

Algoritmi e Strutture Dati

Riders per il concertone (**riders**)

Testo del problema

Slides originali su: judge.science.unitn.it/slides/asd21/prog2.pdf

Premessa. Dopo aver migliorato di gran lunga la viabilità della ridente cittadina¹, il sindaco di Trento e il suo partito «Algoritmi, Cricche e Libertà», hanno conquistato ancora più consensi tra la popolazione. Come è ben noto, il sindaco ama organizzare competizioni algoritmiche e 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, ma non sempre i dati del passato sono indicativi per il futuro e 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 mon sa come smaltire le pizze in eccesso.

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 sindaco ha previsto N punti ristoro - o "chioschi" - 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 e percorribile in entrambe le direzioni. La figura 1 mostra una mappa della città con i vari punti ristoro collegati tra loro.

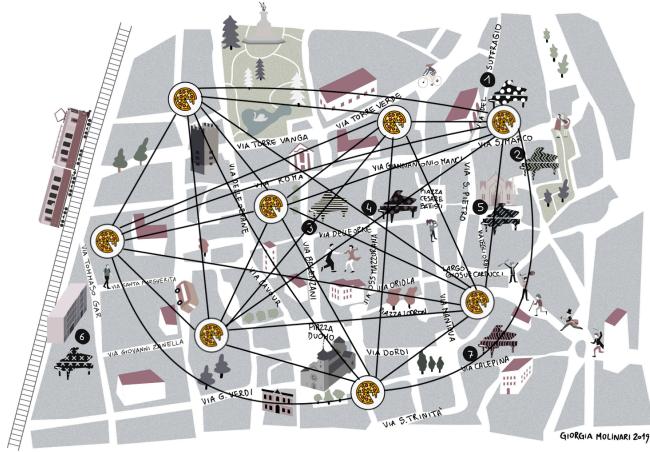


Figure 1: Una mappa della città con 7 punti di interesse.

Clienti poco pazienti. I punti ristoro hanno le seguenti caratteristiche: ogni punto ristoro i dovrà soddisfare C_i clienti e al tempo $t = 0$, presso il punto di ristoro i -esimo, i rispettivi C_i clienti saranno già in fila in attesa della propria pizza. I clienti però sono molto esigenti: infatti, **per 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.

¹Progetto "Cicli Ciclabili" a.a. 2019/2020.

I clienti spazientiti si riverseranno in città commettendo atti di vandalismo urbano, urtando la sensibilità della popolazione trentina.

I riders. Il sindaco ha organizzato una squadra di riders per distribuire le pizze ai vari punti ristoro:

- tutti i riders sono posizionati inizialmente (ovvero al tempo $t = 0$) nella stessa base di partenza S , che non coincide con un punto di ristoro, ovvero non ci sono clienti in attesa alla base;
- 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;
- quando un rider arriva in un punto 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.

Obiettivo

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:

$$P = \sum_{r \in \{1, \dots, R\}} P_r \quad (1)$$

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

P_r non può essere negativo:

$$P_r = \sum_{n \in \pi_r} \max(0, C_n - t_n) \quad (2)$$

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 .

Esempi

Per illustrare un esempio facciamo riferimento alla situazione descritta in Figura 2:

- Ogni punto 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.
- In questo esempio ci sono $R = 2$ riders. Ogni strada ha una certa lunghezza indicata dal valore riportato nel cerchio verde sovrapposto alla strada. Facendo riferimento alla Figura 2, 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. Per esempio, al tempo $t = 0$, il punto ristoro 1 ha $C_1 = 23$ clienti.

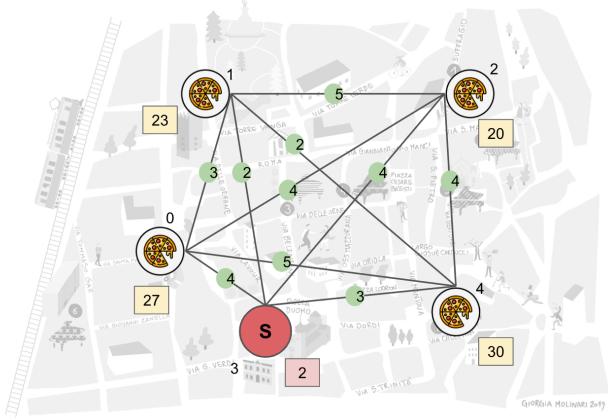


Figure 2: Una mappa della città con 5 punti di interesse. La base S si trova nel chiosco 3. Nei punti di ristoro $\{0, 1, 2, 4\}$ ci sono rispettivamente $C_0 = 27$, $C_1 = 23$, $C_2 = 20$ e $C_4 = 30$.

Esempi di soluzioni valide

Le Figure 3 e 4 mostrano due soluzioni valide per il caso di $R = 2$ riders che partono dalla base S in 3.

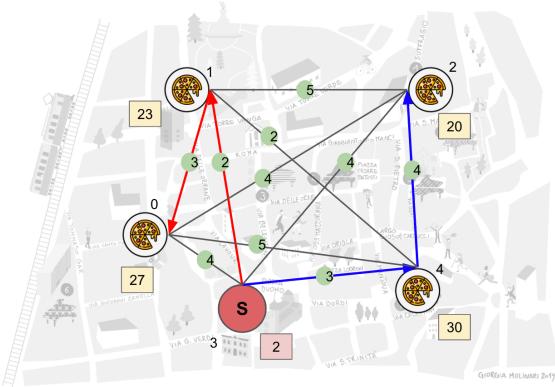


Figure 3: Soluzione valida per il caso con $R = 2$ due riders. Il rider rosso parte da S e arriva al ristoro 1 al tempo $t = 2$ soddisfando $C_1 - 2 = 21$ clienti, prosegue poi verso il ristoro 0 dove giunge al tempo $t = 5$ soddisfando $C_0 - 5 = 22$ clienti. Il rider blu parte da S e arriva al ristoro 4 e poi al ristoro 2 rispettivamente ai tempi 4 e 7 sfamando 26 e 13 clienti. In questo modo vengono serviti in $P_1 = 21 + 22 = 43$ e $P_2 = 27 + 13 = 40$. Quindi in totale vengono serviti $P = P_1 + P_2 = 43 + 40 = 83$ clienti.

Si noti che la soluzione presentata in Figura 3 ottierrà un punteggio maggiore perché soddisfa un maggior numero di clienti.

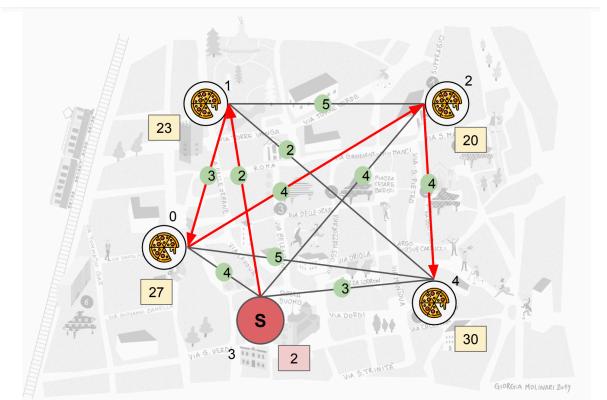


Figure 4: Una soluzione valida alternativa per il caso con $R = 2$ due riders. In questo caso usiamo solo un rider che segue il percorso $\pi_1 = [3, 1, 0, 2, 4]$ che vengono raggiunti rispettivamente ai tempi $[0, 2, 5, 9, 13]$ e accontentando $P_1 = (23 - 2) + (27 - 5) + (20 - 9) + (30 - 13) = 71$ clienti

Esempi di soluzioni non valide

Le Figure 5 e 6 mostrano dei casi di soluzioni non valide: una volta che un rider è passato da un punto ristoro nessun rider può visitarlo di nuovo.

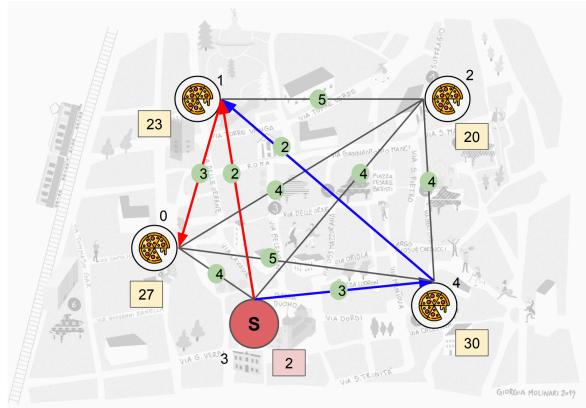


Figure 5: Questa soluzione non è valida perché entrambi i rider rosso e blu passano dal chiosco 1.

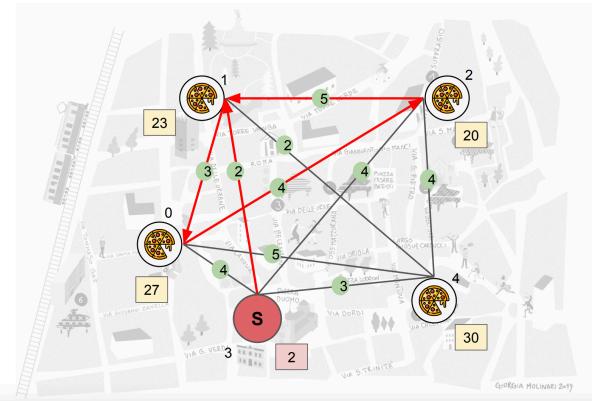


Figure 6: Questa soluzione non è valida perché il rider rosso passa due volte dal chiosco 1.

Input/Output

Input: un file con una mappa dei vari punti ristoro prensenti in città e dei loro collegamenti. In particolare, il file di input è costituito da $2 + N - 1$ righe.

- La prima riga riporta 3 numeri interi: N (int), R (int) e S (int), rispettivamente il numero di punti ristoro, il numero di rider e la base.
- La seconda riga riporta N numeri. L' i -esimo numero C_i (int) rappresenta il numero di persone (int) inizialmente in attesa nel chiosco i . La base S ha sempre zero clienti;
- Le successive $N - 1$ righe descrivono le distanze tra i chioschi, tutte int. 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.

Specifiche dell'input:

```

N R S
C_0 C_1 ... C_{N-1}
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, ogni proposta è composta da $1 + 2 \cdot R + 1$ righe:

- o_1 : La prima riga contiene un numero: P ('int'), ovvero il numero totale di pizze che siete riusciti a consegnare;
- o_2 : 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 $|\pi_r|$ ('int') e nella seconda si indica la sequenza delle città visitate, tutte 'int'. Si parte sempre dalla base e quindi ogni rider visita almeno un nodo (S);

*o*₃ : l'ultima riga contiene tre asterischi ***;

Specifica dell'output:

```
P
|\pi_1|
p_{1,0} p_{1,1} ... p_{1,|\pi_1|-1}
|\pi_2|
p_{2,0} p_{2,1} ... p_{2,|\pi_2|-1}
...
|\pi_R|
p_{R,0} p_{R,1} ... p_{R,|\pi_R|-1}
***
```

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!**
- Il punteggio viene calcolato usando l'**ultima soluzione** terminata con tre asterischi ***.

Per soluzioni valide, 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 entro i limiti di tempo e memoria otterrete il seguente punteggio \mathcal{P} :

$$\mathcal{P} = \max \left(0, \frac{P - \text{minP}}{\text{maxP} - \text{minP}} \right) \cdot 5 \quad (3)$$

nota: non è necessario che calcoliate il punteggio della vostra soluzione - vi servirebbero i parametri dei bound *minP* e *maxP* che non avete. Il vostro obiettivo è in ogni caso di **massimizzare il numero di persone servite P**.

Esempi (punteggio)

Nell'esempio di cui sopra, la risposta del sistema del valutazione è:

Soluzione valida: P: 83, minP: 62, maxP: 87

Il punteggio è calcolato nel modo seguente:

$$P = \frac{83 - 62}{87 - 62} \cdot 5 = 0,84 \cdot 5 = 4,20$$

.1 Spiegazione del punteggio

Per ogni testcase, viene assegnato un punteggio da 0 a 1 *a* e poi moltiplicato per 5, secondo la formula di cui all'Equazione~3. Il messaggio resituito da arena indica i valori delle variabili usate nella formula. Questi valori vi permettono di valutare i punti di forza e i punti deboli della vostra soluzione.

Per esempio:

Soluzione valida: P: 83, minP: 62, maxP: 87

Le variabili stampate nel messaggio sono le seguenti:

- *P*: è il numero di persone soddisfatte prodotto dalla vostra soluzione.
- *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.
- *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.

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 30 punti;
- C'è un limite di 40 sottoposizioni per gruppo;

Limiti e assunzioni

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

Casi di test

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

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 `riders.cpp`. I file `riders.cpp`, `grader.cpp` e `riders.h` devo stare nella stessa cartella.

Sistemi GNU/Linux

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

Se ottene 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 Code³ o Visual Studio 2017⁴ 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 Studio⁶.

Una volta installato Cygwin è possibile simulare quanto avviane su arena compilando il proprio sorgente senza includere l'header `riders.h` e il grader `grader.cpp`:

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

e lanciare il comando come:

```
timeout.exe 5 ./riders
```

`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
5 2 3 27 23 20 0 30 3 4 5 4 2 4 5 2 4 3	82 4 3 1 4 2 2 3 0 *** 83 3 3 1 0 3 3 4 2 ***
10 2 6 48 49 41 30 46 46 0 45 48 34 4 6 7 5 3 5 7 3 5 4 5 3 7 4 5 7 6 5 5 5 3 3 6 3 6 6 6 5 4 5 3 6 5 6 5 4 6 7 4 3 6 7 6 6 5	300 6 6 5 1 3 9 4 5 6 2 7 0 8 *** 302 6 6 5 1 4 3 9 5 6 8 2 7 0 ***
6 1 5 1196 1005 1182 1141 1115 0 138 181 120 167 192 102 175 164 177 118 162 119 104 164 162	3891 6 5 2 3 4 1 0 ***