Algoritmi e Strutture Dati

Game of Thrones II: House of Algorithms (`got2`)

Testo del problema

Slides originali su: judge.science.unitn.it/slides/asd22/prog2.pdf
Quattro anni dopo gli avvenimenti che l'hanno portato sul Trono di Spade[1], Re Albert Iº si trova alle prese con vari intrighi politici.
I nobili del regno tramano contro di lui e, in segreto, complottano per spodestarlo. 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 del regno. Grazie ai sui fidi informatori, Re Albert conosce la rete di tutte le alleanze che i nobili intrattengono tra loro.

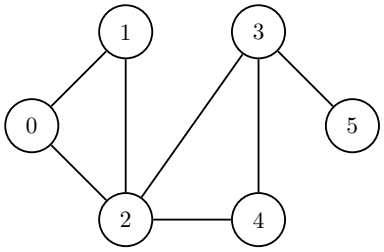

Figura 1: Una mappa con 7 alleanze tra 6 nobili prima del banchetto.

La strategia 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.
Al banchetto reale, un nobile sarà seduto a un certo tavolo *se e solo se* alleato con **tutti e soli** i nobili sedute a quel tavolo. La logistica del banchetto non sarà un problema: il mastro falegname può costruire tavoli di qualsiasi misura, forma o numero di posti.
Questo gioco di potere - con nuove alleanze che vengono formate e antiche che vengono distrutte - 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.

Obiettivo

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

→ Quali nuove alleanze occorrerà instaurare?

→ Quali vecchie alleanze dovranno essere disfatte?

Esempi

Per illustrare un esempio facciamo riferimento alla situazione descritta in Figura 1. Nelle figure, le alleanze create sono rappresentate da archi blu continui e le alleanze distrutte sono rappresentate da archi rossi tratteggiati.

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

Esempi di soluzioni non valide

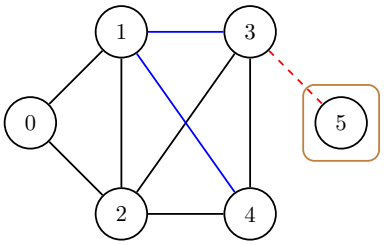

Figura 2: Soluzione non valida: i nobili $\{0, 1, 2, 3, 4\}$ non possono stare allo stesso tavolo in quanto 0 non è alleato con 3 e 4.

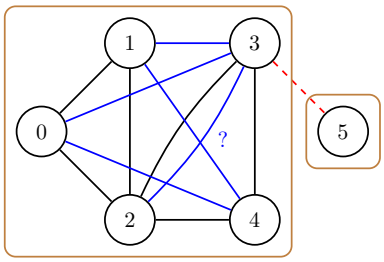

Figura 3: Soluzione non valida: 2 e 3 sono già alleati, non è possibile creare un'alleanza preesistente.

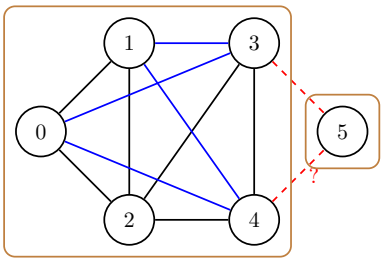

Figura 4: Soluzione non valida: 4 e 5 non sono alleati, non è possibile distruggere un'alleanza non preesistente.

Le Figure 2, 3 e 4, mostrano tre soluzioni errate.

- In Figura 2 vengono create 2 alleanze e ne viene distrutta 1: $+(1,3), +(1,4), -(3,5)$. La soluzione, tuttavia, non è valida perché i tavoli risultanti non rispettano i requisiti. I nobili $\{0, 1, 2, 3, 4\}$ non possono stare allo stesso tavolo in quanto 0 non è alleato con 3 e 4.
- In Figura 3 vengono create 5 alleanze e ne viene distrutta 1: $+(0,3), +(0,4), +(1,3), +(1,4), +(2,3), -(3,5)$. La soluzione, tuttavia, non è valida perché prevede la creazione di un'alleanza tra 2 e 3 che erano già alleati.
- In Figura 4 vengono create 4 alleanze e ne vengono distrutte 2: $+(0,3), +(0,4), +(1,3), +(1,4), -(3,5), -(4,5)$. La soluzione, tuttavia, non è valida perché prevede la distruzione di un'alleanza tra 4 e 5 che non erano precedentemente alleati.

Esempi di soluzioni valide

.1 Soluzioni valide

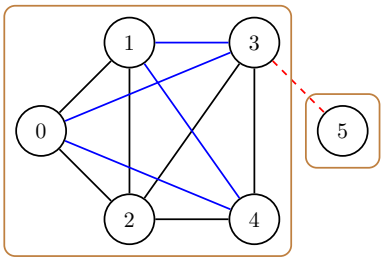

Figura 5: Soluzione valida: un tavolo con un solo nobile (5) è valido.

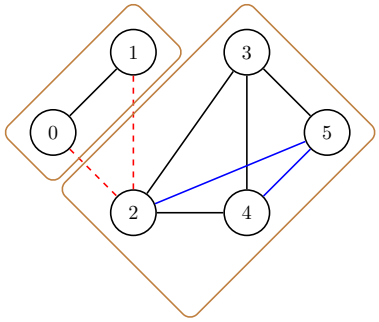

Figura 6

Le Figure 5, 6, 7 e 8 mostrano tre soluzioni valide, ma non ottime.

- In Figura 5 vengono create 4 alleanze e ne viene distrutta 1: $+(0,3), +(0,4), +(1,3), +(1,4), -(3,5)$. Un tavolo con un solo nobile isolato (5) è una disposizione valida.

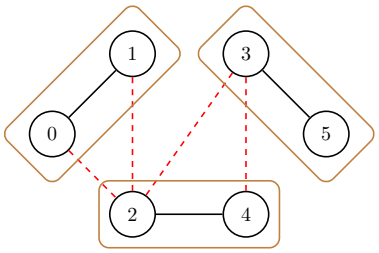

Figura 7: Una soluzione valida ottenuta solo distruggendo alleanze.

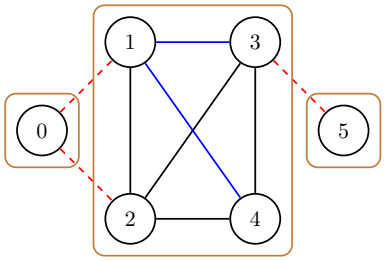

Figura 8: Un'altra soluzione valida ottenuta creando 2 alleanze e distruggendone 3.

- In Figura 6 vengono create 2 alleanze e ne vengono distrutte 2: $+(2,5), +(4,5), -(0,2), -(1,2)$. Un tavolo con due nobili alleati tra loro $(0,1)$ è una disposizione valida.
- In Figura 6 non vengono create alleanze e ne vengono distrutte 4: $-(0,2), -(1,2), -(2,3), -(3,4)$. Una soluzione valida può essere ottenuta solamente distruggendo alleanze o creandone.
- In Figura 8 vengono create 2 alleanze e ne vengono distrutte 3: $+(1,3), +(1,4). -(0,1), -(0,2), -(3,5)$.

.2 Soluzioni ottime

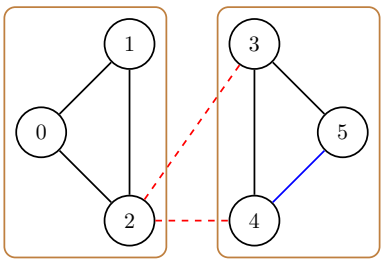

Figura 9: Soluzione ottima: 1 alleanza creata, 2 distrutte.

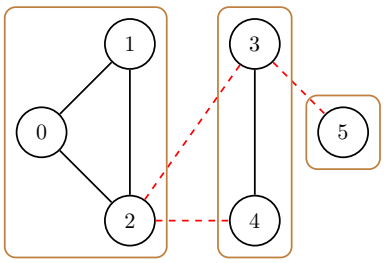

Figura 10: Soluzione ottima: ottenuta solo distruggendo alleanze (3 alleanze distrutte).

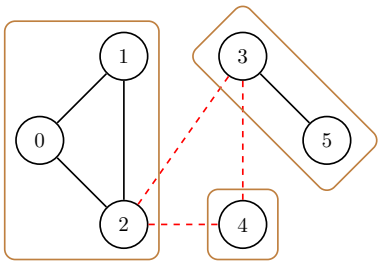

Figura 11: Soluzione ottima: ottenuta solo distruggendo alleanze (3 alleanze distrutte).

Le Figure 9, 10 e 11 mostrano tre soluzioni ottime, ovvero con il minimo numero di alleanze modificate, in questo caso 3.

- In Figura 9 viene creata 1 alleanza e ne vengono distrutte 2: $+(4,5), -(2,3), -(2,4)$.
- In Figura 10 non vengono create alleanze e ne vengono distrutte 3: $-(2,3), -(2,4), -(3,5)$. Una soluzione ottima può essere ottenuta solamente distruggendo alleanze o creandone.
- In Figura 11 non vengono create alleanze e ne vengono distrutte 3: $-(2,3), -(2,4), -(3,4)$. Possono esistere più soluzioni ottime alternative.

Input/Output

Input: un file con la rete delle alleanze tra nobili prima del banchetto. In particolare, il file di input è costituito da $1+M$ 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`) e v (`int`), rappresentanti i due nobili alleati.

Nota: le allenze sono bidirezionali: se u è alleato con v, è implicito che v è alleato con u.

Output: un file con le vostre proposte di soluzione, ogni proposta è composta da $1 + A + R + 1$ righe:

- la prima riga del file di output deve contenere due interi A (`int`) e R (`int`): rispettivamente il numero di alleanze create e distrutte.
- le successive $A + R$ righe specificano le alleanze tra nobili create e distrutte:
  - ciascuna riga è costituita da un segno $+$ o $-$, per specificare se l'alleanza è creata o distrutta, seguito da due interi u e v che specificano i due nobili che costituiscono l'alleanza in questione.
  - la soluzione deve terminare con una riga contenente la stringa $***$ per specificare che si tratta dell'ultima soluzione valida stampata.

Nota: questo vi consente di stampare più soluzioni finché il tempo a disposizione per l'esecuzione del programma non scade. Solo l'ultima soluzione terminata da $***$ verrà considerata per la valutazione del punteggio.

Specifica dell'output:

```
A R
+ u_1 v_1
...
+ u_A v_A
- u_{A+1} v_{A+1}
...
- u_{A+R} v_{A+R}
***
```

Punteggio

- Ci sono 20 casi di test: ogni test assegna un punteggio di massimo 5 punti per un totale massimo teorico di 100 punti.
- Una soluzione è valida se rispetta tutte le richieste. Soluzioni non valide fanno **zero punti**. Per esempio, in tutti i casi seguenti la soluzione da zero punti:
  - il formato di output non è rispettato.
  - la soluzione produce tavoli con nobili alleati non seduti insieme o nobili seduti a uno stesso tavolo ma non alleati.
  - il numero dichiarato di alleanze aggiunte (A) o rimosse (R) non corrisponde a quelle specificate.
  - la soluzione aggiunge (rimuove) alleanze già esistenti (non esistenti).
  - la soluzione aggiunge (rimuove) più volte la stessa alleanza.
- Il punteggio viene calcolato usando l'**ultima soluzione** terminata con tre asterischi `***`.

Per soluzioni valide, i parametri di valutazione sono i seguenti:

- *Mods*: il numero di modifiche apportate alla rete delle allenanze, ovvero $A + R$;
- *minMods*: lower bound del numero di modifiche;
- *maxMods*: upper bound del numero di modifiche;

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

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

Nota: non è necessario che calcoliate il punteggio della vostra soluzione - vi servirebbero i parametri dei bound $minMods$ e $maxMods$ che non avete. Il vostro obiettivo è in ogni caso di **minimizzare il numero di alleanze modificate, ovvero create e distrutte ($A + R$)**.

Esempi (punteggio)

Prendendo l'esempio di Figura 7, la risposta del sistema del valutazione è:

```
Soluzione valida: Mods: 4, minMods: 3, maxMods: 5 0.500000
```

Il punteggio è calcolato nel modo seguente:

$$P = (1 - \frac{4-3}{5-3}) \cdot 5 = 0,5 \cdot 5 = 2,50$$

Prendendo l'esempio di Figura 9, la risposta del sistema del valutazione è:

```
Soluzione valida: Mods: 3, minMods: 3, maxMods: 5 1.000000
```

Il punteggio è calcolato nel modo seguente:

$$P = (1 - \frac{3-3}{5-3}) \cdot 5 = 1,00 \cdot 5 = 5,00$$

Valutazione

Per la valutazione del progetto:

- Conta il punteggio dell'**ultimo sorgente** inviato al sistema;
- Il progetto è superato con un punteggio non inferiore a 15 punti;
- C'è un limite di 40 sottoposizioni per gruppo;

Limiti e assunzioni

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

Casi di test

- Ci sono 20 casi di test in totale:
  - in almeno 3 casi $N \leq 100$.
  - almeno 3 casi si possono risolvere solamente creando alleanze.

Limiti delle risorse

- Tempo di esecuzione: 5 secondi (soft limit), 5,5 secondi (hard limit)
- Memoria: 64 MB

Dataset di esempio

Per gli input forniti nel dataset di esempio non è stata calcolata una soluzione ottima. Per questo motivo il dataset non contiene anche i relativi output, solitamente messi a disposizione.

Istruzioni di compilazione

Di seguito riportiamo le istruzioni per testare i vostri programmi su vari sistemi. Si suppone che il sorgente con il vostro codice si chiami file `got2.cpp`. I file `got2.cpp`, `grader.cpp` e `got2.h` devo stare nella stessa cartella.

Sistemi GNU/Linux

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

Sistemi Mac OS X

Su sistemi Mac OS X usate il seguente comando di compilazione:

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

Se ottente un errore del tipo: `use of undeclared identifier quick_exit`, sostituite in `grader.cpp` l'istruzione `quick_exit(EXIT_SUCCESS);` con `exit(EXIT_SUCCESS);`.

Sistemi Windows

Per il sistema Windows 10 potete installare il "Windows Subsystem for Linux"[2]. Successivamente potete installare i tool necessari per usare Visual Studio Code[3] o Visual Studio 2017[4] seguendo le relative guide riportate nelle note. Usando questo sistema fate attenzione a dove salvate i file e a quale nome gli date in quanto potreste avere delle difficoltà con percorsi che contengano spazi e caratteri speciali.
In alternativa, o per sistemi precedenti a Windows 10 potete installare *Cygwin*[5], un ambiente completamente POSIX-compatibile per Windows. Anche in questo caso esistono guide per configurare i comuni editor disponibili su Windows di modo che utilizzino l'ambiente Cygwin, come per esempio Visual Studio[6].
Una volta installato Cygwin è possibile simulare quanto avviane su arena compilando il proprio sorgente senza includere l'header `got2.h` e il grader `grader.cpp`:

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

e lanciare il comando come:

```
timeout.exe 5 ./got2
```

`timeout.exe` arresterà il programma dopo 5 secondi.

[2] `https://docs.microsoft.com/en-us/windows/wsl/install-win10`
[3] `https://code.visualstudio.com/docs/cpp/config-wsl`
[4] `https://devblogs.microsoft.com/cppblog/targeting-windows-subsystem-for-linux-from-visual-studio/`
[5] `https://www.cygwin.com/`
[6] `https://devblogs.microsoft.com/cppblog/using-mingw-and-cygwin-with-visual-cpp-and-open-folder/`

Esempi di input/output

File `input.txt`	File `output.txt`

`6 7`
`0 1`
`2 0`
`2 3`
`1 2`
`3 4`
`4 2`
`3 5`
	`1 2`
	`+ 4 5`
	`- 2 3`
	`- 2 4`
	`***`

File `input.txt`	File `output.txt`

`18 20`
`0 1`
`7 8`
`16 17`
`4 7`
`5 10`
`8 13`
`8 14`
`5 16`
`5 6`
`12 13`
`0 2`
`11 12`
`8 9`
`2 3`
`11 13`
`14 15`
`0 5`
`3 4`
`2 4`
`7 9`
	`0 8`
	`- 0 1`
	`- 0 2`
	`- 4 7`
	`- 5 10`
	`- 5 16`
	`- 5 6`
	`- 8 13`
	`- 8 14`
	`***`

File `input.txt`	File `output.txt`

`15 19`
`0 1`
`1 2`
`7 8`
`2 7`
`6 7`
`12 14`
`4 5`
`12 13`
`13 14`
`10 12`
`9 10`
`5 14`
`9 11`
`3 9`
`0 9`
`3 4`
`0 2`
`5 8`
`3 5`
	`0 8`
	`- 0 9`
	`- 2 7`
	`- 3 9`
	`- 5 14`
	`- 5 8`
	`- 7 8`
	`- 9 11`
	`- 10 12`
	`***`