Università degli Studi di Camerino
Scuola di Scienze e Tecnologie
Corso di Laurea in Informatica
Corso di Algoritmi e Strutture Dati 2021/2022
Parte di Laboratorio (6 CFU)
Docente: Luca Tesei

Istruzioni per la realizzazione del Miniprogetto 2

ASDL2122MP2

Descrizione

Il miniprogetto ASDL2122MP2 consiste nei seguenti task:

1. Implementare la classe `AdjacencyMatrixUndirectedGraph<L> implements Graph<L>` per realizzare un grafo non orientato utilizzando una variante della rappresentazione dei grafi con matrice di adiacenza
2. Implementare la classe `ForestDisjointSets` utilizzando foreste di alberi dove ogni albero rappresenta uno degli insiemi disgiunti
3. Implementare un algoritmo di calcolo delle componenti connesse di un grafo non orientato usando la classe `ForestDisjointSets`
4. Implementare l'algoritmo di Kruskal per il calcolo di un albero minimo di copertura in un grafo non orientato e pesato usando la classe `ForestDisjointSets`
5. Implementare l'algoritmo di Prim per il calcolo di un albero minimo di copertura in un grafo non orientato e pesato

Grafo

La classe `AdjacencyMatrixUndirectedGraph<L> extends Graph<L>` deve implementare tutti i metodi astratti della classe `Graph<L>` utilizzando una rappresentazione matriciale con queste strutture dati interne:

```
/* Insieme dei nodi e associazione di ogni nodo con il proprio
indice nella matrice di adiacenza */
protected Map<GraphNode<L>, Integer> nodesIndex;

/* Matrice di adiacenza, gli elementi sono null o oggetti della
classe GraphEdge<L>. L'uso di ArrayList permette alla matrice di
aumentare di dimensione gradualmente ad ogni inserimento di un
nuovo nodo e di ridimensionarsi se un nodo viene cancellato.*/
protected ArrayList<ArrayList<GraphEdge<L>>> matrix;
```

Si noti che, a differenza della rappresentazione standard, la matrice di adiacenza non è booleana, cioè `matrix.get(i).get(j)` non contiene un boolean che dice se esiste un arco tra il nodo i e il nodo j, ma contiene `null` se non c'è l'arco tra il nodo i e il nodo j oppure contiene l'oggetto `GraphEdge<L>` corrispondente all'arco stesso se l'arco esiste nel grafo. Questo permette di rappresentare anche il peso come attributo dell'oggetto `GraphEdge<L>` invece di usare una funzione peso w e permette di agevolare l'esecuzione di diversi metodi astratti della classe `Graph<L>` che si devono implementare.

Inoltre, si noti che l'assegnazione dell'indice ai nodi deve seguire l'ordine di inserimento, quindi il primo nodo inserito nel grafo dovrà avere assegnato l'indice 0, il secondo l'indice 1 e così via. In caso di cancellazione di un nodo il suo indice deve essere riciclato (e la matrice di adiacenza deve essere ridimensionata e riadattata di conseguenza). Ad esempio se si cancella il nodo di indice 2 da un grafo con 5 nodi allora i nodi di indice 0 e 1 rimarranno con lo stesso indice, invece i nodi di indice 3 e 4 dovranno avere riassegnato l'indice 2 e 3, rispettivamente.

Disjoint Sets con Foreste

Una collezione di insiemi disgiunti (disjoint sets) è una collezione di insiemi che hanno a due a due intersezione vuota, cioè un qualsiasi elemento o non appartiene alla collezione o, se appartiene, fa parte di un solo insieme tra quelli disgiunti attualmente esistenti. Se consideriamo una collezione di insiemi disgiunti di interi ad esempio possiamo avere:

- [ ] - la collezione vuota
- [{1}] - una collezione che contiene un solo insieme singoletto
- [{1}, {2}, {3}] - una collezione che contiene tre insiemi singoletto disgiunti
- [{1, 2}, {3}] - una collezione che contiene due insiemi disgiunti
- eccetera

In una collezione di insiemi disgiunti ogni insieme disgiunto ha, in ogni momento, un unico **rappresentante**, che è un elemento dell'insieme. Non è importante chi è il rappresentante, deve solo valere sempre la seguente **regola**: se si chiede il rappresentante di un insieme disgiunto due volte e, tra le due richieste, non è stata fatta nessuna modifica all'insieme stesso allora il rappresentante deve risultare essere lo stesso elemento. Se invece l'insieme viene modificato allora il rappresentante può cambiare.

Ci sono molti modi per rappresentare un insieme disgiunto, dai più semplici ai più complicati. La questione fondamentale che guida alla scelta della struttura dati da usare dipende, come succede spesso, delle operazioni che si vogliono implementare sulla struttura e dalla complessità computazionale che si desidera ottenere per esse. Nel caso delle collezioni di insiemi disgiunti si vogliono ottimizzare le seguenti operazioni:

- `makeSet(x)` - crea un nuovo insieme disgiunto singoletto che contiene il solo elemento `x` e `x` è anche il rappresentante dell'insieme
- `findSet(x)` - restituisce un puntatore all'elemento rappresentante dell'insieme disgiunto che attualmente contiene l'elemento `x`
- `union(x, y)` - unisce i due insiemi disgiunti che attualmente contengono l'elemento `x` e l'elemento `y`. Se `x` e `y` fanno già parte dello stesso insieme disgiunto allora

l'operazione non fa niente. Se `x` e `y` fanno parte di insiemi diversi, diciamo `Sx` ed `Sy`, allora `Sx` ed `Sy` vengono eliminati dalla collezione e si inserisce un nuovo insieme disgiunto `Sxy` che contiene tutti gli elementi di `Sx` e di `Sy`. Il rappresentante del nuovo insieme `Sxy` è uno degli elementi di `Sx` o di `Sy`. L'implementazione può fornire indicazioni su chi sarà il rappresentante di `Sxy` oppure può lasciare il criterio di scelta non specificato.

L'interface `DisjointSets<E>` fornisce le API per queste operazioni ed altre operazioni di base su collezioni di insiemi disgiunti. Il generics `E` rappresenta il tipo generico degli elementi che si vogliono inserire nella collezione di insiemi disgiunti.

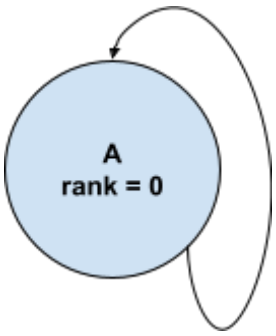
Un esempio di uso delle operazioni principali dove `E` = `Integer` è il seguente:

- all'inizio si parte da una collezione vuota [ ]
- `makeSet(3)` - si ottiene [{3}]
- `makeSet(5)` - si ottiene [{3}, {5}]
- `makeSet(7)` - si ottiene [{3}, {5}, {7}]
- `union(3,7)` - si ottiene [{3, 7}, {5}]
- `makeSet(1)` - si ottiene [{3, 7}, {5}, {1}]
- `makeSet(2)` - si ottiene [{3, 7}, {5}, {1}, {2}]
- `union(1,2)` - si ottiene [{3, 7}, {5}, {1, 2}]
- `union(1,7)` - si ottiene [{3, 7, 1, 2}, {5}]
- `union(3,5)` - si ottiene [{3, 7, 1, 2, 5}]

In questo progetto si deve implementare la collezione di insiemi disgiunti utilizzando **foreste di alberi in cui ogni albero rappresenta uno degli insiemi disgiunti**. In questo tipo di rappresentazione un elemento di un insieme disgiunto è rappresentato come il nodo di un albero che ha i seguenti campi:

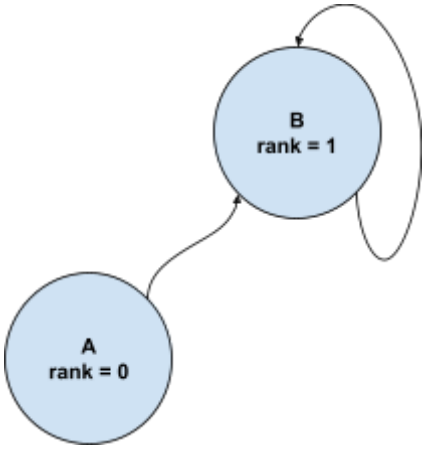
- un puntatore all'elemento di tipo `E`
- un puntatore `parent` al nodo genitore di questo nodo nell'albero di cui l'elemento fa attualmente parte. Nel caso in cui l'elemento è il rappresentante dell'insieme disgiunto allora sarà la radice dell'albero e il puntatore al parent punterà allo stesso nodo: `node.parent == node`
- un intero `rank` che indica il **rango** del nodo. Il rango di un nodo è definito come un limite superiore all'altezza del sottoalbero di cui il nodo è radice. Quindi ad esempio un nodo foglia avrà rango 0 mentre un nodo che ha almeno un nodo figlio avrà rango 1. Comunque, a causa delle euristiche implementate dalle operazioni di `findSet` e `union`, un nodo di rango 4, ad esempio, ad un certo punto potrebbe diventare una foglia. Per questo il rango è un *limite superiore* all'altezza del sottoalbero e non esattamente l'altezza del sottoalbero.

Con questa rappresentazione l'operazione di `makeSet(x)` si riduce a creare un albero-foglia con rango zero e rappresentante di se stesso, in tempo O(1). Graficamente:



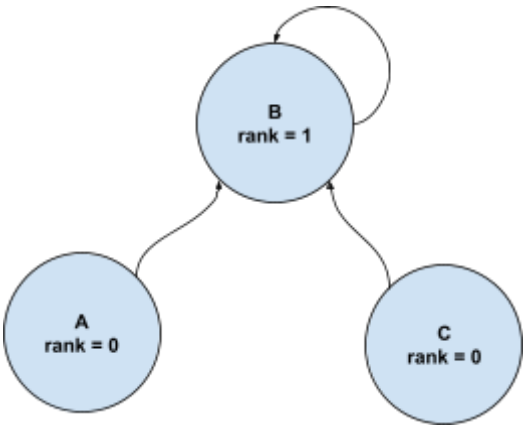
L'operazione di `union(x, y)` deve essere implementata in questo progetto realizzando l'euristica "unione per rango" ovvero se si devono unire due insiemi disgiunti contenenti `x` e `y` si risale ai rappresentanti `Rx` ed `Ry` utilizzando il puntatore `parent` fino a che si arriva alle radici dei rispettivi alberi, dopodiché ci sono due casi:

- le radici `Rx` ed `Ry` hanno lo stesso rango, e in questo caso `Ry` diventerà il rappresentante con rango aumentato di 1. Ad esempio supponiamo di avere due insiemi singoletto {A} e {B}, quindi entrambi albero-foglia con rango 0. L'operazione `union(A, B)` creerà l'albero:

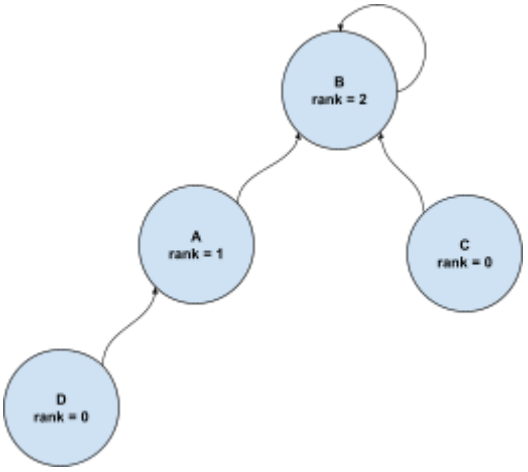


cioè B sarà il nodo radice con rango aumentato di 1 ed A punterà a B come `parent`.

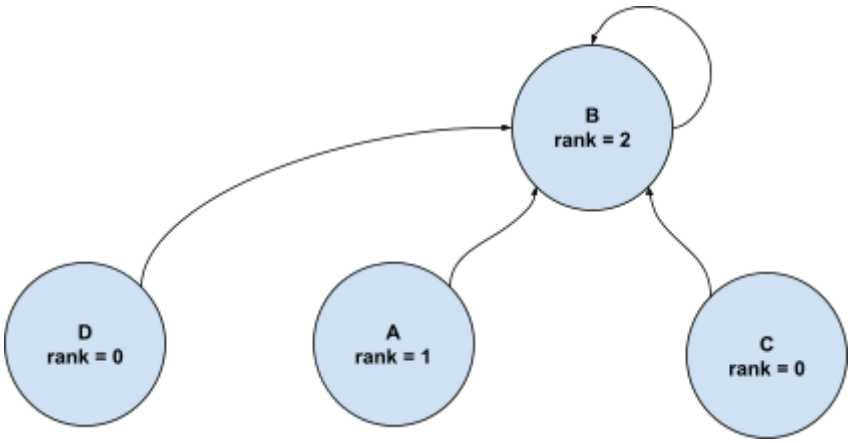
- le radici `Rx` ed `Ry` hanno rango diverso, e quindi la radice con rango maggiore diventerà il rappresentante del nuovo insieme rimanendo con lo stesso rango. Ad esempio se abbiamo due insiemi disgiunti {A,B} come sopra e l'insieme singoletto {C}, l'operazione `union(A, C)` creerà un albero la cui radice sarà B (rappresentante di {A,B}) poiché il rango del nodo B (uguale a 1) è maggiore del rango del nodo C (uguale a 0). Graficamente:



L'operazione di `findSet(x)` in questo progetto deve realizzare l'euristica "compressione del cammino" ovvero, risalendo la catena dei puntatori `parent` deve "appiattire" il più possibile l'albero facendo diventare tutti i nodi visitati figli diretti della radice. Ad esempio se abbiamo un albero come segue:



la chiamata `findSet(D)` dovrà ritornare B, ma avrà come effetto collaterale di cambiare il parent di D e di A con B (tuttavia il parent di A è già B quindi in questo caso il parent di A rimane invariato). Graficamente il nuovo albero dopo la chiamata di `findSet(D)` sarà così:



Si noti che l'aggiornamento dei `parent` durante il `findSet` **non** modifica i ranghi dei nodi.

Abbiamo visto come rappresentare un singolo insieme disgiunto, che va realizzato nella classe `ForestDisjointSets<E>`. Nella classe è già specificata una classe interna `Node<E>` che serve a rappresentare i nodi degli alberi della foresta. Per rappresentare la foresta presente in un certo momento e associare gli elementi presenti ai relativi nodi di tipo `Node<E>` è presente una variabile istanza

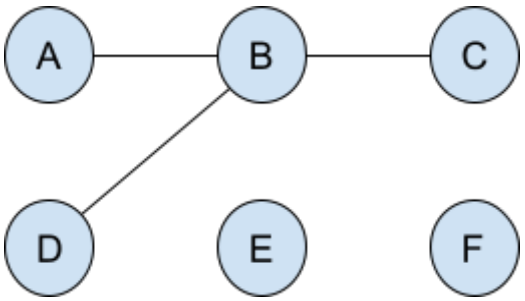
```
protected Map<E, Node<E>> currentElements;
```

Per maggiori dettagli sugli insiemi disgiunti e sulla loro implementazione con foreste si può consultare il Capitolo 21, Sezione 3 del libro di testo

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduzione agli algoritmi 3/ED*. McGraw- Hill, 2010.

Componenti Connesse

In un grafo non orientato si definisce componente connessa un insieme di nodi che sono tutti raggiungibili uno dall'altro seguendo un cammino formato da archi del grafo. Se un grafo ha un'unica componente connessa allora si dice connesso. Ad esempio il grafo



ha tre componenti connesse {A, B, C, D}, {E} ed {F} e quindi non è connesso. Utilizzando le funzionalità della classe `ForestDisjointSet` si deve implementare, nella classe

`UndirectedGraphConnectedComponentsComputer<L>` un semplice algoritmo che calcola tutte le componenti connesse di un grafo non orientato dato. Ci si può ispirare allo pseudocodice contenuto nel Capitolo 21, Sezione 1 del libro di testo

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduzione agli algoritmi 3/ED*. McGraw- Hill, 2010.

Albero minimo ricoprente con algoritmi golosi

Dato un grafo, un albero ricoprente è un sottografo che è un albero e che ha un sottoinsieme degli archi del grafo che permettono di connettere tutti i nodi del grafo con un **unico** cammino. Dato un grafo pesato, un minimo albero ricoprente (minimum spanning tree, MSP) è un albero ricoprente la cui somma dei pesi degli archi è minima.

In questo progetto bisogna implementare due algoritmi golosi per il calcolo dell'albero minimo ricoprente di un grafo non orientato e pesato:

- l'algoritmo di Kruskal, che deve operare utilizzando le funzionalità della classe `ForestDisjointSets<GraphNode<L>>` e deve realizzare un ordinamento degli archi in base al peso; l'algoritmo deve essere implementato all'interno della classe `KruskalMSP<L>` che contiene le relative API;
- l'algoritmo di Prim, che deve implementare una variante della visita in ampiezza utilizzando una coda con priorità (che può essere implementata semplicemente con una `List<GraphNode<L>>`) e un insieme dei nodi già visitati; l'algoritmo deve essere implementato all'interno della classe `PrimMSP<L>` che contiene le relative API;

La decisione su quali strutture dati e algoritmi utilizzare nelle due classi per la gestione dell'ordinamento e della determinazione dell'insieme dei nodi già visitati fa parte dei relativi task e deve essere ispirata ai principi di efficienza in tempo e spazio.

La spiegazione di entrambi gli algoritmi con il relativo pseudocodice si può trovare nel Capitolo 23 del libro di testo

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduzione agli algoritmi 3/ED*. McGraw- Hill, 2010.

Traccia e Implementazione

La traccia del codice è fornita come .zip contenente i template delle seguenti classi/interfacce:

- `it.unicam.cs.asdl2122.mp2.AdjacencyMatrixUndirectedGraph`
- `it.unicam.cs.asdl2122.mp2.AdjacencyMatrixUndirectedGraphTest`
- `it.unicam.cs.asdl2122.mp2.DisjointSets`
- `it.unicam.cs.asdl2122.mp2.ForestDisjointSets`

- `it.unicam.cs.asdl2122.mp2.ForestDisjointSetsTest`
- `it.unicam.cs.asdl2122.mp2.Graph`
- `it.unicam.cs.asdl2122.mp2.GraphEdge`
- `it.unicam.cs.asdl2122.mp2.GraphNode`
- `it.unicam.cs.asdl2122.mp2.KruskalMSP`
- `it.unicam.cs.asdl2122.mp2.KruskalMSPTest`
- `it.unicam.cs.asdl2122.mp2.PrimMSP`
- `it.unicam.cs.asdl2122.mp2.PrimMSPTest`
- `it.unicam.cs.asdl2122.mp2.UndirectedGraphConnectedComponentsComputer`
- `it.unicam.cs.asdl2122.mp2.UndirectedGraphConnectedComponentsComputerTest`

che si possono importare nel proprio IDE preferito. La versione del compilatore Java da definire 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…); più precisamente: **possono essere usati solo i costrutti di Java 5**, la versione 1.8 serve solo per eseguire i test JUnit 5;
- **è obbligatorio usare il set di caratteri utf8** per la codifica dei file del codice sorgente; per far questo impostare tale codifica nelle properties generali del progetto del proprio IDE

Sono fornite le classi di test JUnit che saranno utilizzate per testare il codice consegnato. Quindi è **estremamente importante** leggere bene questo documento e le API scritte nel formato javadoc nei template delle interfacce/classi date. In caso di dubbi si utilizzi il documento google condiviso associato a questo compito denominato "`ASDL2122MP2 Q&A Traccia`" per controllare le risposte a domande già fatte e/o per fare una nuova domanda.

Modalità di Sviluppo e Consegna

Vanno implementati tutti i metodi richiesti (segnalati con commenti della forma `// TODO implementare` nel template delle classi). Nei file consegnati devono rimanere i commenti `// TODO` **solo se** effettivamente quei TODO non sono stati fatti.
Non è consentito:
- aggiungere classi pubbliche;
- modificare la firma (signature) dei metodi già specificati nella traccia;
- modificare le variabili istanza già specificate nella traccia.

E' consentito:
- aggiungere classi interne per fini di implementazione;
- aggiungere metodi privati per fini di implementazione;
- importare classi dai pacchetti della Java SE;

- aggiungere variabili istanza private per fini di implementazione.

Nel file sorgente di ogni classe implementata, nel commento javadoc della classe, modificare il campo @author come indicato:

```
@author Luca Tesei (template) **INSERIRE NOME, COGNOME ED EMAIL
         xxxx@studenti.unicam.it DELLO STUDENTE** (implementazione)
```

Creare una cartella con il seguente nome con le **lettere maiuscole e i trattini** (non underscore!) **esattamente come indicato senza spazi aggiuntivi o altri caratteri**:

ASDL2122-COGNOME-NOME-MP2

ad esempio **ASDL2122-ROSSI-MARIO-MP2**. Nel caso di più nomi/cognomi usare solo il primo cognome e il primo nome. 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 di questa cartella copiare i file sorgenti java modificati con la propria implementazione:

- `AdjacencyMatrixUndirectedGraph.java`
- `ForestDisjointSets.java`
- `KruskalMSP.java`
- `PrimMSP.java`
- `UndirectedGraphConnectedComponentsComputer.java`

che di solito si trovano nella cartella

`cartella-del-progetto/src/it/unicam/cs/asdl2122/mp2`

all'interno del workspace. Non si modifichi il package delle classi/interfacce!
Comprimere la cartella **esclusivamente in formato .zip** (**non rar o altro, solo zip**) e chiamare l'archivio

ASDL2122-COGNOME-NOME-MP2.zip

ATTENZIONE: se i passaggi descritti per la consegna non vengono seguiti **precisamente** (ad esempio nome della cartella sbagliato, contenuto dello zip diverso da quello indicato, formato non zip ecc.) lo studente **perderà automaticamente 3 punti nel voto del miniprogetto**.
Consegnare il file **ASDL2122-COGNOME-NOME-MP2.zip** tramite Google Classroom (usare la funzione consegna associata al post di assegnazione del miniprogetto) entro la data di **scadenza**, cioè **Lunedì 10 Gennaio 2022 ore 18.00**.

Valutazione

ATTENZIONE: Il codice consegnato verrà sottoposto a un **software antiplagio** che lo confronterà con tutti gli altri codici consegnati dagli altri studenti. Il software segnala una percentuale di somiglianza e dei gruppi di studenti con codice probabilmente plagiato.

La valutazione si baserà sui seguenti criteri, in ordine decrescente di importanza:

1. **Codice scritto individualmente. Nel caso di conclamato "plagio" il voto del miniprogetto 2 di tutti i "plagi" verrà decurtato di 10 punti.**
2. **Compilabilità**. Il codice consegnato deve essere in formato **utf8** e deve essere compilabile con il compilatore Java standard **versione 1.8**. In caso contrario ci sarà una penalizzazione in punti del voto finale. In particolare **è vietato l'uso di funzionalità avanzate di Java 8** o superiore, ad esempio lambda-espressioni o stream: possono essere usati solo i costrutti di Java 5 in quanto la versione 1.8 serve solo per eseguire i test JUnit 5.
3. **Testabilità**. Il codice consegnato deve poter essere testato dalle classi JUnit fornite. In particolare, se uno o più test vanno in ciclo e non terminano ciò comporta una penalizzazione.
4. **Correttezza**. Il codice consegnato verrà sottoposto ai test JUnit dati. Il mancato superamento di un test comporterà il decurtamento di un certo numero di punti (a seconda del test, una griglia di valutazione verrà fornita all'atto della riconsegna con il voto). Se una classe o una parte di essa non vengono implementate del tutto allora il punteggio assegnato per quella parte sarà zero e non il punteggio decurtato dal numero di test falliti.
5. **Prova scritta MP2**: le domande non risposte della prova scritta comportano una penalizzazione in punti del voto finale
6. **Prova Scritta MP2**: le domande risposte, ma le cui risposte non sono congruenti con il codice consegnato, comportano una penalizzazione in punti del voto finale
7. **Codice chiaro, leggibile e ben commentato**. Ad esempio un codice in cui non si usano le convenzioni di Java sui nomi delle variabili, classi e metodi (camel) risulta poco leggibile. I commenti dovrebbero essere inseriti in tutti i casi in cui il codice scritto risulta complesso (cicli, if annidati o con condizioni articolate), ma d'altra parte commentare ogni singola riga (ad esempio un assegnamento) risulta inutile e prolisso.
8. Scelta di strutture dati e implementazione **efficienti** sia dal punto di vista del tempo di esecuzione che dello spazio richiesto per la rappresentazione dei dati. Se si deve eseguire un (sotto)task per cui esistono algoritmi/strutture dati che abbiamo esplicitamente visto nel corso e sono ottimi, l'uso di algoritmi/strutture dati che invece sono meno efficienti comporta una penalizzazione. Si tenga inoltre presente che uno spreco enorme di spazio comporta sempre uno spreco enorme di tempo (per scrivere e leggere le variabili in questione) e quindi l'aspetto spazio è sempre da prendere in considerazione nelle scelte che si fanno.

Il voto assegnato al miniprogetto verrà comunicato tramite Google Classroom. Il voto sarà espresso in 31esimi e peserà per il 40% del voto finale ottenuto con le prove parziali per ASDL2122. Il miniprogetto 1 peserà per il 40% e il restante 20% sarà ricavato dalla consegna delle Esercitazioni a Casa (si veda il post "Modalità di Esame ASDL" sul Google Classroom del corso).