Cagnasso Pietro
mat. 256604
Lab. 11 Es. 01

1 Strutture dati

La soluzione dell' esercizio dsi basa su 5 moduli di cui 2 quasi-ADT e 3 ADT di prima classe:

- `datetime` contiene due struct, `Date` e `Time` , per salvare in variabili intere i numeri identificativi di giorno, mese, anno e minuti, ore
- `dailystock` contiene una struttura per salvare le valutazioni giornaliere di un titolo, viene fornita una funzione di aggiornamento dei dati ontenuti nella struttra nel caso si leggano da file transazioni dello stesso titolo con la stessa data
- `stockbst` è l'implementazione di un **albero binario di ricerca** come ADT di prima classe i cui nodi sono istanze del modulo `dailystock`, le funzioni fornite sono tutte standard a parte la visita del BST che 'e stata implementata in modo da risolvere le funzioni di ricerca di massimo e minimo in un intervallo di date. La chiave secondo cui si gestisce la disposizione degli elementi nel BST è la data del del campo `dailystock`
- `stock` è un modulo caratterizzato dal nome del titolo che contiene l'albero binario con tutte le valutazioni, fornisce tutte le funzioni necessarie per un ADT di prima classe. La funzione `STOCKfakeName(char *name)` crea un'istanza di `Stock` contenente il solo nome, questa viene usata per effettuare le ricerche in lista che confrontano due diversi stock in base al loro nome
- `stocklist` è l'implementazione di una **lista** come ADT di prima classe i cui nodi contengono delle istanze del modulo `stock`

Il `main` si occupa di chiamare le funzioni di cui necessita per lo svolgimento delle 4 attività richiesta. Contiene anche due funzioni che non sono altro che wrapper per la ricerca delle quotazioni massime e minime in un intervallo di date.

2 Scelte algoritmiche

1. `dailystock.c`: le funzioni `DailyStock DSTOCKnew(FILE *f)` e `DailyStock DSTOCKsetNull()` sono tipiche funzioni di creazione di un nuovo dato e di creazione di un dato considerabile nullo (assumo che la data "0000/01/01" sia una data non valida). La funzione `DSTOCKupdate(DailyStock *dStock, float newValue, int nNewStock)` è la funzione che si occupa di andare ad aggiornar i dati nel caso in cui venga letta una nuova transazione di azioni dello stesso titolo nella stessa giornata e lo fa andando a salvare nel campo `valueSumWt` il prodotto del prezzo di queste transazioni per il numero di azioni e in `nStockSum` la somma del numero di azioni vendute nella giornata in modo da poter usare i due campi in modo diretto per il calcolo del valore pesato delle azioni: `value = valueSumWt / nStockSum`.
2. `stockbst.c`: questo modulo fornisce delle funzioni standard di gestione degli alberi binari di ricerca il cui ordinamento è basato sulla data contenuta nel campo di tipo `DailyStock`. Sono standard praticamente standard tutte le funzioni presentate ad esclusione di quella di visita dell'albero che è modificata in modo da poter andare a visitare i soli nodi dell' albero che siano compresi tra due date e quella di bilanciamento. La funzione `SBSTvisit(StockBST sBst, Date date1, Date date2, float *min, float *max)`, che sfrutta la funzione `Link TREEvisitRecursive(Link subRoot, Link z, Date date1, Date date2, float *min, float *max)` utilizza le funzioni di confronto di date fornite dal modulo `datetime` per capire se la data del nodo corrente è compresa tra quelle considerate allora procede con la visita in entrambi i figli, nel caso sia maggiore della seconda delle due date solo dal figlio di sinistre altrimenti da quello di destra. L' unica modifica effettuata alla funzione di bilanciamento dell' albero è una verifica a priori sulla differenza del numero di figli destri e sinistri della radice e effettua il bilanciamento solo se questa differenza supera un valore di soglia `S` dato in input.

3. `stock.c`: le fuzioni fornite da questo modulo sono normali funzioni di interfaccia con un ADT di prima classe e quelle responsabili di inizializzzarlo, confrontarlo e liberarne la memoria. La funzione `STOCKfakeName(char *name)` realizza un'istanza di questo modulo che contiene come informazione utile la sola stringa del nome, questo perché la funzione di confronto usa una funzione `strcmp(char *s1, char *s2)` per confrontare due istanze `Stock` e le stringhe usate sono i nomi delle stesse; questo permette di effettuare ricerche in base a una stringa inserita da tastiera visto che la funzione `STOCKinit(FILE *f)` legge i dati direttamente da file.

4. `stocklist.c`: il modulo fornisce funzioni standard su liste progettate come ADT di prima classe per generare, liberare, inserire elementi in ordine, cercare e stampare liste con nodi composti da istanze del modulo `Stock`. Viene fornita una funzione che implementa una semplice ricerca e una chiamata alla funzione `STOCKaddTransaction` che permette di andare ad inserire all' interno del BST di ogni titolo le nuove transsazioni lette da file.

5. `main.c`: si occupa principalmente di inizializzare la lista di titoli con i dati letti da file e di fornire un menú che offre le 4 funzionalitá richieste dal programma: (i) ricerca di un titolo azionario nella lista attraverso la funzione `SLISTsearch(StockList sList, Stock stock)`, (ii) ricerca della quotazione massima e minima in un intervallo di date e (iii) in tutto il periodo, (iv) bilanciamento del bst nel caso in cui la differenza tra figli destri e sinistri sia maggiore di una soglia `S`. Fornisce due funzioni, che sono wrapper della funzione di ricerca nel BST modificata, `MinAndMaxValueInDateInterval(Stock stock)` per risolvere la richiesta (ii) e `MinAndMaxOfAllTime(Stock stock)` per la richiesta (iii). L'unica differenza tra queste due funzioni è che la prima richiede le date in input per fornirle alla funzione di visita, mentre la seconda sfrutta le funzioni `SBSTmin(StockBST sBst)` e `SBSTmax(StockBST sBst)` per estrarre le date estreme tra quelle nell'albero e fornirle come parametro alla funzione di visita.