Alberi

1. Crea una funzione che "pota" l'albero rimuovendo tutti i nodi con valore inferiore a k.

Per la seguente funzione, fornire PRE e POST condizioni. Qualora la soluzione scelta sia iterativa, si faccia la versione ricorsiva.

```
BTree* prune_tree(BTree* root, int k);
```

2. Crea una funzione che conta il numero di percorsi nell'albero la cui somma dei valori dei nodi è uguale a sum.

Per la seguente funzione, fornire PRE e POST condizioni. Qualora la soluzione scelta sia iterativa, si faccia la versione ricorsiva.

```
int count_paths_with_sum(BTree* root, int sum);
```

3. Crea una funzione che trova il percorso più lungo da una foglia a un'altra in un albero binario. Per la seguente funzione, fornire PRE e POST condizioni. Qualora la soluzione scelta sia iterativa, si faccia la versione ricorsiva.

```
int diameter_of_tree(BTree* root);
```

Matrici

4. Crea una funzione che ruota una matrice quadrata di 90 gradi in senso orario. Ad esempio: Input: Output: 1 2 3 7 4 1 4 5 6 -> 8 5 2 7 8 9 9 6 3

Per la seguente funzione, fornire PRE e POST condizioni. Qualora la soluzione scelta sia iterativa, si faccia la versione ricorsiva.

```
void rotate_matrix(int m[][N], int n);
```

5. Ordine spirale: Questo metodo attraversa una matrice partendo dall'angolo in alto a sinistra e procedendo a spirale in senso orario verso il centro. Ad esempio: Input: Ordine spirale: 1 2 3 4 5 6 -> 1, 2, 3, 6, 9, 8, 7, 4, 5 7 8 9

Realizza una funzione in grado di rispettare questo ordine.

```
void spiral_order(int m[][N], int rows, int cols, int result[]);
```

6. Crea una funzione che trova il numero di isole in una matrice binaria. Un'isola è un gruppo di 1 connessi (orizzontalmente o verticalmente) circondati da 0. Per la seguente funzione, fornire PRE e POST condizioni. Qualora la soluzione scelta sia iterativa, si faccia la versione ricorsiva.

```
int count_islands(int matrix[][N], int rows, int cols);
```

Un esempio concreto:

1 1 0 0 0

0 1 0 0 1

1 0 0 1 1

0 0 0 1 0

1 0 1 0 1

In questa matrice:

- La prima isola è composta dagli 1 nelle posizioni (0,0), (0,1), (1,1)
- La seconda isola è composta dagli 1 nelle posizioni (1,4), (2,3), (2,4), (3,3)
- La terza isola è composta dal singolo 1 nella posizione (4,0)
- La quarta isola è composta dal singolo 1 nella posizione (4,2)
- La quinta isola è composta dal singolo 1 nella posizione (4,4)

Liste

7. Crea una funzione che riordina una lista in modo che tutti i nodi pari vengano prima dei nodi dispari, mantenendo l'ordine relativo originale. Per la seguente funzione, fornire PRE e POST condizioni. Qualora la soluzione scelta sia iterativa, si faccia la versione ricorsiva.

```
void segregate_even_odd(Lista** head);
```

8. Crea una funzione che riordina una lista in modo che sia palindroma. Per la seguente funzione, fornire PRE e POST condizioni. Qualora la soluzione scelta sia iterativa, si faccia la versione ricorsiva.

```
void make_palindrome(Lista** head);
```

9. Questa funzione rimuove tutti gli elementi della lista con valori compresi tra start e end (inclusi). Per la seguente funzione, fornire PRE e POST condizioni. Qualora la soluzione scelta sia iterativa, si faccia la versione ricorsiva.

```
void remove_range(Lista** head, int start, int end);
```