NOME: MIRKO COGNOME: TENORE MATRICOLA: 311756

RELAZIONE ESAME: ALGORITMI E STRUTTURE DATI (03AAXO)
APPELLO DEL 13 FEBBRAIO 2024
PROGRAMMAZIONE – PROVA DA 12 PUNTI

ESERCIZIO 1
La funzione `SLISTmerge` è stata strutturata in modo che attraversi la lista concatenata `a` e confronti i suoi elementi con quelli della lista `c` inizialmente vuota, riempiendola man mano secondo le specifiche della traccia, e ripetendo il medesimo procedimento attraversando la lista `b`. Le imprecisioni presenti nel codice elaborato durante l'esame sono state risolte nelle modifiche apportate al codice allegato alla relazione:
1) essendo `SLIST` definita come una lista concatenata **ordinata**, sarebbe stato più corretto utilizzare un inserimento che tenesse conto dell'ordinamento per popolare la lista `c`. Invece, gli elementi sono stati inseriti in coda alla lista di destinazione utilizzando il puntatore `tail`, incluso appositamente nella definizione dell'ADT. È stata quindi introdotta una funzione ausiliaria per l'inserimento ordinato all'interno della lista c: `c->head = sortedIns(c->head, x->item);`
2) Per correggere l'allocazione del puntatore alla lista di tipo `SLIST`, è stato sostituito:
`c = malloc(sizeof(*c))` con `c = malloc(sizeof(*sl);`
3) La funzione deve ritornare il dato di tipo `SLIST`, quindi è stato aggiunto: `return c;`
4) Infine, è stato corretto il calcolo della somma per gli elementi duplicati all'interno della lista `c`, aggiungendo: `y->item.val += x->item.val;`

ESERCIZIO 2
Il vettore di puntatori deve essere ordinato secondo profondità crescente e, a pari profondità, secondo valori interi crescenti. Questa disposizione è compatibile con la popolazione del vettore durante una visita in-order del BST. Questo processo è implementato all'interno della funzione ricorsiva `generate`, con `BSTlevelizedNodes` che funge da wrapper. Nella versione modificata del codice allegato alla relazione, la definizione di queste due funzioni è stata completata rivedendo la gestione dell'indice, che ora inizia da 0 e viene incrementato seguendo l'andamento ricorsivo.

ESERCIZIO 3
La generazione della concatenazione a lunghezza massima che rispetti i vincoli imposti dalla traccia è stata implementata utilizzando il modello del **powerset** (insieme delle parti) poiché esplicitamente richiesto dalla traccia che la stringa fosse generata dalla sequenza ordinata di tutte o da parte delle stringhe in elenco prese al più una volta. Le inesattezze corrette nel codice allegato alla relazione sono le seguenti:
1) la più rilevante è il controllo sulla parità o disparità del codice ASCII che non è significativo per determinare se una lettera è una vocale o una consonante, per cui è stata implementata una funzione che verifica se un carattere è o meno una vocale;
2) ora `checksol` restituisce la lunghezza corrente della stringa concatenata ottenuta;
3) `strcat` non richiede la lunghezza della stringa come parametro;
4) sono stati aggiunti tutti gli argomenti alla chiamata ricorsiva alla funzione `generate;`
5) corretta l'allocazione del vettore sol: `char **sol = malloc(nparole*sizeof(char *))` oltre ad aver allocato la memoria per ogni parola al suo interno;
6) `maxlen` viene passata a `generate` come parametro (attraverso puntatore), non andava inizializzata all'interno della funzione;
7) confronto tra int e puntatore a int errato: `if (currlen > maxlen);`
8) `generate` non deve ritornare `res` ma il valore puntato da `res`;
9) `bestConcat` deve ritornare la stringa risultante;
10) l'iterazione che scorre il vettore `sol` in `checksol` deve arrivare solo fino ad un indice prima di `p-1`, dopodiché viene aggiunta alla concatenazione l'ultima stringa dell'insieme (con indice `p-1`) di partenza poiché soddisfa sempre le condizioni dettate dal problema.