GRAFI

Angelo Di Iorio

Università di Bologna

Esercizio 1

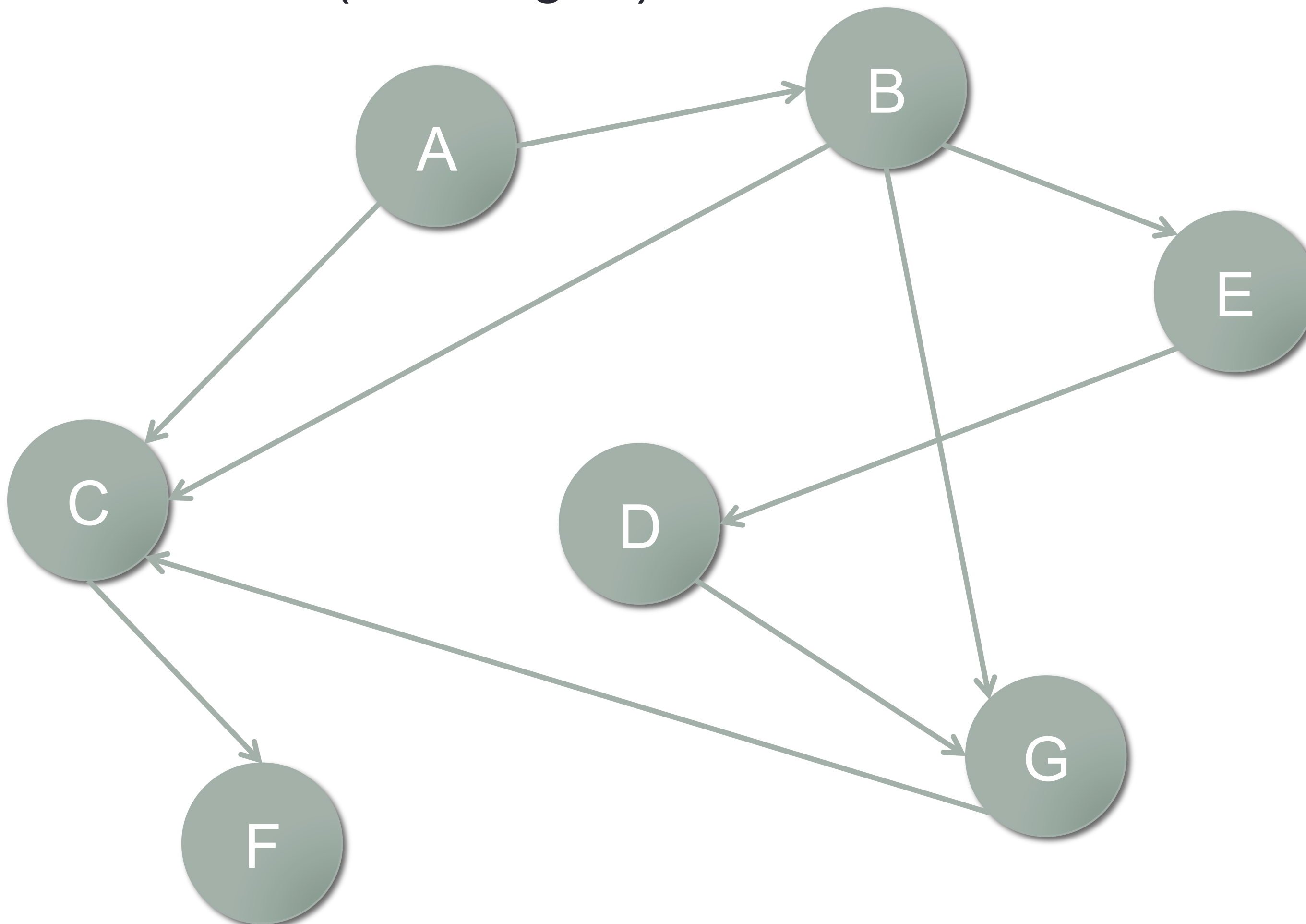
- Implementare una classe Java per memorizzare e manipolare un grafo orientato di nodi di tipo T (Generics Java)
  - Applicazioni: rete di fermate di autobus, città, reti sociali, etc.
- Usare le strutture dati disponibili in Java Collection Framework (`java.util.*`)
- Realizzazione basata su liste di adiacenza
- Interfaccia mostrata nella prossima slide

```java
import java.util.Set;

public interface IGraph<T> {

	public void insertNode(Node<T> u);

	public void deleteNode(Node<T> u);

	public void insertEdge(Node<T> u, Node<T> v);

	public void deleteEdge(Node<T> u, Node<T> v);

	public Set<Node<T>> adj(Node<T> u);

	public Set<Node<T>> V();

	// utility: stampa nodi e liste di adiacenza
	public void print();
}
```

https://diiorio.nws.cs.unibo.it/asd1718/java/

Esercizio 1b

- Utilizzare la precedente API per memorizzare il seguente grafo orientato (di stringhe):



Esercizio 2 (esercizio 9.4 del libro)

- Implementare un metodo per verificare se un grafo non orientato è bipartito oppure no

- Un grafo non orientato G è *bipartito* se l'insieme dei nodi può essere partizionato in due sottoinsiemi tali che nessun arco connette due nodi appartenenti alla stessa parte.

- Definire una classe `GraphVisits` che implementa il metodo:

```
public Boolean isBipartite(Graph<T> graph);
```

- `GraphVisits` deve una classe generica:

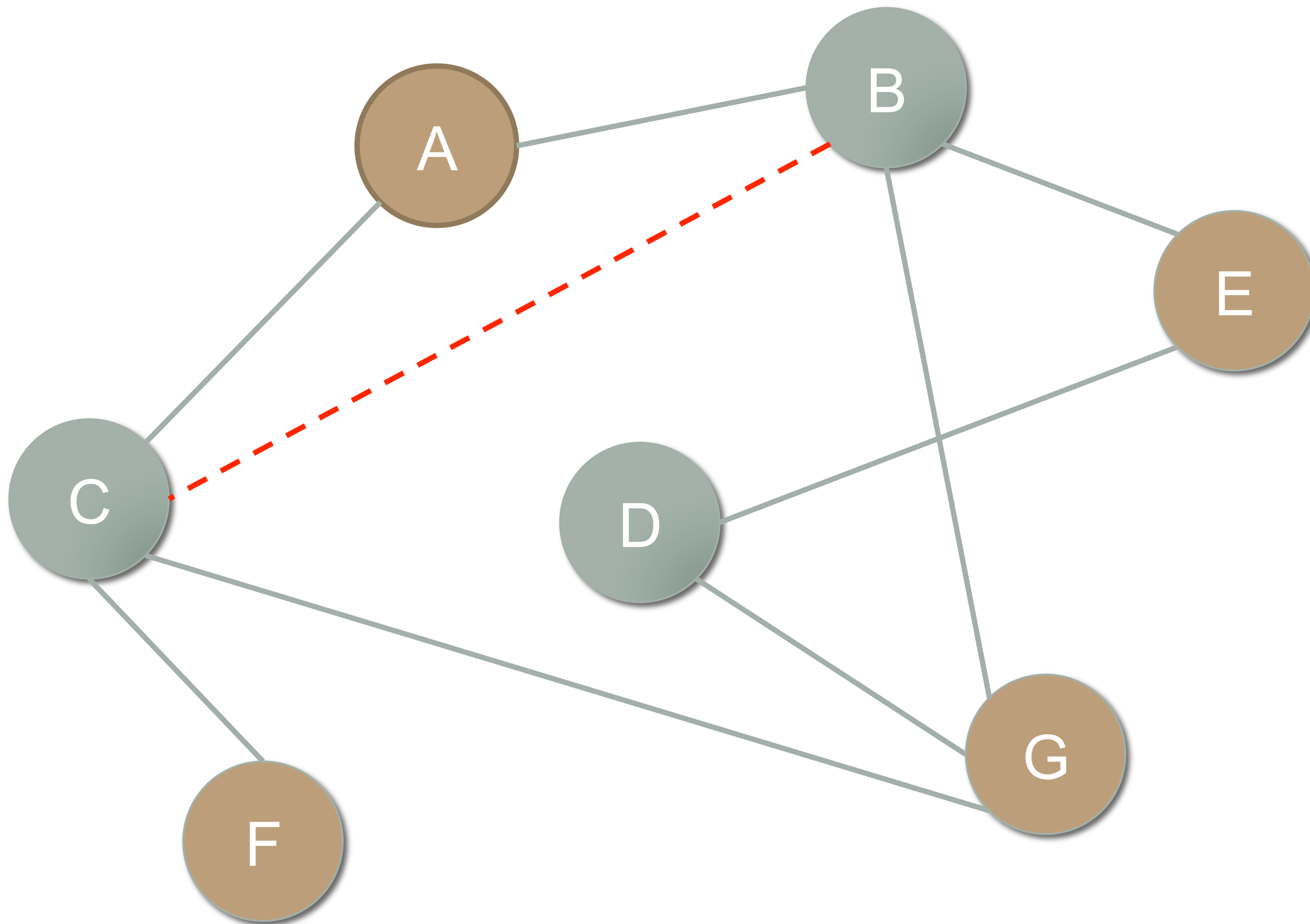
```
GraphVisits<T extends Comparable<T>>
```

Esercizio 2: proprietà

- Un grafo G è *2-colorabile* se ogni nodo può essere colorato di bianco o di rosso in modo che nodi connessi da archi siano colorati con colori distinti.
- Un grafo è bipartito se e solo se è 2-colorabile

Esercizio 2b

- Dati di esempio



Esercizio 3

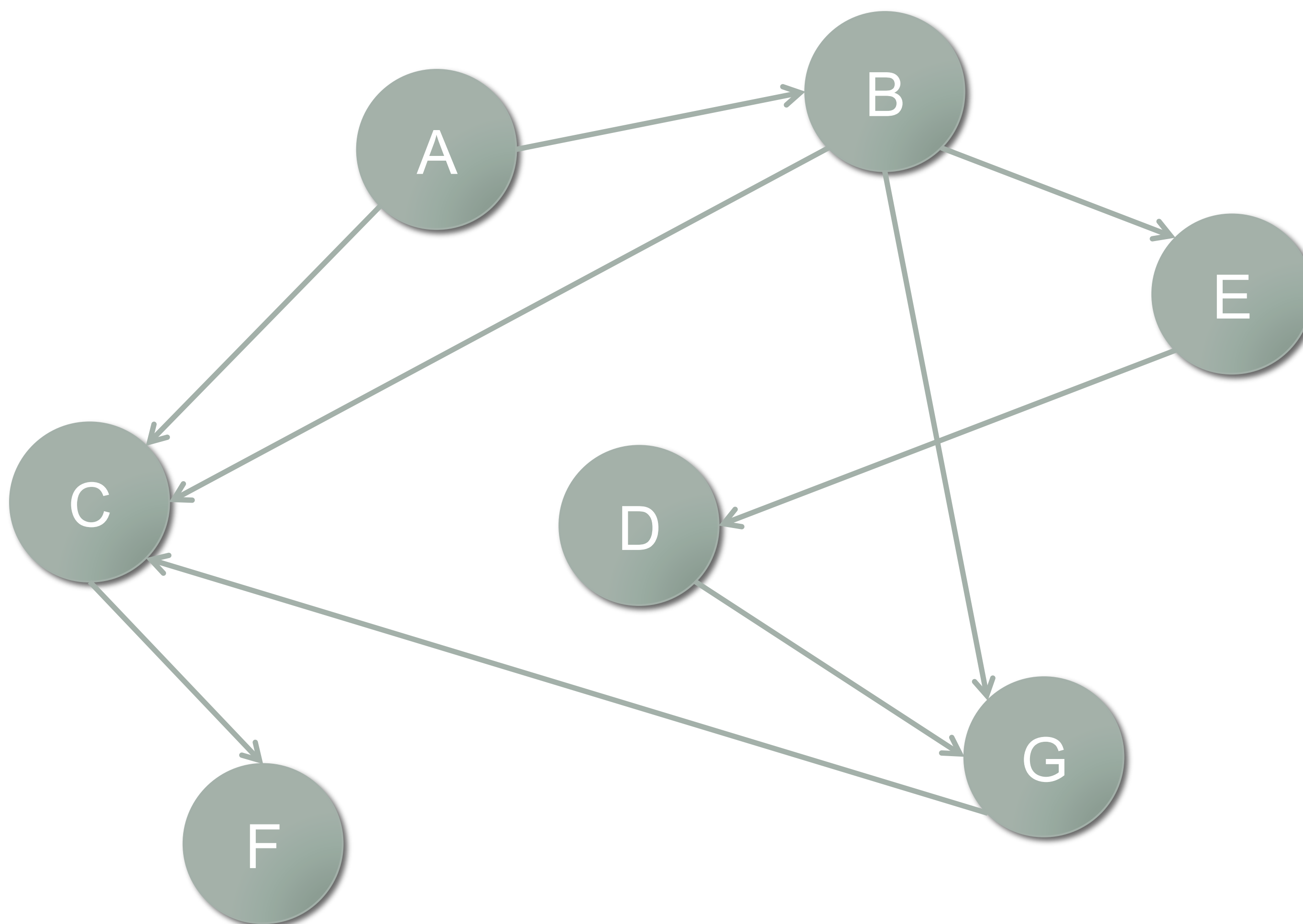
- Implementare un metodo per stampare il cammino BFS da un nodo `u` ad un nodo `v` in un grafo orientato (o un messaggio di errore se il cammino non esiste)

- Aggiungere alla classe `GraphVisits` il metodo:

```
public void printPath(Graph<T> graph,
Node<T> u, Node<T> v)
```

Esercizio 3b

- Dati di esempio



Esercizio 4 (esercizio 9.2 sul libro)

- Scrivere una versione iterativa della DFS (su grafo orientato) dove l'ordine di visita dei nodi e degli archi sia lo stesso di quello della versione ricorsiva.
- Aggiungere alla classe `GraphVisits` due metodi, uno iterativo e uno ricorsivo, che stampano gli archi in ordine di (pre-)visita DFS:

```
public void dfsRec(Graph<T> graph, Node<T> u);

public void dfsIter(Graph<T> graph, Node<T> u);
```

Esercizio 5

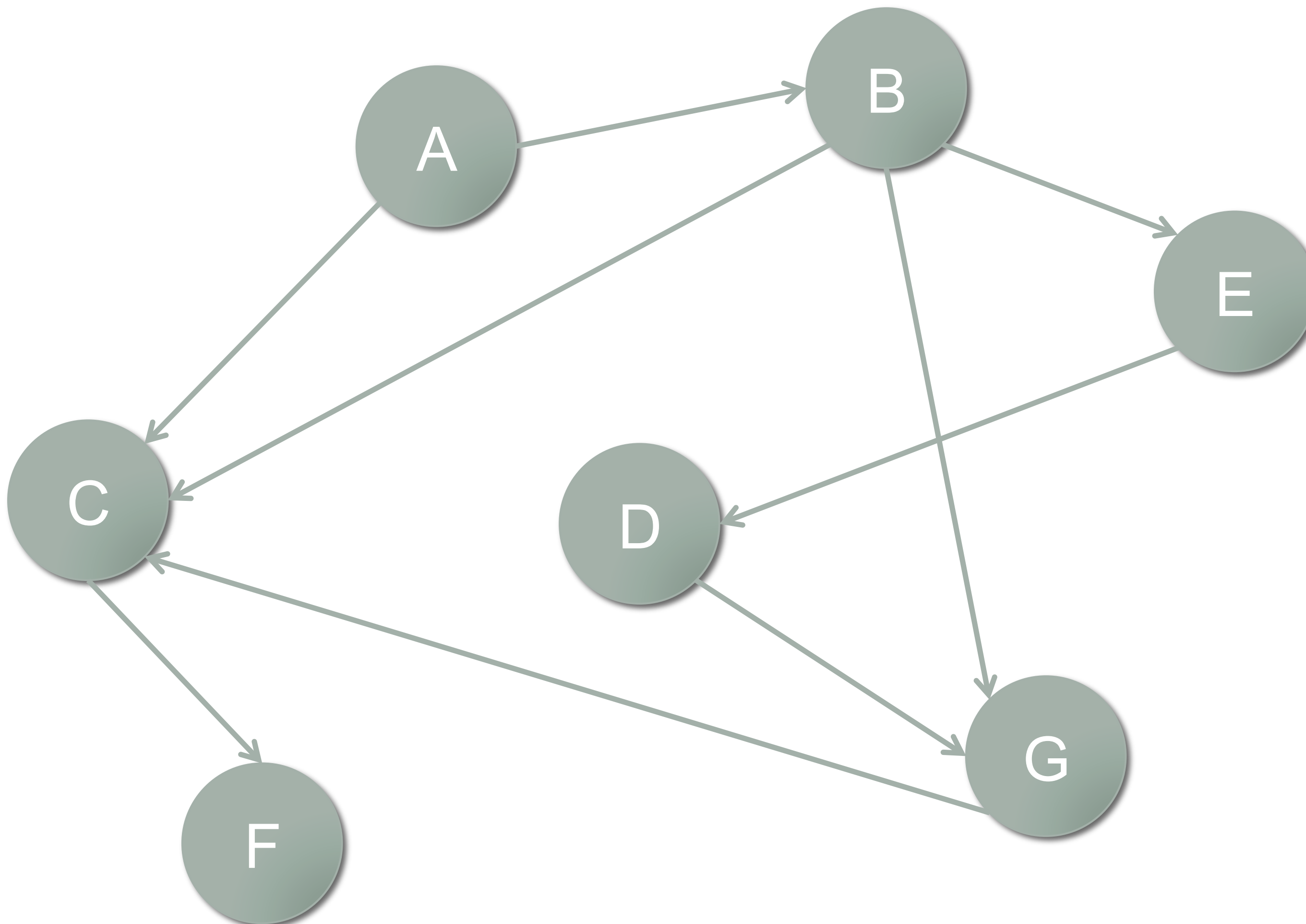
- Implementare un metodo per verificare se un grafo orientato è aciclico oppure no
- Proprietà: un grafo è aciclico se e solo se non esistono archi all'indietro
- Aggiungere alla classe `GraphVisits` il metodo:

```
public Boolean hasCycles(Graph<T> graph,
Node<T> u);
```

- Per semplicità non gestiamo foreste di copertura DFS (cosa si dovrebbe fare altrimenti?)

Esercizio 5b

- Dati di esempio



Esercizio 5c

- Dati di esempio