Algoritmi e Strutture Dati

# Tracce

1. Data una lista di elementi di tipo intero, implementare in C++ le seguenti funzioni

   - `int freq(List<int> &L, int k)`: restituisce il numero di occorrenze dei multipli di `k` nella lista `L`;
   - `void hist(List<int> &L)`: stampa la frequenza di ogni elemento presente nella lista `L`;
   - `void remp(List<int> &L)`: rimuove dalla lista `L` tutti gli elementi seguiti da un numero dispari.

   Prevedere una funzione nel main che contenga una procedura di inserimento automatico e successivamente il test delle funzioni implementate.

2. Dato un albero n-ario di elementi di tipo intero, implementare in C++ la funzione `int max_level(Tree<int>)` che restituisce in output il livello `k` dell'albero per il quale la somma di tutti i nodi di livello `k` è massima.

   Prevedere una funzione nel main che contenga una procedura di inserimento automatico e successivamente il test della funzione implementata.

3. Data una pila di interi, implementare in C++ una funzione che restituisca una nuova pila i cui elementi corrispondano a quelli presenti nella pila data e siano ordinati dall'alto (top dello stack) al basso.

   Prevedere una funzione nel main che contenga una procedura di inserimento automatico e successivamente il test della funzione implementata.

4. Dato un albero binario di elementi di tipo intero, implementare in C++ la funzione `int multipli(Tree<int> T)` che modifica l'etichetta di ogni nodo $n$ dell'albero memorizzandoci il numero dei nodi pari presenti nel sottoalbero radicato in $n$.

   Prevedere una funzione nel main che contenga una procedura di inserimento automatico e successivamente il test della funzione implementata.

5. Una struttura dati acquisisce numeri interi da 1 a 9 che inserisce in tre code secondo il seguente schema.

   Gli interi da 1 a 3 vengono inseriti nella prima coda, quelli da 4 a 6 nella seconda, e i restanti nella terza.

   La struttura mette a disposizione una serie di metodi che permettono di conoscere e cambiare il suo stato.

   Realizzare in C++ la seguente classe che implementa la struttura appena descritta.

```
class Bins {
// inserisce l'intero k nella corrispondente coda
void insert(int k);
// effettua un fuoriCoda dalla coda c
void delete(int c);
// restituisce la media dei valori attualmente memorizzati nella coda c
double mean(int c);
// restituisce il numero di elementi inseriti nella coda c
int freq(int c);

private:
coda<int> bin[3];
}
```

   Prevedere un main che mostri a video il test dei metodi implementati.

6. Un sito di commercio elettronico dispone dei giudizi (voti, rating) che alcuni utenti hanno fornito per alcuni dei suoi prodotti. In particolare dispone di un elenco di giudizi della forma
`<utente, prodotto, giudizio>`,
dove giudizio è un valore numerico da 1 a 5.

   Implementare le seguenti classi C++ (aggiungendo eventuali membri e fornendo la realizzazione dei metodi richiesti), per la gestione di un elenco di giudizi:

```
class rating{
  public:
    string utente;
    string prodotto;
    int rating;
};


class recommender{
  public:
    // inserisce nella lista ratings una nuovo rating
    // <user, item, rating>
    void insert(string user, string item, int rating);

    // restituisce la media dei rating dell'utente user
    double meanUser(string user);

    // restituisce la media dei rating per il prodotto item
    double meanItem(string item);

    // restituisce la lista dei prodotti votati dall'utente user
    List<string> rated(string user);

    // restituisce la lista dei prodotti votati sia dall'utente user1
    // che dall'utente user2
    List<string> common(string user1, string user2);

    // usando il metodo common, calcola quanti prodotti sono stati votati
    // allo stesso modo sia dall'utente user1 che dall'utente user2
   int commonEqual(string user1, string user2);

  private:
    List<rating> ratings;
}
```

   Prevedere un main che acquisisca automaticamente una serie di preferenze e mostri a video il test dei metodi implementati.

   Eseguire i test con il seguente campione di dati

```
recommender r;

r.insert("u1","p1",1);
r.insert("u1","p3",3);
r.insert("u1","p5",4);
r.insert("u2","p1",2);
r.insert("u2","p2",3);
r.insert("u2","p3",3);
```

```
r.insert("u3","p1",1);
r.insert("u3","p2",1);
r.insert("u3","p3",1);
r.insert("u3","p5",5);
r.insert("u4","p3",2);
r.insert("u4","p4",4);
r.insert("u5","p2",3);
r.insert("u5","p3",2);
r.insert("u6","p4",2);
```

7. Fornire in un file `util_bin_tree.cpp` la realizzazione in C++ delle seguenti funzioni che operano su alberi binari di elementi di tipo intero:

   - `int odd(bintree<int> &T, int k)`: calcola e restituisce il numero di nodi di livello $k$ il cui valore dispari;
   - `int even_leafs(bintree<int> &T)`: calcola e restituisce il numero di foglie il cui valore pari;
   - `int leafs_with_even_parent(bintree<int> &T)`: calcola e restituisce il numero di foglie che hanno come genitore un nodo il cui valore pari.

   Prevedere un main che acquisisca automaticamente un albero binario e mostri a video il test delle funzioni implementate.

8. Utilizzando una delle realizzazioni della ADT lista, realizzare in C++ la ADT `ListaOrdinata.h` per la memorizzazione ordinata di elementi di tipo intero (gli elementi compaiono in modo ordinato e sono ammesse più copie dello stesso elemento). **Bisogna realizzare la nuova struttura facendo uso degli operatori della ADT lista**.

   La ADT `ListaOrdinata` mette a disposizione i seguenti operatori:

   - `inserisci(S, elemento)`: inserisce l'elemento nella sequenza ordinata S
   - `rimuovi(S, elemento)`: rimuove la prima occorrenza dell'elemento dalla sequenza S
   - `differenza(S1, S2)`: restituisce una nuova sequenza ordinata ottenuta eliminando da S1 tutti gli elementi presenti in S2
   - `sottolista(S1, S2)`: restiruisce true se la sottolista ordinata S1 sottolista della lista ordinata S2, false altrimenti

   Prevedere un main che acquisisca automaticamente delle sequenze e mostri a video il funzionamento degli operatori implementati.

9. Completare la seguente classe C++ che stabilisce, utilizzando uno stack, se le parentesi '(', ')', '[', ']', '{', e '}' presenti in una stringa sono bilanciate.

```
#include <stdio.h>
#include <stack.h>

template<class T>
class Bilanciate {
public:
  void test(string s);

private:
  Stack<T> S;
}
```

Il metodo `test`, testa se le parentesi contenute nella stringa `s` sono bilanciate. Possono, verificarsi tre casi: a) il metodo legge una parentesi aperta non bilanciata (manca la corrispondente parentesi chiusa); b) legge una parentesi chiusa non bilanciata (manca la corrispondente parentesi aperta); c) tutte le parentesi sono bilanciate. Nei casi di errore, visualizza sullo standard output il tipo di errore e la **posizione nella stringa** in cui è presente l'errore. Prevedere un main che invoca automaticamente il metodo `test` sulle seguenti stringhe:

- ``for (i=0) to 10''.
- ``(3+2)/[3-(2-1)]''.
- ``{1-(3+2)/[3-(2-1)]}''.
- ``{1-(3+2)/[3-(2-1))]}''.
- ``{1-(3+2)/([3-(2-1)]}''.
- ``{{1-(3+2)/([3-(2-1)]}''.
- ``{1-(3+2)/([3-(2-1)]''.

10. Fornire in un file `util_bin_tree.cpp` la realizzazione in C++ delle seguenti funzioni che operano su alberi binari di elementi di tipo intero:

    - `int d3(bintree<int> &T, int k)`: calcola e restituisce il numero di nodi di livello $k$ il cui valore divisibile per 3;
    - `int lp(bintree<int> &T)`: calcola e restituisce il numero di foglie il cui valore pari;
    - `int l2p(bintree<int> &T)`: calcola e restituisce il numero di foglie che hanno come genitore un nodo il cui valore un multiplo di 5.

    Prevedere un main che acquisisca automaticamente un albero binario e mostri a video il test delle funzioni implementate.

11. Implementare un classe C++ che fornisca i seguenti metodi che operano su grafi orientati. Ogni nodo del grafo è etichettato con il nome di uno dei seguenti colori: `rosso`, `verde` o `bianco`.

    - `sameColorPath(Node n1, Node n2)`: restituisce `true` se esiste un path che collega il nodo `n1` al nodo `n2` e tutti i nodi presenti in tale path sono dello stesso colore, `false` altrimenti;
    - `uniformColorPath(Node n1, Node n2)`: restituisce `true` se esiste un path che collega il nodo `n1` al nodo `n2` ed ogni nodo presente in tale path è etichettato con un colore diverso da quello con cui è etichettato il nodo precedente, `false` altrimenti.

    Prevedere una funzione main che contenga una procedura di inserimento automatico e successivamente il test dei metodi implementati.

12. Dato un albero n-ario con nodi aventi etichetta di tipo intero, implementare un metodo in C++ che modifichi l'albero in modo tale che l'etichetta di ogni nodo `n` corrisponda alla somma dei valori delle etichette dei nodi del sottoalbero radicato in `n`.

    Prevedere una funzione main che contenga una procedura di inserimento automatico e successivamente il test dei metodi implementati.

13. Dato un grafo orientato i cui archi hanno peso 1 o $-1$, implementare un metodo in C++ che dati due nodi $a$ e $b$ individui il numero di cammini da $a$ a $b$ tale che, gli archi nella sequenza corrispondente ad ogni cammino siano etichettati con 1. Inoltre fornire la media della lunghezza dei cammini individuati.

    Prevedere una funzione main che contenga una procedura di inserimento automatico e successivamente il test dei metodi implementati.

14. Data una pila di elementi di tipo intero, implementare un metodo in C++ che elimini dalla pila tutti gli elementi maggiori di un certo intero $k$ (l'ordinamento degli elementi della pila risultante deve corrispondere a quello della pila data in input).

    Prevedere una funzione main che contenga una procedura di inserimento automatico e successivamente il test dei metodi implementati.

15. Data una pila di interi, implementare in C++ una funzione che restituisca una nuova pila i cui elementi corrispondano a quelli presenti nella pila data e siano ordinati dall'alto (top dello stack) al basso.

    Prevedere una funzione nel main che contenga una procedura di inserimento automatico e successivamente il test della funzione implementata.

16. Dato un albero n-ario di elementi di tipo intero, implementare in C++ una funzione che calcoli, **ispezionando l'abero in ordine simmetrico (invisita) con valore di** $i = 1$, il numero di nodi aventi come padre un nodo il cui valore è pari.

    Prevedere una funzione nel main che contenga una procedura di inserimento automatico e successivamente il test della funzione implementata.

17. Data una lista di elementi di tipo intero, modificarla in moda da eliminare tutti gli elementi il cui valore è un multiplo della rispettiva posizione ordinale occupata nella lista.

    Prevedere una funzione nel main che contenga una procedura di inserimento automatico e successivamente il test della funzione implementata.

18. Implementare un classe C++ che fornisca i seguenti metodi che operano su grafi orientati. Ogni nodo del grafo è etichettato con un valore intero.

    - `countSame(Node n1)`: restituisce il numero di nodi raggiungibili da `n1` e aventi la sua stessa etichetta;
    - `meanN2(Node n1)`: restituisce la media dei valori delle etichette dei nodi raggiungibili da `n1` con cammini di lunghezza 2.

    Prevedere una funzione main che contenga una procedura di inserimento automatico e successivamente il test dei metodi implementati.