ASD a.a. 2023/24 - Esame del 4 Settembre 2024

Prima di cominciare lo svolgimento leggete attentamente tutto il testo.

Questa prova è suddivisa in tre esercizi.
Vi forniamo un file .zip che contiene per ogni esercizio: un file per completare la funzione da implementare e un programma principale per l'esecuzione di test specifici per quella funzione. Ad esempio, per l'esercizio 1, saranno presenti un file es1.cpp e un file es1-test.o. Per compilare dovrete eseguire `g++ -std=c++11 -Wall es1.cpp es1-test.o -o es1-test`. E per eseguire il test, `./es1-test`. Dovete lavorare solo sui file indicati in ciascuno esercizio. Modificare gli altri file è sbagliato (ovviamente a meno di errata corrige indicata dai docenti).
In questi file dovete implementare le funzioni richieste, esattamente con la *segnatura* con cui sono indicate: nome, tipo restituito, tipo degli argomenti nell'ordine in cui sono dati. Non è consentito modificare queste informazioni. Potete invece fare quello che volete all'interno del corpo delle funzioni: in particolare, se contengono già una istruzione `return`, questa è stata inserita provvisoriamente per rendere compilabili i file ancora vuoti, e **dovrete modificarla in modo appropriato**.
Potete inoltre realizzare altre funzioni in tutti i casi in cui lo ritenete appropriato. Potete inserirvi tutti gli `#include` che vi servono oltre a quello relativo allo header con le funzioni da implementare. Attenzione però che **usare una funzione di libreria per evitare di scrivere del codice richiesto viene contato come errore** (esempio: se è richiesto di scrivere una funzione di ordinamento, usare la funzione `std::sort()` dal modulo di libreria standard `algorithm` è un errore).
Per ciascuno esercizio, vi diamo uno programma principale, che esegue i test. Controllate durante l'esecuzione del programma, quanti sono i test che devono essere superati e controllate l'esito (se non ci sono errori deve essere visualizzato `SI` per tutti).

NB1: soluzioni particolarmente inefficienti potrebbero non ottenere la valutazione anche se forniscono i risultati attesi. Di contro ci riserviamo di premiare con un bonus soluzioni particolarmente ottimali.
NB2: superare positivamente tutti i test di una funzione non implica soluzione corretta e ottimale (e quindi valutazione massima).

1 Presentazione della struttura dati

Lo scopo è di programmare tre funzioni per un tipo di albero utile a memorizzare un insieme di parole. Per ottimizzare lo spazio occupato dall'albero, l'idea è quello di mantenere una sola copia dei prefissi comuni a più parole. Come vedremo in dettaglio in seguito ad esempio le parole `"BEBE"` e `"BIBI"` hanno il prefisso iniziale comune `"B"` che è quindi memorizzato una sola volta, mentre le parole `"BIBI"` e `"BIBIDU"` hanno il prefisso iniziale `"BIBI"` in comune che quindi viene salvato una sola volta.
Tutte le parole considerate sono composte di lettere maiuscole ed ogni nodo dell'albero contiene una lettera. L'albero vuoto sarà rappresantato da `nullptr`. Ogni nodo dell'albero ha due puntatori verso altri nodi: uno verso un figlio (`son`) ed uno verso un fratello (`brother`).
Ogni sequenza di nodi, seguendo i puntatori `brother`, può essere vista come una lista di fratelli che contiene le lettere presenti nella stessa posizione nelle parole con prefisso comune sino a quel punto (ad esempio i fratelli (in seconda posizione) `"E"` e `"I"` per le parole `"BEBE"` e `"BIBI"` che hanno il prefisso iniziale comune `"B"`). Invece con i puntatori `son` si va avanti di una posizione nelle lettere che compongono le parole. Una parola è presente nell'albero se la sua ultima lettera ha un figlio che contiene il carattere `8` (altrimenti sarebbe solo un prefisso di una parola più lunga).

Importante: Se il carattere `8` è presente in una lista di fratelli, è sempre in prima posizione e dopo le lettere presenti sono ordinate seguendo l'ordine alfabetico. Ed in ogni lista di fratelli, non appare mai la stessa lettera due volte.

Alla fine di questo documento sono riportati degli esempi di alberi. In questi esempi se un puntatore punta verso `nullptr`, non è rappresentato.

- L'albero `dt1` contiene una unica parola `"BEBE"`. Nella prima lista (fatta da i puntatori `brother`) abbiamo un unico nodo con la lettera `'B'`. Questo nodo ha poi un puntatore `son` verso una nuova lista di fratelli con un unico nodo con la lettera `'E'` che rappresenta la seconda lettera della parola, etc. Si va avanti fino all'ultimo nodo con una lettera `'E'` che ha un puntatore verso una lista di fratelli che contiene il carattere `'8'`. Ne possiamo dedurre che la parola `"BEBE"` appartiene all'albero.
- Se vogliamo aggiungere la parola `"BIBI"` a questo albero, otteniamo l'albero `dt2`.
- Se vogliamo aggiungere la parola `"BOA"` a questo ultimo albero, otteniamo l'albero `dt3`.
- Se vogliamo aggiungere la parola `"BOBO"` a questo ultimo albero, otteniamo l'albero `dt4`.
- Se vogliamo aggiungere la parola `"BIBIDU"` a questo ultimo albero, otteniamo l'albero `dt5`.
- Se vogliamo aggiungere la parola `"ALLO"` a questo ultimo albero, otteniamo l'albero `dt6`.

Nel file `dict-tree.h` troverete la descrizione della struttura dati e i prototipi delle tre funzione da implementare. **Non dovete modificare questo file!**. Questo file si presenta cosi:

```
typedef char Elem;

struct dictNode{
  Elem val;
  dictNode *brother;
  dictNode *son;
};

typedef dictNode *dictTree;

const dictTree emptyDictTree=nullptr;

/***********************************************/
/* Funzione da implementare                    */
/***********************************************/
//Es 1
//Ritorna il numero di parole nel dizionario
unsigned int nbWords(const dictTree&);

//Es 2
//Verifica se una parola e' nel dizionario
//Ritorna true se e' presente e false altrimenti
bool isPresent(std::string, const dictTree&);

//Es 3
//Ritorna la parola piu' piccola dell'albero
//(secondo l'ordine lessicografico)
//Se l'albero e' vuoto, ritorna la string vuota ""
std::string minWord(const dictTree&);
```

2 Esercizio 1 (4 punti)

Nel file `es1.cpp`, dovete implementare la funzione `unsigned int nbWords(const dictTree& dt)`. Questa funzione ritorna il numero di parole nel albero `dt`.
Esempi con gli alberi dati alla fine di questo documento:

- `nbWords(dt1)` => 1
- `nbWords(dt3)` => 3
- `nbWords(dt6)` => 6

Per testare questa funzione, potete usare il file `es1-test.o` compilando con l'istruzione
`g++ -std=c++11 -Wall es1.cpp es1-test.o -o es1-test`.

3 Esercizio 2 (5 punti)

Nel file `es2.cpp`, dovete implementare la funzione `bool isPresent(std::string w, const dictTree& dt)`. Questa funzione ritorna `true` se la parola `w` è presente nell'albero e `false` altrimenti.
Esempi con gli alberi dati alla fine di questo documento:

- `isPresent("BEBE",dt1)` => `true`
- `isPresent("BE",dt1)` => `false`
- `isPresent("ALLO",dt6)` => `true`
- `isPresent("OLLA",dt6)` => `false`

Per testare questa funzione, potete usare il file `es2-test.o` compilando con l'istruzione:
`g++ -std=c++11 -Wall es2.cpp es2-test.o -o es2-test`.

4 Esercizio 3 (5 punti)

Nel file `es3.cpp`, dovete implementare la funzione `std::string minWord(const dictTree&)`. Questa funzione ritorna la parola più piccola secondo l'ordine lessicografico dell'albero. Se l'albero è vuoto (uguale a `nullptr`), la funzione ritorna `""`. Esempi con gli alberi dati alla fine di questo documento:

- `minWord(dt1)` ritorna `"BEBE"`
- `minWord(dt2)` ritorna `"BEBE"`
- `minWord(dt6)` ritorna `"ALLO"`

Per testare questa funzione, potete usare il file `es3-test.o` compilando con l'istruzione:
`g++ -std=c++11 -Wall es3.cpp es3-test.o -o es3-test`.

5 Suggerimenti sulle string

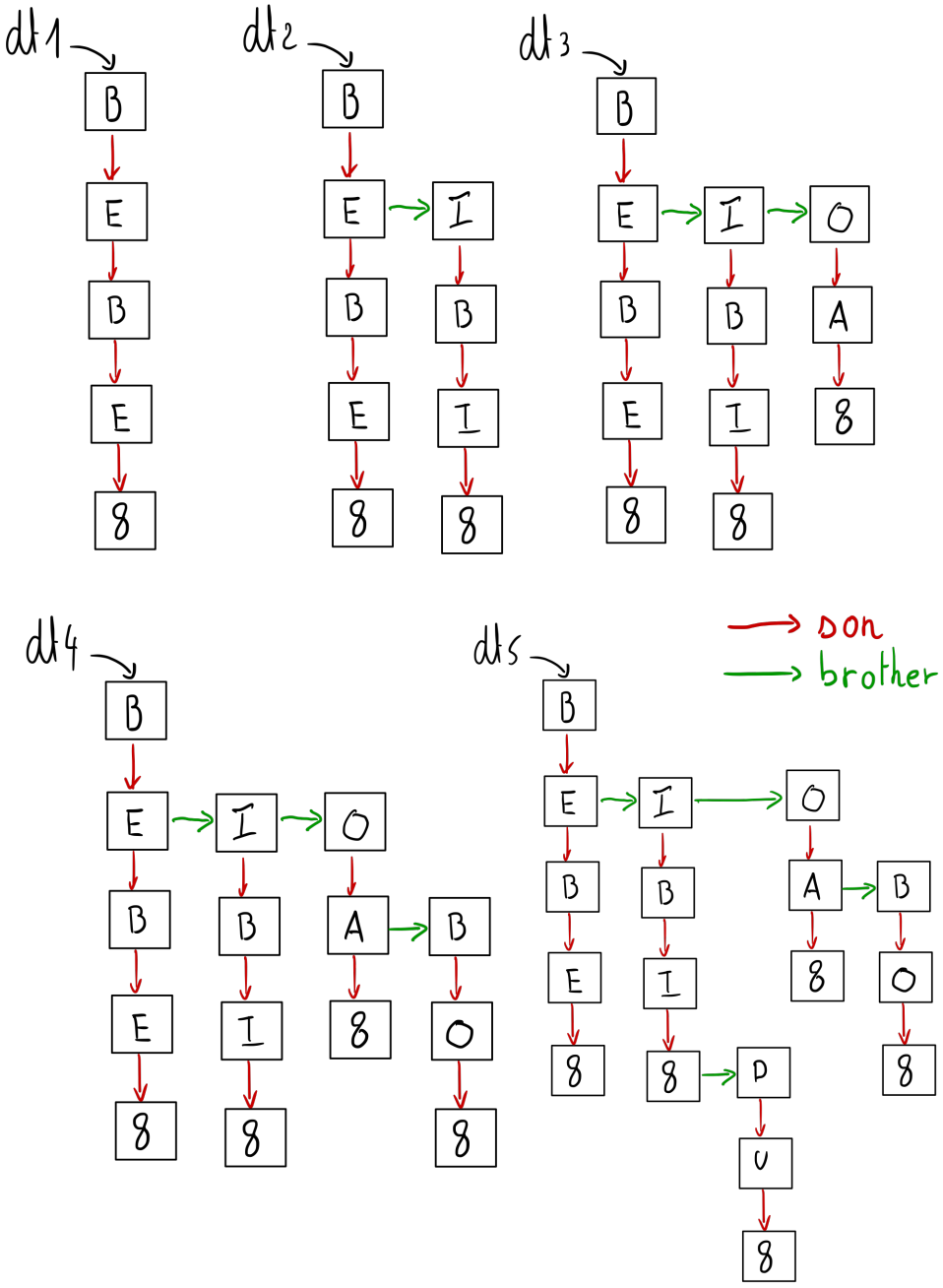
Vi ricordiamo alcune informazioni riguardanti la manipolazione delle `string`.

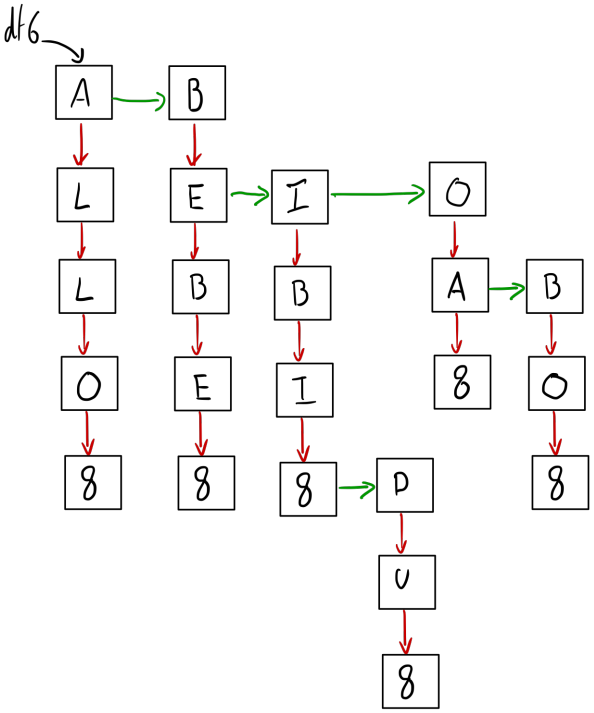
- Se `st` è una `string`, allora facendo `st.length()` potete ricuperare la sua lunghezza. Ad esempio, se abbiamo `string st="ABC"`, allora `st.length()` ritorna 3.
- Se `st` è una `string`, possiamo aggiungere un carattere alla fine con l'operazione `+`. Ad esempio, se abbiamo `string st="ABC"` e se facciamo `st+'D'`, in `st` avremo `"ABCD"`.
- Se `st` è una `string` non vuota, possiamo accedere ai suoi caratteri come si accede agli elementi di un array. Ad esempio, se abbiamo `string st="ABC"`, allora `st[0]` vale `'A'`, `st[1]` vale `'B'` e `st[2]` vale `'C'`.
- Se `st` è una `string` non vuota, possiamo cancellare il suo primo carattere facendo `st.erase(0,1)`. Ad esempio, se abbiamo `string st="ABC"` e facciamo |st.erase(0,1)|, allora dopo `st` vale `"BC"`.

6 Consegna

Per la consegna, creare uno zip con tutti i file forniti.

7 Esempi




df6
A
B
L
E
I
O
L
B
B
A
B
O
E
I
8
O
8
8
8
8
P
V
8