UNIVERSITÀ DELLA CALABRIA

Esercizio 1

Data la seguente porzione di programma rispondere, dando motivazione, alle domande corrispondenti:

```
int main() {
    int* matricola = new int[6] {//Inserire qui la tua matricola};

    // 1. La seguente istruzione è corretta? Se si, cosa stampa?

    cout << *(matricola) + 4 << endl;

    // 2. La seguente istruzione è corretta? Se si, che cosa stampa?

    matricola[3] = *(matricola+2);
    cout << matricola[3] << endl;

    // 3. La seguente istruzione è corretta? Se si, che cosa stampa?

    int* b = matricola;

    for (int i=1; i < 6; i++) {
      if (matricola[i] > *b) {
            b = matricola+i;
      }
    }

    cout << *(b) << endl;

    // 4. Le seguenti istruzioni sono corrette? Motivare la risposta

    delete[] matricola;
    delete b;

    return 0;

}
```

Esercizio 2

Ereditando opportunamente da `vector<int>`, implementare la classe `rotcev`, che ridefinisca i seguenti metodi pubblici di `vector<int>`:

- `int& operator[] (int i)`
- `void push_back(int value)`
- `int front() const`
- `int back() const`
- `unsigned size() const`

`operator[]` deve restituire l'i-esimo elemento a partire da destra, `front` e `back` devono restituire rispettivamente l'ultimo ed il primo elemento del `vector`, mentre `push_back` e `size` devono comportarsi esattamente come in `vector<int>`. Quelli elencati devono essere **gli unici metodi pubblici disponibili nella classe,** oltre ad un costruttore. Deve essere dunque proibito usare i metodi di `vector<int>` sulle istanze di `rotcev`, ed eventuali altre classi che ereditano da `rotcev` non devono avere accesso (neanche all'interno della classe) ai metodi di `vector<int>`.

Fondamenti di Programmazione 2
(Corso di Laurea in Informatica)
Esame del 25/11/2025

UNIVERSITÀ DELLA CALABRIA

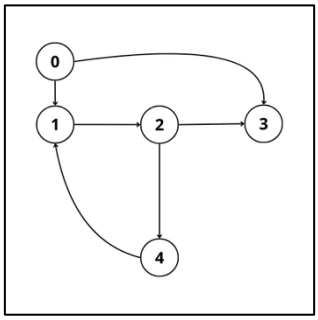
Esercizio 3

Scrivere una funzione che, dato un `Grafo` orientato `g` (implementato tramite la classe `Grafo`), e un suo nodo `x`, restituisce `true` se e solo e se esiste un cammino semplice da `x` ad un qualsiasi altro nodo di `g` dal quale è possibile tornare direttamente al nodo `x` tramite un arco diretto. Se x non ha archi entranti la condizione è da ritenersi automaticamente falsa.

La classe `Grafo` mette a disposizione la seguente interfaccia pubblica di metodi costanti (con `g` istanza della classe):

- `g.n()` restituisce il numero di nodi del grafo,
- `g.m()` restituisce il numero di archi del grafo,
- `g(i,j)` restituisce `true` se esiste l'arco diretto tra il nodo `i` e il nodo `j`.

I nodi sono etichettati con i valori da `0` a `g.n()-1`.



Esempio: Nell'esempio riportato a destra, per x=0 la funzione deve restituire false perché 0 non ha archi entranti, mentre per x=1 la funzione restituisce true in quanto esiste il cammino {1,2,4} che riporta al nodo 1 tramite l'arco (4,1).

Esercizio 4

Stiamo costruendo una simulazione di civiltà medievale. In questa regione sono presenti due fazioni nemiche (**T** ed **F**), ed `n` città, ognuna controllata da una delle due fazioni. Per conto della fazione **T** dobbiamo decidere in quali città piazzare i `k` soldati dell'esercito per tenere sotto sorveglianza l'attività della fazione **F**. Ci viene dato in input:

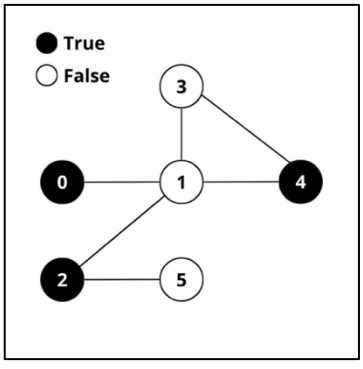
- un *grafo non orientato* `G` (rappresentato dalla classe `Grafo`) con `n` nodi, in cui *ogni nodo rappresenta una delle città* (ogni città è identificata da un numero intero positivo da 0 ad n-1). Dato che ogni città ha una torre d'osservazione dalla quale è possibile scrutare alcune delle città vicine, se dalla torre della città `i` si vede la città `j`, questa informazione è rappresentata da un arco `(i,j)`.
- un `vector<bool>` di taglia `n` in cui ogni indice rappresenta una città e il valore corrispondente riporta la fazione che la controlla: `true` per la fazione **T** e `false` per **F**.
- un intero positivo `k`, corrispondente al numero di soldati da piazzare.

Tenendo conto che ogni soldato deve essere piazzato in esattamente una città, che possiamo piazzare i soldati *solo nelle città controllate dalla nostra fazione* (**T**) e che in ogni città **non** possiamo piazzare più di un soldato, il nostro obiettivo è quello di scrivere un programma C++ in grado di *trovare una sistemazione ai nostri soldati in modo tale da* **riuscire a tenere sotto controllo tutte le città della fazione nemica (dobbiamo "avere visibilità" su ogni città controllata da F con almeno un soldato).** Se è possibile trovare una soluzione il programma deve stamparla su standard output, stampando la stringa "IMPOSSIBILE" nel caso di soluzione inesistente.

La classe `Grafo` mette a disposizione la seguente interfaccia pubblica di metodi costanti (con `g` istanza della classe):

- `g.n()` restituisce il numero di nodi del grafo,
- `g.m()` restituisce il numero di archi del grafo,
- `g(i,j)` restituisce `true` se esiste l'arco diretto tra il nodo `i` e il nodo `j`.

I nodi sono etichettati con i valori da `0` a `g.n()-1`.



Nell'esempio a destra, per k=2, dove la fazione T è indicata con il colore nero e la fazione F con il colore bianco, l'unica soluzione esistente è quella di piazzare un soldato in "2" e un soldato in "4", in modo da avere visibilità in tutte le città "bianche": 1 -> tramite (4,1) e (2,1) 3-> tramite (4,3) e 5-> tramite (2,5).