Università degli Studi di Camerino

Scuola di Scienze e Tecnologie

Corso di Laurea in Informatica

Corso di Algoritmi e Strutture Dati 2019/2020

Parte di Laboratorio (6 CFU)

Docente: Luca Tesei

**Assegnazione del Progetto Totale n. 1**

## Validità del Progetto

Il progetto può essere presentato in uno qualsiasi degli appelli fissati su ESSE3 per l’Anno Accademico 2019/2020, a partire da Febbraio 2020 e fino a Marzo 2021 (Appello VIII 2019-20). Il progetto può essere presentato solo da studenti che si siano iscritti al primo anno nell’Anno Accademico 2018/2019 o precedenti.

## Realizzazione in Gruppo

Questo progetto può essere realizzato singolarmente o in un gruppo di massimo 2 studenti.

## Obiettivi del Progetto

1. Realizzare in Java una implementazione degli algoritmi di allineamento di sequenze di simboli denominati algoritmo di **Needleman-Wunsch** e algoritmo di **Gotoh**, descritti nella sezione seguente. Entrambi i problemi vanno risolti con la tecnica della **programmazione dinamica**.
2. Realizzare **una suite di test JUnit 5** che controlli le funzionalità implementate.
3. **Valutare numericamente le prestazioni** degli algoritmi implementati.
4. Scrivere una **relazione** sul progetto e fare una **presentazione** orale dello stesso utilizzando delle slides e il codice scritto.

## Descrizione del Problema

### Allineamento di Sequenze

Una sequenza è una successione finita di simboli presi da un certo alfabeto. In Java si può tranquillamente equiparare a una stringa di caratteri. In biologia sequenze di lettere specifiche vengono usate per rappresentare filamenti di DNA o RNA. Dopo la scoperta di questi, c’è stata una forte spinta a sviluppare metodi automatici (algoritmi) per il riconoscimento e il confronto di sequenze che ha portato alla nascita della bioinformatica, un ambito di ricerca a cavallo tra l’informatica e la biologia.

In bioinformatica, uno dei problemi su sequenze che risulta utile risolvere in maniera efficiente è quello della similarità. Ciò è giustificato dal fatto che sequenze simili probabilmente hanno avuto una evoluzione simile o si sono “evolute” da antenati comuni. Ma di preciso cosa significa “similarità”? Le risposte possono essere diverse, anche in base al tipo di “relazione biologica” che si cerca. Una delle nozioni di similarità più usate è basata sulla nozione di allineamento di sequenze.

Supponiamo di avere un simbolo speciale, chiamato **gap**, e rappresentato solitamente da un trattino: “-”. Un allineamento di due sequenze date è una coppia di sequenze che contengono potenzialmente dei gap e che indica come trasformare una sequenza nell’altra e viceversa.

Ad esempio, un possibile allineamento delle sequenze ACCCGA e ACTA è solitamente rappresentato in questo modo:

ACCCGA

AC--TA

Questo particolare allineamento indica che la prima A di entrambe le sequenze è stata riconosciuta **uguale** (ha fatto “**match**” in inglese). Anche la seconda C di entrambe le sequenze ha fatto match. La terza C della prima sequenza è stata invece “colmata” con un gap. Se diamo un ordine alle stringhe in allineamento possiamo pensare che la C è stata **cancellata** (se la sequenza di partenza è ACCCGA) o è stata **inserita** (se la sequenza di partenza è ACTA). La stessa cosa succede per la terza C. Per quanto riguarda la G, invece, in questo allineamento è stata **sostituita** con T (o, equivalentemente, la T è stata sostituita con G). Infine l’ultima A fa di nuovo match. Quindi, tirando le somme, nell’allineamento preso in considerazione sono presenti le seguenti **operazioni**: tre match di lettere uguali, una sostituzione di lettera e due inserimenti/cancellazioni.

Naturalmente possono esistere diversi allineamenti fra le stesse stringhe. Ad esempio potrebbe essere considerato il seguente:

ACCCGA

AC-T-A

in questo caso le operazioni sono le stesse dell’altro allineamento, ma sono applicate a lettere differenti. Inoltre, ci sono due “zone di gap” (entrambe di lunghezza 1) invece di una sola di lunghezza 2.

Un altro esempio è l’allineamento delle due sequenze ACGGAT e CCGCTT. Vediamo due possibili esempi di allineamento:

AC-GG-AT

-CCGCT-T

oppure

ACGG---AT

--CCGCT-T

e naturalmente ne esistono di altri, potenzialmente infiniti!

Ma allora qual è l’allineamento “giusto”? L’idea di base è che ad ogni operazione presente nell’allineamento, tranne ovviamente la match, si può assegnare un certo **costo**. Quindi si può prendere come allineamento di due sequenze uno qualsiasi per cui il costo totale delle operazioni (la somma) sia **minimo**. Ne risulta quindi che il problema di trovare l’allineamento tra due sequenze è un problema di **ottimizzazione**.

Formalizziamo ora il problema: siano e due sequenze di lunghezza e . Indichiamo con () l’i-esimo simbolo di () , con (). Un allineamento di eè una coppia in cui e sono sequenze ottenute da e inserendo dei gap e tali che . Definiamo le possibili operazioni come coppie di simboli di e . Una operazionecon e dove o può essere un gap (ma non entrambi) può essere una delle seguenti:

* **match**:con ;
* **sostituzione**: con ;
* **inserimento**: ;
* **cancellazione**: .

Indichiamo con una funzione che assegna i costi alle operazioni, detta **funzione di costo** o **funzione di scoring**. Indichiamo quindi conil costo associato all’operazione e assumiamo che il costo di ogni tipo di operazione sia, in generale, maggiore o uguale di zero tranne che nel caso di match, per cui il costo è uguale a zero. Il **costo totale** associato all’allineamento è quindi definito come:

Date due sequenze e e una funzione di costo , quindi, esistono tantissimi possibili allineamenti ognuno con il suo costo . Poiché gli allineamenti sono sempre coppie di sequenze finite (lunghe un certo ) tutti i costi totali associati agli allineamenti saranno numeri maggiori o uguali di zero. Quindi, esiste il minimo tra tutti i possibili costi totali associati agli allineamenti. Tale numero, che dipende unicamente dalle sequenze di partenza e e dalla funzione di costo scelta ,viene chiamato **distanza di allineamento** ed è definito formalmente come segue:

è un allineamento di e

Vediamo degli esempi. Assumiamo di avere una funzione costo che assegna costo 0 al match e costo 1 a sostituzione, inserimento e cancellazione. Quindi, l’allineamento

ACCCGA

AC--TA

ha costo totale 3, derivante da due cancellazioni e una sostituzione. Anche l’allineamento

ACCCGA

AC-T-A

ha costo totale 3, derivante sempre da due cancellazioni e una sostituzione, anche se fatte in maniera diversa. L’allineamento

AC-GG-AT

-CCGCT-T

corrisponde alla sequenza di operazioni: cancellazione, match, inserimento, match, sostituzione, inserimento, cancellazione, match che somma a un costo totale di 5. L’allineamento

ACGG---AT

--CCGCT-T

corrisponde a: cancellazione, cancellazione, sostituzione, sostituzione, inserimento, inserimento, inserimento, cancellazione, match che somma a un costo totale di 8. In questo caso l’allineamento precedente ha un costo minore, ma non sappiamo se sia il minimo possibile.

Come possiamo procedere per calcolare in maniera efficiente il minimo, cioè la distanza di allineamento? Ci viene in aiuto la **programmazione dinamica**! Possiamo far vedere infatti che è possibile scomporre il problema di allineamento ottimo in sottoproblemi analoghi e che tali sottoproblemi godono della proprietà della sottostruttura ottima.

Definiamo come **prefisso di lunghezza** , con di una sequenza come la sequenza che si ottiene prendendo i primi elementi di . Indichiamo tale prefisso con la notazione

Per risolvere il problema della distanza di allineamento di e con una data funzione di costo partiamo dal considerare i sottoproblemi che riguardano la distanza di allineamento fra prefissi di e . In particolare consideriamo tutti i possibili prefissi che sono per ed per . Considerando anche i prefissi vuoti, indicati con e (sono entrambi la sequenza vuota), otteniamo possibili coppie di prefissi. Mettiamo tutti questi sottoproblemi in una matrice , detta **matrice di allineamento**, con dimensione . Si ha quindi che l’elemento della matrice è definito come:

è un allineamento di e di

L’**algoritmo di Needleman-Wunsch** calcola la matrice utilizzando la seguente ricorrenza:

* , cioè l’allineamento ottimo tra la sequenza vuota () e se stessa () costa zero;
* per tutti gli tali che , , cioè l’allineamento ottimo tra la sequenza e la sequenza vuota () costa come la somma della cancellazione di tutti i simboli di ;
* per tutti i tali che , , cioè l’allineamento ottimo tra la sequenza vuota () e la sequenza costa come la somma dell’inserimento di tutti i simboli di ;
* per tutti gli tali che e per tutti i tali che si ha che

cioè l’allineamento ottimo tra i prefissi e si ottiene considerando il minimo costo che si ottiene dalle possibili operazioni che possono essere fatte:

* match o sostituzione di e più il costo ottimo del sottoproblema ;
* cancellazione di da più il costo ottimo del sottoproblema
* inserimento di in più il costo ottimo del sottoproblema .

Il calcolo della matrice può essere fatto in tempo utilizzando la tecnica della **programmazione dinamica**. Quindi il problema può essere risolto in maniera efficiente e la soluzione è il numero che si trova in . L’ordine di riempimento della matrice dipende dalla ricorrenza definita: la sua determinazione è lasciata al gruppo di studenti.

Facciamo un semplice esempio: supponiamo di voler allineare le sequenze AT e AAGT con una funzione costo che assegna costo 1 a tutte le operazioni tranne la match. La matrice D che risulta dal calcolo della ricorrenza è la seguente.

|  |  | A | A | G | T |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 |
| A | 1 | 0 | 1 | 2 | 3 |
| T | 2 | 1 | 1 | 2 | 2 |

Quindi si ha che . Cioè è possibile allineare AT e AAGT effettuando solo due operazioni con costo 1.

Ma qual è un allineamento ottimo? In questo caso semplice è possibile indovinarlo a mano, ma in generale c’è bisogno di costruire una soluzione ottima. Questo problema si presenta sempre quando si risolvono i problemi con la programmazione dinamica. Si può risolvere in due modi: o si costruisce una matrice gemella a quella usata per calcolare il valore del costo ottimo e si inseriscono lì dentro le informazioni necessarie a ricostruire una soluzione ottima oppure si usa un algoritmo, detto di **traceback**, che legge la matrice e ricostruisce, iterativamente o ricorsivamente, una soluzione ottima ripercorrendo i passi effettuati dall’algoritmo di riempimento della matrice.

Ad esempio è possibile, dalla matrice, ricavare una soluzione ottima seguendo la “strada” indicata dalle caselle verdi:

|  |  | A | A | G | T |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 |
| A | 1 | 0 | 1 | 2 | 3 |
| T | 2 | 1 | 1 | 2 | 2 |

Si parte da per poi passare a per poi passare a per poi passare a e infine approdare a . L’allineamento ottimo corrispondente è il seguente:

-A-T

AAGT

Seguendo un’altra strada:

|  |  | A | A | G | T |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 |
| A | 1 | 0 | 1 | 2 | 3 |
| T | 2 | 1 | 1 | 2 | 2 |

è possibile ottenere un altro allineamento ottimo:

A--T

AAGT

La scelta di quale tecnica usare è lasciata al gruppo di studenti.

### Allineamento di Sequenze con Gap Penalty Affine

Abbiamo appena visto che l’allineamento

-A-T

AAGT

e l’allineamento

A--T

AAGT

sono entrambi ottimi secondo la distanza di allineamento che abbiamo definito. Tuttavia, se ci troviamo in presenza di sequenze genetiche, come in questo caso DNA, si ha che il secondo allineamento è biologicamente più ragionevole del primo. Infatti si pensa che l’evoluzione introduca più facilmente un gap più ampio (in questo caso due gap consecutivi) piuttosto che due gap più corti (in questo caso due gap lunghi 1 separati). Per questo motivo sarebbe preferibile raffinare la distanza di allineamento in modo che i due allineamenti considerati non abbiano più lo stesso costo, ma quello che contiene un gap più ampio abbia un costo minore. Questo è stato fatto introducendo un ulteriore costo negli allineamenti, dovuto appunto al numero e alla minore o maggiore lunghezza delle sequenze di gap introdotte. Per rappresentare questo costo si usa una funzione detta **gap penalty**. Ad esempio, se consideriamo l’allineamento

--A---GTA

AAATTTGT-

il costo delle operazioni è dato dall’introduzione di 3 gap: per i due gap all’inizio della prima sequenza (inserimento di due A) più per i tre gap dopo la A della prima sequenza (inserimento di tre T) più per il gap singolo alla fine della seconda sequenza (cancellazione di una A). Per le altre lettere si hanno dei match, quindi il costo è zero. Se ci fossero delle sostituzioni il loro costo andrebbe sommato ad una ad una utilizzando una funzione di costo classica .

Una funzione di gap penalty “biologicamente ragionevole” è data da quella che viene denominata **affine gap penalty** ed è definita come. Il parametro rappresenta il costo di aprire una nuova sequenza di gap mentre il parametro esprime il costo di aumentare di 1 la lunghezza di una sequenza di gap già iniziata. Anche se può apparire più complicato, il problema di trovare il costo minimo di allineamento tra e date le funzioni di costo e è risolvibile con la programmazione dinamica. L’**algoritmo di Gotoh** definisce le seguenti matrici:

* è definita in maniera simile all’algoritmo di Needleman-Wunsch, tranne che, appunto, il costo è definito dal costo delle sostituzioni dato da e dal costo delle gap penalty dato da;
* è definita come la matrice dei costi ottimali di allineamento dei prefissi e che terminano con un una coppia, cioè quandoviene cancellato;
* è definita come la matrice dei costi ottimali di allineamento dei prefissi e che terminano con un una coppia, cioè quandoviene inserito.

Le tre matrici sono definite, in maniera interlacciata **ma non circolare**, dalle seguenti ricorrenze:

* Ricorrenza per , che usa le matrici e :
  + , cioè l’allineamento ottimo tra la sequenza vuota () e se stessa () costa zero;
  + per tutti gli tali che , , cioè l’allineamento ottimo tra la sequenza e la sequenza vuota () costa come l’introduzione di un gap lungo ;
  + per tutti i tali che , , cioè l’allineamento ottimo tra la sequenza vuota () e la sequenza costa come l’introduzione di un gap lungo ;
  + per tutti gli tali che e per tutti i tali che si ha che

cioè l’allineamento ottimo con funzione di gap penalty affine tra i prefissi e si ottiene considerando il minimo costo che si ottiene dalle possibili operazioni che possono essere fatte:

* + match o sostituzione di e più il costo ottimo del sottoproblema ;
  + cancellazione di da calcolato tenendo conto della funzione di gap penalty affine, si trova in ;
  + inserimento di in calcolato tenendo conto della funzione di gap penalty affine, si trova in .
* Ricorrenza per , che usa la matrice :
  + per tutti i tali che , , la matrice ha una ricorrenza solo sull’indice , quindi non c’è bisogno di inizializzare le caselle con ;
  + per tutti gli tali che e per tutti i tali che si ha che

cioè l’allineamento ottimo con funzione di gap penalty affine si ottiene scegliendo il minimo tra iniziare un nuovo gap cancellazione () e prolungare un gap già iniziato ().

* Ricorrenza per , che usa la matrice :
  + per tutti gli tali che , , la matrice ha una ricorrenza solo sull’indice , quindi non c’è bisogno di inizializzare le caselle con ;
  + per tutti gli tali che e per tutti i tali che si ha che

cioè l’allineamento ottimo con funzione di gap penalty affine si ottiene scegliendo il minimo tra iniziare un nuovo gap di inserimento () e prolungare un gap già iniziato ().

Ad esempio, consideriamo una funzione costo che assegna costo 1 per ogni sostituzione e una funzione di gap penalty affine in cui si è scelto e . Allineiamo le due sequenze e . Le matrici sono le seguenti:

|  |  | A | C | C | T |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 5 | 6 | 7 | 8 |
| C | 5 | 1 | 5 | 6 | 8 |
| C | 6 | 6 | 1 | 5 | 7 |

|  |  | A | C | C | T |
| --- | --- | --- | --- | --- | --- |
|  | 0 |  |  |  |  |
| C | - | 10 | 11 | 12 | 13 |
| C | - | 6 | 10 | 11 | 13 |

|  |  | A | C | C | T |
| --- | --- | --- | --- | --- | --- |
|  | 0 | - | - | - | - |
| C |  | 10 | 6 | 7 | 8 |
| C |  | 11 | 11 | 6 | 7 |

Come per l’algoritmo di Needleman-Wunsch, è possibile calcolare tutti i valori della matrice e delle altre due matrici associate in tempo utilizzando la tecnica della programmazione dinamica. La determinazione di un ordine corretto di calcolo per il riempimento delle tre matrici è lasciata al gruppo di studenti.

Anche nel caso di allineamento con funzione di gap penalty affine è possibile ricostruire una soluzione ottima tramite la memorizzazione di informazioni opportune in matrici gemelle a quelle calcolate oppure tramite la definizione di una opportuna funzione di traceback. Nell’esempio, un allineamento ottimo che può essere ricostruito è il seguente:

CC--

ACCT

Il costo totale è il minimo, come calcolato in , ed è ottenuto come , contando una sostituzione e due gap consecutivi da inserimento. Un altro possibile allineamento ottimo che si può ricostruire dalle matrici seguendo una strada diversa è il seguente:

--CC

ACCT

Anche qui il costo è dovuto a due gap consecutivi da inserimento e a una sostituzione.

La determinazione della tecnica per ricostruire una soluzione ottima in questo caso è lasciata al gruppo di studenti.

## Realizzazione del Progetto

Per realizzare il progetto devono essere completati i seguenti task.

### Importazione della Traccia

La traccia del codice è fornita come progetto Maven (Apache Maven: <https://maven.apache.org/> ) in Eclipse.

**Per il download** (si assume l’uso di Eclipse, se si usano altri IDE i passi sono più o meno analoghi):

* scaricare la traccia in formato .zip del progetto maven/eclipse fornito in allegato al post di assegnazione del progetto in Google Classroom;
* scompattare lo .zip in una cartella di propria scelta (la scelta migliore è dentro la cartella del workspace della propria installazione di eclipse);
* da eclipse scegliere File -> Import… poi Selezionare Maven->Existing Maven Projects poi selezionare (tasto Browse…) la cartella scompattata poi dare Finish;
* a questo punto si ha il project in Eclipse e si possono aprire i sorgenti e modificarli e si possono scrivere e lanciare i test JUnit di ogni classe.

### Implementazione

Vanno implementati tutti i metodi richiesti (segnalati con commenti della forma // TODO testo). La specifica precisa delle API è data con commenti javadoc del codice. Non è consentito:

* aggiungere classi pubbliche;
* modificare la firma dei metodi già specificati nella traccia;
* modificare le variabili istanza già specificate nella traccia.

E’ consentito:

* aggiungere classi interne private per fini di implementazione;
* aggiungere metodi privati per fini di implementazione;
* aggiungere variabili istanza private per fini di implementazione.

La versione del compilatore Java definita nel progetto è la 1.8 (Java 8).

Nell’implementazione:

* **non si possono usare versioni del compilatore superiori a 1.8**;
* **è vietato l’uso di lambda-espressioni** e di funzionalità avanzate di Java 8 o superiore (es. stream, ecc…);
* **è obbligatorio usare il set di caratteri utf8** per la codifica dei file del codice sorgente.

### Fase di Test

Vanno progettati e implementati uno o più test JUnit 5 (<https://junit.org/junit5/docs/current/user-guide/>) per tutti i metodi pubblici delle classi del progetto. In particolare, se un metodo ha più valori di ritorno possibili oppure se può lanciare eccezioni, un test apposito deve essere previsto per ogni caso. I test vanno implementati nelle classi <NomeClasse>Test che sono state già create con lo stub del codice per ogni metodo nella cartella /src/test/java/it/unicam/cs/asdl1920/project1 all’interno del progetto. In caso di più test per lo stesso metodo cambiare i nomi. Se il test controlla il lancio di un’eccezione va inserita una condizione come segue:

@Test

public final void testMethod1() {

assertThrows(<NomeEccezione>.class,

() -> <codice che dovrebbe far lanciare l’eccezione> );

}

Il test avrà successo se verrà lanciata l’eccezione indicata, fallirà altrimenti.

Se il test controlla che il metodo calcoli valori corretti, allora dovrà essere strutturato come segue:

@Test

public final void testMethod1() {

// Codice che prepara i valori per chiamare il metodo da testare

assertTrue(obj.method1() == 5);

assertTrue(obj.isSomething());

assertTrue(obj.method2().equals(objTest));

// Codice che prepara i valori per chiamare il metodo da testare

assertFalse(obj.method3() == null);

assertFalse(obj.method4().equals(""));

}

La clausola assertTrue(espressione booleana o chiamata di metodo booleano) fallisce se il valore dell’espressione booleana o il valore di ritorno del metodo booleano è falso. Nel caso si voglia testare l’uguaglianza di oggetti è preferibile usare il metodo equals e testare il risultato di quest’ultimo: assertTrue(obj.method2().equals(objTest)). La clausola assertFalse(espressione booleana o chiamata di metodo booleano) fallisce se il valore dell’espressione booleana o il valore di ritorno del metodo booleano è vero. Se non ci sono eccezioni non previste e nessuna clausola assertTrue o assertFalse fallisce allora il test avrà successo.

L’implementazione dovrebbe passare tutti i test forniti.

### Valutazione Numerica delle Prestazioni

Eseguire il framework di valutazione numerica degli algoritmi di allineamento (fornito nel codice) acquisendo i dati per l’algoritmo di Needleman-Wunsch e per quello di Gotoh. I dati prodotti dal framework di valutazione (file .csv, comma-separated values [https://it.wikipedia. org/wiki/Comma-separated\_values](https://it.wikipedia.) ) dovranno essere elaborati con un foglio elettronico (ad esempio Fogli Google, Excel, OpenOffice o LibreOffice) o con un framework che permette l’uso di tabelle di dati e calcoli statistici (ad esempio R, MatLab o Mathematica). **Per semplificare** questa analisi il framework applica l’algoritmo implementato nelle classi NeedlemanWuschAligner e GotohAligner a coppie di stringhe che hanno la stessa lunghezza *n*, per cui ci si aspetta che il tempo di esecuzione sia *O(n2)*, corrispondente a *O(mn)* quando *m = n.* I dati devono essere usati per calcolare, per valori di lunghezza crescenti di coppie di sequenze (stringhe) generate casualmente:

* il minimo (caso ottimo),
* il massimo (caso pessimo),
* la media aritmetica (caso medio) e
* la deviazione standard (caso medio)

dei seguenti valori:

* tempo di esecuzione dell’algoritmo implementato nella classe NeedlemanWuschAligner in nanosecondi e
* tempo di esecuzione dell’algoritmo implementato nella classe GotohAligner in nanosecondi.

I dati elaborati dovranno poi essere usati per creare dei grafici che permettano di valutare le prestazioni degli algoritmi implementati. In particolare, **per ognuno dei due algoritmi**, deve essere generato un grafico a linee sovrapposte che riporta le curve (spezzate) del valore minimo, massimo, medio, medio + deviazione standard, medio - deviazione standard del tempo di esecuzione in nanosecondi dell’algoritmoper le varie varie coppie di sequenze. Sulle ascisse vanno riportati i valori *n* crescenti della lunghezza delle due sequenze analizzate. Nello stesso grafico va inoltre riportata la curva (spezzata) *c1 \* n2* dove il valore della costante di scala *c1* deve essere scelto opportunamente per permettere la visualizzazione efficace delle varie curve mostrate e il loro andamento asintotico. **Opzionalmente**: se la linea spezzata del massimo (caso pessimo) presenta molti picchi fuori scala (dovuti, si spera, ad attività della Java Virtual Machine o del sistema operativo che non si possono spegnere) si può ovviare non considerando i valori outliers nel calcolo del massimo. Per la definizione dei valori outliers si può seguire uno degli approcci illustrati a lezione.

Tutti i grafici prodotti, corredati da una breve descrizione testuale, devono essere riportati nella relazione scritta del progetto.

Per una corretta esecuzione del framework di valutazione si devono evitare tutte le possibili interferenze di altri processi in esecuzione nel computer che si sta utilizzando. Quindi si consiglia di:

* chiudere tutte le applicazioni e tutti i servizi del sistema operativo non necessari;
* staccare/disattivare la rete;
* assicurarsi di aver implementato correttamente (usando la programmazione dinamica) le classi NeedlemanWuschAligner e GotohAligner;
* eseguire la classe main del framework da riga di comando invece che dall’interno di Eclipse. Per far questo, salvare tutto e chiudere Eclipse. Poi aprire una finestra terminale (dipende dal sistema operativo, ad esempio su Windows bisogna eseguire cmd.exe oppure Prompt dei comandi). Eseguire:

> cd <path del workspace>/asdl1920-project1-traccia/target/classes + INVIO

sostituendo a <path del workspace> il percorso della cartella workspace di Eclipse (oppure il percorso della cartella dove si trova il progetto se non è nel workspace di Eclipse) e poi eseguire:

> java -cp . it.unicam.cs.asdl1920.project1.SequenceAlignmentEvaluationFramework CSVData

in questo modo verranno generati i due file con i dati evalfram.csv e sequences.csv all’interno della cartella

<path del workspace>/asdl1920-project1-traccia/target/classes/CSVData/

si può indicare un’altra cartella mettendone il percorso nella linea di comando al posto di CSVData. Se non si indica niente i due file vengono generati nella cartella corrente. Il file sequences.csv è solo per la verifica delle sequenze generate e non deve essere utilizzato per generare i grafici.

### Scrittura della relazione

La relazione scritta deve illustrare le parti salienti del progetto, in particolare l’analisi svolta per la risoluzione del problema, le principali soluzioni implementative e i risultati del framework di valutazione. Nella relazione possono essere riportati e commentati dei pezzi di codice che sono significativi, ad esempio il core dell’implementazione degli algoritmi di programmazione dinamica.

La relazione può essere scritta con un editor di testi WYSIWYG (What You See Is What You Get) come Microsoft Word o OpenOffice oppure con il sistema LATEX. In quest’ultimo caso un mini corso è disponibile qui: [http://didattica.cs.unicam.it/doku.php?id=informazioni:documentiutile: main](http://didattica.cs.unicam.it/doku.php?id=informazioni:documentiutile:)

La relazione deve essere fornita come file PDF.

### Preparazione della presentazione orale

La prova orale consiste, nella fase iniziale, in una presentazione del progetto da parte di tutti i componenti del gruppo. Questa presentazione dovrà essere supportata da slides e dall’esposizione del codice implementato. Può essere realizzata anche una piccola demo.

Tutti i componenti del gruppo dovranno esporre la presentazione, dividendosi il tempo necessario, che comunque **non può eccedere i 15 minuti**.

La presentazione deve esporre le fasi salienti del progetto e i risultati ottenuti, permettendo di avere un panoramica completa del lavoro svolto.

Le slides della presentazione devono essere sintetiche e possibilmente contenere immagini o parole chiave per punti. **Non fare delle slides che contengono frasi lunghe; le slide devono essere comprese a colpo d’occhio**.

## **Modalità di Consegna**

Va compilato il file DACOMPILARE.txt presente nella cartella principale del progetto, inserendo i dati richiesti. Ogni studente ha un proprio spazio Google Drive corrispondente all’indirizzo istituzionale:

nome.cognome@studenti.unicam.it

(o una variante di questo schema). Il primo componente del gruppo (in ordine alfabetico per cognome) dovrà creare nel suo spazio Google Drive una cartella che si chiami esattamente così:

**ASDL1920-NOME-COGNOME-PT1**

ad esempio **ASDL1920-MARIO-ROSSI-PT1**. Nel caso di più nomi/cognomi usare solo il primo nome e il primo cognome. Nel caso di lettere accentate nel nome/cognome usare le corrispondenti lettere non accentate (maiuscole). Nel caso di apostrofi o altri segni nel nome/cognome ometterli. Nel caso di particelle nel nome o nel cognome, ad esempio De Rossi o De’ Rossi, attaccarle (DEROSSI).

All’interno della cartella Google Drive creata va caricata la cartella project1 (con esattamente questo nome) contenente il progetto implementato con la stessa struttura di file e cartelle fornita con la traccia. Si tratta di copiare e caricare direttamente così com’è la cartella del progetto all’interno del workspace di Eclipse (o nella posizione in cui si trova se non è nel workspace). Tutti i file devono essere caricati, compresi quelli di progetto di Eclipse (o altro IDE) e quelli di Maven.

I risultati del framework di valutazione, cioè i file .csv generati e i file (Microsof Excel o altro) usati per l’elaborazione devono essere caricati nella cartella

project1/src/main/evaluationFrameworkData

La relazione in formato PDF deve essere caricata nella cartella

project1/src/main/RelazioneScritta

Una volta caricati tutti i file, il primo componente del gruppo (in ordine alfabetico per cognome) deve condividere la cartella **ASDL1920-NOME-COGNOME-PT1** e tutti i file che contiene con la possibilità di modificare (flag “Può Modificare” in fase di condivisione) con il docente luca.tesei@unicam.it e con tutti gli altri componenti del gruppo (usando l’email istituzionale unicam).

La data entro cui consegnare il progetto (entro le 23.59 di quel giorno) è fissata per ogni appello in ESSE3 (Prova Parziale con nome “XXX Consegna Progetto Tot ASDL 2019-20”). Farà fede la data di caricamento dei file su Google Drive. Entro il giorno prima della data di scadenza (scadenza iscrizioni in ESSE3) tutti gli studenti del gruppo devono registrarsi su ESSE3 alla prova parziale corrispondente.

Il giorno successivo alla consegna sarà fissato il calendario degli orali, che si svolgeranno nei giorni successivi. Tutti i componenti del gruppo saranno convocati all’orale tramite un invito di Google Calendar sulla loro email istituzionale unicam. In caso di problemi per la data e l’ora indicate contattare subito il docente via email.

## Valutazione

**Nota Bene**: il codice consegnato verrà sottoposto a un **controllo antiplagio** utilizzando un tool specifico. A seconda del grado di somiglianza con il codice consegnato da altri gruppi, anche in appelli precedenti, la commissione di esame potrà decidere di **non accettare** il progetto o di **decurtare il punteggio** comune riguardante il progetto realizzato (si veda sotto) di un certo numero di punti.

La valutazione della prova di Laboratorio di Algoritmi si baserà sui seguenti criteri, in ordine decrescente di importanza:

1. Qualità dell’analisi del problema, della sua soluzione e della sua esposizione, nel codice, nella relazione scritta e nella presentazione orale.
2. Qualità e successo dei test JUnit definiti e implementati.
3. Contenuto e qualità della relazione scritta e dei grafici in essa contenuti riguardanti i risultati del framework di valutazione.
4. Codice chiaro e ben commentato.
5. Qualità delle slides, della presentazione orale e delle risposte alle domande.

Durante la presentazione orale o successivamente ci saranno domande specifiche ai componenti del gruppo. Il voto finale consiste di una parte comune riguardante il progetto realizzato e di una parte individuale riguardante la prova orale di ogni componente del gruppo. Alla fine ogni componente del gruppo riceverà il proprio voto in trentesimi.