Progetto Totale n. 1 di
Laboratorio di Algoritmi e Strutture Dati

Anno Accademico 2018/19

Corso di Laurea in Informatica

Università di Camerino

Prof. Luca Tesei

Obiettivi del Progetto

1. Realizzare in Java una implementazione dell'algoritmo di Nussinov che, data una sequenza di nucleotidi, trova una struttura secondaria di RNA senza pseudonodi con un numero massimo di legami deboli. Il problema va risolto con la tecnica di programmazione dinamica.
2. Realizzare una suite di test JUnit 4 che controlli le funzionalità implementate.
3. Valutare numericamente le prestazioni dell'algoritmo implementato.
4. Scrivere una relazione sul progetto e fare una presentazione orale dello stesso utilizzando delle slides e il codice scritto.

Realizzazione in Gruppo

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

Descrizione

Introduzione biologica

L'acido ribonucleico (RNA) è un polimero lineare composto da quattro differenti tipi di nucleotidi che sono collegati tra loro da un legame forte a formare una sequenza detta *struttura primaria*. La sequenza è orientata da sinistra a destra seguendo la polarità da $5'$ a $3'$. I quattro tipi di nucleotidi sono: Adenina (A), Guanina (G), Citosina (C) e Uracile (U). Alcune sequenze di RNA si attorcigliano su loro stesse creando una *struttura secondaria*. Durante questo processo i nucleotidi non adiacenti della sequenza corrispondente alla struttura primaria possono creare tra di loro dei legami cosiddetti deboli. I tipi di legami deboli che si possono formare, nella maggior parte dei casi, sono le coppie di base di Watson e Crick (G-C o C-G e A-U o U-A) o le coppie "oscillanti" - wobble in inglese - (G-U o U-G). Questi legami, detti *coppie ammissibili*, fanno diminuire l'energia libera della molecola e il processo naturale porta alla formazione di una struttura secondaria tale che l'energia libera della molecola risultante è minima [2].

Una struttura secondaria è detta senza *pseudonodi* se i legami deboli di cui è composta non formano incroci; se è presente almeno un incrocio allora la struttura è detta con pseudonodi. La Figura 1 mostra una struttura secondaria con pseudonodi in cui la prima parte (A) è composta da legami deboli che non si incrociano, mentre nella parte (B) alcuni legami creano degli incroci.

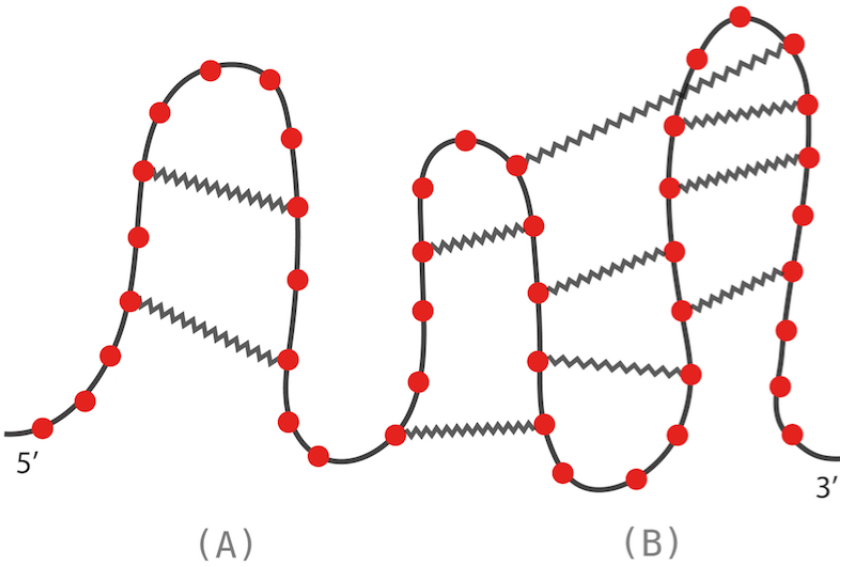

Figura 1: Una struttura secondaria di RNA. Ogni nucleotide è rappresentato da un pallino rosso, i legami forti della struttura primaria sono disegnati con una linea continua e i legami deboli della struttura secondaria sono disegnati con una linea a zigzag. La parte (A) è senza pseudonodi mentre la parte (B) contiene degli incroci, facendo diventare l'intera struttura con pseudonodi.

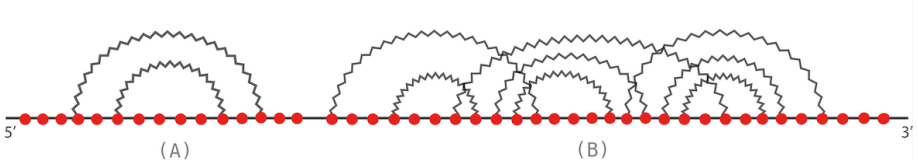

Figura 2: La struttura secondaria di Figura 1 rappresentata come diagramma. Gli pseudonodi nella parte (B) sono chiaramente visibili come archi curvi che si incrociano.

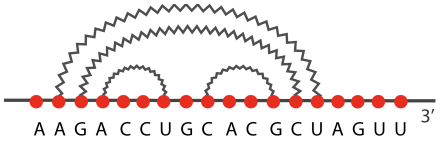

Figura 3: Una struttura secondaria senza pseudonodi rappresentata come diagramma. I legami deboli rispettano i vincoli sulle coppie che si possono formare.

Rappresentazione di strutture secondarie

Per visualizzare meglio una struttura secondaria e i suoi legami deboli si usa spesso una rappresentazione detta *diagramma*. In un diagramma, la sequenza di nucleotidi vine disegnata orizzontalmente da 5' a 3' connettendo tutti i nucleotidi con un arco diritto orizzontale. I legami deboli invece vengono rappresentati con archi curvi sopra la riga orizzontale che connettono i nucleotidi coinvolti. La struttura secondaria di Figura 1 è rappresentata come diagramma nella Figura 2. Si osserva abbastanza facilmente che le parti in cui gli archi curvi non si incrociano rappresentano pezzi di struttura senza pseudonodi, mentre se ci sono almeno due archi curvi che si incrociano allora si è in presenza di pseudonodi. In Figura 3 c'è la rappresentazione a diagramma di una struttura secondaria. In essa sono riportati anche i codici dei nucleotidi della struttura primaria. Si noti che le coppie che hanno legami deboli sono tutte di un tipo consentito.

Una struttura secondaria senza pseudonodi si può rappresentare anche con una stringa tramite la *notazione dot-bracket*. In questa notazione, se il numero di nucleotidi è minore o uguale di 50, viene prima rappresentata la stringa di nucleotidi, seguita da un carattere di fine linea (a capo, "\n" in Java), seguita da una stringa di punti e parentesi tonde bilanciate che rappresenta i legami. Ad esempio, la struttura in Figura 3 è rappresentata in notazione dot-bracket come segue:

```
AAGACCUGCACGCUAGUU
.(((..).(..))).....
```

Un ulteriore modo per rappresentare la struttura secondaria è quello di associare alla stringa di nucleotidi un insieme di coppie di indici che indicano i legami deboli. Ad esempio, la struttura in Figura 3 è rappresentata dall'insieme di coppie di indici $\{(2, 14), (3, 13), (4, 7), (9, 12)\}$. In questa rappresentazione gli indici nelle coppie sono la posizione dei nucleotidi nella struttura primaria. Si noti che il primo nucleotide ha posizione 1.

Nel caso in cui la sequenza di nucleotidi sia lunga più di 50, allora la notazione dot-bracket prevede di spezzarla in linee successive (quindi separate da "a capo") tutte lunghe esattamente 50 nucleotidi tranne l'ultima. Dopo quest'ultima ci deve essere un carattere di "a capo" a cui deve seguire la rappresentazione dei legami deboli con i punti e le parentesi bilanciate, anch'essa spezzata in linee lunghe esattamente 50 tranne l'ultima. Ad esempio, una struttura secondaria la cui stringa di nucleotidi è più lunga di 50 è la seguente, con la relativa rappresentazione in notazione dot-bracket:

```
UUGAUUACGGAUCAAUUGAUUACGGAUCAAGACUACGGUUUGAUUACGGA
UCAA
((((((.....))))))((((((.....))))))..........((((((.....)
))))
```

La stessa struttura, se rappresentata con insiemi di coppie di indici, corrisponde a

$\{(5, 11), (4, 12), (3, 13), (2, 14), (1, 15), (20, 26), (19, 27), (18, 28), (17, 29), (16, 30), (44, 50), (43, 51), (42, 52), (41, 53), (40, 54)\}$

Diamo adesso delle definizioni utili per la definizione dell'algoritmo di Nussinov nella prossima sezione. Se S è una sequenza di nucleotidi lunga n indichiamo con $S[i]$, per $1 \leq i \leq n$, il nucleotide in posizione i (le posizioni iniziano da 1).

Definizione 1 (Struttura secondaria) *Sia S una sequenza di nucleotidi lunga n. Una struttura secondaria P di S è un insieme di coppie di indici, $P \subseteq \{1, 2, \ldots, n\} \times \{1, 2, \ldots, n\}$, che rappresentano legami deboli. Per ogni coppia $(i, j) \in P$ deve valere $i < j$ e $S[i]$-$S[j]$ deve essere una coppia ammissibile di nucleotidi. Inoltre, in P non possono esistere coppie diverse che coinvolgono gli stessi indici.*

Ad esempio, se $S =$ `ACUA`, le coppie $(2, 2)$ e $(3, 2)$ non sono possibili in una struttura secondaria di S perché non vale la condizione $i < j$; la coppia $(2, 4)$ non è possibile in una struttura secondaria di S perché $S[2]$-$S[4]$, cioè `C-A`, non è una coppia ammissibile di nucleotidi. Inoltre, nella stessa struttura secondaria non possono esistere le coppie $(1, 3)$ e $(3, 4)$ perché hanno in comune l'indice 3. Ciò significherebbe che nella struttura secondaria il nucleotide in posizione 3 è coinvolto in più di un legame debole e questo non è possibile.

Definizione 2 (Cardinalità di una struttura secondaria) *Data una struttura secondaria P, la sua cardinalità, indicata con $|P|$, è definita come il numero di coppie presenti in P.*

Definizione 3 (Struttura secondaria senza pseudonodi) *Sia P una struttura secondaria. P è detta senza pseudonodi se per ogni coppia $(i, j) \in P$ vale che ogni altra coppia $(i', j') \in P$ soddisfa una delle seguenti condizioni:*

1. *$i' < i < j < j'$, cioè la coppia (i', j') "contiene" la coppia (i, j);*
2. *$i < i' < j' < j$, cioè la coppia (i, j) "contiene" la coppia (i', j');*
3. *$j' < i$, cioè la coppia (i', j') "precede" la coppia (i, j);*
4. *$j < i'$, cioè la coppia (i', j') "segue" la coppia (i, j);*

In caso contrario la struttura P è detta con pseudonodi.

Ad esempio, $P = \{(1, 10), (2, 9), (4, 7), (5, 12)\}$ è con pseudonodi perché la coppia $(5, 12)$ "incrocia" le altre. Infatti, se prendiamo ad esempio la coppia $(4, 7)$, essa non contiene, non è contenuta, non segue e non precede la coppia $(5, 12)$. Quindi nessuna delle quattro condizioni della definizione vale.

Definizione 4 ((i, j)-sottostruttura) *Sia S una sequenza di nucleotidi lunga n. P è detta (i, j)-sottostruttura di S se e solo se valgono tutte le seguenti condizioni:*

- *$1 \leq i < j \leq n$;*
- *P è una struttura secondaria di S;*
- *per ogni coppia $(h, k) \in P$ si ha che h e k appartengono all'intervallo $[i..j]$ (estremi compresi).*

Ad esempio, se S è una sequenza lunga 7, allora $P = \{(2, 5), (3, 4)\}$, assumendo che sia una struttura secondaria di S, è una $(1, 4)$-sottostruttura di S.

Algoritmo di Nussinov

L'algoritmo di Nussinov et al. [1] è uno dei primi algoritmi di folding (predizione) di strutture secondarie sviluppato utilizzando la tecnica della programmazione dinamica. Esso predice strutture che non corrispondono alle reali molecole che si costituiscono in natura, ma rappresenta una ottima base di partenza per lo sviluppo di algoritmi di folding più realistici. Infatti, dopo di esso sono stati sviluppati molti altri algoritmi che tengono conto dell'energia libera della molecola di RNA (ad esempio il classico algoritmo di Zuker e Stiegler del 1981 [2]) o di altre informazioni biologiche, ma che si basano sull'idea originale dell'algoritmo di Nussinov.

Illustriamo adesso l'idea principale dell'algoritmo.

Input: una sequenza di nucleotidi S di lunghezza n.
Output: una struttura secondaria P di S senza pseudonodi che ha il numero massimo possibile di legami deboli.

Definizione 5 (Matrice di Nussinov) *La matrice di Nussinov N è una matrice di dimesione $n \times (n + 1)$ in cui ogni elemento $N_{i,j}$, tale che $1 \leq i \leq n$ e $i - 1 \leq j \leq n$, è definito come segue:*

$$N_{i,j} = \max\{|P| \mid P \text{ è una } (i, j)\text{-sottostruttura di } S \text{ senza pseudonodi } \}$$

La matrice di Nussinov si può calcolare con un approccio di programmazione dinamica tramite la seguente ricorrenza:

- per $1 \leq i \leq n$, $N_{i,i} = 0$ ed $N_{i,i-1} = 0$;

- per $1 \leq i < j \leq n$ si ha che:

$$N_{i,j} = \max \begin{cases} N_{i,j-1} \\ \max_{(i \leq k < j)}(N_{i,k-1} + N_{k+1,j-1} + 1) \end{cases} \quad \text{se} \quad S[k]\text{-}S[j] \text{ è una coppia ammissibile}$$

cioè, per calcolare l'elemento $N_{i,j}$ si deve calcolare il massimo tra $N_{i,j-1}$ e un'altro massimo. Quest'ultimo massimo si ottiene facendo variare k in $i \leq k < j$ e calcolando il valore dell'espressione $(N_{i,k-1} + N_{k+1,j-1} + 1)$.

Si noti che nel caso in cui $k = i$, $N_{i,k-1}$ corrisponde a $N_{i,i-1}$ che è inizializzato a 0.

La *soluzione ottima*, cioè il massimo numero di legami deboli per la sequenza S, si trova in $N_{1,n}$. Il calcolo della matrice di Nussinov per gli indici i, j con $i < j$ deve essere fatto in maniera *bottom-up* (come è tipico della tecnica di programmazione dinamica). L'ordine di calcolo degli elementi della matrice deve essere dedotto dalla struttura della ricorrenza. La struttura secondaria ottima associata alla soluzione ottima deve essere ricavata tramite informazioni accessorie (per esempio memorizzate in un'altra matrice) calcolate durante la risoluzione della ricorrenza o tramite la navigazione della matrice di Nussinov già precedentemente calcolata. A questo proposito si veda per esempio il problema della parentesizzazione ottima del prodotto di una sequenza di matrici visto a lezione.

Il calcolo della matrice in maniera bottom-up permette di risolvere il problema in tempo $\mathcal{O}(n^3)$ e in spazio $\mathcal{O}(n^2)$. Ciò deve essere verificato (per quanto riguarda il tempo) tramite il framework di valutazione numerica fornito nel codice della traccia.

Un piccolo esempio è il seguente. Sia $S =$ `GCACGACG`. Il numero massimo di legami deboli possibili in S è 3 e una struttura secondaria ottima è $P = \{(1, 2), (4, 8), (5, 7)\}$.

Realizzazione

Per realizzare il progetto devono essere completati i seguenti task.

Importazione del codice della traccia

Scaricare il progetto maven/eclipse fornito nel wiki (`http://didattica.cs.unicam.it/doku.php?id=didattica:triennale:asd:ay_1819:lab`) e importarlo in Eclipse (o altro IDE di programmazione) tramite le opportune funzioni di importazione per i progetti Maven.

Analisi del problema

Determinare la strategia di implementazione dell'algoritmo di programmazione dinamica la cui logica è stata introdotta nella sezione precedente. Come spiegato a lezione, per implementare correttamente un algoritmo con questa tecnica la parte di analisi "su carta" è determinante. Una attenzione maniacale va riservata agli indici, fonti di errori e bug molto difficili da eliminare dopo l'implementazione.

Scrittura del codice

Implementare le parti mancanti delle classi fornite, in particolare `SecondaryStructure` e `NussinovFolder`. Le API dei metodi di `NussinovFolder` sono specificate nell'interfaccia `FoldingAlgorithm`. Ogni punto da completare è segnalato da un tag `\\ TODO`. E' possibile aggiungere variabili istanza e metodi pubblici o privati, ma non è possibile cancellare o modificare le variabili istanza e i metodi che sono specificati.

Fase di test

Implementare uno o più test JUnit 4 per tutti i metodi implementati. In particolare, se un metodo ha più valori di ritorno possibili oppure può lanciare eccezioni, un test apposito deve essere previsto per ogni caso. I test vanno implementati nelle due classi `SecondaryStructureTest` e `NussinovFolderTest`,

che sono state già create con lo stub del codice per ogni metodo. In caso di più test per lo stesso metodo cambiare i nomi. Se il test controlla il lancio di una eccezione va inserita una condizione come segue:

```
@Test(timeout = 10000, expected = IllegalArgumentException.class)
    public final void testMethod1() {
        // Codice che dovrebbe far lanciare l'eccezione IllegalArgumentException
    }
```

Il test avrà successo se verrà lanciata l'eccezione indicata, fallirà altrimenti. Il tag `timeout = 10000` serve a far fallire il test se il tempo di esecuzione eccede i 10000 millisecondi, cioè 10 secondi. In questo modo se un metodo va in ciclo il test fallisce in un tempo ragionevole.

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

```
@Test(timeout = 10000)
    public final void testMethod1() {
        // Codice che prepara i valori per chiamare il metodo da testare
        assertEquals(15, obj.method1());
```

La clausola `assertEquals(valore atteso, valore calcolato)` fallisce se il valore atteso e il valore calcolato non sono uguali. Nel caso i valori siano oggetti è preferibile usare il metodo equals e testare il risultato di quest'ultimo: `assertEquals(true, obj.method1().equals(testValueObj))`. Se non ci sono eccezioni e nessuna clausola `assertEquals` fallisce allora il test avrà successo.

Valutazione numerica delle prestazioni

Eseguire il framework di valutazione numerica degli algoritmi di folding (fornito nel codice) acquisendo i dati per l'algoritmo di Nussinov implementato. I dati prodotti dal framework di valutazione (file .csv, comma-separated values `https://it.wikipedia.org/wiki/Comma-separated_values`) dovranno essere elaborati con un foglio elettronico (ad esempio Microsoft Excel o OpenOffice) o con un framework che permette l'uso di tabelle di dati e calcoli statistici (ad esempio R, MatLab o Mathematica) per calcolare, per valori di lunghezza crescenti delle sequenze di nucleotidi generate casualmente, i seguenti valori:

- il minimo (caso ottimo);
- il massimo (caso pessimo);
- la media aritmetica (caso medio);
- la deviazione standard (caso medio)

del tempo di esecuzione dell'algoritmo di Nussinov in nanosecondi.

I valori elaborati dovranno poi essere usati per creare dei grafici che permettano di valutare visivamente le prestazioni dell'algoritmo. Tale valutazione dovrà essere riportata, insieme ai grafici prodotti, nella relazione scritta del progetto. In particolare i grafici delle prestazioni nei vari casi dell'algoritmo dovranno essere comparati con i grafici delle funzioni $f(n) = n^2$ e $g(n) = n^3$.

Per il caso medio il grafico deve riportare la linea spezzata che, per valori crescenti di n, corrisponde al tempo medio di esecuzione e due linee spezzate che corrispondono al valore medio più la deviazione standard e al valore medio meno la deviazione standard.

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 la rete;

- 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`). Una volta aperto il terminale scrivere:

  ```
  > cd <path del workspace>/project1/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 scrivere:

  ```
  > java -cp . it.unicam.cs.asdl1819.project1.FoldingAlgorithmEvaluationFramework + INVIO
  ```

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 dell'algoritmo 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`.

La relazione deve essere fornita come file PDF.

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:

`ASDL1819-NOME-COGNOME-PT1`

ad esempio `ASDL1819-MARIO-ROSSI-PT1`.

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 caricare direttamente così com'è la cartella `project1` 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 `ASDL1819-NOME-COGNOME-PT1` e tutti i file che contiene con la possibilità di

modificare (flag "Può Modificare" in fase di condivisione) con `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 Lab aa 2018-19"). 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 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.

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.

Valutazione

La valutazione della prova di Laboratorio di Algoritmi si baserà sui seguenti criteri:

- Qualità dell'analisi del problema e sua esposizione nella relazione scritta e nella presentazione orale.
- Contenuto e qualità della relazione scritta e dei grafici in essa contenuti riguardanti i risultati del framework di valutazione.
- Qualità e successo dei test JUnit implementati.
- Codice chiaro e ben commentato.
- 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 suo voto in trentesimi.

Riferimenti bibliografici

[1] R. Nussinov, G. Pieczenik, J. R. Griggs, and D. J. Kleitman. Algorithms for Loop Matchings. *SIAM J. Appl. Math.*, 35(1):68–82, 1978.

[2] M. Zuker and P. Stiegler. Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. *Nucleic Acids Research*, 9(1):133–148, 1981.