

Nubeam-dedup: **un metodo efficiente per la deduplicazione delle sequenze di DNA**

Veltri Lorenzo, Saguto Federica, Sestito Martina

Università di Roma, La Sapienza

Nubeam-dedup è un algoritmo progettato per rimuovere i duplicati di sequenze di DNA senza l'uso di un genoma di riferimento.

- Utilizza operazioni su matrici per generare il numero **Nubeam**.
- Basato su una funzione hash (numero **Nubeam**) per le sequenze di DNA.
- Riduce l'uso della RAM e accelera il processo di deduplicazione.

L'algoritmo Nubeam-Dedup

- Rappresentazione delle sequenze in forma binaria.

ATCAGC
100100010000001001000010

- Costruzione di matrici per ogni nucleotide.

$$M_1 = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$

$$M_0 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$$



- Calcolo del numero **Nubeam** tramite prodotto di matrici.

$$\text{tr}(WM_B) = \text{tr} \begin{bmatrix} 1 & \sqrt{3} \\ \sqrt{2} & \sqrt{5} \end{bmatrix} \begin{bmatrix} 6007 & 3296 \\ 4424 & 2423 \end{bmatrix} = 23740.35$$

- Deduplicazione basata sul confronto tra numeri Nubeam:
 - metodo single-end:** sequenze con lo stesso numero Nubeam vengono considerate duplicati
 - metodo paired-end:** una coppia di sequenze è considerata duplicato solo se i numeri Nubeam di entrambe le sequenze sono uguali a entrambi i numeri Nubeam di un'altra coppia di sequenze

• struttura del file fastQ:

- codice identificativo contrassegnato da una "@" iniziale;
- sequenza di DNA formata da basi azotate;
- codice identificativo identico al precedente contrassegnato da un "+" iniziale (opzionale);
- valutazione di qualità della sequenza

I dati necessari per la deduplicazione sono il **codice identificativo** (@) e la **sequenza** di nucleotidi

- si caricano i pacchetti necessari (**os**, **math**, **pyspark**) e si apre la connessione con Spark
- si definiscono le funzioni **numero_Nubeam_da_sequenza_basi_azotate()** e **complementare_sequenza()** per calcolare il **numero di Nubeam** e il **complementare** di una sequenza

Implementazione su Spark - sistemazione dei dati

(1)

```
dDb = sc.textFile()
```

`['@...', 'sequenza', '+...', 'qualità']`

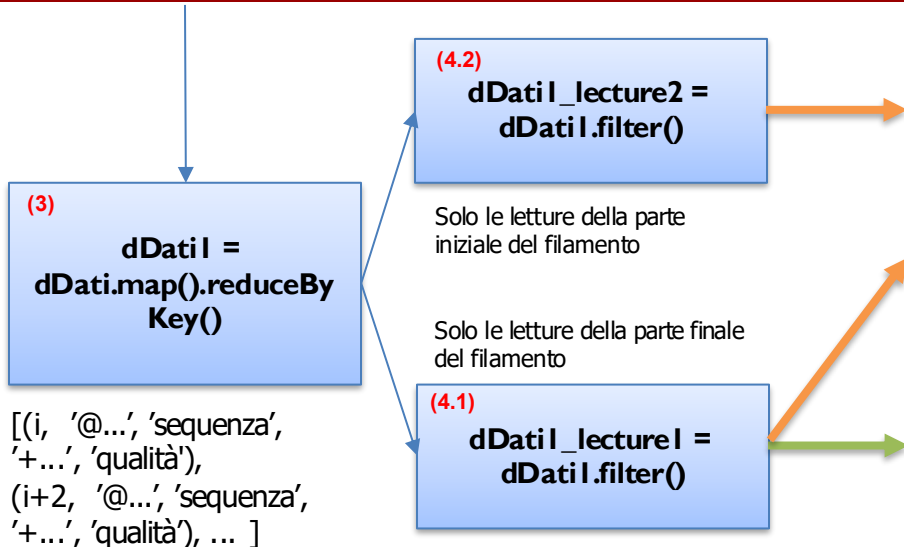
(2)

```
dDati =  
dDb.zipWithIndex()
```

`[('@...', 0), ('sequenza', 1),
('+...', 2), ('qualità', 3)]`



Implementazione su Spark - sistemazione dei dati



Implementazione su Spark - metodo Single-end

(5s)

```
dNubeamSE =  
dDatiI_lectureI.map()
```

`[(numero nubeam, 'sequenza'),
...]`

(6s)

```
dNubeam_compSE =  
dDatiI_lectureI.map()
```


`[(numero nubeam
complementare, 'sequenza
complementare'), ...]`



SAPIENZA
UNIVERSITÀ DI ROMA

Implementazione su Spark - metodo Single-end

(7s)



```
dSE =  
dNubeamSE.union(dN  
ubeam_compSE)
```

`[(numero nubeam, 'sequenza'),
... ,(numero nubeam
complementare, 'sequenza
complementare'), ...]`

(8s)

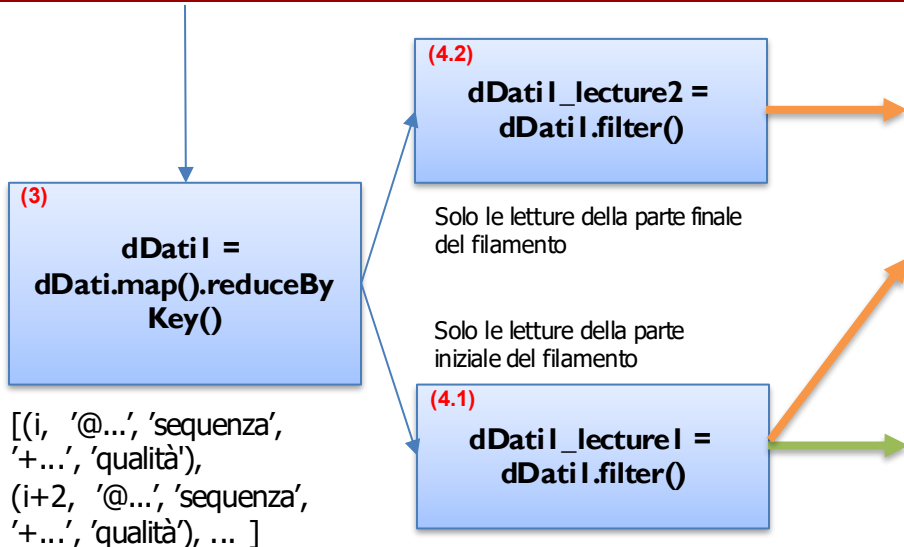


```
dUnordered_set =  
dSE.distinct()
```

Lista delle sequenze e delle
sequenze complementari senza i
duplicati



Implementazione su Spark - sistemazione dei dati



Implementazione su Spark - Paired-end

(5.2p)

**dDati l_lecture2_unif
=
dDati l_lecture2.map()**

[('@...', ['sequenza']), ...]

Ad ogni identificativo ('@...') della lettura viene tolta la sequenza alfanumerica (/2 o 2:N:0:CAGATC) che indica che quella lettura proviene dalla parte finale del filamento.

(5.1p)

**dDati l_lecture1_unif
=
dDati l_lecture1.map()**


[('@...', ['sequenza']), ...]

Ad ogni identificativo ('@...') della lettura viene tolta la sequenza alfanumerica (/1 o 1:N:0:CAGATC) che indica che quella lettura proviene dalla parte iniziale del filamento.



[('@...', ['sequenza']), ...]
Si raggruppano le sequenze con
lo stesso identificativo ('@...')

(6p)



```
dDati1_unif  
= dDati1_lecture1_unif  
.union(dDati1_lecture2  
_unif).reduceByKey()
```



Implementazione su Spark - Paired-end

(7p)

```
dNubeamPE =  
dDati l_unif.filter().ma  
p()
```

```
[('sequenza 1', nubeam,  
'sequenza 2', nubeam), ... ]
```

(8p)

```
dNubeam_compPE =  
dDati l_unif.filter().ma  
p()
```

```
[('sequenza complementare 1', nubeam complementare,  
'sequenza complementare 2', nubeam complementare), ... ]
```



Implementazione su Spark - Paired-end

(10p)

```
dPE_Unici =  
dPE.distinct()
```

Lista delle sequenze e delle sequenze complementari senza i duplicati

(9p)

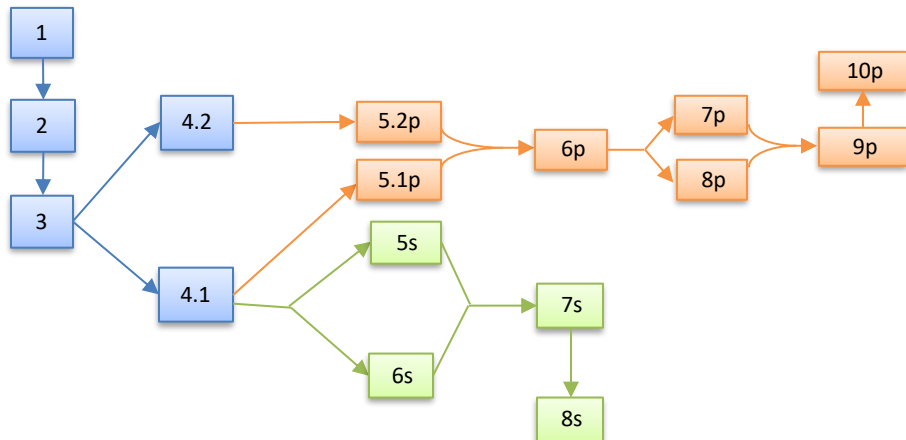
```
dPE =  
dNubeamPE.union(dNubeam_compPE)
```

Si uniscono le sequenze e le sequenze complementari, con i rispettivi nubeam, in un'unica RDD



SAPIENZA
UNIVERSITÀ DI ROMA

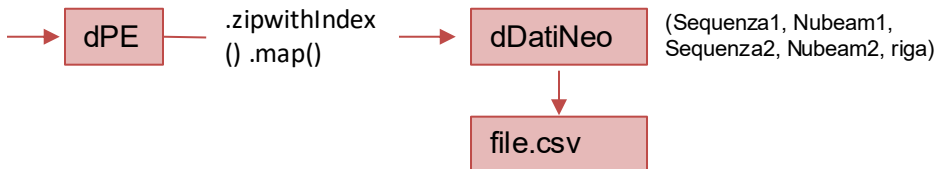
Implementazione su Spark – Schema riassuntivo



Blu: Preparazione dei dati
Arancione: Metodo Paired-End
Verde: Metodo Single-End

Implementazione su Neo4j - Creazione del dataset

- Si stabilisce la connessione con **Neo4j** tramite **PySpark**
- Si sistemano i dati, tramite PySpark, per poter creare un file .csv da utilizzare come dataset da caricare in Neo4j



- Si carica il file.csv su Neo4j

Implementazione su Neo4j - Creazione dei nodi

Si creano:

- un nodo per ogni sequenza iniziale, per ogni sequenza finale e per i loro complementari (**284500 nodi**)

```
LOAD CSV WITH HEADERS FROM 'file:///data1.csv' AS line
CREATE (s1:SEQUENZA1{seq: line.sequenza1, n1: line.nubeam1,
    r1: line.riga}),
    (s2:SEQUENZA2{seq: line.sequenza1, n2: line.nubeam2,
    r2: line.riga})
```

- un nodo per ogni numero nubeam delle sequenze iniziali, eliminando i duplicati grazie alla funzione DISTINCT di Neo4j (**132520 nodi**);
- un nodo per ogni numero nubeam delle sequenze finali, eliminando i duplicati grazie alla funzione DISTINCT di Neo4j (**132866 nodi**);



Si creano:

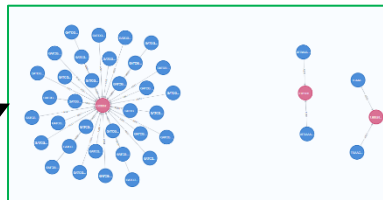
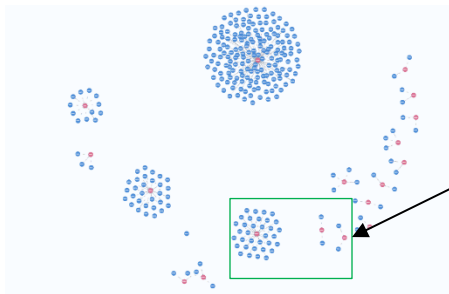
- un arco, di tipo *Ass1*, che colleghi ogni sequenza iniziale al corrispondente numero Nubeam (**142250 archi**);

```
MATCH (n1), (s1)
WHERE s1.n1 = n1.nubeam_id1
CREATE (s1)-[a: ASS1]->(n1)
```

- un arco, di tipo *Ass2*, che colleghi ogni sequenza finale al corrispondente numero Nubeam (**142250 archi**);
- un arco, di tipo *Coppia*, che colleghi ogni sequenza iniziale alla corrispondente sequenza finale (**142250 archi**).

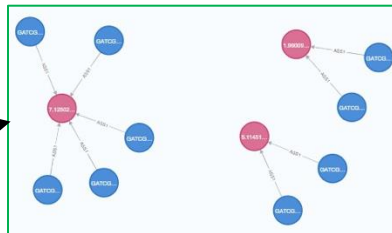
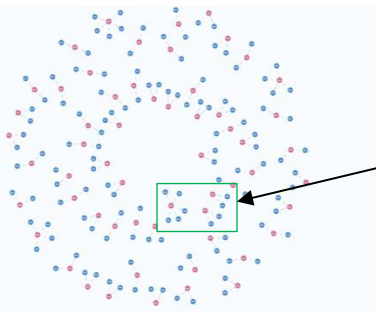
Interrogazione del DataBase – Single-End «Giocattolo»

```
MATCH p=(n1)-[r:ASS1]->(s1)  
WITH n1, collect(p) AS percorsi  
WHERE size(percorsi)>1  
RETURN percorsi
```



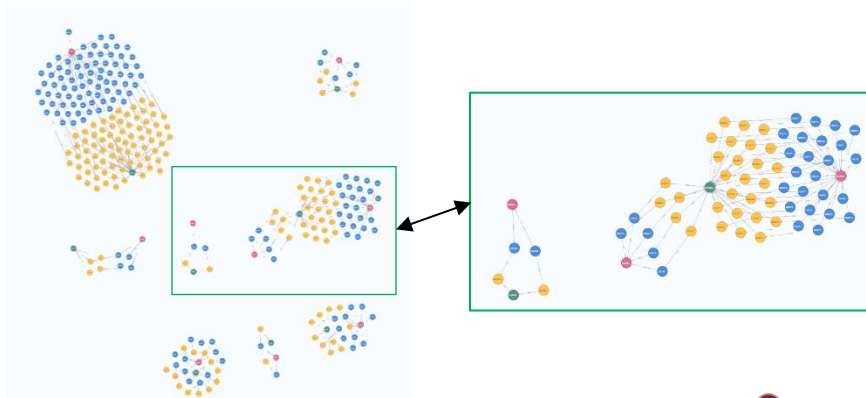
Interrogazione del DataBase – Single-End «Girasoli»

```
MATCH p=(n1)-[r:ASS1]->(s1)  
WITH n1, collect(p) AS percorsi  
WHERE size(percorsi)>1  
RETURN percorsi
```



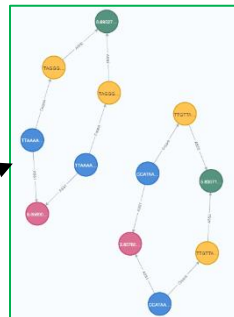
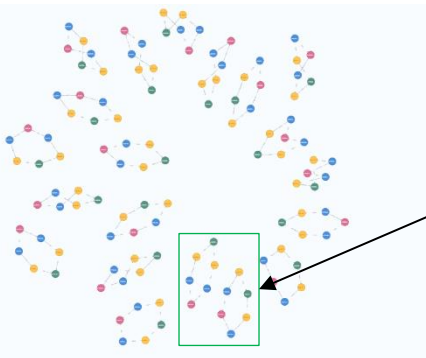
Interrogazione del DataBase – Paired-End «Giocattolo»

```
MATCH (n1), (n2) MATCH p = (n1)-[a1:ASS1]-(s1)-[*]->(n2)
WITH n1, n2, collect(p) AS percorsi
WHERE size(percorsi) >= 2
RETURN n1,n2,percorsi
```



Interrogazione del DataBase – Paired-End «Girasoli»

```
MATCH (n1), (n2) MATCH p = (n1)-[a1:ASS1]-(s1)-[*]->(n2)
WITH n1, n2, collect(p) AS percorsi
WHERE size(percorsi) >= 2
RETURN n1,n2,percorsi
```



Risultati – Analisi statistica

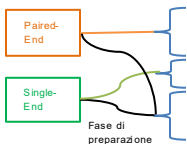
Dataset «giocattolo»

letture	142250	
iniziali	71125	50% del totale
finali	71125	50% del totale
Duplicati SE	9730	13.68% delle letture iniziali 6.84% del totale
Duplicati PE	2252	1.45% del totale

Dataset girasoli

letture	279535	
iniziali	137757	50% del totale
finali	137757	50% del totale
Duplicati SE	4089	2.96% delle letture iniziali 1.46% del totale
Duplicati PE	4041	1.46% del totale

Risultati – Tempi di esecuzione

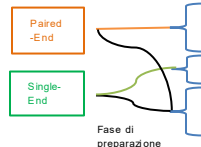


Passo	Descrizione	Durata	Tasks	Dati input	Shuffle read	Shuffle write
5	distinct	31 s	8/8		21.3 MiB	20.0 MiB
3	reduceByKey	0.6 s	4/4		41.9 MiB	10.6 MiB
2	distinct	21 s	4/4		41.9 MiB	9.6 MiB
1	reduceByKey	0,8 s	2/2	37.8 MiB		20.9 MiB
0	zipWithIndex	0,9 s	2/2	37.8 MiB		

Table 1: Risultati computazionali algoritmo Nubeam-dedup dataset "giocattolo"

Tempo (secondi)		
prep	1.7	
SE	21	
PE		31.6
Tot	22.7	33.3

Dati scambiati (MB)		
prep	20.9	
SE	51.5	
PE		93.8
Tot	72.4	114.7



Passo	Descrizione	Durata	Tasks	Dati input	Shuffle read	Shuffle write
5	distinct	1.7 min	12/12		61.9 MiB	59.4 MiB
3	reduceByKey	1 s	6/6		102.2 MiB	30.9 MiB
2	distinct	56 s	6/6		102.2 MiB	29.6 MiB
1	reduceByKey	1 s	3/3	95.2 MiB		51.1 MiB
0	zipWithIndex	1 s	3/3	95.2 MiB		

Table 2: Risultati computazionali algoritmo Nubeam-dedup dataset girasoli

Tempo (secondi)		
prep	2	
SE	56	
PE		118
Tot	58	120

Dati scambiati (MB)		
prep	51.1	
SE	131.8	
PE		254.4
Tot	182.9	305.5