Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ
(ТУСУР)

Кафедра компьютерных систем в управлении и проектировании
(КСУП)

Отчёт по контрольной работе № 2
по дисциплине
«Искусственный интеллект»
Вариант 3

Томск – 2021

Вариант 3 "Миссионеры и людоеды". Поиск в глубину

Миссионеры и людоеды:

Три миссионера и три людоеда находятся по одну сторону реки, через которую они хотят переправиться. В их распоряжении имеется лодка, которая может выдержать вес только двух человек. Кроме того, если в какой-то момент число людоедов станет больше числа миссионеров, миссионеры будут съедены независимо от того, на каком берегу реки это случится.

Указания к решению:

Различные состояния этой задачи однозначно задаются
информацией, на каком берегу находятся лодка и сколько миссионеров и людоедов на этом же берегу.

Поэтому структура

```
state(ЛокализацияЛодки, ЧислоМиссионеровНаТомБерегуГдеЛодка,
ЧислоЛюдоедовНаТомБерегуГдеЛодка)
```

полностью описывает состояние. Допустимые состояния для решения задачи - это те, когда людоеды не могут съесть миссионеров ни на том берегу, где лодка, ни на противоположном,

Возможные значение первого аргумента: атомы west (западный берег) и east (восточный берег). Возможные значения остальных аргументов: 0, 1, 2 или 3.

Начальное состояние: state(east,3, 3).
Конечное состояние: state(west,3,3).

Решение:

По приведенным тут указаниям решить задачу не удалось, так как мало знать сколько людоедов и миссионеров, на берегу - надо знать еще сколько находится в лодке - иначе как знать сколько из них высадить?

Поэтому было взято такое состояние:

```
state(БерегЛодки,
      берег(Миссионеров, Людоедов),
      лодка(МиссионеровЛодка, ЛюдоедовЛодка)).
```

Однако, допустим мы попали в состояние когда на восточном берегу 2 миссионера и 1 людоед, а также в лодке 2 людоеда. **Надо определить возможное следующее состояние.** Чтобы сделать это надо сначала рассчитать "а что там, на втором берегу", вычислить можно так:

```
МиссионеровНаДругом = 3 -  Миссионеров -  МиссионеровЛодка,
ЛюдоедовНаДругом = 3 -  Людоедов  -  ЛюдоедовЛодка.
```

Так как такой расчет надо выполнять часто - для этого написан предикат:

```
другой_берег(west, east).
другой_берег(east, west).

расчет_друго_берега(
  берег(Миссионеров, Людоедов),
  лодка(МиссионеровЛодка, ЛюдоедовЛодка),
  берег(МиссионеровНаДругом, ЛюдоедовНаДругом)):-

    МиссионеровНаДругом is 3 -  Миссионеров -  МиссионеровЛодка,
      ЛюдоедовНаДругом is 3 -  Людоедов  -  ЛюдоедовЛодка.
```

Тут виден также вспомогательный предикат, возвращающий противоположный берег (west->east и наоборот).

В ходе поиска решения программа будет перебирать состояния, для каждого из которых надо проверять допустимо ли оно. Допустимость - это не только Миссионеров >= Людоедов, ведь если миссионеров нет - то людоедов на берегу можно оставить сколько угодно. Это также оформлено в виде отдельного правила:

```
можно_оставить(берег(Миссионеров, Людоедов)):-
    % если миссионеров больше
    Миссионеров >= Людоедов;
    % или если некого есть
    Миссионеров is 0.
```

Сама по себе смена состояний - это не так просто ведь:

- исходя из состояния лодки и берега надо понять кто поплывет назад;
- исходя из состояния лодки и берега надо определить кто там на противоположном берегу.

Поэтому предикат поиска следующего состояния разбит на 2 части - разгрузку/загрузку, которые выполняются в рамках текущего берега, а также - переправы (при этом меняется берег):

```
следующее_состояние(Состояние, Следующее):-
    разгрузка_загрузка(Состояние, СостояниеПослеПерегрузки),
    переправа(СостояниеПослеПерегрузки, Следующее).
```

Цель разгрузки и загрузки - сформировать лодку для отплытия назад из тех людей, что есть в лодке и на текущем берегу:

```
разгрузка_загрузка(
  state(БерегЛодки,
        берег(МиссионеровБерег, ЛюдоедовБерег),
        лодка(МиссионеровЛодка, ЛюдоедовЛодка)),

  state(БерегЛодки,
        берег(МиссионеровБерегПосле, ЛюдоедовБерегПосле),
        лодка(МиссионеровЛодкаПосле, ЛюдоедовЛодкаПосле))):-

    % выгрузка (всех сложить):
    ВсегоМиссионеров is МиссионеровБерег + МиссионеровЛодка,
    ВсегоЛюдоедов is ЛюдоедовБерег + ЛюдоедовЛодка,

    % выбрать кто плывет назад:
    between(0, 2, МиссионеровЛодкаПосле),
    МиссионеровЛодкаПосле =< ВсегоМиссионеров,

    between(0, 2, ЛюдоедовЛодкаПосле),
    ЛюдоедовЛодкаПосле =< ВсегоЛюдоедов,

    % проверка емкости лодки
    ЛюдейВЛодке is ЛюдоедовЛодкаПосле + МиссионеровЛодкаПосле,
    ЛюдейВЛодке > 0, ЛюдейВЛодке =< 2,
```

```
    % расчет кто остался на берегу
    МиссионеровБерегПосле is ВсегоМиссионеров - МиссионеровЛодкаПосле,
    ЛюдоедовБерегПосле is ВсегоЛюдоедов - ЛюдоедовЛодкаПосле,

    % проверка что на берегу не съедят миссонеров:
    можно_оставить(берег(МиссионеровБерегПосле, ЛюдоедовБерегПосле)).
```

Назад поплывет 0, 1 или 2 миссионера, а также 0, 1 или 2 людоеда. Для перебора их количества используется встроенная функция between. Она работает так:

```
% Вызов:
?- between(3, 7, X).
% Результаты:
X = 3;
X = 4;
X = 5;
X = 6;
X = 7.
```

То есть она перебирает все числа в диапазоне.

После того как выбрано не более двух людоедов и не более двух миссионеров - программа считает их сумму (сколько человек поплывет в лодке) и проверяет ограничение вместимости лодки.

Исходя из того, кто оказался в лодке - программа определяет кто остался на берегу.

При выполнении переправы содержимое лодки оказывается на другом берегу, однако чтобы узнать "кто там" - выполняется расчет другого берега, как было показано выше:

```
переправа(state(БерегЛодки,
        берег(МиссионеровБерег, ЛюдоедовБерег),
        лодка(МиссионеровЛодка, ЛюдоедовЛодка)),

        state(ДругойБерегЛодки,
        берег(МиссионеровДругойБерег, ЛюдоедовДругойБерег),
        лодка(МиссионеровЛодка, ЛюдоедовЛодка))):-

    другой_берег(БерегЛодки, ДругойБерегЛодки),
    расчет_друго_берега(
        берег(МиссионеровБерег, ЛюдоедовБерег),
        лодка(МиссионеровЛодка, ЛюдоедовЛодка),
        берег(МиссионеровДругойБерег, ЛюдоедовДругойБерег)
    ).
```

Теперь, когда описано правило смены состояний, остается лишь добавить правило поиска в глубину:

```
% поиск в глубину
dfs(Состояние, _, Path):-
    Состояние = state(west,
                      берег(МиссионеровБерег, ЛюдоедовБерег),
```

```
                        лодка(МиссионеровЛодка, ЛюдоедовЛодка)),
    ВсегоМиссионеров is МиссионеровБерег + МиссионеровЛодка,
    ВсегоЛюдоедов is ЛюдоедовБерег + ЛюдоедовЛодка,
    ВсегоМиссионеров is 3, ВсегоЛюдоедов is 3,
    Path = [Состояние, state(west, берег(3, 3), лодка(0, 0))].
dfs(Состояние, Старые, Path):-
    следующее_состояние(Состояние, Следующее),
      not(member(Следующее, Старые)),
      dfs(Следующее, [Состояние|Старые], ПутьИзСледующего),
    Path = [Состояние|ПутьИзСледующего].
```

Это правило состоит из двух частей. Первая часть проверяет окончание переправы. Такое случится если ложка окажется на правом берегу west и если собрать лодку и текущий берег - то на них окажутся все 3 людоеда и все 3 миссионера.

Такой случай описан отдельно, так как если из этого состояния выполнить генерацию следующих - то программа обязательно сформирует назад НЕПУСТУЮ лодку, а значит - на берегу будут не все люди.

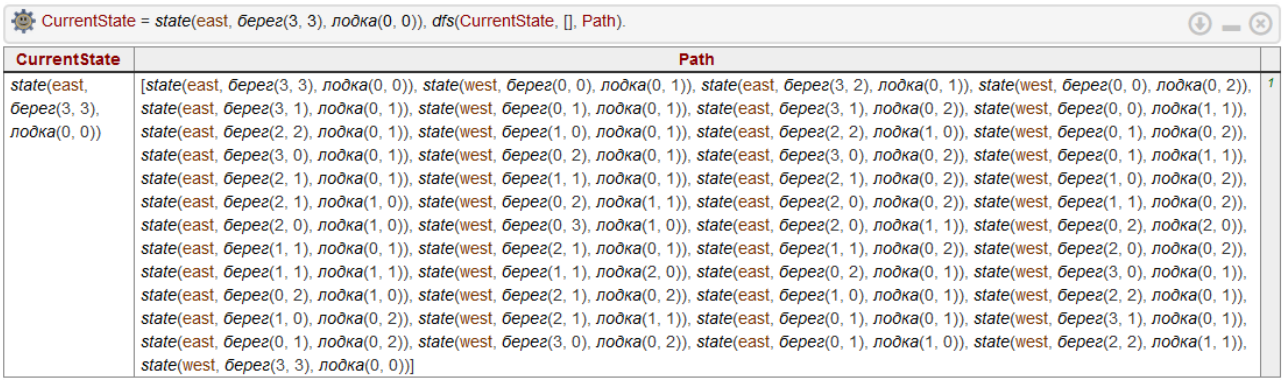
Если же текущее состояние не конечное - то на его основе формируется следующее состояние и программа ищет из него.

Чтобы один людоед не плавал вечно с одного берега на другой - программа хранит список уже посещенных состояний (передается в dfs в виде второго аргумента - списка).

Вызывать функцию надо так:

```
CurrentState = state(east, берег(3, 3), лодка(0, 0)),
dfs(CurrentState, [], Path).
```

Результат ее работы:


CurrentState = state(east, берег(3, 3), лодка(0, 0)), dfs(CurrentState, [], Path).

CurrentState	Path
state(east, берег(3, 3), лодка(0, 0))	[state(east, берег(3, 3), лодка(0, 0)), state(west, берег(0, 0), лодка(0, 1)), state(east, берег(3, 2), лодка(0, 1)), state(west, берег(0, 0), лодка(0, 2)), state(east, берег(3, 1), лодка(0, 1)), state(west, берег(0, 1), лодка(0, 1)), state(east, берег(3, 1), лодка(0, 2)), state(west, берег(0, 0), лодка(1, 1)), state(east, берег(2, 2), лодка(0, 1)), state(west, берег(1, 0), лодка(0, 1)), state(east, берег(2, 2), лодка(1, 0)), state(west, берег(0, 1), лодка(0, 2)), state(east, берег(3, 0), лодка(0, 1)), state(west, берег(0, 2), лодка(0, 1)), state(east, берег(3, 0), лодка(0, 2)), state(west, берег(0, 1), лодка(1, 1)), state(east, берег(2, 1), лодка(0, 1)), state(west, берег(1, 1), лодка(0, 1)), state(east, берег(2, 1), лодка(0, 2)), state(west, берег(1, 0), лодка(0, 2)), state(east, берег(2, 1), лодка(1, 0)), state(west, берег(0, 2), лодка(1, 1)), state(east, берег(2, 0), лодка(0, 2)), state(west, берег(1, 1), лодка(0, 2)), state(east, берег(2, 0), лодка(1, 0)), state(west, берег(0, 3), лодка(1, 0)), state(east, берег(2, 0), лодка(1, 1)), state(west, берег(0, 2), лодка(2, 0)), state(east, берег(1, 1), лодка(0, 1)), state(west, берег(2, 1), лодка(0, 1)), state(east, берег(1, 1), лодка(0, 2)), state(west, берег(2, 0), лодка(0, 2)), state(east, берег(1, 1), лодка(1, 1)), state(west, берег(1, 1), лодка(2, 0)), state(east, берег(0, 2), лодка(0, 1)), state(west, берег(3, 0), лодка(0, 1)), state(east, берег(0, 2), лодка(1, 0)), state(west, берег(2, 1), лодка(0, 2)), state(east, берег(1, 0), лодка(0, 1)), state(west, берег(2, 2), лодка(0, 1)), state(east, берег(1, 0), лодка(0, 2)), state(west, берег(2, 1), лодка(1, 1)), state(east, берег(0, 1), лодка(0, 1)), state(west, берег(3, 1), лодка(0, 1)), state(east, берег(0, 1), лодка(0, 2)), state(west, берег(3, 0), лодка(0, 2)), state(east, берег(0, 1), лодка(1, 0)), state(west, берег(2, 2), лодка(1, 1)), state(west, берег(3, 3), лодка(0, 0))]