Laboratorio di Algoritmi e Strutture Dati I

Esercitazione 9

Comunicazioni di servizio

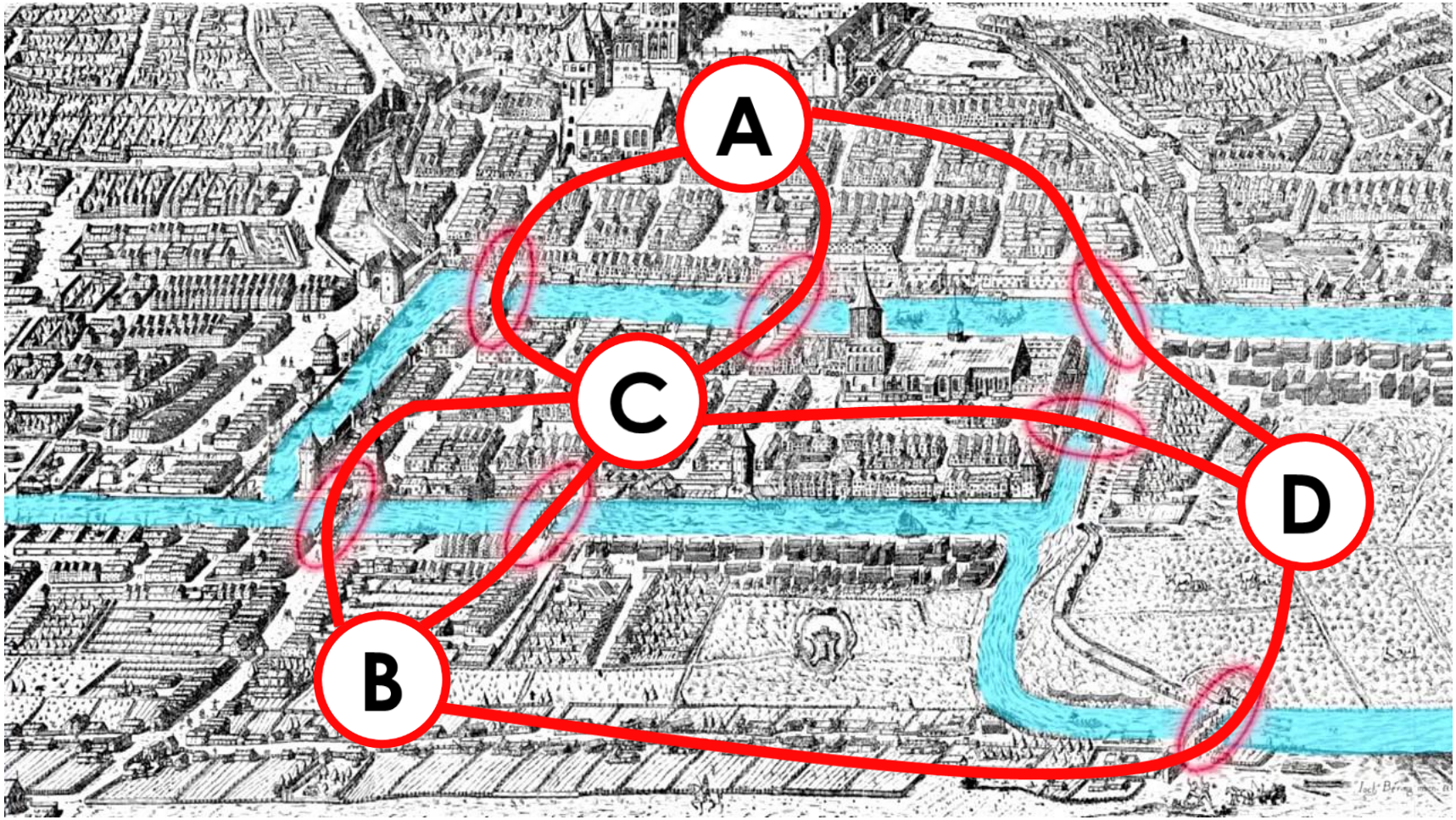
- Martedì **26 Maggio**: secondo test di laboratorio
  - Comprenderà le lezioni di laboratorio dalla 7 alla 10 (inclusa)
  - Dà diritto a 1 punto bonus
  - Prenotazioni aperte su eLearning (scadenza domenica 24 Maggio)
  - Anche chi non ha partecipato al 1° test può comunque fare il 2°

- NB: avete la possibilità di guadagnare un **terzo** punto bonus!
  - Mini progetto + relazione (algoritmi di ordinamento)
  - Dà diritto a 1 punto bonus extra
  - Una settimana di tempo
  - Più informazioni durante le ultime due lezioni

Lezione 9: i ponti di Königsberg

È possibile attraversare tutti i ponti una e una sola volta?

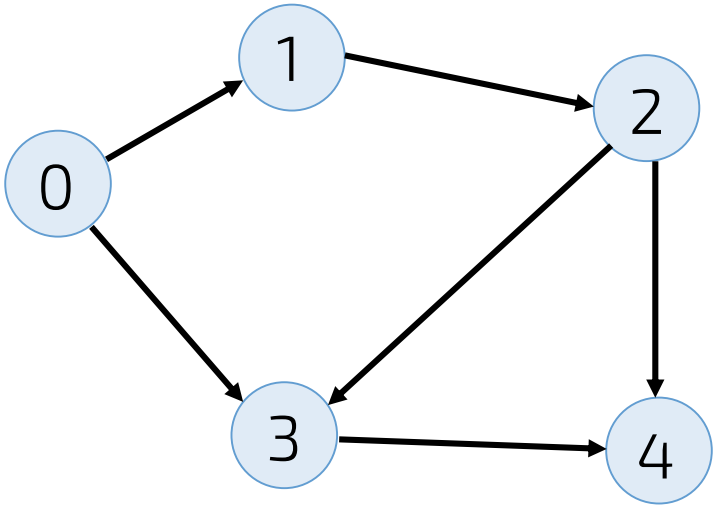
Prima soluzione del problema da parte di Eulero: nasce la teoria dei grafi



Lezione 9: grafi

Un grafo **G** è una struttura dati astratta non lineare composta da:

- Un insieme finito e non vuoto di **nodi**, **N(G)**;
- Un insieme finito (anche vuoto) di **lati** (archi), **L(G)**.



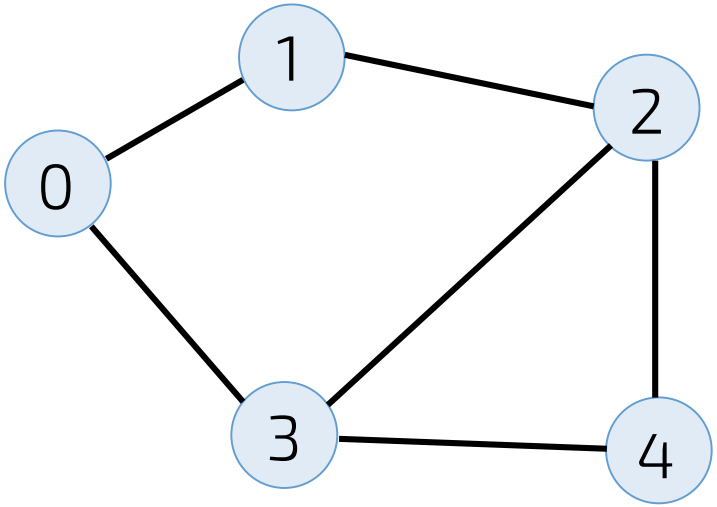
5

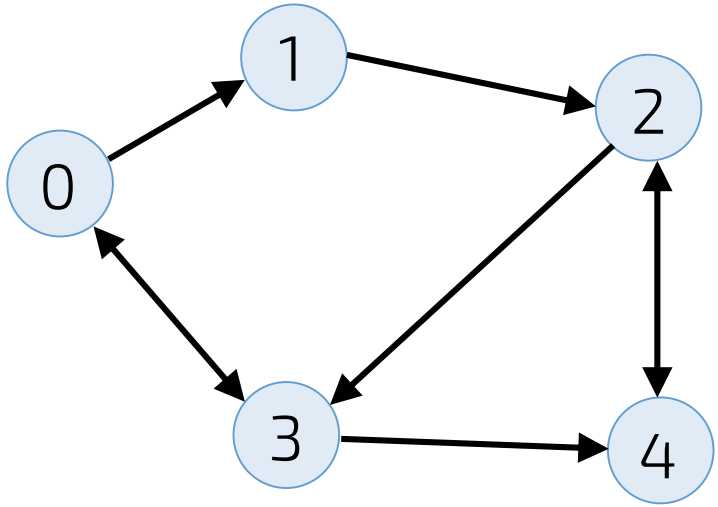
N(G) = { 0, 1, 2, 3, 4, 5 }

L(G) = { (0,1); (0,3); (1,2); (2,3);
(2,4); (3,4) }

Lezione 9: grafi, orientati e non orientati

- Un grafo **non orientato** è un grafo in cui la coppia di vertici che rappresentano un lato qualsiasi non è ordinata.
- Un grafo **orientato** è un grafo in cui un lato è rappresentato da una coppia ordinata di vertici. In questo caso si parla di coda e testa del lato.
  - Esempio: nell'arco (0-1), 0 rappresenta la coda e 1 la testa.

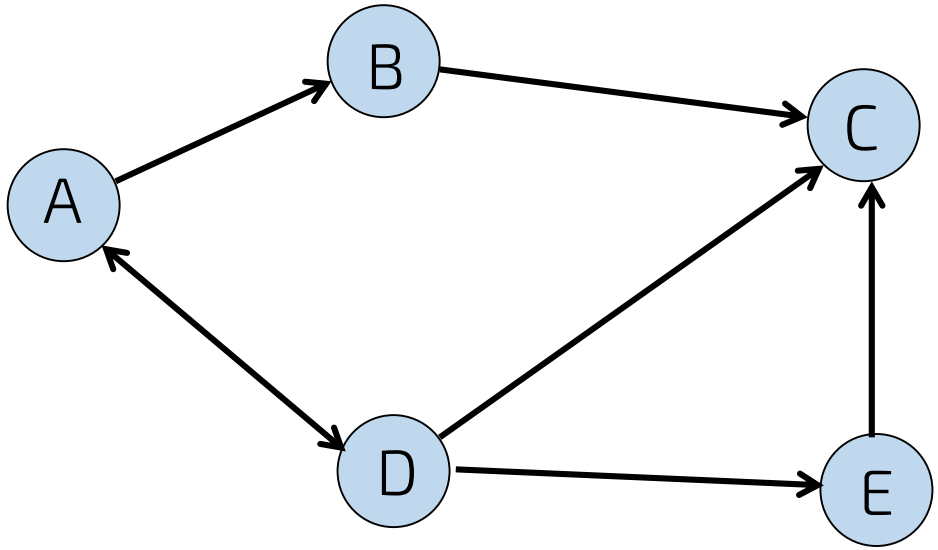




Lezione 9: grafi, adiacenze

Due vertici di un grafo si dicono **adiacenti** se esiste un lato che li unisce.

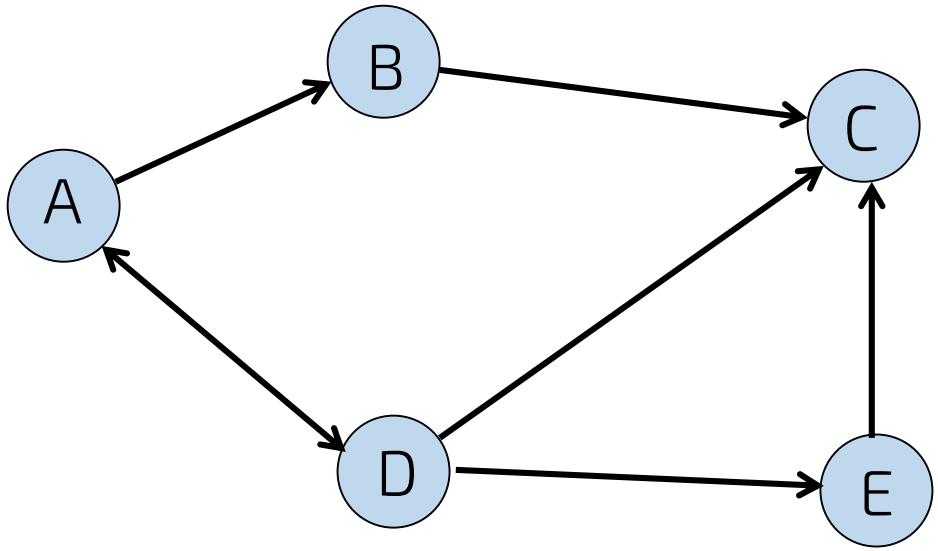
- Nel caso di un grafo orientato, l'adiacenza è stabilita dall'orientamento degli archi.



- B è adiacente ad A
- C è adiacente a B, a D e ad E
- A è adiacente a D e viceversa
- B **NON** è adiacente a D **NÉ** a C

Lezione 9: grafi, incidenze

Un lato (V1, V2) (o <V1,V2> o <V2,V1>) si dice **incidente** nei vertici V1 e V2.



- < A,B > è incidente da A a B
- < A,D > è incidente da A a D
- < D,A > è incidente da D a A
- ecc…

Lezione 9: grafi e possibili rappresentazioni

Esistono diverse possibili rappresentazioni dei grafi:

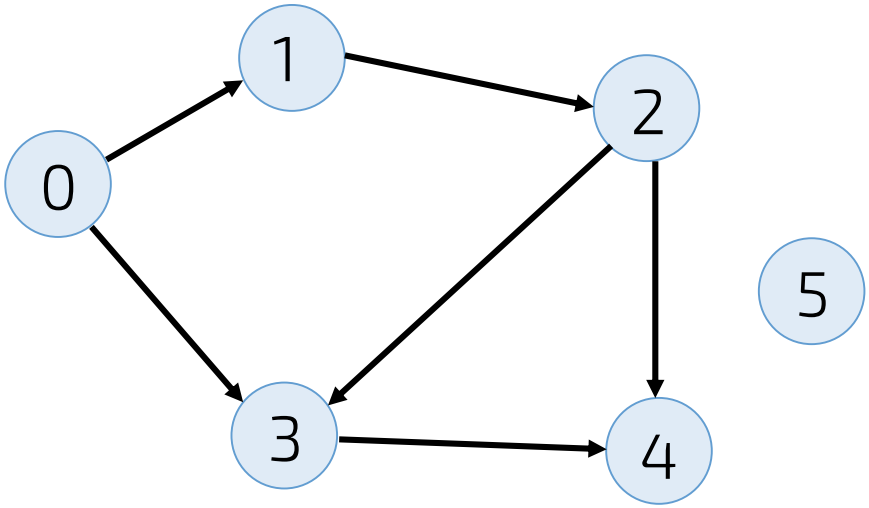
- Matrici di adiacenza
- Liste di adiacenza
  - Concatenate
  - Sequenziali
- Multiliste di adiacenza

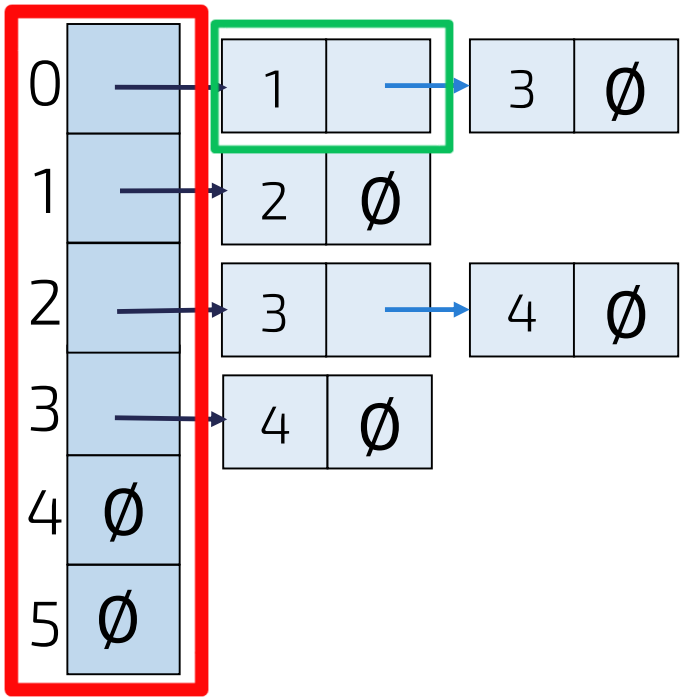
In laboratorio implementerete le **liste di adiacenza concatenate**: vediamo come.

Lezione 9: grafi, liste di adiacenza

- Si usa un array di puntatori a nodo per rappresentare gli id dei nodi del grafo
- Per ogni nodo si costruisce una lista concatenata contenente i lati in uscita dal nodo ordinati per ID
- Gli elementi dell'array sono i puntatori alle liste di adiacenze

```
typedef struct node
{
    int vertex_id;
    struct node* link;
}Node;

Node* grafo[MAX_VERTICI];
```

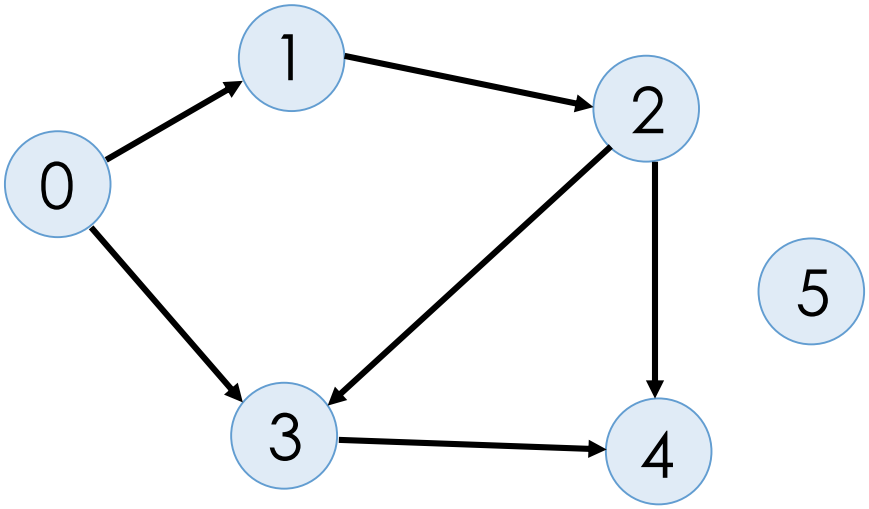


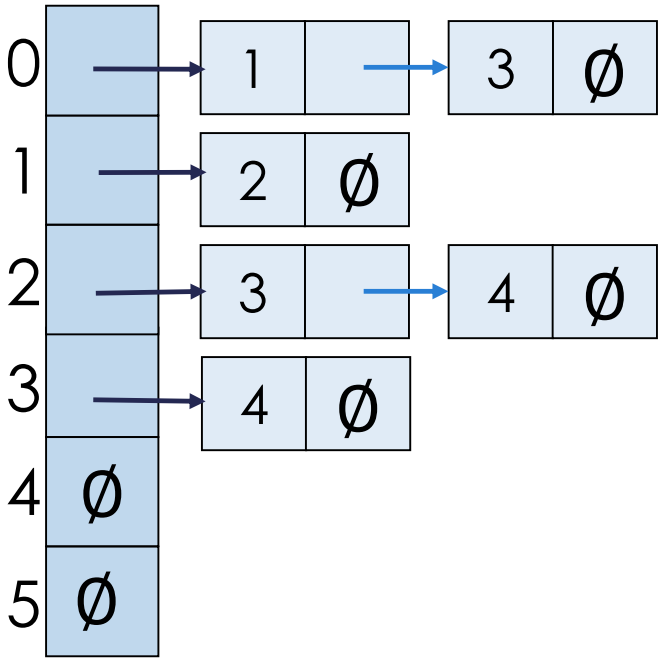


Lezione 9: visita in profondità (DFS)

Dobbiamo utilizzare una struttura dati ausiliaria: stack (solo idealmente)

- Si definisce un nodo di partenza da cui iniziare la visita
- Quando visitiamo un nodo lo mettiamo nello stack (push) e visitiamo ricorsivamente il primo nodo adiacente non ancora visitato
- Dopo che abbiamo visitato tutte le adiacenze di un nodo, lo togliamo dallo stack (pop)





Sequenza di vertici raggiungibili da 0 in profondità: 0 – 1 – 2 – 3 – 4

Lezione 9: visita in profondità (DFS)

Dobbiamo utilizzare una struttura dati ausiliaria: stack (solo idealmente)

- Si definisce un nodo di partenza da cui iniziare la visita;
- Quando visitiamo un nodo lo mettiamo nello stack (push) e visitiamo ricorsivamente il primo nodo adiacente non ancora visitato;
- Dopo che abbiamo visitato tutte le adiacenze di un nodo, lo togliamo dallo stack (pop).

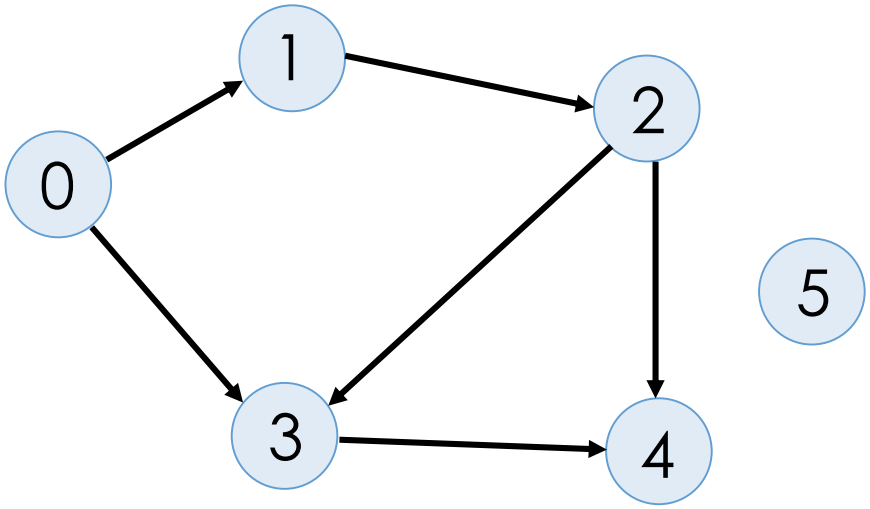
- NB#1: non c'è bisogno di implementare lo stack perché lo simuliamo con una serie di chiamate ricorsive.
- NB#2: abbiamo necessità di un array "visitato" dove teniamo traccia di quali nodi abbiamo già visitato.

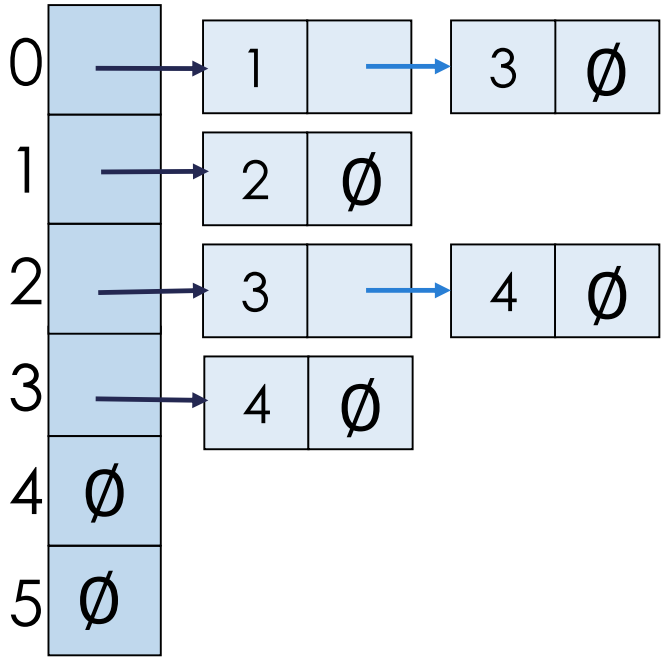
```
algoritmo visita_profondita(intero id) → void

visita nodo id
marca id come visitato
tmp punta al campo lista_adj di grafo[id]

while (tmp ≠ NULL) do
    id_next = vertex di tmp
    if (nodo id_next NON ancora visitato) then
          visita_profondita(id_next)

    tmp punta a tmp->next
```

Lezione 9: visita in ampiezza (BFS)

Dobbiamo utilizzare una struttura dati ausiliaria: coda (reale)

- Si definisce un nodo di partenza da cui iniziare la visita;
- Quando visitiamo un nodo lo inseriamo nella coda (push) e visitiamo **tutta** la sua lista di adiacenze;
- Quando abbiamo visitato **tutta** la sua lista di adiacenze lo eliminiamo dalla coda (pop).





Sequenza vertici raggiungibili da 0 in ampiezza: 0 – 1 – 3 – 2 – 4

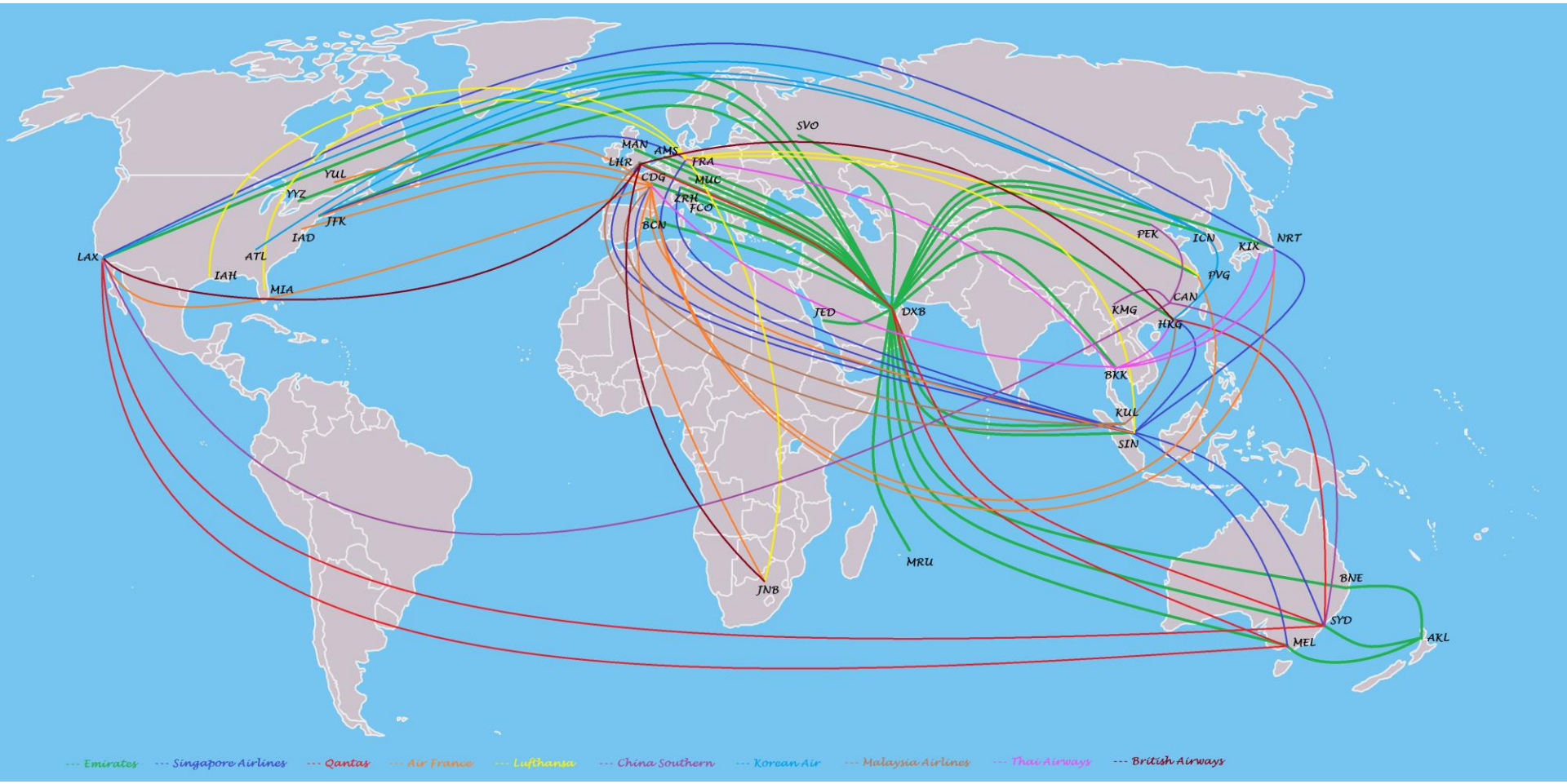
Lezione 9: visita in ampiezza (BFS)

Dobbiamo utilizzare una struttura dati ausiliaria: coda (reale)

- Si definisce un nodo di partenza da cui iniziare la visita;
- Quando visitiamo un nodo lo inseriamo nella coda (push) e visitiamo tutta la sua lista di adiacenze;
- Quando abbiamo visitato tutta la sua lista di adiacenze lo eliminiamo dalla coda (pop).
- NB#1: possiamo (**dobbiamo**) riciclare TUTTA l'implementazione della struttura dati **Coda** dall'esercitazione 5.
- NB#2: abbiamo necessità di un array "visitato" dove teniamo traccia di quali nodi abbiamo già visitato.

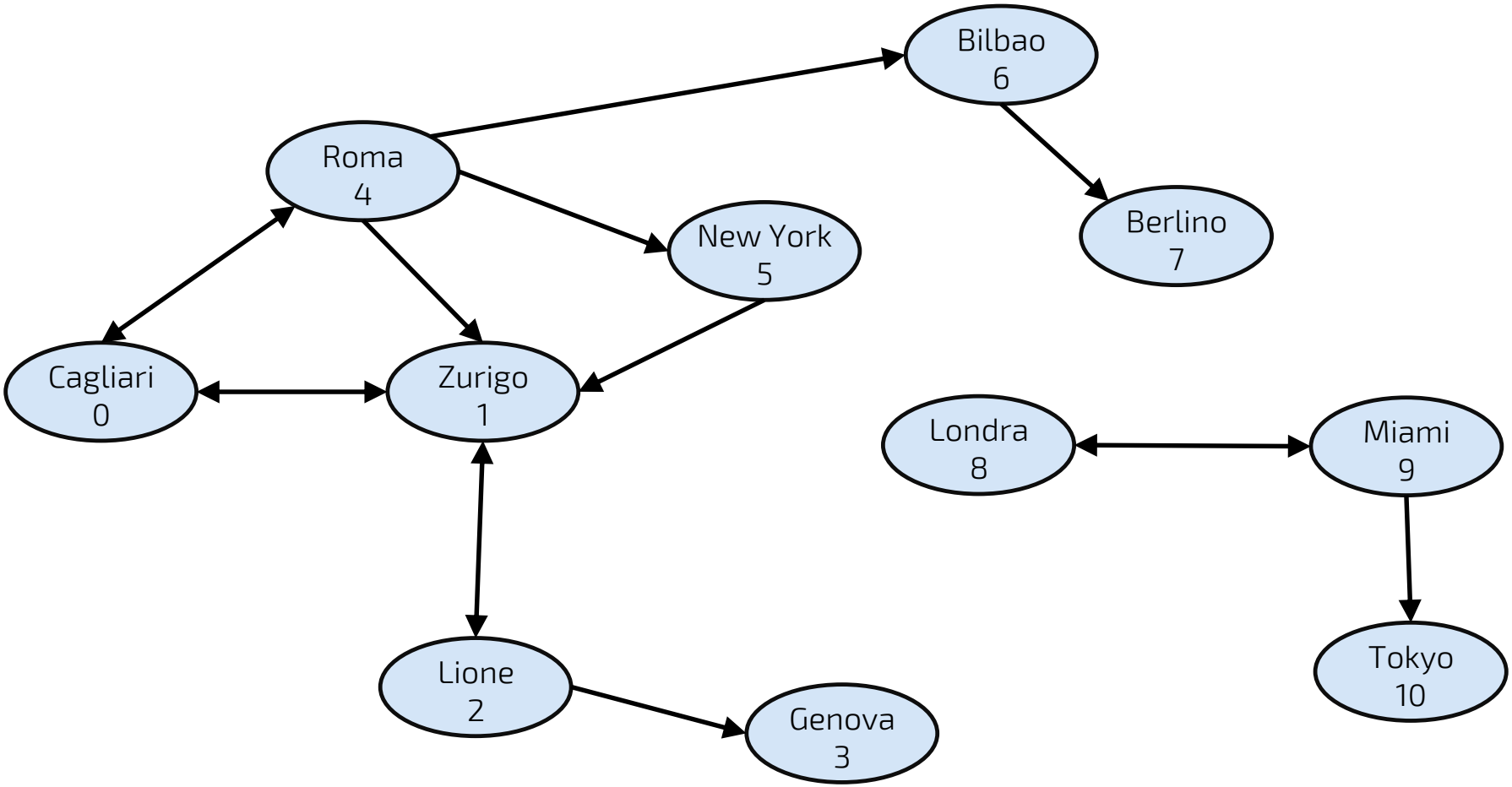
```
algoritmo visita_ampiezza(intero id) → void

visita il nodo id
marca id come visitato

inizializza la coda C
pushQueue(C, id)

while ( not isEmptyQueue(C) ) do
    i ← popQueue(C)
    tmp punta al campo lista_adj di grafo[i]

    while (tmp ≠ NULL) do
          if (tmp non visitato) then
               pushQueue(C, tmp->vertex_id)
               visita tmp
               marca tmp come visitato
         tmp punta a tmp->next
```

Esercizio 9: rotte aeree



Esercizio 9: grafo delle città

Vogliamo rappresentare, mediante un **grafo orientato**, le connessioni tra diverse città offerte da una certa compagnia aerea. Ad esempio:



Esercizio 9: grafo delle città

Ogni città è rappresentata dalla seguente struttura:

```
typedef struct
{
    char nome[DIM];
    int  abitanti;
    char nazione[DIM];
    Node* lista_adj;
}Citta;
```

Un nodo della lista di adiacenza del grafo è rappresentato dalla seguente struttura:

```
typedef struct node
{
    int vertex_id;
    struct node* link;
}Node;
```

Esercizio 9: progetto base

- Andate su eLearning e copiate il codice del progetto base per l'esercitazione odierna.
- Create un nuovo progetto su Clion e incollateci il codice.
- Troverete già definite le varie strutture e diverse funzioni, tra cui:
  - *carica_citta_test(Citta grafo[])* che inserisce, all'interno della lista che rappresenta il grafo, le varie città delle slide precedenti (nelle stesse posizioni, id, dell'esempio).
  - *carica_grafo_test_orientato(...)* che crea la lista di adiacenze relativa al grafo delle città mostrato in esempio. NB: funzionerà solo dopo aver implementato la funzione *crea_arco(...)*.
  - *carica_grafo_test_non_orientato(...)* che crea la lista di adiacenze relativa al grafo delle città mostrato in esempio, nella sua versione NON orientata. NB: funzionerà solo dopo aver implementato la funzione *crea_arco(...)*.
  - I prototipi di tutte le funzioni da implementare (escluse quelle per la gestione della coda, che potete prendere direttamente dall'esercitazione 5).

Esercizio 9: grafo delle città

- Dovrete implementare, seguendo lo schema dato dai prototipi già presenti nel file, le seguenti funzioni:
  - *aggiungi_nodo(...)*: aggiunge una città nel grafo chiedendo i dati all'utente.
  - *crea_arco(...)*: crea un arco tra due nodi esistenti (orientato).
  - *stampa_lista_adiacenze(...)*: stampa la lista di adiacenze che rappresenta l'intero grafo.
  - *DFS(...)*: stampa la visita in profondità del grafo, dato un nodo di partenza.
  - *BFS(...)*: stampa la visita in ampiezza del grafo, dato un nodo di partenza.
  - *cancella_arco_orientato(...)*: cancella un arco dalla lista di adiacenza.
  - *cancella_arco_non_orientato(...)*: cancella entrambi gli archi dalla lista di adiacenza.
  - *componenti_connesse(...)*: stampa le varie componenti connesse del grafo (NB: usare un grafo NON orientato!).
    - Hint: Sfruttare una delle visite implementate finché non si sono visitati tutti i nodi del grafo.
  - Un menu che consenta di scegliere quale operazione effettuare.

Esercizio 9: lista di adiacenze

La stampa della lista di adiacenza dovrà dare il seguente risultato :

Cagliari -> Zurigo -> Roma

Zurigo -> Cagliari -> Lione

Lione -> Zurigo -> Genova

Genova

Roma -> Cagliari -> Zurigo -> New York -> Bilbao

New York -> Zurigo

Bilbao -> Berlino

Berlino

Londra -> Miami

Miami -> Londra -> Tokyo

Tokyo

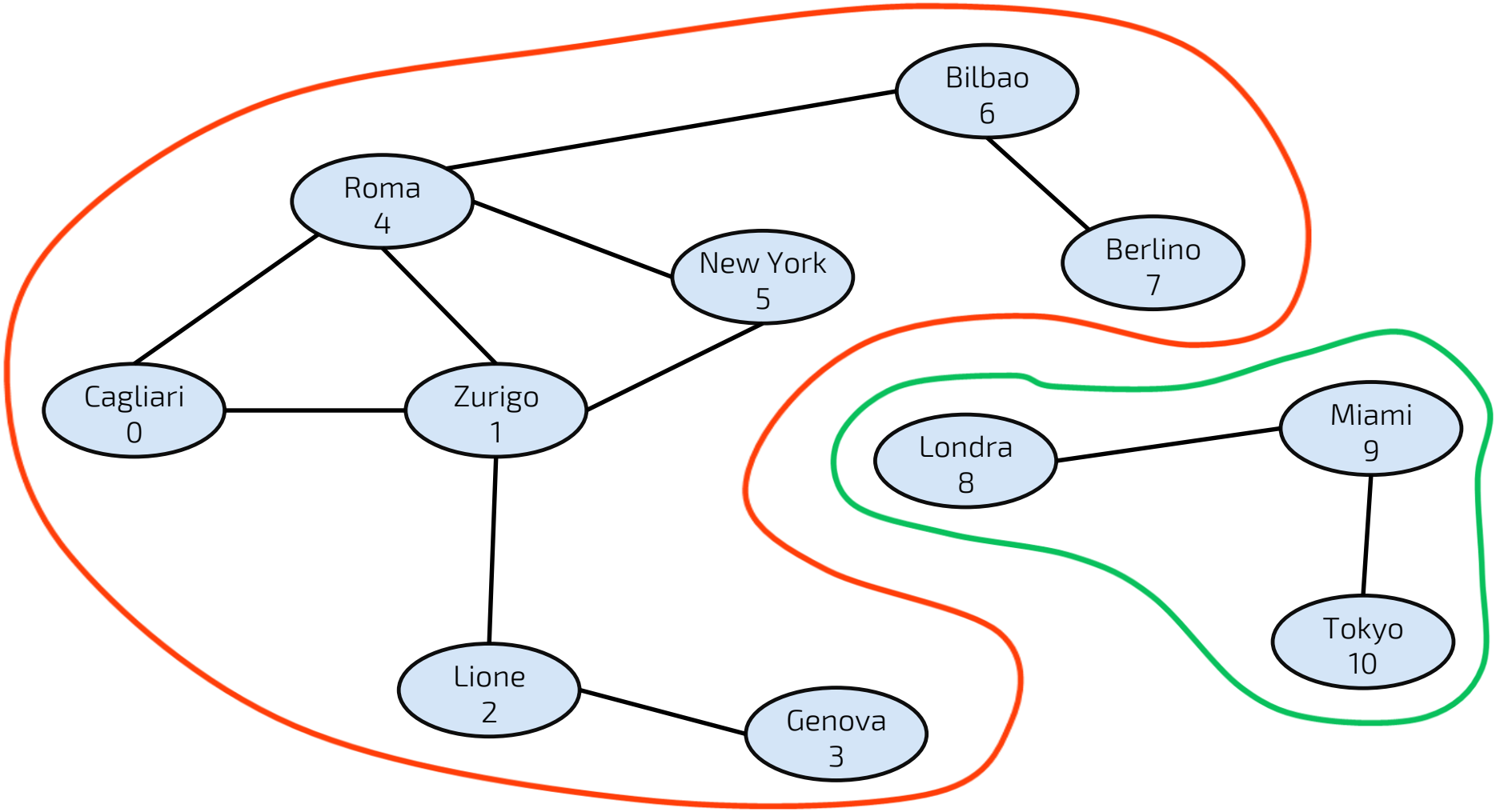
Esercizio 9: visite

- Esempio di visita in profondità a partire da Cagliari:
  - Cagliari, Zurigo, Lione, Genova, Roma, New York, Bilbao, Berlino.
- Esempio di visita in profondità a partire da Roma:
  - Roma, Cagliari, Zurigo, Lione, Genova, New York, Bilbao, Berlino.

- Esempio di visita in ampiezza a partire da Cagliari:
  - Cagliari, Zurigo, Roma, Lione, New York, Bilbao, Genova, Berlino.
- Esempio di visita in ampiezza a partire da Roma:
  - Roma, Cagliari, Zurigo, New York, Bilbao, Lione, Berlino, Genova.

Esercizio 9: grafo delle città

Ricerca delle componenti connesse:



end().