Secondo Progetto ASD 2021/2022


RIDERS
PER IL CONCERTONE

Il sindaco più amato

Dopo aver migliorato di gran lunga la viabilità della ridente cittadina[1], il sindaco di Trento e il suo partito “Algoritmi, Cricche e Libertà”, hanno conquistato ancora più consensi tra la popolazione.

[1] Progetto “Cicli Ciclabili” a.a. 2019/2020: c’è della lore qui.

Competizioni affollate

Come è ben noto, il sindaco ama organizzare competizioni algoritmiche. Anno dopo anno, anche grazie alle vittorie dei cittadini, ha notato un incremento esponenziale nel numero di partecipanti. Ha quindi cercato di prevedere quante pizze comprare per Hashcode 2022.

Figura: Una delle tante vittorie dei cittadini

Competizioni affollate

Non sempre i dati del passato sono indicativi per il futuro, però il sindaco è abituato a modellare i suoi algoritmi per il caso pessimo.

Competizioni affollate

Non sempre i dati del passato sono indicativi per il futuro, però il sindaco è abituato a modellare i suoi algoritmi per il caso pessimo. Quindi ha comprato 10^7 pizze per quest'anno. Purtroppo le previsioni sono state un filino ottimistiche, Hashcode 2022 è andato ed adesso non sa come smaltire le pizze in eccesso.

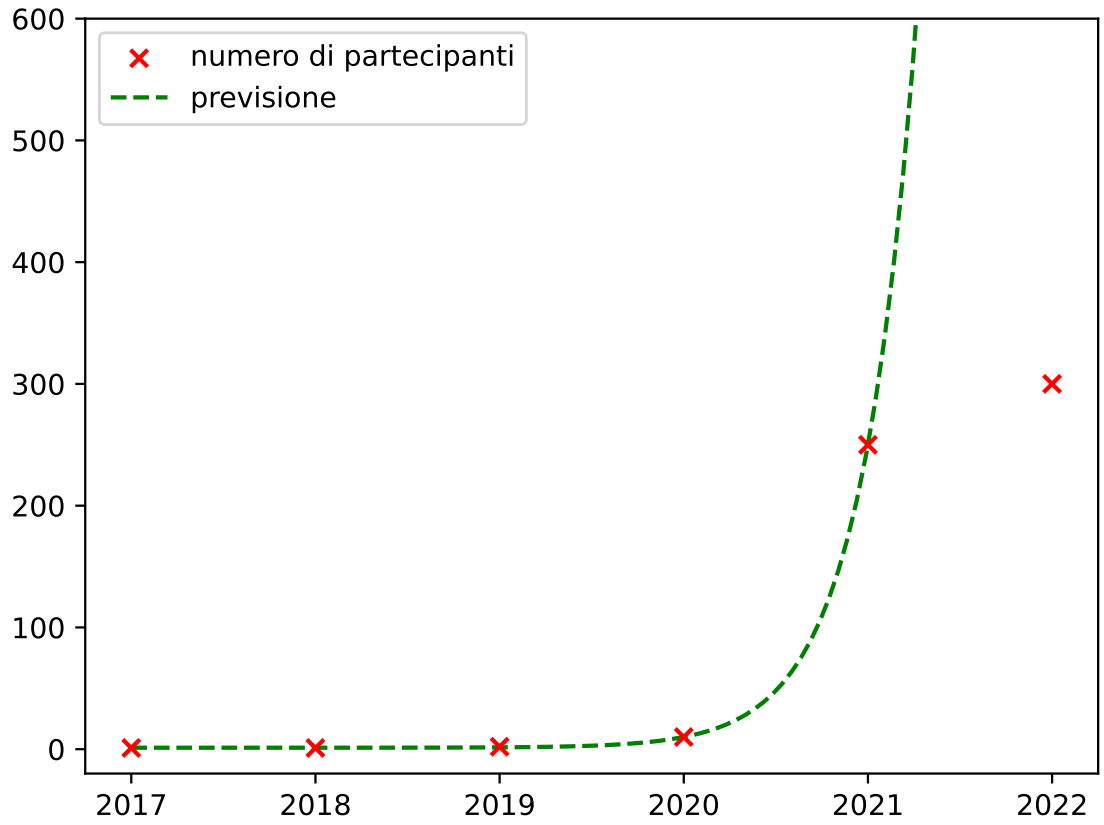


Figura: La previsione del sindaco

Il grande evento

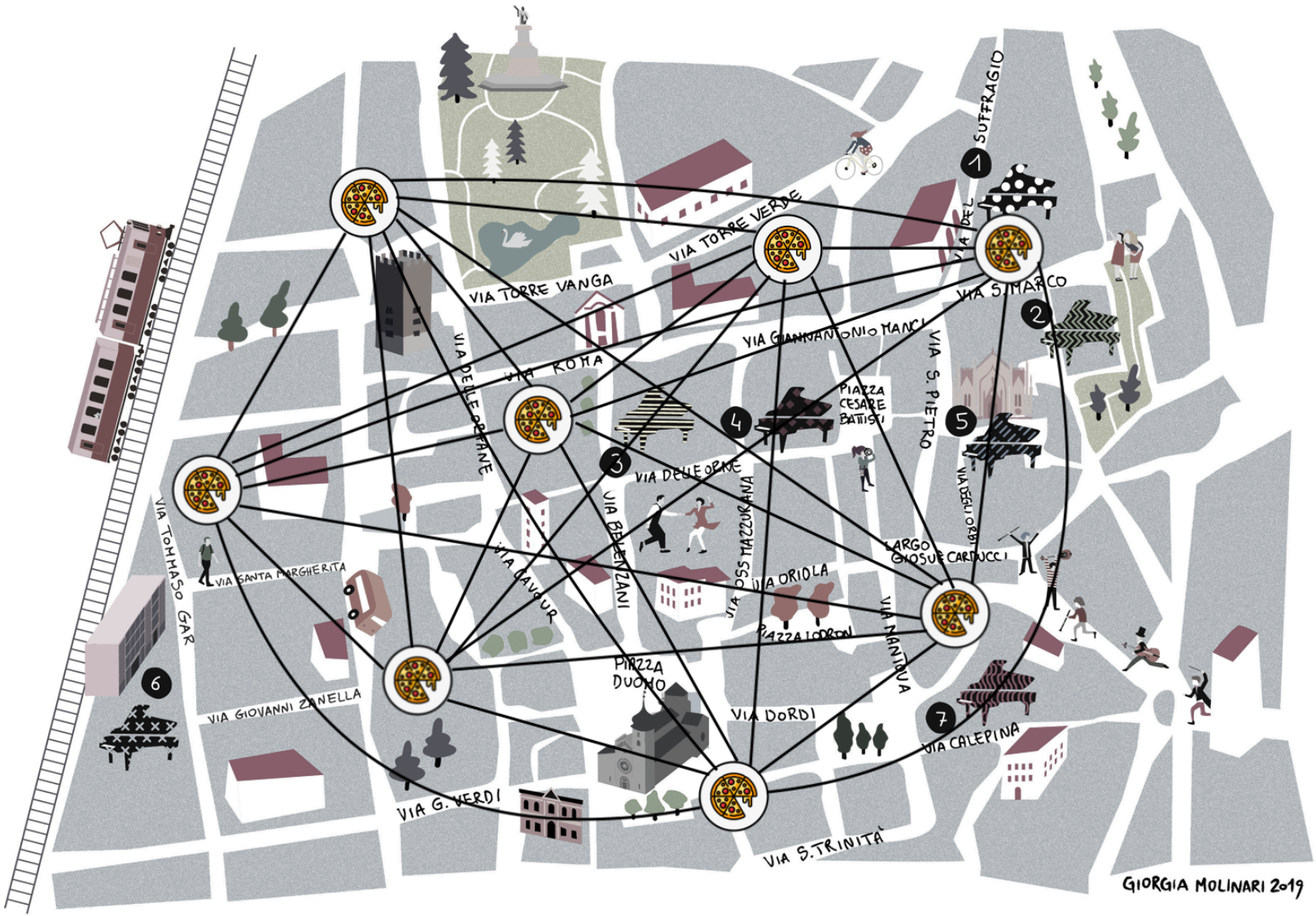
Essendo un grande fan dell'NP-HARD rock, per rimediare ha organizzato un concerto che porterà in città un enorme numero di persone che consumeranno una gran quantità di pizza.

Il grande evento

I PUNTI RISTORO (I)

Il sindaco ha previsto N punti ristoro sparsi per la città, che distribuiranno la pizza agli spettatori del concerto.
Ogni punto ristoro è raggiungibile da ogni altro punto ristoro tramite una strada di una certa lunghezza percorribile in entrambe le direzioni.

I PUNTI RISTORO (I)



Clienti poco pazienti (I)

- Ogni punto ristoro i dovrà soddisfare C_i clienti.
- Al tempo $t = 0$, presso il punto di ristoro i, i rispettivi C_i clienti saranno già in fila in attesa della propria pizza.
- I clienti però sono molto esigenti. Infatti, ogni unità di tempo, un cliente andrà via spazientito dall'attesa. In altre parole, ogni punto ristoro perderà un cliente per ogni unità di tempo di attesa.

Clienti poco pazienti (II)

Figura: I clienti spazientiti si riverseranno in città commettendo atti di vandalismo urbano, urtando la sensibilità della popolazione trentina. Bisogna quindi minimizzare il numero di clienti spazientiti.

I RIDERS (I)

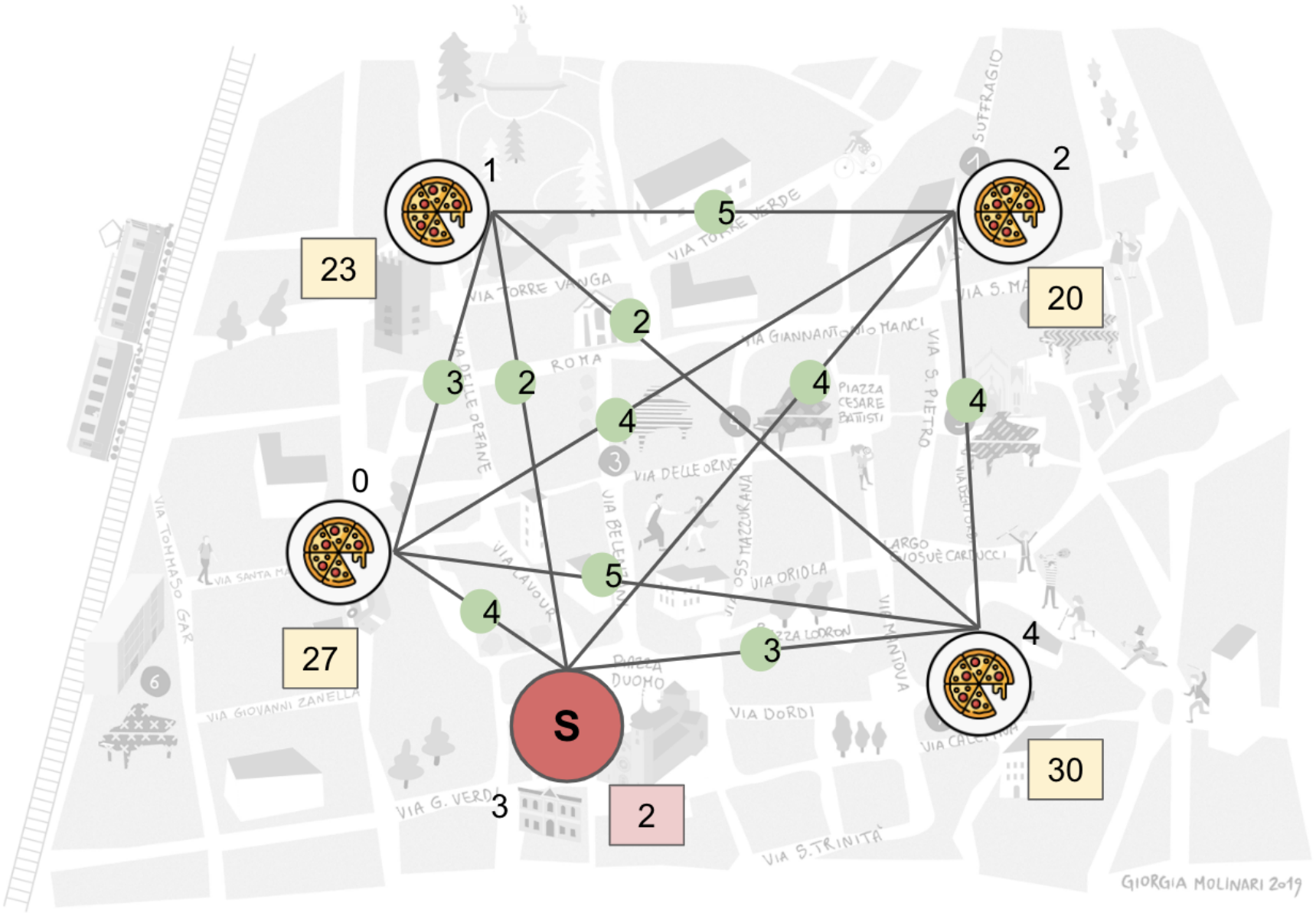
Il sindaco ha organizzato una squadra di riders per distribuire le pizze ai centri di ristoro:

- Tutti i riders sono posizionati inizialmente (ovvero al tempo $t = 0$) nella stessa base di partenza S.
- Tutti i riders viaggiano alla stessa velocità, ovvero una unità di lunghezza per ogni unità di tempo.
- I riders sono indipendenti, ovvero viaggiano in parallelo.

I RIDERS (II)

- Quando un rider arriva in un punto di ristoro, lascierà **sempre** tutte le pizze richieste dal punto ristoro. Questo è permesso dalle borse progettate dal famoso **FabLab**, che permettono di conservare e trasportare *infinite* pizze senza sforzo.
- Una conseguenza diretta di ciò è anche la possibilità di passare da un punto di ristoro all'altro senza dover tornare in base a ricaricare le pizze. Quindi, una volta scaricate le pizze ad un centro ristoro, il rider potrà spostarsi direttamente al prossimo centro ristoro.
- Una volta che un centro ristoro è stato soddisfatto, **non sarà più possibile** per nessun rider **tornare** in quel centro di ristoro. Tutti i clienti che erano rimasti andranno via felici, e nessun altro cliente arriverà. Inoltre, nessun rider può tornare alla base.
- Non è necessario visitare tutti i punti di ristoro.

Esempio: mappa



Esempio: mappa

Nella mappa precedente:

- Ogni ristoro è collegato ad ogni altro ristoro da una strada. La base è uno di questi punti ed è indicato da S in rosso (il 3 nell'esempio).
- Il rettangolo rosso vicino ad S riporta il numero R di riders.
- Ogni strada ha una certa lunghezza indicata dal valore riportato nel cerchio verde sovrapposto alla strada. Es: la strada che porta da $S = 3$ al ristoro 1 ha lunghezza 2.
- Una strada di lunghezza L è percorsa da un rider in L unità di tempo. I rider partono tutti da S.
- Al tempo $t = 0$, ogni punto ristoro ha un certo numero di clienti in attesa. Nella mappa questo è indicato dal valore nel rettangolo giallo vicino al punto ristoro. Es: al tempo $t = 0$, il punto ristoro 1 ha 23 clienti.

Il vostro compito

Il sindaco si sta preparando per il concertone mangiando una pizza e ripassando la playlist del cantante, e nel frattempo chiede a voi di sviluppare una strategia per la distribuzione delle pizze ai punti ristoro. Siccome vorrebbe evitare spiacevoli graffiti in giro per la città, vi chiede di trovare una strategia che massimizzi il valore

$$P = \sum_{r \in \{1,\ldots,R\}} P_r$$

dove P_r è il numero di clienti soddisfatti da un rider r.

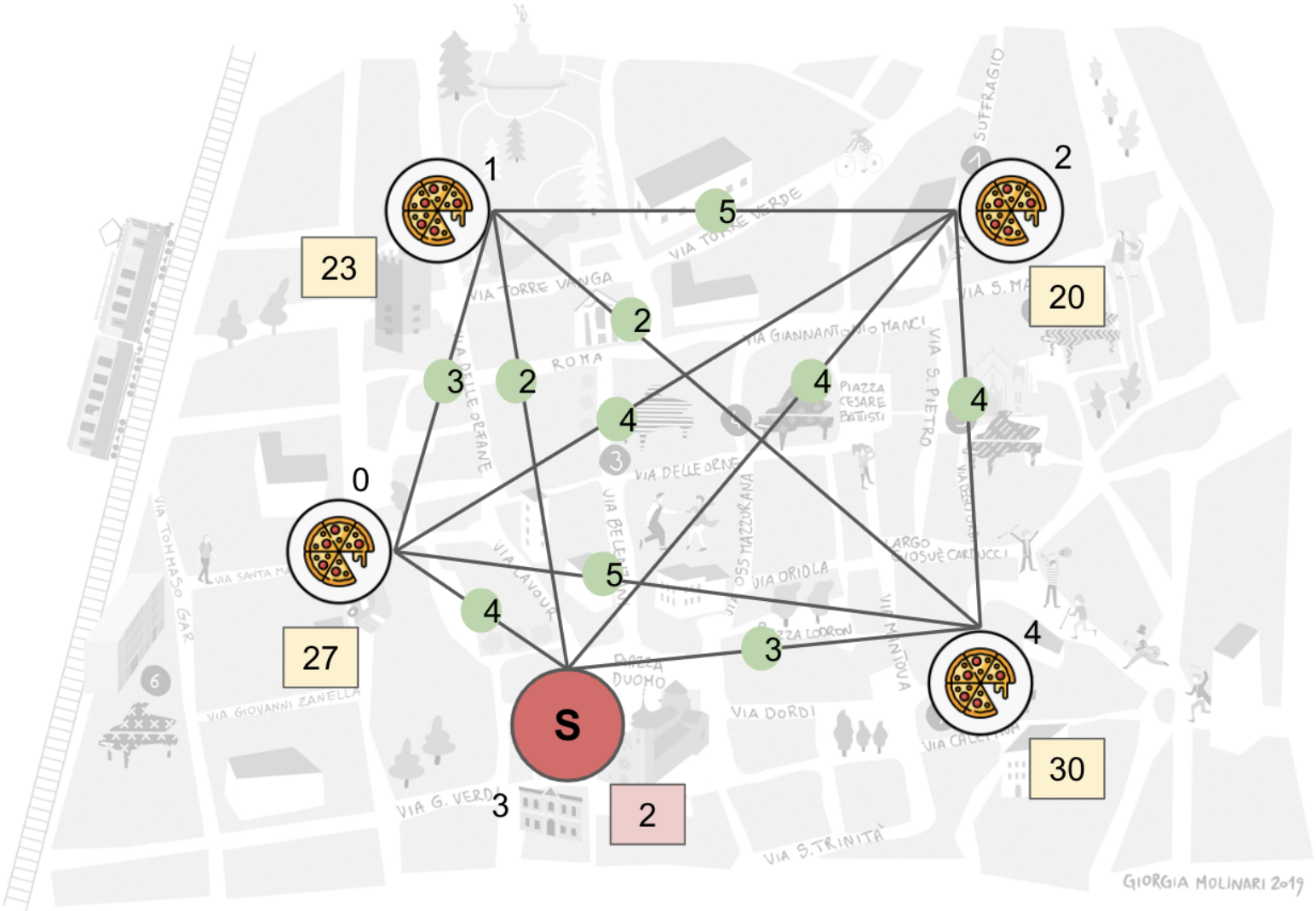
Il vostro compito

P_r non può essere negativo:

$$P_r = \sum_{n \in \pi_r} \max(0, C_n - t_n)$$

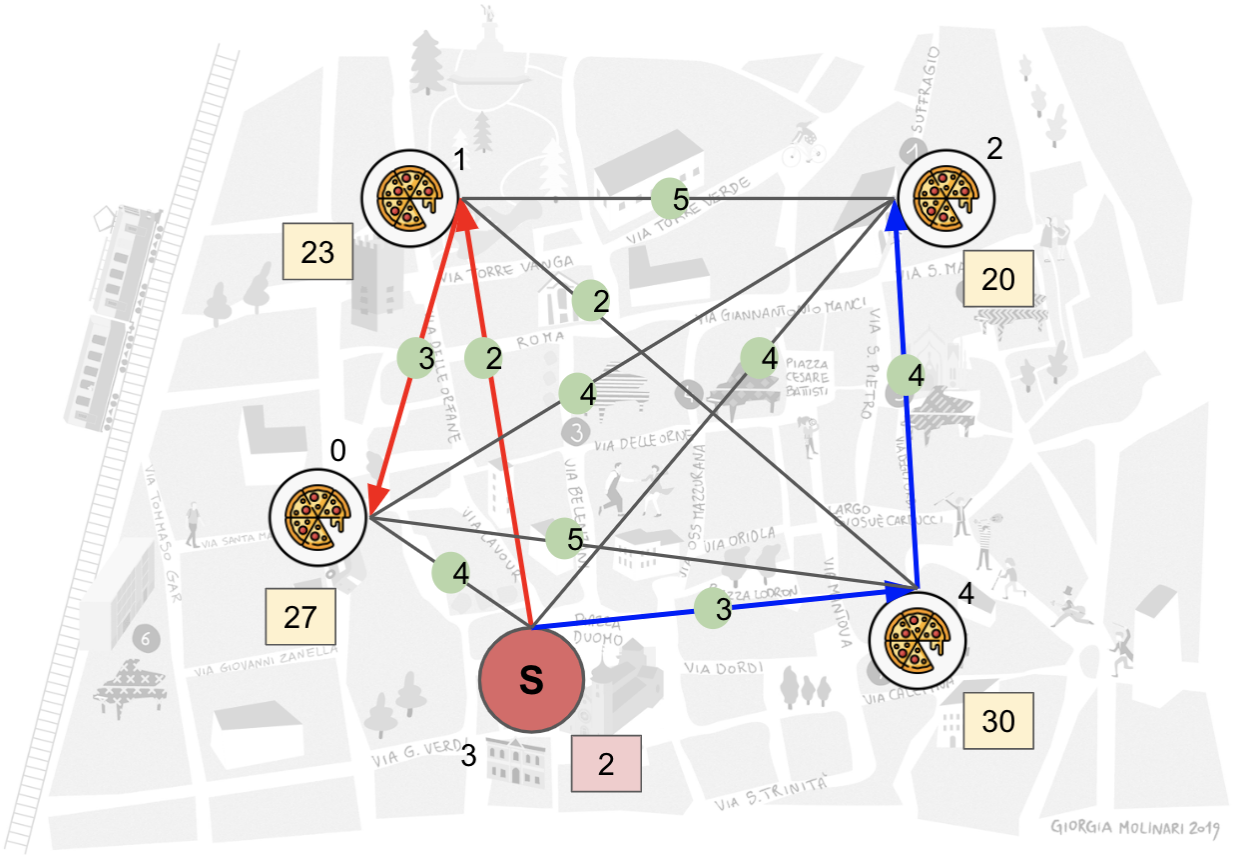
dove π_r è il cammino del rider r, C_n è il numero iniziale di clienti (al tempo $t = 0$) del ristoro n e t_n è il tempo a cui è stato visitato il ristoro n.

Esempio (I)



Ritorniamo al nostro esempio

Esempio (I)



Soluzione valida: ci sono due rider; il rider rosso parte da S e arriva al ristoro 1 al tempo $t = 2$ soddisfando 21 clienti e al ristoro 0 al tempo 5 soddisfando 22 clienti. Il rider blu parte da S e arriva al ristoro 4 e poi al ristoro 2 rispettivamente ai tempi 4 e 7 sfamando 27 e 13 clienti.

Esempio (I)

Il rider rosso fa il percorso 3-1-0

- Impiega 2 unità di tempo per andare dalla base nel nodo 3 al ristoro al nodo 1
- Al ristoro 1 sono rimasti $23 - 2 = 21$ clienti
- Il rider impiega altre 3 unità di tempo per andare da 1 a 0 (per un totale di 5 unità di tempo per il viaggio completo)
- Al ristoro 0 sono rimasti $27 - 5 = 22$ clienti
- Il rider rosso ha accontentato $22 + 21 = 43$ clienti

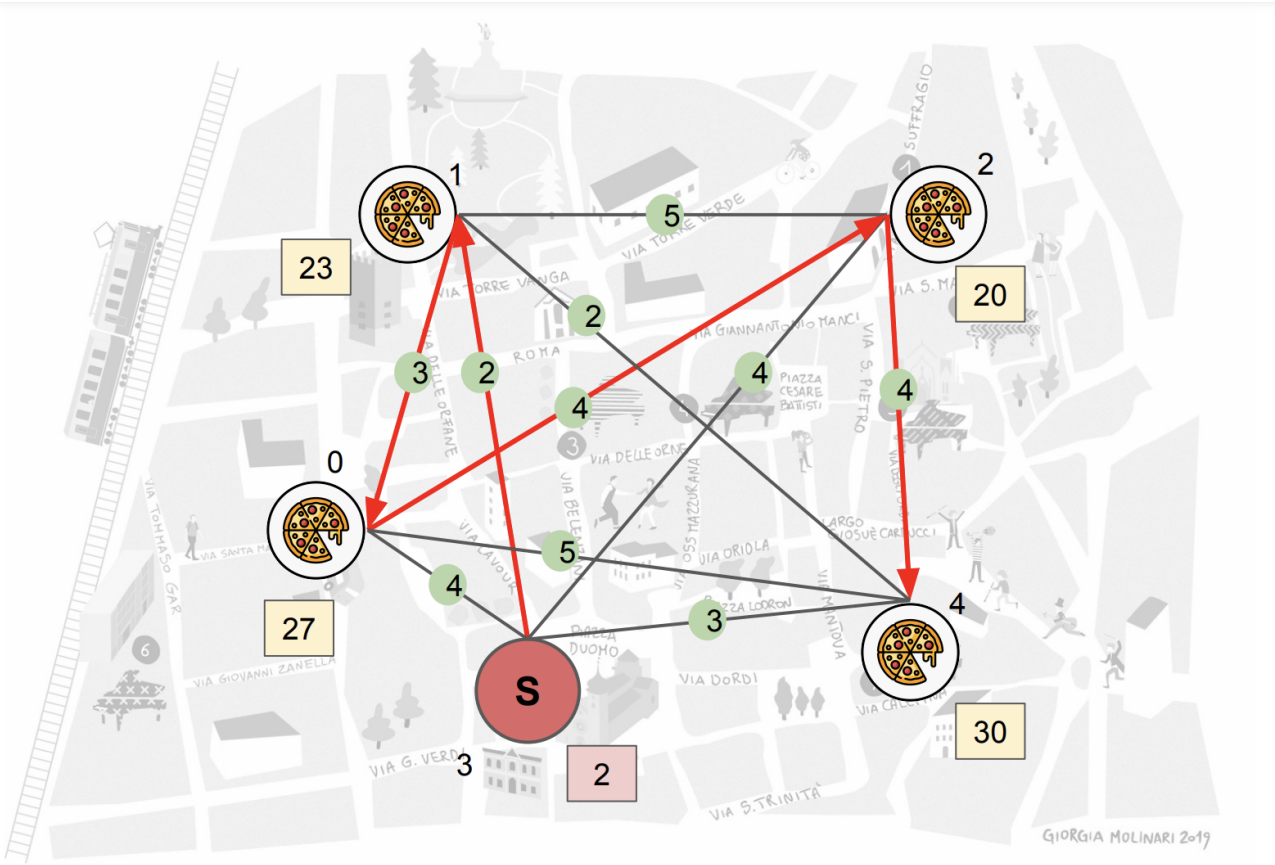
Esempio (I)

Il rider blu fa il percorso 3-4-2

- Impiega 3 unità di tempo per andare dalla base nel nodo 3 al ristoro al nodo 4
- Al ristoro 4 sono rimasti $30 - 3 = 27$ clienti
- Il rider impiega altre 4 unità di tempo per andare da 4 a 2 (per un totale di 7 unità di tempo per il viaggio completo)
- Al ristoro 2 sono rimasti $20 - 7 = 13$ clienti
- Il rider blu ha accontentato $27 + 13 = 40$ clienti

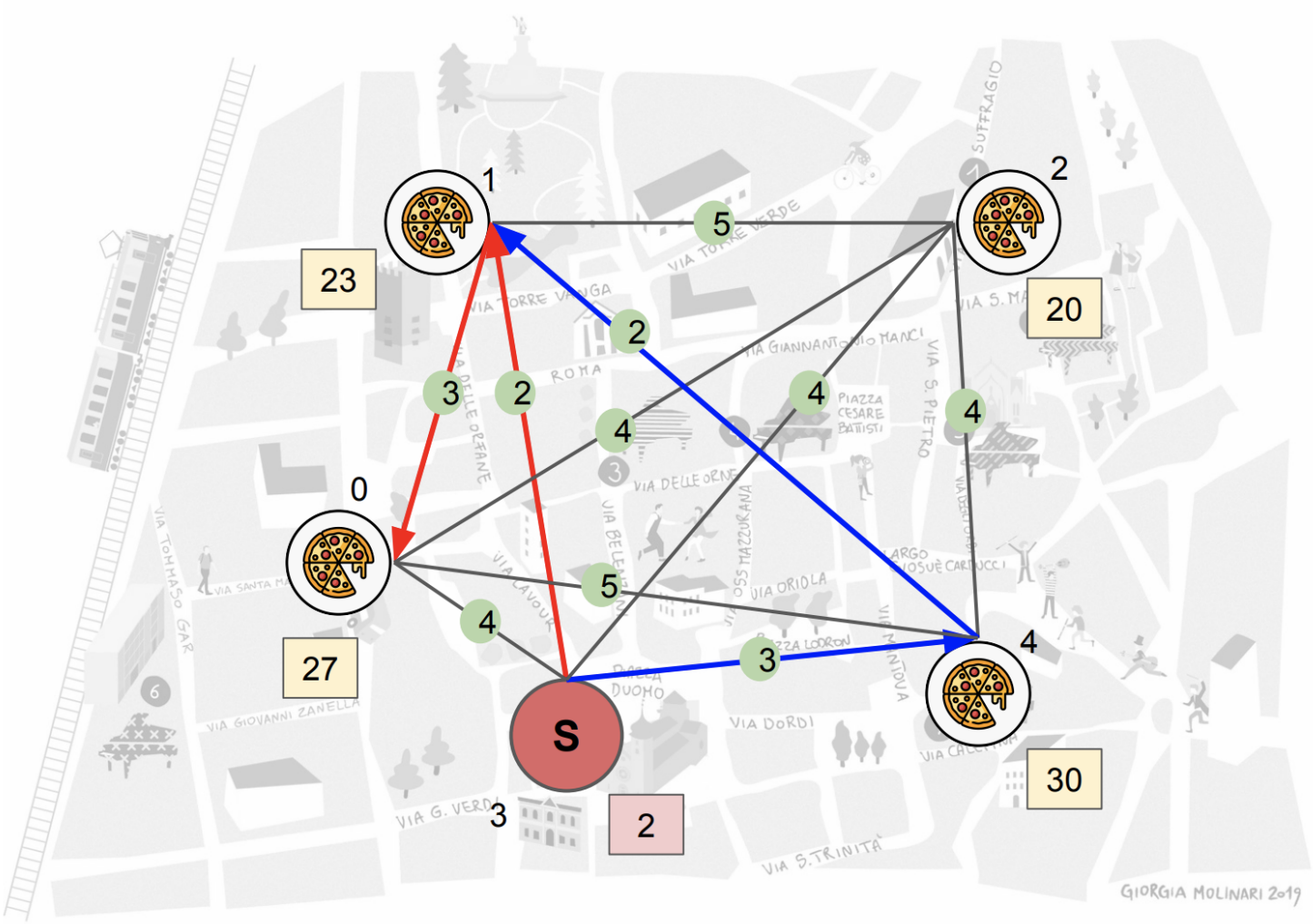
In totale sono stati accontentati 83 clienti.

Esempio (II)



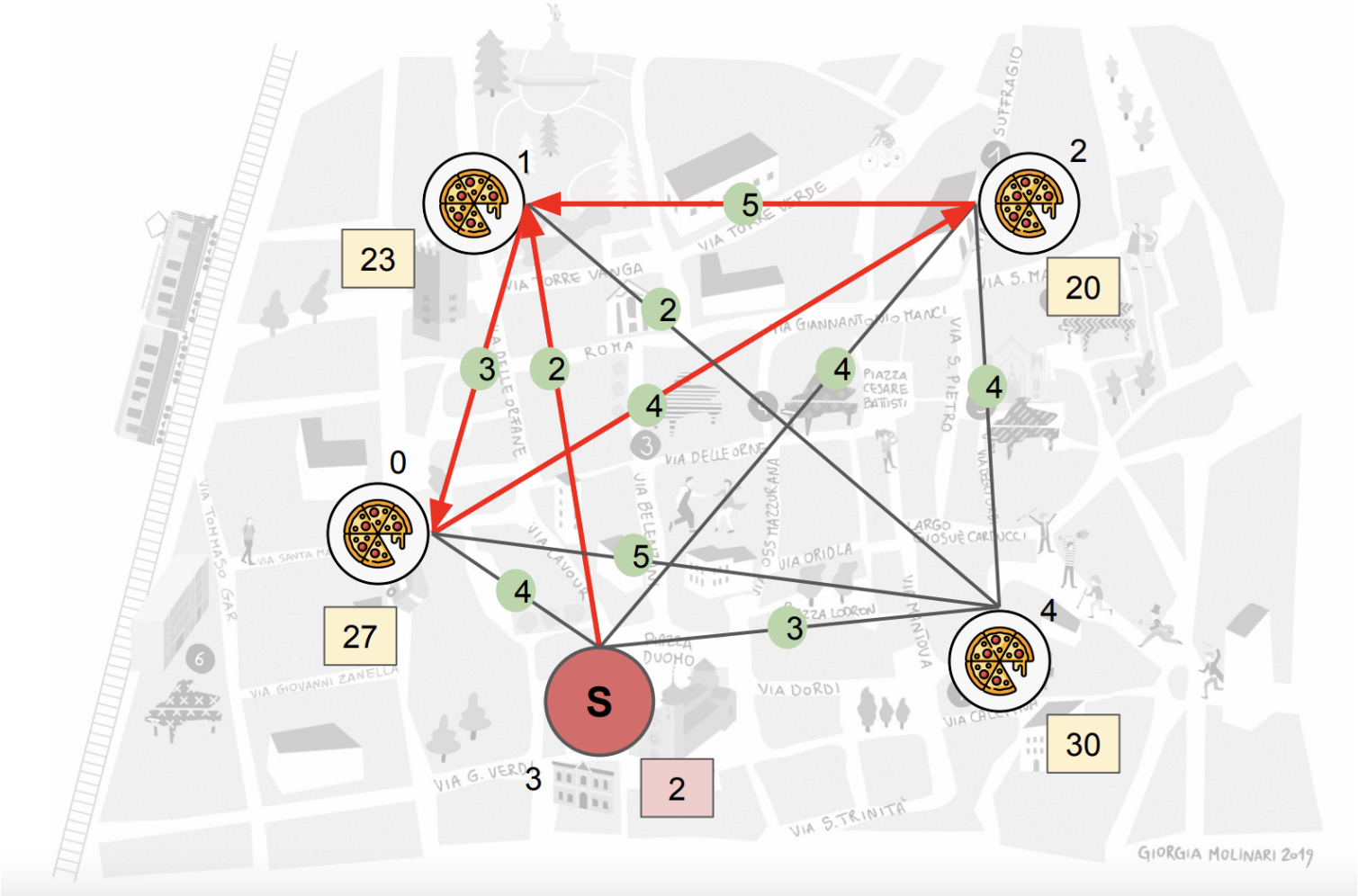
Soluzione valida: ci sono due rider, tuttavia ne usiamo solo uno. Riusciamo ad accontentare 71 clienti.

Esempio di soluzione errata I



Questa soluzione non è valida perché il rider rosso passa da 1 al tempo $t = 2$ e il rider blu ci ripassa al tempo $t = 5$.

Esempio di soluzione errata II



Questa soluzione non è valida perché il rider rosso passa da 1 al tempo $t = 2$ e ci ripassa al tempo $t = 14$.

Input

Un file con $2 + N - 1$ righe.

- La prima riga riporta 3 numeri interi: N (`int`), R (`int`) e S (`int`), rispettivamente il numero di punti di ristoro, il numero di rider e la base.
- La seconda riga riporta N numeri. L' i-esimo numero C_i rappresenta il numero di persone (`int`) inizialmente in attesa nel punto di ristoro i. La base S ha sempre zero clienti;
- Le successive $N - 1$ righe descrivono le distanze tra i punti di ristoro. La prima riga contiene il peso (`int`) dell'arco da 1 a 0; la seconda riga contiene i pesi degli archi da 2 a 0 e 1, e così via.

```
w(1, 0)
w(2, 0) w(2, 1)
...
w(N-1, 0) w(N-1, N-2)
```

Output

Un file con le vostre proposte di soluzione sulla mappa, ogni proposta è composta da $1 + 2 \cdot R + 1$ righe:

- La prima riga contiene un numero: P (`int`), ovvero il numero totale di pizze che siete riusciti a consegnare;
- Le successive $2 \cdot R$ righe contengono le informazioni sul percorso di ciascun rider da 1 a R. Per ciascun rider r, nella prima riga si indica il numero di ristori visitati da quel rider $|\pi_r|$ e nella seconda si indica la sequenza dei punti di ristoro visitati. Si parte sempre dalla base e quindi ogni rider visita almeno un nodo (S);
- l'ultima riga contiene tre asterischi $***$;

Esempio I

Input:

```
5 2 3
27 23 20 0 30
3
4 5
4 2 4
5 2 4 3
```

Output:

```
83
3
3 1 0
3
3 4 2
***
```

Note su Input

Assunzioni generali

- $1 \leq N \leq 2.000$
- $0 \leq S < N$
- $1 \leq R \leq N$
- $1 \leq w_{\text{min}} \leq w_{\text{max}} \leq 1.000.000$
- $0 \leq C_i \leq 1.000.000$
- $P_S = 0$
- Ogni grafo è completo.
- Ogni grafo è non diretto.
- Il grafo non è geometrico, non necessariamente vale la disuguglianza triangolare.

Casi di test

- Ci sono 20 casi di test in totale.
- In almeno 4 casi su 20 c'è un solo rider.

Limiti di tempo e memoria

I limiti di tempo e memoria sono:

- Tempo limite massimo: 5 secondi.
- Memoria massima: 64 MB.

$\Rightarrow$ Limite di 40 sottoposizioni per gruppo.

$\Rightarrow$ Potete provare con un dataset equivalente sulla vostra macchina

$\Rightarrow$ **Nota:** il dataset di esempio non riporta gli output.

Output

Potete stampare le soluzioni in maniera incrementale:

- Importate `riders.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 "riders.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 `riders.cpp`):

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

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

I file `riders.cpp`, `grader.cpp` e `riders.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

Ogni caso di test vale 5 punti. Il punteggio massimo è di 100 punti.

I parametri di valutazione sono i seguenti:

- `P`: il numero di persone servite dato dalla vostra soluzione;
- `maxP`: upper bound del numero di persone servibili;
- `minP`: lower bound del numero di persone servibili dato da una soluzione che fa scelte a caso;

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

$$\mathcal{P} = \max\left(0, \frac{\texttt{P} - \texttt{minP}}{\texttt{maxP} - \texttt{minP}}\right) \cdot 5$$

Bounds

Lower bound

`minP` è il numero di persone soddisfatte risultante dalla media di 10 risultati di una soluzione che sceglie strade in modo casuale. Non è un vero e proprio lower bound (si può fare peggio, ma in quel caso prendete zero punti), ma una soglia per valutare la vostra soluzione.

Upper bound

`maxP` è un upper bound calcolato sommando il punteggio massimo che può essere dato da ogni punto di ristoro, ovvero il numero di persone inizialmente in attesa al punto *i* - la lunghezza del cammino minimo da S ad *i*. Non è sempre possibile raggiungere questo upper bound, quindi è impossibile fare 100 punti.

Punteggio

⇒ La **sufficienza è posta a 30 punti**.

✗ se una qualsiasi informazione data in output non è corretta (ad esempio la lunghezza del percorso di un rider non è quella dichiarata), si ottengono 0 punti.

✗ se una vostra soluzione dovesse dare un punteggio inferiore al lower bound, verranno assegnati 0 punti per quel caso di test.

Punti bonus per l'esame

L'assegnazione punti avviene in maniera competitiva:

- **3 punti** ai gruppi nel primo terzile della classica (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 30).

$\Rightarrow$ Classifica:
https://judge.science.unitn.it/arena/ranking/

Consegna

Consegna: martedì 31 maggio 2022 ore 12:00

Per caricare il vostro codice, recatevi su
`https://judge.science.unitn.it/arena/`

Suggerimenti

Cominciate subito a lavorare al progetto per presentarvi al prossimo ricevimento (martedì 24 maggio) con tutte le domande che vorrete fare.
In ogni caso, sappiate che:

- potete venire a ricevimento
- risponderemo alle vostre mail

Do's

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

Dont's

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)

Ricevimento

Date e orari

- martedì 24 maggio 2022 dalle 13:30 alle 15:30 (lab);
- mercoledì 25 maggio 2022 dalle 13:30 alle 14:30;
- giovedì 26 maggio 2022 dalle 11:30 alle 13:30 (lab);
- venerdì 27 maggio 2022 dalle 15:30 alle 16:30;

$\Rightarrow$ I ricevimenti si svolgeranno in aula (in orario di laboratorio) e/o su Discord (in orario non laboratorio).

$\Rightarrow$ Per qualsiasi domanda mandateci una mail a: `asd.disi@unitn.it`.

Buon lavoro!

Voglio una soluzione approssimata
Di quelle soluzioni fatte così
Voglio una soluzione che se ne frega
Che se ne frega di tutto sì
Voglio una soluzione che non è formale
Di quelle che non dimostri mai
Voglio una soluzione
di quelle che non si sa mai...

*E poi implementeremo pure A**
oppure proveremo con branch-and-bound
O forse non convergeremo mai
Ognuno a rincorrere i suoi bound
Ognuno col suo path, ognuno diverso
E ognuno in fondo greedy
Dentro i nodi suoi, suoi