bruijn graph. We employ a graph neural network-based architecture to detect these erroneous edges, replacing the corresponding heuristic step in traditional genome assembly algorithms. To overcome the lack of datasets with known ground truth, we simulate realistic read sets using error profiles learned from real reads. We propose a new method for erroneous edge classification in de Bruijn graph that relies on Graph Neural Networks. In our experiments, we demonstrate that the new method can reliably replace a more complicated heuristic approach and reduce the overall execution time.

DBG usati anche prima dell'avvento delle NGS...

## Sequenziamento per ibridazione

- ✓ Suggerito come metodo alternativo nel 1988
- ✓ Basato su ibridazione
- ✓ Primo DNA microarray nel 1994
  - ✓ superficie a cui vengono attaccati dei *probes*
  - ✓ viene applicata una soluzione contenente il frammento da sequenziare che è stato etichettato con sostanza fluorescente
  - ✓ ognuno dei *probes* ibridizza con la sua versione complementare del frammento
  - ✓ tramite esame radiografico si vanno a determinare i *probes* che hanno ibridato con il frammento, per ottenere lo spettro complementare del frammento
  - ✓ Si ricostruisce il frammento

## probes di lunghezza 4

# Sequenziamento per ibridazione

# Sequenziamento per ibridazione

	AA	AC	AG	AT	CA	CC	CG	CT	GA	GC	GG	GT	TA	TC	TG	TT
AA																
AC																
AG																
AT																
CA																
CC																
CG																
CT																
GA																
GC																
GG																
TA																
TC																
TG																
TT																

4-mer CGCC

# Sequenziamento per ibridazione

	AA	AC	AG	AT	CA	CC	CG	CT	GA	GC	GG	GT	TA	TC	TG	TT
AA																
AC																
AG																
AT																
CA																
CC																
CG																
CT																
GA																
GC																
GG																
TA																
TC																
TG																
TT																

DNA template → TATCCGTTT

# Sequenziamento per ibridazione

	AA	AC	AG	AT	CA	CC	CG	CT	GA	GC	GG	GT	TA	TC	TG	TT
AA																
AC																
AG																
AT																
CA																
CC																
CG																
CT																
GA																
GC																
GG																
TA																
TC																
TG																
TT																

DNA template → TATCCGTTT

Complement → ATAGGCAAA

# Sequenziamento per ibridazione

	AA	AC	AG	AT	CA	CC	CG	CT	GA	GC	GG	GT	TA	TC	TG	TT
AA																
AC																
AG																
AT																
CA																
CC																
CG																
CT																
GA																
GC																
GG																
TA																
TC																
TG																
TT																

DNA template → TATCCGTTT

Complement → ATAGGCAAA

Spettro (complementare) di ordine 4:  
ATAG, TAGG, AGGC, GGCA, GCAA, CAAA

ma se k è 30....

# Sequenziamento per ibridazione

	AA	AC	AG	AT	CA	CC	CG	CT	GA	GC	GG	GT	TA	TC	TG	TT
AA																
AC																
AG																
AT																
CA																
CC																
CG																
CT																
GA																
GC																
GG																
TA																
TC																
TG																
TT																

Spettro (complementare) di ordine 4:  
ATAG, TAGG, AGGC, GGCA, GCAA, CAAA  
+dBG ho la ricostruzione...