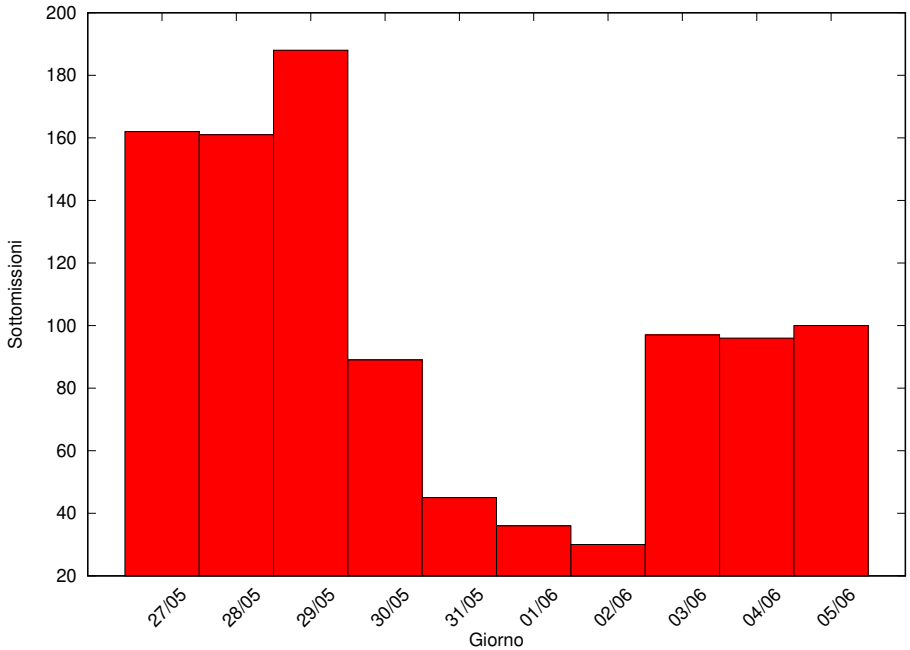
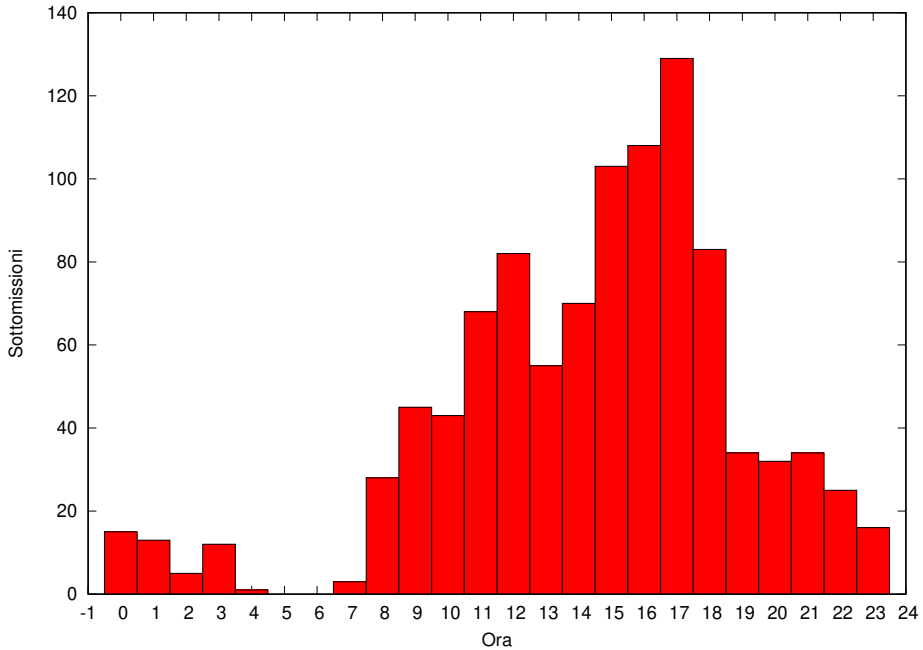
Secondo Progetto ASD 2024/2025

Statistiche

Numero sottoposizioni: `1004`





- 50 gruppi hanno fatto almeno una sottoposizione, di cui 49 hanno raggiunto la sufficienza;
- 157 studenti iscritti, di cui 128 appartenenti a gruppi che hanno fatto almeno una sottoposizione;

Risultati

Punteggi

- $P < 30$ $\longrightarrow$ progetto non passato.
- $30 \leq P < 75$ $\longrightarrow$ 1 punto bonus (10 gruppi).
- $75 \leq P < 88$ $\longrightarrow$ 2 punti bonus (19 gruppi).
- $88 \leq P < 88.45$ $\longrightarrow$ 3 punti bonus (6 gruppi).
- $P \geq 88.45$ $\longrightarrow$ 3.5 punti bonus (14 gruppi).

Classifiche e sorgenti sul sito (controllate i numeri di matricola):

https://asdlab.disi.unitn.it/slides/asd24/classifica_prog2.pdf

Problema

Vi è dato un insieme U di elementi e una collezione $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ di sottoinsiemi di U.

Sapendo di poter selezionare liberamente un sottoinsieme di elementi da U:

- qual è il sottoinsieme minimo $H \subseteq U$ tale che $H \cap S_i \neq \emptyset$ per ogni $S_i \in \mathcal{S}$?
- ovvero, qual è il numero minimo di elementi che "colpiscono" tutti i sottoinsiemi della collezione?

In letteratura, questo problema è noto come **hitting set problem** ed è uno dei problemi classici nella teoria della complessità.

Possiamo rappresentare i dati come un ipergrafo, dove gli elementi di U sono i nodi e gli insiemi S_i sono gli iperarchi. Quando tutti gli insiemi S_i hanno cardinalità 2, l'ipergrafo si riduce a un grafo e il problema corrisponde al **Vertex Cover**.

Sottoproblema sufficienza: hitting set piccoli

Idea: nonostante il progetto sia sugli algoritmi approssimati, nessuno ha mai detto che posso essere usate altre tecniche, come dire, più brutali... **Yes, we have Brute Force at home.**
Dati quindi 24 votanti o meno, posso provare tutte le possibili combinazioni di votanti in ordine crescente rispetto al loro numero.
L'implementazione migliore usa un bitset come base.
Conseguenza: Una volta trovata **una prima soluzione valida**, essa è **ottima.**

Complessità

- ⇒ soluzione: `brute-force.cpp`
- ⇒ complessità:

$$\mathcal{O}(2^n \cdot N \cdot M)$$

- ⇒ 30 punti visto che 6 casi vengono coperti

Tutto questo hype per così poco?
No.
Perché non farsi furbi e semplicemente controllare se $N < 24$ con un bel if? Se sì, chiamiamo questo algoritmo.

La prima soluzione del progetto

La prima soluzione che ha ottenuto punti al progetto è stata...

Idea

Scorriamo tutte le riunioni in ordine e, per ognuna:

- aggiungiamo il primo votante della riunione alla soluzione,
- solo se non è già stato selezionato in precedenza.

Soluzione semplice che rappresenta un buon punto di partenza e può essere utilizzata come baseline per confronti successivi.

Complessità

$\Rightarrow$ complessità:

$$\mathcal{O}(M \cdot N)$$

$\Rightarrow \approx 45$ punti

Strategia greedy

Possiamo scegliere in modo più efficace quali votanti inserire nel nostro hitting set. Per farlo utilizziamo un approccio **greedy**.

Idea

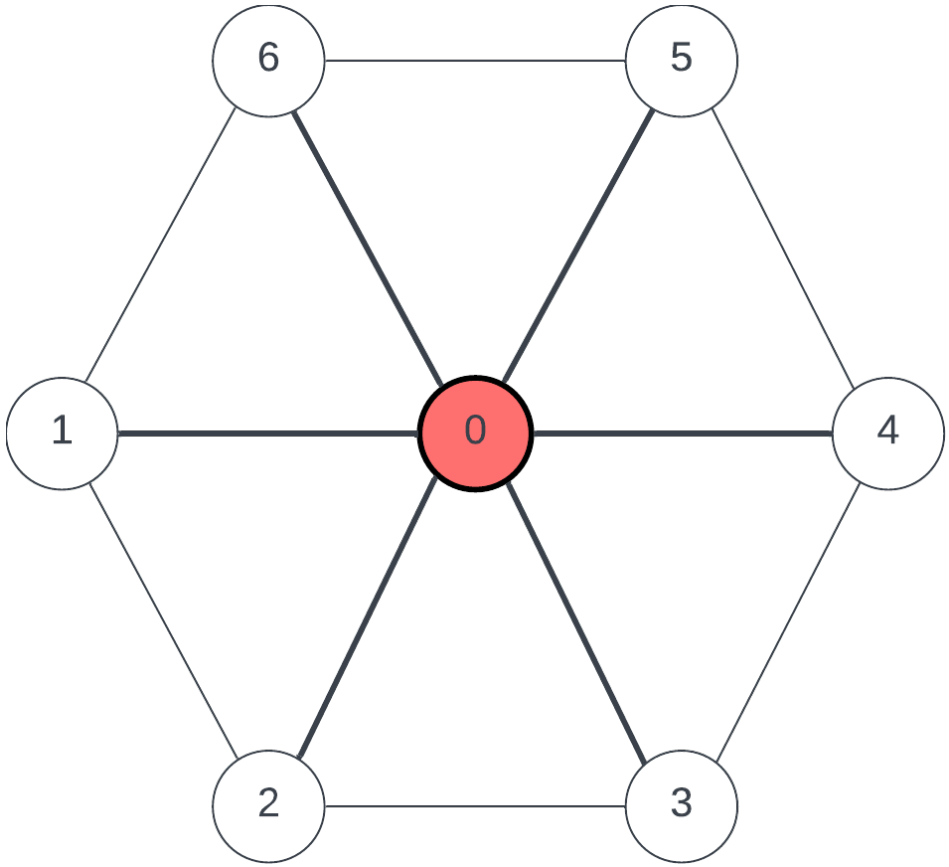
Ripetere finché tutte le riunioni sono coperte:

- scegliere il votante coinvolto nel maggior numero di riunioni ancora scoperte;
- aggiungerlo alla soluzione;
- rimuovere tutte le riunioni in cui è coinvolto.

Scelta locale, ma spesso efficace. Vediamola su un esempio.

Esempio I - greedy in azione

- 7 votanti, 12 riunioni (ogni riunione ha esattamente 2 votanti)
- La greedy seleziona il nodo con grado massimo, ripetutamente.
- Ad esempio, il nodo 0 è coinvolto in 6 riunioni $\rightarrow$ viene selezionato per primo.
- Dopo aver selezionato 0, molte riunioni sono già coperte.



Esempio II - greedy sub-ottimo

```
5 6
3 1 2 3
3 1 3 4
3 1 4 5
3 1 2 5
2 2 3
2 4 5
```

- La **greedy** seleziona prima il nodo **1** (presente in 4 riunioni), poi **3** e infine **5**, ottenendo $\{1, 3, 5\}$.
- Tuttavia, la soluzione ottima è $\{\mathbf{2}, \mathbf{4}\}$: copre tutte le riunioni con soli 2 votanti.

La greedy effettua una scelta locale non lungimirante, trascurando soluzioni globalmente migliori.

Osservazioni sulla greedy

- La greedy non garantisce la soluzione ottima, ma spesso ottiene un buon hitting set.
- **Garanzia teorica**: si può dimostrare che la greedy trova una soluzione con cardinalità al più H_d volte quella ottima, dove:
  - d = cardinalità massima dei sottoinsiemi in $\mathcal{S}$
  - $H_d = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{d}$ (numero armonico)
- **Esempi pratici**:
  - $d = 2$: $H_2 = 1.5$ (fattore di approssimazione 1.5)
  - $d = 3$: $H_3 \approx 1.83$ (fattore di approssimazione $\sim$1.8)
  - d grande: $H_d \approx \ln(d)$ (fattore logaritmico)

Significato: nel caso peggiore, la greedy trova una soluzione che è al massimo H_d volte più grande di quella ottima.

Complessità

$\Rightarrow$ complessità:

$$\mathcal{O}(N \cdot M)$$

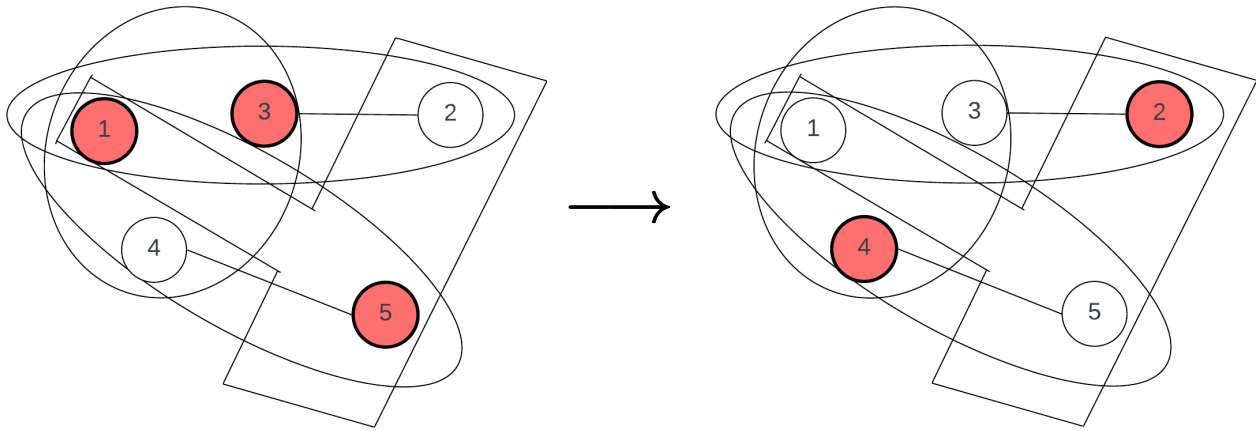
$\Rightarrow \approx$ 88 punti

Migliorare una soluzione: ricerca locale

Dato una soluzione greedy, possiamo alterarla in modo controllato ripetutamente e vedere se il numero dei votanti diminuisce.

Osservazione

Posso andare ad aggiungere un certo numero di votanti. Poi rimuovo quelli ridondanti, ossia quelli le cui riunioni sono coperti da altri votanti.



Migliorare una soluzione: ricerca locale

- Quanti però aggiungerne? In molti hanno provato ad aggiungerne pochi in modo randomico.
- **Problema:** non si riesce ad uscire dalla zona del minimo locale del greedy.
- Bisogna capire quale è il numero giusto da aggiungere volta per volta...
- Come? **Provando!** Il migliore sembra essersi dimostrato aggiungere all'inizio 15 votanti e poi dopo un certo numero di iterazioni (ordine di qualche migliaia) scendere a 5.

Migliorare la Soluzione: Ricerca Locale

⇒ soluzione: `greedy-local-search.cpp`

⇒ complessità:

$$\mathcal{O}(N \cdot M)$$

⇒ $\approx$ 90 punti