ASD 24/25 - Laboratorio 10 Individuale

1 Introduzione

In questo laboratorio ci proponiamo di progettare e implementare una struttura dati a supporto della gestione dei processi produttivi di un'industria semplificata. L'industria è in grado di fabbricare oggetti, detti *item*, i quali possono essere di due tipi:

- **Basic item**: oggetti elementari, disponibili inizialmente in quantità finita, che non dipendono da altri item per essere ottenuti;
- **Item composti**: oggetti che possono essere prodotti solo a partire da uno o più altri item (basic o composti), secondo una specifica relazione di dipendenza.

Nella struttura dati vengono registrati gli item che l'industria può produrre e, per ciascuno, l'elenco degli altri item necessari alla sua fabbricazione. Il sistema di dipendenze tra oggetti può essere rappresentato come un grafo orientato aciclico (DAG), dove un arco da un item A a un item B indica che B è necessario per produrre A.

Per comprendere meglio il contesto, consideriamo un esempio in cui l'industria può produrre i seguenti quattro item: o_1, o_2, o_3 e o_4, con le seguenti dipendenze:

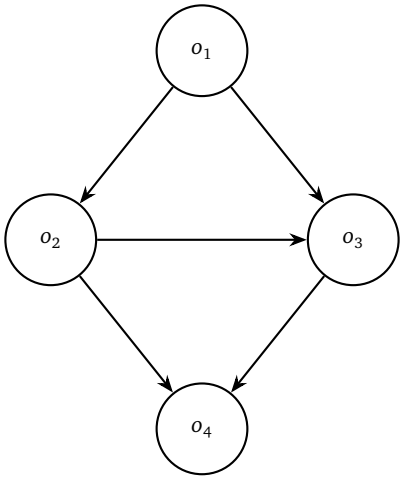
- Per fabbricare o_2 è necessario o_1;
- Per fabbricare o_3 sono necessari o_2 e o_1;
- Per fabbricare o_4 sono necessari o_3 e o_2.

In questo esempio, o_1 è l'unico *basic item*, mentre tutti gli altri dipendono (direttamente o indirettamente) da esso. Vogliamo, in particolare, realizzare una funzione che — data la quantità disponibile di ogni basic item — calcoli quante unità di un item qualsiasi possono essere effettivamente prodotte. Si assume che, per ogni relazione di dipendenza, sia richiesto esattamente un esemplare di ciascun item componente per fabbricare un esemplare del prodotto finale.

Nel caso dell'esempio, se si dispone di un certo numero di o_1, è possibile determinare quante unità si possono ottenere degli altri item. Di seguito, alcune casistiche illustrative:

- con 1 unità di o_1 si può produrre 1 unità di o_2;
- con 2 unità di o_1 si possono produrre 2 unità di o_2, oppure 1 unità di o_3 (1 per o_1 diretto e 1 per produrre o_2);
- con 3 unità di o_1 si possono produrre: 3 unità di o_2, oppure 1 unità di o_3, oppure 1 unità di o_4 (2 unità per o_3, 1 per o_2);
- con 4 unità di o_1 si possono produrre 4 unità di o_2, oppure 2 unità di o_3, oppure 1 unità di o_4;
- con 5 unità di o_1 si possono produrre 5 unità di o_2, oppure 2 unità di o_3, oppure 1 unità di o_4;
- con 6 unità di o_1 si possono produrre 6 unità di o_2, oppure 3 unità di o_3, oppure 2 unità di o_4;
- ecc.

Le relazioni di dipendenza tra gli item descritti nell'esempio precedente possono essere rappresentate mediante il seguente grafo orientato aciclico (DAG). In questo grafo, ciascun nodo rappresenta un item, e un arco diretto da un nodo A a un nodo B indica che l'item A necessita dell'item B per essere prodotto.



2 Obiettivo

L'obiettivo del laboratorio individuale è implementare le funzionalità di base di una struttura dati che modella il funzionamento di un'industria semplificata. L'industria è in grado di produrre oggetti (detti *item*), alcuni dei quali sono *basic item*, ovvero elementi di base che non dipendono da altri. Gli altri item possono essere prodotti solo se sono disponibili le relative componenti. Lo studente dovrà progettare e realizzare il codice C++ per:

- **Creare una nuova industria** inizialmente vuota.
- **Inserire nuovi item**:
  - *Basic item*, cioè oggetti senza dipendenze;
  - *Item composti*, definiti a partire da un insieme non vuoto di altri item (le componenti).
- **Verificare la presenza di un item** dato il suo nome.
- **Rimuovere un item**, eliminando ricorsivamente anche tutti gli item che ne dipendono direttamente o indirettamente.
- **Aggiornare la quantità di un basic item**, sommando (o sottraendo) un valore intero specificato. La quantità non può mai diventare negativa.
- **Ottenere le dipendenze dirette** di un item: data la descrizione dell'industria, è possibile ottenere la lista degli item necessari per produrre un certo item.
- **Ottenere gli item che dipendono** da un dato item, sia in modo diretto (immediato) sia lungo una catena di dipendenze.
- **Calcolare quante unità di un certo item** è possibile produrre, date le quantità attualmente disponibili dei basic item.

3 Dettaglio delle Funzioni da Implementare in C++

Un main per fare tests automatici vi sarà fornito, voi dovrete quindi progettare e implementare il codice delle funzioni presenti nel file `industry.cpp` che troverete, assieme ad altri file, all'interno del file .zip scaricabile da Aulaweb nella sezione relativa al laboratorio 10. Più in dettaglio le funzioni da implementare sono le seguenti dichiarate nel file `industry.h`.
Su aulaweb troverete degli esempi di chiamate di questa funzione per capire meglio cosa devono fare

```
#ifndef INDUSTRY_H
#define INDUSTRY_H

#include <cstddef>
#include <string>
#include "list-array.h"

namespace industry {
  struct st_Industry;
  typedef st_Industry* Industry;

  /*****************************************************************/
  /*                          Funzioni da implementare             */
  /*****************************************************************/

  // Crea e restituisce un'istanza vuota di Industry.
  Industry createEmptyIndustry();

  // Inserisce un nuovo basic item di nome 'name' nell'industria.
  // Se esiste gia' un item con quel nome, la funzione restituisce false e non fa nulla.
  // Altrimenti inserisce l'item e restituisce true.
  // Si assume che, quando viene inserito un basic item, la quantita iniziale sia 0.
  bool insertBasicItem(Industry& indus, std::string name);

  // Inserisce un nuovo item di nome 'name' nell'industria.
  // 'components' e' un array NON VUOTO di lunghezza 's' che contiene i nomi degli item
  // da cui dipende il nuovo item.
  // Se esiste gia' un item con quel nome, la funzione restituisce false e non fa nulla.
  // Se uno qualsiasi degli item indicati in 'components' non esiste nell'industria,
  // la funzione restituisce false e non fa nulla.
  // Altrimenti inserisce l'item e restituisce true.
  bool insertItem(Industry& indus, std::string name, std::string* components, size_t s);

  // Restituisce true se esiste un item con il nome 'name' nell'industria, false altrimenti.
```

```
  bool isPresentItem(const Industry& indus, std::string name);

  // Rimuove l'item di nome 'name' dall'industria.
  // Se esiste almeno un altro item che dipende direttamente o indirettamente da 'name',
  // verra' rimosso anche quello.
  // Se non esiste un item con quel nome, la funzione restituisce false e non fa nulla.
  // Altrimenti, rimuove l'item (e quelli dipendenti) e restituisce true.
  bool removeItem(Industry& indus, std::string name);

  // Aumenta o diminuisce la quantita dell'item di base di nome 'name' di un valore 'v'.
  // Se 'v' e' negativo e la quantita corrente e' Q, la nuova quantita sara' max(Q + v, 0).
  // La quantita non puo' mai diventare negativa.
  // Se non esiste un basic item con quel nome, la funzione restituisce false e non fa nulla.
  // Altrimenti restituisce true.
  bool addBasicItem(Industry& indus, std::string name, int v);

  // Riempie la lista 'lres' (in ordine lessicografico) con i nomi degli item
  // da cui l'item di nome 'name' dipende direttamente.
  // Se l'item non esiste, la funzione restituisce false e imposta 'lres' a nullptr.
  // Altrimenti restituisce true.
  bool listNeed(const Industry& indus, std::string name, list::List& lres);

  // Riempie la lista 'lres' (in ordine lessicografico) con i nomi degli item
  // che dipendono direttamente dall'item di nome 'name'.
  // Se l'item non esiste, la funzione restituisce false e imposta 'lres' a nullptr.
  // Altrimenti restituisce true.
  bool listNeededBy(const Industry& indus, std::string name, list::List& lres);

  // Riempie la lista 'lres' (in ordine lessicografico) con i nomi degli item
  // che dipendono (direttamente o indirettamente) dall'item di nome 'name'.
  // Esempio: se o1 dipende da o2 e o2 dipende da o3,
  // allora listNeededByChain("o3") restituira' o2 e o1.
  // Se invece si usa listNeededBy("o3"), la lista non includera' o1
  // perche' non dipende direttamente da o3.
  // Se l'item non esiste, la funzione restituisce false e imposta 'lres' a nullptr.
  // Altrimenti restituisce true.
  bool listNeededByChain(const Industry& indus, std::string name, list::List& lres);

  // Calcola e memorizza in 'res' il numero massimo di item di nome 'name'
  // che si possono costruire con le quantita attualmente disponibili dei basic item.
  // Se l'item non esiste, la funzione restituisce false e imposta 'res' a 0.
  // Altrimenti restituisce true.
  bool howManyItem(const Industry& indus, std::string name, unsigned& res);
}

#endif
```

E' possibile chiaramente inserire altre funzioni ausiliarie utili per l'implementazione.
Come potete notare alcune funzioni fanno uso di Liste. Per questo motivo vi forniamo un'implementazione già completa delle liste nei file `list-array.cpp` e `list-array.h`.

4 Tests manuali e automatici

Potete iniziare ad implementare il laboratorio realizzando inoltre un vostro main per testare manualmente il funzionamento della varie funzioni.
In seguito, vi forniremo un insieme di test automatici per controllare parzialmente la validità delle vostre funzioni.

5 Consegna

Per la consegna, creare uno `zip` con tutti i file forniti. In particolare:

- Tutti i file necessari a compilare il programma
- Una breve relazione (formato txt, doc o pdf) in cui vengono descritte le scelte implementative effettuate. Includete nella relazione anche il comando usato per compilare il vostro codice.

6 Punti e Valutazione

Il punteggio massimo di quattro punti si raggiunge non solo se le funzioni e le strutture dati sono corrette, ma anche se sono implementate in modo efficiente e non ci sono problemi “di stile”. In particolare, sono elementi apprezzati ai fini della valutazione (l’elenco non è esaustivo):

- corretta indentazione del codice
- uso di identificatori significativi
- introduzione di funzioni ausiliarie quando appropriato
- commenti significativi (non devono essere "traduzioni" del codice in lingua corrente, ma spiegazioni ad alto livello dell’algoritmo eseguito e del motivo per cui avete fatto alcune scelte nell’implementazione).

La consegna è obbligatoria solo per coloro che hanno sottoscritto il patto.
Consegnare entro la scadenza riportata su Aulaweb.