Laboratorio per il corso di Algoritmi e Strutture Dati

Nel seguito verranno descritti i tre esercizi da consegnare al fine di potere superare la prova di laboratorio per il corso di Algoritmi e Strutture Dati.

Durante la scrittura del codice è richiesto di usare in modo appropriato il sistema di versioning `Git`. Questa richiesta implica quanto segue:

- il progetto di laboratorio va inizializzato "clonando" il repository del laboratorio come descritto nel file Git;
- come è prassi nei moderni ambienti di sviluppo, è richiesto di effettuare commit frequenti. L'ideale è un commit per ogni blocco di lavoro terminato (es. creazione e test di una nuova funzione, soluzione di un baco, creazione di una nuova interfaccia, ...);
- ogni membro del gruppo dovrebbe effettuare il commit delle modifiche che lo hanno visto come principale sviluppatore;
- al termine del lavoro si dovrà consegnare l'intero repository.

Il file `Git.md` contiene un esempio di come usare Git per lo sviluppo degli esercizi proposti per questo laboratorio.

Linguaggio in cui sviluppare il laboratorio

È lasciata libertà allo studente di implementare il codice usando Java o C. Come potrete verificare gli esercizi chiedono di realizzare strutture generiche. Seguono alcuni suggerimenti circa come realizzarle nei due linguaggi accettati.

Nota importante: Con "strutture dati generiche" si fa riferimento al fatto che le strutture dati realizzate devono poter essere utilizzate con tipi di dato non noti a tempo di compilazione. Sebbene in Java la soluzione più in linea con il moderno utilizzo del linguaggio richiederebbe la creazione di classi parametriche, tutte le scelte implementative (compresa la decisione di usare o meno classi parametriche) sono lasciate agli studenti.

Suggerimenti (C): Nel caso del C è necessario capire come meglio approssimare l'idea di strutture generiche utilizzando quanto permesso dal linguaggio. Un approccio comune è far sì che le funzioni che manipolano le strutture dati prendano in input puntatori a void e utilizzino qualche funzione fornita dall'utente per accedere alle componenti necessarie. *Nota*: chi è in grado di realizzare tipi di dato astratto tramite tipi opachi è incoraggiato a procedere in questa direzione.

Suggerimenti (Java): Nel caso si scelga di utilizzare Java è possibile (e consigliato) usare gli ArrayList invece degli array nativi al fine di semplificare l'implementazione delle strutture generiche.

Uso di librerie esterne e/o native del linguaggio scelto

A parte gli ArrayList è vietato usare altre strutture dati *di base* offerte dal linguaggio in uso (es. code, liste, stack, ...). È lecito (ma non obbligatorio) avvalersi di strutture dati più complesse quando la loro realizzazione non è richiesta da uno degli esercizi proposti. Ad esempio è lecito utilizzare una libreria che implementa le code con priorità, ma non una che implementa i dizionari (perché questi ultimi sono oggetto dell'Esercizio 2).

Qualità dell'implementazione

È parte del mandato degli esercizi la realizzazione di codice di buona qualità. Per "buona qualità" intendiamo codice ben modularizzato, ben commentato e ben testato.

Alcuni suggerimenti:

- verificare che il codice sia suddiviso correttamente in package o moduli;
- aggiungere un commento, prima di una definizione, che spiega il funzionamento dell'oggetto definito. Evitare quando possibile di commentare direttamente il codice in sé (se il codice è ben scritto, i commenti in genere non servono);
- la lunghezza di un metodo/funzione è in genere un campanello di allarme: se essa cresce troppo, probabilmente è necessario rifattorizzare il codice spezzando la funzione in più parti. In linea di massima si può consigliare di intervenire quando la funzione cresce sopra le 30 righe (considerando anche commenti e spazi bianchi);
- sono accettabili commenti in italiano, sebbene siano preferibili in inglese;
- tutti i nomi (e.g., nomi di variabili, di metodi, di classi, etc.) devono essere significativi e in inglese;
- il codice deve essere correttamente indentato; impostare l'indentazione a 2 caratteri (un'indentazione di 4 caratteri è ammessa ma scoraggiata) e impostare l'editor in modo che inserisca "soft tabs" (i.e., deve inserire il numero corretto di spazi invece che un carattere di tabulazione).
- per dare i nomi agli identificatori, seguire le convenzioni in uso per il linguaggio scelto:
  - Java: i nomi dei package sono tutti in minuscolo senza separazione fra le parole; i nomi dei tipi (classi, interfacce, ecc.) iniziano con una lettera maiuscola e proseguono in camel case (es. TheClass), i nomi dei metodi e delle variabili iniziano con una lettera minuscola e proseguono in camel case (es. theMethod), i nomi delle costanti sono tutti in maiuscolo e in formato snake case (es. THE_CONSTANT);
  - C: macro e costanti sono tutti in maiuscolo e in formato snake case (es. THE_MACRO, THE_CONSTANT); i nomi di tipo (e.g. struct, typedefs, enums, ...) iniziano con una lettera maiuscola e proseguono in camel case (e.g., TheType, TheStruct); i nomi di funzione iniziano con una lettera minuscola e proseguono in snake case (e.g., the_function());
- i file vanno salvati in formato UTF8.

Esercizio 1

Implementare i seguenti algoritmi di ordinamento:

- Insertion Sort

- Quick Sort
- Merge Sort
- Heap Sort

Ogni algoritmo va implementato in modo tale da poter essere utilizzato su un generico tipo `T`. L'implementazione degli algoritmi deve permettere di specificare il criterio secondo cui ordinare i dati. *Suggerimento*: Usare l'interfaccia java.util.Comparator (o, nel caso di una implementazione C, un puntatore a funzione).

Unit testing

Implementare gli unit-test degli algoritmi secondo le indicazioni suggerite nel documento Unit Testing.

Confronti

Il file `records.csv` che potete trovare seguendo il path `/usr/NFS/Linux/labalgoritmi/datasets/` (in laboratorio von Neumann, selezionare il disco Y) contiene 20 milioni di record da ordinare. Ogni record è descritto su una riga e contiene i seguenti campi:

- id: (tipo intero) identificatore univoco del record;
- field1: (tipo stringa) contiene parole estratte dalla divina commedia, potete assumere che i valori non contengano spazi o virgole;
- field2: (tipo intero);
- field3: (tipo floating point);

Il formato è un CSV standard: i campi sono separati da virgole; i record sono separati da "\n".

Usando ciascuno degli algoritmi implementati, si ordinino i dati contenuti nel file `records.csv` in ordine non decrescente secondo i valori contenuti nei tre campi "field" (i.e., per ogni algoritmo è necessario ripetere l'ordinamento tre volte, una volta per ciascun campo).

Si misurino i tempi di risposta e si crei una breve relazione in cui si riportano i risultati ottenuti insieme a un loro commento. Nel caso l'ordinamento si protragga per più di 10 minuti potete interrompere l'esecuzione e riportare un fallimento dell'operazione. I risultati sono quelli che vi sareste aspettati? Se sì, perché? Se no, fate delle ipotesi circa il motivo per cui gli algoritmi non funzionano come vi aspettate, verificatele e riportate quanto scoperto nella relazione. *Opzionale:* avete qualche idea per risolvere gli eventuali problemi riscontrati? Se sì implementatela e verificate se e come la situazione migliora.

Esercizio 2

Si consideri la definizione del tipo di dato astratto Dizionario. Si implementi il tipo di dato astratto utilizzando gli alberi di ricerca "semplici" e almeno una tra le seguenti strutture dati:

- Alberi Rosso Neri

- Tabelle Hash

Ogni implementazione deve permettere di usare tipi generici sia per il tipo chiave, sia per il tipo valore.

Unit Testing

Implementare gli unit-test degli algoritmi secondo le indicazioni suggerite nel documento Unit Testing.

Confronti

Si inseriscano i dati contenuti nel file `records.csv` nei dizionari implementati. Si ripeta questa operazione usando come chiave i tre campi presenti nel file (in tutti i casi il "valore" è l'intero record). Nel caso un record abbia chiave uguale a uno già presente è corretto sostituire l'istanza precedentemente memorizzata con il nuovo valore.

Si misurino i tempi di risposta in modo da poter rispondere alle seguenti domande:

- Qual'è il tempo di inserimento dei 20.000.000 di record nei due container?
- Una volta caricati i record nei container, misurate il tempo di accesso a 1.000.000 di chiavi scelte a caso. Che differenza c'è nei tempi di risposta dei due container realizzati?

Rispondete alla domanda precedente cancellando i record invece di accedervi.

Si crei una breve relazione a commento dei risultati ottenuti. I risultati sono quelli che vi sareste aspettati? Se sì, perché? Se no, fate delle ipotesi circa il motivo per cui gli algoritmi non funzionano come vi aspettate e verificatele.

Esercizio 3

Si implementi la struttura dati Grafo diretto in modo che sia ottimale per dati sparsi.

Il file `italian_dist_graph.csv` che potete recuperare seguendo il path `/usr/NFS/Linux/labalgoritmi/datasets/` (in laboratorio von Neumann, selezionare il disco Y) contiene le distanze in metri tra varie località italiane e una frazione delle località a loro più vicine. Il formato è un CSV standard: i campi sono separati da virgole; i record sono separati da "\n".

Ogni record contiene i seguenti dati:

- località 1: (tipo stringa) nome della località "sorgente". La stringa può contenere spazi, non può contenere virgole;
- località 2: (tipo stringa) nome della località "destinazione". La stringa può contenere spazi, non può contenere virgole;
- distanza: (tipo float) distanza in metri tra le due località.

Note:

- potete intrepretare le informazioni presenti nelle righe del file come archi **non diretti** (i.e., probabilmente vorrete inserire nel vostro grafo sia l'arco di andata che quello di ritorno a fronte di ogni riga letta).
- il file è stato creato a partire da un dataset poco accurato. I dati riportati contengono inesattezze e imprecisioni.

Unit Testing

Implementare gli unit-test degli algoritmi secondo le indicazioni suggerite nel documento Unit Testing.

Analisi grafo e cammini minimi

Scrivere un programma per rispondere a ciascuno dei seguenti problemi:

- trovare il cammino minimo tra due località; si assuma che gli archi riportati nel file `italian_dist_graph.csv` siano connessioni percorribili;
- determinare la dimensione di ogni componente fortemente connessa del grafo.

Controlli

Una implementazione corretta della ricerca del cammino minimo dovrebbe riportare un cammino minimo tra "torino" e "catania" lungo ~1207.68Km.

Qual è il cammino minimo tra "torino" e "borsoi"?