

SECONDO PROGETTO ASD 2022/2023



IL RITORNO DEL RE



Quattro anni or sono¹, **Re Albert I** conquistò il Trono di Spade, e ora pericolosi vari intrighi politici lo coinvolgono.

I nobili del regno tramano contro di lui e, in segreto, complottano per spodestarlo.

¹ **Game of (approximated) Thrones**, a.a. 2018/2019

Per eliminare le minacce, ha deciso di organizzare un banchetto a cui tutti i nobili sono invitati. Questo banchetto sarà la chiave per riorganizzare gli equilibri.



Grazie ai suoi fidi informatori, Re Albert conosce la *rete* di tutte le alleanze che i nobili intrattengono tra loro.

Il piano del Re consiste nel separare i nobili in vari gruppi con strette relazioni tra di loro, più facili da controllare.



A tale scopo, alleanze esistenti si interromperanno e nuove alleanze verranno create, in base a come Re Albert sceglierà i posti a sedere.

Un nobile si dovrà sedere a un certo tavolo **se e solo se**, nella scelta del Re, sarà alleato con **tutti e soli** i nobili seduti a quello stesso tavolo.

La logistica del banchetto non sarà un problema:

- ⇒ il mastro falegname può costruire tavoli di qualsiasi misura, forma o numero di posti.



ALCUNE DIFFICOLTÀ

Costruire o distruggere relazioni tuttavia è molto difficile, per questo motivo il Re vuole che la sua azione sia **il più efficiente possibile**.

Il suo obiettivo è quindi quello di intervenire il meno possibile, **minimizzando** il numero di alleanze che vanno create o distrutte.



Quali sono le manipolazioni minime necessarie per preservare l'ordine nel banchetto?

- Quali nuove alleanze occorrerà instaurare?
- Quali vecchie alleanze dovranno essere disfatte?

Un file con $1 + M$ righe.

- La prima riga riporta 2 numeri interi: N (`int`) e M (`int`), rispettivamente il numero di nobili e il numero di alleanze tra nobili.
- Le successive M righe riportano le alleanze tra nobili. Ogni riga riporta 2 interi U (`int`), V (`int`), rappresentanti i due nobili alleati.

NOTA

Le alleanze sono bidirezionali, quindi se U è alleato con V , è implicito che V è alleato con U .

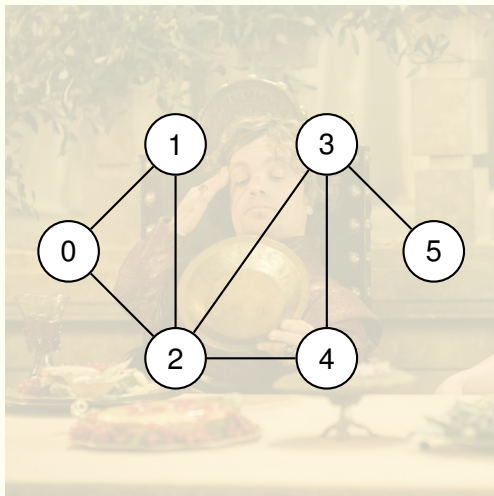
Un file con le vostre proposte di soluzione, ogni proposta ha le seguenti caratteristiche:

- 1 La prima riga del file di output deve contenere due interi A e R : il numero di alleanze create e il numero di alleanze interrotte grazie alla disposizione dei tavoli.
- 2 Le successive $A + R$ righe specificano le alleanze tra nobili create e interrotte.
- 3 Ciascuna riga è costituita da un segno $+$ o $-$, per specificare se l'alleanza è creata o interrotta, seguito da due interi U e V che specificano i due nobili che costituiscono l'alleanza in questione.
- 4 La soluzione deve terminare con una riga contenente la stringa `***` per specificare che si tratta dell'ultima soluzione valida stampata.
 - ⇒ Questo vi consente di stampare più soluzioni finché il tempo a disposizione per l'esecuzione del programma non scade.
 - ⇒ Solo l'ultima soluzione "chiusa" da `***` verrà considerata per la valutazione del punteggio.

ESEMPIO

Una rete di alleanze tra nobili prima del banchetto:
6 nobili, 7 alleanze.

6 7
0 1
2 0
2 3
1 2
3 4
4 2
3 5



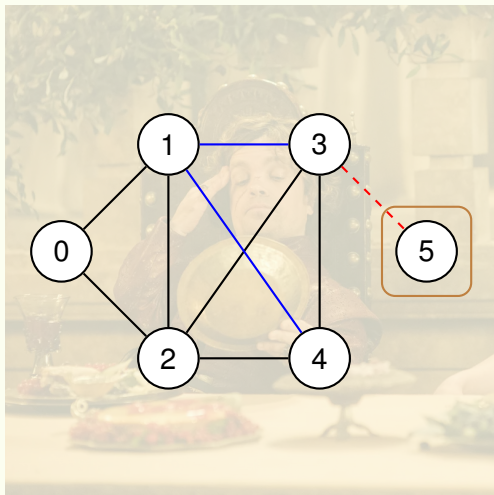
ESEMPIO ERRATO

Soluzione proposta:

```
2 1
+ 1 3
+ 1 4
- 3 5
***
```

Soluzione non valida!

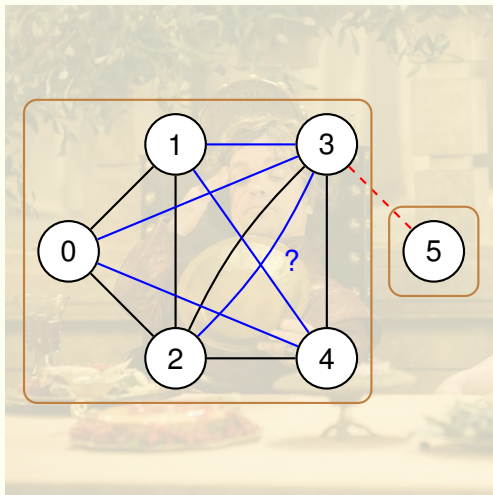
I nobili $\{0, 1, 2, 3, 4\}$ non possono stare allo stesso tavolo: 0 non è alleato con 3 e 4.



ESEMPIO ERRATO

5 1
+ 0 3
+ 0 4
+ 1 3
+ 1 4
+ 2 3
- 3 5

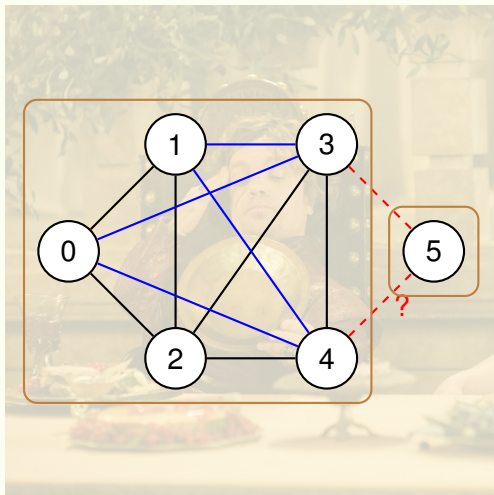
Soluzione non valida! 2 e 3
sono già alleati, non è
possibile creare un'alleanza
preesistente.



ESEMPIO ERRATO

4 2
+ 0 3
+ 0 4
+ 1 3
+ 1 4
- 3 5
- 4 5

Soluzione non valida! 4 e 5
non sono alleati, non è
possibile distruggere
un'alleanza non
preesistente.

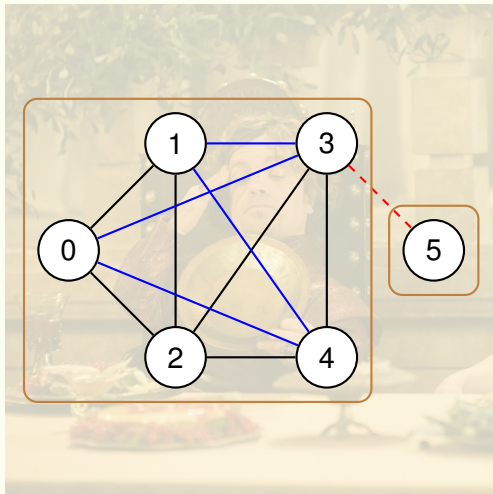


ESEMPIO VALIDO

Soluzione corretta. 4
allenze create, 1 distrutta:

```
4 1
+ 0 3
+ 0 4
+ 1 3
+ 1 4
- 3 5
***
```

Un tavolo con un solo nobile
(5) è valido.

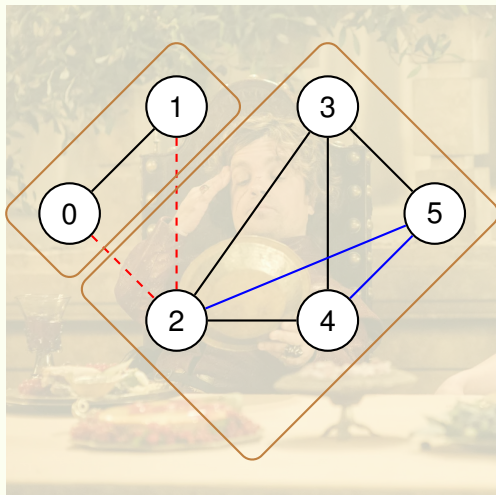


ESEMPIO VALIDO

Soluzione corretta. 2
alleanze create, 2 distrutte:

```
2 2  
+ 2 5  
+ 4 5  
- 0 2  
- 1 2  
***
```

Un tavolo con due nobili
alleati tra loro (0, 1) è valido.

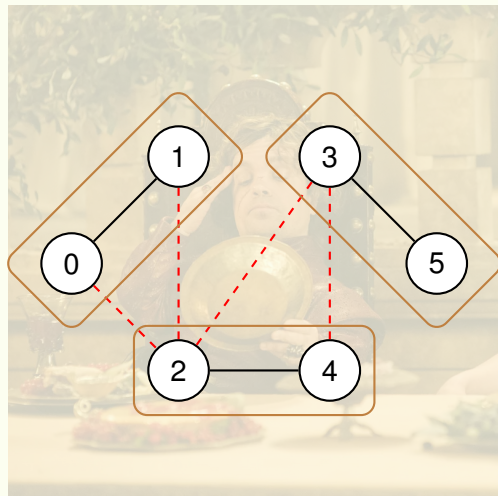


ESEMPIO VALIDO

Soluzione corretta. 0
allenze create, 4 distrutte:

0 4
- 0 2
- 1 2
- 2 3
- 3 4

Sono valide anche soluzioni
in cui le allenza vengono
solo distrutte o solo create.

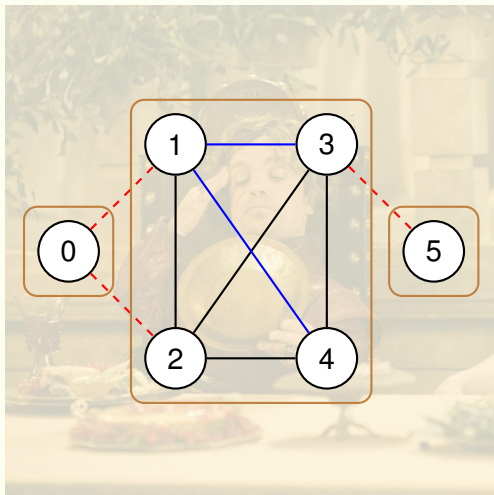


ESEMPIO VALIDO

Soluzione corretta. 2
allenze create, 3 distrutte:

```
2 3
+ 1 3
+ 1 4
- 0 1
- 0 2
- 3 5
***
```

Ci sono molte soluzioni
valide.



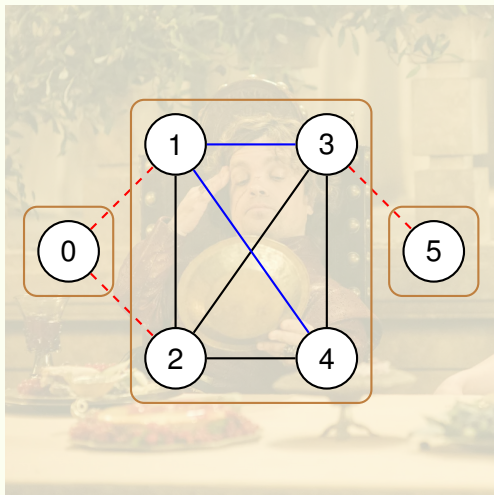
ESEMPIO VALIDO

Soluzione corretta. 2
allenze create, 3 distrutte:

```
2 3  
+ 1 3  
+ 1 4  
- 0 1  
- 0 2  
- 3 5
```

Ci sono molte soluzioni
valide.

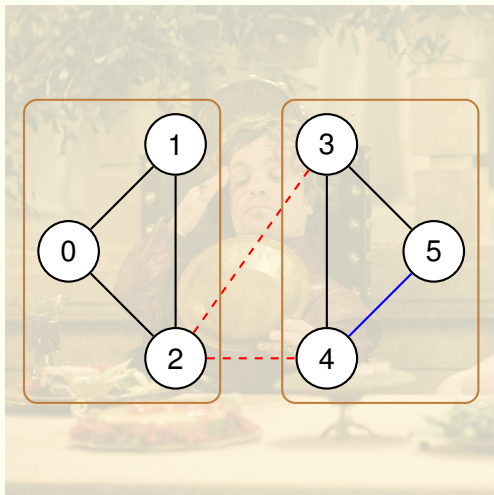
Si può fare di meglio?



ESEMPIO OTTIMO

Soluzione ottima, ovvero
con il minimo numero di
alleanze modificate (3). 1
creata, 2 distrutte:

```
1 2  
+ 4 5  
- 2 3  
- 2 4  
***
```



ESEMPIO OTTIMO

Soluzione ottima. 0 alleanze create, 3 distrutte:

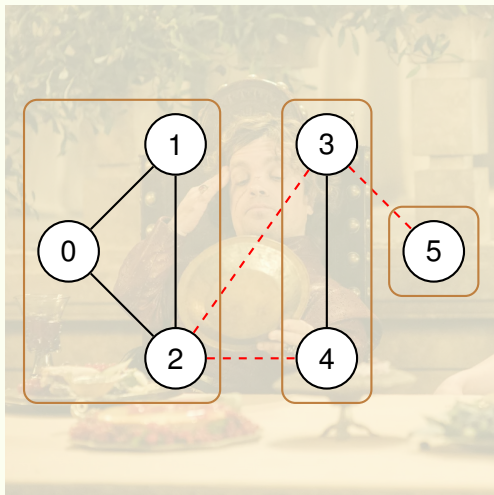
0 3

- 2 3

- 2 4

- 3 5

Una soluzione ottima si può ottenere anche solo distruggendo alleanze (o solo creandone nuove).

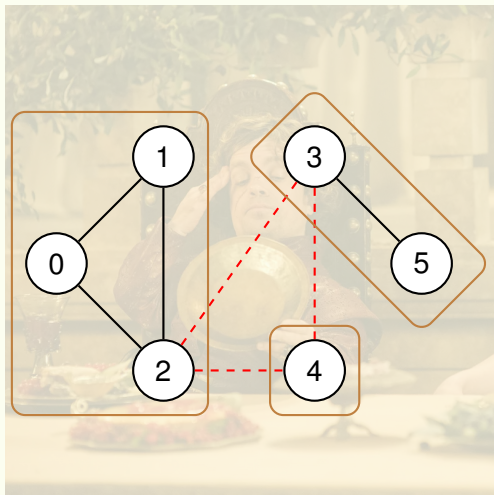


ESEMPIO OTTIMO

Soluzione ottima. 0 allenze create, 3 distrutte:

0 3
- 2 3
- 2 4
- 3 4

Soluzioni ottime possono essere ottenute in modi diversi.



ASSUNZIONI GENERALI

- $1 \leq N \leq 5.000$
- $1 \leq M \leq 1.000.000$
- Ogni grafo è non diretto.
- Ogni grafo può avere più componenti connesse.

Ci sono 20 casi di test in totale:

- in almeno 3 casi $N \leq 100$.
- in almeno 3 casi è possibile ottenere la soluzione ottima solamente creando alleanze.
- nei casi restanti non ci sono particolari limitazioni.

Per la sufficienza si possono risolvere i 3 casi risolvibili solamente creando alleanze.

I limiti di tempo e memoria sono:

- ▶ Tempo limite massimo: 5 secondi (soft limit), 5,5 secondi (hard limit).
- ▶ Memoria massima: 64 MB.
- ⇒ Limite di **40 sottoposizioni** per gruppo.
- ⇒ Potete provare con un dataset equivalente sulla vostra macchina (sito: <https://judge.science.unitn.it/slides/>).

Potete stampare le soluzioni in maniera incrementale:

- Importate `got2.h` (scaricabile da judge).
- Man mano che migliorate la soluzione, scrivetela in output terminando la riga con `***`.
- La libreria arresterà il programma prima del timeout.

```
... include delle librerie di sistema ...  
#include "got2.h"  
int main() {  
    ...  
}
```

Note:

- Il `main` va sempre dichiarato come `int main()` o `int main(void)`.
- Il correttore considererà l'**ultima soluzione** terminata da `***` quindi, anche se non stampate soluzioni multiple, terminate l'output con `***`.

COMPILARE IN LOCALE

Per testare le vostre soluzioni in locale (supponiamo che il vostro file si chiami `got2.cpp`):

- Scaricate `grader.cpp`
- Il comando di compilazione è il seguente

```
/usr/bin/g++ -DEVAL -std=c++11 -O2 -pipe -static -s -o  
got2 grader.cpp got2.cpp
```

I file `got2.cpp`, `grader.cpp` e `got2.h` devono essere nella stessa cartella.

Per sistemi Mac OS X e Windows vedere la nota nel testo.

NOTA

Per questo esercizio è necessario usare C++, non è possibile usare C.

PUNTEGGIO I

Ogni caso di test vale da 0 a 5 punti in base a quanto la vostra soluzione si avvicina al numero minimo di modifiche possibile per il grafo di input. Il punteggio massimo è di 100 punti.

Per ogni caso di test per cui la vostra soluzione fornisce un output entro i limiti di tempo e memoria otterrete il seguente punteggio \mathcal{P} :

$$\mathcal{P} = \max \left(0, \min \left(1, 1 - \frac{\text{Mods} - \text{minMods}}{\text{maxMods} - \text{minMods}} \right) \right) \cdot 5 \quad (1)$$

dove $\text{Mods} = A + R$.

- ⇒ maxMods e minMods sono rispettivamente il lower bound e l'upper bound per ogni soluzione.
- ⇒ il punteggio è dato dal numero di modifiche riscalato nell'intervallo $[\text{minMods}, \text{maxMods}]$ e moltiplicato per 5.
- ⇒ se la vostra soluzione fa più di maxMods modifiche prendete 0 punti

Se una soluzione non rispetta tutti i requisiti si ottengono **0 punti**.

- ✗ il formato di output non è rispettato.
 - ✗ la soluzione produce una rete di alleanze che non rispetta i requisiti imposti dai tavoli.
 - ✗ il numero dichiarato di alleanze aggiunte o rimosse non corrisponde a quelle specificate.
 - ✗ la soluzione aggiunge (rimuove) alleanze già esistenti (non esistenti).
 - ✗ la soluzione aggiunge (rimuove) più volte la stessa alleanza.
- ⇒ La **sufficienza è posta a 15 punti**.

L'assegnazione punti avviene in maniera competitiva:

- **3 punti** ai gruppi nel primo terzile della classifica (primo terzo della classifica);
- **2 punti** ai gruppi nel secondo terzile della classifica (secondo terzo della classifica);
- **1 punto** ai gruppi nel terzo terzile della classifica (ultimo terzo della classifica).

Vengono considerati nella classifica per l'assegnazione dei punti solamente i **gruppi che raggiungono la sufficienza** (punteggio maggiore o uguale a 15).

⇒ Classifica:

<https://judge.science.unitn.it/arena/ranking/>

Consegna: mercoledì 31 maggio 2023 ore 18:00

Per caricare il vostro codice, recatevi su

<https://judge.science.unitn.it/arena/>

Ricordiamo che nel calcolo del punteggio verrà considerata l'**ultima** soluzione consegnata.

SUGGERIMENTI

Cominciate subito a lavorare al progetto per presentarvi al prossimo ricevimento (giovedì 25 maggio) con tutte le domande che vorrete fare. In ogni caso, sappiate che:

- potete venire a ricevimento
- rispondiamo su Telegram
- risponderemo alle vostre mail

È PERMESSO:

- Discutere all'interno del gruppo
- Chiedere chiarimenti sul testo
- Chiedere opinioni su soluzioni
- Sfruttare codice fornito nei laboratori
- Utilizzare pseudocodice da libri o Wikipedia
- Richiedere aiuto (anche pesante) per la soluzione “minima”
- Venire a ricevimento
- **Mandare meme** (algoritmici) sul nostro canale Telegram: it's your time to shine!

È VIETATO:

- Discutere con altri gruppi
- Mettere il proprio codice su repository pubblici
- Utilizzare codice scritto da altri
- Condividere codice (abbiamo potenti mezzi!)
- Chiedere suggerimenti online (es: stackoverflow)
- Fare riferimenti a soluzioni del progetto nei meme

DATE E ORARI

- giovedì 25 maggio 2023 dalle 11:30 alle 13:30 (A101);
- ...;
- ...;
- ...;
- martedì 30 maggio 2023 dalle 13:30 alle 15:30 (A101);

- ⇒ Negli orari di ricevimento saremo a disposizione online.
Se avete bisogno di una mano, avvisate sul gruppo telegram.
Risponderemo in chat privata o tramite una chiamata zoom.
- ⇒ Per qualsiasi domanda mandateci una mail a:
`asd.disi@unitn.it` oppure contattateci su Telegram.