Università degli Studi di Camerino
Scuola di Scienze e Tecnologie
Corso di Laurea in Informatica
Corso di Algoritmi e Strutture Dati 2020/2021
Parte di Laboratorio (6 CFU)
Docente: Luca Tesei

Istruzioni per la realizzazione del Miniprogetto 2

ASDL2021MP2

Descrizione

Il miniprogetto ASDL2021MP2 consiste nei seguenti task:

1. Implementare correttamente le classi `MapAdjacentListDirectedGraph<L>` e `AdjacencyMatrixUndirectedGraph<L>` entrambe sottoclassi della classe astratta `Graph<L>`. Le specifiche dettagliate delle classi sono descritte nel template. L'implementazione di queste classi è stata assegnata anche nelle esercitazioni a casa. Si può quindi usare/migliorare il codice consegnato nelle esercitazioni, facendo attenzione però al fatto che qui verrà verificato l'eventuale plagio.
2. Implementare la classe `StronglyConnectedComponentsFinder<L>` assumendo che il grafo passato come parametro al metodo `findStronglyConnectedComponents(Graph<L> g)` sia un grafo orientato, ad esempio uno realizzato con un oggetto della classe `MapAdjacentListDirectedGraph<L>`
3. Implementare la classe `PrimMSP<L>` che calcola un albero minimo di copertura di un grafo non orientato e pesato (con pesi non negativi) utilizzando l'algoritmo goloso di Prim. Si assuma ad esempio che il grafo passato al metodo `computeMSP(Graph<L> g, GraphNode<L> s)` sia un oggetto della classe `AdjacencyMatrixUndirectedGraph<L>`. Per realizzare questa classe è richiesto l'uso della classe `TernaryHeapMinPriorityQueue` che era stata assegnata nel Miniprogetto 1 (ASDL2021MP1) poiché l'algoritmo utilizza una coda di priorità per gestire i nodi durante l'esecuzione. L'interfaccia `PriorityQueueElement` è il ponte che permette di utilizzare la classe del MP1 in quanto la classe `GraphNode<L>` è stata modificata in modo tale che implementi `PriorityQueueElement`. E' possibile riutilizzare il codice che era stato consegnato nel MP1 oppure migliorarlo, se lo si ritiene necessario
4. Implementare la classe `ElMamunCaravanSolver` che risolve il problema della parentesizzazione ottima (minima e massima) di una espressione aritmetica ben

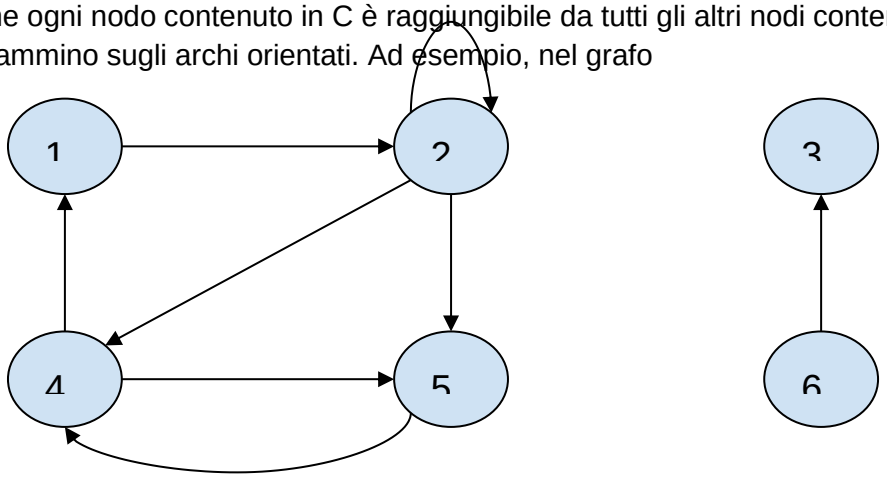
formata contenente solo le cifre da 0 a 9 e gli operatori + e *. La soluzione deve essere calcolata tramite la tecnica della programmazione dinamica.

Rappresentazione di grafi

Sono date già specificate nel template: la classe astratta `Graph<L>` per un generico grafo, la classe `GraphNode<L>` per un generico nodo del grafo e la classe `GraphEdge<L>` per un generico arco. Le due classi da implementare usano delle varianti delle rappresentazioni con liste di adiacenza e matrice di adiacenza. Si noti che esiste una rappresentazione mediante matrice di incidenza che in questo progetto non viene considerata. Le varianti proposte sono descritte in dettaglio nei template delle classi e fanno uso opportuno della classe `ArrayList<E>` e delle interface `Map<K,V>` e `Set<E>` nel pacchetto `java.util`. Le varianti migliorano le prestazioni (in termini di tempo di esecuzione, ma non di spazio occupato) di alcune delle operazioni richieste sul grafo a patto che si usino, come classi che implementano le interface, le classi `HashMap<K,V>` e `HashSet<E>`.

Componenti Fortemente Connesse

In un grafo orientato una componente fortemente connessa è un sottoinsieme C dei nodi del grafo tale che ogni nodo contenuto in C è raggiungibile da tutti gli altri nodi contenuti in C tramite un cammino sugli archi orientati. Ad esempio, nel grafo



ci sono 3 componenti fortemente connesse: C_1 = {1, 2, 4, 5}, C_2 = {3} e C_5 = {6}. Si noti che le componenti C_2 e C_3 sono divise perché da 3 non è possibile raggiungere 6 con nessun cammino.
Esistono diversi algoritmi per calcolare le componenti fortemente connesse. Ad esempio, nella sezione 22.5 del libro di testo

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

è specificato un algoritmo che si basa sulla visita in profondità del grafo e sulla generazione del grafo trasposto. Si consiglia di implementare questo algoritmo, ma se se si vuole utilizzarne un altro è necessario fornire i riferimenti e le spiegazioni necessarie nei commenti del codice.

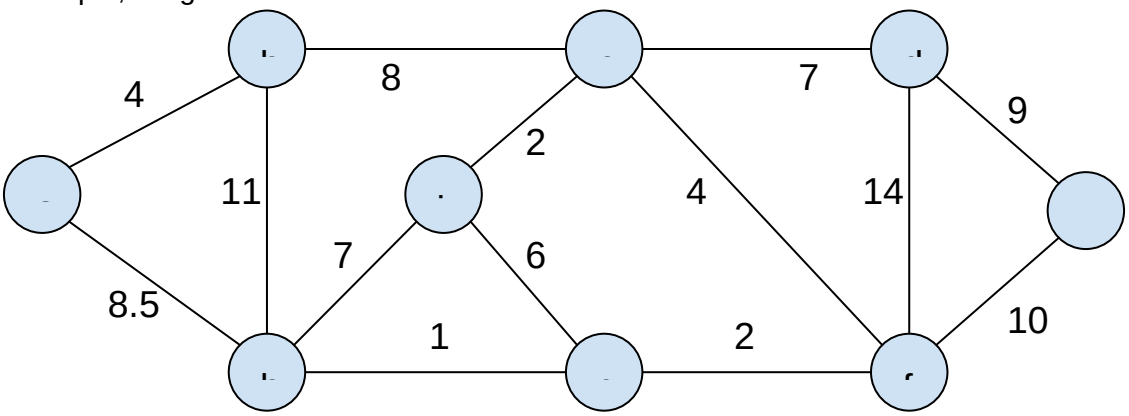
Algoritmo di Prim per Albero Minimo di Copertura

Dato un grafo non orientato e pesato (con pesi non negativi), l'algoritmo di Prim, specificato nella sezione 23.2 del libro di testo

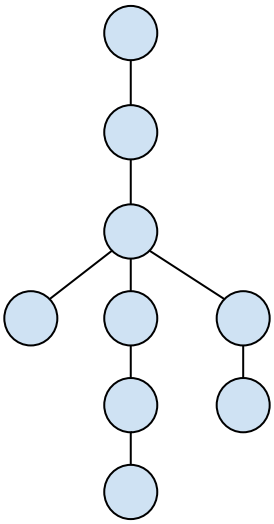
T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduzione agli algoritmi 3/ED*. McGraw- Hill, 2010.

è un algoritmo goloso che trova un albero minimo di copertura del grafo a partire da un certo nodo sorgente. Esso opera in maniera simile all'algoritmo di Dijkstra per il calcolo dei cammini minimi con sorgente singola.

Ad esempio, sul grafo



l'albero di copertura minimo calcolato dall'algoritmo di Prim a partire dal nodo sorgente "a" è il seguente:



per un costo totale di 4 + 8 + 7 + 9 + 2 + 4 + 2 + 1 = 37.

Nell'algoritmo di Prim, come in quello di Dijkstra, è importante che i nodi del grafo ancora da visitare siano mantenuti in una coda di priorità in modo da rendere efficiente sia l'estrazione del minimo sia l'operazione di `decreasePriority`. A tal scopo la classe `GraphNode<L>` è stata modificata facendole implementare l'interface `PriorityQueueElement` e per la coda di min priorità viene richiesto di usare la propria implementazione (o un suo miglioramento se necessario) della classe `TernaryHeapMinPriorityQueue` implementata nel Miniprogetto 1 ASDL2021MP1.
Come specificato nel template, il risultato del codice implementato sarà testato analizzando i nodi del grafo dopo l'esecuzione dell'algoritmo utilizzando il campo `previous` dei nodi per determinare l'albero minimo di copertura calcolato. Naturalmente il campo `previous` del nodo sorgente dovrà essere `null` (e dovrà essere l'unico `null`).

Calif El Mamun's Caravan Problem

Si tratta di un problema che riguarda la parentesizzazione di espressioni aritmetiche, simile a quello della parentesizzazione del prodotto di matrici. Per semplificare consideriamo espressioni aritmetiche in cui gli operatori sono solo + e * e gli operandi possono essere solo le cifre da 0 a 9. L'espressione deve essere ben formata e non deve contenere parentesi né spazi. Esempi: 1, 1+3, 2+3*4, 4*3+0*3+2*1+3*9, eccetera.

Prendiamo come esempio l'espressione 1+2*3*4+5 e consideriamo quindi il problema di trovare una parentesizzazione che massimizza il risultato dell'espressione e una parentesizzazione che lo minimizza. La parentesizzazione ((1+2)*(3*(4+5))) dà come risultato 81 e si può dimostrare che è il massimo valore che si può ottenere considerando tutte le possibili parentesizzazioni. Di converso, la parentesizzazione (1+((2*(3*4))+5)) dà come risultato 30 ed è il valore minimo ottenibile.

Il problema si può formalizzare come segue. Consideriamo una espressione e lunga N i cui elementi sono quindi e[0], e[1], … , e[N-1]. Se l'espressione è ben formata allora e[0] ed e[N-1] saranno cifre, e[1] sarà un operatore, e[2] sarà una cifra e così via. Per scomporre il problema in sottoproblemi consideriamo un generico sottoproblema della forma m[i,j] dove 0 <= i <= j < N, e[i] è una cifra ed e[j] è una cifra. Il valore di m[i,j] è definito come il migliore (massimo o minimo) valore che si può ottenere considerando tutte le possibili parentesizzazioni dell'espressione e[i] e[i+1] … e[j]. Quindi m[0,N-1] rappresenta la soluzione del problema principale. Vale la seguente ricorrenza per 0 <= i <= j < N:

- se i == j ed e[i] è una cifra allora m[i,j] = e[i]
- se i < j ed e[i], e[j] sono cifre allora m[i,j] = *best*(m[i,i+k] e[i+k+1] m[i+k+2,j]) dove k = 0, 2, 4, 6, … per tutti i valori che rispettano i + k + 2 < j, e[i+k+1] è l'operatore + o l'operatore * e la funzione *best* è una funzione obiettivo che calcola, facendo variare k nei valori consentiti, il valore migliore (massimo o minimo) che si ottiene utilizzando l'operatore e[i+k+1] tra i due operandi m[i,i+k] e m[i+k+2,j]. k rappresenta quindi il punto in cui si spezza l'espressione ottenendo due sottoespressioni separate dall'operatore e[i+k+1]. Quando k è il valore selezionato dalla funzione *best* si ha una parentesizzazione ottima dell'espressione e[i]...e[j] della forma
  "(x e[i+k+1] y)" dove x è una parentesizzazione ottima della sottoespressione e[i]...e[i+k] e y è una parentesizzazione ottima della sottoespressione e[i+k+2]…e[j]

- i casi in cui e[i] ed e[j] non sono cifre non ci interessano e i relativi valori di m potranno essere considerati nulli senza produrre errori; lo stesso dicasi per i casi in cui j < i.

Per risolvere il problema con la programmazione dinamica rappresentiamo la funzione m con una matrice NxN e calcoliamo i valori per i <= j in maniera bottom-up utilizzando un ordine opportuno di riempimento delle caselle. La soluzione del problema principale si troverà quindi in m[0,N-1]. Per ricordare la scelta effettuata dalla funzione *best* durante il calcolo, e quindi ricostruire la parentesizzazione ottima, utilizziamo una matrice accessoria b[i,j] dove inseriamo via via i valori di k sui quali la funzione *best* ha dato come risultato il valore ottimo per il sottoproblema m[i,j].

Sono fornite già pronte nel template le classi `Expression`, `ExpressionItem` e `ItemType` che si occupano di fare il parsing di una stringa che contiene l'espressione e verificano che l'espressione sia ben formata. L'interface `ObjectiveFunction` rappresenta una generica funzione *best* che sceglie il migliore in una lista di candidati (in questo caso numeri interi) e, per le finalità di traceback, dà anche l'indice del migliore che è stato selezionato. Le due classi `MaximumFunction` e `MinimumFunction` (anch'esse già fornite) implementano le due diverse `ObjectiveFunction` che ci servono (una calcola il massimo valore e l'altra il minimo). La classe `ElMamunCaravanSolver` va implementata seguendo la formalizzazione data sopra e utilizzando la programmazione dinamica. In particolare la matrice m corrisponde alla variabile istanza `table` e la matrice b corrisponde alla variabile istanza `tracebackTable`.

Il codice seguente mostra come usare le classi fornite e quella da implementare:

```
…
// create the two objective functions
ObjectiveFunction max = new MaximumFunction();
ObjectiveFunction min = new MinimumFunction();
// create the expression
Expression f = new Expression("1+2*3*4+5");
// create the solver
ElMamunCaravanSolver solver = new ElMamunCaravanSolver(f);
// solve using the max objective function
solver.solve(max);
// print results
System.out.println(solver.getOptimalSolution()(;
// prints "81"
System.out.println(solver.getOptimalParenthesization());
// prints "((1+2)*(3*(4+5)))"
// solve using the min objective function
solver.solve(min);
// print results
System.out.println(solver.getOptimalSolution()(;
// prints "30"
System.out.println(solver.getOptimalParenthesization());
// prints "(1+((2*(3*4))+5))"
```

Traccia e Implementazione

Il template del codice è fornito come .zip contenente le seguenti classi/interfacce:

- Task 1:
  - `it.unicam.cs.asdl2021.mp2.Graph`
  - `it.unicam.cs.asdl2021.mp2.GraphEdge`
  - `it.unicam.cs.asdl2021.mp2.GraphNode`
  - `it.unicam.cs.asdl2021.mp2.MapAdjacentListDirectedGraph` - Da implementare
  - `it.unicam.cs.asdl2021.mp2.AdjacencyMatrixUndirectedGraph` - Da implementare
- Task 2:
  - `it.unicam.cs.asdl2021.mp2.StronglyConnectedComponentsFinder` - Da implementare
- Task 3:
  - `it.unicam.cs.asdl2021.mp2.PriorityQueueElement`
  - `it.unicam.cs.asdl2021.mp2.TernaryHeapMinPriorityQueue` - Da implementare
  - `it.unicam.cs.asdl2021.mp2.PrimMSP` - Da implementare
- Task 4:
  - `it.unicam.cs.asdl2021.mp2.Expression`
  - `it.unicam.cs.asdl2021.mp2.ExpressionItem`
  - `it.unicam.cs.asdl2021.mp2.ItemType`
  - `it.unicam.cs.asdl2021.mp2.ObjectiveFunction`
  - `it.unicam.cs.asdl2021.mp2.MinimumFunction`
  - `it.unicam.cs.asdl2021.mp2.MaximumFunction`
  - `it.unicam.cs.asdl2021.mp2.ElMamunCaravanSolver` - Da implementare

I file si possono importare in Eclipse o nel proprio IDE preferito. La versione del compilatore Java da definire nel progetto Eclipse (o altro IDE) 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.

Non 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 classi in modo da implementare i metodi richiesti nella maniera corretta. In caso di dubbi si utilizzi il documento google condiviso associato a questo compito denominato "`ASDL2021MP2 Q&A Traccia`" per controllare le risposte a domande già fatte e/o per fare una nuova domanda.

La scrittura di propri test JUnit per controllare che la propria implementazione funzioni bene è fortemente consigliata. Tuttavia tali test non dovranno essere consegnati

come parte del progetto implementato, saranno solo a garanzia personale di correttezza del codice.

Modalità di Sviluppo e Consegna

Vanno implementati tutti i metodi richiesti (segnalati con commenti della forma `// TODO implementare` nel template delle classi). 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 private per fini di implementazione;
- aggiungere metodi privati per fini di implementazione;
- 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: "INSERIRE NOME E COGNOME DELLO STUDENTE - INSERIRE ANCHE L'EMAIL xxxx@studenti.unicam.it".

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**:

ASDL2021-COGNOME-NOME-MP2

ad esempio **ASDL2021-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:

- `MapAdjacentListDirectedGraph.java`
- `AdjacencyMatrixUndirectedGraph.java`
- `StronglyConnectedComponentsFinder.java`
- `TernaryHeapMinPriorityQueue.java`
- `PrimMSP.java`
- `ElMamunCaravanSolver.java`

che di solito si trovano nella cartella

```
cartella-del-progetto/src/it/unicam/cs/asdl2021/mp2
```

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

ASDL2021-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 **ASDL2021-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 18 Gennaio 2021 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 miniproject 2 di tutti i "plagi" verrà decurtato di 8 punti.**
2. **Correttezza**. Il codice consegnato verrà sottoposto a dei test JUnit che controlleranno tutte le funzionalità. Tali test verranno resi pubblici dopo la data di scadenza per la consegna in modo da poter essere eseguiti per vedere eventuali errori commessi. Sarà possibile chiedere spiegazioni e/o chiarimenti riguardo ai test. 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).
3. **Codice chiaro, leggibile e ben commentato**.
4. Scelta di strutture dati e implementazione **efficienti** sia dal punto di vista del tempo di esecuzione che dello spazio richiesto.

Il voto assegnato al miniprogetto verrà comunicato tramite Google Classroom. Il voto sarà espresso in 30esimi e peserà per il 40% del voto finale ottenuto con le prove parziali per ASDL2021. Il miniprogetto 1 pesa già per il 40% e il restante 20% è ricavato dalla consegna delle Esercitazioni a Casa (si veda il documento Google condiviso "`ASDL2021 Prove Parziali`" per tutti i dettagli).