ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ

Ακαδημαϊκό Έτος 2022-2023

ΤΕΧΝΙΚΗ ΑΝΑΦΟΡΑ

3η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ

1.Τουρίστες και φακός

Ζήκος Σπύρος
Α.Μ.: 1084581

Πάτρα 2022-23

Παραδοχές:

Ο χρήστης καθορίζει την αρχική κατάσταση ως εξής:
1) Δίνει τον συνολικό αριθμό τουριστών που προβλήματος.
2) Δίνει την πλευρά του ποταμού στην οποία βρίσκεται κάθε τουρίστας στην αρχή του προβλήματος καθώς το πρόβλημα αναφέρει ότι οι τουρίστες είναι μοιρασμένοι στις δύο όχθες ενός ποταμού.
3) Δίνει τον χρόνο που χρειάζεται κάθε τουρίστας για να περάσει τη γέφυρα.
4) Δίνει τη θέση του φακού.
5) Δίνει την κατάσταση-στόχο, δηλαδή τη μεριά στην οποία πρέπει να καταλήξουν όλοι οι τουρίστες.

Το πρόγραμμα δεν τερματίζει όταν επιτυγχάνεται η κατάσταση-στόχος, αλλά εκτυπώνει κατάλληλο μήνυμα.

Ο κανόνας insTour/0 μπορεί να χρησιμοποιηθεί και ως κανόνας επανεκκίνησης του προγράμματος και όχι μόνο για τον καθορισμό της αρχικής κατάστασης.

Ο κανόνας move/0 (ο οποίος εκτελεί κάθε μεταφορά) καλεί τον κανόνα pr/0 (ο οποίος εκτυπώνει την τρέχουσα κατάσταση) μετά από κάθε μεταφορά για τη διευκόλυνση του χρήστη.

Ο κανόνας move/0 επιτρέπει την εισαγωγή της κατεύθυνσης προς την οποία θα γίνει η μετακίνηση, αν και δεν χρειάζεται απαραίτητα, επειδή η κατεύθυνση θα μπορούσε να καθοριστεί αυτόματα από τη θέση του φακού. Παρόλα αυτά, επέλεξα να δώσω στον χρήστη μεγαλύτερη ελευθερία.

Για την εσωτερική αναπαράσταση των τουριστών χρησιμοποίησα ids, δηλαδή ξεχωριστούς αριθμούς για κάθε τουρίστα από το 1 μέχρι το πλήθος τους. Αυτό έγινε για την καλύτερη διαχείριση τους και τη βελτίωση της ευελιξίας του προγράμματος. Με αυτόν τον τρόπο αποφεύγονται ευκολότερα οι σύγχυσεις κατά τη μετακίνηση των τουριστών με τον ίδιο χρόνο μετακίνησης.

Σε κανένα σημείο δεν πρέπει να δοθεί ως είσοδος στο πρόγραμμα αριθμός ακολουθούμενος από γράμματα (π.χ. 123asd) γιατί προκύπτει σφάλμα το οποίο δεν μπορεί να αποτραπεί. Αν αυτό συμβεί σε κάποιο σημείο, τότε επανακινήστε το πρόγραμμα καλώντας τον κανόνα insTour/0.

Στην αναφορά αυτή, τα ερωτήματα (γ) και (δ) έχουν συγχωνευτεί και περιγράφονται από τα κατηγορήματα και τους κανόνες που έχουν χρησιμοποιηθεί στο πρόγραμμα.

Για το ερώτημα (ε), υποθέτουμε ότι η όχθη στην οποία θέλουν να φτάσουν όλοι οι τουρίστες είναι η κάτω. Άρα η κατάσταση-στόχος είναι "down". Τα άλλα δεδομένα εισόδου είναι ίδια με αυτά της αρχικής κατάστασης της εκφώνησης.

Ο κώδικας έχει περιοριστεί ως προς το πλάτος του κατά το γράψιμό του, ώστε να αποφευχθεί η παραμόρφωση του κατά τη μεταφορά του στο έγγραφο Word.

Χρησιμοποιείται η εντολή ‘:-style_check(-singleton).’ για τη σίγαση των warnings."

α)

Για να αναπαρασταθεί κάθε κατάσταση του προβλήματος, θα χρησιμοποιήσουμε τα κατηγορήματα:

1. Τοποθεσία ενός τουρίστα(location/2): Αυτό θα είναι σημαντικό για να ξέρουμε την πλευρά του ποταμού στην οποια βρίσκεται κάθε τουρίστας. Χρησιμοποιείται ως 'location(tourist_id, up/down)'

2. Πλήθος τουριστών σε κάθε όχθη(count/2): Εάν ξέρουμε το πλήθος των τουριστών σε κάθε όχθη, μπορούμε να κάνουμε πιο εύκολα την μεταφορά και να ελέγξουμε τους περιορισμούς που πρέπει να ισχύουν. Χρησιμοποιείται ως 'count(up/down, numberOfTourists)'

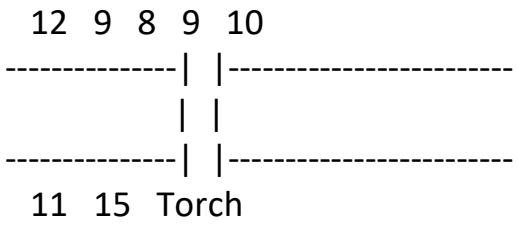
3. Χρόνος που κάθε τουρίστας χρειάζεται για να περάσει την γέφυρα(travel_time/2): Αυτό θα μας βοηθήσει να εκτιμήσουμε το χρονικό κόστος κάθε μεταφοράς.
Χρησιμοποιείται ως 'travel_time(tourist_id, tourist_transition_time)'

4. Τοποθεσία του φακού(torch_location/1): Εάν ξέρουμε που βρίσκεται ο φακός πριν από κάθε μεταφορά, μπορούμε να ξέρουμε αν αυτή μπορεί να πραγματοποιηθεί.
Χρησιμοποιείται ως 'torch_location(up/down)'

5. Η κατάσταση στόχος(goal/1): Θα μας βοηθήσει να αφήσουμε αύτον που εισάγει τα δεδομένα στο πρόγραμμα να ορίσει την όχθη στην οποία πρέπει να καταλήξουν οι τουρίστες.
Χρησιμοποιείται ως 'goal(up/down)'

β)

Η απεικόνιση της κατάστασης της εκφώνησης του παραδείγματος θα είναι η εξής:



Ομοίως θα απεικονιστούν και οι υπόλοιπες καταστάσεις.

γ) δ)

Στην αρχή του προγράμματος ‘καθαρίζουμε’ όλα τα δυναμικαά κατηγορήματα ώστε να μην επηρεάζονται από προηγούμενες εκτελέσεις του προγράμματος.

Στην συνέχεια ορίζουμε ως δυναμικά τα κατηγορήματα:

location/2
count/2
torch_location/1
travel_time/2
counter/1
touristNo/1
touristNoPr/1
goal/1

Το location αφορά στην τοποθεσία των τουριστών (πάνω-κάτω από το ποτάμι).

Το count αφορά στον αριθμό των τουριστών που βρίσκονται σε κάθε όχθη.

Το torch_location χρησιμοποιείται για να ξέρουμε που βρίσκεται ο φακός.

Το travel_time αφορά στον χρόνο μετακίνησης κάθε τουρίστα.

Ο counter χρησιμεύει στην καταμέτρηση του συνολικού χρόνου των μεταφορών των τουριστών.

Το touristNo και το touristNoPr αποθηκεύουν το συνολικό πλήθος των τουριστών.
Η διαφορά του touristNo και του touristNoPr είναι οτι:
το πρώτο χρησιμοποιείται κατά την εισαγωγή των τουριστών στο πρόγραμμα για να δώσει κατάλληλα id σε κάθε τουρίστα βάσει του συνολικού τους αριθμού και η τιμή του μεταβάλλεται ώστε να αποδοθούν id στους τουρίστες, ενώ
το δεύτερο χρησιμοποιείται κατά την εκτύπωση της εκάστοτε παρούσας κατάστασης στον χρήστη.

Το goal αποθηκεύει την κατάσταση στόχο, δηλαδή την μεριά στην οποία πρέπει να καταλήξουν όλοι οι τουρίστες.

Ύστερα ορίζω 2 κανόνες (suma/3, suba/3) που χρησιμεύουν στην πραγματοποίηση πρόσθεσης και αφαίρεσης τα οποία θα χρησιμοποιηθούν από επόμενους κανόνες.

Το επόμενο μέρος εστιάζει στην εισαγωγή των τουριστών.

Δημιουργούνται οι κανόνες:

fix_init_count/1 το οποίο μετρά τους τουρίστες κάθε πλευράς,

add_torch/0 το οποίο τοποθετεί τον φακό στην πλευρά που ορίζει ο χρήστης,

reduce_tour_count/0 το οποίο μειώνει τον μετρητή touristNo κατά 1,

add_tourist/3 το οποίο εισάγει δυναμικά έναν τουρίστα στο πρόγραμμα,

add_tourists/0 το οποίο χρησιμοποιώντας τα προηγούμενα κατηγορήματα εισάγει τους χρήστες που ζητά ο χρήστης και ενημερώνει το count/2,

insTour/0 το οποίο καθαρίζει το πρόγραμμα από δεδομένα που προέρχονται από προηγούμενες εκτελέσεις του και χειρίζεται δυναμικά την εισαγωγή χρηστών, την τοποθέτηση του φακού και τον καθορισμό της κατάστασης-στόχου.

Ύστερα δημιουργούμε κανόνες για την εκτύπωση της κάθε κατάστασης.

Ο κανόνας prints/2 τυπώνει έναν τουρίστα S μόνο αν η θέση του είναι X.

Ο κανόνας printπ/1 τυπώνει τον πυρσό μόνο αν η θέση του είναι X.

Ο κανόνας printUpDown/2 τυπώνει όλους τους τουρίστες που βρίσκονται στη μεριά D του ποταμού.

Ο κανόνας pr/0 χρησιμοποιεί όλους τους προηγούμενους κανόνες για να εκτυπώσει την τρέχουσα κατάσταση στον χρήστη.

Οι επόμενοι κανόνες αφορούν την μετακίνηση των τουριστών και τον έλεγχο των δεδομένων που δίνει ο χρήστης για να κάνει την μετακίνηση.

Ο κανόνας equals_up_down/2 χρησιμοποιείται για να ελέγξει αν το στοιχείο X είναι ίσο με “up” ή ”down” κάνοντας το R 1 αν ισούται ενώ αν δεν ισούται, το R γίνεται 0.

Ο κανόνας allow_movement/3 χρησιμοποιείται για να ελέγξει την μετακίνηση Num αριθμού ατόμων προς κατεύθυνση Dir και ανάλογα με το αν επιτρέπεται ή όχι η μετακίνηση κάνει την Out ίση με 1 ή 0 αντιστοιχα.
Αυτός ο κανόνας φροντίζει κάθε μετακίνηση να αφορά 1 έως 3 άτομα και να μην αφήνει σε καμία όχθη έναν τουρίστα μόνο του.

Ο κανόνας fix_count/2 διορθώνει το κατηγόρημα count/2 κάθε φορά που γίνεται μία μετακίνηση.

Ο κανόνας move_torch/1 μετακινεί τον φακό σε μία όχθη του ποταμιού.

Ο κανόνας other_side/2 χρησιμοποιείται για να πάρουμε την αντίθετη όχθη του ποταμού (π.χ. αν δωσουμε “up” να παρουμε “down”).

Ο κανόνας max/3 επιλέγει το μέγιστο μεταξύ 2 αριθμών.

Ο κανόνας move_one_persoon/2 αφορά στην εισαγωγή από τον χρήστη και στην μετακινηση ενός τουρίστα και χρησιμοποιείται πιο κάτω για μετακινήσεις πολλών τουριστών.

Ο κανόνας update_time/1 χρησιμοποιείται για να προσθέσει Τ χρονικες στιγμες στον μετρητή του συνολικού χρόνου των μετακινήσεων.

Ο κανόνας move_da_people/2 χρησιμοποιείται για να υλοποιήσει μια μετακίνηση ενός μέχρι τριών ατόμων. Για να το επιτύχει, παίρνει δεδομένα από τον χρήστη και καλεί τον κανόνα move_one_person/2.

Ο κανόνας finalState/0 ελέγχει αν η κατάσταση στην οποια καταλήξαμε μετά την μετακίνηση είναι κατάσταση-στόχος και αν είναι τότε τυπώνει κατάλληλο μήνυμα.

Τέλος, ο κανόνας move/0 καλείται από τον χρήστη ώστε να γίνει μια μεταφορά τουριστών. Χρησιμοποιώντας τους προηγουμένους κανόνες ελέγχει και πραγματοποιεί κάθε μεταφορά τουριστών.

Κατά την αρχικοποίηση της εκτέλεσης του προγράμματος καλούμε τον κανόνα insTour/0 για να διευκολύνει τον χρήστη να εισάγει την αρχική κατάσταση του προβλήματος.

ε)

Ανοίγουμε ένα windows terminal και μεταφερόμαστε με την εντολή cd στο φάκελο στον οποίο υπάρχει ο κώδικας και ύστερα εκτελούμε ‘swipl’.

```
PS D:\Desktop> swipl
Welcome to SWI-Prolog (threaded, 64 bits, version 9.0.3)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

1 ?-
```

Ύστερα, εκτελούμε το αχείο της prolog με “consult(‘ask3.pl’).” και καλούμαστε να δώσουμε την αρχική κατάσταση του προβλήματος.

```
2 ?- consult('ask3.pl').
Total number of tourists: 7.
Location of tourist number 1: |: up.
Time needed for this tourist to pass the river: |: 12.
Location of tourist number 2: |: up.
Time needed for this tourist to pass the river: |: 9.
Location of tourist number 3: |: up.
Time needed for this tourist to pass the river: |: 8.
Location of tourist number 4: |: up.
Time needed for this tourist to pass the river: |: 9.
Location of tourist number 5: |: up.
Time needed for this tourist to pass the river: |: 10.
Location of tourist number 6: |: down.
Time needed for this tourist to pass the river: |: 15.
Location of tourist number 7: |: down.
Time needed for this tourist to pass the river: |: 11.
Location of the torch (up/down): |: down.
Side of the river that all tourists must end up to (up/down): |: down.


   10   9   8   9   12
---------------|  |-------------------------
               |  |
---------------|  |-------------------------
   11   15   Torch

Total time: 0.


To move type 'move.'
To print current state type 'pr.'
true.

2 ?-
```

Μετά, εκτελούμε μια μεταφορά 2 τουριστών με χρόνους 11 και 15 στην πάνω όχθη.

```
2 ?- move.
Number of tourists to travel (1-3): 2.
Travel to direction (up/down): |: up.
Time of the first tourist: |: 11.
Time of the second tourist: |: 15.

->Transition time: 15


   11   15   10   9   8   9   12   Torch
---------------|  |--------------------------
               |  |
---------------|  |--------------------------


Total time: 15.


true.
```

Έπειτα, εκτελούμε μεταφορά 3 τουριστών με χρόνους 15, 12 και 8 στην κάτω όχθη.

```
3 ?- move.
Number of tourists to travel (1-3): 3.
Travel to direction (up/down): |: down.
Time of the first tourist: |: 15.
Time of the second tourist: |: 12.
Time of the third tourist: |: 8.

->Transition time: 15


   11   10   9   9
---------------|  |--------------------------
               |  |
---------------|  |--------------------------
   15   8   12   Torch

Total time: 30.


true.
```

Ύστερα, εκτελούμε μεταφορά 1 τουρίστα με χρόνο 8 στην πάνω όχθη.

```
3 ?- move.
Number of tourists to travel (1-3): 1.
Travel to direction (up/down): |: up.
Time of the tourist: |: 8.

->Transition time: 8


   11   10   9   8   9   Torch
---------------|  |-------------------------
               |  |
---------------|  |-------------------------
   15   12

Total time: 38.


true.
```

Μετά, εκτελούμε μεταφορά 3 τουριστών με χρόνους 8, 10 και 11 στην κάτω όχθη.

```
3 ?- move.
Number of tourists to travel (1-3): 3.
Travel to direction (up/down): |: down.
Time of the first tourist: |: 8.
Time of the second tourist: |: 10.
Time of the third tourist: |: 11.

->Transition time: 11


   9   9
---------------|  |-------------------------
               |  |
---------------|  |-------------------------
   11   15   10   8   12   Torch

Total time: 49.


true.
```

Έπειτα, εκτελούμε μεταφορά 1 τουρίστα με χρόνο 8 στην πάνω όχθη.

```
3 ?- move.
Number of tourists to travel (1-3): 1.
Travel to direction (up/down): |: up.
Time of the tourist: |: 8.

->Transition time: 8


   9   8   9   Torch
---------------|  |--------------------------
               |  |
---------------|  |--------------------------
   11   15   10   12

Total time: 57.


true.
```

Ύστερα, εκτελούμε μεταφορά 3 τουριστών με χρόνους 8, 9 και 9 στην κάτω όχθη.

```
3 ?- move.
Number of tourists to travel (1-3): 3.
Travel to direction (up/down): |: down.
Time of the first tourist: |: 8.
Time of the second tourist: |: 9.
Time of the third tourist: |: 9.

->Transition time: 9



---------------|  |-------------------------
               |  |
---------------|  |-------------------------
   11   15   10   9   8   9   12   Torch

Total time: 66.



GOAL STATE ACHIEVED!!!
All tourists are in the corrent side of the river!!!
true.

3 ?-
```

Το πρόγραμμα εκτυπώνει μήνυμα ότι βρισκόμαστε στην κατάσταση-στόχο και βλέπουμε ότι σύνολο οι μεταφορές είχαν διάρκεια 66 χρονικές μονάδες.

ΚΩΔΙΚΑΣ

```
:- abolish(location/2).
:- abolish(count/2).
:- abolish(torch_location/1).
:- abolish(travel_time/2).
:- abolish(counter/1).
:- abolish(touristNo/1).
:- abolish(touristNoPr/1).
:- abolish(goal/1).

:-style_check(-singleton).

:- dynamic(location/2).
:- dynamic(count/2).
:- dynamic(torch_location/1).
:- dynamic(travel_time/2).
:- dynamic(counter/1).
:- dynamic(touristNo/1).
:- dynamic(touristNoPr/1).
:- dynamic(goal/1).

counter(0).


suma(X, Y, Z) :- Z is X + Y.
suba(X, Y, Z) :- Z is X - Y.


fix_init_count(Dir) :-
                        count(Dir, D),
                        retractall(count(Dir, _)),
                        suma(D, 1, Piz),
                        assertz(count(Dir, Piz)).


add_torch :-
   write('Location of the torch (up/down): '),
   read(X), count(X, N), N>0,
   assertz(torch_location(X)),!;
   write('Not a valid location!\n'), add_torch.
```

```
reduce_tour_count :- touristNo(X),
   retractall(touristNo(X)),
   suba(X,1,S), assertz(touristNo(S)).

add_tourist(No, Location, Time) :-
   assertz(location(No,Location)),
   assertz(travel_time(No,Time)).

add_tourists :- touristNo(X), X = 0.
add_tourists :-
   touristNo(X), X =\= 0,
   touristNoPr(Y),
   suba(Y,X,Z), suma(Z,1,W),
   write('Location of tourist number '),
   write(W),
   write(': '),read(Location),
   write('Time needed for this tourist '),
   write('to pass the river: '),
   read(Time),
   add_tourist(X,Location,Time),
   fix_init_count(Location),
   reduce_tour_count, add_tourists.


insTour :-
   retractall(location(X,_)),
   retractall(travel_time(X,_)),
   retractall(touristNo(X)),
   retractall(touristNoPr(X)),
   retractall(torch_location(X)),
   retractall(count(X,_)),
   retractall(goal(X)),

   write('Total number of tourists: '),
   read(TourNo), integer(TourNo), TourNo>0,

   assertz(count(up,0)),
   assertz(count(down,0)),
   assertz(touristNo(TourNo)),
   assertz(touristNoPr(TourNo)),
   add_tourists, add_torch,

   write('Side of the river that all tourists must '),
   write('end up to (up/down): '),
   read(Goal), (Goal=up;Goal=down), assertz(goal(Goal)),

   pr, write('To move type \'move.\'\n'),
```

```
   write('To print current state type \'pr.\'\n'),
   !;
   write('Invalid Input!\n'),insTour.


prints(S, X) :- X=up, not(location(S,X)),!.
prints(S, X) :- X=up, location(S,X), travel_time(S,Y),
   tab(3), write(Y),!.
prints(S, X) :- X=down, not(location(S,X)),!.
prints(S, X) :- X=down, location(S,X), travel_time(S,Y),
   tab(3), write(Y).

printp(X) :- X=up, torch_location(up),
   tab(3), write('Torch'),!.
printp(X) :- X=up, torch_location(down),!.
printp(X) :- X=down, torch_location(down),
   tab(3), write('Torch'),!.
printp(X) :- X=down, torch_location(up),!.


printUpDown(X, D) :- touristNoPr(N), X>N, !.
printUpDown(X, D) :- location(X, D),prints(X,D),
   suma(X,1,S),printUpDown(S, D),!;
   suma(X,1,S),printUpDown(S, D).


pr :- nl, nl,
   printUpDown(1, up),
   printp(up),
   nl,
   write('---------------|  |-------------------------'),nl,
   write('               |  |                         '),nl,
   write('---------------|  |-------------------------'),nl,
   printUpDown(1, down),
   printp(down), nl, nl,
   write('Total time: '), counter(W), write(W),
   write('.'),
   nl,nl,nl,!.

equals_up_down(X,R) :- X=up,R=1,!;X=down,R=1,!;R=0.
```

```
allow_movement(Num, Dir, Out) :-

   (integer(Num),
   (Dir=up,
      torch_location(down),
      count(down,DownNo),
      (Num=1,count(up,C),C=\=0;Num=\=1),
      DownNo - Num =\= 1,
      DownNo - Num >= 0;
   Dir=down,
      torch_location(up),
      count(up,UpNo),
      (Num=1,count(down,C),C=\=0;Num=\=1),
      UpNo - Num =\= 1,
      UpNo - Num >= 0
   ),
   (Num = 1; Num = 2; Num = 3), Out=1,!);

   not(integer(Num)),
      write('Invalid number of tourists!'),Out=0, !;
   Num < 1, write('Invalid number of tourists!'),Out=0,!;
   Num > 3,
   write('Too many tourists! The bridge might '),
      write('collapse!'),Out=0, !;

   equals_up_down(Dir,R), R=0, write('Not a valid '),
      write('direction!'),Out=0,!;

   Dir=up,Num=1,count(up,C),C=0,
   write('Cant leave one person alone! '),
      write('Too dangerous!'),Out=0, !;
   Dir=down,Num=1,count(down,C),C=0,
   write('Cant leave one person alone! '),
      write('Too dangerous!'),Out=0, !;

   Dir=up,count(down,C),suba(C, Num, E), E < 0,
   write('Not enough tourists for the transit '),
      write('to happen!'),Out=0, !;
   Dir=down,count(up,C),suba(C, Num, E), E < 0,
   write('Not enough tourists for the transit '),
      write('to happen!'),Out=0, !;

   Dir=up,torch_location(up),
   write('Cant move in the dark...!'),Out=0, !;
   Dir=down,torch_location(down),
   write('Cant move in the dark...!'),Out=0, !;
   Dir=up,count(down,C),suba(C, Num, E), E = 1,
```

```
   write('Cant leave one person alone! '),
      write('Too dangerous!'),Out=0, !;
   Dir=down,count(up,C),suba(C, Num, E), E = 1,
   write('Cant leave one person alone! '),
      write('Too dangerous!'),Out=0, !.



fix_count(Num, Dir) :-
   count(up, Up),
   count(down, Down),
   retractall(count(up, _)),
   retractall(count(down, _)),
   (Dir=up,

      suma(Up, Num, Piz),
      suba(Down, Num, Pizz)
                  ;
   Dir=down,
      suba(Up, Num, Piz),
      suma(Down, Num, Pizz)
   ),
   assertz(count(up, Piz)),
   assertz(count(down, Pizz)).


move_torch(Dir) :-
   retractall(torch_location(_)),
   assertz(torch_location(Dir)).

other_side(X, R) :- X=up,R=down,!; X=down,R=up,!.

max(A,B,C) :- A>=B, C=A,!; A<B, C=B.

move_one_person(Dir, F) :-
   read(F),travel_time(X, F), other_side(Dir, DirR),
      location(X,DirR),
      retractall(location(X, DirR)),
      assertz(location(X, Dir)),!;
   write('Tourist does not exists :('),nl,
      write('Try Again: '),move_one_person(Dir,F).


update_time(T) :- counter(X), retractall(counter(X)),
                  suma(X,T,S), assertz(counter(S)).

move_da_people(Num,Dir) :-
```

```
    Num=1, write('Time of the tourist: '),
       move_one_person(Dir, T), nl,
       write('->Transition time: '), write(T), nl,
       update_time(T),!;

    Num=2, write('Time of the first tourist: '),
       move_one_person(Dir, T1),
       write('Time of the second tourist: '),
       move_one_person(Dir, T2),
       max(T1,T2,T), nl,
       write('->Transition time: '), write(T), nl,
       update_time(T),!;

    Num=3, write('Time of the first tourist: '),
       move_one_person(Dir, T1),
       write('Time of the second tourist: '),
       move_one_person(Dir, T2),
       write('Time of the third tourist: '),
       move_one_person(Dir, T3),
       max(T1,T2,T12), max(T12,T3,T), nl,
       write('->Transition time: '), write(T), nl,
       update_time(T),!.


finalState :- goal(Goal), count(Goal, X), touristNoPr(N),
    X=N, write('\nGOAL STATE ACHIEVED!!!\n'),
    write('All tourists are in the corrent '),
    write('side of the river!!!\n'),!;
    goal(Goal), count(Goal, X),!.


move :-
    write('Number of tourists to travel (1-3): '),
    read(Tr_count),
    write('Travel to direction (up/down): '), read(Dir),

    allow_movement(Tr_count, Dir, Out),
    Out=1,
       fix_count(Tr_count, Dir),
       move_torch(Dir),
       move_da_people(Tr_count,Dir),pr,finalState,!;
    Out=0,pr,!.


:-initialization(insTour).
```