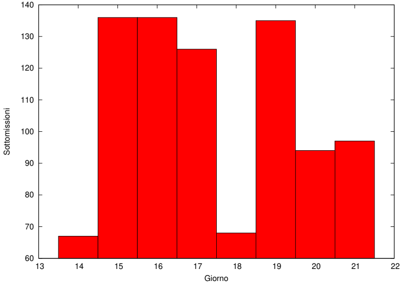
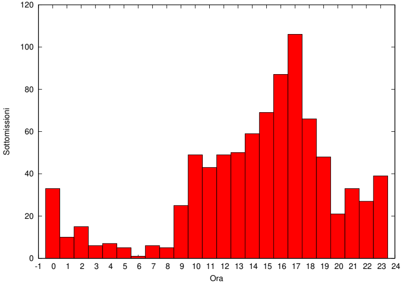
AmongASD

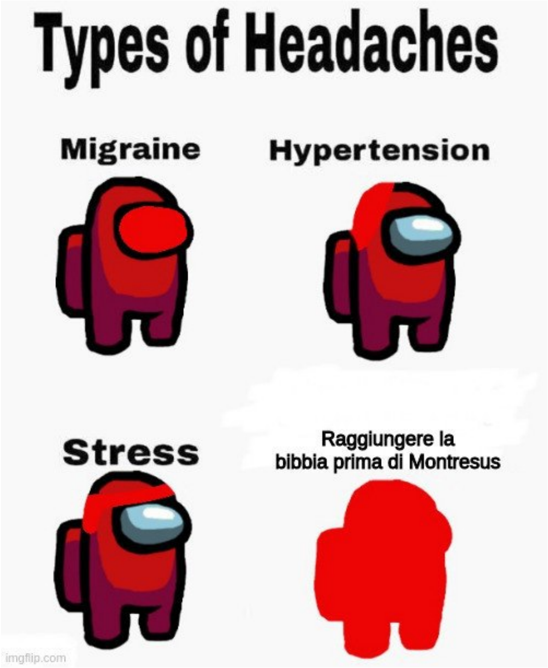
STATISTICHE

Numero sottoposizioni: `860`





- 80 gruppi hanno fatto almeno una sottoposizione, di cui 79 hanno raggiunto la sufficienza;
- 204 studenti iscritti, di cui 195 appartenenti a gruppi che hanno fatto almeno una sottoposizione (1 ritiro);


Types of Headaches
Migraine
Hypertension
Stress
Raggiungere la
bibbia prima di Montresus
imgflip.com

RISULTATI


Jim Sturgess Kate Bosworth con Laurence Fishburne y Kevin Spacey
A
J
21
BLACKJACK

RISULTATI

PUNTEGGI

- $P < 30$ $\longrightarrow$ progetto non passato.
- $30 \leq P < 70$ $\longrightarrow$ 1 punto bonus (17 gruppi).
- $70 \leq P < 100$ $\longrightarrow$ 2 punti bonus (30 gruppi).
- $P = 100$ $\longrightarrow$ 3 punti bonus (32 gruppi).

Classifiche e sorgenti sul sito (controllate i numeri di matricola):
`https://judge.science.unitn.it/slides/asd22/classifica_prog1.pdf`

Il problema I

Consideriamo un grafo orientato e pesato $G = (V, E)$ con N nodi, M archi con peso fissato e K archi “con ventole”, il cui peso può variare in un intervallo $[T_{\mathrm{min}}, T_{\mathrm{max}}]$.

Il Prof. Montresus parte dal nodo I, gli studenti partono dal nodo S, ed entrambi vogliono raggiungere il nodo F in cui si trova il FabLab.

Il problema II

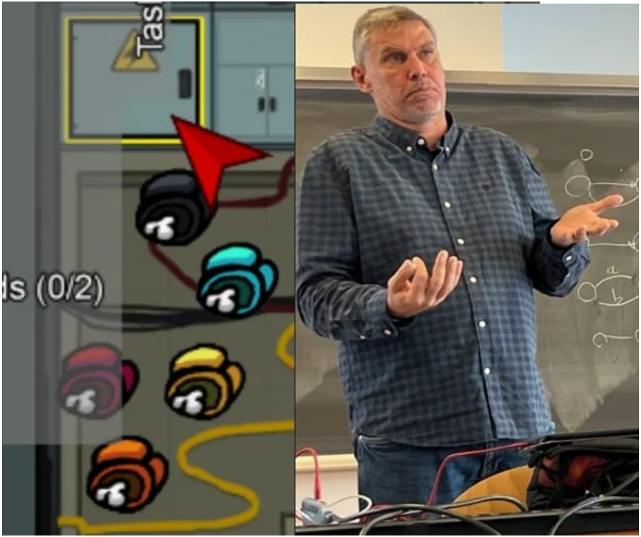
Il Prof. Montresus ha il potere di decidere a quale valore impostare il peso dei K archi con ventole.

Impostati i pesi, sia il Prof. Montresus che gli studenti seguono un **percorso ottimo** per arrivare al FabLab, ossia quello di **peso minimo**.

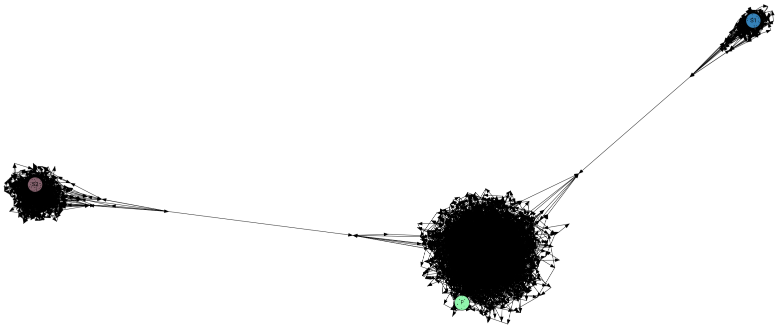
Problema

Riuscirà il Prof. Montresus a **trovare una configurazione di pesi che gli consenta di arrivare al FabLab prima degli studenti**?

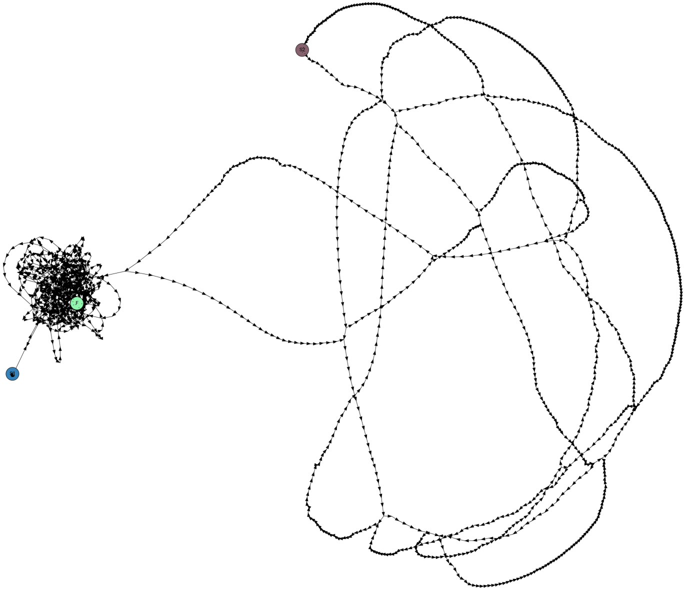
Quando il comitato etico ti chiede cosa è successo nel FabLab



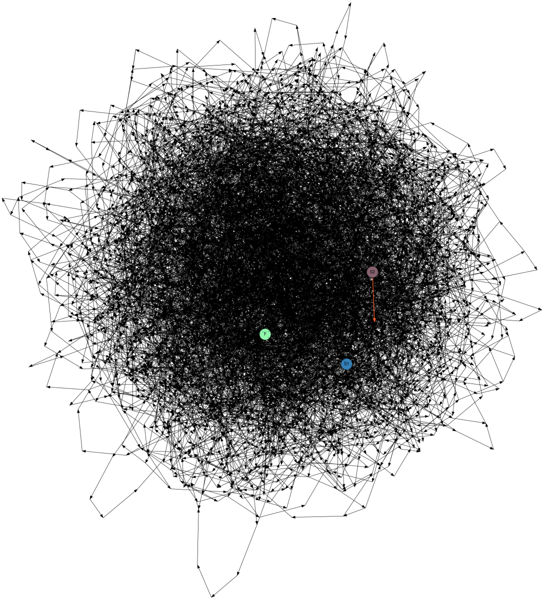
Testcase - componenti fortemente connesse



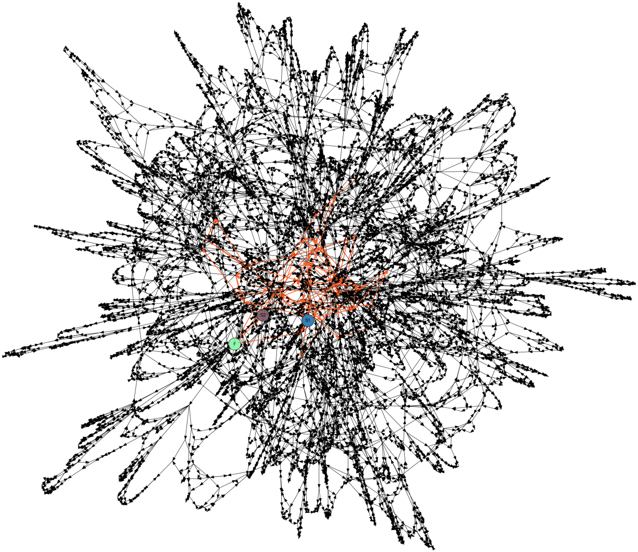
TESTCASE - COMPONENTI FORTEMENTE CONNESSE



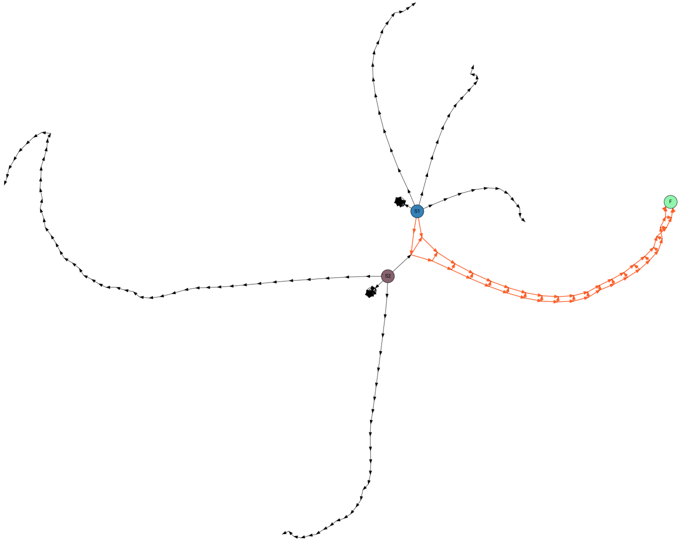
TESTCASE - GRAFO RANDOM, $K = 1$



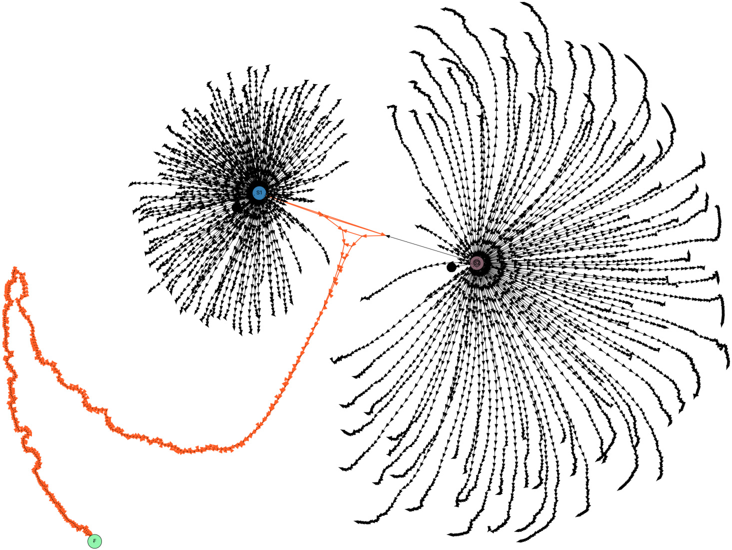
TESTCASE - GRAFO RANDOM, $K \geq 2$



TESTCASE - PERCORSI CONDIVISI, $K \geq 2$



Testcase - percorsi condivisi, $K \geq 2$



Sottoproblema: $K = 0$, Grafi non pesati

Osservazioni

- Se $K = 0$, il Prof. Montresus non deve prendere alcuna decisione, deve solo controllare se $\text{dist}_I(F) < \text{dist}_S(F)$ (rispett. $=, >$) per vedere se il risultato sarà una vittoria (rispett. pareggio, sconfitta)
- Se ogni arco ha peso 1, possiamo ignorare i pesi
$\implies \text{dist}_I(F), \text{dist}_S(F)$ possono essere trovate con una BFS

Quindi, è sufficiente fare due visite in ampiezza da I e da S per verificare il vincitore. Durante la visita possiamo salvarci il percorso fatto dai due nodi.

- ⇒ soluzione: `sol_30.cpp`
- ⇒ complessità: $O(N + M)$, 70 SLOC
- ⇒ 30 punti

SOTTOPROBLEMA: GRAFI PESATI, $K = 0$

Nel caso il grafo in input sia pesato, non è più sufficiente una BFS per trovare i percorsi minimi.

Come teniamo conto dei pesi degli archi per trovare il cammino di peso minimo per arrivare ad F?

PROBLEMA GENERALE: GRAFI PESATI, $K = 0$

PROBLEMA GENERALE: GRAFI PESATI, $K = 0$

ALGORITMO DI DIJKSTRA

L'algoritmo di Dijkstra permette di calcolare le **distanza minime** dei nodi **in un grafo pesato** rispetto a un nodo sorgente.

Invece di una semplice coda, usiamo una **coda con priorità**, che ci consente di **estrarre** ogni volta **il nodo** non ancora visitato con **distanza minima**.

Sottoproblema: Grafi pesati, $K = 0$

```
typedef pair<int, int> ii;
...
vector<int> dist(n, INF);
dist[source] = 0;
priority_queue<ii, vector<ii>, greater<ii>> q;
q.emplace(0, source);
while (!q.empty()) {
    int dv = q.top().first, v = q.top().second;
    q.pop(); // estraggo il nodo con dist minima O(logN)
    if (dv != dist[v]) continue;
    for (int e: g[v]) {
        int to = edges[e].to;
        int newdist = dist[v] + edges[e].w;
        if (newdist < dist[to]) { // O(M) aggiornamenti
            dist[to] = newdist;
            q.emplace(dist[to], to); // O(logN)
        }
    }
}
```

Problema generale: Grafi pesati, $K = 0$

Se non ci sono corridoi con ventole, il problema si risolve con gli stessi controlli visti prima, usando Dijkstra invece che BFS.

COMPLESSITÀ

- ⇒ soluzione: `sol_50_dijkstra_k0.cpp`
- ⇒ complessità: $O(M \cdot \log N)$, 75 SLOC
- ⇒ 50 punti

SOTTOPROBLEMA: GRAFI PESATI, $K = 1$

Sottoproblema: Grafi pesati, $K = 1$

In questo caso, il Prof. Montresus deve prendere una decisione su un solo arco.

Deve però scegliere un valore compreso tra T_{min} e T_{max}. Ha senso provare tutti i valori dell'intervallo?

Sottoproblema: Grafi pesati, $K = 1$

Osservazione

L'arco con ventola servirà per avvantaggiare l'impostore, oppure per svantaggiare lo studente, oppure resterà inutilizzato.

Di conseguenza, converrà sempre settarlo al **peso minimo** (per tentare di **avvantaggiare** I) o al **peso massimo** (per tentare di **svantaggiare** S). Nel caso in cui la scelta del peso non influenzi il vincitore, la scelta del valore è indifferente.

SOTTOPROBLEMA: GRAFI PESATI, $K = 1$

Si calcolano vincitore e distanze **due volte**: la prima settando il peso dell'arco al **minimo**, la seconda settandolo al **massimo**.

Si stampa in output la decisione più vantaggiosa per I.
Questa soluzione richiede 4 chiamate dell'algoritmo di Dijkstra (2 per ciascun tentativo).

- ⇒ soluzione: `sol_70_dijkstra_k1.cpp`
- ⇒ complessità: $O(M \cdot \log N)$, 152 SLOC
- ⇒ 70 punti

Caso generale: Grafi pesati, $K \geq 2$

La soluzione precedente non può essere usata per K troppo grandi, infatti provando tutte le possibili combinazioni di pesi dovremmo fare nel caso peggiore 2^K volte Dijkstra!

Caso generale: Grafi pesati, $K \geq 2$

Idea

In un grafo generico, un arco (a, b) è **vantaggioso per l'impostore** se $\text{dist}_I(a) < \text{dist}_S(a)$: qualunque sia il percorso dal nodo b a F, se I e S passano per l'arco (a, b) arriverà sempre prima I.

L'idea è dunque effettuare una visita del grafo e **settare al minimo gli archi con ventole che sono vantaggiosi** per I, al massimo gli altri.

Caso generale: Grafi pesati, $K \geq 2$

Dividiamo il calcolo della soluzione in due parti:

- all'inizio cerchiamo di far vincere il Prof. Montresus;
- in seguito, se non siamo riusciti a far vincere l'impostore, proviamo a ottenere un pareggio.

Alla fine stampiamo il miglior risultato ottenuto.

Sottoproblema: proviamo a far vincere I

Facciamo partire l'algoritmo di Dijkstra contemporaneamente da I e da S, inserendoli entrambi in coda con distanza 0.

Per ogni nodo, cerchiamo di capire quale dei due giocatori arriva prima, assegnando i pesi sulle ventole di conseguenza.

SOTTOPROBLEMA: PROVIAMO A FAR VINCERE I

Quando un nodo a viene estratto dalla coda, controllo la sua distanza dall'impostore e dallo studente:

- $\text{dist}_I(a) < \text{dist}_S(a)$: ***I* arriva prima di *S*** al nodo a. Di conseguenza settiamo i pesi di tutte le **ventole** (a, b) **uscenti** dal nodo a **al valore minimo** possibile, con l'obiettivo di avvantaggiare I.
- $\text{dist}_I(a) > \text{dist}_S(a)$: ***S* arriva prima di *I*** al nodo a. Di conseguenza settiamo i pesi di tutte le **ventole** (a, b) **uscenti** dal nodo a **al valore massimo** possibile, con l'obiettivo di svantaggiare S.
- $\text{dist}_I(a) = \text{dist}_S(a)$: ***I* e *S* arrivano contemporaneamente** al nodo a. Siccome vogliamo evitare un pareggio (per ora), bisogna evitare che questo percorso diventi un percorso minimo per entrambi. Di conseguenza settiamo i pesi di tutte le **ventole uscenti** dal nodo a **al valore massimo**.

Sottoproblema: proviamo a far vincere I

Se con questo procedimento riusciamo ad arrivare a una soluzione in cui il vincitore è l'impostore, stampiamo l'output ottenuto e concludiamo.

In caso non fosse possibile ottenere una vittoria per *I*, **proviamo a vedere se è possibile raggiungere un pareggio**.
Per fare ciò, ripetiamo lo stesso procedimento visto precedentemente, con un'unica modifica:

nel caso in cui ci sia un nodo *a* con $\mathbf{dist}_{\boldsymbol{I}}\boldsymbol{(a)} = \mathbf{dist}_{\boldsymbol{S}}\boldsymbol{(a)}$, settiamo i **pesi di tutte le ventole uscenti dal nodo 'a' al valore minimo**, con l'obiettivo di far diventare il percorso corrente un **percorso minimo per entrambi**, con cui arriveranno contemporaneamente al FabLab.

Se con questo procedimento riusciamo ad arrivare a una soluzione in cui il Prof. Montresus e lo studente pareggiano, stampiamo l'output ottenuto e concludiamo.

Altrimenti, siamo in una situazione in cui I non può mai vincere. In questo caso, basta stampare una qualsiasi configurazione di pesi e percorsi corrispondenti.

Complessità

⇒ soluzione: `sol_100_mlogn.cpp`

⇒ complessità: $O(M \cdot \log N)$, 138 SLOC

⇒ 100 punti