프로그래밍 언어 HW5: PROLOG

B743014 양혜진

May 28, 2021

1 hanoi.pl

1.1 코드 설명

하노이탑 퍼즐을 프롤로그로 구현했다. hanoi(N) 쿼리문에 하노이탑 원판 개수를 입력하면 해당 개수의 원판들을 1에서 2로 옮기게 된다. 만약 N 이 0보다 작은 경우에는 하노이 퍼즐의 조건이 성립되지 않으므로 해당 경우는 예외처리해주었다. honoi(N) 내부에서는 move(N,Start,End, Via) 규칙을 쿼리하게 된다. move()에 들어가는 변수 N은 하노이탑의 원판 개수이고, Start, End, Via 변수는 원판을 Start 위치에서 Via 위치를 경유하여 End 위치로 옮기기 위해 사용하는 변수이다.

하노이탑 퍼즐의 핵심은 가장 큰 밑바닥 원판부터 원하는 위치로 이동시켜야 한다는 것이다. 만약 3개의 원판을 위치 1에서 위치 2로 이동시키는 경우, 3번 원판을 2로 이동시키기 위해서는 1번 원판과 2번 원판이 위치 3에 있어야 한다. 그러므로 원판 3개를 위치 2로 이동시키기 위해서는 우선 원판 2개를 위치 3으로 이동시켜주어야 하는 것이다. 또한, 원판 2개를 위치 3으로 이동시키기 위해서는 가장 위의 원판 1개를 위치 2로 이동시켜주어야 한다. 이렇게 원판을 N개 옮기기 위해서는 원판을 N-1개 옮기는 것이 필요하다. 따라서 해당 퍼즐은 마지막 원판(제일 위의 원판, N=1)이 나올때까지 move()를 재귀 호출하였다. 그 다음, 마지막 원판에 다다르면, N번 원판을 Start 위치에서 End 위치로 이동 시킨다는 내용을 출력해주었다. 해당 내용의 출력에는 write()를 사용해주었다.

move() 내에서 원판이 제일 위의 원판, 즉, N이 1이 아닌경우에는 해당 원판들을 도착지인 위치가 아닌 위치로 이동시켜주어야 한다. 그렇게 해야지 그 아래의 가장 마지막 원판을 도착지로 이동시킬 수 있기 때문이다. 따라서 move(N-1,Start,Via,End)로 move 를 재귀호출해주도록 한다. 그 다음에는 가장 바닥에 있는 원판을 Start 에서 End 위치로 이동시킨다는 내용을 출력해준다. 그 다음에는, 옮겨두었던 N-1개의 원판을 다시 도착지가 위치가 아닌 위치, 즉 경유지 Via 에서 End 위치로 옮겨주도록 move(N-1,Via,End,Start)를 다시 재귀호출해주도록 한다.

1.2 코드

```
hanoi(N):-
    N > 0 ->
    move(N,1,2,3).

move(N,Start,End,Via):-
    N =:= 1 ->
    print(N,Start,End);
    N1 is N - 1,
    move(N1,Start,Via,End),
    print(N,Start,End),
    move(N1,Via,End,Start).

print(N,Start,End):-
```

```
    write(N), write('->['),
    write(Start), write(','),
    write(End), write(']'), nl.
```

1.3 trace 실행결과

hanoi(3)을 trace 에서 실행한 결과이다.

```
[trace]  ?- hanoi(3).
   Call: (10) hanoi(3) ? creep
   Call: (11) 3>0 ? creep
   Exit: (11) 3>0 ? creep
   Call: (11) move(3, 1, 2, 3) ? creep
   Call: (12) 3=:=1 ? creep
   Fail: (12) 3=:=1 ? creep
   Redo: (11) move(3, 1, 2, 3) ? creep
   Call: (12) _6784 is 3+ -1 ? creep
   Exit: (12) 2 is 3+ -1 ? creep
   Call: (12) move(2, 1, 3, 2) ? creep
   Call: (13) 2=:=1 ? creep
   Fail: (13) 2=:=1 ? creep
   Redo: (12) move(2, 1, 3, 2) ? creep
   Call: (13) _7054 is 2+ -1 ? creep
   Exit: (13) 1 is 2+ -1 ? creep
   Call: (13) move(1, 1, 2, 3) ? creep
   Call: (14) 1=:=1 ? creep
   Exit: (14) 1=:=1 ? creep
   Call: (14) print(1, 1, 2) ? creep
   Call: (15) write(1) ? creep
1
   Exit: (15) write(1) ? creep
   Call: (15) write('->[') ? creep
->[
   Exit: (15) write('->[') ? creep
   Call: (15) write(1) ? creep
1
   Exit: (15) write(1) ? creep
   Call: (15) write(',') ? creep
,
   Exit: (15) write(',') ? creep
   Call: (15) write(2) ? creep
2
   Exit: (15) write(2) ? creep
   Call: (15) write(']') ? creep
]
   Exit: (15) write(']') ? creep
   Call: (15) nl ? creep

   Exit: (15) nl ? creep
   Exit: (14) print(1, 1, 2) ? creep
   Exit: (13) move(1, 1, 2, 3) ? creep
   Call: (13) print(2, 1, 3) ? creep
   Call: (14) write(2) ? creep
2
   Exit: (14) write(2) ? creep
   Call: (14) write('->[') ? creep
->[
   Exit: (14) write('->[') ? creep
   Call: (14) write(1) ? creep
1
   Exit: (14) write(1) ? creep
   Call: (14) write(',') ? creep
```

```
,
   Exit: (14) write(',') ? creep
   Call: (14) write(3) ? creep
3
   Exit: (14) write(3) ? creep
   Call: (14) write(']') ? creep
]
   Exit: (14) write(']') ? creep
   Call: (14) nl ? creep

   Exit: (14) nl ? creep
   Exit: (13) print(2, 1, 3) ? creep
   Call: (13) move(1, 2, 3, 1) ? creep
   Call: (14) 1=:=1 ? creep
   Exit: (14) 1=:=1 ? creep
   Call: (14) print(1, 2, 3) ? creep
   Call: (15) write(1) ? creep
1
   Exit: (15) write(1) ? creep
   Call: (15) write('->[') ? creep
->[
   Exit: (15) write('->[') ? creep
   Call: (15) write(2) ? creep
2
   Exit: (15) write(2) ? creep
   Call: (15) write(',') ? creep
,
   Exit: (15) write(',') ? creep
   Call: (15) write(3) ? creep
3
   Exit: (15) write(3) ? creep
   Call: (15) write(']') ? creep
]
   Exit: (15) write(']') ? creep
   Call: (15) nl ? creep

   Exit: (15) nl ? creep
   Exit: (14) print(1, 2, 3) ? creep
   Exit: (13) move(1, 2, 3, 1) ? creep
   Exit: (12) move(2, 1, 3, 2) ? creep
   Call: (12) print(3, 1, 2) ? creep
   Call: (13) write(3) ? creep
3
   Exit: (13) write(3) ? creep
   Call: (13) write('->[') ? creep
->[
   Exit: (13) write('->[') ? creep
   Call: (13) write(1) ? creep
1
   Exit: (13) write(1) ? creep
   Call: (13) write(',') ? creep
,
   Exit: (13) write(',') ? creep
   Call: (13) write(2) ? creep
2
   Exit: (13) write(2) ? creep
   Call: (13) write(']') ? creep
]
   Exit: (13) write(']') ? creep
   Call: (13) nl ? creep

   Exit: (13) nl ? creep
   Exit: (12) print(3, 1, 2) ? creep
   Call: (12) move(2, 3, 2, 1) ? creep
```

```
   Call: (13) 2=:=1 ? creep
   Fail: (13) 2=:=1 ? creep
   Redo: (12) move(2, 3, 2, 1) ? creep
   Call: (13) _10536 is 2+ -1 ? creep
   Exit: (13) 1 is 2+ -1 ? creep
   Call: (13) move(1, 3, 1, 2) ? creep
   Call: (14) 1=:=1 ? creep
   Exit: (14) 1=:=1 ? creep
   Call: (14) print(1, 3, 1) ? creep
   Call: (15) write(1) ? creep
1
   Exit: (15) write(1) ? creep
   Call: (15) write('->[') ? creep
->[
   Exit: (15) write('->[') ? creep
   Call: (15) write(3) ? creep
3
   Exit: (15) write(3) ? creep
   Call: (15) write(',') ? creep
,
   Exit: (15) write(',') ? creep
   Call: (15) write(1) ? creep
1
   Exit: (15) write(1) ? creep
   Call: (15) write(']') ? creep
]
   Exit: (15) write(']') ? creep
   Call: (15) nl ? creep

   Exit: (15) nl ? creep
   Exit: (14) print(1, 3, 1) ? creep
   Exit: (13) move(1, 3, 1, 2) ? creep
   Call: (13) print(2, 3, 2) ? creep
   Call: (14) write(2) ? creep
2
   Exit: (14) write(2) ? creep
   Call: (14) write('->[') ? creep
->[
   Exit: (14) write('->[') ? creep
   Call: (14) write(3) ? creep
3
   Exit: (14) write(3) ? creep
   Call: (14) write(',') ? creep
,
   Exit: (14) write(',') ? creep
   Call: (14) write(2) ? creep
2
   Exit: (14) write(2) ? creep
   Call: (14) write(']') ? creep
]
   Exit: (14) write(']') ? creep
   Call: (14) nl ? creep

   Exit: (14) nl ? creep
   Exit: (13) print(2, 3, 2) ? creep
   Call: (13) move(1, 1, 2, 3) ? creep
   Call: (14) 1=:=1 ? creep
   Exit: (14) 1=:=1 ? creep
   Call: (14) print(1, 1, 2) ? creep
   Call: (15) write(1) ? creep
1
   Exit: (15) write(1) ? creep
   Call: (15) write('->[') ? creep
->[
```

```
   Exit: (15) write('->[') ? creep
   Call: (15) write(1) ? creep
1
   Exit: (15) write(1) ? creep
   Call: (15) write(',') ? creep
,
   Exit: (15) write(',') ? creep
   Call: (15) write(2) ? creep
2
   Exit: (15) write(2) ? creep
   Call: (15) write(']') ? creep
]
   Exit: (15) write(']') ? creep
   Call: (15) nl ? creep

   Exit: (15) nl ? creep
   Exit: (14) print(1, 1, 2) ? creep
   Exit: (13) move(1, 1, 2, 3) ? creep
   Exit: (12) move(2, 3, 2, 1) ? creep
   Exit: (11) move(3, 1, 2, 3) ? creep
   Exit: (10) hanoi(3) ? creep
true.
```

2 quickSort

2.1 코드 설명

퀵소트는 분할 정복 방법을 통해 구현된 정렬 알고리즘의 일종이다. 프롤로그로 퀵소트를 구현하기 위해서 qSortHelper(List,Sorted), divide(Pivot,Lesser,Greater) rule을 작성했다. 먼저 qSortHelper는 quickSort 에서 들어온 List를 재귀호출을 통해 가장 작은 단위로 쪼개기 위해 만들었다. 프롤로그에서는 다른 언어처럼 함수에 의한 반환 값 같은 것이 존재하지 않기 때문에 어떤 값을 결론으로 도출하고 싶다면 rule 에 들어온 변수에 결론을 대입해주어야 한다. 그래서 qSortHelper에는 Sorted 라는 변수를 입력하게 되어있고, 정렬된 리스트는 Sorted 가 된다.

qSortHelper는 들어온 리스트의 가장 첫 번째 값을 Pivot 으로 두고, Pivot 을 제외한 리스트의 나머지 값들을 Pivot 보다 작은 값의 Lesser 와 Pivot 보다 큰 값의 Greater 로 나누게 된다. 이렇게 Pivot 을 기준으로 하여 List 의 값들을 나누기 위해 divide(Pivot,List,Lesser,Greater) 을 호출하게 된다. divide 에서는 List의 값중 맨 앞의 값을 H 로 두고, 만약 H의 값이 Pivot 의 값보다 크면 H를 Greater 에 추가하고, H의 값이 Pivot 의 값보다 작으면 H를 Lesser 에 추가하도록 하였다. 프롤로그에서는 리스트의 값을 인덱스로 접근할 수 없고, 리스트의 앞에서부터 값에 접근할 수 있기 때문에 이러한 방식으로 List의 값을 Pivot 기준으로 나누게 된다. 또한, divide 의 rule 에 들어오는 Lesser 또는 Greater 의 앞에 값 H를 추가할 수 있다.

이러한 과정을통해 qSortHelper 내부에서는 devide로 Lesser 과 Greater 리스트를 얻게 된다. 그 다음, 도출한 Lesser 와 Greater 를 출력하고 qSortHelper 를 재귀호출하여 Lesser 리스트는 SortedLesser 로, Greater 리스트는 SortedGreater 로 정렬하도록 한다. 그러면 재귀호출된 qSortHelper 내부에서 다시 devide 를 통해 Lesser 와 Greater 를 만들어내게 된다.

qSortHelper 와 divide 가 재귀호출되는 과정에서 재귀가 언제까지 성립되어야 하는지를 규칙에 추가해주어야 한다. qSortHelper()의 경우, 빈 리스트가 들어올 경우 더 이상 정렬할 값이 없는 것이기 때문에 재귀호출을 멈추어야 하므로 qSortHelper([],[]):-!. 를 추가해주었다. 또한, divide()의 경우에도 빈 리스트가 들어올 경우 더 이상 Pivot 보다 큰값과 작은 값을 나눌 수 없기 때문에 재귀호출을 멈추어야 하므로 divide(_,[],[],[]):-!. 를 추가해주었다.

qSortHelper 내부에서 qSortHelper 를 재귀호출하여 Lesser 과 Greater 를 SortedLesser 과 SortedGreater 로 정렬한 다음에는 merge 를 사용하여 Lesser, Greater, 과 Pivot 을 하나의 리스트로 병합한 Sorted 를 만들고, 이를 writeMerge 로 출력해주었다.

2.2 코드

```
quickSort([]) :-!.
quickSort(List):-
    qSortHelper(List,_).

qSortHelper([],[]) :-!.
qSortHelper([Pivot|List],Sorted):-
    divide(Pivot,List,Lesser,Greater),
    writeDevide(Pivot,Lesser,Greater),
    qSortHelper(Lesser,SortedLesser),
    qSortHelper(Greater,SortedGreater),
    merge(SortedLesser,[Pivot|SortedGreater],Sorted),
    writeMerge(SortedLesser,Pivot,SortedGreater).

divide(_,[],[],[]) :-!.
divide(Pivot,[H|List],Lesser,[H|Greater]):-
    Pivot =< H -> divide(Pivot,List,Lesser,Greater).
divide(Pivot,[H|List],[H|Lesser],Greater):-
    Pivot > H -> divide(Pivot,List,Lesser,Greater).
```

```
writeDevide(_ ,[],[]) :-!.
writeDevide(Pivot,Lesser,Greater):-
    write('divide = '), write(Pivot), write(' | '),
    write(Lesser), write(Greater), nl.

writeMerge(SortedLesser,Pivot,SortedGreater):-
    write('merge : '),
    list_empty(SortedLesser,R1),list_empty(SortedGreater,R2),
    R1,R2 ->
    write('['), write(Pivot), write(']'), nl;
    write(SortedLesser),
    write('['), write(Pivot), write(']'),
    write(SortedGreater), nl.

list_empty ([], true).
list_empty ([_|_], false).
```

2.3 trace 실행결과

quickSort([7,3,1,2,9,5,4,8])을 trace 에서 실행한 결과이다.

```
[trace]  ?- quickSort ([7,3,1,2,9,5,4,8]) .
Call: (10) quickSort([7, 3, 1, 2, 9, 5, 4, 8]) ? creep
Call: (11) qSortHelper([7, 3, 1, 2, 9, 5, 4, 8], _22254) ? creep
Call: (12) divide(7, [3, 1, 2, 9, 5, 4, 8], _22300, _22302) ? creep
Call: (13) 7=<3 ? creep
Fail: (13) 7=<3 ? creep
Redo: (12) divide(7, [3, 1, 2, 9, 5, 4, 8], _22438, _22440) ? creep
Call: (13) 7>3 ? creep
Exit: (13) 7>3 ? creep
Call: (13) divide(7, [1, 2, 9, 5, 4, 8], _22428, _22578) ? creep
Call: (14) 7=<1 ? creep
Fail: (14) 7=<1 ? creep
Redo: (13) divide(7, [1, 2, 9, 5, 4, 8], _22428, _22716) ? creep
Call: (14) 7>1 ? creep
Exit: (14) 7>1 ? creep
Call: (14) divide(7, [2, 9, 5, 4, 8], _22704, _22854) ? creep
Call: (15) 7=<2 ? creep
Fail: (15) 7=<2 ? creep
Redo: (14) divide(7, [2, 9, 5, 4, 8], _22704, _22992) ? creep
Call: (15) 7>2 ? creep
Exit: (15) 7>2 ? creep
Call: (15) divide(7, [9, 5, 4, 8], _22980, _23130) ? creep
Call: (16) 7=<9 ? creep
Exit: (16) 7=<9 ? creep
Call: (16) divide(7, [5, 4, 8], _22980, _23118) ? creep
Call: (17) 7=<5 ? creep
Fail: (17) 7=<5 ? creep
Redo: (16) divide(7, [5, 4, 8], _22980, _23118) ? creep
Call: (17) 7>5 ? creep
Exit: (17) 7>5 ? creep
Call: (17) divide(7, [4, 8], _23394, _23118) ? creep
Call: (18) 7=<4 ? creep
Fail: (18) 7=<4 ? creep
Redo: (17) divide(7, [4, 8], _23394, _23118) ? creep
Call: (18) 7>4 ? creep
Exit: (18) 7>4 ? creep
Call: (18) divide(7, [8], _23670, _23118) ? creep
Call: (19) 7=<8 ? creep
Exit: (19) 7=<8 ? creep
Call: (19) divide(7, [], _23670, _23808) ? creep
Exit: (19) divide(7, [], [], []) ? creep
```

```
   Exit: (18) divide(7, [8], [], [8]) ? creep
   Exit: (17) divide(7, [4, 8], [4], [8]) ? creep
   Exit: (16) divide(7, [5, 4, 8], [5, 4], [8]) ? creep
   Exit: (15) divide(7, [9, 5, 4, 8], [5, 4], [9, 8]) ? creep
   Exit: (14) divide(7, [2, 9, 5, 4, 8], [2, 5, 4], [9, 8]) ? creep
   Exit: (13) divide(7, [1, 2, 9, 5, 4, 8], [1, 2, 5, 4], [9, 8]) ? creep
   Exit: (12) divide(7, [3, 1, 2, 9, 5, 4, 8], [3, 1, 2, 5, 4], [9, 8]) ? creep
   Call: (12) writeDevide(7, [3, 1, 2, 5, 4], [9, 8]) ? creep
   Call: (13) write('divide = ') ? creep
divide =
   Exit: (13) write('divide = ') ? creep
   Call: (13) write(7) ? creep
7
   Exit: (13) write(7) ? creep
   Call: (13) write(' | ') ? creep
 |
   Exit: (13) write(' | ') ? creep
   Call: (13) write([3, 1, 2, 5, 4]) ? creep
 [3,1,2,5,4]
   Exit: (13) write([3, 1, 2, 5, 4]) ? creep
   Call: (13) write([9, 8]) ? creep
[9,8]
   Exit: (13) write([9, 8]) ? creep
   Call: (13) nl ? creep

   Exit: (13) nl ? creep
   Exit: (12) writeDevide(7, [3, 1, 2, 5, 4], [9, 8]) ? creep
   Call: (12) qSortHelper([3, 1, 2, 5, 4], _24966) ? creep
   Call: (13) divide(3, [1, 2, 5, 4], _25012, _25014) ? creep
   Call: (14) 3=<1 ? creep
   Fail: (14) 3=<1 ? creep
   Redo: (13) divide(3, [1, 2, 5, 4], _25150, _25152) ? creep
   Call: (14) 3>1 ? creep
   Exit: (14) 3>1 ? creep
   Call: (14) divide(3, [2, 5, 4], _25140, _25290) ? creep
   Call: (15) 3=<2 ? creep
   Fail: (15) 3=<2 ? creep
   Redo: (14) divide(3, [2, 5, 4], _25140, _25428) ? creep
   Call: (15) 3>2 ? creep
   Exit: (15) 3>2 ? creep
   Call: (15) divide(3, [5, 4], _25416, _25566) ? creep
   Call: (16) 3=<5 ? creep
   Exit: (16) 3=<5 ? creep
   Call: (16) divide(3, [4], _25416, _25554) ? creep
   Call: (17) 3=<4 ? creep
   Exit: (17) 3=<4 ? creep
   Call: (17) divide(3, [], _25416, _25692) ? creep
   Exit: (17) divide(3, [], [], []) ? creep
   Exit: (16) divide(3, [4], [], [4]) ? creep
   Exit: (15) divide(3, [5, 4], [], [5, 4]) ? creep
   Exit: (14) divide(3, [2, 5, 4], [2], [5, 4]) ? creep
   Exit: (13) divide(3, [1, 2, 5, 4], [1, 2], [5, 4]) ? creep
   Call: (13) writeDevide(3, [1, 2], [5, 4]) ? creep
   Call: (14) write('divide = ') ? creep
divide =
   Exit: (14) write('divide = ') ? creep
   Call: (14) write(3) ? creep
3
   Exit: (14) write(3) ? creep
   Call: (14) write(' | ') ? creep
 |
   Exit: (14) write(' | ') ? creep
   Call: (14) write([1, 2]) ? creep
 [1,2]
```

```
   Exit: (14) write([1, 2]) ? creep
   Call: (14) write([5, 4]) ? creep
[5,4]
   Exit: (14) write([5, 4]) ? creep
   Call: (14) nl ? creep

   Exit: (14) nl ? creep
   Exit: (13) writeDevide(3, [1, 2], [5, 4]) ? creep
   Call: (13) qSortHelper([1, 2], _26718) ? creep
   Call: (14) divide(1, [2], _26764, _26766) ? creep
   Call: (15) 1=<2 ? creep
   Exit: (15) 1=<2 ? creep
   Call: (15) divide(1, [], _26902, _26754) ? creep
   Exit: (15) divide(1, [], [], []) ? creep
   Exit: (14) divide(1, [2], [], [2]) ? creep
   Call: (14) writeDevide(1, [], [2]) ? creep
   Call: (15) write('divide = ') ? creep
divide =
   Exit: (15) write('divide = ') ? creep
   Call: (15) write(1) ? creep
1
   Exit: (15) write(1) ? creep
   Call: (15) write(' | ') ? creep
 |
   Exit: (15) write(' | ') ? creep
   Call: (15) write([]) ? creep
[]
   Exit: (15) write([]) ? creep
   Call: (15) write([2]) ? creep
[2]
   Exit: (15) write([2]) ? creep
   Call: (15) nl ? creep

   Exit: (15) nl ? creep
   Exit: (14) writeDevide(1, [], [2]) ? creep
   Call: (14) qSortHelper([], _27648) ? creep
   Exit: (14) qSortHelper([], []) ? creep
   Call: (14) qSortHelper([2], _27736) ? creep
   Call: (15) divide(2, [], _27782, _27784) ? creep
   Exit: (15) divide(2, [], [], []) ? creep
   Call: (15) writeDevide(2, [], []) ? creep
   Exit: (15) writeDevide(2, [], []) ? creep
   Call: (15) qSortHelper([], _27956) ? creep
   Exit: (15) qSortHelper([], []) ? creep
   Call: (15) qSortHelper([], _28044) ? creep
   Exit: (15) qSortHelper([], []) ? creep
   Call: (15) backward_compatibility:merge([], [2], _28140) ? creep
   Exit: (15) backward_compatibility:merge([], [2], [2]) ? creep
   Call: (15) writeMerge([], 2, []) ? creep
   Call: (16) write('merge : ') ? creep
merge :
   Exit: (16) write('merge : ') ? creep
   Call: (16) list_empty([], _28358) ? creep
   Exit: (16) list_empty([], true) ? creep
   Call: (16) list_empty([], _28446) ? creep
   Exit: (16) list_empty([], true) ? creep
   Call: (16) true ? creep
   Exit: (16) true ? creep
   Call: (16) true ? creep
   Exit: (16) true ? creep
   Call: (16) write('[') ? creep
[
   Exit: (16) write('[') ? creep
   Call: (16) write(2) ? creep
```

```
2
   Exit: (16) write(2) ? creep
   Call: (16) write(']') ? creep
]
   Exit: (16) write(']') ? creep
   Call: (16) nl ? creep

   Exit: (16) nl ? creep
   Exit: (15) writeMerge([], 2, []) ? creep
   Exit: (14) qSortHelper([2], [2]) ? creep
   Call: (14) backward_compatibility:merge([], [1, 2], _29158) ? creep
   Exit: (14) backward_compatibility:merge([], [1, 2], [1, 2]) ? creep
   Call: (14) writeMerge([], 1, [2]) ? creep
   Call: (15) write('merge : ') ? creep
merge :
   Exit: (15) write('merge : ') ? creep
   Call: (15) list_empty ([], _29376) ? creep
   Exit: (15) list_empty ([], true) ? creep
   Call: (15) list_empty ([2], _29464) ? creep
   Exit: (15) list_empty ([2], false) ? creep
   Call: (15) true ? creep
   Exit: (15) true ? creep
   Call: (15) false ? creep
   Fail: (15) false ? creep
   Redo: (14) writeMerge([], 1, [2]) ? creep
   Call: (15) write ([]) ? creep
[]
   Exit: (15) write ([]) ? creep
   Call: (15) write('[') ? creep
[
   Exit: (15) write('[') ? creep
   Call: (15) write(1) ? creep
1
   Exit: (15) write(1) ? creep
   Call: (15) write(']') ? creep
]
   Exit: (15) write(']') ? creep
   Call: (15) write ([2]) ? creep
[2]
   Exit: (15) write ([2]) ? creep
   Call: (15) nl ? creep

   Exit: (15) nl ? creep
   Exit: (14) writeMerge([], 1, [2]) ? creep
   Exit: (13) qSortHelper([1, 2], [1, 2]) ? creep
   Call: (13) qSortHelper([5, 4], _30388) ? creep
   Call: (14) divide(5, [4], _30434, _30436) ? creep
   Call: (15) 5=<4 ? creep
   Fail: (15) 5=<4 ? creep
   Redo: (14) divide(5, [4], _30572, _30574) ? creep
   Call: (15) 5>4 ? creep
   Exit: (15) 5>4 ? creep
   Call: (15) divide(5, [], _30562, _30712) ? creep
   Exit: (15) divide(5, [], [], []) ? creep
   Exit: (14) divide(5, [4], [4], []) ? creep
   Call: (14) writeDevide(5, [4], []) ? creep
   Call: (15) write('divide = ') ? creep
divide =
   Exit: (15) write('divide = ') ? creep
   Call: (15) write(5) ? creep
5
   Exit: (15) write(5) ? creep
   Call: (15) write(' | ') ? creep
 |
```

```
   Exit: (15) write(' | ') ? creep
   Call: (15) write([4]) ? creep
[4]
   Exit: (15) write([4]) ? creep
   Call: (15) write([]) ? creep
[]
   Exit: (15) write([]) ? creep
   Call: (15) nl ? creep

   Exit: (15) nl ? creep
   Exit: (14) writeDevide(5, [4], []) ? creep
   Call: (14) qSortHelper([4], _31456) ? creep
   Call: (15) divide(4, [], _31502, _31504) ? creep
   Exit: (15) divide(4, [], [], []) ? creep
   Call: (15) writeDevide(4, [], []) ? creep
   Exit: (15) writeDevide(4, [], []) ? creep
   Call: (15) qSortHelper([], _31676) ? creep
   Exit: (15) qSortHelper([], []) ? creep
   Call: (15) qSortHelper([], _31764) ? creep
   Exit: (15) qSortHelper([], []) ? creep
   Call: (15) backward_compatibility:merge([], [4], _31860) ? creep
   Exit: (15) backward_compatibility:merge([], [4], [4]) ? creep
   Call: (15) writeMerge([], 4, []) ? creep
   Call: (16) write('merge : ') ? creep
merge :
   Exit: (16) write('merge : ') ? creep
   Call: (16) list_empty([], _32078) ? creep
   Exit: (16) list_empty([], true) ? creep
   Call: (16) list_empty([], _32166) ? creep
   Exit: (16) list_empty([], true) ? creep
   Call: (16) true ? creep
   Exit: (16) true ? creep
   Call: (16) true ? creep
   Exit: (16) true ? creep
   Call: (16) write('[') ? creep
[
   Exit: (16) write('[') ? creep
   Call: (16) write(4) ? creep
4
   Exit: (16) write(4) ? creep
   Call: (16) write(']') ? creep
]
   Exit: (16) write(']') ? creep
   Call: (16) nl ? creep

   Exit: (16) nl ? creep
   Exit: (15) writeMerge([], 4, []) ? creep
   Exit: (14) qSortHelper([4], [4]) ? creep
   Call: (14) qSortHelper([], _32870) ? creep
   Exit: (14) qSortHelper([], []) ? creep
   Call: (14) backward_compatibility:merge([4], [5], _32966) ? creep
   Call: (15) 4@=<5 ? creep
   Exit: (15) 4@=<5 ? creep
   Call: (15) _32954=4 ? creep
   Exit: (15) 4=4 ? creep
   Call: (15) backward_compatibility:merge([], [5], _32956) ? creep
   Exit: (15) backward_compatibility:merge([], [5], [5]) ? creep
   Exit: (14) backward_compatibility:merge([4], [5], [4, 5]) ? creep
   Call: (14) writeMerge([4], 5, []) ? creep
   Call: (15) write('merge : ') ? creep
merge :
   Exit: (15) write('merge : ') ? creep
   Call: (15) list_empty([4], _33460) ? creep
   Exit: (15) list_empty([4], false) ? creep
```

```
   Call: (15) list_empty ([], _33548) ? creep
   Exit: (15) list_empty ([], true) ? creep
   Call: (15) false ? creep
   Fail: (15) false ? creep
   Redo: (14) writeMerge([4], 5, []) ? creep
   Call: (15) write ([4]) ? creep
[4]
   Exit: (15) write ([4]) ? creep
   Call: (15) write('[') ? creep
[
   Exit: (15) write('[') ? creep
   Call: (15) write(5) ? creep
5
   Exit: (15) write(5) ? creep
   Call: (15) write(']') ? creep
]
   Exit: (15) write(']') ? creep
   Call: (15) write ([]) ? creep
[]
   Exit: (15) write ([]) ? creep
   Call: (15) nl ? creep

   Exit: (15) nl ? creep
   Exit: (14) writeMerge([4], 5, []) ? creep
   Exit: (13) qSortHelper([5, 4], [4, 5]) ? creep
   Call: (13) backward_compatibility:merge([1, 2], [3, 4, 5], _34392) ? creep
   Call: (14) 1@=<3 ? creep
   Exit: (14) 1@=<3 ? creep
   Call: (14) _34380=1 ? creep
   Exit: (14) 1=1 ? creep
   Call: (14) backward_compatibility:merge([2], [3, 4, 5], _34382) ? creep
   Call: (15) 2@=<3 ? creep
   Exit: (15) 2@=<3 ? creep
   Call: (15) _34612=2 ? creep
   Exit: (15) 2=2 ? creep
   Call: (15) backward_compatibility:merge([], [3, 4, 5], _34614) ? creep
   Exit: (15) backward_compatibility:merge([], [3, 4, 5], [3, 4, 5]) ? creep
   Exit: (14) backward_compatibility:merge([2], [3, 4, 5], [2, 3, 4, 5]) ? creep
   Exit: (13) backward_compatibility:merge([1, 2], [3, 4, 5], [1, 2, 3, 4, 5]) ? creep
   Call: (13) writeMerge([1, 2], 3, [4, 5]) ? creep
   Call: (14) write('merge : ') ? creep
merge :
   Exit: (14) write('merge : ') ? creep
   Call: (14) list_empty ([1, 2], _35162) ? creep
   Exit: (14) list_empty ([1, 2], false) ? creep
   Call: (14) list_empty ([4, 5], _35250) ? creep
   Exit: (14) list_empty ([4, 5], false) ? creep
   Call: (14) false ? creep
   Fail: (14) false ? creep
   Redo: (13) writeMerge([1, 2], 3, [4, 5]) ? creep
   Call: (14) write ([1, 2]) ? creep
[1,2]
   Exit: (14) write ([1, 2]) ? creep
   Call: (14) write('[') ? creep
[
   Exit: (14) write('[') ? creep
   Call: (14) write(3) ? creep
3
   Exit: (14) write(3) ? creep
   Call: (14) write(']') ? creep
]
   Exit: (14) write(']') ? creep
   Call: (14) write ([4, 5]) ? creep
[4,5]
```

```
   Exit: (14) write([4, 5]) ? creep
   Call: (14) nl ? creep

   Exit: (14) nl ? creep
   Exit: (13) writeMerge([1, 2], 3, [4, 5]) ? creep
   Exit: (12) qSortHelper([3, 1, 2, 5, 4], [1, 2, 3, 4, 5]) ? creep
   Call: (12) qSortHelper([9, 8], _36086) ? creep
   Call: (13) divide(9, [8], _36132, _36134) ? creep
   Call: (14) 9=<8 ? creep
   Fail: (14) 9=<8 ? creep
   Redo: (13) divide(9, [8], _36270, _36272) ? creep
   Call: (14) 9>8 ? creep
   Exit: (14) 9>8 ? creep
   Call: (14) divide(9, [], _36260, _36410) ? creep
   Exit: (14) divide(9, [], [], []) ? creep
   Exit: (13) divide(9, [8], [8], []) ? creep
   Call: (13) writeDevide(9, [8], []) ? creep
   Call: (14) write('divide = ') ? creep
divide =
   Exit: (14) write('divide = ') ? creep
   Call: (14) write(9) ? creep
9
   Exit: (14) write(9) ? creep
   Call: (14) write(' | ') ? creep
 |
   Exit: (14) write(' | ') ? creep
   Call: (14) write([8]) ? creep
[8]
   Exit: (14) write([8]) ? creep
   Call: (14) write([]) ? creep
[]
   Exit: (14) write([]) ? creep
   Call: (14) nl ? creep

   Exit: (14) nl ? creep
   Exit: (13) writeDevide(9, [8], []) ? creep
   Call: (13) qSortHelper([8], _37154) ? creep
   Call: (14) divide(8, [], _37200, _37202) ? creep
   Exit: (14) divide(8, [], [], []) ? creep
   Call: (14) writeDevide(8, [], []) ? creep
   Exit: (14) writeDevide(8, [], []) ? creep
   Call: (14) qSortHelper([], _37374) ? creep
   Exit: (14) qSortHelper([], []) ? creep
   Call: (14) qSortHelper([], _37462) ? creep
   Exit: (14) qSortHelper([], []) ? creep
   Call: (14) backward_compatibility:merge([], [8], _37558) ? creep
   Exit: (14) backward_compatibility:merge([], [8], [8]) ? creep
   Call: (14) writeMerge([], 8, []) ? creep
   Call: (15) write('merge : ') ? creep
merge :
   Exit: (15) write('merge : ') ? creep
   Call: (15) list_empty ([], _37776) ? creep
   Exit: (15) list_empty ([], true) ? creep
   Call: (15) list_empty ([], _37864) ? creep
   Exit: (15) list_empty ([], true) ? creep
   Call: (15) true ? creep
   Exit: (15) true ? creep
   Call: (15) true ? creep
   Exit: (15) true ? creep
   Call: (15) write('[') ? creep
[
   Exit: (15) write('[') ? creep
   Call: (15) write(8) ? creep
8
```

```
   Exit: (15) write(8) ? creep
   Call: (15) write(']') ? creep
]
   Exit: (15) write(']') ? creep
   Call: (15) nl ? creep

   Exit: (15) nl ? creep
   Exit: (14) writeMerge([], 8, []) ? creep
   Exit: (13) qSortHelper([8], [8]) ? creep
   Call: (13) qSortHelper([], _38568) ? creep
   Exit: (13) qSortHelper([], []) ? creep
   Call: (13) backward_compatibility:merge([8], [9], _38664) ? creep
   Call: (14) 8@=<9 ? creep
   Exit: (14) 8@=<9 ? creep
   Call: (14) _38652=8 ? creep
   Exit: (14) 8=8 ? creep
   Call: (14) backward_compatibility:merge([], [9], _38654) ? creep
   Exit: (14) backward_compatibility:merge([], [9], [9]) ? creep
   Exit: (13) backward_compatibility:merge([8], [9], [8, 9]) ? creep
   Call: (13) writeMerge([8], 9, []) ? creep
   Call: (14) write('merge : ') ? creep
merge :
   Exit: (14) write('merge : ') ? creep
   Call: (14) list_empty ([8], _39158) ? creep
   Exit: (14) list_empty ([8], false) ? creep
   Call: (14) list_empty ([], _39246) ? creep
   Exit: (14) list_empty ([], true) ? creep
   Call: (14) false ? creep
   Fail: (14) false ? creep
   Redo: (13) writeMerge([8], 9, []) ? creep
   Call: (14) write([8]) ? creep
[8]
   Exit: (14) write([8]) ? creep
   Call: (14) write('[') ? creep
[
   Exit: (14) write('[') ? creep
   Call: (14) write(9) ? creep
9
   Exit: (14) write(9) ? creep
   Call: (14) write(']') ? creep
]
   Exit: (14) write(']') ? creep
   Call: (14) write([]) ? creep
[]
   Exit: (14) write([]) ? creep
   Call: (14) nl ? creep

   Exit: (14) nl ? creep
   Exit: (13) writeMerge([8], 9, []) ? creep
   Exit: (12) qSortHelper([9, 8], [8, 9]) ? creep
   Call: (12) backward_compatibility:merge([1, 2, 3, 4, 5], [7, 8, 9], _40090) ? creep
   Call: (13) 1@=<7 ? creep
   Exit: (13) 1@=<7 ? creep
   Call: (13) _40078=1 ? creep
   Exit: (13) 1=1 ? creep
   Call: (13) backward_compatibility:merge([2, 3, 4, 5], [7, 8, 9], _40080) ? creep
   Call: (14) 2@=<7 ? creep
   Exit: (14) 2@=<7 ? creep
   Call: (14) _40310=2 ? creep
   Exit: (14) 2=2 ? creep
   Call: (14) backward_compatibility:merge([3, 4, 5], [7, 8, 9], _40312) ? creep
   Call: (15) 3@=<7 ? creep
   Exit: (15) 3@=<7 ? creep
   Call: (15) _40542=3 ? creep
```

```
   Exit: (15) 3=3 ? creep
   Call: (15) backward_compatibility:merge([4, 5], [7, 8, 9], _40544) ? creep
   Call: (16) 4@=<7 ? creep
   Exit: (16) 4@=<7 ? creep
   Call: (16) _40774=4 ? creep
   Exit: (16) 4=4 ? creep
   Call: (16) backward_compatibility:merge([5], [7, 8, 9], _40776) ? creep
   Call: (17) 5@=<7 ? creep
   Exit: (17) 5@=<7 ? creep
   Call: (17) _41006=5 ? creep
   Exit: (17) 5=5 ? creep
   Call: (17) backward_compatibility:merge([], [7, 8, 9], _41008) ? creep
   Exit: (17) backward_compatibility:merge([], [7, 8, 9], [7, 8, 9]) ? creep
   Exit: (16) backward_compatibility:merge([5], [7, 8, 9], [5, 7, 8, 9]) ? creep
   Exit: (15) backward_compatibility:merge([4, 5], [7, 8, 9], [4, 5, 7, 8, 9]) ? creep
   Exit: (14) backward_compatibility:merge([3, 4, 5], [7, 8, 9], [3, 4, 5, 7, 8, 9]) ? creep
   Exit: (13) backward_compatibility:merge([2, 3, 4, 5], [7, 8, 9], [2, 3, 4, 5, 7, 8, 9]) ? creep
   Exit: (12) backward_compatibility:merge([1, 2, 3, 4, 5], [7, 8, 9], [1, 2, 3, 4, 5, 7, 8|...]) ?
        creep
   Call: (12) writeMerge([1, 2, 3, 4, 5], 7, [8, 9]) ? creep
   Call: (13) write('merge : ') ? creep
merge :
   Exit: (13) write('merge : ') ? creep
   Call: (13) list_empty([1, 2, 3, 4, 5], _41688) ? creep
   Exit: (13) list_empty([1, 2, 3, 4, 5], false) ? creep
   Call: (13) list_empty([8, 9], _41776) ? creep
   Exit: (13) list_empty([8, 9], false) ? creep
   Call: (13) false ? creep
   Fail: (13) false ? creep
   Redo: (12) writeMerge([1, 2, 3, 4, 5], 7, [8, 9]) ? creep
   Call: (13) write([1, 2, 3, 4, 5]) ? creep
 [1,2,3,4,5]
   Exit: (13) write([1, 2, 3, 4, 5]) ? creep
   Call: (13) write('[') ? creep
[
   Exit: (13) write('[') ? creep
   Call: (13) write(7) ? creep
7
   Exit: (13) write(7) ? creep
   Call: (13) write(']') ? creep
]
   Exit: (13) write(']') ? creep
   Call: (13) write([8, 9]) ? creep
 [8,9]
   Exit: (13) write([8, 9]) ? creep
   Call: (13) nl ? creep

   Exit: (13) nl ? creep
   Exit: (12) writeMerge([1, 2, 3, 4, 5], 7, [8, 9]) ? creep
   Exit: (11) qSortHelper([7, 3, 1, 2, 9, 5, 4, 8], [1, 2, 3, 4, 5, 7, 8, 9]) ? creep
   Exit: (10) quickSort([7, 3, 1, 2, 9, 5, 4, 8]) ? creep
true .
```

3 shortest path.pl

shortest path 는 노드 간의 최단거리를 출력하는 문제이다. 먼저 edge() 를 통해 노드 간의 연결과 거리 fact 를 작성했다. 가령, edge(1,2,6)은 1번 노드와 2번 노드가 연결되어 있으며, 거리 6만큼 떨어져 있다는 의미이다. 그 다음, path()를 사용해서 노드 간의 연결 규칙을 정의했다.

path(X,Y,[X,Y],D)에서는 노드 X와 Y가 edge(X,Y,D)라는 fact 로 연결되어 있으면, 해당 경로를 [X,Y]라는 리스트로 변수로 만들고, 해당 경로의 거리를 D라는 변수로 만드는 규칙이다. path(X,Z,[X—W],D)에서는 노드 X와 Y가 edge(X,Y,D1)라는 fact 로 연결되어 있고, 노드 Y와 Z 는 W라는 리스트의 경로를 가지고, D2 거리만큼 떨어진 path로 쿼리되는 경우, 노드 X와 Z 는 [X—W]의 경로로 표현되고, X와 Z 사이의 거리는 D1 + D2 라는 의미이다.

sp(X,Y)를 통해 노드 X 와 Y 사이의 최단 경로를 쿼리하면, shortestPath() 규칙을 통해 X와 Y 사이의 최단 경로를 구하게 된다. shortestPath() 내부에서는 findall()을 사용해서 X 와 Y 사이의 가능한 모든 경로의 리스트를 구하게 된다. 가능한 모든 경로의 리스트는 경로의 거리 D 와 경로의 리스트 P 를 하나의 원소로 하는 리스트를 Set 이라는 리스트에 저장하게 된다.

그 다음, shortestPath()는 findall()로 찾은 가능한 모든 경로의 리스트에 sort 를 사용해서 거리가 가장 짧은 원소가 리스트의 가장 앞에 오는 Sorted 변수를 만든다. 그 다음, Sorted = [[MinD,MinP]—_]. 구문을 통해 Sorted 에서 가장 앞에 오는 원소의 MinD 와 MinP 를 shortestPath 의 입력인 MinP 와 MinD 에 대입하여 최단 경로와 거리를 도출하였다.

그런데 만약, 노드의 X 값이 Y값보다 큰 값인 경우, sp() 내부에서 Y에서 X 로 가는 경로를 도출한 다음 경로를 역순으로 하여 출력하도록 했다. 노드들은 단방향이 아닌 양방향으로 연결되어 있기 때문에 X 에서 Y로 가는 최단 경로와 Y에서 X로 가는 최단 경로가 같기 때문이다.

```
edge(1,2,6).
edge(1,3,3).
edge(2,3,2).
edge(2,4,5).
edge(3,4,3).
edge(3,5,4).
edge(4,5,2).
edge(4,6,3).
edge(5,6,5).

path(X,Y,[X,Y],D):-
    edge(X,Y,D).

path(X,Z,[X|W],D):-
    edge(X,Y,D1),
    path(Y,Z,W,D2),
    D is D1 + D2.

shortestPath(X,X,[X,X],0):- !.
shortestPath(X,Y,MinP,MinD):-
    findall([D,P],path(X,Y,P,D),Set),
    getMin(Set,MinD,MinP).

getMin(Set,MinD,MinP):-
    sort(Set,Sorted),
    Sorted = [[MinD,MinP]|_].

sp(X,Y):-
    X < Y ->
    shortestPath(X,Y,Path,Distance),
    write(Path), nl,
    write(Distance);
    shortestPath(Y,X,Path,Distance),
    reverse(Path,ReversedPath),
    write(ReversedPath), nl,
    write(Distance).
```

3.1 trace 실행결과

sp(1,3)을 trace 에서 실행한 결과이다.

```
[trace]  ?- sp(1,3).
   Call: (10) sp(1, 3) ? creep
   Call: (11) 1<3 ? creep
   Exit: (11) 1<3 ? creep
   Call: (11) shortestPath(1, 3, _5392, _5394) ? creep
^  Call: (12) findall([_5380, _5386], path(1, 3, _5386, _5380), _5458) ? creep
   Call: (17) path(1, 3, _5386, _5380) ? creep
   Call: (18) edge(1, 3, _5380) ? creep
   Exit: (18) edge(1, 3, 3) ? creep
   Exit: (17) path(1, 3, [1, 3], 3) ? creep
   Redo: (17) path(1, 3, _5386, _5380) ? creep
   Call: (18) edge(1, _5772, _5774) ? creep
   Exit: (18) edge(1, 2, 6) ? creep
   Call: (18) path(2, 3, _5714, _5864) ? creep
   Call: (19) edge(2, 3, _5918) ? creep
   Exit: (19) edge(2, 3, 2) ? creep
   Exit: (18) path(2, 3, [2, 3], 2) ? creep
   Call: (18) _5380 is 6+2 ? creep
   Exit: (18) 8 is 6+2 ? creep
   Exit: (17) path(1, 3, [1, 2, 3], 8) ? creep
   Redo: (19) edge(2, 3, _6188) ? creep
   Fail: (19) edge(2, 3, _6232) ? creep
   Redo: (18) path(2, 3, _5714, _6278) ? creep
   Call: (19) edge(2, _6324, _6326) ? creep
   Exit: (19) edge(2, 3, 2) ? creep
   Call: (19) path(3, 3, _6266, _6416) ? creep
   Call: (20) edge(3, 3, _6470) ? creep
   Fail: (20) edge(3, 3, _6514) ? creep
   Redo: (19) path(3, 3, _6266, _6560) ? creep
   Call: (20) edge(3, _6606, _6608) ? creep
   Exit: (20) edge(3, 4, 3) ? creep
   Call: (20) path(4, 3, _6548, _6698) ? creep
   Call: (21) edge(4, 3, _6752) ? creep
   Fail: (21) edge(4, 3, _6796) ? creep
   Redo: (20) path(4, 3, _6548, _6842) ? creep
   Call: (21) edge(4, _6888, _6890) ? creep
   Exit: (21) edge(4, 5, 2) ? creep
   Call: (21) path(5, 3, _6830, _6980) ? creep
   Call: (22) edge(5, 3, _7034) ? creep
   Fail: (22) edge(5, 3, _7078) ? creep
   Redo: (21) path(5, 3, _6830, _7124) ? creep
   Call: (22) edge(5, _7170, _7172) ? creep
   Exit: (22) edge(5, 6, 5) ? creep
   Call: (22) path(6, 3, _7112, _7262) ? creep
   Call: (23) edge(6, 3, _7316) ? creep
   Fail: (23) edge(6, 3, _7360) ? creep
   Redo: (22) path(6, 3, _7112, _7406) ? creep
   Call: (23) edge(6, _7452, _7454) ? creep
   Fail: (23) edge(6, _7496, _7498) ? creep
   Fail: (22) path(6, 3, _7112, _7544) ? creep
   Fail: (21) path(5, 3, _6830, _7588) ? creep
   Redo: (21) edge(4, _7628, _7630) ? creep
   Exit: (21) edge(4, 6, 3) ? creep
   Call: (21) path(6, 3, _6830, _7720) ? creep
   Call: (22) edge(6, 3, _7774) ? creep
   Fail: (22) edge(6, 3, _7818) ? creep
   Redo: (21) path(6, 3, _6830, _7864) ? creep
   Call: (22) edge(6, _7910, _7912) ? creep
   Fail: (22) edge(6, _7954, _7956) ? creep
   Fail: (21) path(6, 3, _6830, _8002) ? creep
```

Fail: (20) path(4, 3, _6548, _8046) ? creep
Redo: (20) edge(3, _8086, _8088) ? creep
Exit: (20) edge(3, 5, 4) ? creep
Call: (20) path(5, 3, _6548, _8178) ? creep
Call: (21) edge(5, 3, _8232) ? creep
Fail: (21) edge(5, 3, _8276) ? creep
Redo: (20) path(5, 3, _6548, _8322) ? creep
Call: (21) edge(5, _8368, _8370) ? creep
Exit: (21) edge(5, 6, 5) ? creep
Call: (21) path(6, 3, _8310, _8460) ? creep
Call: (22) edge(6, 3, _8514) ? creep
Fail: (22) edge(6, 3, _8558) ? creep
Redo: (21) path(6, 3, _8310, _8604) ? creep
Call: (22) edge(6, _8650, _8652) ? creep
Fail: (22) edge(6, _8694, _8696) ? creep
Fail: (21) path(6, 3, _8310, _8742) ? creep
Fail: (20) path(5, 3, _6548, _8786) ? creep
Fail: (19) path(3, 3, _6266, _8830) ? creep
Redo: (19) edge(2, _8870, _8872) ? creep
Exit: (19) edge(2, 4, 5) ? creep
Call: (19) path(4, 3, _6266, _8962) ? creep
Call: (20) edge(4, 3, _9016) ? creep
Fail: (20) edge(4, 3, _9060) ? creep
Redo: (19) path(4, 3, _6266, _9106) ? creep
Call: (20) edge(4, _9152, _9154) ? creep
Exit: (20) edge(4, 5, 2) ? creep
Call: (20) path(5, 3, _9094, _9244) ? creep
Call: (21) edge(5, 3, _9298) ? creep
Fail: (21) edge(5, 3, _9342) ? creep
Redo: (20) path(5, 3, _9094, _9388) ? creep
Call: (21) edge(5, _9434, _9436) ? creep
Exit: (21) edge(5, 6, 5) ? creep
Call: (21) path(6, 3, _9376, _9526) ? creep
Call: (22) edge(6, 3, _9580) ? creep
Fail: (22) edge(6, 3, _9624) ? creep
Redo: (21) path(6, 3, _9376, _9670) ? creep
Call: (22) edge(6, _9716, _9718) ? creep
Fail: (22) edge(6, _9760, _9762) ? creep
Fail: (21) path(6, 3, _9376, _9808) ? creep
Fail: (20) path(5, 3, _9094, _9852) ? creep
Redo: (20) edge(4, _9892, _9894) ? creep
Exit: (20) edge(4, 6, 3) ? creep
Call: (20) path(6, 3, _9094, _9984) ? creep
Call: (21) edge(6, 3, _10038) ? creep
Fail: (21) edge(6, 3, _10082) ? creep
Redo: (20) path(6, 3, _9094, _10128) ? creep
Call: (21) edge(6, _10174, _10176) ? creep
Fail: (21) edge(6, _10218, _10220) ? creep
Fail: (20) path(6, 3, _9094, _10266) ? creep
Fail: (19) path(4, 3, _6266, _10310) ? creep
Fail: (18) path(2, 3, _5714, _10354) ? creep
Redo: (18) edge(1, _10394, _10396) ? creep
Exit: (18) edge(1, 3, 3) ? creep
Call: (18) path(3, 3, _5714, _10486) ? creep
Call: (19) edge(3, 3, _10540) ? creep
Fail: (19) edge(3, 3, _10584) ? creep
Redo: (18) path(3, 3, _5714, _10630) ? creep
Call: (19) edge(3, _10676, _10678) ? creep
Exit: (19) edge(3, 4, 3) ? creep
Call: (19) path(4, 3, _10618, _10768) ? creep
Call: (20) edge(4, 3, _10822) ? creep
Fail: (20) edge(4, 3, _10866) ? creep
Redo: (19) path(4, 3, _10618, _10912) ? creep
Call: (20) edge(4, _10958, _10960) ? creep

```
   Exit: (20) edge(4, 5, 2) ? creep
   Call: (20) path(5, 3, _10900, _11050) ? creep
   Call: (21) edge(5, 3, _11104) ? creep
   Fail: (21) edge(5, 3, _11148) ? creep
   Redo: (20) path(5, 3, _10900, _11194) ? creep
   Call: (21) edge(5, _11240, _11242) ? creep
   Exit: (21) edge(5, 6, 5) ? creep
   Call: (21) path(6, 3, _11182, _11332) ? creep
   Call: (22) edge(6, 3, _11386) ? creep
   Fail: (22) edge(6, 3, _11430) ? creep
   Redo: (21) path(6, 3, _11182, _11476) ? creep
   Call: (22) edge(6, _11522, _11524) ? creep
   Fail: (22) edge(6, _11566, _11568) ? creep
   Fail: (21) path(6, 3, _11182, _11614) ? creep
   Fail: (20) path(5, 3, _10900, _11658) ? creep
   Redo: (20) edge(4, _11698, _11700) ? creep
   Exit: (20) edge(4, 6, 3) ? creep
   Call: (20) path(6, 3, _10900, _11790) ? creep
   Call: (21) edge(6, 3, _11844) ? creep
   Fail: (21) edge(6, 3, _11888) ? creep
   Redo: (20) path(6, 3, _10900, _11934) ? creep
   Call: (21) edge(6, _11980, _11982) ? creep
   Fail: (21) edge(6, _12024, _12026) ? creep
   Fail: (20) path(6, 3, _10900, _12072) ? creep
   Fail: (19) path(4, 3, _10618, _12116) ? creep
   Redo: (19) edge(3, _12156, _12158) ? creep
   Exit: (19) edge(3, 5, 4) ? creep
   Call: (19) path(5, 3, _10618, _12248) ? creep
   Call: (20) edge(5, 3, _12302) ? creep
   Fail: (20) edge(5, 3, _12346) ? creep
   Redo: (19) path(5, 3, _10618, _12392) ? creep
   Call: (20) edge(5, _12438, _12440) ? creep
   Exit: (20) edge(5, 6, 5) ? creep
   Call: (20) path(6, 3, _12380, _12530) ? creep
   Call: (21) edge(6, 3, _12584) ? creep
   Fail: (21) edge(6, 3, _12628) ? creep
   Redo: (20) path(6, 3, _12380, _12674) ? creep
   Call: (21) edge(6, _12720, _12722) ? creep
   Fail: (21) edge(6, _12764, _12766) ? creep
   Fail: (20) path(6, 3, _12380, _12812) ? creep
   Fail: (19) path(5, 3, _10618, _12856) ? creep
   Fail: (18) path(3, 3, _5714, _12900) ? creep
   Fail: (17) path(1, 3, _5386, _5380) ? creep
^  Exit: (12) findall([_5380, _5386], user:path(1, 3, _5386, _5380), [[3, [1, 3]], [8, [1, 2, 3]]]) ?
      creep
   Call: (12) getMin([[3, [1, 3]], [8, [1, 2, 3]]], _13094, _13096) ? creep
   Call: (13) sort([[3, [1, 3]], [8, [1, 2, 3]]], _13138) ? creep
   Exit: (13) sort([[3, [1, 3]], [8, [1, 2, 3]]], [[3, [1, 3]], [8, [1, 2, 3]]]) ? creep
   Call: (13) [[3, [1, 3]], [8, [1, 2, 3]]]=[[_13190, _13196]|_13186] ? creep
   Exit: (13) [[3, [1, 3]], [8, [1, 2, 3]]]=[[3, [1, 3]], [8, [1, 2, 3]]] ? creep
   Exit: (12) getMin([[3, [1, 3]], [8, [1, 2, 3]]], 3, [1, 3]) ? creep
   Exit: (11) shortestPath(1, 3, [1, 3], 3) ? creep
   Call: (11) write([1, 3]) ? creep
[1,3]
   Exit: (11) write([1, 3]) ? creep
   Call: (11) nl ? creep

   Exit: (11) nl ? creep
   Call: (11) write(3) ? creep
3
   Exit: (11) write(3) ? creep
   Exit: (10) sp(1, 3) ? creep
true.
```
