Progetto ASP – Automated Reasoning – A.A. 2023/2024

Katiuscia Pellone, mat. 288642

Samuele Ramassone, mat. 291365

Introduzione

Il problema presentato rappresenta una versione alternativa dei classici problemi di visita di un grafo.
La città viene rappresentata attraverso un grafo, dove i nodi sono luoghi e gli archi le strade.
Ogni strada, caratterizzata dalla lunghezza e dalla difficoltà, può essere attraversata in salita o in discesa, e questo cambia la distanza percepita dal ciclista.
Le strade sono di tre tipi:

1. Soft (in piano): la distanza in salita e in discesa è uguale (in termini di distanza percepita);
2. Medium (saliscendi): ogni km di tipo medium pesa come 2km in piano, la distanza è la stessa in salita ed in discesa;
3. Hard (salita dura): ogni km di tipo hard pesa come 5km in piano, la distanza in discesa è zero (il ciclista non pedala).

In questo caso il visitatore del grafo è un ciclista che fondamentalmente ha due obiettivi:

1. Percorrere tutte le strade della propria zona (rappresentata dal grafo);
2. percorrere le salite "hard" in salita almeno una volta.

L'unico vincolo è che al massimo è in grado di percorrere un certo numero di km al giorno, che si riducono a seconda dei tratti in salita, e di strade quotidiane.
Il ciclista non deve partire necessariamente ogni giorno dallo stesso nodo ma ogni giorno deve ritornare esattamente da dove è partito, i.e., compiere un ciclo chiuso.
Nella soluzione finale si vanno a minimizzare il numero di giorni necessari.

Descrizione codice

La codifica del problema, con i relativi commenti, è contenuta nel file *problema.txt*.
Nella prima parte del file sono presenti i fatti che rappresentano l'istanza presa in considerazione. L'istanza è formata da un elenco di nodi (luoghi del paese, incroci, ...) e di archi (strade) caratterizzati da una lunghezza, espressa in km, e da un livello di difficoltà. Si assume che tutte le strade siano a doppio senso, per cui sono state aggiunte delle regole che rendono il grafo bidirezionale e per adattare le lunghezze in base alla tipologia di strada.

```
% Il numero massimo di km è stato espresso attraverso l'uso di una costante.
#const max_distance_per_day = 20.

% Fatti – Grafo espresso come nodi e archi
% Qui è riportata una parte di una delle istanze solo a scopo esemplificativo
nodo(a;b;c;d;e).
lunghezza(a, b, 2, soft).
lunghezza(c, a, 1, medium).
lunghezza(b, d, 1, hard).
```

% Gli archi soft e medium percorsi in discesa non cambiano la propria lunghezza "percepita"
arco(X, Y, C, soft) :- lunghezza(X, Y, C, soft).
arco(Y, X, C, soft) :- lunghezza(X, Y, C, soft).

*arco(X, Y, C*2, medium) :- lunghezza(X, Y, C, medium).*
*arco(Y, X, C*2, medium) :- lunghezza(X, Y, C, medium).*

%Gli archi hard percorsi in discesa hanno una distanza "percepita" pari a zero
*arco(X, Y, C*5, hard) :- lunghezza(X, Y, C, hard).*
arco(Y, X, 0, slope) :- lunghezza(X, Y, C, hard).

Segue la definizione delle regole e dei vincoli necessari al problema. Per illustrare il cammino percorso dal ciclista viene utilizzato *visited(X, Y, I, D)* dove i primi due parametri sono gli estremi dell'arco (X, Y), visitato durante lo step I nel giorno D.

1 { start(X, D) : nodo(X) } 1 :- giorno(D).
Bisogna garantire che ogni giorno il ciclista parta da un solo nodo.

1 { visited(X, Y, 0, 1) : arco(X, Y, _, _) } 1 :- start(X, 1).
Questa regola inizializza il percorso del ciclista al giorno 1. Partendo dal nodo iniziale X, si visita solo un arco uscente da esso.

0 { visited(X, Y, I, D) : arco(X, Y, _, _) } 1 :- step(I), step(I-1), giorno(D), visited(_, X, I-1, D).
Con questa regola vengono ricorsivamente scelti gli archi da visitare costruendo un cammino. Se allo step precedente il ciclista è arrivato al nodo X, allo step successivo percorrerà una strada che parte da quel nodo.

Le regole che seguono servono a calcolare la distanza percorsa dal ciclista ogni giorno. Si utilizza un predicato *cumulative_distance(I, D, Total)* per indicare i km percorsi nel giorno D fino allo step I.

cumulative_distance(0, 1, C) :- visited(X, Y, 0, 1), arco(X, Y, C, _).
La regola è definita in modo ricorsivo a partire dal caso base (step 0, giorno 1). La distanza percorsa è pari alla lunghezza dell'arco attraversato. Si assume che le lunghezze dei singoli archi non siano superiori alla massima distanza percorribile dal ciclista, altrimenti il problema risulterebbe insoddisfacibile.

```
cumulative_distance(I, D, Total + C) :- visited(X, Y, I, D),  visited(_, _, I-1, D),
                                        cumulative_distance(I-1, D, Total),
                                        arco(X, Y, C, _).
```
Con questa regola viene incrementata la distanza percorsa fino allo step I-1 con la lunghezza della strada visitata all'i-esimo step.

cumulative_distance(I+1, D+1, C) :- last_step(I, D), visited(X, Y, I+1, D+1),arco(X, Y, C, _).
Questa regola inizializza la distanza del giorno successivo ponendola uguale alla lunghezza del primo arco visitato. Con *last_step(I, D)* si indica l'identificatore dell'ultimo step eseguito al

giorno D, e con *visited(X, Y, I+1, D+1)* si prende l'arco (X, Y) visitato al primo step del giorno successivo.

Per implementare il passaggio al giorno successivo, è stata utilizzata una regola che trova gli archi "validi" da visitare nei giorni che seguono, ossia quegli archi che non possono essere visitati nella giornata corrente.

```
valid_arc(X, Y, I, D+1, C) :- step(I), cumulative_distance(I-1, D, Total), arco(X, Y, C, _),
                              Total + C > max_distance_per_day, start(Node, D),
                              visited(_, Node, I-1, D).
```

Con questa regola è rappresentata la situazione in cui il ciclista deve fare l'i-esimo step, l'arco (X, Y) non è visitabile (senza superare la distanza massima) e allo step i-1 è tornato al nodo da cui è partito. Tale arco diventa valido per essere visitato il giorno successivo.

```
0 { visited(X, Y, I, D+1) : valid_arc(X, Y, I, D+1, C) } 1.
```

Questa regola garantisce che venga scelto al più un solo arco, tra quelli validi, per inizializzare il percorso del giorno D+1.

```
giorno(D) :- visited(_, _, _, D).
```

Questa regola definisce semplicemente un nuovo giorno quando viene visitato un arco in quel giorno.

Uno dei vincoli del ciclista è quello di percorrere tutte le strade, e di percorrere quelle hard in salita almeno una volta. Avendo le strade a doppio senso, una strada è considerata "percorsa" se è stata visitata in almeno uno dei due versi. È stato utilizzato *check(X, Y)* per effettuare questa verifica.

```
check(X, Y) :- visited(X, Y, _, _).
check(X, Y) :- visited(Y, X, _, _).

% vincolo per garantire che tutti gli archi siano visitati in uno dei due lati
:- arco(X, Y, _, _), not check(X, Y).
% vincolo per garantire che le strade hard siano percorse in salita
:- arco(X, Y, _, hard), not visited(X, Y, _, _).

% Identifica l'ultimo step del ciclista
last_step(I) :- step(I), not visited(_, _, I+1, _), visited(_, _, I, _).

% Identifica l'ultimo step in un giorno D
last_step(I, D) :- step(I), not visited(_, _, I+1, D), visited(_, _, I, D).

:- last_step(I, D), start(Y, D+1), not visited(Y, _, I+1, D+1).
```

Questo vincolo garantisce che il nodo di partenza del giorno successivo coincida con il nodo previsto in *start(Y, D+1)*.

```
:- last_step(I, D), start(X, D), not visited(_, X, I, D).
```

Questo vincolo garantisce che il ciclista all'ultimo step della giornata torni al nodo da cui è partito.

:- step(I), visited(_, _, I, D), visited(_, _, I, D1), D != D1.
Con questo vincolo si specifica che non è possibile avere più di un arco visitato nello stesso step in giorni differenti.

:- giorno(D), cumulative_distance(_, D, T), T > max_distance_per_day.
Con questo vincolo si assicura che la distanza totale percorsa in giornata non superi il massimo consentito.

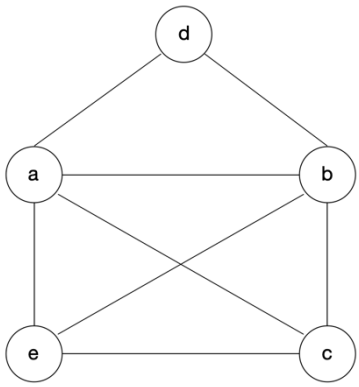
L'obiettivo è di minimizzare il numero di giorni necessari per percorrere tutte le strade.
#minimize { 1, D : visited(_, _, _, D) }.

Risultati

I test sono stati eseguiti su Clingo 5.7.0, versione Online.

Esempio 1 – Distanza massima giornaliera: 20km

Il grafo di questo esempio è illustrato sotto. I dettagli sulla tipologia di arco e sulla lunghezza (originale, non percepita dal ciclista) sono elencati vicino per una migliore leggibilità.



(a, b, 2, soft)
(b, c, 2, soft)
(c, a, 1, medium)
(a, d, 1, medium)
(b, d, 1, hard)
(b, e, 1, hard)
(a, e, 1, hard)
(c, e, 1, soft)

Il risolutore trova una soluzione ottima di due giorni, che è coerente dal momento che sono presenti quattro strade hard la cui lunghezza percepita, per ognuna, è pari a 5. Solo per visitare queste sono necessari 20 km, quindi un giorno intero. Con un altro giorno è possibile visitare il resto del paesino.

```
clingo version 5.7.0
Reading from stdin
Solving...
Answer: 1
start(a,1) visited(a,e,0,1) cumulative_distance(0,1,5) visited(e,b,1,1) cumulative_distance(1,1,5) vi
Optimization: 2
OPTIMUM FOUND

Models       : 1
  Optimum    : yes
Optimization : 2
Calls        : 1
Time         : 0.149s (Solving: 0.01s 1st Model: 0.01s Unsat: 0.00s)
CPU Time     : 0.000s
```

Nello screen non è possibile vedere interamente l'output, ma il risultato ottenuto è quello che segue. Per ogni giorno sono stati evidenziati il primo e l'ultimo arco visitato, dove è possibile notare come i nodi iniziali e finali coincidano. Inoltre, è stata evidenziata anche la distanza a fine giornata, e si può osservare che è inferiore al massimo consentito.

start(b,1)
visited(b,e,0,1) cumulative_distance(0,1,5)
visited(e,c,1,1) cumulative_distance(1,1,6)
visited(c,a,2,1) cumulative_distance(2,1,8)
visited(a,e,3,1) cumulative_distance(3,1,13)
visited(e,c,4,1) cumulative_distance(4,1,14)
visited(c,b,5,1) cumulative_distance(5,1,16)
last_step(5,1)
start(b,2)
visited(b,d,6,2) cumulative_distance(6,2,5)
visited(d,a,7,2) cumulative_distance(7,2,7)
visited(a,b,8,2) cumulative_distance(8,2,9)
visited(b,e,9,2) cumulative_distance(9,2,14)
visited(e,b,10,2) cumulative_distance(10,2,14)
last_step(10,2)

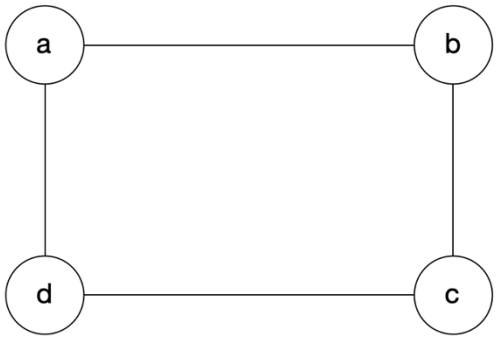
Esempio 1 – Distanza massima giornaliera: 30km
Per lo stesso grafo, aumentando la distanza è possibile ottenere una soluzione che richiede un giorno.

```
clingo version 5.7.0
Reading from stdin
Solving...
Answer: 1
start(e,1) visited(e,a,0,1) cumulative_distance(0,1,0) visited(a,b,1,1) cumulative_distance(1,1,2) vi
Optimization: 1
OPTIMUM FOUND

Models       : 1
  Optimum    : yes
Optimization : 1
Calls        : 1
Time         : 0.077s (Solving: 0.02s 1st Model: 0.02s Unsat: 0.00s)
CPU Time     : 0.000s
```

start(e,1)
visited(e,a,0,1) cumulative_distance(0,1,0)
visited(a,b,1,1) cumulative_distance(1,1,2)
visited(b,d,2,1) cumulative_distance(2,1,7)
visited(d,a,3,1) cumulative_distance(3,1,9)
visited(a,c,4,1) cumulative_distance(4,1,11)
visited(c,b,5,1) cumulative_distance(5,1,13)
visited(b,a,6,1) cumulative_distance(6,1,15)
visited(a,e,7,1) cumulative_distance(7,1,20)
visited(e,c,8,1) cumulative_distance(8,1,21)
visited(c,b,9,1) cumulative_distance(9,1,23)
visited(b,e,10,1) cumulative_distance(10,1,28)
last_step(10,1)

Esempio 2 – Distanza massima pari alla somma delle lunghezze percepite dal ciclista



(a, b, 4, soft)
(b, c, 3, medium)
(c, d, 2, hard)
(d, a, 1, soft)

max_km = 4 + 3*2 + 2*5 + 1 = 21

Il solver trova correttamente la soluzione corrispondente a un giorno.

```
clingo version 5.7.0
Reading from stdin
Solving...
Answer: 1
start(b,1) visited(b,c,0,1) cumulative_distance(0,1,6) visited(c,d,1,1) cumulative_distance(1,1,16) v
Optimization: 1
OPTIMUM FOUND

Models       : 1
  Optimum    : yes
Optimization : 1
Calls        : 1
Time         : 0.121s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.000s
```

start(b,1)
visited(b,c,0,1) cumulative_distance(0,1,6)
visited(c,d,1,1) cumulative_distance(1,1,16)
visited(d,a,2,1) cumulative_distance(2,1,17)
visited(a,b,3,1) cumulative_distance(3,1,21)
last_step(3,1)