BIOINFORMATICA
II

ALLINEAMENTO GENOMA

Argomenti

01 ALLINEAMENTO AL GENOMA

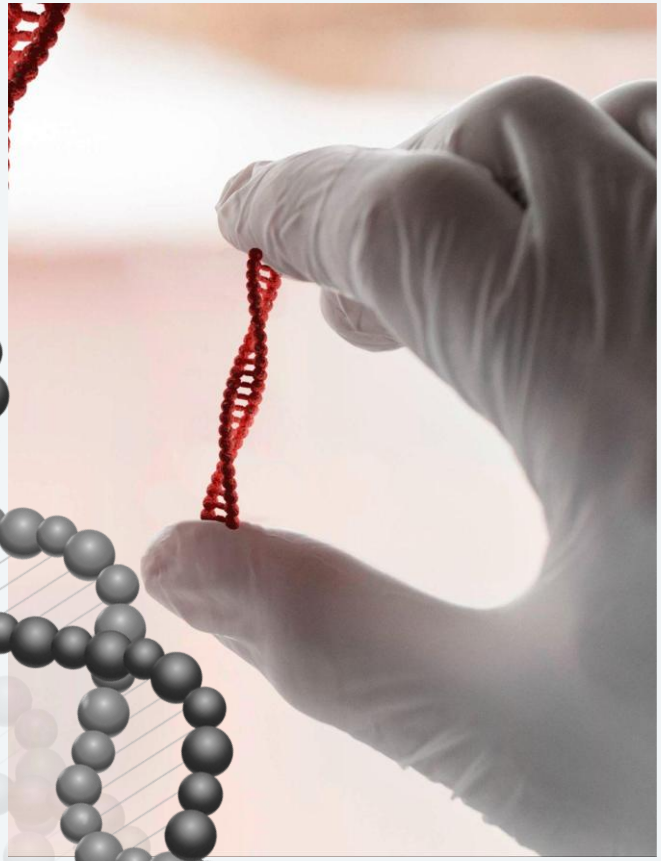
02 ALGORTIMI E PRINCIPI DI COMPLESSITÁ COMPUTAZIONALE

03 ALGORITMO NEEDLEMAN-WUNSCH

04 SPACED SEED

05 TRASFORMATA BURROWS-WHEELER

06 SOFTWARE PER L'ALLINEAMENTO



01

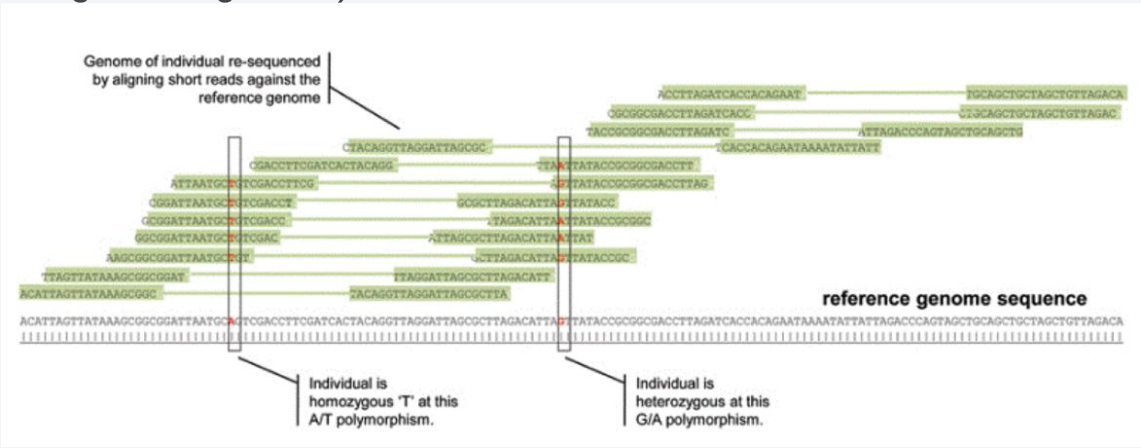
ALLINEAMENTO AL GENOMA

ALLINEAMENTO AL GENOMA

L'allineamento al genoma è una tecnica usata per confrontare due o più sequenze di DNA, RNA o proteine per identificare regioni simili

Gli allineamenti possono essere fatti tra sequenze complete di genomi o tra frammenti di sequenze

Queste somiglianze possono indicare relazioni evolutive, strutture funzionali o aree di interesse biologico (come geni o regolatori)



ALLINEAMENTO AL GENOMA: Utilità

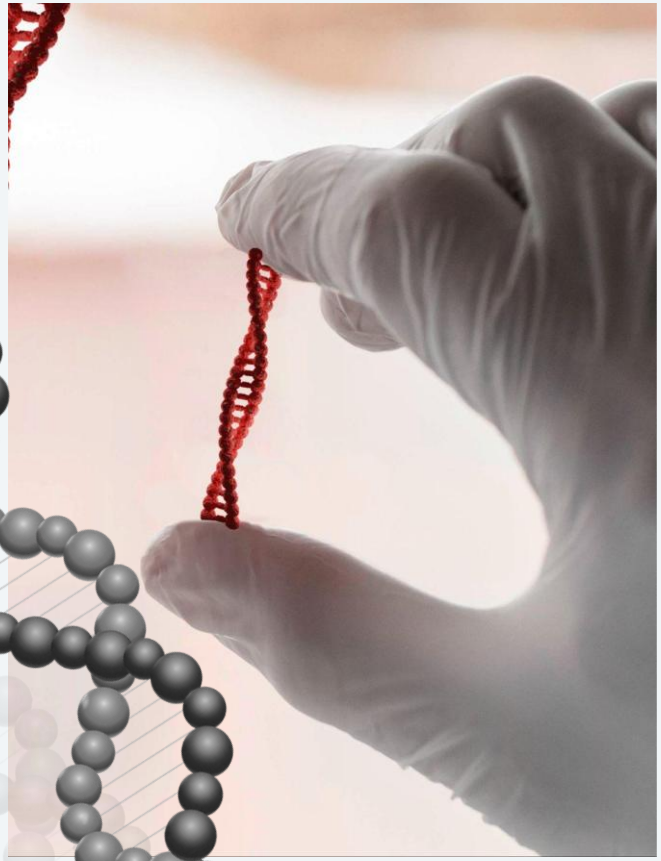
Studio dell'evoluzione: permette di confrontare i genomi di diverse specie per identificare geni comuni e capire la loro divergenza evolutiva
Annotazione del genoma: identifica regioni funzionali come geni, promotori e introni confrontando un genoma con uno già annotato
Analisi della biodiversità: utile per confrontare le sequenze all'interno di una stessa specie o tra specie diverse
Identificazione di mutazioni: utile per trovare varianti genetiche che possono essere associate a malattie
Medicina personalizzata: può individuare varianti genetiche specifiche per guidare trattamenti su misura

ALLINEAMENTO AL GENOMA: Tipi

Allineamento globale: confronta le sequenze lungo tutta la loro lunghezza. È utile quando si ha una buona corrispondenza tra sequenze (es. genomi di specie strettamente correlate)
Algoritmo tipico: Needleman-Wunsch (programmazione dinamica)

Allineamento locale: trova regioni simili in porzioni di sequenze, ignorando altre parti. È utile per sequenze molto diverse
Algoritmi tipici: BTW+FM index; Smith-Waterman (variante di N-W);

Allineamento multiplo: confronta più di due sequenze contemporaneamente. È spesso usato per studi filogenetici o per individuare motivi conservati



02

ALGORTIMI E PRINCIPI DI COMPLESSITÀ COMPUTAZIONALE

ALGORTIMO: Definizione

Un algoritmo informatico è una successione di istruzioni o passi che definiscono le operazioni da eseguire sui dati per ottenere i risultati

Un algoritmo è uno schema in cui:

- I passi che definiscono le operazioni da compiere devono essere “elementari”, ovvero non ulteriormente scomponibili (atomicità);
- Le operazioni da eseguire devono essere interpretabili in modo diretto e univoco dall’esecutore, sia esso umano o artificiale (non ambiguità);
- L’algoritmo deve essere finito, ossia composto da un numero definito di passi correlati ad una quantità definita di dati in ingresso (finitezza);
- L’esecuzione dello schema deve avvenire entro un tempo finito (terminazione);
- L’esecuzione dello schema algoritmico deve condurre ad un unico risultato (effettività)

FONDAMENTI DI ALGORTIMI: Esempio

Problema: Dato un vettore di numeri, calcolare la somma di tutti i valori

Algoritmo:

Inizializza una variabile somma a 0.

Per ogni elemento del vettore:
Aggiungi il valore dell'elemento alla variabile somma

Restituisci la variabile somma.

Input: vettore V di dimensione n

```
somma ← 0
FOR i FROM 0 TO n-1
  somma ← somma + V[i]
END
```

Output: somma

FONDAMENTI DI ALGORTIMI: Esempio

Problema: Dato un vettore di numeri, ordinarlo in ordine crescente

Algoritmo:

Ripeti per $n-1$ iterazioni:

- Per ogni coppia di elementi consecutivi nel vettore:
  Se il primo elemento è maggiore del secondo, scambiali di posto.

Continua fino a quando l'intero vettore è ordinato.

```
Input: vettore V di dimensione n

FOR i FROM 0 TO n-1
  FOR j FROM 0 TO n-2
    IF (V[j] > V[j+1]) THEN Scambia V[j] e V[j+1]
    END_IF
  END_FOR
END_FOR
Output: V
```

COMPLESSITÀ COMPUTAZIONALE

La complessità computazionale studia le risorse necessarie per risolvere un algoritmo
Le risorse principali analizzate sono:
Tempo: il numero di passi/operazioni necessari per eseguire l'algoritmo
Spazio: la quantità di memoria utilizzata durante l'esecuzione

L'obiettivo è classificare i problemi in relazione a quanto velocemente la loro difficoltà cresce rispetto alla dimensione dell'input, indicata con n

COMPLESSITÀ COMPUTAZIONALE

La complessità è espressa usando la notazione asintotica, come la Big-O (O(f(n)), che descrive il comportamento dell'algoritmo per input molto grandi

Quando si analizza la complessità computazionale di un algoritmo, si considera il caso pessimo
O(f(n))), che rappresenta il limite superiore del tempo di esecuzione

COMPLESSITÀ COMPUTAZIONALE

Complessità Costante O(1)
Definizione: Il tempo di esecuzione non dipende dall'input
Descrizione: si elaborano dati in maniera sequenziale

Esempio:
Una operazione aritmetica
La lettura di un elemento di un vettore

COMPLESSITÀ COMPUTAZIONALE

Complessità Lineare O(n)
Definizione: Il tempo di esecuzione cresce in proporzione diretta alla dimensione dell'input (n)
Descrizione: Ogni elemento dell'input è elaborato una volta. Tipico di algoritmi che eseguono una singola scansione (o iterazione) dei dati

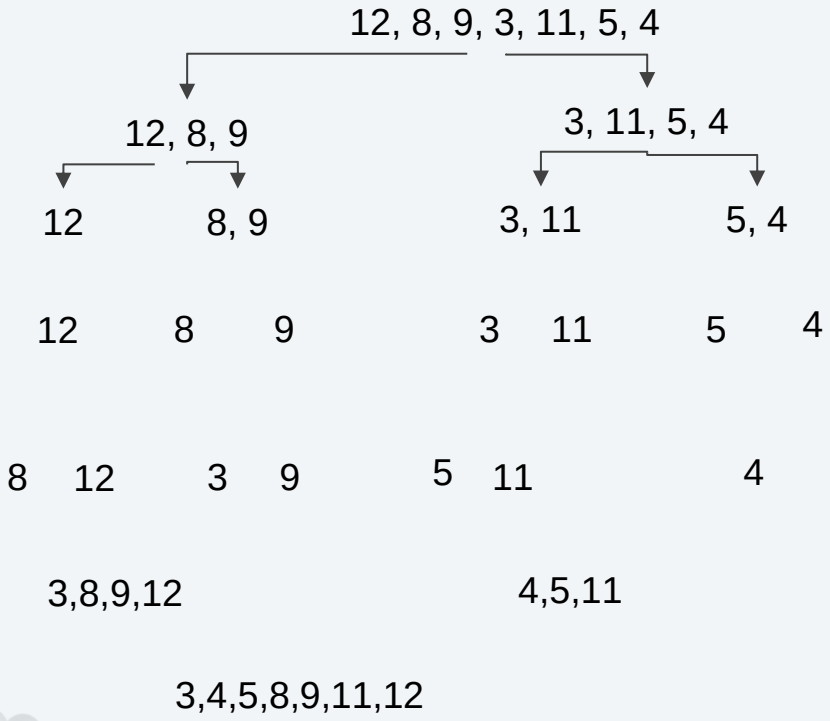
Esempio:
Scorrere un vettore per trovare un elemento.
Calcolare la somma di un vettore di n numeri.

COMPLESSITÀ COMPUTAZIONALE

Complessità Pseudo-lineare O(nlog(n))
Definizione: Il tempo di esecuzione cresce in proporzione alla dimensione dell'input (n) per il valore log(n)
Tipico di algoritmi che dividono e combinano i dati, come nei metodi di ordinamento efficienti

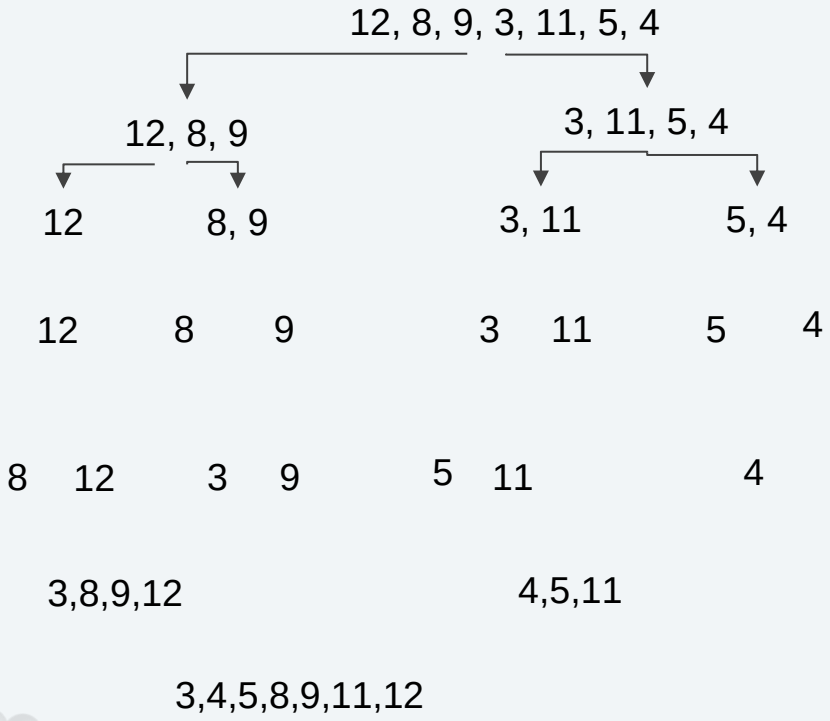
Esempio:
Merge Sort e Quick Sort (ordinamento di array).
Ricerca in un albero bilanciato

COMPLESSITÀ COMPUTAZIONALE




```
#MAIN
# Definizione del vettore da ordinare
arr = [12, 11, 13, 5, 6, 7]
# Calcolo della lunghezza del vettore da ordinare
n = len(arr)
#Stampa del vettore NON ordinato
print("Given array is")
for i in range(n):
    print("%d" % arr[i],end=" ")
#Richiamo del metodo di ordinamento MergeSort
mergeSort(arr, 0, n-1)
#Stampa del vettore  ordinato
print("\n\nSorted array is")
for i in range(n):
    print("%d" % arr[i],end=" ")
```

COMPLESSITÀ COMPUTAZIONALE



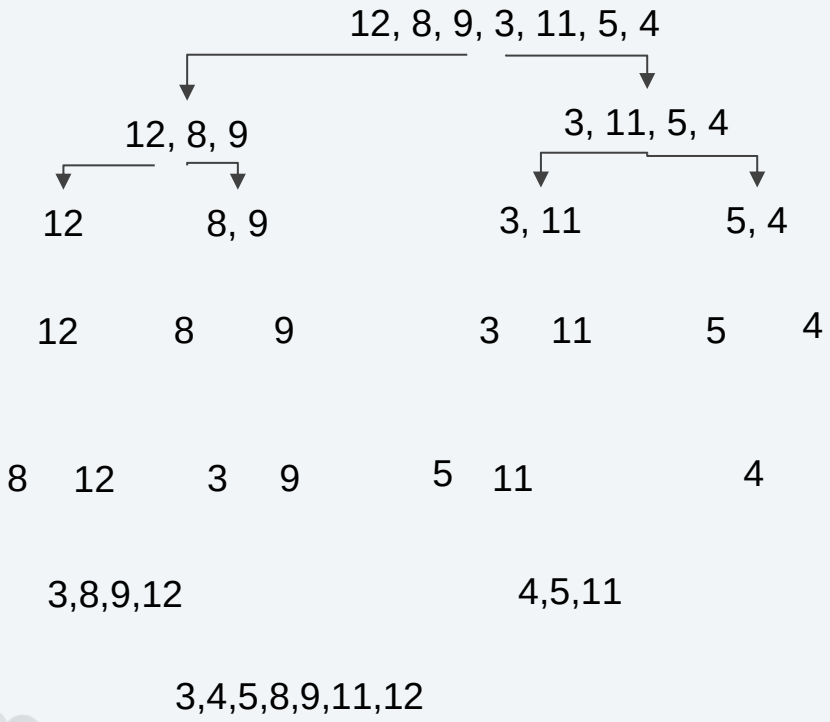

```
# Funzione principale
def mergeSort(arr, l, r):
    if l < r:

        # Si cerca il punto in mezzo del vettore
        m = l+(r-l)//2

        # Si richiama la funzione sulla prima metà dei
        #dati
        mergeSort(arr, l, m)
        # Si richiama la funzione sulla seconda metà
        # dei dati
        mergeSort(arr, m+1, r)

       # Si unisce il risultato
        merge(arr, l, m, r)
```

COMPLESSITÀ COMPUTAZIONALE




```
def merge(arr, l, m, r):
    n1 = m - l + 1
    n2 = r - m
    # crea due vettori temporanee L e R
    L = [0] * (n1)
    R = [0] * (n2)
    # copia i dati di ARR nei due vettori temporanei L e R
    for i in range(0, n1):
        L[i] = arr[l + i]
    for j in range(0, n2):
        R[j] = arr[m + 1 + j]
    # unisci i due vettori temporanei nel vettore i ARR ordinando i valori (parziali)
    i = 0     # indice L
    j = 0     # indice R
    k = l     # indice dell'array unito
    while i < n1 and j < n2:
        if L[i] <= R[j]:
            arr[k] = L[i]
            i += 1
        else:
            arr[k] = R[j]
            j += 1
        k += 1
    # Copia i rimanenti elementi di L in ARR (se ce ne sono)
    while i < n1:
        arr[k] = L[i]
        i += 1
        k += 1
    # Copia i rimanenti elementi di R in ARR (se ce ne sono)
    while j < n2:
        arr[k] = R[j]
        j += 1
        k += 1
```

O(n(log(n))

COMPLESSITÀ COMPUTAZIONALE

Complessità Esponenziale $O(2^n)$
Definizione: Il tempo di esecuzione cresce in modo esponenziale con la dimensione dell'input (n)

Descrizione: Si verifica quando l'algoritmo esplora tutte le possibili combinazioni di input (es. problemi di enumerazione o ottimizzazione).
Tipico di algoritmi non ottimizzati per problemi NP-completi, come il commesso viaggiatore; Backtracking senza ottimizzazioni per risolvere Sudoku o giocare a scacchi; generare tutti i sottoinsiemi di un insieme.

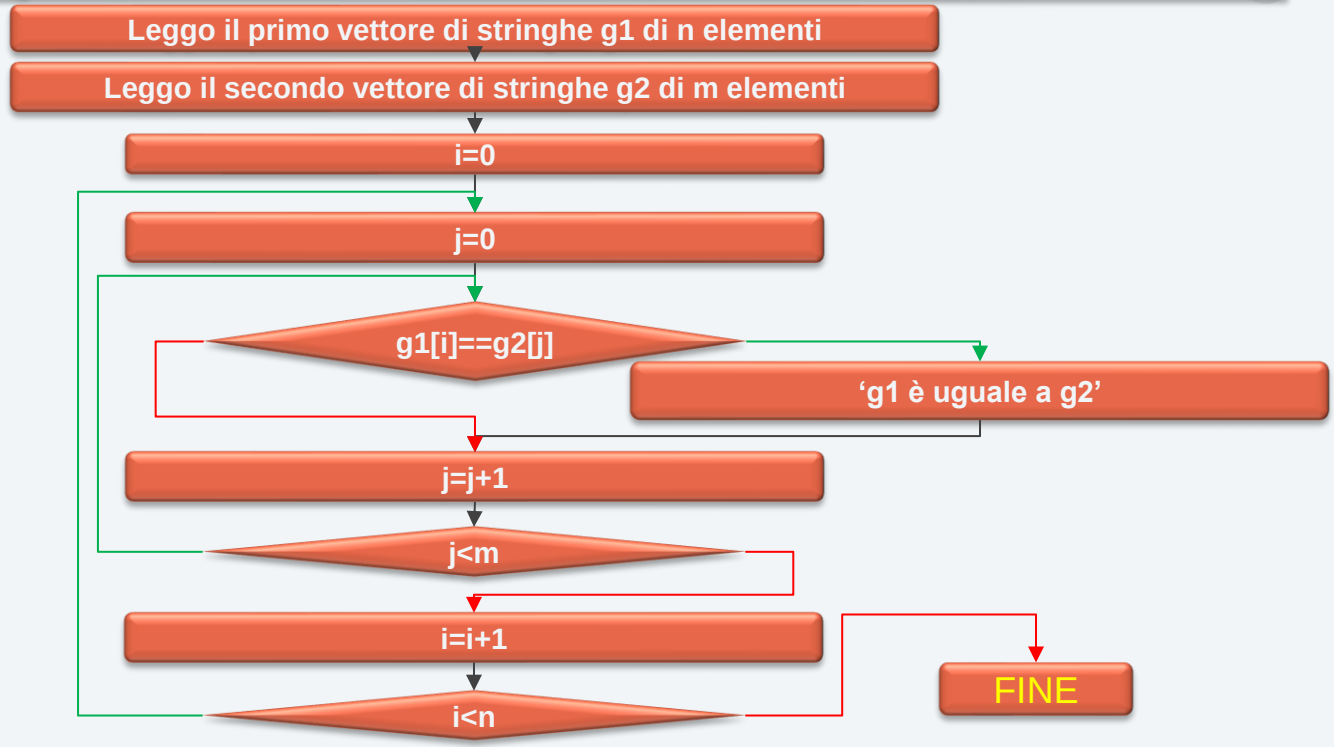
COMPLESSITÀ COMPUTAZIONALE

Esempio di crescita

	O(n)	O(nlogn)	$O(n^2)$	$O(2^n)$
10	10	23	100	1.024
100	100	460	10.000	$1 \cdot 10^{30}$
1000	1.000	6.907	1.000.000	$1 \cdot E^{301}$
10000	10.000	92.103	100.000.000	$2 \cdot E^{3010}$

COMPLESSITÀ COMPUTAZIONALE

Dati due vettori di stringhe confrontare le stringhe del primo vettore con quelle del secondo ogni stringa è lunga k caratteri



COMPLESSITÀ COMPUTAZIONALE

Date due vettori di stringhe di lunghezza n e m e riportare il numero di stringhe (di dimensione k) uguali

```
// Lettura delle stringhe del secondo gruppo
    printf("\nInserisci le stringhe del primo gruppo:\n");
    for (int i = 0; i < n; i++) {
        printf("Stringa %d: ", i + 1);
        scanf("%s", group1[i]);
    }
```

O(n)

```
// Lettura delle stringhe del secondo gruppo
printf("\nInserisci le stringhe del secondo gruppo:\n");
for (int i = 0; i < m; i++) {
    printf("Stringa %d: ", i + 1);
    scanf("%s", group2[i]);
}
```

O(m)

COMPLESSITÀ COMPUTAZIONALE

```
// Confronto tra le stringhe dei due gruppi
   printf("\nConfronto delle stringhe:\n");
   for (int i = 0; i < n; i++) {
       for (int j = 0; j < m; j++) {
           if (strcmp(group1[i], group2[j]) == 0) {
             printf("Stringa '%s' gruppo 1 = stringa '%s' gruppo 2.\n", group1[i], group2[j]);
           }
       }
   }
```

$O(m \cdot n \cdot k)$

```
STRCMP
uguale=true
for (int q = 0; q < k; q++) {
            if (group1[i][q]!=group2[j][q]) {uguale=false;}
        }
    }
```

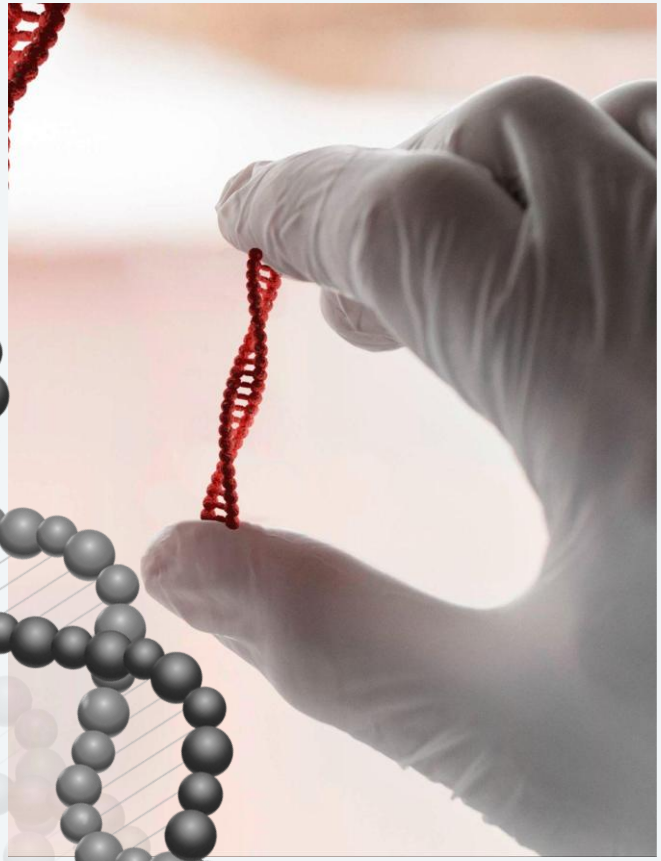
Se m,n e k hanno la stessa dimensione n: $O(n^3)$

COMPLESSITÀ COMPUTAZIONALE

La complessità è: $O(n)+O(n)+O(n^3)$ cioè $O(n^3)$ in quanto dominate sulle altre

Qualora si avessero 100000 reads di lunghezza 10000 da confrontare con altrettanti segmenti del genoma si impiegherebbero 1.000.000.000.000 tempi di calcolo.

Qualora un elaboratore avesse frequenza di 1Ghz
Impiegherebbe 1000 secondi (16 minuti circa)



03

ALGORITMO NEEDLEMAN-WUNSCH

Needleman-Wunsch

L'algoritmo di **Needleman-Wunsch** è uno dei metodi, basato sulla programmazione dinamica, più utilizzati per l'allineamento di due sequenze.
L’algoritmo N-W utilizza una matrice per calcolare il punteggio ottimale e successivamente ricostruire il miglior allineamento

PROGRAMMAZIONE DIMANICA

La **programmazione dinamica** è una tecnica utilizzata per risolvere problemi complessi suddividendoli in sottoproblemi più semplici, e viene comunemente applicata nell'allineamento di sequenze genomiche

Nel contesto dell'allineamento di sequenze, l'obiettivo è trovare il miglior allineamento tra due sequenze (reads e DNA/proteine), tenendo conto di corrispondenze, sostituzioni, inserzioni e delezioni

Needleman-Wunsch: Algoritmo

1. **Definizione della matrice**
    1. Si costruisce una matrice M di dimensioni (n+1)×(m+1), dove n e m sono le lunghezze delle due sequenze da allineare
    2. Alla riga 0-esima si riporta il gap (carattere –); analogamente nella colonna 0-esima si riporta il gap (carattere -)
    3. La riga *i-esima* rappresenta i caratteri della prima sequenza fino alla posizione *i*, mentre la colonna *j-esima* rappresenta i caratteri della seconda sequenza fino alla posizione *j*

Needleman-Wunsch: Algoritmo

Prima sequenza: AGT
Seconda sequenza: GTT

NW Matrix	-	A	G	T
-				
G				
T				
T				

Needleman-Wunsch: Algoritmo

2. Definizione dei punteggi

I punteggi di ***Match Score***, ***Mismatch Penalty*** e ***Gap Penalty*** sono i criteri utilizzati per assegnare un valore numerico a ogni passo dell'allineamento, in modo da quantificare la qualità di un allineamento. I punteggi guidano l'algoritmo nella scelta del percorso migliore nella matrice di programmazione dinamica

***Match Score* (+1)**

È il punteggio assegnato quando un carattere di una sequenza corrisponde esattamente al carattere equivalente nell'altra sequenza

***Mismatch Penalty* (-1)**

È la penalità applicata quando un carattere di una sequenza non corrisponde al carattere equivalente nell'altra sequenza

***Gap Penalty* (-2)**

È la penalità applicata quando si introduce un gap (spazio vuoto) in una sequenza per allinearla con l'altra

Motivo: l'introduzione di un gap rappresenta un'inserzione/delezione (indel) nella sequenza, che biologicamente può indicare una mutazione strutturale o un errore

Needleman-Wunsch: Algoritmo

Prima sequenza: AGT
Seconda sequenza: GTT

Match Score: +1
Mismatch Penalty: -1
Gap Penalty: -2

NW Matrix	-	A	G	T
-				
G				
T				
T				

Needleman-Wunsch: Algoritmo

3. Inizializzazione

Si inizializzano la riga 0 e la colonna 0 della matrice con penalità cumulative, poiché l'allineamento iniziale implica solo gap

$M[i][0]=i \cdot Gap\_Penalty$

$M[0][j]=j \cdot Gap\_Penalty$

Needleman-Wunsch: Algoritmo

Prima sequenza: AGT
Seconda sequenza: GTT

Match Score: +1
Mismatch Penalty: -1
Gap Penalty: -2

		0	1	2	3
	NW Matrix	-	G	T	T
0	-	0	-2	-4	-6
1	A	-2			
2	G	-4			
3	T	-6			

Needleman-Wunsch: Algoritmo

4. Popolamento della matrice
Ogni cella M[i][j] è calcolata scegliendo il massimo tra:

- M[i−1][j−1]+Match_Score/Mismatch_Penalty: *corrispondenza o non corrispondenza (sostituzione) della posizione corrente cioè* M[i,j]
- M[i−1][j]+Gap_Penalty
- M[i][j−1]+Gap_Penalty

Needleman-Wunsch: Algoritmo

Prima sequenza: AGT
Seconda sequenza: GTT

Match Score: +1
Mismatch Penalty: -1
Gap Penalty: -2

		0	1	2	3
	NW Matrix	-	G	T	T
0	-	0	-2	-4	-6
1	A	-2	**-1**		
2	G	-4			
3	T	-6			

M[1,1]= max(M[0,0]-1 (*miss match A/G*),M[0,1]-2, M[1,0]-2)=max(-1,-4,-4)=-1

Needleman-Wunsch: Algoritmo

Prima sequenza: AGT
Seconda sequenza: GTT

Match Score: +1
Mismatch Penalty: -1
Gap Penalty: -2

		0	1	2	3
	NW Matrix	-	G	T	T
0	-	0	-2	-4	-6
1	A	-2	-1	**-3**	
2	G	-4			
3	T	-6			

M[1,2]= max(M[0,1]-1,M[0,2]-2, M[1,1]-2)=max(-3,-6,-3)=-3

Needleman-Wunsch: Algoritmo

Prima sequenza: AGT
Seconda sequenza: GTT

Match Score: +1
Mismatch Penalty: -1
Gap Penalty: -2

		0	1	2	3
	NW Matrix	-	G	T	T
0	-	0	-2	-4	-6
1	A	-2	-1	-3	**-5**
2	G	-4			
3	T	-6			

M[1,3]= max(M[0,2]-1,M[0,3]-2, M[1,2]-2)=max(-5,-8,-5)=-5

Needleman-Wunsch: Algoritmo

Prima sequenza: AGT
Seconda sequenza: GTT

Match Score: +1
Mismatch Penalty: -1
Gap Penalty: -2

		0	1	2	3
	NW Matrix	-	G	T	T
0	-	0	-2	-4	-6
1	A	-2	-1	-3	-5
2	G	-4	**-1**		
3	T	-6			

M[2,1]= max(M[1,0]+1,M[1,1]-2, M[2,0]-2)=max(-1,-3,-6)=-1

Needleman-Wunsch: Algoritmo

Prima sequenza: AGT
Seconda sequenza: GTT

Match Score: +1
Mismatch Penalty: -1
Gap Penalty: -2

		0	1	2	3
	NW Matrix	-	G	T	T
0	-	0	-2	-4	-6
1	A	-2	-1	-3	-5
2	G	-4	-1	**-2**	
3	T	-6			

M[2,2]= max(M[1,1]-1,M[1,2]-2, M[2,1]-2)=max(-2,-5,-6)=-2

Needleman-Wunsch: Algoritmo

Prima sequenza: AGT
Seconda sequenza: GTT

Match Score: +1
Mismatch Penalty: -1
Gap Penalty: -2

		0	1	2	3
	NW Matrix	-	G	T	T
0	-	0	-2	-4	-6
1	A	-2	-1	-3	-5
2	G	-4	-1	-2	**-4**
3	T	-6			

M[2,3]= max(M[1,2]-1,M[1,3]-2, M[2,2]-2)=max(-4,-7,-4)=-4

Needleman-Wunsch: Algoritmo

Prima sequenza: AGT
Seconda sequenza: GTT

Match Score: +1
Mismatch Penalty: -1
Gap Penalty: -2

		0	1	2	3
	NW Matrix	-	G	T	T
0	-	0	-2	-4	-6
1	A	-2	-1	-3	-5
2	G	-4	-1	-2	-4
3	T	-6	**-3**		

M[3,1]= max(M[2,0]-1,M[2,1]-2, M[3,0]-2)=max(-5,-3,-8)=-3

Needleman-Wunsch: Algoritmo

Prima sequenza: AGT
Seconda sequenza: GTT

Match Score: +1
Mismatch Penalty: -1
Gap Penalty: -2

		0	1	2	3
	NW Matrix	-	G	T	T
0	-	0	-2	-4	-6
1	A	-2	-1	-3	-5
2	G	-4	-1	-2	-4
3	T	-6	-3	**0**	

M[3,2]= max(M[2,1]+1,M[2,2]-2, M[3,1]-2)=max(0,-4,-5)=0

Needleman-Wunsch: Algoritmo

Prima sequenza: AGT
Seconda sequenza: GTT

Match Score: +1
Mismatch Penalty: -1
Gap Penalty: -2

		0	1	2	3
	NW Matrix	-	G	T	T
0	-	0	-2	-4	-6
1	A	-2	-1	-3	-5
2	G	-4	-1	-2	-4
3	T	-6	-3	0	**-1**

M[3,3]= max(M[2,2]+1,M[2,3]-2, M[3,2]-2)=max(-1,-6,-2)=-1

Needleman-Wunsch: Algoritmo

5. Ricostruzione dell'allineamento

Partendo dalla cella in basso a destra della matrice (M[n+1][m+1]), si risale indietro scegliendo il percorso che ha portato al punteggio massimo:

- Diagonale: corrispondenza o sostituzione.
- Sopra: gap nella seconda Sequenza
- Sinistra: gap nella prima Sequenza

Needleman-Wunsch: Algoritmo

Prima sequenza: AGT
Seconda sequenza: GTT

Match Score: +1
Mismatch Penalty: -1
Gap Penalty: -2

		0	1	2	3
	NW Matrix	-	G	T	T
0	-	0	-2	-4	-6
1	A	-2	-1	-3	-5
2	G	-4	-1	-2	-4
3	T	-6	-3	0	-1

Passo 1:

Cella M[3,3]=−1
Confronto T (Sequ.1) con T (Sequ.2)

Deriva da:
Diagonale (M[2,2]+match score=−2+1=−1)

Allineamento parziale:
Sequenza 1: T
•**Sequenza 2**: T

Needleman-Wunsch: Algoritmo

Prima sequenza: AGT
Seconda sequenza: GTT

Match Score: +1
Mismatch Penalty: -1
Gap Penalty: -2

		0	1	2	3
	NW Matrix	-	G	T	T
0	-	0	-2	-4	-6
1	A	-2	-1	-3	-5
2	G	-4	-1	-2	-4
3	T	-6	-3	0	-1

Passo 2:

Cella M[2,2]=−2
Confronto G (Sequ.1) con T (Sequ.2)

Deriva da:
Diagonale (M[1,1]+match score=−1-1=−2)

Allineamento parziale:
Sequenza 1: GT
•**Sequenza 2**: TT

Needleman-Wunsch: Algoritmo

Prima sequenza: AGT
Seconda sequenza: GTT

Match Score: +1
Mismatch Penalty: -1
Gap Penalty: -2

		0	1	2	3
	NW Matrix	-	G	T	T
0	-	0	-2	-4	-6
1	A	-2	-1	-3	-5
2	G	-4	-1	-2	-4
3	T	-6	-3	0	-1

Passo 3:

Cella M[1,1]=−1
Confronto A (Sequ.1) con G (Sequ.2)

Deriva da:
Diagonale (M[0,0]+match score=0-1=−1)

Miglior allineamento:
Sequenza 1: AGT
Sequenza 2: GTT

Needleman-Wunsch: Ricostruzione (esempio)

Prima sequenza: AGTCCCAT
Seconda sequenza: AGATTCCAT

Match Score: +1
Mismatch Penalty: -1
Gap Penalty: -2

	-	A	G	A	T	T	C	C	A	T
-	0	-2	-4	-6	-8	-10	-12	-14	-16	-18
A	-2	1	-1	-3	-5	-7	-9	-11	-13	-15
G	-4	-1	2	0	-2	-4	-6	-8	-10	-12
T	-6	-3	0	1	1	-1	-3	-5	-7	-9
C	-8	-5	-2	-1	0	0	0	-2	-4	-6
C	-10	-7	-4	-3	-2	-1	1	1	-1	-3
C	-12	-9	-6	-5	-4	-3	0	2	0	-2
A	-14	-11	-8	-5	-6	-5	-2	0	3	-1
T	-16	-13	-10	-7	-4	-5	-4	-2	1	4

Miglior allineamento
Sequenza 1: AG-TCCCAT
Sequenza 2: AGATTCCAT

Needleman-Wunsch: Complessità

L'algoritmo di **Needleman-Wunsch per costruire la matrice richiede una complessità computazionale temporale $O(m \cdot n)$**

Si costruisce una matrice di dimensione m×n, dove m e n sono le lunghezze delle due sequenze da confrontare.
Ogni cella della matrice è riempita calcolando un valore basato sui valori delle celle vicine (sopra, sinistra e diagonale) secondo una funzione di punteggio.
Operazioni richieste:
Calcolo del valore di ciascuna cella: O(1).
Numero totale di celle: $m \cdot n$

Needleman-Wunsch: Complessità

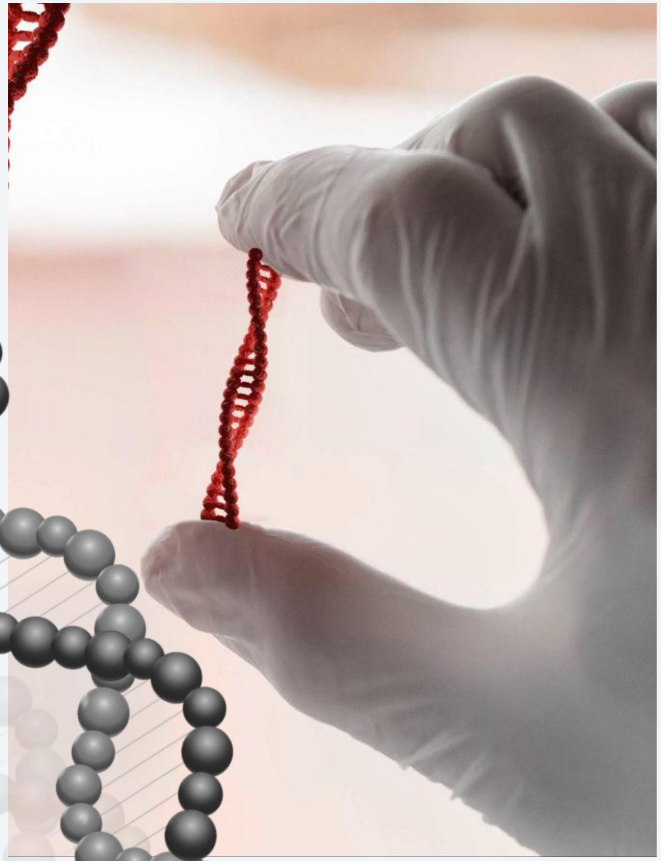
L'algoritmo di **Needleman-Wunsch per recuperare l'allineamento dopo la costruzione della matrice è $O(m+n)$**

Dopo aver riempito la matrice, si risale dalla cella in basso a destra (che contiene il punteggio ottimale) alla cella in alto a sinistra.
A ogni passo, si decide se si proviene da una cella diagonale (allineamento), da una cella superiore (gap nella seconda sequenza), o da una cella a sinistra (gap nella prima sequenza).
Operazioni richieste:
Ogni passo del backtracking richiede $O(1)$.
Il numero massimo di passi è pari a m+n, perché si percorrono al massimo tutte le righe (m) e colonne (n) della matrice.

Needleman-Wunsch: Complessità

La complessità complessiva dell'algoritmo Needleman-Wunsch combina le due fasi:
Costruzione della matrice: O(m·n).
Recupero dell'allineamento: O(m+n).
Poiché il termine O(m·n) domina O(m+n) la complessità totale dell'algoritmo è: O(m·n)

Questo lo rende praticabile per sequenze relativamente brevi, ma inefficiente per sequenze molto lunghe (come interi genomi), dove si preferiscono algoritmi approssimati o più efficienti, come BLAST o algoritmi di programmazione dinamica ottimizzata (es. Hirschberg).



03

SPACE SEED

SPACED-SEED

Lo **Spaced-Seed** è una tecnica utilizzata per la ricerca di sequenze in bioinformatica, spesso impiegata per il confronto di sequenze genomiche o per la ricerca di similarità tra sequenze.

L'idea alla base di questo algoritmo è quella di creare "semi" (*seeds*) disgiunti, che consistono in una sottosequenza di simboli a intervalli (spazi) regolari. Questi semi sono poi usati per indicizzare e velocizzare la ricerca.

SPACED-SEED: Esempio

Sequenze:

Sequenza 1: ACCTTGGC
Sequenza 2: ATCTCAGC

SPACED-SEED: Funzionamento

1 Creazione Spaced-seeds: un spaced-seed è una sottosequenza che è costruita selezionando simboli di una sequenza originale a intervalli regolari.
Un seed è costituito dalla selezione di simboli da una sequenza di nucleotidi, dove sono scelti solo alcuni simboli a precise posizioni e gli altri sono ignorati.

Ad esempio, dato il **pattern** "1101", si selezionano i simboli corrispondenti ai posti indicati da "1". Questo vuol dire che i seleziona il primo, il terzo e il quarto simbolo di una sequenza

SPACED-SEED: Esempio

Pattern di spaced-seed: **101011000**

Questo significa che si prende il 1°, 3°, 5° e 6° simbolo dalla sequenza

Sequenza 1: ACATTGGCA cioè AATG

Sequenza 2: ATCTCAGCT cioè ACCA

SPACED-SEED: Funzionamento

2. Indirizzamento della ricerca: il metodo prevede che i spaced-seeds siano usati per indicizzare una sequenza, permettendo di risparmiare tempo di calcolo durante la ricerca di somiglianze tra sequenze

SPACED-SEED: Esempio

Pattern di spaced-seed: **101011000**

Sequenza 1 read: ACATTGGCA AATG

Sequenza 2 read: ATCTCAGCT ACCA

Sequenza 1 genoma: ATATTCCAC AATC

Sequenza 2 genoma: CTACTCAAT CATC

Sequenza 3 genoma: TAGGGTAGC TGGT

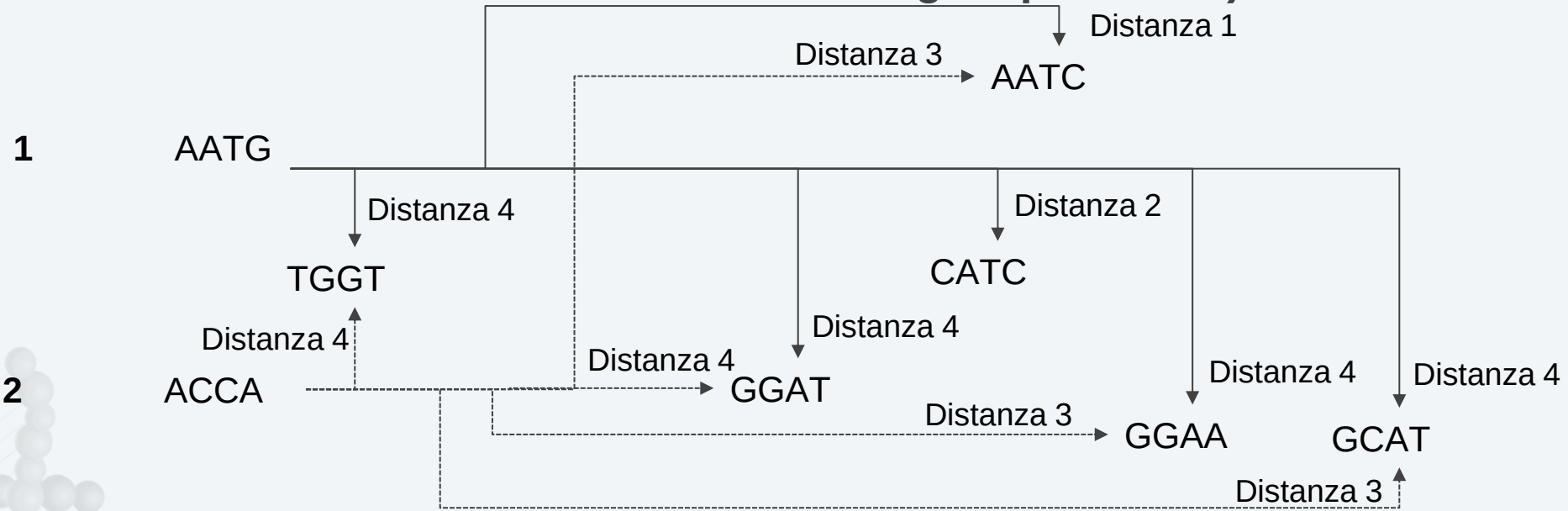
Sequenza 4 genoma: GACTATTGG GCAT

Sequenza 5 genoma: GAGAAACCC GGAA

Sequenza 6 genoma: GTGTATCGC GGAT

SPACED-SEED: Esempio

Indicizzazione e migliore corrispondenza (minore distanza: cioè numero di caratteri differenti ad ugual posizione)



SPACED-SEED: Esempio

Seme 1: **AATG**
Migliore corrispondenza: **AATC**

Si può osservare che i semi non corrispondono esattamente, ma se li confrontiamo con un certo margine di tolleranza (per esempio, permettendo una sostituzioni tra i semi), si può dire che sono abbastanza simili. Nonostante ci sia una discrepanza (G vs. C nel quarto simbolo), l'algoritmo di spaced-seed è ancora utile, perché la ricerca si concentra su sezioni ridotte e può essere estesa solo su queste aree dove i semi corrispondono. Questo migliora l'efficienza rispetto a confrontare le sequenze intere.

SPACED-SEED: Esempio

Seme 2: **ACCA**
Migliore corrispondenza: **GCAT**

In questo caso, le sequenze contengono differenze sostanziali che non sono catturate dal pattern del spaced-seed.
Il spaced-seed indexing non ha funzionato bene, perché le grandi differenze tra i semi estratti non riflettono una somiglianza significativa tra le sequenze. In presenza di mutazioni o grandi differenze, il metodo diventa meno efficace.
Inoltre, il metodo di spaced-seed non tiene conto di altre possibili corrispondenze al di fuori dei semi selezionati, quindi la ricerca potrebbe fallire nel rilevare somiglianze più sottili

SEED-and-EXTEND

L’**algoritmo "Seed-and-Extend" è una strategia di allineamento che cerca di ottimizzare la ricerca di corrispondenze tra una sequenza di riferimento (genoma) e una read**

Il metodo *Seed-and-Extend* utilizza semi (*seeds*) e indici per migliorare l'efficienza della ricerca di corrispondenze tra sequenze
Inizia dividendo sia la sequenza di riferimento (genoma) sia la read in "semi" (seeds), che sono piccole sottosequenze
Successivamente, questi semi sono utilizzati per cercare rapidamente corrispondenze in un indice

SEED-and-EXTEND: Algoritmo di semina

L'algoritmo funziona in due fasi:

1.Fase di semina (Seed):

La sequenza di riferimento (genoma) e le read sono suddivise in piccoli sottostringhe, chiamati **seeds**.

Un seed è di solito una sottosequenza di dimensione fissa (ad esempio, 15 nucleotidi) ed è scelto in modo da essere facilmente identificabile tramite un indice

2.Fase di estensione (Extend):

Dopo aver trovato i semi corrispondenti tra la read e il genoma, l'algoritmo estende la corrispondenza per cercare una corrispondenza più lunga

L'estensione continua fino a quando la corrispondenza non termina o non è raggiunta una certa soglia di similarità

SEED-and-EXTEND: Algoritmo di ricerca

La ricerca avviene sfruttando una **tabella hash**

Una **tabella hash** è una struttura dati progettata per svolgere ricerche in modo veloce

Utilizzando un vettore come questo:
my_array = ['Pete', 'Jones', 'Lisa', 'Bob', 'Siri']
Per trovare "Bob" è necessario confrontare ogni nome, elemento per elemento, fino a trovare ‘Bob’

Per rendere l'interazione con l'elenco dei nomi molto veloce, si può usare una tabella hash.

SEED-and-EXTEND: Tabella hash

Una **tabella hash** è una struttura dati progettata per svolgere ricerche in modo veloce

Utilizzando un vettore come questo:
my_array = ['Pete', 'Jones', 'Lisa', 'Bob', 'Siri']
Per trovare ‘Bob’ è necessario confrontare ogni nome, elemento per elemento, fino a trovare ‘Bob’

Per rendere l'interazione con l'elenco dei nomi veloce, si può usare una tabella hash

SEED-and-EXTEND: Tabella hash

Per semplificare, si supponga che ci siano al massimo 10 nomi nell'elenco, quindi la tabella hash deve avere una dimensione fissa di 10 elementi (cioè i ***bucket***)

my_hash_set = [None,None,None,None,None,None,None,None,None,None]

SEED-and-EXTEND: Tabella hash

Si si vuole memorizzare un nome direttamente nel posto giusto all'interno della tabella hash si deve sfruttare una **funzione hash**, cioè una funzione matematica/informatica che restituisce un valore numerico

Una **funzione hash** può essere realizzata in molti modi, dipende dal programmatore. Un modo comune consiste nel trovare un modo per convertire il valore in un numero che sia uguale a uno dei numeri di indice della tabella hash, nel caso di esempio un numero compreso tra 0 e 9.
Un funzione hash facile per questo caso è quella che somma il valore ASCII di ciascun carattere del nome e svolge il modulo 10 del valore risultante per ottenere i numeri di indice 0-9.

SEED-and-EXTEND: Tabella hash

```
def hash_function(value):
    sum_of_chars = 0
    for char in value:
        sum_of_chars += ord(char)

    return sum_of_chars % 10
```

Il carattere "B" ha il valore ASCII 66, "o" ha 111 e "b" ha 98. Sommandoli insieme si ottiene 275. Il modulo 10 di 275 è 5, quindi "Bob" è memorizzato come elemento di matrice all'indice 5.

Il numero restituito dalla funzione hash è chiamato **codice hash**

SEED-and-EXTEND: Tabella hash

La tabella hash ora assume è così composta:
my_hash_set = [None,None,None,None,None,'Bob',None,None,None,None]

Il vettore si ‘popola’ di bucket applicando più volte la funzione hash ai diversi nomi:
my_hash_set = [None,'Jones',None,'Lisa',None,'Bob',None,'Siri','Pete',None]

Per scoprire se ‘Pete’ è memorizzato nell'array, si deve calcolare la funzione hash di "Pete"
Il codice hash è 8
Andando all'indice 8 della tabella hash si trova ‘Pete’ (senza controllare altri elementi)

SEED-and-EXTEND: Tabella hash

Aggiungendo "Stuart" alla tabella hash si ottiene il codice hash 3 creando una **collisione** perché all'indice 3 è già memorizzato il nome 'Lisa'

Per risolvere una collisione, si può fare spazio a più elementi nello stesso bucket (concatenamento) che deve essere organizzato come un vettore o una lista

Implementando i bucket come vettori, 'Stuart' può essere memorizzato all'indice 3 ottenendo la seguente tabella hash:

my_hash_set = [[None],['Jones'],[None],['Lisa', 'Stuart'], [None], ['Bob'], [None], ['Siri'], ['Pete'], [None]]

La ricerca di 'Stuart' (o di 'Lisa') richiede l'accesso al bucket 3 e poi lo scorrimento del vettore associato

SEED-and-EXTEND: Esempio

Si supponga di avere il genoma di riferimento e una read da confrontare.
Si utilizza un esempio semplice con una lunghezza di seed di 3 nucleotidi per semplificare il procedimento.

Sequenza del genoma di riferimento: **ACCTTGGC**
Sequenza della read: **CTTGGA**

SEED-and-EXTEND: Esempio

Sequenza del genoma di riferimento: **ACCTTGGC**
Sequenza della read: **CTTGGA**

Step 1: Creazione dei semi (seeds)
Suddividere entrambe le sequenze in semi di 3 nucleotidi

I semi di lunghezza 3 (k-mers) del genoma sono:
ACC
CCT
CTT
TTG
TGG
GGC

I semi di lunghezza 3 (k-mers) della read sono:
CTT
TTG
TGG
GGA

SEED-and-EXTEND: Esempio

Step 2: Creazione dell'indice

Si costruisce un indice che memorizza la posizione di ogni seed nel genoma di riferimento.

Questo indice è fondamentalmente una *tabella hash* che mappa ogni seed alla sua posizione nel genoma

Indice genoma:

1	ACC
2	CCT
3	CTT
4	TTG
5	TGG
6	GGC

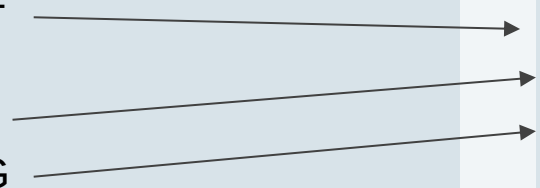
SEED-and-EXTEND: Esempio

Step 3: Ricerca dei semi corrispondenti

Ora si cercano i semi della read nell'indice del genoma: si confrontano i semi della read con quelli dell'indice.

Indice genoma:

1	ACC
2	CCT
3	CTT
4	TTG
5	TGG
6	GGC



I semi di lunghezza 3 (k-mers) della read sono:

CTT
TTG
TGG
GGA

SEED-and-EXTEND Variante

Step 1: Creazione seed

Si stabilisce una lunghezza e si spezza il genoma e le reads in seed di tale misura (nell’esempio 4 basi) creando sottostringhe

Genoma: ACCCTATAAACATATATCTGGGCAGCGACTAG

ACCC TATA AACA TATA TCTG GGCA GCGA CTAG

Reads: ACCCTATAAACAGCGA

ACCC TATA AACA GCGA

SEED-and-EXTEND Variante

Step 2: Creazione dizionario reads

Si crea un dizionario con le sottostringhe per le reads in base alle loro posizioni

DIZIONARIO				
	ACCC	TATA	AACA	GCGA
1				
2				
3				
4				
5				
6				

SEED-and-EXTEND Variante

Step 3: Suddivisione genoma

Si suddivide il genoma dandogli una struttura analoga a quella del dizionario.

SNP e INDEL sono tollerati grazie ai caratteri di spaziatura **** tra i seed

ACCC * * * * AACA * * * *
* * * * TATA * * * * TATA
TCTG * * * * GCGA * * * *
* * * * GGCA * * * * CTAG

ACCC TATA * * * * * * * *
* * * * * * * * AACA TATA
TATA TCTG * * * * * * * *
* * * * * * * * GGCA GCGA

segue…

SEED-and-EXTEND Variante

Step 4: Matching

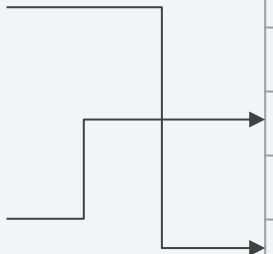
Si cercano le parti in comune e le si indicizzano

$O(n)$

ACCC * * * * AACA * * * *
* * * * TATA * * * * TATA
TCTG * * * * GCGA * * * *
* * * * GGCA * * * * CTAG

ACCC TATA * * * * * * * *
* * * * * * * * AACA TATA
TATA TCTG * * * * * * * *
* * * * * * * * GGCA GCGA

segue…



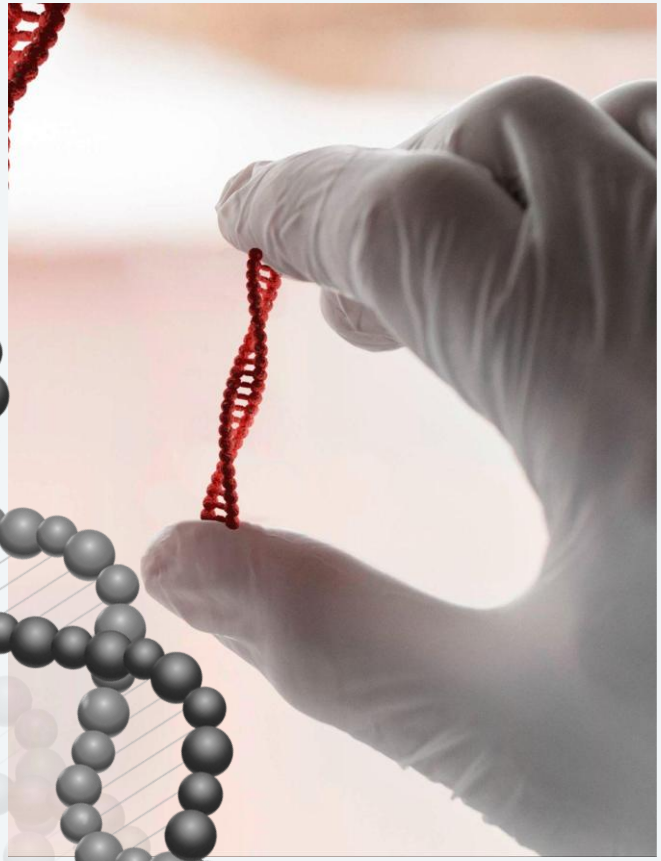
DIZIONARIO				
	ACCC	TATA	AACA	GCGA
1	■	■		
2			■	■
3		■		■
4	■		■	
5		■	■	
6	■			■

SEED-and-EXTEND: Svantaggi

Tollerante a polimorfismi che colpiscono al massimo due seed

La creazione dell'indice del genoma di riferimento è un passaggio dispendioso in termini di tempo e risorse (circa 50GB di memoria per indicizzare il genoma umano)

Velocità dipendente dalle risorse computazionali a disposizione



04

TRASFORMATA BURROWS-WHEELER

BWT: Introduzione

La **Trasformata di Burrows-Wheeler** (BWT) è un algoritmo che riorganizza una stringa in modo tale che le ripetizioni di caratteri simili siano raggruppate, semplificando la compressione e rendendo più efficienti le operazioni di ricerca.

In bioinformatica, è ampiamente utilizzata per rappresentare sequenze di DNA, ad esempio nei software di allineamento delle sequenze.

BWT: Algoritmo

Aggiunta di un terminatore alla stringa di nucleotidi
Un carattere speciale (es. $) che non compare mai nella stringa è aggiunto alla fine per indicarne la fine

↓

Generazione di tutte le rotazioni della stringa
Creare tutte le possibili rotazioni circolari della stringa

↓

Ordinamento delle rotazioni in ordine lessicografico
Le rotazioni sono disposte in ordine alfabetico

↓

Estrazione dell'ultima colonna
Prendere l'ultimo carattere di ogni rotazione ordinata per ottenere la trasformata BWT.

Il risultato della BWT è una stringa in cui i caratteri identici sono spesso raggruppati, rendendo più efficiente la compressione tramite algoritmi come Run-Length Encoding (RLE) o per essere meglio indicizzati

BWT: Esempio

Aggiunta di un terminatore alla stringa di nucleotidi
Un carattere speciale (es. $) che non compare mai nella stringa è aggiunto alla fine per indicarne la fine

Data la sequenza di DNA: **ACGAAG**

Si aggiunge il terminatore $:

La sequenza diventa: **ACGAAG$**

BWT: Algoritmo

Generazione di tutte le rotazioni della stringa
Creare tutte le possibili rotazioni circolari della stringa

Rotazione circolare

ACGAAG$
CGAAG$A
GAAG$AC
AAG$ACG
AG$ACGA
G$ACGAA
$ACGAAG

ACGAAG$ ACGAAG$← CGAAG$A

Si fa uno shift logico

BWT: Algoritmo

Ordinamento delle rotazioni in ordine lessicografico
Le rotazioni sono disposte in ordine alfabetico

Ordinamento lessicografico

$ACGAAG
AAG$ACG
ACGAAG$
AG$ACGA
CGAAG$A
G$ACGAA
GAAG$AC

BWT: Algoritmo

Estrazione dell'ultima colonna
Prendere l'ultimo carattere di ogni rotazione ordinata, per ottenere la trasformata BWT.

Estrazione dell'ultima colonna

$ACGAAG
AAG$ACG
ACGAAG$
AG$ACGA
CGAAG$A
G$ACGAA
GAAG$AC

Output: GG$AAAC

O(n) rotazione
O(nlog(n)) ordinamento

$O(n)+O(n\log(n))\rightarrow O(n\log(n))$

BWT: Caratteristiche del metodo

I nucleotidi tendono a essere raggruppati in ordine alfabetico, facilitando la compressione.

Output: GG$AAAC

Possibile compressione RLE: 2G1$3A1C

BWT: Caratteristiche del metodo

Si dimostra che per reads lunghe la compressione cresce

Sequenza:

AAAAAAAAAAAAAAAAATGGGGGTCTCTCTCTCTGGGGGGTCTCTCT
CTCTGGGGGTAAAAAAAAAAAAAAAAAAAAAAAAAAAA$ 88 byte

BWT:

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAATTTTTTTTTTTTTTTTTTTTT
GGGGGGGGGGGGGGGGGGGGGCCCCCCCCCCCCCCCC$

Possible compressione RLE:

A30T21G21C16 8 byte

Tasso di compressione= [100(1-8/88)]%=90.91%

BWT Inversa

La trasformata di Burrows-Wheeler gode della reversibilità: dalla stringa BWT è possibile ricostruire l'originale utilizzando l'algoritmo inverso (se non fosse garantita questa proprietà basterebbe un semplice ordinamento dei caratteri)

L’algoritmo inverso opera prendendo la stringa finale di BWT
Si dispone sulla prima colonna di una ipotetica matrice e si ottiene la seconda colonna ordinando i caratteri della prima colonna. Continuando in questo modo si ricostruisce l'intera lista. Alla fine ottenuto un insieme di stringhe lunghe quanto quella derivante da BWT si seleziona l’elemento con il carattere '$' alla fine (è il testo originale)

BWT Inversa

Inizio

Scrivere l’ultima colonna BWT

BWT (ultima colonna)	
G	
G	
$	
A	
A	
A	
C	

BWT Inversa

I Iterazione

Ordinare in modo lessicografico e concatenare la parte ordinata con quella originale

BWT (ultima colonna)	Ordinamento	Unione
G	$	G$
G	A	GA
$	A	$A
A	A	AA
A	C	AC
A	G	AG
C	G	CG

BWT Inversa

II Iterazione

Ordinare in modo lessicografico e concatenare la parte ordinata con quella originale

BWT (ultima colonna)	Nuova colonna	Ordinamento	Unione
G	G$	$A	G$A
G	GA	AA	GAA
$	$A	AC	$AC
A	AA	AG	AAG
A	AC	CG	ACG
A	AG	G$	AG$
C	CG	GA	CGA

BWT Inversa

III Iterazione

Ordinare in modo lessicografico e concatenare la parte ordinata con quella originale

BWT (ultima colonna)	Nuova colonna	Ordinamento	Unione
G	G$A	$AC	G$AC
G	GAA	AAG	GAAG
$	$AC	ACG	$ACG
A	AAG	AG$	AAG$
A	ACG	CGA	ACGA
A	AG$	G$A	AG$A
C	CGA	GAA	CGAA

BWT Inversa

IV Iterazione

Ordinare in modo lessicografico e concatenare la parte ordinata con quella originale

BWT (ultima colonna)	Nuova colonna	Ordinamento	Unione
G	G$AC	$ACG	G$ACG
G	GAAG	AAG$	GAAG$
$	$ACG	ACGA	$ACGA
A	AAG$	AG$A	AAG$A
A	ACGA	CGAA	ACGAA
A	AG$A	G$AC	AG$AC
C	CGAA	GAAG	CGAAG

BWT Inversa

V Iterazione

Ordinare in modo lessicografico e concatenare la parte ordinata con quella originale

BWT (ultima colonna)	Nuova colonna	Ordinamento	Unione
G	G$ACG	$ACGA	G$ACGA
G	GAAG$	AAG$A	GAAG$A
$	$ACGA	ACGAA	$ACGAA
A	AAG$A	AG$AC	AAG$AC
A	ACGAA	CGAAG	ACGAAG
A	AG$AC	G$ACG	AG$ACG
C	CGAAG	GAAG$	CGAAG$

BWT Inversa

VI Iterazione (ultima)

Ordinare in modo lessicografico e concatenare la parte ordinata con quella originale. Il risultato è quello con il simbolo $ alla fine

BWT (ultima colonna)	Nuova colonna	Ordinamento	Unione
G	G$ACGA	$ACGAA	G$ACGAA
G	GAAG$A	AAG$AC	GAAG$AC
$	$ACGAA	ACGAAG	$ACGAAG
A	AAG$AC	AG$ACG	AAG$ACG
A	ACGAAG	CGAAG$	ACGAAG$
A	AG$ACG	G$ACGA	AG$ACGA
C	CGAAG$	GAAG$A	CGAAG$A

Stringa originale
ACGAAG

BWT: Uso

La Burrows-Wheeler Transform (BWT), combinata con una struttura dati chiamata FM-Index, fornisce una tecnica avanzata per effettuare ricerche rapide ed efficienti di sotto-sequenze (pattern) all'interno di grandi sequenze genomiche o testi.

La struttura dati FM-Index pende il nome dai due informatici che l’hanno definita: Paolo Ferragina e Giovanni Manzin

Paolo Ferragina and Giovanni Manzini (2005). *Indexing Compressed Text*. Journal of the ACM, 52, 4 (Jul. 2005). p. 553

FM-Index: Generalità

L'FM-Index è una struttura dati che:

- Usa la BWT per comprimere il testo di input.
- Supporta la ricerca di pattern senza dover decomprimere il testo

L'FM-Index combina la BWT con altre strutture ausiliarie:

- **Occ-Table** (tabella delle occorrenze): Memorizza quante volte ogni carattere appare nella BWT fino a una certa posizione
- **C-Table** (array C): Indica la posizione iniziale di ogni carattere nella colonna iniziale dell’array ordinato (della matrice di rotazioni della BWT)

FM-Index: Generalità

La ricerca con FM-Index si basa sul concetto di **backward search**:

- La BWT consente di navigare "indietro" nella matrice delle rotazioni, usando la relazione tra la prima e l'ultima colonna dell'array BWT ordinato (proprietà LF-mapping).
- Invece di scansionare tutto il testo, la ricerca avviene tramite operazioni rapide sulle strutture ausiliarie (Occ e C).

FM-Index: Esempio

Sequenza originale: ACGAAG Sequenza da trovare =CGA
BWT (colonna finale): GG$AAAC

C-Table: posizioni iniziali dei caratteri nella colonna iniziale ordinata dopo BWT, cioè considerando la prima colonna della matrice ordinata derivata dopo BWT segnare dove appare la prima volta il carattere in esame…

A quale posizione per la prima volta si trova il carattere x guardando la prima colonna della matrice?

Carattere	Posizione
$	0
A	1
C	4
G	5

Posizione	Permutazione
0	**$**ACGAAG
1	**A**AG$ACG
2	**A**CGAAG$
3	**A**G$ACGA
4	**C**GAAG$A
5	**G**$ACGAA
6	**G**AAG$AC

FM-Index: Esempio

Occ-Table: (tabella cumulativa di occorrenze per ogni carattere in BWT)
Si analizza quante volte è presente il nucleotide nella sequenza finale BWTa partire dalla posizione 0..

Quanti x ci sono fino alla posizione y guardando l'ultima colonna della BWT?

Posizione	A	C	G	$
0	0	0	1	0
1	0	0	2	0
2	0	0	2	1
3	1	0	2	1
4	2	0	2	1
5	3	0	2	1
6	3	1	2	1

Posizione	
0	$ACGAAG
1	AAG$ACG
2	ACGAAG$
3	AG$ACGA
4	CGAAG$A
5	G$ACGAA
6	GAAG$AC

Cercare **"CGA"** significa partire dall'ultimo carattere **A** e procedere **all'indietro** attraverso i caratteri **G** e **C**.

FM-Index: Esempio (ricerca)

Pattern da trovare =CGA

Trovare **A** (ultimo carattere del pattern)

Usando la C-Table, si vede che le occorrenze di A iniziano alla posizione uno: CTable(A)=1

Per calcolare l'intervallo di ricerca, si usa lo LF-Mapping con intervallo [1,6]:

start=CTable(A)+Occ(A, start-1)=1+Occ(A, 0)=1+0=1

end=CTable (A)+Occ(A, end)−1=1+Occ(A, 6)−1=1+3-1=3

Carattere	Posizione
$	0
A	1
C	4
G	5

Posizione	A	C	G	$
0	0	0	1	0
1	0	0	2	0
2	0	0	2	1
3	1	0	2	1
4	2	0	2	1
5	3	0	2	1
6	3	1	2	1

FM-Index: Esempio (ricerca)

Sequenza da trovare =CGA

Trovare **G** (penultimo carattere del pattern)
Considerare solo le righe dell'intervallo [1, 3] trovate precedentemente e considerare il carattere G: CTable(G)=5

Carattere	Posizione
$	0
A	1
C	4
G	5

Per calcolare il nuovo intervallo, si usa lo LF-Mapping nell'intervallo [1,3]:
start=CTable(G)+Occ(G, start-1)=5+Occ(G, 0)=5+1=6
end=CTable(G)+Occ(G, end)−1=5+Occ(G, 3)−1=5+2−1=6

Posizione	A	C	G	$
0	0	0	1	0
1	0	0	2	0
2	0	0	2	1
3	1	0	2	1
4	2	0	2	1
5	3	0	2	1
6	3	1	2	1

Intervallo per G: [6, 6]

FM-Index: Esempio (ricerca)

Sequenza da trovare =CGA

Trovare **C** (primo carattere del pattern)
Considerare solo la riga corrispondente all'intervallo [6, 6] e analizzare il carattere C: CTable(C)=4

Per calcolare il nuovo intervallo, si usa l'LF-Mapping sull'intervallo [6,6]:
start=C(C)+Occ(C, start - 1)=4+Occ(C, 5)=4+1=5
end=C(C)+Occ(C, end)−1=4+Occ(C, 6)−1=4+1−1=4

Intervallo per C: [4, 4]

Carattere	Posizione
$	0
A	1
C	4
G	5

Posizione	A	C	G	$
0	0	0	1	0
1	0	0	2	0
2	0	0	2	1
3	1	0	2	1
4	2	0	2	1
5	3	0	2	1
6	3	1	2	1

FM-Index: Esempio (ricerca)

L’ intervallo finale [x,y] rappresenta tutte le righe della matrice di rotazioni in cui il pattern CGA è il prefisso

Poiché l'intervallo contiene una sola riga [4,4]*, significa che CGA appare una sola volta nella sequenza originale*

BWT ordinata

0 $ACGAAG
1 AAG$ACG
2 ACGAAG$
3 AG$ACGA
4 CGAAG$A
5 G$ACGAA
6 GAAG$AC

FM-Index: Esempio

Guardando la posizione del marcatore di terminazione si capisce il numero di rotazioni svolte (in questo caso uno).
La posizione del patter individuato pertanto, nella stringa originale, inizia dalla posizione immediatamente successiva

Risultato:
La rotazione CGAAG$A corrisponde ad uno spostamento della sequenza originale ACGAAG$ e ci indica che il pattern CGA inizia alla posizione 2 della stringa originale

Conferma del risultato
Sequenza originale: ACGAAG$ Pattern cercato: CGA
Partendo dalla posizione 2 inclusa, leggiamo 3 caratteri (CGA), verificando che il pattern appare una volta sola

BWT e FM-Index: Vantaggi

Efficienza computazionale: La BWT e l'FM-index permettono di effettuare ricerche di allineamento in modo molto efficiente.
La fusione di questi due algoritmi riduce il tempo di calcolo necessario per trovare corrispondenze tra la read e il genoma di riferimento, il che è particolarmente importante quando si lavora con sequenze genomiche lunghe

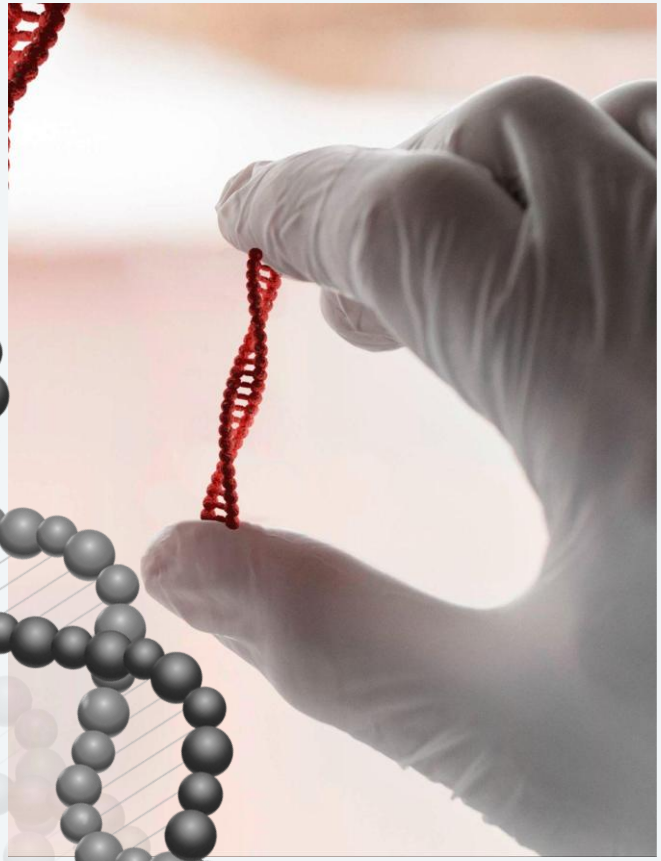
Gestione di mutazioni e indel: L'FM-index è in grado di gestire efficacemente
variazioni genomiche come mutazioni e indels.
Questo lo rende adatto per l'allineamento di sequenze con mutazioni o variazioni rispetto al riferimento, come sequenze di tumori o di popolazioni genetiche diverse.

BWT e FM-Index: Vantaggi

Spazio di memoria ridotto: L'FM-index richiede meno spazio di memoria rispetto ad alcune altre strutture dati utilizzate nell'allineamento genetico. Questo significa che è possibile mantenere l'indice in memoria anche su hardware con risorse limitate

Adattabilità a diverse applicazioni: L'FM-index può essere utilizzato per una vasta gamma di applicazioni nell'analisi genomica, tra cui l'allineamento di sequenze DNA e RNA, la ricerca di varianti genetiche, la determinazione della similarità tra sequenze, la ricerca di regioni funzionali e altro ancora

Software ampiamente disponibili: Esistono diversi software ben sviluppati che sfruttano l'FM-index, come: BWA, Bowtie, Bowtie2 e HISAT2



05

SOFTWARE PER L’ALLINEAMENTO

SOFTWARE ALLINEAMENTO

Programma	Tecnica di Indicizzazione	Tecnica di Allineamento	Specificità
Bowtie2	FM-index (Burrows-Wheeler Transform)	Allineamento completo con tolleranza agli errori	Ottimizzato per reads corte, tollera errori e variazioni
BWA	FM-index (Burrows-Wheeler Transform)	BWA-MEM: Allineamento semi-globale con tolleranza agli errori	Adatto a reads lunghe, gestione di allineamenti secondari
HISAT2	FM-index + Indice Grafo (Graph-based index)	Allineamento tramite indice basato su grafo, gestisce lo splicing	Ottimizzato per RNA-seq, gestisce giunzioni di splicing
Salmon	K-mer index (Pseudo-allineamento)	Pseudo-allineamento senza allineamento esplicito	Quantifica direttamente l'espressione, corregge bias, molto veloce

BOWTIE2

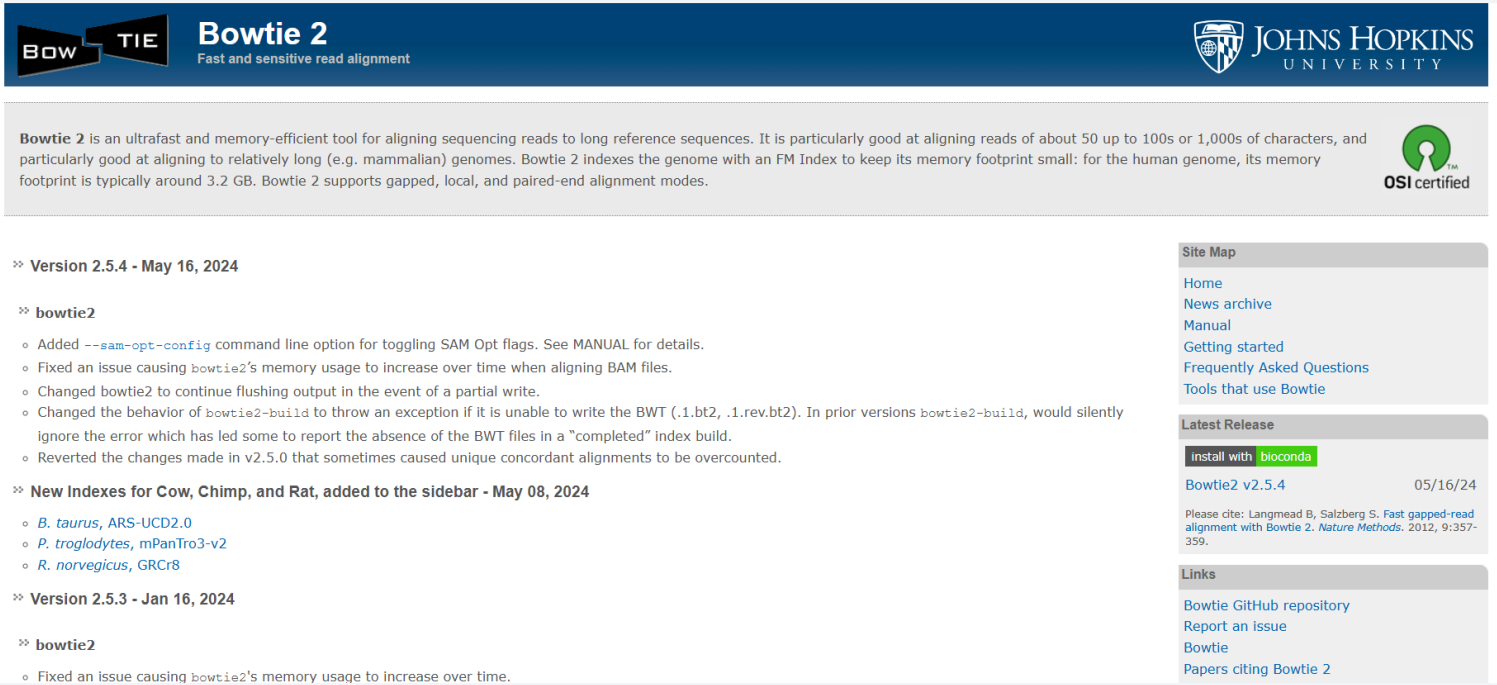
Bowtie2 è un software open-source progettato per l'allineamento veloce e accurato di sequenze di DNA o RNA contro grandi genomi di riferimento, come quelli di umani, animali o batteri. È particolarmente adatto per dati derivanti da tecnologie di sequenziamento ad alta produttività (NGS, Next-Generation Sequencing).

BOWTIE2: Algoritmo

1. **Indice del Genoma**: Prima di poter eseguire un allineamento, Bowtie2 costruisce un indice del genoma di riferimento. L'indice è una struttura dati che accelera la ricerca di corrispondenze tra sequenze di lettura e genoma.
2. **Allineamento**: Durante l'allineamento, Bowtie2 utilizza un algoritmo basato su BWT (Burrows-Wheeler Transform) e FM-index per cercare rapidamente corrispondenze esatte o approssimate. È in grado di gestire letture corte, lunghe e accoppiate (paired-end).
3. **Tolleranza agli Errori**: Supporta mismatch, indel (inserzioni o delezioni) e altre imperfezioni nelle sequenze per adattarsi ai dati reali, che spesso contengono errori.

BOWTIE2: Sito

https://bowtie-bio.sourceforge.net/bowtie2/index.shtml



BOWTIE2: Installazione

È sufficiente il download e la decompressione del file per la propria piattaforma di riferimento (Windows, Linux, Mac)
https://sourceforge.net/projects/bowtie-bio/files/bowtie2/2.5.4/

BOWTIE2: Comando (indice)

Creazione indice del genoma di riferimento

bowtie2-build <genome_fasta> <index_base_name>

Dove

<genome_fasta>: file FASTA contenente il genoma di riferimento.

<index_base_name>: prefisso per i file di indice generati.

BOWTIE2: Comando (allineamento)

Allineamento delle sequenze (single end)

bowtie2 -x <index_base_name> -U <input_reads> -S <output.sam>

Dove:

-x <index_base_name>: prefisso dell'indice del genoma creato con bowtie2-build.

-U <input_reads>: file FASTQ con letture non accoppiate (single-end).

-S <output.sam>: file SAM (Sequence Alignment/Map) per gli allineamenti.

Allineamento delle sequenze (paired end)

bowtie2 -x <index_base_name> -1 <reads_1.fastq> -2 <reads_2.fastq> -S <output.sam>

Dove

-1: file FASTQ per la prima lettura accoppiata.

-2: file FASTQ per la seconda lettura accoppiata.

BOWTIE2: Opzioni

OPZIONE	Descrizione
-p <threads>	numero di core per l'elaborazione parallela (es. -p 4 usa 4 core).
--very-fast	utilizza una modalità di allineamento molto veloce, ma meno sensibile.
--fast	bilancia velocità e sensibilità
--sensitive	(impostazione predefinita) sensibilità moderata.
--very-sensitive	aumenta la sensibilità riducendo la velocità
-k <number>	restituisce il numero massimo di allineamenti validi per ogni reads (default: 1)
--no-unal	non mostra le letture non allineate nell'output
--local	utilizza la modalità di allineamento locale (local alignment)
--end-to-end	(impostazione predefinita) utilizza la modalità di allineamento completo (end-to-end)

BOWTIE2: Esempio

Creazione di un indice:

```
bowtie2-build genome.fa genome_index
```

Allineamento di letture single-end:

```
bowtie2 -x genome_index -U reads.fastq -S output.sam
```

Allineamento di letture paired-end:

```
bowtie2 -x genome_index -1 reads_1.fastq -2 reads_2.fastq -S output.sam -p 8
--very-sensitive
```

BWA

BWA (Burrows-Wheeler Aligner) è un software bioinformatico utilizzato per allineare sequenze di DNA a un genoma di riferimento. È particolarmente utile per analisi di dati di sequenziamento ad alta capacità (NGS, Next-Generation Sequencing). BWA implementa algoritmi efficienti basati sul Burrows-Wheeler Transform e sull'indice FM, che lo rendono veloce e adatto per grandi dataset genomici.

BWA: Caratteristiche

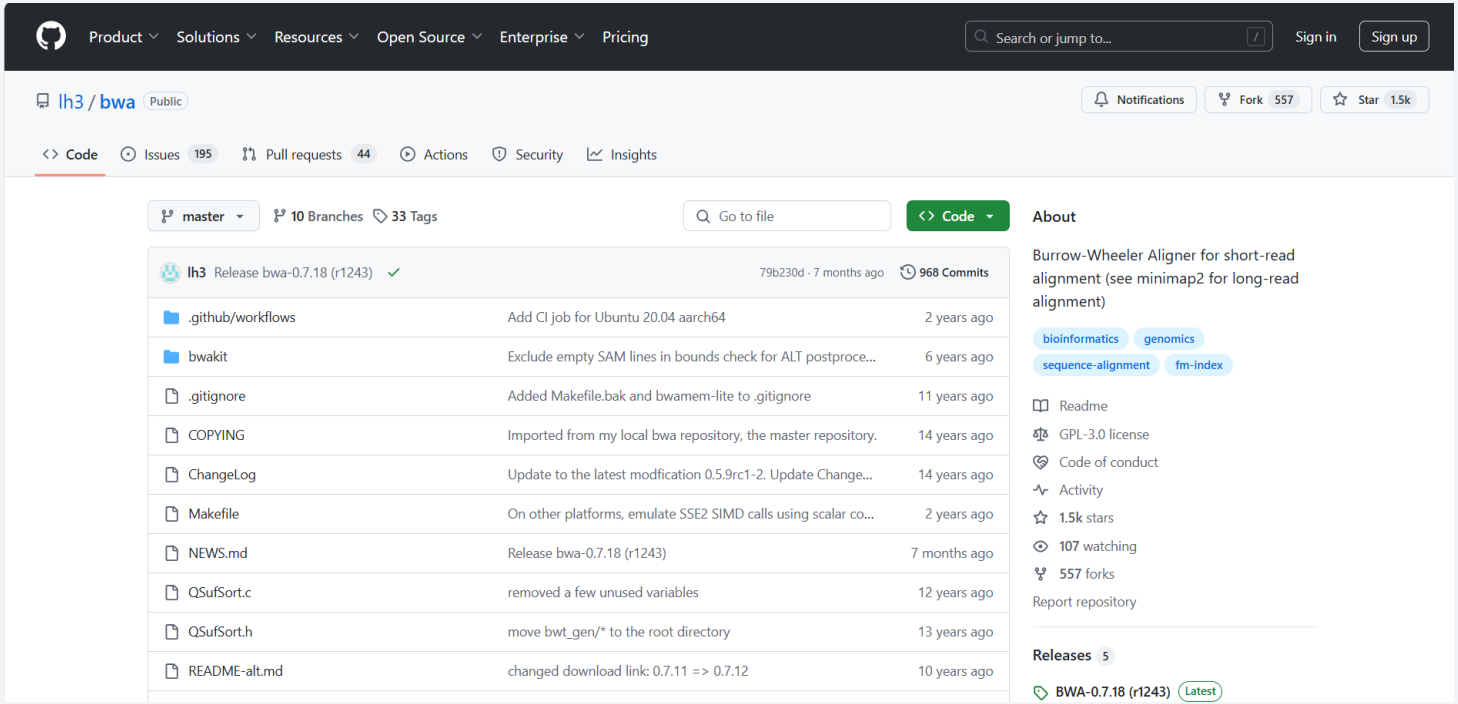
- Allineamenti multipli: BWA supporta l'allineamento di sequenze di lettura multiple e permette di trovare tutte le posizioni di allineamento possibili per ciascuna sequenza di query, fornendo una panoramica completa delle corrispondenze tra le sequenze di lettura e il riferimento.
- Gestione di indel: BWA è in grado di gestire sequenze di lettura con indel (inserimenti/eliminazioni) rispetto al genoma di riferimento. Ciò lo rende adatto per l'allineamento di dati di sequenziamento che possono contenere variazioni genomiche.
- Configurabilità: Gli utenti possono configurare vari parametri di allineamento per personalizzare l'analisi in base alle esigenze specifiche del progetto, inclusi i parametri di mismatch e gap, il tipo di output desiderato e altro ancora.

BWA: Caratteristiche

- Compatibilità con formati standard: BWA supporta numerosi formati di input e output standard dell'industria, come il formato FASTQ per le sequenze di lettura e il formato SAM/BAM per i risultati di allineamento, il che facilita l'integrazione con altri strumenti bioinformatici.
- Ampia documentazione e comunità di supporto: BWA è ben documentato, con un manuale utente completo e una comunità di sviluppatori e utenti attiva che fornisce supporto e risorse online.
- Open source e gratuito: BWA è un software open source distribuito sotto una licenza libera, il che significa che è gratuito da utilizzare e può essere personalizzato o esteso per adattarsi alle specifiche esigenze

BWA: Sito

https://github.com/lh3/bwa



BWA: Installazione

È sufficiente eseguire il comando su Linux:

```
git clone https://github.com/lh3/bwa.git
cd bwa
make
```

BWA: Comando (indice)

Creare un indice del genoma di riferimento
Prima di eseguire l'allineamento, è necessario creare un indice del genoma:

```
bwa index reference_genome.fasta
```

Questo comando genera diversi file indicizzati nella stessa directory del genoma di riferimento.

BWA: Comando (allineamento)

Eseguire l'allineamento

Per allineare sequenze in formato FASTQ (singole o paired-end) al genoma di riferimento:

```
bwa mem reference_genome.fasta reads_R1.fastq reads_R2.fastq > output.sam
```

Dove:

reference_genome.fasta: file del genoma di riferimento.

reads_R1.fastq e reads_R2.fastq: file contenenti le letture di sequenziamento.

output.sam: file SAM generato come output, contenente gli allineamenti.

BWA: Opzioni principali

OPZIONE	Descrizione
-t INT	Numero di thread da usare per parallelizzare il processo (es. -t 8 per 8 thread)
-o FILE	Specifica un file di output (se diverso da stdout)
-k INT	Lunghezza minima del seed per iniziare un allineamento (default: 19)
-T INT	Soglia minima del punteggio per riportare un allineamento (default: 30)

BWA vs BOWTIE2

Entrambi i tool sono ampiamente utilizzati nella comunità bioinformatica e hanno contribuito in modo significativo all'analisi dei dati di sequenziamento ad alto rendimento. La scelta tra i due spesso dipende dalla natura specifica dei dati e dagli obiettivi dell'analisi.

BWA vs BOWTIE2

Metodologia di Indicizzazione

- Bowtie2 utilizza un approccio basato sulla Burrows-Wheeler Transform (BWT) per costruire il suo indice.
- BWA anch'esso utilizza la BWT, ma ha diversi algoritmi, tra cui BWA-backtrack, BWA-SW e BWA-MEM

Tipi di Reads Supportate

- Bowtie2 supporta reads più lunghi rispetto alla sua versione precedente (Bowtie) e può gestire reads di qualsiasi lunghezza con una qualità di allineamento variabile.
- BWA è particolarmente efficiente con reads di lunghezza moderata (come 70-100bp da Illumina). *Tuttavia, con BWA-MEM, può gestire reads molto lunghi come quelli prodotti da sequenziatori come*
- *Oxford Nanopore o PacBio*

BWA vs BOWTIE2

Scenari di Utilizzo

- Bowtie2 è spesso preferito per applicazioni che richiedono tempi di elaborazione rapidi, come applicazioni trascrittomiche dove è necessario mappare molte reads su trascritti noti.
- BWA, in particolare BWA-MEM, è spesso la scelta di riferimento per allineare reads su genomi di riferimento completi, come il genoma umano, a causa della sua accuratezza e efficienza

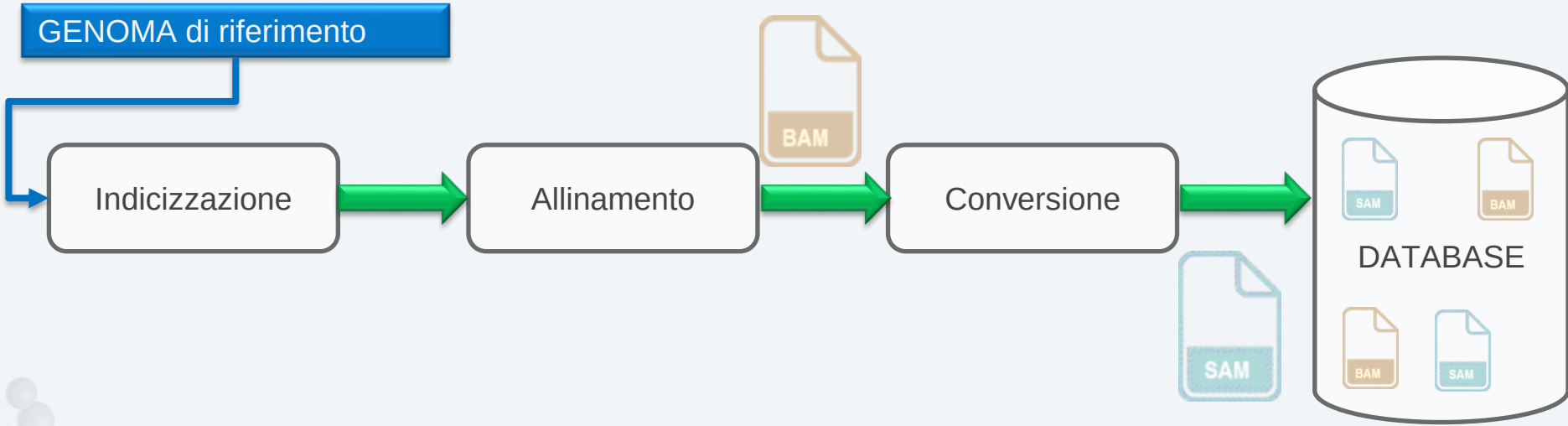
Funzionalità Aggiuntive

- BWA-MEM, uno degli algoritmi di BWA, offre una migliore gestione delle regioni con alti tassi di variabilità o ripetizioni, e può identificare sub-optimal allineamenti, cosa che può essere utile per identificare varianti strutturali.

BWA vs BOWTIE2

Prestazioni:
Sebbene sia Bowtie2 che BWA siano efficienti, le loro prestazioni possono variare in base alla natura dei dati e alle specifiche esigenze del progetto. Ad esempio, Bowtie2 potrebbe essere più veloce in certi contesti, mentre BWA potrebbe offrire un allineamento più accurato per reads complesse o problematiche

WORKFLOW
MAPPING READS CONTRO GENOMA



Grazie!

Domande?

franco.liberati@unitus.it
deb.scienceontheweb.com