# Homework 1 - Dinamiche su network

Moubane Abdelouahab - s305716, Racca Riccardo - s315163 e Vay Edoardo - s316737

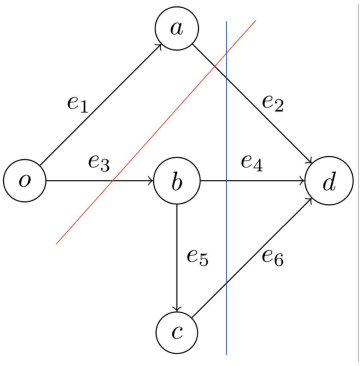
November 13, 2022

## Exercise 1

## a)

*Collaborazione con M. A. Longo S308756, F. Scaramozzino S312856*

La minima capacità che deve essere rimossa dal grafo per interrompere il flusso tra o - d corrisponde alla capacità di minimo taglio. Da questo concludiamo che è sufficiente rimuovere 3 unità di capacità in uno dei due min-cut presenti, ovvero $\{o, a\}$ e $\{o, a, b, c\}$.

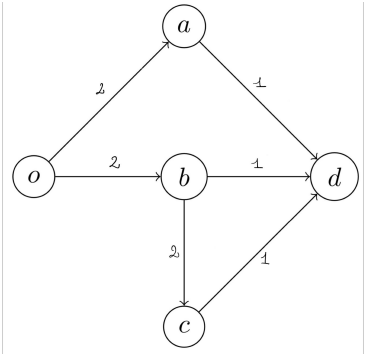



## b)

*Collaborazione con M. A. Longo S308756, F. Scaramozzino S312856*

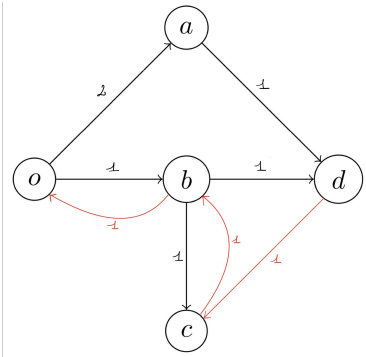
Per ottenere la capacità massima totale rimovibile dal grafo lasciando comunque invariato il massimo throughput, applichiamo il metodo di Ford - Fulkerson, il quale ci permette di distribuire in modo ottimale i flussi sugli archi, e determinare anche le capacità residue inutilizzate. Sommando queste ultime otteniamo il risultato pari a 2.

Per esplicitare quanto anticipato, di seguito esibiamo l'algoritmo di Ford e Fulkerson.

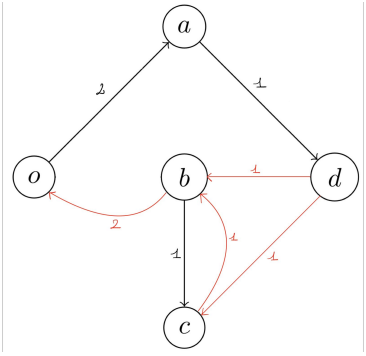



Cominciamo con: $f^{(0)} = (0,0,0,0,0,0)$, $E_0 = (e_1, e_2, e_3, e_4, e_5, e_6)$, $c_0 = (2,1,2,1,2,1)$

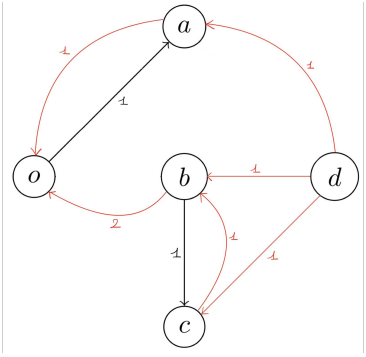
scelgo $\gamma^{(0)} = (o,b,c,d)$, a questo punto $\varepsilon = min(2,2,1) = 1$ e allora $f^{(1)} = (0,0,1,0,1,1)$, $E = (e_1, e_2, e_3, e_4, e_5, e_7, e_8, e_9)$, $c_1 = (2,1,1,1,1,0)$. Ottenendo:




scelgo $\gamma^{(1)} = (o,b,d)$, a questo punto $\varepsilon = min(1,1) = 1$ e allora $f^{(2)} = (0,0,2,1,1,1)$, $E = (e_1, e_5, e_7, e_8, e_9, e_{10}, e_{11})$, $c_2 = (2,1,0,0,1,0)$. Con grafo residuo:




scelgo $\gamma^{(2)} = (o,a,d)$, a questo punto $\varepsilon = min(2,1) = 1$ e allora $f^{(3)} = (1,1,2,1,1,1)$, $E = (e_1, e_5, e_7, e_8, e_9, e_{10}, e_{11}, e_{12}, e_{13})$, $c_3 = (1,0,0,0,1,0)$.



Nel grafo di scarto non ci sono più cammini possibili da $o$ a $d \rightarrow$ **STOP**.

# c)

La base, su cui si fonda il nostro ragionamento, è quella di aumentare le capacità dei mincuts in modo da farli raggiungere le capacità dei che tagli di capacità poco sopra delle capacità dei mincuts, indecheremo da ora in poi il valore di questa capacità con $C_+$ e il valore della capacità dei mincuts con $C_{mincut}$, si noti che durante il ragionamento questi valori verranno aggiornati a ogni step. Un'osservazione importante che teniamo a fare è che l'aumento di capacità di un arco e porta al medesimo aumento delle capacità dei tagli attraversati in modo uscente da e. Il ragionamento effettuato di seguito va a step, dove per ogni step avremmo un insieme di tagli che sono tutti mincut che indicheremo con $C_m$, l'obbiettivo è scrivere questo insieme come unione di minor numero n di sottoinsiemi di $C_m$ tale che siano formati da mincuts attraversati da uno stesso arco. A parità di sottoinsiemi scelgo gli archi che infuenzano il maggior numero di tagli contemporaneamete.

Tagli del grafo

$U_1 = \{0\}$

$U_2 = \{0, a\}$

$U_3 = \{0, b\}$

$U_4 = \{0, c\}$

$U_5 = \{0, a, b\}$

$U_6 = \{0, a, c\}$

$U_7 = \{0, b, c\}$

$U_8 = \{0, a, b, c\}$

Capacità tagli

$C_1 = c_1 + c_3 = 4$

$C_2 = c_2 + c_3 = 3$

$C_3 = c_1 + c_4 + c_5 = 5$

$C_4 = c_1 + c_3 + c_6 = 5$

$C_5 = c_2 + c_4 + c_5 = 4$

$C_6 = c_2 + c_3 + c_6 = 4$

$C_7 = c_1 + c_4 + c_6 = 4$

$C_8 = c_2 + c_4 + c_6 = 3$

| Arco e | Capacità dei tagli dipendenti dall'arco e |
|---|---|
| $e_1$ | $C_1, C_3, C_4, C_5$ |
| $e_2$ | $C_2, C_5, C_6, C_8$ |
| $e_3$ | $C_1, C_2, C_4, C_6$ |
| $e_4$ | $C_3, C_5, C_7, C_8$ |
| $e_5$ | $C_3, C_5$ |
| $e_6$ | $C_4, C_6, C_7, C_8$ |

Iterazioni del ragionamento:

1. $\mathcal{C}_m = \{C_2, C_8\} \rightarrow e_2$

   | Arco e | Capacità dei mincuts dipendenti dall'arco e |
   |---|---|
   | $e_1$ | $\emptyset$ |
   | $e_2$ | $C_2, C_8$ |
   | $e_3$ | $C_2$ |
   | $e_4$ | $C_8$ |
   | $e_5$ | $\emptyset$ |
   | $e_6$ | $C_8$ |

   $n = 1$ quindi se $x \leq n(C_+ - C_{mincut})$ aumento di $\frac{x}{n} = x$ la capacità di $e_2$ che aumenta contemporaneamente i mincuts $C_2$ e $C_8$ e si raggiunge la configurazione finale di massimizzazione dei mincuts.

   Invece se $x > n(C_+ - C_{mincut})$ aumentiamo la capacità dell'arco $e_2$ di $\frac{n(C_+ - C_{mincut})}{n} = 1$, la capacità residua $x_1 = x - n(C_+ - C_{mincut})$ verrà usata nel prossimo step con ragionamento analogo. I nuovi valori delle capacità dei tagli in questo primo step sono i seguenti:

   $C_1 = 4$

   $C_2 = 4$

   $C_3 = 5$

   $C_4 = 5$

   $C_5 = 5$

   $C_6 = 5$

   $C_7 = 4$

   $C_8 = 4$

2. $\mathcal{C}_m = \{C_1, C_2, C_7, C_8\} = \{C_1, C_2\} \cup \{C_7, C_8\} \rightarrow e_3$ e $e_4$

   | Arco e | Capacità dei mincuts dipendenti dall'arco e |
   |---|---|
   | $e_1$ | $C_1$ |
   | $e_2$ | $C_2, C_8$ |
   | $e_3$ | $C_1, C_2$ |
   | $e_4$ | $C_7, C_8$ |
   | $e_5$ | $\emptyset$ |
   | $e_6$ | $C_7, C_8$ |

$n = 2$ se fosse $x_1 \leq n(C_+ - C_{mincut}) = 2$ aumenteremo di $\frac{x_1}{2}$ la capacità di $e_3$ e $e_4$ i quali aumenterebbero contemporaneamente i mincuts $C_1, C_2, C_7$ e $C_8$ e si raggiungebbe la configurazione finale di massimizzazione dei mincuts. Visto che siamo considerando $x \to \infty$ siamo nel $x_1 > 2$ quindi dobbiamo aumentare la capacità dei archi $e_3$ e $e_4$ di $(C_+ - C_{mincut}) = 1$, la capacità residua $x_2 = x_1 - n(C_+ - C_{mincut})$ verrà usata nel successivo step seguendo gli stessi criteri dei precedenti step. Ora i valori delle capacità dei tagli sono i seguenti:

$C_1 = 5$

$C_2 = 5$

$C_3 = 6$

$C_4 = 6$

$C_5 = 6$

$C_6 = 6$

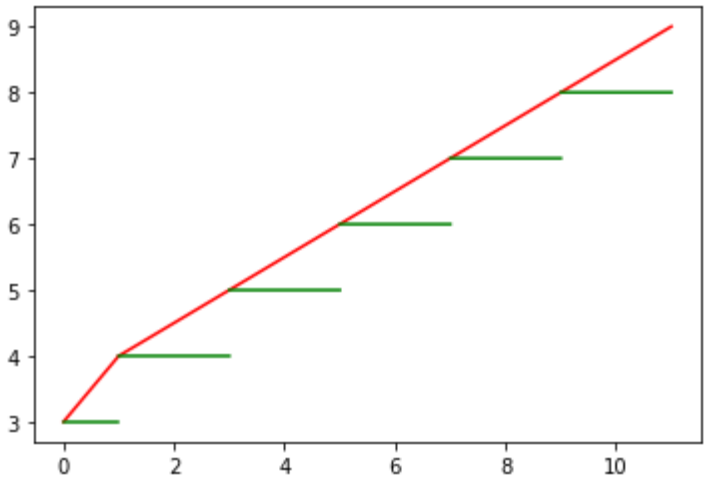
$C_7 = 5$

$C_8 = 5$

3. $\mathcal{C}_m = \{C_1, C_2, C_7, C_8\} = \{C_1, C_2\} \cup \{C_7, C_8\} \to e_3$ e $e_4$

   Si può osservare che i mincuts sono gli stessi del punto precedente, quindi la scelta dei archi sarà la medesima e anche l'incremento delle capacità dei vari tagli sarà lo stesso. Questo ragionamento si può iterare per un qualunque successivo step e si può dedurre che l'incremento sarà sempre $\frac{x_i}{2}$ all' $(i+1)-$esimo step.

In conclusione il grafico di $f(x)$ è il seguente:




Il grafico in rosso è la $f(x)$ nel caso di $x$ continuo, invece in verde si considera $x$ discreto.

# Exercise 2

## a)

Per calcolare il social optimum flow vector dobbiamo risolvere il seguente problema di minimizzazione, che indica il tempo medio di permanenza sulla rete dei *drivers*.

$$\min_{f \geq 0, Bf = \nu} \sum_{e \in \varepsilon} f_e \cdot \tau_e(f_e)$$

$$\phi_1 = f_1 = f_2 = f_3, \phi_2 = f_4 = f_5 = f_6$$

$$\begin{cases} \phi_1 + \phi_2 = 2 \\ \phi_1(\tau_1 + \tau_2 + \tau_3) + \phi_2(\tau_4 + T_5 + \tau_6) = h(\phi_1, \phi_2) \end{cases}$$

$$\Rightarrow h(\phi_1) = 10\phi_1^2 - 20\phi_1 + 24$$

$$\begin{cases} h'(\phi_1) = 0 \\ h'(\phi_1) = 20\phi_1 - 20 \end{cases} \quad , h'' \geq 0 \Rightarrow \phi_1 = 1$$

Da cui otteniamo:

$$\Rightarrow z^* = [z_1, z_2]^T = [1, 1]^T$$

$$\Rightarrow f^* = [f_1, \dots, f_6]^T = [1, 1, 1, 1, 1, 1]^T$$

## b)

È noto dalla teoria che per calcolare l'equilibrio di Wardrop dobbiamo imporre delle condizioni sui tempi di percorrenza dei vari paths, in modo tale che risultino vantaggiosi secondo il punto di vista, egoistico, del singolo utente. Perciò, noti:

$$\begin{cases} T_1 = \tau_1 + \tau_2 + \tau_3 = 3z_1 + z_1 + 1 + z_1 + 1 \\ T_2 = \tau_4 + \tau_5 + \tau_6 = 2z_2 + 2 + 3z_2 \end{cases}$$

Imponiamo:

$$z_1 > 0 \Rightarrow T_1 \leq T_2 \Rightarrow 5z_1 + 2 \leq 5z_2 + 2 \Rightarrow z_1 \leq z_2$$

$$z_2 > 0 \Rightarrow T_2 \leq T_1 \Rightarrow 5z_2 + 2 \leq 5z_1 + 2 \Rightarrow z_2 \leq z_1$$

Concordi alle precedenti ipotesi imponiamo il seguente sistema per valutare la distribuzione che porta all'equilibrio di Wardrop.

$$\begin{cases} z_1 + z_2 = 2 \\ z_1 \leq z_2 \\ z_2 \leq z_1 \end{cases} \Rightarrow z_1 = z_2 = 1$$

Che corrisponde a:

$$\Rightarrow z^{(0)} = [z_1, z_2]^T = [1, 1]^T$$

$$\Rightarrow f^{(0)} = [f_1, \dots, f_6]^T = [1, 1, 1, 1, 1, 1]^T$$

Calcolando, infine, il Price Of Anarchy di questa rete con la precedente distribuzione di flussi:

$$PoA = \frac{\sum_{e \in E} f_e^{(0)} \tau_e\left(f_e^{(0)}\right)}{\sum_{e \in E} f_e^{*} \tau_e\left(f_e^{*}\right)} = \frac{14}{14} = 1$$

Otteniamo che l'equilibrio di Wardrop coincide con l'ottimo di sistema.

# c)

Per la determinazione dell'arco $e_7$ da aggiungere al grafo di partenza, abbiamo seguito i seguenti criteri di scelta:

- Simmetria: Abbiamo riscontrato che fosse fondamentale influenzare equamente i due paths del grafo originale nella creazione del terzo path.
- Costi fissi: Nei grafi in cui si verifica il paradosso di Braess è ricorrente il fatto che il percorso attraversante l'arco aggiunto faccia saltare agli utenti alcuni archi con funzione di costo dipendente da una costante sommata.

Seguendo questi criteri abbiamo deciso di posizionare l'arco $e_7$ con coda sul nodo $n_1$ e testa sul nodo $n_4$. Per verificare la correttezza della nostra scelta calcoleremo ottimo sociale ed equilibrio di Wardrop sul nuovo grafo, valutando, infine, se il Price Of Anarchy finale risulta maggiore di 1 (ovvero se si verifica effetivamente il Paradosso di Braess).

Impostiamo il seguente sistema per calcolare l'ottimo sociale:

$$\begin{cases} z_1 + z_2 + z_3 = 2 \\ 3(z_1 + z_3)^2 + 2(z_1 + 1)z_1 + 2z_2^2 + 2z_2 + 3(z_2 + z_3)^2 = h(z_1, z_2, z_3) \end{cases}$$

$$\Rightarrow h\left(z_{1,} z_2\right) = 6z_1^2 + 6z_2 + 2z_1 z_2 - 14z_1 - 14z_2 + 28$$

$$\begin{cases} \dfrac{\partial h}{\partial z_1} = 12z_1 + 2z_2 - 14 = 0 \\ \dfrac{\partial h}{\partial z_2} = 12z_2 + 2z_1 - 14 = 0 \end{cases} \Rightarrow z_1 = z_2$$

$$\Rightarrow 12z_2 + 2z_1 - 14 = 0 \Rightarrow z_1 = 1 \Rightarrow z_2 = 1 \wedge z_3 = 0$$

Da cui otteniamo i seguenti vettori di flusso ottimale su archi e su paths.

$$\Rightarrow z^* = \left[z_1^*, z_2^*, z_3^*\right]^T = [1, 1, 0]^T$$

$$\Rightarrow f^* = \left[f_1^*, \dots, f_7^*\right]^T = [1, 1, 1, 1, 1, 1, 0]^T$$

Ora procediamo, invece, con il calcolo dell'equilibrio di Wardrop:

$$\begin{aligned} T_1 &= 5z_1 + 3z_3 + 2 \\ T_2 &= 5z_1 + 3z_3 + 2 \\ T_3 &= 3z_1 + 3z_2 + 7z_3 \end{aligned}$$

$$z_1 > 0 \Rightarrow \begin{cases} T_1 \leq T_2 \Rightarrow z_1 \leq z_2 \\ T_1 \leq T_3 \Rightarrow 2z_1 - 3z_2 - 4z_3 + 2 \leq 0 \end{cases}$$

$$z_2 > 0 \Rightarrow \begin{cases} T_2 \leq T_1 \Rightarrow z_2 \leq z_1 \\ T_2 \leq T_3 \Rightarrow -3z_1 + 2z_2 - 4z_3 + 2 \leq 0 \end{cases}$$

$$z_3 > 0 \Rightarrow \begin{cases} T_3 \leq T_1 \Rightarrow -2z_2 + 3z_2 + 4z_3 - 2 \leq 0 \\ T_3 \leq T_2 \Rightarrow 3z_1 - 2z_2 + 4z_3 - 2 \leq 0 \end{cases}$$

$$\begin{cases} z_1 > 0 \\ z_2 > 0 \\ z_3 > 0 \\ z_1 + z_2 + z_3 = 2 \end{cases} \Rightarrow \begin{cases} z_1 = z_2 \\ 3z_1 - 2z_2 + 4z_3 - 2 = 0 \\ 2z_1 - 3z_2 - 4z_3 + 2 = 0 \\ z_1 + z_2 + z_3 = 2 \end{cases}$$

Che ci restituisce i seguenti vettori dei flussi:

$$\Rightarrow z^{(0)} = [z_1, z_2, z_3]^T = \left[\frac{6}{7}, \frac{6}{7}, \frac{2}{7}\right]^T$$

$$\Rightarrow f^{(0)} = [f_1, \dots, f_7]^T = \left[\frac{8}{7}, \frac{6}{7}, \frac{6}{7}, \frac{6}{7}, \frac{6}{7}, \frac{8}{7}, \frac{2}{7}\right]^T$$

E dal seguente calcolo del Price Of Anarchy:

$$PoA = \frac{14.29}{14} > 1$$

Abbiamo conferma che nonostante sia stato introdotto un arco percorribile nel sistema, esso non migliori in termini di: tempo medio di permanenza dell'utente all'interno della rete.

# d)

Basandosi sulla teoria sappiamo che il vettore di tolls ottimale, $\omega_e$, è definito come un vettore le cui componenti, se sommate alle corrispondenti funzioni di costo relative agli archi, modificano le scelte degli utenti, portandoli nonostante il loro comportamento egoistico a raggiungere l'ottimo sociale. E' inoltre noto che, dati il vettore di flussi ottimale sugli archi e il vettore delle derivate delle funzioni di costo, tramite la seguente formula si ottiene un vettore di tolls ottimale, cioè:

$$\omega_e^* = f_e^* \cdot \tau_e'\left(f_e^*\right)$$

Avendo vettore ottimale dei flussi:

$$f^* = \left[f_1^*, \dots, f_7^*\right]^T = [1, 1, 1, 1, 1, 1, 0]^T$$

e vettore delle funzioni di costo:

$$\tau(x) = [3x, x + 1, x + 2, 2x, 2, 3x, x]^T$$

Si ottiene:

$$\Rightarrow \omega^* = [3, 1, 1, 2, 0, 3, 0]^T$$

Concordi alla definizione scritta poco sopra, per ottenere un optimal toll vector a supporto completo è sufficiente, una volta trovato $\omega_e^*$, mantenere la scelta degli utenti invariata. Seguendo tale ragionamento notiamo che la chiave sia quella di modificare in modo equo il costo di ciascun path per lasciare immutata la loro "convenienza" agli occhi dei *drivers*. Concludendo che sia sufficiente sommare ad ogni funzione di costo la medesima quantità costante $k > 0$.

Per verificare la validità della nostra scelta abbiamo calcolato l'equilibrio di Wardop con la nuova configurazione di tolls a supporto completo. Giungendo, correttamente, al vettore di flussi calcolato tramite lo studio dell'ottimo sociale.

$$\omega^+ = \omega^* + 1^T \cdot k, k > 0$$

Aggiungendo $k > 0$ si hanno i seguenti tempi di percorrenza su paths:

$$T_1 = 5z_1 + 3z_3 + 2 + (5 + 3k)$$
$$T_2 = 5z_1 + 3z_3 + 2 + (5 + 3k)$$
$$T_3 = 3z_1 + 3z_2 + 7z_3 + (6 + 3k)$$

Da cui ricaviamo:

$$\Rightarrow z^{(0)} = [z_1, z_2, z_3]^T = [1, 1, 0]^T$$

$$\Rightarrow f^{(0)} = [f_1, \dots, f_7]^T = [1, 1, 1, 1, 1, 1, 0]^T$$

Per quanto riguarda, invece, la scelta del vettore di tolls ottimale a supporto unicamente sull'arco $e_7$ abbiamo intuito che è sufficiente imporre un toll grande abbastanza da far desistere i *drivers* nella scelta del percorso passante per l'arco $e_7$. Perciò abbiamo imposto il seguente sistema:

$$T_1 = 5z_1 + 3z_3 + 2$$
$$T_2 = 5z_1 + 3z_3 + 2$$
$$T_3 = 3(z_1 + z_3) + (z_3 + c) + 3(z_2 + z_3)$$

$$\begin{cases} T_1 \leq T_2 \\ T_2 \leq T_1 \end{cases} \Rightarrow z_1 = z_2$$

Ottendo che è sufficiente inserire un toll avente valore:

$$\begin{cases} T_1 \leq T_3 \Rightarrow z_1 \leq \dfrac{z_3 + 4 + c}{5} \\ z_1 = z_2 \\ z_1 + z_2 + z_3 = 2 \\ z_1 = 1 \end{cases} \Rightarrow c \geq 1$$

# e)

Calcoliamo l'ottimo sociale nel nuovo grafo imponendo alla funzione di costo anche il vincolo del traffico totale:

$$\begin{cases} z_1 + z_2 + z_3 = \chi \\ 3(z_1 + z_3)^2 + 2(z_1 + 1)z_1 + 2z_2^2 + 2z_2 + 3(z_2 + z_3)^2 + (\alpha z_3 + 2)z_3 = h(z_1, z_2, z_3) \end{cases}$$

$$h(z_1, z_2) = (5 + \alpha)\left(z_1^2 + z_2^2\right) - (2\alpha + 6)(z_1\chi + z_2\chi) + 2\alpha z_1 z_2 + 2\chi + (6 + \alpha)\chi^2$$

$$\begin{cases} \dfrac{\partial h}{\partial z_1} = 2(5 + \alpha)z_1 - (2\alpha + 6)\chi + 2\alpha z_2 = 0 \\ \dfrac{\partial h}{\partial z_2} = 2(5 + \alpha)z_2 - (2\alpha + 6)\chi + 2\alpha z_1 = 0 \end{cases}$$

$$\Rightarrow z_1 = z_2 = \frac{6 + 2\alpha}{10 + 4\alpha}x \Rightarrow z_3 < 0$$

$$z_1 = \chi - z_2$$

$$\Rightarrow \begin{cases} z_1 = \chi - z_2 \\ h(z_1, z_2) \end{cases}$$

$$\Rightarrow h(z_2) = 10z_2^2 - 10\chi z_2 + 2\chi$$

$$h'(z_2) = 20z_2 - 10\chi = 0 \Rightarrow z_2 = \frac{\chi}{2} \Rightarrow z_1 = \frac{\chi}{2}$$

$$\Rightarrow z^{(0)} = [z_1, z_2, z_3]^T = \left[\frac{\chi}{2}, \frac{\chi}{2}, 0\right]^T$$

$$\Rightarrow f^{(0)} = [f_1, \dots, f_7]^T = \left[\frac{\chi}{2}, \frac{\chi}{2}, \frac{\chi}{2}, \frac{\chi}{2}, \frac{\chi}{2}, \frac{\chi}{2}, 0\right]^T$$

Non è necessario calcolare il flusso al Wardrop infatti, basta analizzare il grafo con i flussi distribuiti al Social Optimum e verificare che ad ogni nodo la scelta di tutti gli utenti sia il path più veloce. Rapidamente si può notare come al nodo $n_1$ gli utenti abbiano la scelta di spostarsi sul path $z_3$ o proseguire, come prima, sul path $z_1$: il tempo di percorrenza da $n_1$ alla destinazione al momento del Social Optimum è pari a $\chi + 2$ lungo il path $z_1$, mentre se scegliessero il path $z_3$ allora impiegherebbero $3\frac{\chi}{2} + 2$ che è maggiore del precedente. Questo implica che nessun utente sceglierà mai un percorso diverso da quello del social optimum e quindi l'equilibrio di Wardrop è uguale al Social Optimum. Per questo il PoA è pari a 1 per qualsiasi valore di $\alpha$ e $\chi$; quindi il vettore di toll ottimali è banalmente il vettore nullo.

# Exercise 3

## a)

*Collaborazione con M. A. Longo S308756, F. Scaramozzino S312856*

```
import numpy as np
matrix = np.asarray([
    [0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0],
    [1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0],
    [1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0],
    [1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0],
    [1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
    [1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1],
    [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]
])
N = 14
matrix1 = matrix.reshape((N, N))
```

```
print('\n Degree Centrality')
z = np.sum(matrix, axis=0)
for i, j in enumerate(z):
    print(f'Nodo: {i+1}: \t {j}')

eigenvalue, eigenvectors = np.linalg.eigh(matrix.T)
n = np.argmax(eigenvalue)
lambda_ = (1 / (np.sum(eigenvectors[:, n]))) * eigenvectors[:, n]
print('\n Eigenvectors centrality')
for i, j in enumerate(lambda_):
    print(f'Nodo: {i+1}: \t {round(j, 4)}')

pi = z/np.sum(z)
print('\n Invariant Distribution centrality')
for i, j in enumerate(pi):
    print(f'Nodo: {i+1}: \t {round(j, 4)}')

d = np.sum(matrix, axis=0)
D = np.diag(d)
# calculate the P matrix
P = np.matmul(np.linalg.inv(D),matrix)
```

Listing 1: Katz centrality

```
 Degree Centrality
Nodo: 1:         5
Nodo: 2:         5
Nodo: 3:         5
Nodo: 4:         5
Nodo: 5:         5
Nodo: 6:         6
Nodo: 7:         2
Nodo: 8:         2
Nodo: 9:         6
Nodo: 10:        1
Nodo: 11:        1
Nodo: 12:        1
Nodo: 13:        1
Nodo: 14:        1
 Eigenvectors centrality
Nodo: 1:         0.1583
Nodo: 2:         0.1583
Nodo: 3:         0.1583
Nodo: 4:         0.1583
Nodo: 5:         0.1583
Nodo: 6:         0.1639
Nodo: 7:         0.034
Nodo: 8:         0.0071
Nodo: 9:         0.0018
Nodo: 10:        0.0003
Nodo: 11:        0.0003
Nodo: 12:        0.0003
```

```
Nodo: 13:        0.0003
Nodo: 14:        0.0003
 Invariant Distribution centrality
Nodo: 1:         0.1087
Nodo: 2:         0.1087
Nodo: 3:         0.1087
Nodo: 4:         0.1087
Nodo: 5:         0.1087
Nodo: 6:         0.1304
Nodo: 7:         0.0435
Nodo: 8:         0.0435
Nodo: 9:         0.1304
Nodo: 10:        0.0217
Nodo: 11:        0.0217
Nodo: 12:        0.0217
Nodo: 13:        0.0217
Nodo: 14:        0.0217
```

La degree centrality sappiamo essere una misura di centralità esclusivamente legata alla forma del grafo. Essa infatti attribuisce come valore di centralità al nodo i-esimo, esattamente la somma dei pesi degli archi entranti in esso: $\omega_i^- = \sum_j W_{ji}$. In formule vale che: $z_i = \omega_i^-$ dove $z$ rappresenta proprio il vettore di centralità dei nodi. I risultati, concordi a quanto anticipato, evidenziano che i nodi con maggiore degree centrality sono $n_6$ ed $n_9$.

Per aggiungere, intuitvamente, il concetto che non tutti i collegamenti sono ugualmente importanti ai fini del calcolo della centralità introduciamo la eigenvector centrality, con l'obiettivo di considerare più influenti, nel calcolo della centralità di un nodo generico $n_e$, i links provenienti da nodi molto centrali. Tradotto in formule il calcolo diventa: $z = (\lambda_W)^{-1} W' z$, dove $\lambda_W$ è l'autovalore dominante di $W$, e $z$ è l'autovettore non-negativo, normalizzato e unico (perché $G$ è fortemente connesso) di $W'$ rispetto a $\lambda_W$. I risultati rispecchiano la descrizione appena accennata, in cui sottolineiamo che il nodo avente più centralità è proprio $n_6$, essendo per l'appunto collegato con molti nodi a loro volta aventi alta centralità.

A questo punto è bene sottolineare, però, come la eigenvector centrality non sia del tutto esaustiva nel modellizzare la centralità di una rete. Essa tralascia il fatto, intuitivo, che se si è solamente un link tra moltissimi altri, allora il nostro link è ragionevole pensare che non sia così importante. Seguendo questa idea, un nodo molto centrale deve distribuire il proprio ammontare di centralità in funzione del numero di nodi a cui la distribuisce. Così facendo si evita che nodi con alta centralità possano influenzare troppo una rete. Questa misura di centralità nuova prende il nome di invariant distribution centrality. In formule: $z = P' z$ dove (data W matrice di adiacenza e $D = diag(w)$) $P = D^{-1} \cdot W$, $z$ è la distribuzione invariante (unica se G è fortemente connesso). I risultati racchiudono correttamente questa variazione interpretativa, attribuendo maggiore centralità a $n_6$ e $n_9$. Notiamo come $n_9$ abbia la stessa centralità di $n_6$ malgrado presenti collegamenti provenienti da nodi con valori di centralità bassa (al contrario di $n_6$ i cui in-neighbors sono molto centrali). La differenza determinante sta nel fatto che i nodi entranti in $n_9$ comunicano solo con esso, quindi la loro centralità sfocia tutta in $n_9$, essendo l'unico out-neighbor.

## b)

```python
def katz_centrality(matrix, beta, iteration, u):
    '''
    Returns the katz centrality of a matrix.

    :param matrix: adjacency matrix of the graph
        the matrix must be implemented with the pythonGraph interface written for PCS lessons.
    :param beta: The beta parameter for the PageRank algorithm
    :param iteration: The number of iteration parameter for the PageRank algorithm
    :param u: vector of the initial centrality of each node;
    '''
    # initialize the centrality of all nodes to the given u
    centrality = u
    # compute the eigenvalue of the matrix rappresenting the graph
    eig = np.linalg.eigvals(np.asarray(matrix))
    # compute the alpha value using the maximum eigenvalue
    alpha = (1-beta)/np.max(eig)
    # iterate the centratality computation
    for _ in range(iteration):
        # compute the centratality of each iteration computing the new vaule using the previous iteration
        centrality = alpha * np.matmul(matrix, centrality) + beta * u

    return centrality
```

Listing 2: Katz centrality

```
Nodo 1: 0.0729
Nodo 2: 0.0729
Nodo 3: 0.0729
Nodo 4: 0.0729
Nodo 5: 0.0729
Nodo 6: 0.0768
Nodo 7: 0.027
Nodo 8: 0.0198
Nodo 9: 0.0269
Nodo 10: 0.0153
Nodo 11: 0.0153
Nodo 12: 0.0153
Nodo 13: 0.0153
Nodo 14: 0.0153
```

## c)

```python
def page_rank(graph, beta, iteration, u = None) -> dict:
    '''
    Returns the page rank of a graph.

    :param graph: The graph to search the page rank centrality
        the graph must be implemented with the pythonGraph interface written for PCS lessons.
```

```
    :param beta: The beta parameter for the PageRank algorithm
    :param iteration: The number of iteration parameter for the PageRank
    algorithm
    :param u: vector of the initial centrality of each node;
        if None, then the initial vector is a uniform distribution

    :return: The page rank centrality of the graph for each iteration as a
    dictionary (keys are the nodes id).
    '''
    # initialize to default vector of uniforme distribution
    if u is None:
        n_nodes = len(graph.nodes.keys())
        u = [1/n_nodes for i in range(n_nodes)]

    # dictionary of keys for each node
    node_centrality = {}

    # initialize the initial value of each node at the initial value in u
    for i, node in enumerate(graph.nodes):
        node_centrality[node] = [u[i]]

    # iterate the calculation of the centrality
    for j in range(iteration):
        # iterate on each node on the graph
        for i, graph_node in enumerate(graph.nodes):
            # find all the in neighbours of the node
            _ , in_neighbours = graph.nodes[graph_node].get_neighbours()
            # initialize the extrinzic centrality to 0
            sum_centrality = 0
            # iterate on each in neighbour of the node
            for in_node in in_neighbours:
                # calculate the out deegre of the node
                grado_out, _ = in_node.degrees()
                # update the centrality to the one given by this node
                sum_centrality += (node_centrality[in_node.id][j]) / grado_out

            # save the final value in the dictionary
            node_centrality[graph_node].append((1-beta) * sum_centrality + beta
    * u[i])

    #return the dictionary
    return node_centrality
```

Listing 3: Bonacich

```
Nodo 1: 0.0762
Nodo 2: 0.0762
Nodo 3: 0.0762
Nodo 4: 0.0762
Nodo 5: 0.0762
Nodo 6: 0.0959
Nodo 7: 0.0524
Nodo 8: 0.059
Nodo 9: 0.221
Nodo 10: 0.0382
Nodo 11: 0.0382
```

```
Nodo 12: 0.0382
Nodo 13: 0.0382
Nodo 14: 0.0382
```

Alternative method

```python
def page_rank_alg(matrix, beta, iteration, u):
    '''
    Returns the page rank of a graph.

    :param graph: The graph to search the page rank centrality
        the graph must be implemented with the pythonGraph interface written for PCS lessons.
    :param beta: The beta parameter for the PageRank algorithm
    :param iteration: The number of iteration parameter for the PageRank algorithm
    :param u: vector of the initial centrality of each node;
        if None, then the initial vector is a uniform distribution
    :return: The page rank centrality of the graph for each iteration
     as a dictionary (keys are the nodes id).
    '''
    # dimension of the square matrix
    N = len(matrix[0])
    matrix = matrix.reshape((N, N))

    # calculate the degree of the nodes
    d = np.sum(matrix, axis=0)
    D = np.diag(d)
    # calculate the P matrix
    P = np.matmul(np.linalg.inv(D),matrix)
    # initialize the centrality
    centrality = np.zeros((iteration, N))
    # first iteration equal to the initial centrality
    for i in range(N):
        centrality[0, i] = u[i]

    for k in range(1, iteration):
        for i in range(N):
            centrality[k, i] = (1-beta) * np.dot(P[:, i], centrality[k-1]) + beta * u[i]

    return centrality
```

Listing 4: Bonacich

# d)

Per condurre l'analisi completa dei risultati, richiameremo alcuni concetti chiave sulle due centralità in oggetto, basandoci su esse trarremo le nostre conclusioni sui dati ottenuti.

Nota la seguente formula per il calcolo del Katz centrality vector:

$$z = \frac{1-\beta}{\lambda_w} W' z + \beta\mu$$

Osserviamo che al variare di $\beta$ nell'intervallo $(0, 1]$ la centralità di Katz subisce delle variazioni a livello interpretativo. Nello specifico:

- per $\beta \to 0$: la Katz centrality coincide con la eigenvector centrality.
- per $\beta = 1$: la Katz centrality coincide con il vettore delle centralità intrinseche.

Nel nostro caso abbiamo imposto $\beta = 0.15$, un valore piuttosto vicino a 0, che ragionevolmente ci fa ottenere una centralità simile a quella che ci si potrebbe aspettare dalla eigenvector centrality. A supporto di quanto detto, sappiamo che quest'ultima attribuisce alti valori di centralità hai nodi con molti in-neighbors (come la degree centrality) tenendo comunque in considerazione il fatto che i link provenienti da nodi dotati di un elevata centralità debbano essere più influenti.

Grazie a quanto appena detto possiamo convincerci facilmente di come i risultati del punto b) evidenzino valori di centralità più alta nella parte destra del grafo, collegata localmente in modo completo. Inoltre, notiamo come al nodo $n_6$ venga attribuita la centralità maggiore, concorde al fatto che è il nodo del grafo con più collegamenti.

Un potenziale problema della Katz centrality, però, è dato dal fatto che se un nodo con alta centralità ha molti links, allora tutti i nodi da esso raggiunti ottengono un grande incremento di centralità. In molti casi, tuttavia, è poco significativo se un nodo è solamente uno tra innumerevoli ad essere collegato. Per questo la centralità distribuita da un nodo centrale ai propri out-neighbors deve essere adeguatamente diluita. Questo porta a tenere in considerazione, non solo il numero di links e l'importanza di essi ma anche quanto esclusivi questi links siano.

Per tenere in considerazione tutto quello appena detto, si utilizza la classificazione di centralità Page-Rank, con seguente formula:

$$z = (1 - \beta)P'z + \beta\mu$$

Focalizzando l'attenzione sul grafo in analisi notiamo come i risultati ottenuti siano concordi a quanto descritto per la Page-Rank centrality. Entrando più nello specifico osserviamo che il nodo a cui viene attribuito il valore più alto di centralità è $n_9$. Questo non dovrebbe stupire, infatti, esso ha sì un discreto numero di collegamenti ma a fare la differenza è proprio il fatto che i suoi in-neghbors siano collegati solemente con $n_9$ stesso.