# Un problema di consegne

© Gualtiero Grassucci, 2019 – gualtiero.grassucci@liceograssilatina.org
Liceo scientifico G.B. Grassi di Latina

Research in Action (RiA) – www.researchinaction.it

# Il problema del postino cinese



- Problema posto da Mei-Ku Kwan nel 1962:

  Costruire un circuito di lunghezza minima che attraversi tutti gli archi di un grafo

- Il nome deriva dal fatto che il percorso ottimale per la consegna della posta richiede di passare per ogni strada di competenza una sola volta
- Il problema è *banale* in un grafo euleriano
- In grafi non-euleriani diventa una questione davvero interessante
- Ne abbiamo *tirato fuori* tre/quattro laboratori che possono essere proposti in successione o affrontati separatamente

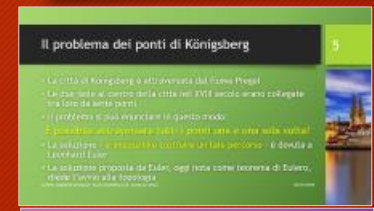
Si può costruire un circuito euleriano in un grafo dove questo circuito non c'è?

Il primo problema e il primo vero risultato della teoria dei grafi e della topologia

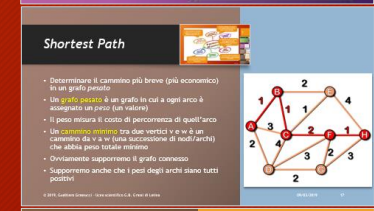

Il teorema di Euler si applica solo a grafi connessi, come decidere se un grafo è connesso?

Un circuito euleriano risolve immediatamente il problema del postino

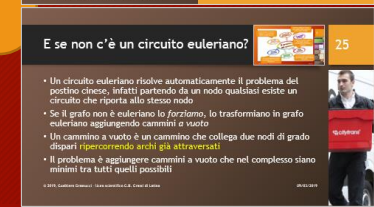
La ricerca della via più breve sarà essenziale, come vedremo più avanti











# Il problema dei ponti di Königsberg



- La città di Königsberg è attraversata dal fiume Pregel
- Le due isole al centro della città nel XVIII secolo erano collegate tra loro da sette ponti
- Il problema si può enunciare in questo modo:

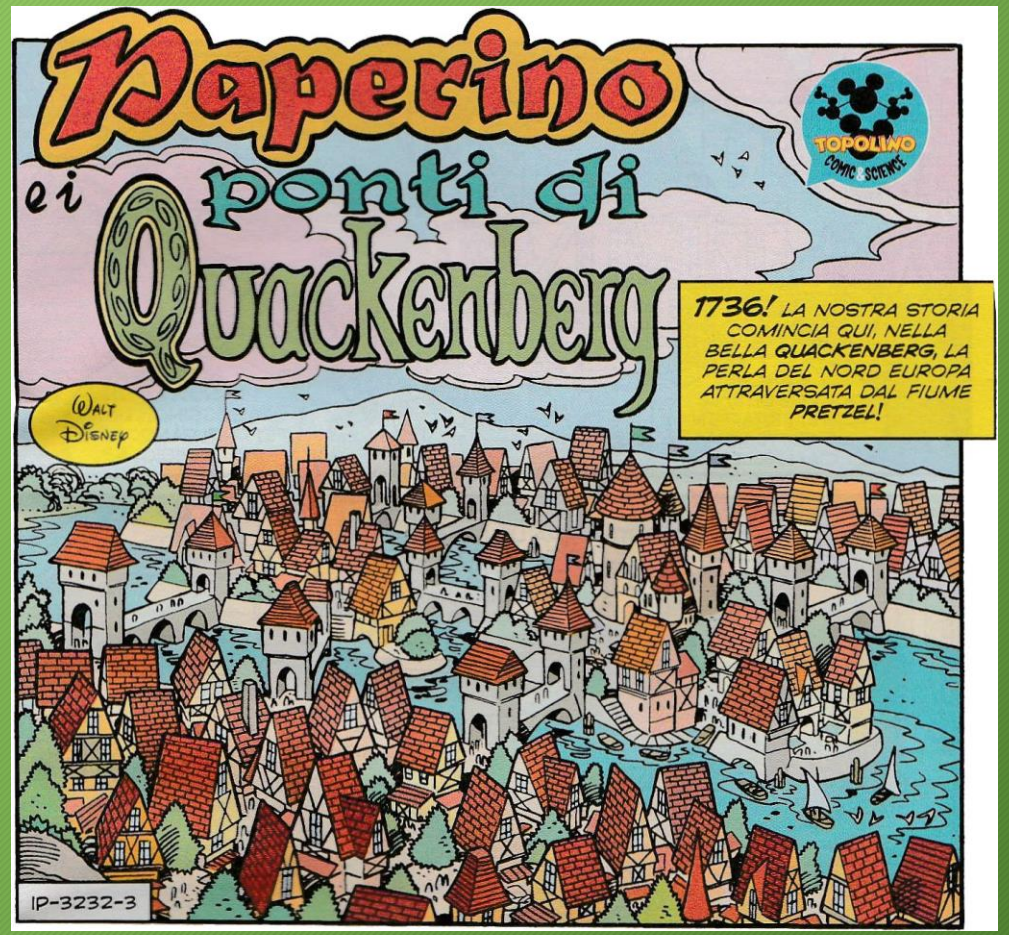
**È possibile attraversare tutti i ponti una e una sola volta?**

- La soluzione - è impossibile costruire un tale percorso - è dovuta a Leonhard Euler
- La soluzione proposta da Euler, oggi nota come teorema di Eulero, diede l'avvio alla topologia

# I ponti di Quackenberg

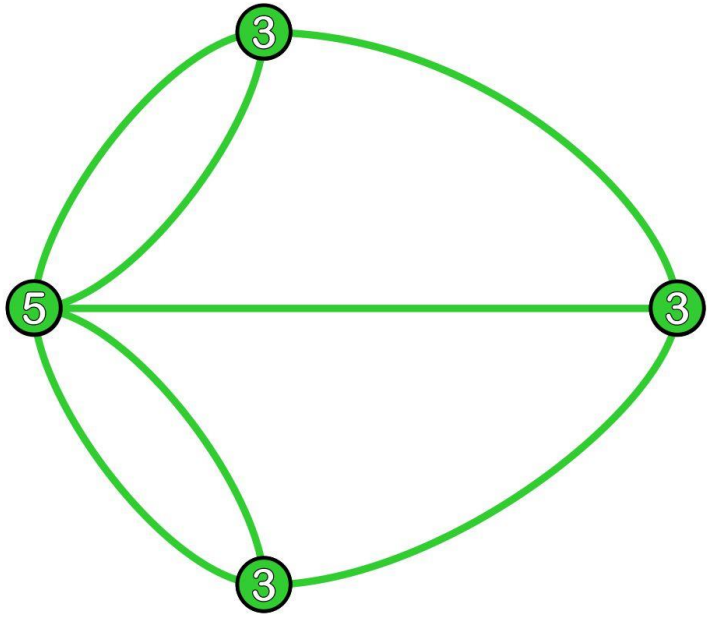


- Grazie alla collaborazione tra la Panini/Disney e la collana *Comic & Science* del CNR su Topolino n. 3232 è stata pubblicata la storia *Paperino e i ponti di Quackenberg*
- Racconta la storia e spiega la soluzione proposta da Euler (e c'è anche un esercizio)
- Soggetto di Alberto Saracco e Francesco Artibani
- Disegni di Marco Mazzarello



# Un modello per il problema dei ponti di …






# Il teorema di Euler



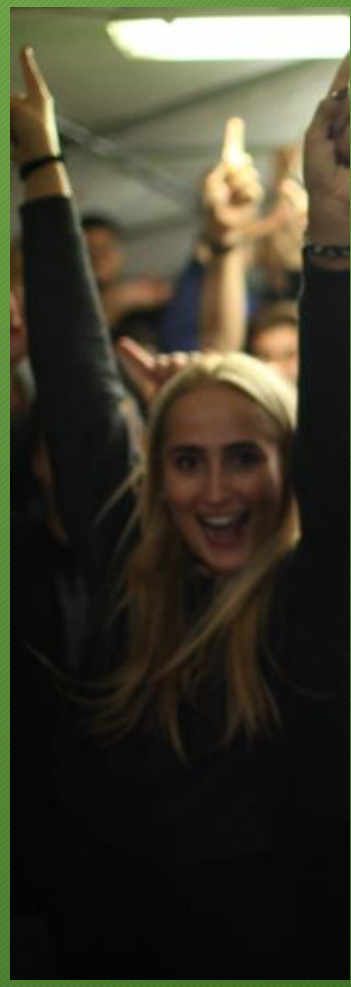

- In un grafo connesso esiste un cammino euleriano (un cammino che percorre ogni arco una e una sola volta) se e solo se i nodi di grado dispari sono esattamente due
  - Il cammino deve necessariamente partire da uno dei nodi di grado dispari e terminare nell'altro
- Se tutti i nodi sono di grado pari esiste un circuito euleriano
  - Il circuito può iniziare in un nodo qualsiasi e termina, ovviamente, nello stesso nodo

Il teorema non fornisce un algoritmo per costruire il cammino/circuito euleriano, nel caso esista

# Il laboratorio



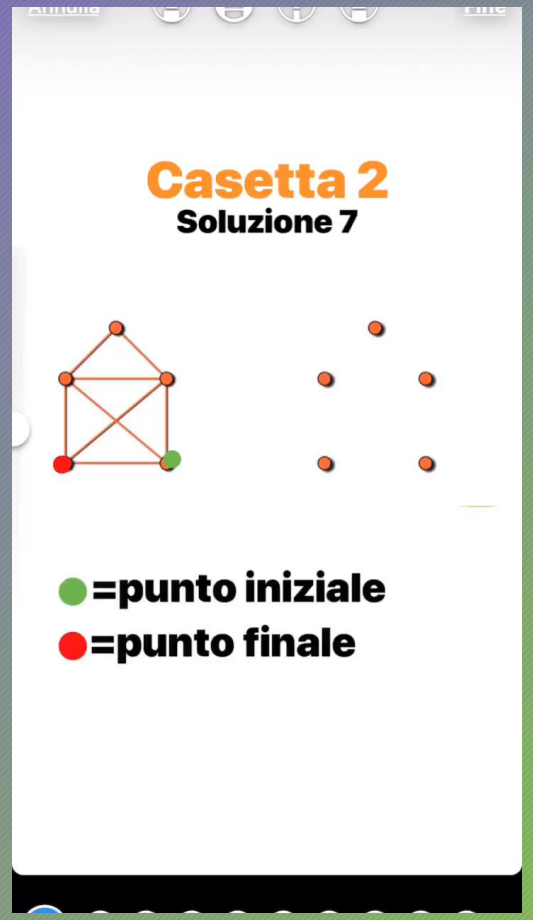

- Il problema dei ponti di Königsberg
  - Problema difficile, dopo molti tentativi non si intravede una soluzione
- Grafi più semplici
  - Concentrare l'attenzione non sulla soluzione, ma su le proprietà comuni alle varie soluzioni
- Generalizzare: il teorema di Euler
- Casi particolari: affinare l'enunciato del teorema
- Di nuovo il problema dei ponti: il teorema è lo strumento per risolverlo

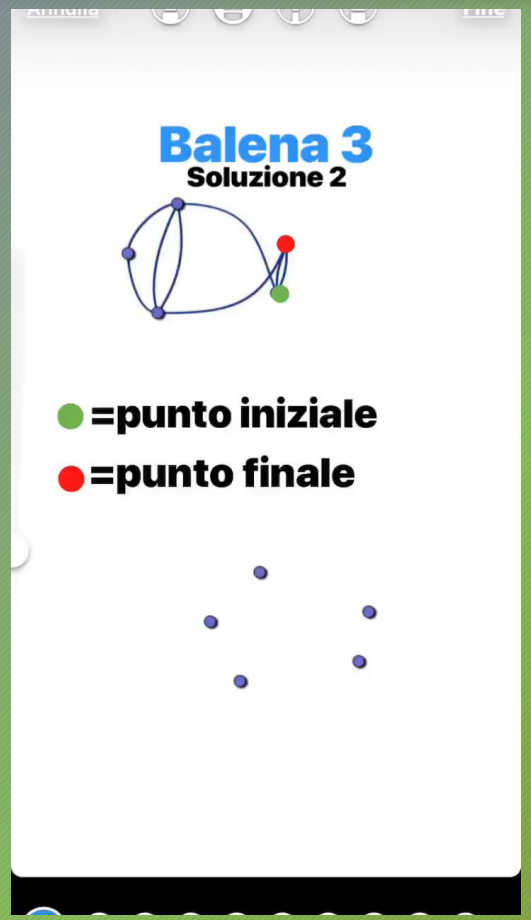


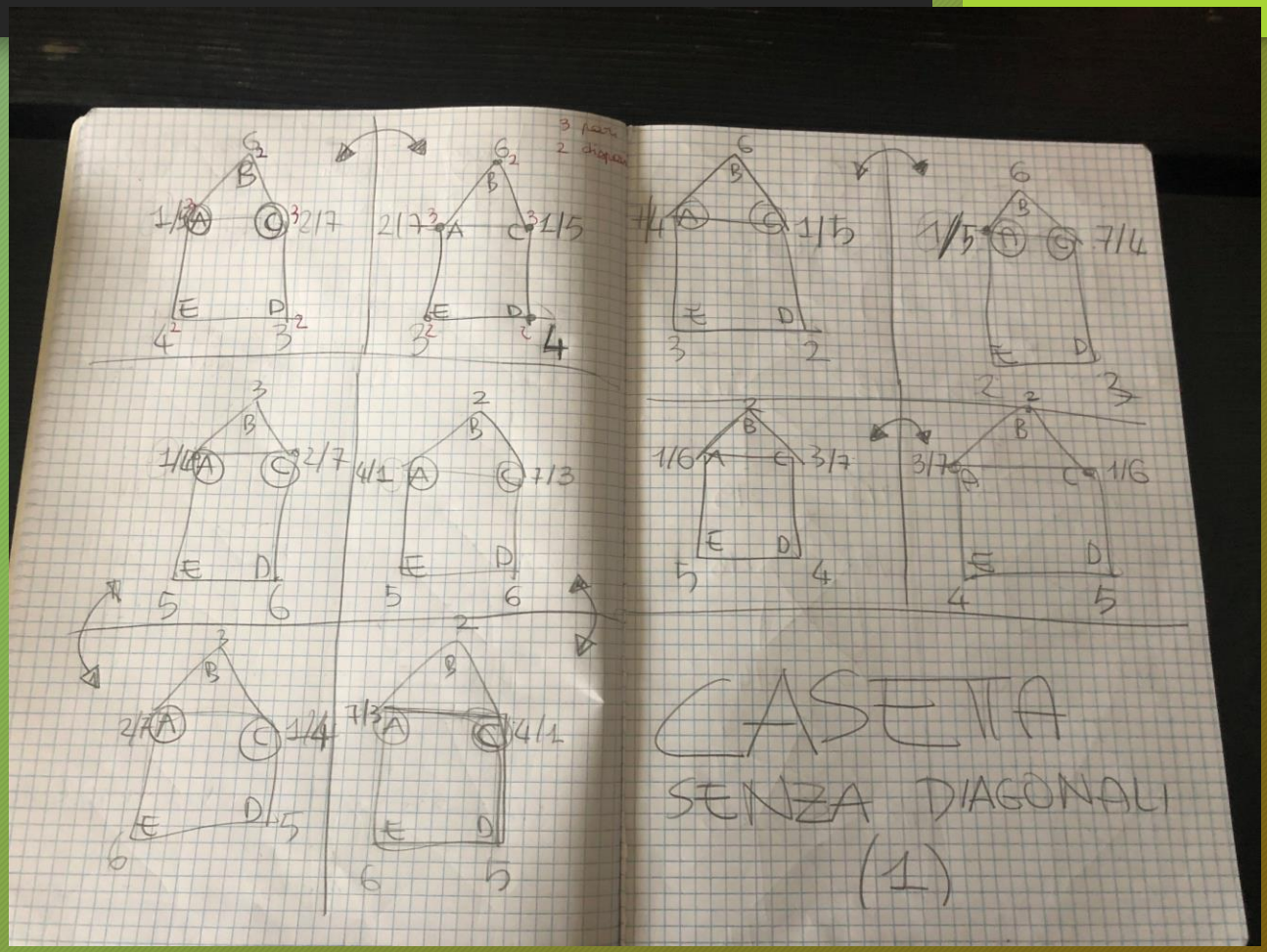
L'obiettivo del laboratorio è quello di guidare gli studenti alla scoperta del teorema e, se possibile, a una sua giustificazione (se non a una dimostrazione)

# All’opera …




Casetta 2
Soluzione 7
=punto iniziale
=punto finale

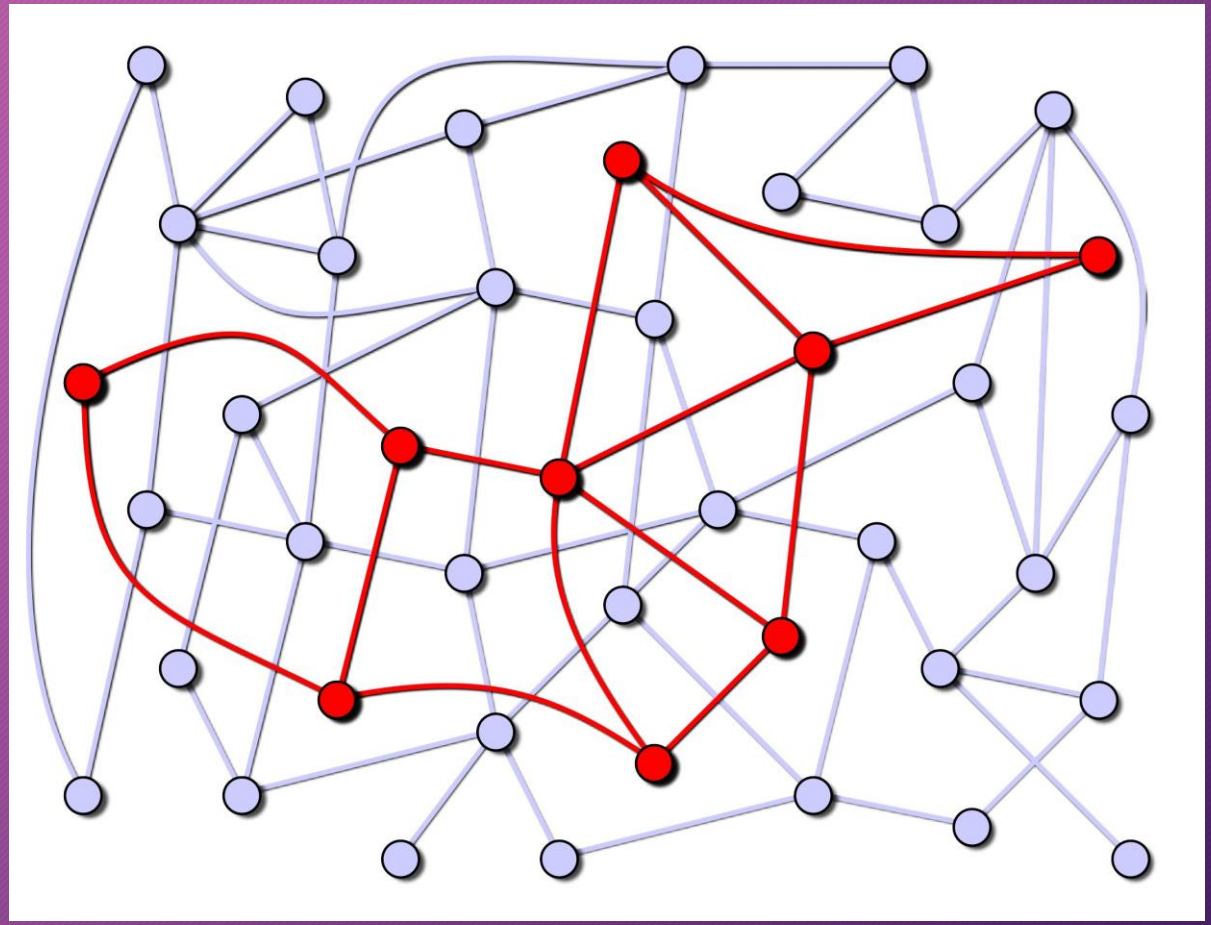

Balena 3
Soluzione 2
=punto iniziale
=punto finale


CASETTA
SENZA DIAGONALI
(1)

# Grafo connesso: definizione



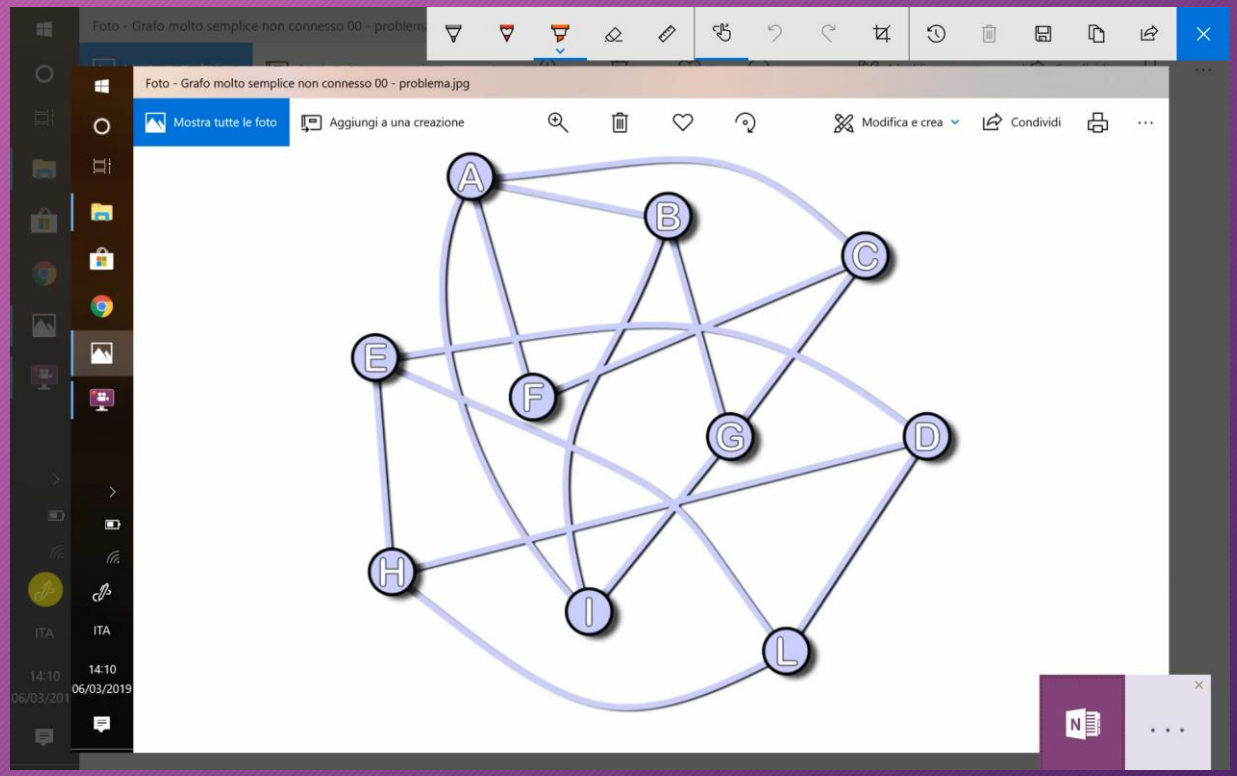
- Il teorema di Euler si applica solo a grafi connessi
- Un grafo *G(V, E)* si dice connesso se esiste un cammino per ogni coppia *(v, w)* di vertici del grafo
- In altre parole, se è possibile determinare una successione di nodi/archi nel grafo tali che il primo nodo sia *v* e l'ultimo sia *w*



# Grafo non connesso: esempio



- Applichiamo la procedura per determinare se il grafo mostrato è connesso
- Scegliamo un nodo qualsiasi e inseriamo tutti i nodi a esso collegati tra quelli possibili
- Ripetiamo la procedura per ciascun nodo possibile
- Se al termine ci sono ancora nodi da visitare il grafo non è connesso!



# Grafo connesso: la procedura

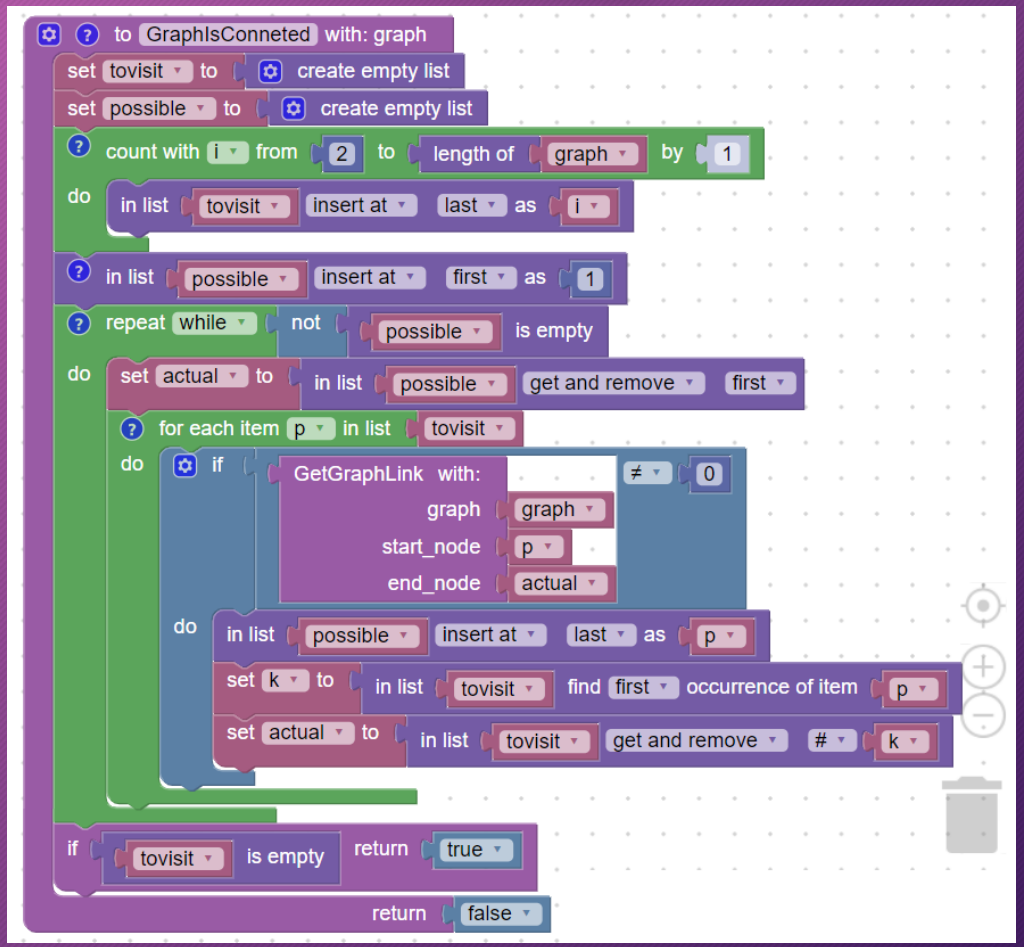


```
Input: il grafo da analizzare
inserisci tutti i nodi nella lista davisitare
aggiungi davisitare[1] a possibili[1]
elimina davisitare[1] dalla lista davisitare
finchè ci sono elementi in possibili
    attuale <- possibili[1]
    elimina possibili[1] dalla lista possibili
    per ogni elemento i in davisitare
        se i è collegato a attuale allora
            elimina i dalla lista davisitare
            aggiungi i alla lista possibili
se davisitare è vuoto allora
            il grafo è connesso
```

# La procedura implementata con Blockly



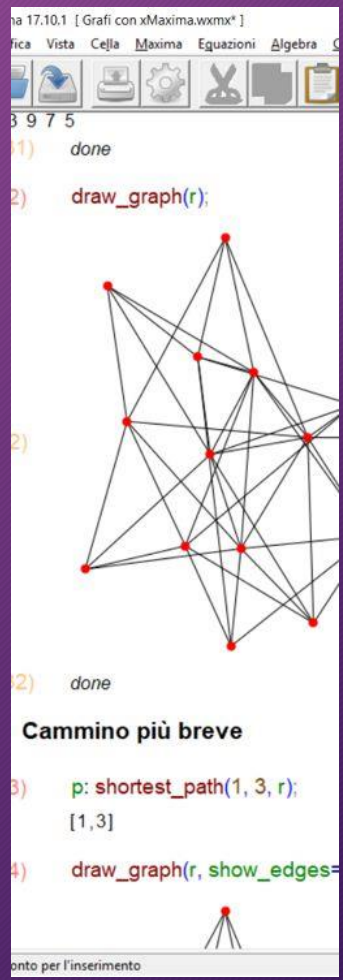
```
def GraphIsConneted(graph):
    # Aggiunge tutti i nodi tranne il primo alla lista dei nodi # da visitare
    # (nodi ancora non raggiunti da nessun cammino)
    i_end = float(len(graph))
    for i in (2 <= i_end) and upRange(2, i_end, 1) or downRange(2, i_end, 1):
        tovisit.append(i)
    # La lista possibili contiene via via i nodi collegati al nodo attuale ma
    # che sono ancora da visitare. Inizialmente contiene solo il primo nodo.
    possible.insert(0, 1)
    # Controlla i cammini a partire dai nodi nella lista possibili.
    while not not len(possible):
        actual = possible.pop(0)
        # Controlla se il nodo attuale è collegato ai nodi …
        # (che non sono stati raggiunti da nessun cammino).
        # Se collegato il nodo viene inserito nella lista dei nodi da visitare.
    for p in tovisit:
        if GetGraphLink(graph, p, actual) != 0:
        possible.append(p)
        k = first_index(tovisit, p)
        actual = tovisit.pop(int(k - 1))
    if not len(tovisit):
        return True
    return False
```



# Laboratorio



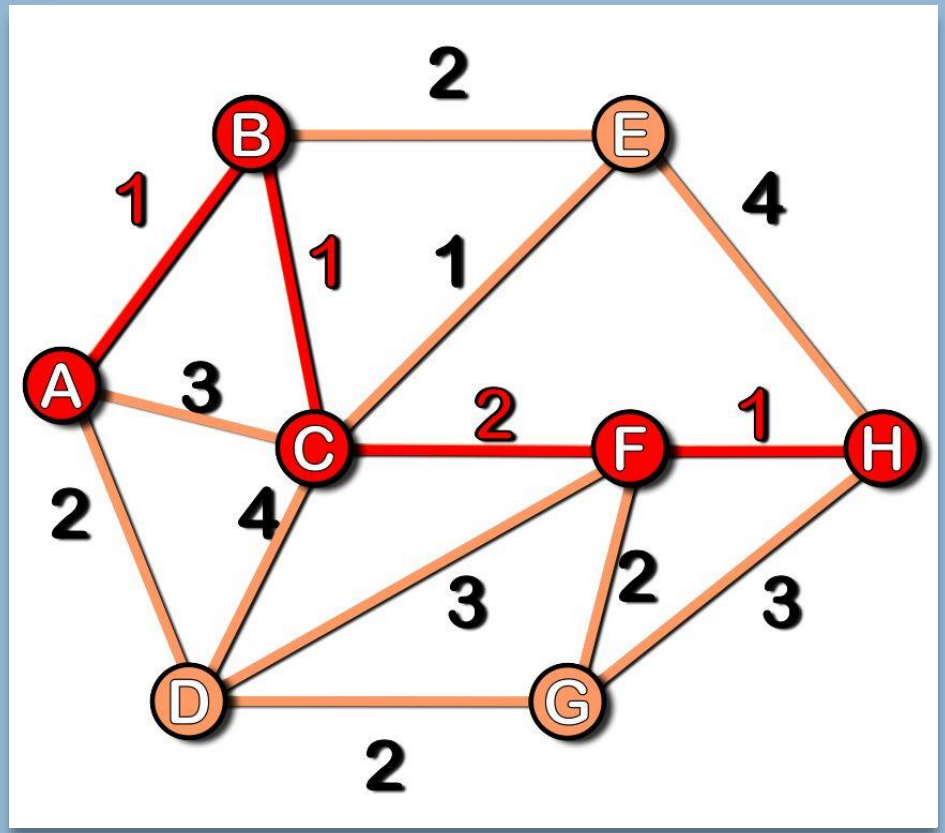

- La procedura è descritta in linguaggio naturale
- Si cerca di applicarla a problemi diversi via via più complessi (xMaxima permette di generare grafi casuali da usare nelle esercitazioni e di controllare la correttezza delle soluzioni)
- Si chiede di formalizzare la procedura in modo quanto più possibile rigoroso (ma sempre senza usare un linguaggio di programmazione)
- Si costruisce un *blocco* con Blockly per implementarla




L'obiettivo del laboratorio è quello di formalizzare la procedura in modo *abbastanza* formale e se possibile implementarla con Blockly

# *Shortest Path*

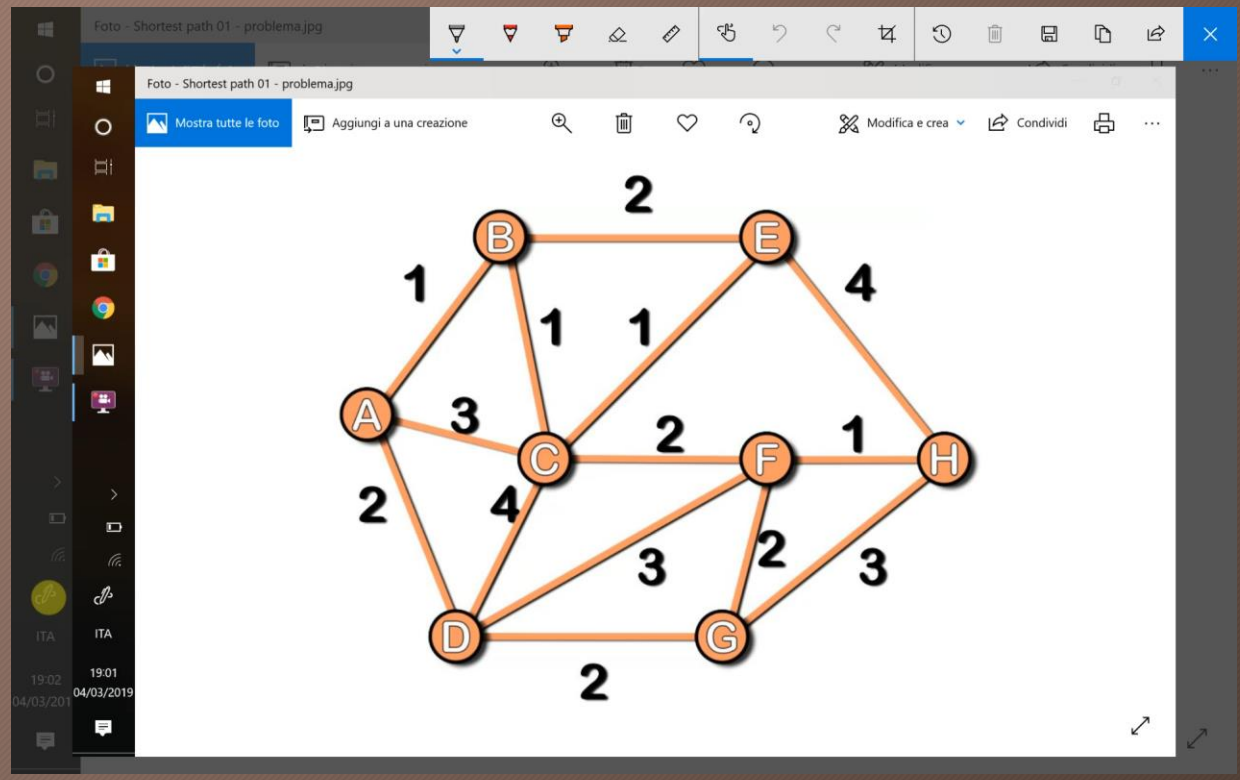

- Determinare il cammino più breve (più economico) in un grafo *pesato*
- Un grafo pesato è un grafo in cui a ogni arco è assegnato un *peso* (un valore)
- Il peso misura il costo di percorrenza di quell'arco
- Un cammino minimo tra due vertici v e w è un cammino da v a w (una successione di nodi/archi) che abbia peso totale minimo
- Ovviamente supporremo il grafo connesso
- Supporremo anche che i pesi degli archi siano tutti positivi



# *Shortest Path*: esempio



- Applichiamo la procedura per cercare il cammino più breve tra il nodo A e il nodo G
- In realtà, eseguendo l'algoritmo fino a esaurire tutti i nodi, si ricavano tutti i cammini minimi da A a qualunque altro nodo
- Questa proprietà della procedura ci sarà utile in seguito



# *Shortest Path*: la procedura, inizializzazione



| Input: il grafo pesato, il nodo di partenza *p*, il nodo di arrivo *a* |
|---|
| poni tutti gli elementi di *lunghezza* pari a ∞ |
| poni tutti gli elementi di *precedente* pari a *null* |
| poni tutti i nodi del grafo in *nodi* |
| aggiungi *p* a *visitati* |
| rimuovi *p* da *nodi* |
| *lunghezza[p]* <- 0 |
| per ogni nodo *n* collegato al nodo *p* |
| *lunghezza[n]* <- peso dell'arco da *p* a *n* |
| *n* <- *precedente[n]* |

# *Shortest Path*: la procedura, ciclo principale



```
Inizializzazione …
finché a non è in visitati
    v <- il nodo di nodi per cui lunghezza[v] è minima
    aggiungi v alla lista visitati
    rimuovi v dalla lista nodi
    per ogni elemento i in nodi
        lunghezza[i] <- la lunghezza del cammino minimo ricalcolata
        precedente[i] <- il nuovo precedente
```

Queste due istruzioni sono abbastanza ambigue e vanno specificate meglio!

# *Shortest Path*: la procedura, il cammino



| Ciclo principale … |
| --- |
| ***n <- precedente[a]*** |
| **finché *n* non è *p*** |
| **    aggiungi n come primo elemento di cammino** |
| ***    n <- precedente[n]*** |
| **aggiungi *p* come primo elemento di cammino** |

# Laboratorio

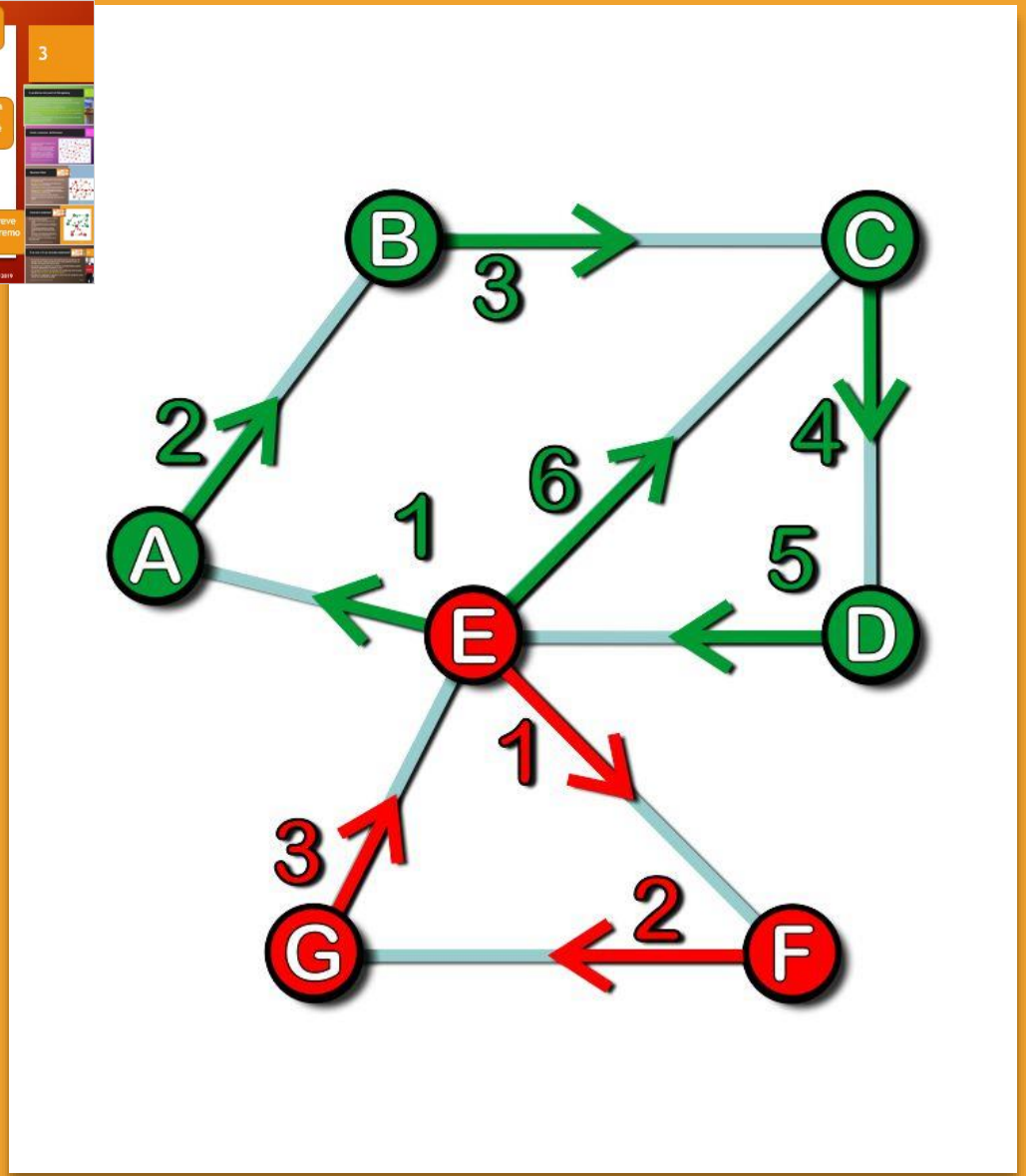


- La procedura è compresa e applicata anche solo usando *carta e penna*
- Si cerca di applicarla a grafi via via più complessi (xMaxima permette di generare grafi casuali da usare nelle esercitazioni e di controllare la correttezza delle soluzioni)
- È poi formalizzata in modo quanto più possibile rigoroso
- Si può *testare* usando Blockly

L'obiettivo è comprendere la procedura e saperla applicare. Se possibile anche quello di formalizzarla in linguaggio naturale.

# Cammini euleriani

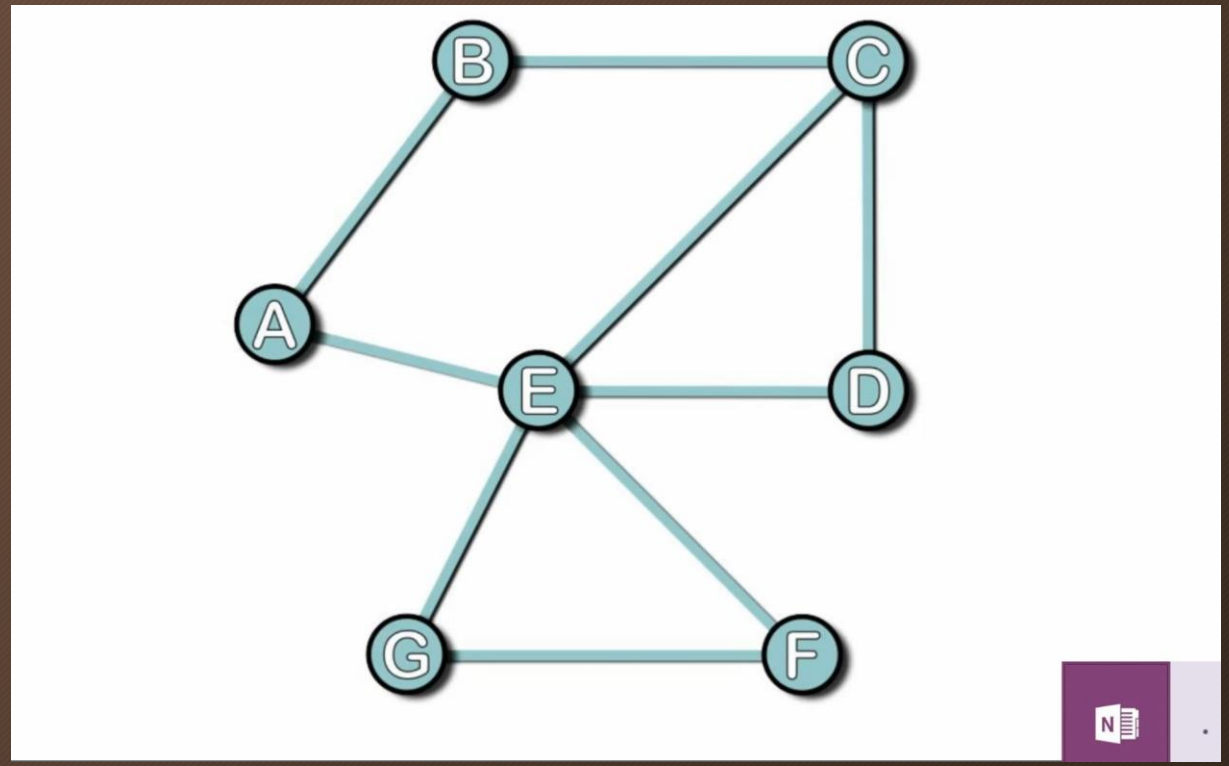

- Costruire un cammino (circuito euleriano) in un grafo in cui è possibile è piuttosto semplice:
  - Si sceglie un nodo di grado dispari come nodo iniziale
  - Si sceglie casualmente un arco e ci si *sposta* sul nodo successivo fino a quando non si può andare più avanti
  - Se sono rimasti fuori degli archi, si ripete il procedimento partendo da uno dei nodi (già attraversati, diciamo *n*) che ha ancora un arco libero
  - Si uniscono i due cammini percorrendo il primo fino al nodo *n*, poi il secondo fino a tornare a *n* e poi si prosegue per il primo cammino
- Se il grafo è euleriano si può partire da un nodo qualunque!



# Cammini euleriani: esempio



- Applichiamo la procedura per determinare un cammino euleriano
- Partiamo da un nodo di grado dispari e andiamo avanti scegliendo casualmente
- Se alcuni archi *restano fuori* si ripete la procedura e poi i due cammini si uniscono!
- Se il grafo è euleriano possiamo iniziare da un nodo qualsiasi



# E se non c’è un circuito euleriano?




- Un circuito euleriano risolve automaticamente il problema del postino cinese, infatti partendo da un nodo qualsiasi esiste un circuito che riporta allo stesso nodo
- Se il grafo non è euleriano lo *forziamo*, lo trasformiano in grafo euleriano aggiungendo cammini *a vuoto*
- Un cammino a vuoto è un cammino che collega due nodi di grado dispari ripercorrendo archi già attraversati
- Il problema è aggiungere cammini a vuoto che nel complesso siano minimi tra tutti quelli possibili

# Concretamente

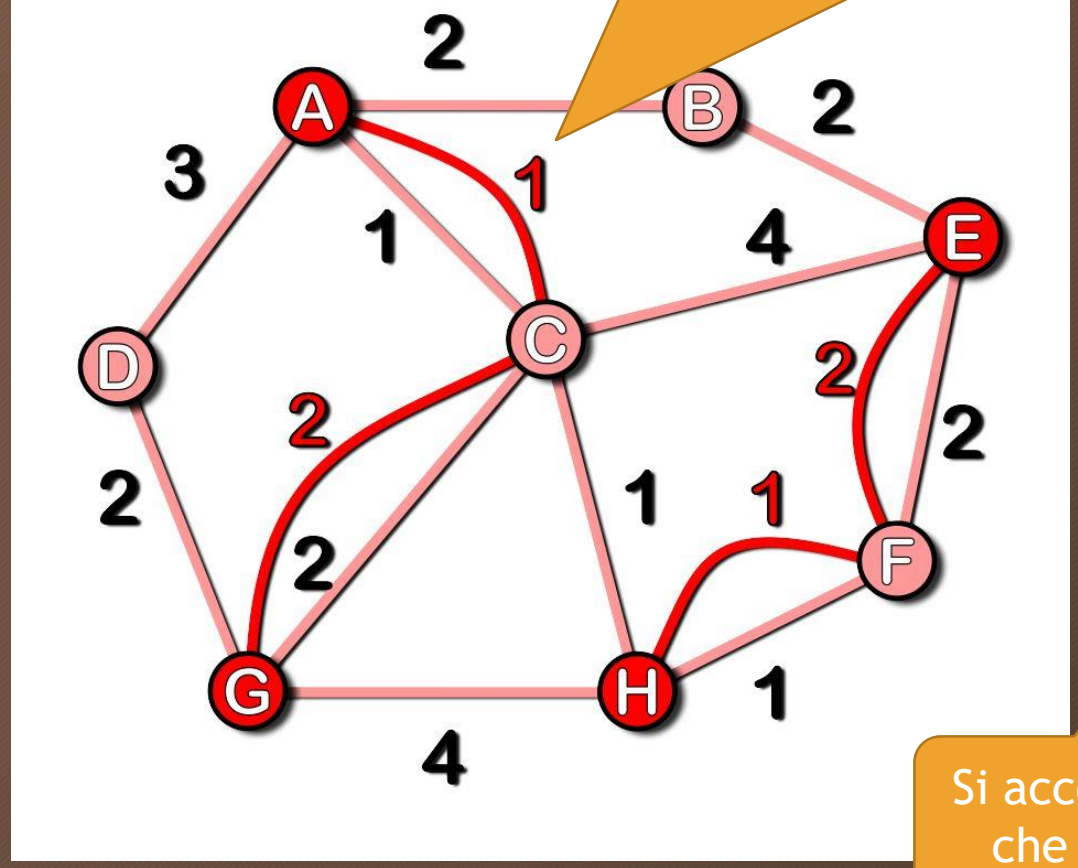


- Si crea un nuovo grafo
  - ha per nodi i soli nodi di grado dispari
  - per archi i cammini minimi tra ogni coppia di nodi (di grado dispari)
- Si cerca di collegare coppie di nodi in questo nuovo grafo in modo da avere un cammino totale minimo (problema di accoppiamento o di trasporto ottimo)
- Nel grafo originale si aggiungono cammini a vuoto corrispondenti ai cammini minimi trovati
- Se colleghiamo a due a due i nodi di grado dispari, questi diventano di grado pari e ora il grafo è euleriano

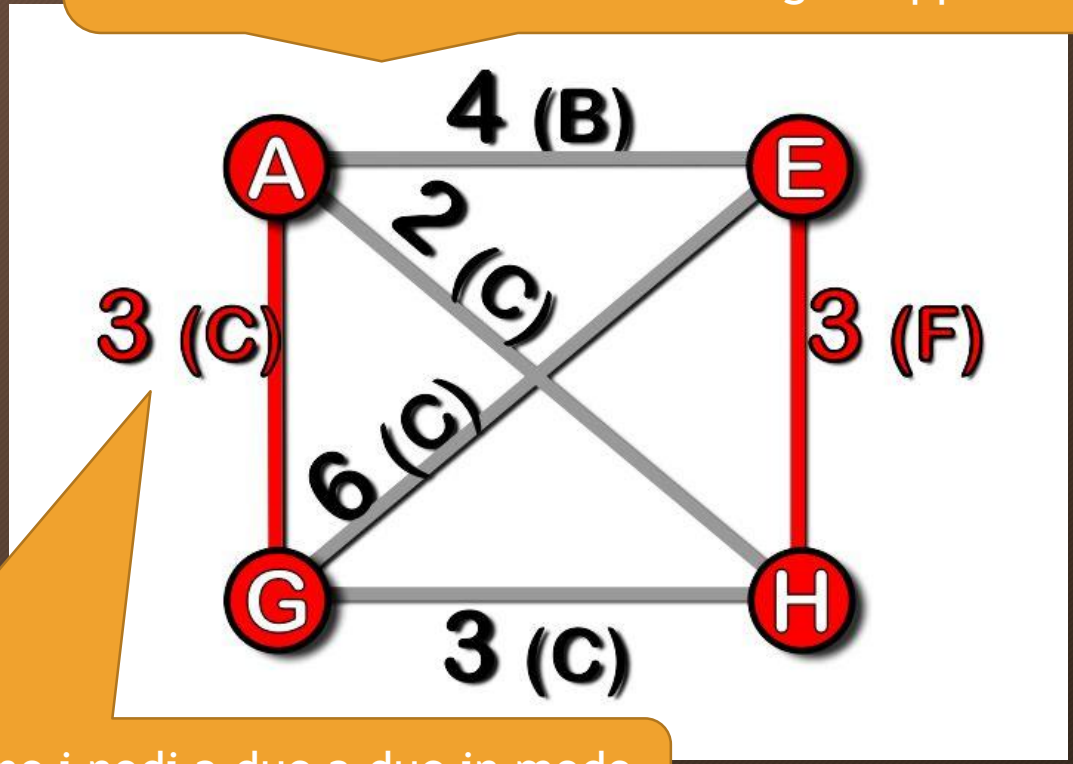
# Concretamente: un esempio



I cammini minimi scelti sono trasformati (raddoppiati) aggiungendo i cammini a vuoto




Si crea un grafo contenente i soli nodi di grado dispari e come peso degli archi la lunghezza del cammino minimo tra ogni coppia




Si accoppiano i nodi a due a due in modo che il cammino totale sia il più breve

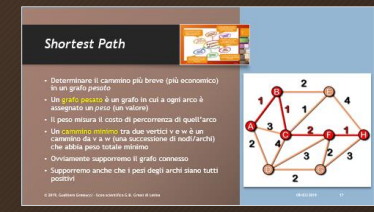
# Laboratorio

L'obiettivo è quello di riuscire a risolvere il problema del postino cinese anche sono con carta e penna. È importante far notare come tutto quanto appreso durante i laboratori può essere usato per risolvere il problema finale!



- La procedura è compresa e applicata anche solo usando *carta e penna*
- Si cerca di applicarla a grafi via via più complessi (xMaxima permette di generare grafi casuali da usare nelle esercitazioni e di controllare la correttezza delle soluzioni)
- È poi formalizzata in modo quanto più possibile rigoroso
- Si può *testare* usando Blockly

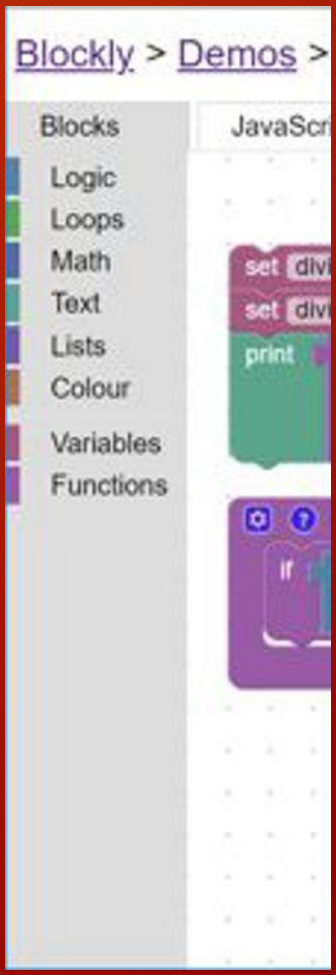





# Research in Action



- *Research in Action* è uno dei progetti di alternanza scuola-lavoro del liceo G.B. Grassi di Latina
- Sul sito del progetto - researchinaction.it: due laboratori (e i relativi materiali di supporto) dedicati al percorso che abbiamo appena illustrato
- Un *Toolbox* che, tra le altre cose, introduce alla topologia e suggerisce come formalizzare in linguaggio naturale procedure e algoritmi
- Sullo stesso sito, alla pagina researchinaction.it/myblockly/graph.html, si può usare *Blockly* per *giocare* con i grafi (la pagina è in allestimento, si prevede di completarla per l'inizio del prossimo anno scolastico)



# Bibliografia minima




- AA. VV. - *Matematita*, centro interuniversitario di ricerca per la comunicazione e l'apprendimento informale della matematica - http://www.matematita.it/progetti/laboratori.php
- AA. VV. – *Quattro passi in centro* – LSS G.B. Grassi – http://researchinaction.it/.../06-Quattro-passi-in-centro.pdf
- AA. VV. – *La via breve* – LSS G.B. Grassi – …
- Casolo, C. e Fumagalli, F. – *Corso di teoria dei grafi e combinatoria* – Dipartimento di matematica U. Dini, Università degli studi di Firenze - http://web.math.unifi.it/users/casolo/dispense/Teoria_Grafi.pdf
- Gritzmann, P. e Brandenberg, R. – *Alla ricerca della via più breve* – Springler