




MAGGIO 2024

PROGRAMMAZIONE & STRUTTURE DATI

WORKBOOK

ALFONSO PISCITELLI
@CLUELAB - UNISA



Sommario

Sommario	0
ESERCIZI SU ADT	4
1. ADT per una playlist musicale	4
2. ADT per un catalogo di libri	4
3. ADT per una lista della spesa	4
4. ADT per una concessionaria di automobili	4
5. ADT per una rubrica telefonica	4
6. Creazione di un ADT per gestire una lista.....	5
7. ADT per uno stack	5
8. ADT per un sistema di gestione di ordini online	5
9. ADT per un gestore di file	5
10. ADT per un sistema di gestione di un ristorante	5
ESERCIZI SU LISTA.....	6
1. Creare una lista vuota e verificare se è vuota:	6
2. Filtro su lista	6
3. Inversione degli Elementi della lista.....	6
4. Ricerca del minimo (e spostamento in testa).....	7
5. Ricerca del massimo (spostamento in coda).....	7
6. Ricerca ed eliminazione di un elemento.....	7
7. Conteggio degli elementi	7
8. Ordinamento della lista	8
ESERCIZI SULLA CODA	9
1. Coda vuota	9
2. Stampare gli elementi di una coda	9
3. Copiare una coda.....	9
4. Implementare una coda utilizzando una lista concatenata.....	9
5. Inversione degli Elementi della coda.....	10
6. Ricerca del minimo (e top della coda)	10
7. Ricerca del massimo (e spostamento in coda)	10
8. Inversione di una coda.....	11

9. Verifica se una stringa è palindroma usando una coda	11
10. Simulare un sistema di ordini in un ristorante	11
ESERCIZI SULLO STACK	12
1. Verifica Stack Vuoto	12
2. Operazioni su stack.....	12
3. Filtro su stack	12
4. Inversione degli Elementi dello Stack.....	13
5. Ricerca del minimo (e top dello stack)	13
6. Ricerca del massimo (e coda dello stack)	13
7. Ricerca di un elemento nello stack (1)	14
8. Ricerca ed eliminazione di un elemento.....	14
9. Conteggio degli elementi	14
10. Copia stack.....	15
11. Esercizio: Valutazione di un'espressione postfix.....	16
12. Esercizio: Ricerca del percorso di un labirinto	17
13. Esercizio: Conversione da espressione postfix ad espressione infix	18
14. Esercizio: Conversione da espressione infix ad espressione postfix.....	19
ESERCIZI SU ALBERI BINARI	20
1. Verifica se un albero è vuoto	20
2. Costruzione di alberi binari	20
3. Inserimento di un elemento in BTree	22
4. Inserimento di un elemento in BTree (2)	22
5. Conteggio dei nodi	23
6. Conteggio dei nodi che hanno il figlio destro	24
7. Ricerca di un elemento.....	24
8. Eliminare un nodo da un albero	25
9. Stampa dell'albero a livelli	25
10. Alberi identici	26
11. Esercizio: Conversione da espressione postfix ad infix (BTree).....	27
12. Il cammino più lungo	28
13. Il cammino più corto	29
14. Bilanciare un albero	30

ESERCIZI SU ADT

Un Abstract Data Type (ADT) è un concetto fondamentale nella programmazione che rappresenta una struttura dati con un insieme di operazioni ben definite, indipendenti dall'implementazione concreta di quella struttura. In altre parole, un ADT definisce un tipo di dato astratto insieme alle operazioni che possono essere eseguite su di esso, senza rivelare i dettagli interni di come queste operazioni sono effettivamente implementate.

1. ADT per una playlist musicale

Definire una struttura per una canzone con campi come titolo, artista, album, e durata. Creare un ADT per una playlist che supporti operazioni come l'aggiunta di una canzone, la rimozione di una canzone, la riproduzione della canzone corrente e il cambio della canzone corrente.

2. ADT per un catalogo di libri

Creare una struttura per un libro con campi come titolo, autore, genere e anno di pubblicazione. Implementare un ADT per un catalogo di libri che permetta di aggiungere un libro, cercare un libro per titolo o autore, e stampare l'intero catalogo.

3. ADT per una lista della spesa

Definire una struttura per un elemento della lista della spesa con campi come nome del prodotto, quantità e prezzo unitario. Creare un ADT per una lista della spesa che permetta di aggiungere un elemento, rimuovere un elemento, calcolare il totale della spesa e stampare la lista.

4. ADT per una concessionaria di automobili

Creare una struttura per un'automobile con campi come marca, modello, anno di produzione e prezzo. Implementare un ADT per una concessionaria di automobili che supporti operazioni come l'aggiunta di un'auto all'inventario, la ricerca di un'auto per marca o modello, la vendita di un'auto e la stampa dell'inventario.

5. ADT per una rubrica telefonica

Definire una struttura per un contatto con campi come nome, cognome, numero di telefono e email. Creare un ADT per una rubrica telefonica che permetta di aggiungere un contatto, cercare un contatto per nome o numero di telefono, aggiornare i dettagli di un contatto e stampare l'intera rubrica.

6. Creazione di un ADT per gestire una lista

Definire una struttura per un nodo di lista e implementare funzioni per l'inserimento di un elemento in testa, in coda e in posizione specifica, la rimozione di un elemento, e la stampa della lista.

7. ADT per uno stack

Creare un ADT per uno stack utilizzando un array. Implementare le operazioni di push, pop, peek e verificare se lo stack è vuoto.

8. ADT per un sistema di gestione di ordini online

Creare una struttura per un ordine con campi come ID ordine, prodotti ordinati e totale dell'ordine. Implementare un ADT per gestire ordini online che supporti operazioni di aggiunta di un ordine, cancellazione di un ordine, calcolo del totale degli ordini e visualizzazione degli ordini per data.

9. ADT per un gestore di file

Definire una struttura per un file con campi come nome, dimensione e tipo di file. Creare un ADT per un gestore di file che permetta di aggiungere un file, cercare un file per nome o tipo, eliminare un file e ordinare i file per dimensione o nome.

10. ADT per un sistema di gestione di un ristorante

Definire una struttura per un piatto con campi come nome, prezzo e ingredienti. Creare un ADT per gestire un menu di un ristorante che permetta di aggiungere un piatto al menu, modificare un piatto, eliminare un piatto e ordinare i piatti per prezzo o nome.

ESERCIZI SU LISTA

1. Creare una lista vuota e verificare se è vuota:

Creare una lista utilizzando la funzione `newList()` e verificare se è vuota utilizzando la funzione `isEmpty()`.

2. Filtro su lista

Scrivi una funzione che, utilizzando in input una lista e un carattere, rimuovi dalla lista tutti gli elementi che iniziano per quel carattere. Ad esempio, se il carattere scelto è **b**, nella lista andranno rimossi tutti gli elementi che iniziano per b.

- Effettuare il filtro utilizzando una nuova lista
- Effettuare il filtro restituendo la stessa lista, senza utilizzarne una nuova

Input		Output
casa buongiorno testo lenzuolo	b	casa testo lenzuolo
ciao tonno luna gatto	t	ciao luna gatto
pollo punto palla mango	p	Mango
pollo punto palla	p	[lista vuota]

3. Inversione degli Elementi della lista

Scrivi una funzione che inverta l'ordine degli elementi in una lista senza utilizzare strutture dati aggiuntive.

- Costruire l'input seguente utilizzando le funzioni nel **main.c**
- Costruire l'input seguente utilizzando un file. Il file è così composto

```
casa buongiorno testo lenzuolo
ciao tonno luna gatto
pollo punto palla mango
```

Input	Output
casa buongiorno testo lenzuolo	lenzuolo testo buongiorno casa
ciao tonno luna gatto	gatto luna tonno ciao
pollo punto palla mango	mango palla punto pollo

4. Ricerca del minimo (e spostamento in testa)

Scrivere una funzione *massimoLista* che, utilizzando una lista fornita in input, cerchi l'elemento di valore massimo nella lista e lo porti in prima posizione (cioè raggiungibile subito con un *getHead*).

Input	Output
10 2 13 8 6	2 10 13 8 6
1 4 5 9 2	1 4 5 9 2
9 8 6 10 1	1 9 8 6 10

5. Ricerca del massimo (spostamento in coda)

Scrivere una funzione *minimoLista* che, utilizzando una lista fornita in input, cerchi l'elemento di valore minimo nella lista e lo porti in ultima posizione

Input	Output
10 2 13 8 6	10 2 8 6 13
1 4 5 9 2	1 4 5 2 9
9 8 6 10 1	9 8 6 1 10

6. Ricerca ed eliminazione di un elemento

Scrivere una funzione *cercaEdElimina* che, utilizzando una lista fornita in input e un elemento (item), verifichi se è presente e lo rimuova dalla lista.

- [E] La funzione può restituire un nuova lista
- [H] La funzione non deve restituire un nuova lista ma lavorare sulla stessa

Input		Output
casa buongiorno testo lenzuolo	casa	buongiorno testo lenzuolo
ciao tonno luna gatto	Tana	ciao tonno luna gatto
pollo punto palla mango	pinna	pollo punto palla mango
pollo punto palla	pollo	pollo punto palla

7. Conteggio degli elementi

Scrivere una funzione *contaLista* che, utilizzando una lista fornita in input, conti il numero di elementi presenti nella lista.

Input	Output
10 2 13 8 6	6
1 4 5 9 2	5
9 8 6 10 1 7 2	7

8. Ordinamento della lista

Scrivere una funzione *ordinaLista* che, utilizzando una lista fornita in input, la restituisca ordinata in modo crescente, utilizzando un algoritmo di ordinamento a piacere. Successivamente implementare le seguenti varianti:

- [E] Ordinare la lista utilizzando *SelectionSort*
- [M] Ordinare la lista utilizzando *BubbleSort*
- [M] Ordinare la lista utilizzando *MergeSort*
- [H] Ordinare la lista utilizzando *SelectionSort* ricorsivo
- [H] Ordinare la lista utilizzando *MergeSort* ricorsivo

Input	Output
10 2 13 8 6	2 6 8 10 13
1 4 5 9 2	1 2 4 5 9
9 8 6 10 1 7 2	1 2 6 7 8 9 10

ESERCIZI SULLA CODA

La struttura dati coda è un tipo di struttura lineare che segue il principio "First In, First Out" (FIFO), il che significa che l'elemento che entra per primo è quello che esce per primo. Ecco una sintesi delle caratteristiche principali della struttura dati coda:

Operazioni fondamentali:

- `enqueue(elemento)`: Aggiunge un elemento alla fine della coda.
- `dequeue()`: Rimuove e restituisce l'elemento all'inizio della coda.
- `front()`: Restituisce l'elemento all'inizio della coda senza rimuoverlo.
- `isEmpty()`: Verifica se la coda è vuota.
- `size()`: restituisce il numero di elementi presenti nella coda

1. Coda vuota

Scrivi una funzione che verifichi se una coda è vuota utilizzando la funzione `isEmpty`. Crea una coda vuota, verifica se è vuota e stampa il risultato.

2. Stampare gli elementi di una coda

Scrivi una funzione che stampi gli elementi presenti in una coda senza modificarli. Crea una coda, aggiungi alcuni elementi e usa la funzione per stampare gli elementi della coda.

3. Copiare una coda

Scrivi una funzione che crei una copia di una coda esistente. Crea una coda, aggiungi alcuni elementi e quindi crea una copia della coda utilizzando la funzione. Stampa entrambe le code per verificare che la copia sia stata creata correttamente.

4. Implementare una coda utilizzando una lista concatenata

Implementa una coda utilizzando una lista concatenata in C. Crea le funzioni `enqueue` per aggiungere un elemento alla coda e `dequeue` per rimuovere un elemento dalla coda. Testa la coda inserendo alcuni elementi e verificando l'ordine di uscita.

5. Inversione degli Elementi della coda

Scrivi una funzione che inverta l'ordine degli elementi in una coda senza utilizzare strutture dati aggiuntive.

- c. Costruire l'input seguente utilizzando le funzioni nel **main.c**
- d. Costruire l'input seguente utilizzando un file. Il file è così composto

```

casa buongiorno testo lenzuolo
ciao tonno luna gatto
pollo punto palla mango
  
```

Input	Output
casa buongiorno testo lenzuolo	lenzuolo testo buongiorno casa
ciao tonno luna gatto	gatto luna tonno ciao
pollo punto palla mango	mango palla punto pollo

6. Ricerca del minimo (e top della coda)

Scrivere una funzione *massimoCoda* che, utilizzando una coda fornita in input, cerchi l'elemento di valore minimo nella coda e lo porti in prima posizione

Input	Output
10 2 13 8 6	2 10 13 8 6
1 4 5 9 2	1 4 5 9 2
9 8 6 10 1	1 9 8 6 10

7. Ricerca del massimo (e spostamento in coda)

Scrivere una funzione *minimoCoda* che, utilizzando una coda fornita in input, cerchi l'elemento di valore minimo nella coda e lo porti in ultima posizione.

Input	Output
10 2 13 8 6	10 2 8 6 13
1 4 5 9 2	1 4 5 2 9
9 8 6 10 1	9 8 6 1 10

8. Inversione di una coda

Scrivi una funzione che inverta l'ordine degli elementi in una coda utilizzando solo operazioni standard di coda come `enqueue`, `dequeue`, e `isEmpty`. Testa la funzione con una coda di esempio e stampa l'ordine invertito degli elementi.

Input	Output
10 2 13 8 6	6 8 13 2 10
1 4 5 9 2	2 9 5 4 1
9 8 6 10 1	1 10 6 8 9

9. Verifica se una stringa è palindroma usando una coda

Scrivi una funzione che verifichi se una stringa è palindroma utilizzando una coda. La funzione dovrebbe prendere una stringa in input, inserire ogni carattere in una coda, e quindi confrontare i caratteri estratti dalla coda con quelli della stringa per determinare se è palindroma o no. Stampare il risultato della verifica. **Nota: non considerare gli spazi**

Input	Output
i topi non avevano nipoti	La stringa è palindroma
non dire gatto se non ce l'hai nel sacco	La stringa non è palindroma
angolo bar a bologna	La stringa è palindroma

10. Simulare un sistema di ordini in un ristorante

Usa una coda per simulare un sistema di ordini in un ristorante. Ogni ordine ha un numero di tavolo e un piatto. Implementa le funzioni per aggiungere un ordine alla coda (`enqueue`) e per servire un ordine (`dequeue`). Stampa gli ordini serviti in ordine.

ESERCIZI SULLO STACK

Uno stack è una struttura dati lineare che segue il principio LIFO (Last In, First Out), il che significa che l'ultimo elemento inserito è il primo ad essere rimosso. Gli elementi sono aggiunti e rimossi dallo stack attraverso due operazioni principali: push (inserimento) e pop (rimozione).

Operazioni Fondamentali sugli Stack:

- newStack(): Crea un nuovo stack vuoto e restituisce un'istanza di stack.
- isEmptyStack(Stack): Verifica se lo stack è vuoto.
- push(Stack, Item): Inserisce un elemento nello stack.
- pop(Stack): Rimuove e restituisce l'elemento in cima allo stack.
- top(Stack): Restituisce l'elemento in cima allo stack senza rimuoverlo.
- printStack(Stack): Stampa tutti gli elementi presenti nello stack.

1. Verifica Stack Vuoto

Scrivi una funzione che verifichi se uno stack è vuoto utilizzando la funzione `isEmptyStack`.

Input	Output
<i>[stack vuoto]</i>	Lo stack è vuoto
10 20	Lo stack non è vuoto

2. Operazioni su stack

Scrivi una funzione che stampi i seguenti output utilizzando degli stack. (ogni parola è un elemento dello stack).

1. ciao gente
2. gente bella ciao come va
3. ei fu siccome immobile

3. Filtro su stack

Scrivi una funzione che, utilizzando in input uno stack e un carattere, rimuovi dallo stack tutti gli elementi che iniziano per quel carattere. Ad esempio, se il carattere scelto è **b**, nello stack andranno rimossi tutti gli elementi che iniziano per b.

- c. Effettuare il filtro utilizzando un nuovo stack
- d. Effettuare il filtro restituendo lo stesso stack, senza utilizzarne uno nuovo

Input		Output
casa buongiorno testo lenzuolo	b	casa testo lenzuolo
ciao tonno luna gatto	t	ciao luna gatto
pollo punto palla mango	p	Mango
pollo punto palla	p	<i>[stack vuoto]</i>

4. Inversione degli Elementi dello Stack

Scrivi una funzione che inverta l'ordine degli elementi in uno stack senza utilizzare strutture dati aggiuntive.

- e. Costruire l'input seguente utilizzando le funzioni nel **main.c**
- f. Costruire l'input seguente utilizzando un file. Il file è così composto

```
casa buongiorno testo lenzuolo
ciao tonno luna gatto
pollo punto palla mango
```

Input	Output
casa buongiorno testo lenzuolo	lenzuolo testo buongiorno casa
ciao tonno luna gatto	gatto luna tonno ciao
pollo punto palla mango	mango palla punto pollo

5. Ricerca del minimo (e top dello stack)

Scrivere una funzione *massimoStack* che, utilizzando uno stack fornito in input, cerchi l'elemento di valore minimo nello stack e lo porti in ultima posizione

Input	Output
10 2 13 8 6	2 10 13 8 6
1 4 5 9 2	1 4 5 9 2
9 8 6 10 1	1 9 8 6 10

6. Ricerca del massimo (e coda dello stack)

Scrivere una funzione *minimoStack* che, utilizzando uno stack fornito in input, cerchi l'elemento di valore minimo nello stack e lo porti in prima posizione (cioè raggiungibile subito con un top).

Input	Output
10 2 13 8 6	10 2 8 6 13
1 4 5 9 2	1 4 5 2 9
9 8 6 10 1	9 8 6 1 10

7. Ricerca di un elemento nello stack (1)

Scrivere una funzione *cerca* che, utilizzando uno stack fornito in input e un elemento (item), verifichi se è presente restituendo TRUE (1) o FALSE (0).

Input		Output
casa buongiorno testo lenzuolo	lenzuolo	TRUE
ciao tonno luna gatto	Tana	FALSE
pollo punto palla mango	pinna	FALSE
pollo punto palla	punto	TRUE

8. Ricerca ed eliminazione di un elemento

Scrivere una funzione *cercaEdElimina* che, utilizzando uno stack fornito in input e un elemento (item), verifichi se è presente e lo rimuova dallo stack. Stampare poi lo stack risultante.

- [E] La funzione può restituire un nuovo stack
- [H] La funzione non deve restituire un nuovo stack ma lavorare sullo stesso

Input		Output
casa buongiorno testo lenzuolo	lenzuolo	casa buongiorno testo
ciao tonno luna gatto	Tana	ciao tonno luna gatto
pollo punto palla mango	pinna	pollo punto palla mango
pollo punto palla	punto	pollo palla

9. Conteggio degli elementi

Scrivere una funzione *contaStack* che, utilizzando uno stack fornito in input, conti il numero di elementi presenti nello stack.

- [H] utilizzando il risultato della funzione *contaStack*, scrivere una funzione *cercaMultiplo* che cerchi il multiplo di 3 più vicino. Restituire AGGIUNGERE se il multiplo è successivo, RIMUOVERE se è precedente, OK se è già multiplo.

Input	Output
10 2 13 8 6	6 (OK)
1 4 5 9 2	5 (AGGIUNGERE)
9 8 6 10 1 7 2	7 (RIMUOVERE)

10. Copia stack

Scrivere una funzione *copiaStack* che, utilizzando uno stack fornito in input, copi il contenuto dello stack (preservando l'ordine), nelle seguenti strutture dati:

- [E] array
- [E] stack
- [M\] coda
- [H] binaryTree (il primo top è radice, i due successivi sono figli della radice ecc).

Input
10 2 13 8 6
1 4 5 9 2
9 8 6 10 1 7 2

11. Esercizio: Valutazione di un'espressione postfix

Implementare un programma che valuti un'espressione matematica in notazione postfix utilizzando uno stack. L'espressione può contenere operatori come +, -, *, / e operandi numerici. Si svolgano i seguenti punti:

- Leggere un'espressione in notazione postfix.
- Valutare l'espressione utilizzando uno stack per gestire gli operatori e gli operandi.
- Stampare il risultato dell'espressione.

Si indichi anche, utilizzando i commenti nel codice, la complessità asintotica della procedura implementata, fornendo una breve giustificazione sulla risposta.

ATTENZIONE: NON modificare item.h, stack.h, stack.c. Si completi il progetto fornito sul canale teams (sezione File, canale tutorato, esercizio-stack).

Testare la procedura sui seguenti **parametro/lista leggendo l'input da file input.txt (crearlo come indicato nell'esempio) dove ogni riga corrisponde ad un'espressione**

Esempio di file di input:

```
3 4 + 2 *
10 5 3 - /
2 4 6 + * 5 /
8 12 4 / +
15 7 - 10 5 / *
```

--- Espressione 1 ---

Espressione postfix: 3 4 + 2 *

Risultato: 14

--- Espressione 2 ---

Espressione postfix: 10 5 3 - /

Risultato: 5

--- Espressione 3 ---

Espressione postfix: 2 4 6 + * 5 /

Risultato: 4

--- Espressione 4 ---

Espressione postfix: 8 12 4 / +

Risultato: 11

--- Espressione 5 ---

Espressione postfix: 15 7 - 10 5 / *

Risultato: 16

12. Esercizio: Ricerca del percorso di un labirinto

Implementare un algoritmo di per trovare un percorso valido in un labirinto. Utilizzare uno stack per mantenere lo stato corrente durante l'esplorazione del labirinto. Si svolgano i seguenti punti:

- Leggere il labirinto da un file di input.
- Implementare la funzione di backtracking per trovare un percorso valido nel labirinto.
- Utilizzare uno stack per mantenere lo stato corrente durante l'esplorazione.
- Stampare il percorso trovato nel labirinto, se esiste.

Si indichi anche, utilizzando i commenti nel codice, la complessità asintotica della procedura implementata, fornendo una breve giustificazione sulla risposta.

ATTENZIONE: NON modificare item.h, stack.h, stack.c. Si completi il progetto fornito sul canale teams (sezione File, canale tutorato, esercizio-stack).

Testare la procedura sui seguenti **parametro/lista leggendo l'input da file input.txt (crearlo come indicato nell'esempio) dove ogni riga corrisponde ad un'espressione**

Esempio di file di input:

```
#####
#S      #
#  #  #  #
#   #   #
#  #   #
#   #   #
##### E
```

Legenda:

#: muro

S: punto di partenza

E: punto di arrivo

Percorso trovato:

```
#####
#S      #
#*###  #
#***#  #
#  ***#
#   **#
#    *#
##### *E
```

13. Esercizio: Conversione da espressione postfix ad espressione infix

Implementare una funzione che converte un'espressione matematica da notazione postfix a infix utilizzando uno stack per mantenere gli operatori. Si svolgano i seguenti punti:

- Implementare una funzione `isOperator` per verificare se un carattere è un operatore.
- Implementare una funzione `precedence` per determinare la precedenza degli operatori.
- Implementare la funzione `infixToPostfix` per la conversione da infix a postfix utilizzando uno stack.
- Utilizzare un file di input per leggere un'espressione infix.
- Stampare l'espressione in notazione postfix ottenuta.

Si indichi anche, utilizzando i commenti nel codice, la complessità asintotica della procedura implementata, fornendo una breve giustificazione sulla risposta.

ATTENZIONE: NON modificare `item.h`, `stack.h`, `stack.c`. Si completi il progetto fornito sul canale teams (sezione File, canale tutorato, esercizio-stack).

Testare la procedura sui seguenti **parametro/lista leggendo l'input da file `input.txt` (crearlo come indicato nell'esempio) dove ogni riga corrisponde ad un'espressione**

Esempio di file di input:

```
3 4 + 2 *
10 5 3 - /
2 4 6 + * 5 /
8 12 4 / +
15 7 - 10 5 / *
```

--- Espressione 1 ---

Espressione postfix: 3 4 + 2 *

Espressione infix: (3 + 4) * 2

--- Espressione 2 ---

Espressione postfix: 10 5 3 - /

Espressione infix: 10 / (5 - 3)

--- Espressione 3 ---

Espressione postfix: 2 4 6 + * 5 /

Espressione infix: 2 * (4 + 6) / 5

--- Espressione 4 ---

Espressione postfix: 8 12 4 / +

Espressione infix: 8 + 12 / 4

--- Espressione 5 ---

Espressione postfix: 15 7 - 10 5 / *

Espressione infix: (15 - 7) * (10 / 5)

14. Esercizio: Conversione da espressione infix ad espressione postfix

Implementare una funzione che converte un'espressione matematica da notazione infix a postfix utilizzando uno stack per mantenere gli operatori. Si svolgano i seguenti punti:

- Implementare una funzione `isOperator` per verificare se un carattere è un operatore.
- Implementare una funzione `precedence` per determinare la precedenza degli operatori.
- Implementare la funzione `infixToPostfix` per la conversione da infix a postfix utilizzando uno stack.
- Utilizzare un file di input per leggere un'espressione infix.
- Stampare l'espressione in notazione postfix ottenuta.

Si indichi anche, utilizzando i commenti nel codice, la complessità asintotica della procedura implementata, fornendo una breve giustificazione sulla risposta.

ATTENZIONE: NON modificare `item.h`, `stack.h`, `stack.c`. Si completi il progetto fornito sul canale teams (sezione File, canale tutorato, esercizio-stack).

Testare la procedura sui seguenti **parametro/lista leggendo l'input da file `input.txt` (crearlo come indicato nell'esempio) dove ogni riga corrisponde ad un'espressione**

Esempio di file di input:

```
(3 + 4) * 2
10 / (5 - 3)
2 * (4 + 6) / 5
8 + 12 / 4
(15 - 7) * (10 / 5)

--- Espressione 1 ---
Espressione infix: (3 + 4) * 2
Espressione postfix: 3 4 + 2 *

--- Espressione 2 ---
Espressione infix: 10 / (5 - 3)
Espressione postfix: 10 5 3 - /

--- Espressione 3 ---
Espressione infix: 2 * (4 + 6) / 5
Espressione postfix: 2 4 6 + * 5 /

--- Espressione 4 ---
Espressione infix: 8 + 12 / 4
Espressione postfix: 8 12 4 / +

--- Espressione 5 ---
Espressione infix: (15 - 7) * (10 / 5)
Espressione postfix: 15 7 - 10 5 / *
```

ESERCIZI SU ALBERI BINARI

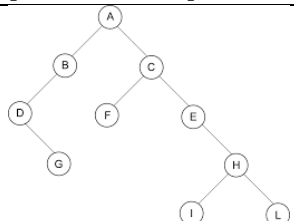
Un binary tree è una struttura dati gerarchica in cui ogni nodo ha al massimo due figli, chiamati figlio sinistro e figlio destro. Ogni nodo può avere zero, uno o due figli, rendendo la struttura simile a un albero reale con un tronco e due rami.

Terminologia Importante:

- Radice: Il nodo in cima all'albero, senza genitori.
- Nodo Padre: Un nodo che ha almeno un figlio.
- Nodo Figlio: Un nodo collegato direttamente a un nodo padre.
- Foglia (Leaf): Un nodo senza figli.
- Nodo Interno: Un nodo con almeno un figlio.
- Livello di un Nodo: La distanza dal nodo radice. La radice ha livello 0, i suoi figli diretti hanno livello 1, e così via.
- Altezza dell'Albero: Il numero massimo di livelli nell'albero.

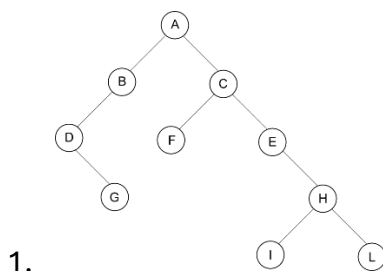
1. Verifica se un albero è vuoto

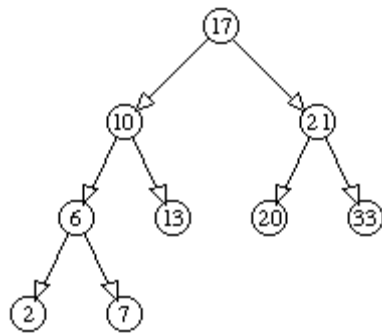
Scrivere una funzione `isEmptyTree` che verifichi se un albero binario è vuoto.

Input	Output
[albero vuoto]	L'albero è vuoto
	L'albero non è vuoto

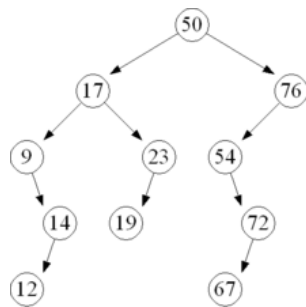
2. Costruzione di alberi binari

Utilizzando l'implementazione della struttura dati Albero Binario (BTree), creare i seguenti alberi binari e stamparli in output.

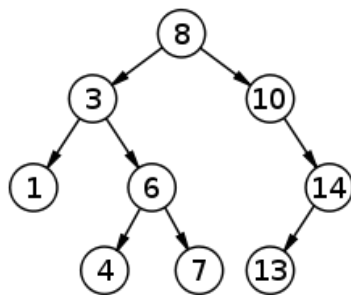




2.



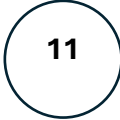
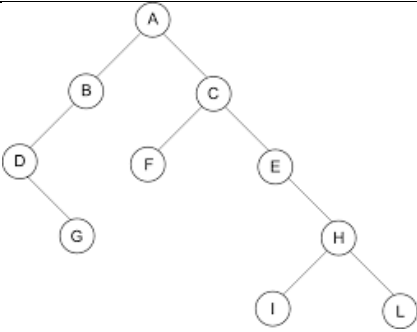
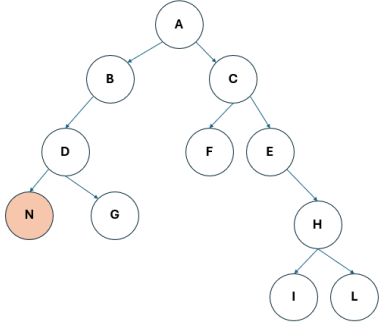
3.



4.

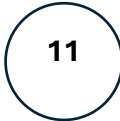
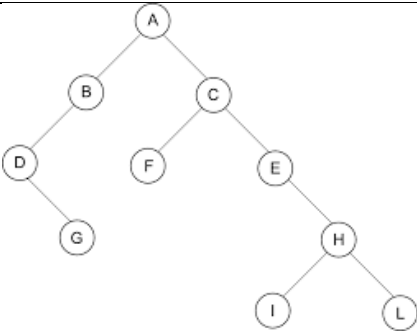
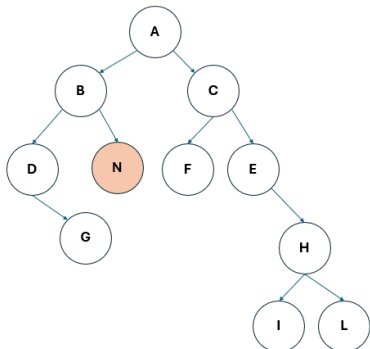
3. Inserimento di un elemento in BTree

Scrivere una funzione `insertNode` che inserisca un nuovo nodo con un dato elemento in un albero binario. Il nodo sarà aggiunto come figlio sinistro al primo nodo che non lo ha, utilizzando una visita pre-order.

Input		Output
[albero vuoto]	1	
	N	

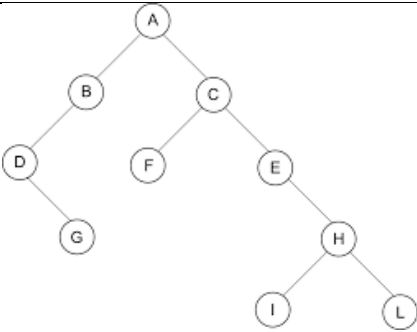
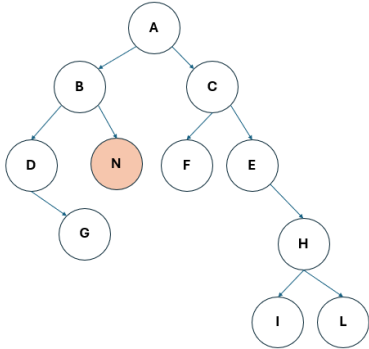
4. Inserimento di un elemento in BTree (2)

Scrivere una funzione `insertNode` che inserisca un nuovo nodo con un dato elemento in un albero binario. Il nodo sarà aggiunto come figlio destro al primo nodo che non lo ha, utilizzando una visita pre-order.

Input		Output
[albero vuoto]	1	
	N	

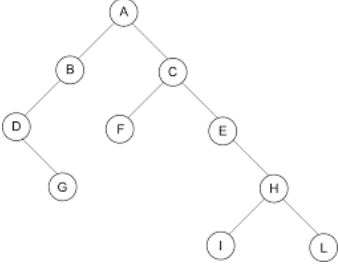
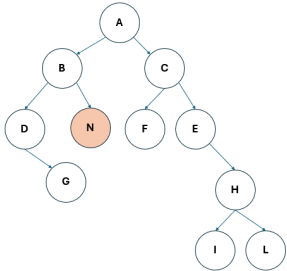
5. Conteggio dei nodi

Scrivere una funzione `countNodes` che conti il numero totale di nodi presenti in un albero binario.

Input	Output
[albero vuoto]	0
	10
	11

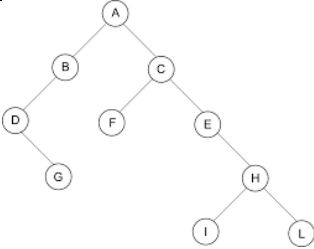
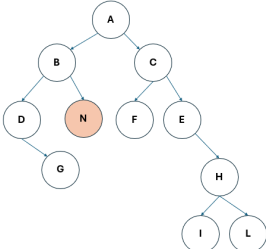
6. Conteggio dei nodi che hanno il figlio destro

Scrivere una funzione `countNodes` che conti il numero totale di nodi che **possiede solo il figlio destro** (figlio sinistro è NULL) presenti in un albero binario.

Input	Output
[albero vuoto]	0
	2
	1

7. Ricerca di un elemento

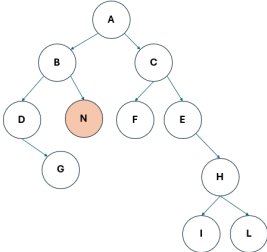
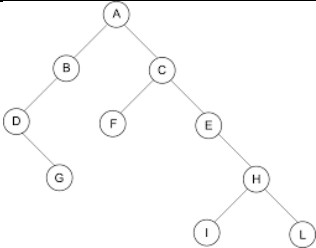
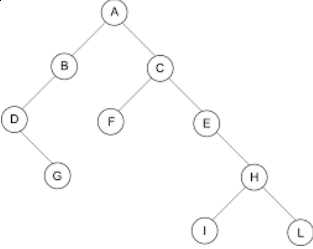
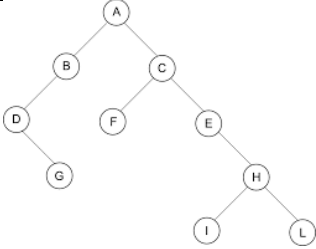
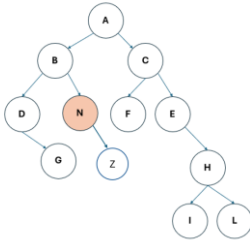
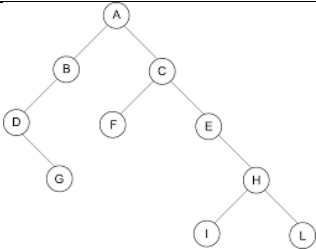
Scrivere una funzione `searchElement` che cerchi un elemento specifico in un albero binario e restituisca il nodo contenente quell'elemento, se presente.

Input	Output
[albero vuoto]	0
 El. Da cercare: L	L
 El. Da cercare: Z	NULL

8. Eliminare un nodo da un albero

Scrivere una funzione `deleteNode` che elimini un nodo contenente un elemento specifico dall'albero binario. Se il nodo non è una foglia, verrà eliminato l'intero sottoalbero.

- [H] Se il nodo non è foglia, uno dei figli (con priorità a sinistra) diventa il nuovo “nodo padre”, prendendo il suo posto nell'albero.
- [H] Se il nodo non è foglia, il figlio con il valore maggiore diventa il nuovo “nodo padre”, prendendo il suo posto nell'albero.

Input	Output
[albero vuoto]	0
 <p>El. Da eliminare: N</p>	
 <p>El. Da eliminare</p>	
 <p>El. Da eliminare: N</p>	

9. Stampa dell'albero a livelli

Scrivere una funzione `printLevelOrder` che stampi gli elementi di un albero binario livello per livello.

Input	Output
[albero vuoto]	

	ABCDNFEGHIL
	ABCDFEGHIL

10. Alberi identici

Scrivere una funzione `areIdentical` che verifichi se due alberi binari sono identici.

Input	Output
<i>[albero vuoto]</i> <i>[albero vuoto]</i>	Gli alberi sono identici
	Gli alberi sono identici
	Gli alberi non sono identici
	Gli alberi non sono identici

11. Esercizio: Conversione da espressione postfix ad infix (BTree)

Implementare una funzione che converte un'espressione matematica da notazione postfix a infix utilizzando un albero binario. Si svolgano i seguenti punti

- A partire dalla stringa in input, generare l'albero binario corrispondente.
- Hint: le foglie conterranno i valori, i nodi interni gli operatori
- Effettuare la stampa dell'espressione (aggiungendo le parentesi), utilizzando la visita opportuna.

Si indichi anche, utilizzando i commenti nel codice, la complessità asintotica della procedura implementata, fornendo una breve giustificazione sulla risposta.

ATTENZIONE: NON modificare item.h, btree.h, btree.c. Si completi il progetto fornito sul canale teams (sezione File, canale tutorato, esercizio-btree).

Testare la procedura sui seguenti **parametro/lista leggendo l'input da file input.txt (crearlo come indicato nell'esempio) dove ogni riga corrisponde ad un'espressione**

Esempio di file di input:

```
3 4 + 2 *
10 5 - 3 /
2 4 + 6 * 5 /
8 12 / 4 +
15 7 - 10 + 5 / *
```

--- Espressione 1 ---

Espressione postfix: 3 4 + 2 *

Espressione infix: (3 + 4) * 2

--- Espressione 2 ---

Espressione postfix: 10 5 - 3 /

Espressione infix: (10 - 5) / 3

--- Espressione 3 ---

Espressione postfix: 2 4 + 6 * 5 /

Espressione infix: ((2 + 4) * 6) / 5

--- Espressione 4 ---

Espressione postfix: 8 12 / 4 +

Espressione infix: (8 / 12) + 4

--- Espressione 5 ---

Espressione postfix: 15 7 - 10 + 5 *

Espressione infix: ((15 - 7) + 10) * 5)

12. Il cammino più lungo

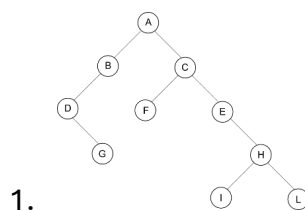
Implementare una funzione che, dato un albero binario, ricerchi e stampi il cammino più lungo a partire dalla radice fino ad una foglia. Si svolgano i seguenti punti:

- Costruire l'albero binario secondo gli esempi forniti graficamente
- Cercare il cammino più lungo tra la foglia e la radice
- *Hint: Si possono usare strutture dati ausiliarie*

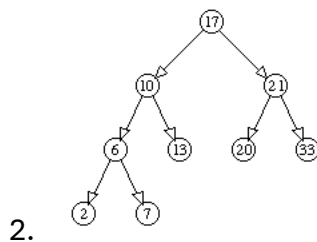
Si indichi anche, utilizzando i commenti nel codice, la complessità asintotica della procedura implementata, fornendo una breve giustificazione sulla risposta.

ATTENZIONE: NON modificare item.h, btree.h, btree.c. Si completi il progetto fornito sul canale teams (sezione File, canale tutorato, esercizio-btree).

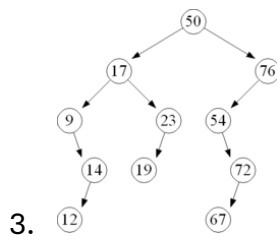
Testare la procedura sui seguenti alberi



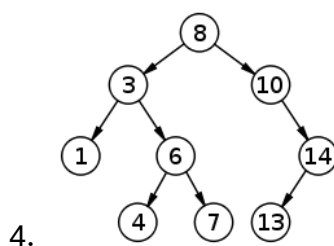
Output: A-C-E-H-I (oppure A-C-E-H-L)



Output: 17-10-6-2 (oppure 17-10-6-7)



Output: 50-17-9-14-12 (oppure 50-76-54-72-67)



Output: 8-10-14-13 (oppure 8-3-6-4, 8-3-4-7)

13. Il cammino più corto

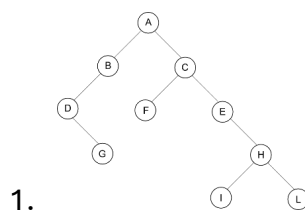
Implementare una funzione che, dato un albero binario, ricerchi e stampi il cammino più corto a partire dalla radice fino alla foglia. Si svolgano i seguenti punti:

- Costruire l'albero binario secondo gli esempi forniti graficamente
- Cercare il cammino più corto tra la foglia e la radice
- *Hint: Si possono usare strutture dati ausiliarie*

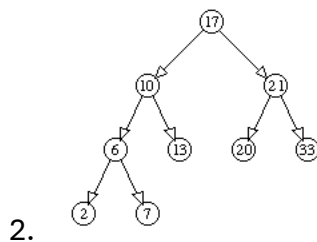
Si indichi anche, utilizzando i commenti nel codice, la complessità asintotica della procedura implementata, fornendo una breve giustificazione sulla risposta.

ATTENZIONE: NON modificare item.h, btree.h, btree.c. Si completi il progetto fornito sul canale teams (sezione File, canale tutorato, esercizio-btree).

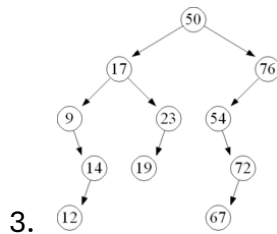
Testare la procedura sui seguenti alberi



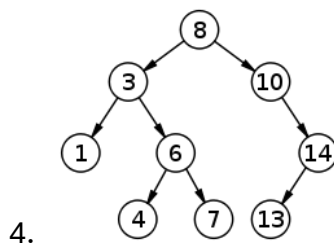
Output: A-C-F



Output: 17-10-13



Output: 50-17-23-19



Output: 8-3-1

14. Bilanciare un albero

Implementare una funzione che, dato un albero binario di numeri interi, lo bilanci secondo il criterio seguente: **la somma dei valori del sottoalbero destro deve essere uguale alla somma dei valori del sottoalbero sinistro**. Per bilanciare si aggiunga un nodo foglia (nel sottoalbero sinistro o destro) che consenta di rispettare il criterio.

- Costruire l'albero binario secondo gli esempi forniti graficamente
- Verificare se, rispetto alla radice, l'albero è bilanciato

Si indichi anche, utilizzando i commenti nel codice, la complessità asintotica della procedura implementata, fornendo una breve giustificazione sulla risposta.

ATTENZIONE: NON modificare item.h, btree.h, btree.c. Si completi il progetto fornito sul canale teams (sezione File, canale tutorato, esercizio-btree).

Testare la procedura sui seguenti alberi

Input	Output
