Stutende: Scorsone Salvatore	**Matricola**: s247144	**Algoritmi e Programmazione**

Relazione prova di programmazione (18 punti) Appello del 21/02/2020

Strutture dati utilizzate

Il problema in esame gestisce i menù di un ristorante. Vi sono disponibili N piatti, salvati su file, con quattro campi (nome,tipo,contenuto e prezzo); si aggiunge un quinto campo, ordine, che rappresenta la posizione del piatto all'interno del vettore di piatti inizializzato nel "**main.c**". Si è realizzato un'ADT di prima classe per la rappresentazione del dato "piatto". In aggiunta al codice scritto in aula riguardante l'ADT piatto e i relativi ".c" e ".h" vi è una nuova funzione: *tornaPrezzo(PIATTO p).*

Il singolo "menù" è implementato come "quasi ADT", all'interno sono presenti i campi totPrezzo che indica il prezzo totale (che viene aggiornato aggiungendo i piatti e il loro relativo costo), un vettore di interi che rappresentano gli indici nel vettore di piatti, istanziato nel main, dei relativi piatti scelti e il campo nP che ne rappresenta il numero.

È presente inoltre un BST in cui sono salvati i vari menù creati dalla funzione di calcolo combinatorio. Nella radice il BST presenta l'Item con maggiore precedenza. Potendo considerare le funzioni di creazione e di inserimento del BST di libreria, in aula mi sono limitato a chiamarle. Si nota l'inserimento dell'*ITEMsetNull()* in menu.c e menu.h in quanto richiesta dalla *BSTinit()* e altre funzioni. Si è scelto di identificare il valore nullo con un menù di prezzo INT_MAX in quanto la *precedenza* è inversamente proporzionale al prezzo (non si è scelto appunto un prezzo negativo perché verrebbe "preferito" a qualsiasi prezzo reale) e con un numero di piatti (nP) uguale a -1 e quindi "impossibile". Si nota che avrei potuto utilizzare la *BSTempty()* per condizionare l'inserimento in testa nel caso di BST vuoto e demandare il resto degli inserimenti alla *BSTinsert_leafR(…)*. Non lo si è fatto con la correzione da casa per uniformità col codice prodotto in aula.
Si è definito il tipo Item con il "*typedef pMenu Item*". Si è inoltre sostituita la generica *ItemCompare(…)* con la *menuCompare(…)*.

Funzione di confronto del dato "menu"

L'ordine con cui inserire i menù all'interno del BST è dato dalla funzione *menuCompare(…).*
Tale funzione riceve come parametri due puntatori a due differenti menù. Confronta il prezzo dei due menù, se il prezzo del primo è maggiore al prezzo del secondo, ritorna "-1", se il prezzo del primo è minore a quello del secondo, ritorna "1". Ciò perché per il BST ha priorità maggiore il menù col prezzo minore.
Se i due menù hanno identico prezzo, allora, si confrontano il contenuto dei due vettori "piatti".
Tale contenuto, altro non è che un intero che rappresenta la posizione del piatto nel vettore di piatti del "main.c". Va da sé che basta il confronto di questo intero per determinare la precedenza tra i due piatti.
L'indice più piccolo precede il più grande.
Se iterando per tutti i piatti l'uguaglianza persiste anche per gli indici, possiamo definire uguali i due menù.

Modello di calcolo combinatorio

Nella realizzazione della funzione di calcolo combinatorio vi è un errore di fondo che ha condizionato la scelta sul modello da usare. Nella lettura del testo purtroppo mi è sfuggito che il dato int P venisse passato da linea di comando. In assenza di questo, fondamentale, dato ho cercato di applicare il modello di calcolo combinatorio più adatto conoscendo soltanto il numero totale di piatti (dato dal file piatti.txt). Ho quindi optato per le permutazioni con ripetizione (vi è un errore nel codice scritto in aula, ho chiamato due volte la funzione, dopo il backtrack non va richiamata), effettivamente avrei potuto capire d'aver scelto la strada

sbagliata notando che l'ordinamento dei piatti non conta. Nella correzione effettuata a casa ho applicato il corretto modello di calcolo combinatorio (conscio di sapere anche l'intero P, letto da linea di comando), e cioè il modello delle combinazioni con ripetizione. In entrambi i casi (codice prodotto in aula e correzione fatta a casa), dovendo considerare anche eventuali bis, si è scelto di utilizzare un vettore di "marcaggio" int *mark di dimensione P (k nel codice), con ogni cella inizializzata a 2, ad indicare appunto gli eventuali bis. L'utilizzo del mark inizializzato a 2 è la differenza rispetto ai modelli classici visti in aula.
Nella funzione *generaMenu(…)* si generano dinamicamente (e liberano successivamente) i due vettori mark e sol. Si crea anche un menu m temporaneo e statico che andrà a rappresentare il menu da inserire di volta in volta nel BST nella funzione ricorsiva. Nella condizione di terminazione si copiano in m.piatti i valori di sol e si determina il prezzo totale come somma dei prezzi dei singoli piatti. Si procede poi all'inserimento all'interno del BST, differenziando l'inserimento in radice rispetto all'inserimento in foglia tramite un ifelse e la *menuCompare(…)* applicata al confronto tra il massimo del BST e "l'attuale" menu.

Funzione di stampa

La funzione di stampa non è stata ultimata in aula per mancanza di tempo, vengono definite le funzioni di stampa del dato "piatto" e del dato "menu" nei rispettivi ".h" e ".c".
Ho aggiunto, commentato, il codice relativo alla funzione di stampa dell'intero BST, basta richiamare la funzione *BSTvisit(…)* con visita post-order. Vi sono soltanto due semplici modifiche rispetto alla classica funzione di libreria: l' *ITEMstore(…)* nella *treePrintR(…)* viene sostituita dalla già definita *menuPrint(…).*
Essa ha bisogno anche del vettore di piatti per reperire le informazioni dei singoli piatti. La seconda modifica, appunto, consiste nel passare tale vettore alla *BSTvisit(…)* e alla *treePrintR(…)* (usata dalla *BSTvisit(…)* per la stampa).