Department of Computer Science
Technical University of Cluj-Napoca

Artificial Intelligence

Laboratory activity

Name: Cristea Tudor Iosif
Group: 30433
Email: cristeatudor6@gmail.com

Teaching Assistant: Adrian Groza
Adrian.Groza@cs.utcluj.ro

Contents

Table 1: Lab scheduling

Activity	**Deadline**
Searching agents, Linux, Latex, Python, Pacman	W_1
Uninformed search	W_2
Informed Search	W_3
Adversarial search	W_4
Propositional logic	W_5
First order logic	W_6
Inference in first order logic	W_7
Knowledge representation in first order logic	W_8
Classical planning	W_9
Contingent, conformant and probabilistic planning	W_{10}
Multi-agent planing	W_{11}
Modelling planning domains	W_{12}
Planning with event calculus	W_{14}

Lab organisation.

1. Laboratory work is 25% from the final grade.
2. There are three deliverables in total: 1. Search, 2. Logic, 3. Planning.
3. Before each deadline, you have to send your work (latex documentation/code) at moodle.cs.utcluj.ro
4. We use Linux and Latex
5. Plagiarism: Don't be a cheater! Cheating affects your colleagues, scholarships and a lot more.

Chapter 1

A1: Search

In this chapter, the work/code is done on a modified version of the *Pac-Man* game. This alternative version was developed in Python and for Linux based operating systems by the University of California, Berkeley.

1.1 Search Agent Problem

The first problem that needs to be solved is the so-called *"Search Agent"* problem which consists of a starting position from which the *Pac-Man* character begins the search and an ending position which needs to be reached. Multiple searching algorithms are implemented and presented below as solutions to this problem.

1.1.1 Random Search

Firstly, a simple, but naïve solution is to choose a random position from one of the four available successors. Evidently, this makes the number of steps taken to reach the *ending position* completely unpredictable, even if the map remains unchanged.

1.1.2 Depth First Search

Secondly, an intuitive solution is inspired from graphs and that is the *Depth First Search* algorithm. For this algorithm, a separate class for representing the nodes that will be *pushed* and *popped* from a *stack* is used, since a Last-In-First-Out approach needs to be adopted. Starting from the last node which was popped from the stack (which also contains the ending position), the "resulting graph" is traversed until the root (which has no parent) of the graph is reached. While doing so, every node's corresponding move is inserted into the list of moves. However, since the moves are needed in the opposite order, the list of moves is reversed. This solution is significantly better than the first one, when it comes to the execution time.

1.1.3 Breadth First Search

Another solution inspired from graphs is the other well-known searching algorithm, namely *Breadth First Search*. The solution is almost identical to the last one with the simple modification of the order in which the nodes are extracted from the array, which actually is now a queue (a First-In-First-Out approach needs to be adopted). Basically, we don't extract the last element inserted but the first. In the restricted set of test cases, *BFS* performed equally or better than *DFS*.

1.1.4 Iterative Deepening Search

A modified version of the *BFS* algorithm can also be used to solve this problem, namely the *Iterative Deepening Search* algorithm, which uses a *depth* parameter in order to perform numerous *DFS* searches at different depths, until the first good one is reached. Basically, this parameter tells the algorithm how many nodes should the path have. The algorithm incorporates the advantages of the two search algorithms, by combining the speed and the use of less memory of the *DFS* algorithm while being an optimal algorithm thanks to the optimality property of the *BFS* algorithm.

1.1.5 Uniform Cost Search

The last uninformed search algorithm is the *Uniform Cost Search* algorithm which is inspired by *Dijkstra's Algorithm*. It implies the usage of a priority queue in which a modified version of the *Node* class is used, the only modification being that the cost of the path from the start and up to that node is also stored. Then, for every node that is not yet visited, we append to the priority queue, the successors of the node and continue the search by selecting the node that has the smallest cost associated to it, until we reach the goal.

1.1.6 A* Search

Moving on to solutions that are commonly known in the vast domain of AI, the first solution for this problem comes in the form of the *A* Search* algorithm, an informed search algorithm. This is a modified version of the *Uniform Cost Search* algorithm, in which a heuristic is added to every node, so $f(n) = g(n) + h(n)$, where g is the cost of the path from the start node to the current node and h represents the heuristic from the current node to the goal node. This heuristic is a number which represents an estimate of the cost that it takes the agent to reach the goal from that node, and for it to be a good/admissible heuristic, it has to be an optimistic one, meaning that it has to underestimate the real cost of getting from the node to the goal. Therefore, the algorithm keeps track of two different costs, one without the heuristic added and one with the heuristic added and the priority queue will use the latter. The rest of the algorithm is similar to the previous one.

Heuristics

The heuristics that can be used for this algorithm (and for some of the following ones) are the *Manhattan distance* and *Euclidean distance*. However, I have added two more, namely the *Chebyshev distance* and the *Octile distance*. These last two expand less nodes than the first two. Moreover, I also implemented a modified version of all four heuristics starting from the idea of comparing the heuristic from the current node to the goal node with the heuristic from the current node to the starting node and taking the minimum value. Unfortunately, these modified versions of the heuristics are less efficient than their unmodified counterparts.

1.1.7 Weighted A* Search

A modified version of the *A* Search* algorithm is the *Weighted A* Search* algorithm, which multiplies every node's cost by a weight and the heuristic value by another weight in order to make the algorithm run a little faster (works only in some cases). These weights have the purpose of putting emphasis on $g(n)$ or $h(n)$, respectively. They can also be dynamic in the sense that, throughout a running instance of the program, they can change their values depending on the distance from the current state to the goal state. Thus, the previous formula

for f becomes: *f(n) = w1 * g(n) + w2 * h(n).* Other than this, the algorithm works based on the same principles.

1.1.8 BEAM Search

Another version of the *A* Search* algorithm is the *BEAM Search* algorithm which restricts the size of the priority queue to be at most a certain constant, β, and so, the number of choices for the next node is somewhat pruned. The main advantage of this algorithm is the fact that it uses less memory, but the main disadvantage is the fact that it is not a complete algorithm due to the fact that the global optimal choice could be pruned without being expanded. Therefore, the value for the β parameter has to be chosen carefully and specifically for the problem that needs to be solved. In my own testing on the *Pac-Man* search problem, a value of $\beta = 6$ is good for the small and medium mazes, but for the tiny maze a value of $\beta = 4$ is more appropriate whereas for the big maze a value of $\beta = 11$ is good enough (the heuristic used for all of these values is the null heuristic).

1.1.9 Hill Climbing Search

Another searching algorithm that can be implemented for this problem is the *"Hill Climbing"* algorithm, which uses a fitness function in order to select the best possible successor. Unfortunately, in our *Pac-Man* game, since all the successors have the same fitness function value, the algorithm is not sufficient, and therefore, incomplete. However, in order to make it complete, I implemented one of its variants, namely the *"Random Restart Hill Climbing"*, that simply chooses randomly from the grid (excluding the previous starting positions, the goal position and wall positions), a new starting position from which the algorithm restarts and tries to find the goal. This version of the algorithm is very inefficient, but it is complete. Another aspect that is worth mentioning is the fact that due to the current settings of the game, even though the path to the goal is found successfully and the list of moves is returned, these moves will not be rendered visually since the problem has a starting state which contains a starting position that cannot be changed (I tried to change it, but I was unsuccessful).

1.1.10 Genetic Algorithm/Programming Search

The last searching algorithm that I chose to implement is based on *Genetic Algorithms*. In the beginning of the algorithm, a finite number of valid paths (6 in this case) are randomly generated. Then, for a certain number of generations, the following actions are performed on the population:

- for every population, the fitness score is computed; the scores are computed using the octile distance between each position in the path list with the position of the goal

- using random members from the current population, an attempt at crossovering them is made, and if successful, a child is created

- for the crossover procedure, the middle index of the list of coordinates of parent1 and the middle index of the list of coordinates of parent2 are considered and the coordinates found at those indexes are compared; if they are equal, then the new path is constructed by concatenating the first half of parent1 with the second half of parent2 (or vice-versa if we are at the second batch); if they are not equal, then a potential common neighbour is searched; if this fails too, then the first middle index is decremented and the second one is incremented and the last two steps are repeated; when a common neighbour is found, than the two subpaths are concatenated and in between them, the common neighbour is inserted

- afterwards, a mutation is applied on the child and its fitness score is generated; the mutation basically consists of a small improvement that is applied on the child's path by

removing one redundant repetition of coordinates that occurs (e.g.: $(1,2) \rightarrow (1,3) \rightarrow (1,2)$, in this example (1,3) and (1,2) are deleted from the list)

- if the child's fitness score is better (smaller) than the "best parent's" score, then the search is stopped and the child is returned from the function; otherwise, the search continues until a suitable child is eventually found; if a suitable child cannot be obtained no matter what, then those parents cannot create a child, and so the process continues with a new pair of parents

- in the end, the path has to be created in order to display it inside the game; for this purpose, the list of population members is ordered ascendingly by the fitness score, so that the first member (the best one) is selected to be displayed

1.1.11 Comparing The Performances Of All Algorithms (Four Directions: N, S, W, E)

In the following three tables (one for each of the three maps: small, medium and big), all of the performances of the search algorithms that were implemented are compared in terms of Nodes Expanded (N.E.), Cost and Time. For the genetic algorithm, only the results for the small map were put in the table, but instead of randomly generating the paths as part of the algorithm, they were randomly pregenerated, but the same ones were used as population members for the genetic algorithm.

(C.W. = Cost Weight; H.W. = Heuristic Weight; B.W. = Beam Width)

Algorithm	Heuristic/Generations	C.W.	H.W.	B.W.	N.E.	Cost	Time
Random	-	-	-	-	1788	1787	0.0
DFS	-	-	-	-	149	57	0.0
BFS	-	-	-	-	186	19	0.0
IDS	-	-	-	-	11027	57	0.9
UCS	-	-	-	-	186	19	0.0
A*	manhattan	-	-	-	76	19	0.0
A*	euclidean	-	-	-	100	19	0.0
A*	chebyshev	-	-	-	102	19	0.0
A*	octile	-	-	-	59	19	0.0
A*	bidirectionalManhattan	-	-	-	127	19	0.0
A*	bidirectionalEuclidean	-	-	-	135	19	0.0
A*	bidirectionalChebyshev	-	-	-	142	19	0.0
A*	bidirectionalOctile	-	-	-	116	19	0.0
Weighted A*	octile	1	2	-	69	19	0.0
Weighted A*	octile	2	1	-	132	19	0.0
Weighted A*	octile	1	3	-	71	19	0.0
Weighted A*	octile	3	1	-	149	19	0.0
Weighted A*	octile	2	3	-	67	19	0.0
Weighted A*	octile	3	2	-	112	19	0.0
BEAM	octile	-	-	6	54	37	0.0
BEAM	octile	-	-	8	54	35	0.0
BEAM	octile	-	-	10	62	33	0.0
Hill Climb	-	-	-	-	367	999999	0.0
Genetic	5	-	-	-	35764	295	0.1
Genetic	10	-	-	-	36506	289	0.2
Genetic	50	-	-	-	32072	113	0.2
Genetic	100	-	-	-	26806	213	0.1
Genetic	500	-	-	-	26790	119	0.1

Table 1.1: For small map

Algorithm	**Heuristic**	**C.W.**	**H.W.**	**B.W.**	**N.E.**	**Cost**	**Time**
Random	-	-	-	-	24785	24784	0.3
DFS	-	-	-	-	551	148	0.2
BFS	-	-	-	-	548	68	0.2
IDS	-	-	-	-	151526	148	36.4
UCS	-	-	-	-	548	68	0.1
A*	manhattan	-	-	-	414	68	0.1
A*	euclidean	-	-	-	429	68	0.1
A*	chebyshev	-	-	-	435	68	0.1
A*	octile	-	-	-	419	19	0.0
A*	bidirectionalManhattan	-	-	-	449	68	0.2
A*	bidirectionalEuclidean	-	-	-	459	68	0.2
A*	bidirectionalChebyshev	-	-	-	468	68	0.2
A*	bidirectionalOctile	-	-	-	435	19	0.2
Weighted A*	octile	1	2	-	380	68	0.1
Weighted A*	octile	2	1	-	490	68	0.2
Weighted A*	octile	1	3	-	111	74	0.0
Weighted A*	octile	3	1	-	498	68	0.2
Weighted A*	octile	2	3	-	438	68	0.2
Weighted A*	octile	3	2	-	478	68	0.2
BEAM	octile	-	-	11	302	152	0.1
BEAM	octile	-	-	15	306	152	0.1
BEAM	octile	-	-	19	360	134	0.1
Hill Climb	-	-	-	-	3314	999999	0.1

Table 1.2: For medium map

Algorithm	Heuristic	C.W.	H.W.	B.W.	N.E.	Cost	Time
Random	-	-	-	-	105937	105936	1.2
DFS	-	-	-	-	1029	312	0.7
BFS	-	-	-	-	1237	210	1.0
IDS	-	-	-	-	528907	312	245.9
UCS	-	-	-	-	1237	210	0.3
A*	manhattan	-	-	-	1077	210	0.8
A*	euclidean	-	-	-	1101	210	0.8
A*	chebyshev	-	-	-	1146	210	0.9
A*	octile	-	-	-	1065	210	0.8
A*	bidirectionalManhattan	-	-	-	1077	210	0.8
A*	bidirectionalEuclidean	-	-	-	1101	210	0.8
A*	bidirectionalChebyshev	-	-	-	1146	210	0.9
A*	bidirectionalOctile	-	-	-	1067	210	0.8
Weighted A*	octile	1	2	-	950	210	0.7
Weighted A*	octile	2	1	-	1153	210	0.9
Weighted A*	octile	1	3	-	910	210	0.6
Weighted A*	octile	3	1	-	1207	210	1.0
Weighted A*	octile	2	3	-	959	210	0.7
Weighted A*	octile	3	2	-	1121	210	0.9
BEAM	octile	-	-	11	1203	212	1.0
BEAM	octile	-	-	16	1234	210	1.0
BEAM	octile	-	-	21	1238	210	1.1
Hill Climb	-	-	-	-	5628	999999	0.1

Table 1.3: For big map

1.1.12 Comparing The Performances Of All Algorithms (Eight Directions: N, S, W, E, NE, NW, SE, SW)

In order for *Pac-Man* to be able to go on diagonal directions as well, the following modifications have to be made to the game:

Listing 1.1: In the *game.py* file

```
class Directions:
    NORTH = 'North'
    SOUTH = 'South'
    EAST = 'East'
    WEST = 'West'
    NORTH_WEST = 'North-West'
    NORTH_EAST = 'North-East'
    SOUTH_WEST = 'South-West'
    SOUTH_EAST = 'South-East'
    STOP = 'Stop'
    ...
    ...
    class Actions:
    """A collection of static methods for manipulating move
       actions."""
```

```
    # Directions
    _directions = {Directions.NORTH:       (0, 1),
                   Directions.SOUTH:       (0, -1),
                   Directions.EAST:        (1, 0),
                   Directions.WEST:        (-1, 0),
                   Directions.NORTH_WEST:  (-1, 1),
                   Directions.NORTH_EAST:  (1, 1),
                   Directions.SOUTH_WEST:  (-1, -1),
                   Directions.SOUTH_EAST:  (1, -1),
                   Directions.STOP:        (0, 0)}
    ...
    ...
    def vectorToDirection(vector):
        dx, dy = vector
        if dy > 0 and dx < 0:
            return Directions.NORTH_WEST
        if dy > 0 and dx > 0:
            return Directions.NORTH_EAST
        if dy < 0 and dx < 0:
            return Directions.SOUTH_WEST
        if dy < 0 and dx > 0:
            return Directions.SOUTH_EAST
        if dy > 0:
            return Directions.NORTH
        if dy < 0:
            return Directions.SOUTH
        if dx < 0:
            return Directions.WEST
        if dx > 0:
            return Directions.EAST
        return Directions.STOP
```

Listing 1.2: In the *searchAgents.py* file

```
    def getSuccessors2(self, state):
        successors = []
        for action in [Directions.NORTH, Directions.SOUTH,
           Directions.EAST, Directions.WEST, Directions.
           NORTH_EAST, Directions.NORTH_WEST, Directions.
           SOUTH_EAST, Directions.SOUTH_WEST]:
            x,y = state
            dx, dy = Actions.directionToVector(action)
            nextx, nexty = int(x + dx), int(y + dy)
            if not self.walls[nextx][nexty]:
                nextState = (nextx, nexty)
                cost = self.costFn(nextState)
                successors.append( ( nextState, action, cost) )

        # Bookkeeping for display purposes
        self._expanded += 1 # DO NOT CHANGE
        if state not in self._visited:
```

```
            self._visited[state] = True
            self._visitedlist.append(state)

        return successors
```

Below, another three tables, which compare the performance metrics of the implemented algorithms, are presented. Their format is the same as before, with the exception that the genetic algorithm is not part of the comparison in this case.

Algorithm	**Heuristic**	**C.W.**	**H.W.**	**B.W.**	**N.E.**	**Cost**	**Time**
Random	-	-	-	-	1242	1241	0.0
DFS	-	-	-	-	217	109	0.1
BFS	-	-	-	-	260	15	0.1
IDS	-	-	-	-	23437	109	3.1
UCS	-	-	-	-	260	15	0.0
A*	manhattan	-	-	-	72	15	0.0
A*	euclidean	-	-	-	82	15	0.0
A*	chebyshev	-	-	-	109	15	0.0
A*	octile	-	-	-	68	15	0.0
A*	bidirectionalManhattan	-	-	-	171	15	0.0
A*	bidirectionalEuclidean	-	-	-	179	15	0.0
A*	bidirectionalChebyshev	-	-	-	192	15	0.0
A*	bidirectionalOctile	-	-	-	141	15	0.0
Weighted A*	octile	1	2	-	77	15	0.0
Weighted A*	octile	2	1	-	165	15	0.0
Weighted A*	octile	1	3	-	74	23	0.0
Weighted A*	octile	3	1	-	210	15	0.0
Weighted A*	octile	2	3	-	75	15	0.0
Weighted A*	octile	3	2	-	93	15	0.0
BEAM	octile	-	-	6	72	28	0.0
BEAM	octile	-	-	8	64	24	0.0
BEAM	octile	-	-	10	64	23	0.0
Hill Climb	-	-	-	-	779	999999	0.0

Table 1.4: For small map

Algorithm	**Heuristic**	**C.W.**	**H.W.**	**B.W.**	**N.E.**	**Cost**	**Time**
Random	-	-	-	-	4755	4754	0.1
DFS	-	-	-	-	719	210	0.5
BFS	-	-	-	-	728	58	0.5
IDS	-	-	-	-	258122	210	108.6
UCS	-	-	-	-	728	58	0.2
A*	manhattan	-	-	-	548	59	0.1
A*	euclidean	-	-	-	545	58	0.1
A*	chebyshev	-	-	-	562	58	0.1
A*	octile	-	-	-	529	59	0.1
A*	bidirectionalManhattan	-	-	-	576	58	0.1
A*	bidirectionalEuclidean	-	-	-	588	58	0.1
A*	bidirectionalChebyshev	-	-	-	604	58	0.1
A*	bidirectionalOctile	-	-	-	560	58	0.1
Weighted A*	octile	1	2	-	110	59	0.0
Weighted A*	octile	2	1	-	641	58	0.1
Weighted A*	octile	1	3	-	110	59	0.0
Weighted A*	octile	3	1	-	660	58	0.1
Weighted A*	octile	2	3	-	116	59	0.0
Weighted A*	octile	3	2	-	620	58	0.1
BEAM	octile	-	-	11	124	60	0.0
BEAM	octile	-	-	15	131	60	0.0
BEAM	octile	-	-	34	247	80	0.0
Hill Climb	-	-	-	-	1554	999999	0.0

Table 1.5: For medium map

Algorithm	Heuristic	C.W.	H.W.	B.W.	N.E.	Cost	Time
Random	-	-	-	-	88007	88006	1.1
DFS	-	-	-	-	1511	668	2.3
BFS	-	-	-	-	1763	156	3.3
IDS	-	-	-	-	1140806	668	1087.0
UCS	-	-	-	-	1763	156	0.9
A*	manhattan	-	-	-	1364	156	0.6
A*	euclidean	-	-	-	1396	156	0.6
A*	chebyshev	-	-	-	1401	156	0.6
A*	octile	-	-	-	1351	156	0.6
A*	bidirectionalManhattan	-	-	-	1377	156	0.6
A*	bidirectionalEuclidean	-	-	-	1396	156	0.6
A*	bidirectionalChebyshev	-	-	-	1401	156	0.6
A*	bidirectionalOctile	-	-	-	1367	156	0.6
Weighted A*	octile	1	2	-	1262	156	0.5
Weighted A*	octile	2	1	-	1557	156	0.8
Weighted A*	octile	1	3	-	1196	156	0.5
Weighted A*	octile	3	1	-	1631	156	0.8
Weighted A*	octile	2	3	-	1293	156	0.5
Weighted A*	octile	3	2	-	1513	156	0.7
BEAM	octile	-	-	40	1156	160	0.4
BEAM	octile	-	-	50	1285	159	0.5
BEAM	octile	-	-	60	1291	159	0.5
Hill Climb	-	-	-	-	15779	999999	0.2

Table 1.6: For big map

1.2 Corners Problem

The second problem that needs to be solved is the *"Corners Problem"* which consists of a starting position from which *Pac-Man* begins the search and four ending positions represented by the four corners of the maze's map, all of which need to be reached. The searching algorithms have already been implemented (the same ones from the first problem also work here) and so, all that needs to be done is to define a new representation of the state of this search problem, meaning that the following functions need to be implemented: *getStartState()*, *isGoalState()* and *getSuccessors()*. Here, in order for the already implemented algorithms to work on this new problem without modifying them, a new structure needs to be added alongside the coordinates of a state. for this, I have chosen an appearance list meaning that, whenever a new node is created, at the beginning, of the search, it will contain a list of four zeros ([0,0,0,0]), but, once a corner has been reached, the list changes one of the four values to one ([1,0,0,0]), corresponding to that corner. In this way, we avoid two problems:

1) The search algorithm does not get stuck in a corner due to the fact that the positions around it were already visited

2)There is a distinction between every corner of the map since we have four different elements in the list.

Additionally, another aspect which can be implemented is the *cornersHeuristic()* function which has to be an admissible and efficient heuristic that is specific only to this search problem. Something like the Manhattan distance heuristic can also be used here, but the most appropriate one that I could find is a heuristic which is equal to the maximum octile distance to the unvisited

corners. To this number, the number of unvisited corners is added, at the end. For reference, the number of nodes expanded (on the *mediumCorners* map), using the Manhattan distance heuristic, is 2266, while using the octile distance heuristic, this number drops to about 2048, which is not a big difference, but as the map gets larger, so does the gap between these two numbers (for the *bigCorners* map, Manhattan expands 8902 nodes while octile expands 6974 nodes).

1.3 All Food Problem

The third problem that needs to be solved is the *"All Food Problem"* which consists of a starting position from which *Pac-Man* begins the search and there are multiple food pellets across the map, all of which need to be eaten by the game's character. for this problem, there is no need to define another representation for its state because the three functions which were mentioned in the previous paragraph are already implemented. The challenge for this problem is too find an admissible and good heuristic, that can decrease the number of expanded nodes as much as possible. Starting from the idea of the previous problem's heuristic, I first tried the maximum octile distance summed up with the number of remaining food pellets. However, this did not work so well for this problem (in fact it was so bad, that the game was frozen for a very long time, until I got bored and manually closed it), so instead of taking the maximum distance, I decided to take the sum of all octile distances to the remaining food pellets summed up with the number of remaining food pellets. This yielded a very good result in comparison with the previous heuristic (10915 expanded nodes), but it still wasn't enough, so I came up with another idea which turned out to be quite good. Since the octile distance is the largest out of the four distances that I implemented, namely the Manhattan distance, the Euclidean distance, the Chebyshev distance and the octile distance itself, I decided to multiply the latter by two and subtract the Manhattan one. Again, a visible improvement was achieved (9462 expanded nodes), so then I continued this idea and thus, multiplied the octile distance by four and subtracted the sum of all the other distances. The number of expanded nodes continue to decrease (down to 8520). The last improvement that I found was to multiply the final result with a weight that I kept increasing until I found the smallest possible number of expanded nodes that I could obtain (4646 expanded nodes obtained with a weight equal to 500). Increasing the weight further did not affect the performance in any way. Thus, this last improvement (with the weight) is the same as having a *Weighted A** algorithm and setting the weight of the heuristic to 500. In addition, this heuristic can also be applied to the previous problem, since the *"All Food Problem"* is essentially a generalization of the *"Corners Problem"*.

Chapter 2

A2: Logics

In this chapter, the work/code is done using Prover9 and/or Mace4. Prover9 is an autoamted theorem prover for first-order logic and equational logic, while Mace4 searches for finite models and counterexamples.

2.1 Warm-Up Logic Puzzles

2.1.1 Tricky Heights 1

Problem: James is taller than Kate and Carly. Sammy is shorter than Kate. Natalie is shorter than Kate and Sammy, however Sammy is shorter than Carly. Who is the shortest?

Solution: In order from tallest to shortest - James, Kate, Carly, Sammy, Natalie. Notice that no-one is shorter than Natalie.

Solution by Mace4:

Listing 2.1: Fragment from *heightPuzzle1.out*

```
    interpretation( 5, [number=1, seconds=0], [

        function(Carly, [ 2 ]),
        function(James, [ 4 ]),
        function(Kate, [ 3 ]),
        function(Natalie, [ 0 ]),
        function(Sammy, [ 1 ]),

        relation(shorter(_,_), [
                            0, 1, 1, 1, 1,
                            0, 0, 1, 1, 1,
                            0, 0, 0, 1, 1,
                            0, 0, 0, 0, 1,
                            0, 0, 0, 0, 0 ]),
        relation(taller(_,_), [
                            0, 0, 0, 0, 0,
                            1, 0, 0, 0, 0,
                            1, 1, 0, 0, 0,
                            1, 1, 1, 0, 0,
                            1, 1, 1, 1, 0 ])
]).
```

2.1.2 Tricky Heights 2

Problem: Rachel and Jessica are taller than John. Maria is shorter than Lara and Lara is taller than Sally. John is taller than Lara. Who is the tallest out of John, Maria, Lara and Sally?

Solution: The answer is John. Remember, it was only out of John, Maria, Lara and Sally.

Solution by Mace4:

Listing 2.2: Fragment from *heightPuzzle2.out*

```
interpretation( 6, [number=1, seconds=0], [

        function(Jessica, [ 4 ]),
        function(John, [ 3 ]),
        function(Lara, [ 2 ]),
        function(Maria, [ 0 ]),
        function(Rachel, [ 5 ]),
        function(Sally, [ 1 ]),

        relation(shorter(_,_), [
                              0, 1, 1, 1, 1, 1,
                              0, 0, 1, 1, 1, 1,
                              0, 0, 0, 1, 1, 1,
                              0, 0, 0, 0, 1, 1,
                              0, 0, 0, 0, 0, 1,
                              0, 0, 0, 0, 0, 0 ]),
        relation(taller(_,_), [
                              0, 0, 0, 0, 0, 0,
                              1, 0, 0, 0, 0, 0,
                              1, 1, 0, 0, 0, 0,
                              1, 1, 1, 0, 0, 0,
                              1, 1, 1, 1, 0, 0,
                              1, 1, 1, 1, 1, 0 ])
]).
```

2.1.3 Tricky Heights 3

Problem: Joy is taller than Cassie. Joy is taller than Brian who is taller than Cassie and Geoff. Geoff is shorter than Val who is shorter than Cassie. Is Val the shortest?

Solution: The answer is NO. Joy, Brian, Cassie, Val, Geoff. Geoff is shortest. Geoff is shorter than Val, remember.

Solution by Mace4:

Listing 2.3: Fragment from *heightPuzzle3.out*

```
interpretation( 5, [number=1, seconds=0], [

        function(Brian, [ 3 ]),
        function(Cassie, [ 2 ]),
        function(Geoff, [ 0 ]),
```

```
        function(Joy, [ 4 ]),
        function(Val, [ 1 ]),

        relation(shorter(_,_), [
                              0, 1, 1, 1, 1,
                              0, 0, 1, 1, 1,
                              0, 0, 0, 1, 1,
                              0, 0, 0, 0, 1,
                              0, 0, 0, 0, 0 ]),
        relation(taller(_,_), [
                              0, 0, 0, 0, 0,
                              1, 0, 0, 0, 0,
                              1, 1, 0, 0, 0,
                              1, 1, 1, 0, 0,
                              1, 1, 1, 1, 0 ])
]).
```

2.2 Einstein Puzzles

2.2.1 Einstein Puzzle 1 - Couples Borrowing Books

Problem: Eight married couples meet to lend one another some books. Couples have the same surname, employment and car. Each couple has a favorite color. Furthermore we know the following facts:

1. Daniella Black and her husband work as Shop-Assistants.
2. The book "The Seadog" was brought by a couple who drive a Fiat and love the color red.
3. Owen and his wife Victoria like the color brown.
4. Stan Horricks and his wife Hannah like the color white.
5. Jenny Smith and her husband work as Warehouse Managers and they drive a Wartburg.
6. Monica and her husband Alexander borrowed the book "Grandfather Joseph".
7. Mathew and his wife like the color pink and brought the book "Mulatka Gabriela".
8. Irene and her husband Oto work as Accountants.
9. The book "We Were Five" was borrowed by a couple driving a Trabant and was brought by the Smith couple.
10. The Cermaks are both Ticket-Collectors who brought the book "Shed Stoat", which was borrowed by the Zajac couple.
11. Mr and Mrs Kuril are both Doctors who borrowed the book "Slovacko Judge".
12. Paul and his wife like the color green.
13. Veronica Dvorak and her husband like the color blue.
14. Rick and his wife brought the book "Slovacko Judge" and they drive a Ziguli.
15. One couple brought the book "Dame Commissar" and borrowed the book "Mulatka Gabriela".
16. The couple who drive a Dacia, love the color violet.
17. The couple who work as Teachers borrowed the book "Dame Commissar".
18. The couple who work as Agriculturalists drive a Moskvic.
19. Pamela and her husband drive a Renault and brought the book "Grandfather Joseph".
20. Pamela and her husband borrowed the book that Mr and Mrs Zajac brought.
21. Robert and his wife like the color yellow and borrowed the book "The Modern Comedy".

22. Mr and Mrs Swain work as Shoppers.
23. "The Modern Comedy" was brought by a couple driving a Skoda.

Who likes Violet? And can you find out everything about everyone from this?

Solution: Monica and Alexander Cermak like Violet.

Names	Occupation	Car	Color	Brought	Borrowed
Daniella and Mathew Black	Shop-Assistants	Trabant	pink	"Mulatka Gabriela"	"We Were Five"
Victoria and Owen Kuril	Doctors	Skoda	brown	"The Modern Comedy"	"Slovacko Judge"
Hannah and Stan Horricks	Agriculturalists	Moskvic	white	"Dame Commissar"	"Mulatka Gabriela"
Jenny and Robert Smith	Warehouse Managers	Wartburg	yellow	"We Were Five"	"The Modern Comedy"
Monica and Alexander Cermak	Ticket-Collectors	Dacia	violet	"Shed Stoat"	"Grandfather Joseph"
Irene and Oto Zajac	Accountants	Fiat	red	"The Seadog"	"Shed Stoat"
Pamela and Paul Swain	Shoppers	Renault	green	"Grandfather Joseph"	"The Seadog"
Veronica and Rick Dvorak	Teachers	Ziguli	blue	"Slovacko Judge"	"Dame Commissar"

Solution by Mace4:

Listing 2.4: Fragment from *einsteinPuzzle1.out*

```
interpretation( 8, [number=1, seconds=2], [

        function(Accountant, [ 0 ]),
        function(Agriculturist, [ 1 ]),
        function(Alexander, [ 6 ]),
        function(Black, [ 3 ]),
        function(Blue, [ 5 ]),
        function(Brown, [ 2 ]),
        function(Cermak, [ 6 ]),
        function(Dacia, [ 6 ]),
        function(DameCommissar, [ 0 ]),
        function(Daniella, [ 3 ]),
        function(Doctor, [ 2 ]),
        function(Dvorak, [ 5 ]),
        function(Fiat, [ 0 ]),
        function(GrandfatherJoseph, [ 1 ]),
        function(Green, [ 4 ]),
        function(Hannah, [ 1 ]),
        function(Horricks, [ 1 ]),
        function(Irene, [ 0 ]),
        function(Jenny, [ 7 ]),
        function(Kuril, [ 2 ]),
        function(Matthew, [ 3 ]),
        function(Monica, [ 6 ]),
        function(Moskvic, [ 1 ]),
        function(MulatkaGabriela, [ 2 ]),
        function(Oto, [ 0 ]),
        function(Owen, [ 2 ]),
        function(Pamela, [ 4 ]),
        function(Paul, [ 4 ]),
        function(Pink, [ 3 ]),
        function(Red, [ 0 ]),
        function(Renault, [ 4 ]),
        function(Rick, [ 5 ]),
```

```
        function(Robert, [ 7 ]),
        function(ShedStoat, [ 3 ]),
        function(ShopAssistant, [ 3 ]),
        function(Shopper, [ 4 ]),
        function(Skoda, [ 2 ]),
        function(SlovackoJudge, [ 4 ]),
        function(Smith, [ 7 ]),
        function(Stan, [ 1 ]),
        function(Swain, [ 4 ]),
        function(Teacher, [ 5 ]),
        function(TheModernComedy, [ 5 ]),
        function(TheSeadog, [ 6 ]),
        function(TicketCollector, [ 6 ]),
        function(Trabant, [ 3 ]),
        function(Veronica, [ 5 ]),
        function(Victoria, [ 2 ]),
        function(Violet, [ 6 ]),
        function(WarehouseManager, [ 7 ]),
        function(Wartburg, [ 7 ]),
        function(WeWereFive, [ 7 ]),
        function(White, [ 1 ]),
        function(Yellow, [ 7 ]),
        function(Zajac, [ 0 ]),
        function(Ziguli, [ 5 ]),

        relation(borrowed(_,_), [
                              0, 0, 0, 1, 0, 0, 0, 0,
                              0, 0, 1, 0, 0, 0, 0, 0,
                              0, 0, 0, 0, 1, 0, 0, 0,
                              0, 0, 0, 0, 0, 0, 0, 1,
                              0, 0, 0, 0, 0, 0, 1, 0,
                              1, 0, 0, 0, 0, 0, 0, 0,
                              0, 1, 0, 0, 0, 0, 0, 0,
                              0, 0, 0, 0, 0, 1, 0, 0 ]),
        relation(brought(_,_), [
                              0, 0, 0, 0, 0, 0, 1, 0,
                              1, 0, 0, 0, 0, 0, 0, 0,
                              0, 0, 0, 0, 0, 1, 0, 0,
                              0, 0, 1, 0, 0, 0, 0, 0,
                              0, 1, 0, 0, 0, 0, 0, 0,
                              0, 0, 0, 0, 1, 0, 0, 0,
                              0, 0, 0, 1, 0, 0, 0, 0,
                              0, 0, 0, 0, 0, 0, 0, 1 ])
]).
```

2.2.2 Einstein Puzzle 2 - Employees and Departments

Problem: Alex, Betty, Carol, Dan, Earl, Fay, George and Harry are eight employees of an organization.

They work in three departments: Personnel, Administration and Marketing with not more than three of them in any department.

Each of them has a different choice of sports from Football, Cricket, Volleyball, Badminton, Lawn Tennis, Basketball, Hockey and Table Tennis not necessarily in the same order.

Dan works in Administration and does not like either Football or Cricket.
Fay works in Personnel with only Alex who likes Table Tennis.
Earl and Harry do not work in the same department as Dan.
Carol likes Hockey and does not work in Marketing.
George does not work in Administration and does not like either Cricket or Badminton.
One of those who work in Administration likes Football.
The one who likes Volleyball works in Personnel.
None of those who work in Administration likes either Badminton or Lawn Tennis. Harry does not like Cricket.

Who are the employees who work in the Administration Department?

In which Department does Earl work?

Solution: Betty, Carol and Dan work in the Administration Department.

Earl works in the Marketing Department.

Solution by Mace4:

Listing 2.5: Fragment from *einsteinPuzzle2.out*

```
interpretation( 8, [number=1, seconds=0], [

        function(Administration, [ 1 ]),
        function(Alex, [ 0 ]),
        function(Badminton, [ 7 ]),
        function(Betty, [ 1 ]),
        function(Basketball, [ 3 ]),
        function(Carol, [ 2 ]),
        function(Cricket, [ 4 ]),
        function(Dan, [ 3 ]),
        function(Earl, [ 4 ]),
        function(Fay, [ 5 ]),
        function(Footbal, [ 1 ]),
        function(George, [ 6 ]),
        function(Harry, [ 7 ]),
        function(Hockey, [ 2 ]),
        function(LawnTennis, [ 6 ]),
        function(Marketing, [ 2 ]),
        function(Personnel, [ 0 ]),
        function(TableTennis, [ 0 ]),
        function(Volleyball, [ 5 ]),

        relation(Department(_), [ 1, 1, 1, 0, 0, 0, 0, 0 ]),

        function(WorksIn(_,_), [
                        1, 0, 0, 0, 0, 0, 0, 0,
                        0, 1, 0, 0, 0, 0, 0, 0,
```

```
                                0, 1, 0, 0, 0, 0, 0, 0,
                                0, 1, 0, 0, 0, 0, 0, 0,
                                0, 0, 1, 0, 0, 0, 0, 0,
                                1, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 1, 0, 0, 0, 0, 0,
                                0, 0, 1, 0, 0, 0, 0, 0 ]),

        relation(WorksWith(_,_), [
                                0, 0, 0, 0, 0, 1, 0, 0,
                                0, 0, 1, 1, 0, 0, 0, 0,
                                0, 1, 0, 1, 0, 0, 0, 0,
                                0, 1, 1, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 1, 1,
                                1, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 1, 0, 0, 1,
                                0, 0, 0, 0, 1, 0, 1, 0 ])
]).
```

2.2.3 Einstein Puzzle 3 - GARDENS

Problem: Five friends have their gardens next to one another, where they grow three kinds of crops: fruits (apple, pear, nut, cherry), vegetables (carrot, parsley, gourd, onion) and flowers (aster, rose, tulip, lily).

1. They grow 12 different varieties.
2. Everybody grows exactly 4 different varieties.
3. Each variety is at least in one garden.
4. Only one variety is in 4 gardens.
5. Only in one garden are all 3 kinds of crops.
6. Only in one garden are all 4 varieties of one kind of crops.
7. Pear is only in the two border gardens.
8. Paul's garden is in the middle with no lily.
9. Aster grower doesn't grow vegetables.
10. Rose growers don't grow parsley.
11. Nuts grower has also gourd and parsley.
12. In the first garden are apples and cherries.
13. Only in two gardens are cherries.
14. Sam has onions and cherries.
15. Luke grows exactly two kinds of fruit.
16. Tulip is only in two gardens.
17. Apple is in a single garden.
18. Only in one garden next to Zick's is parsley.
19. Sam's garden is not on the border.
20. Hank grows neither vegetables nor asters.
21. Paul has exactly three kinds of vegetable.

Who has which garden and what is grown where?

Solution:
Hank: pear apple cherry rose
Sam: cherry onion rose tulip

Paul: carrot gourd onion rose
Zick: aster rose tulip lily
Luke: pear nut gourd parsley

Solution by Mace4:

Listing 2.6: Fragment from *einsteinPuzzle3.out*

```
interpretation( 12, [number=1, seconds=27], [

        function(Apple, [ 0 ]),
        function(Aster, [ 8 ]),
        function(Carrot, [ 4 ]),
        function(Cherry, [ 3 ]),
        function(Gourd, [ 6 ]),
        function(Hank, [ 1 ]),
        function(Lily, [11 ]),
        function(Luke, [ 5 ]),
        function(Nut, [ 2 ]),
        function(Onion, [ 7 ]),
        function(Parsley, [ 5 ]),
        function(Paul, [ 3 ]),
        function(Pear, [ 1 ]),
        function(Rose, [ 9 ]),
        function(Sam, [ 2 ]),
        function(Tulip, [10 ]),
        function(Zick, [ 4 ]),

        function(inGarden(_,_), [
                        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                        1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0,
                        0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0,
                        0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0,
                        0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
                        0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0,
                        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]),

        relation(flower(_), [ 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1 ]),
        relation(fruit(_), [ 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0 ]),
        relation(validGarden(_), [ 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0
           ]),
        relation(vegetable(_), [ 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0 ])
]).
```

2.3 Sam Loyd Puzzles

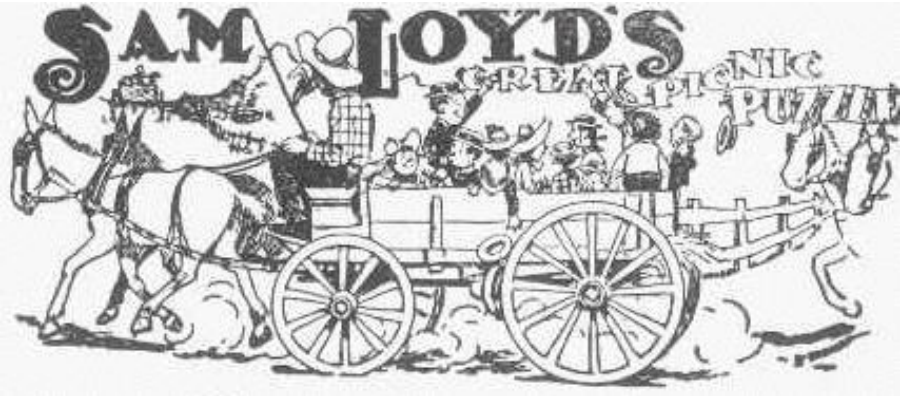
2.3.1 Sam Loyd Puzzle 1 - The Great Annual Picnic

Problem: When they started off on the great annual picnic every wagon in town was pressed into service, each one carrying the same number of people.

Half way to the picnic ground ten wagons broke down, so it was necessary for each of the remaining wagons to carry one more person.

When they started for home it was discovered that fifteen more wagons were out of commission, so on the return trip there were three persons more in each wagon than when they started out in the morning.

Now who can tell how many people attended the great annual picnic?



Solution: There must have been 900 picnickers who would be seated 9 to a wagon if there were 100 vehicles, or 10 to a wagon after 10 of the wagons had broken.

When they started for home with 75 wagons, it was necessary for 12 persons to ride in each wagon (3 more than the 9 per wagon in the morning).

Solution by Mace4:

Listing 2.7: Fragment from *samLoydPuzzle1.out*

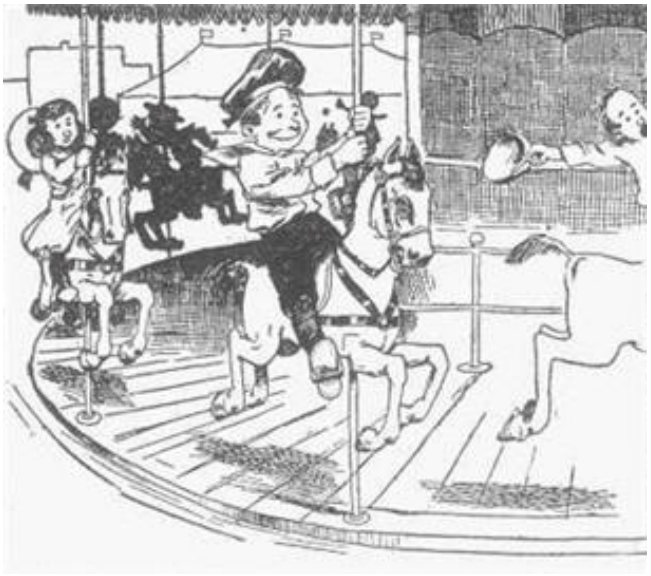
```
interpretation( 901, [number=1, seconds=12], [

        function(People0, [ 9 ]),
        function(People1, [10 ]),
        function(People2, [12 ]),
        function(Wagon0, [100 ]),
        function(Wagon1, [90 ]),
        function(Wagon2, [75 ])
]).
```

2.3.2 Sam Loyd Puzzle 2 - Sammy Riding a Carousel

Problem: While enjoying a giddy ride at the carousel Sammy propounded a puzzle which reflects much credit to his mental abilities.

"One third of the number of kids riding ahead of me, added to three-quarter of those riding behind me gives the correct number of children on this Merry-Go-Round" is the way he puts it; but it will puzzle you quite a little to tell just how many riders there were at this whirling circus.



Solution: There must have been a multiple of 3 as well as 4 plus one children.

So, the minimum number of children riding on this Merry-Go-Round could be thirteen.

Those who rode ahead of Sammy at the same time came behind him.

If there were twelve, we simply add three-quarter of twelve to one-third of twelve, which gives thirteen, the total number including Sammy himself.

Solution by Mace4:

Listing 2.8: Fragment from *samLoydPuzzle2.out*

```
interpretation( 30, [number=1, seconds=0], [

        function(childrenWithSammy, [13 ]),
        function(childrenWithoutSammy, [12 ])
]).
```

2.3.3 Sam Loyd Puzzle 3 - Arab Sheikh Paradox

Problem: An Arab Sheik, finding himself about to die, called his sons about him and said: "Divide my camels among you in the proportion of one-half of the herd to the eldest son, the

second son one-third, and to the youngest son one-ninth." Thereupon the oldest son cried: "O, my father, one-half, one-third, and one-ninth do not constitute a whole. To whom, therefore, shall the remainder of the herd be given?"

"To any poor man who may be standing by when the division is made," replied the Sheik, who thereupon died.

When the herd was collected, a new difficulty arose. The number of the camels could not be divided either by two or three or nine. While the brothers were disputing, a poor but crafty Bedouin, standing by with his camel, exclaimed, "Behold, I will sell you my beast for ten pieces of silver, so that you may then divide the herd."

Seeing that the addition of one camel would solve the difficulty, the brothers jumped at the offer, and proceeded to divide the herd, but when each had received his allotted portion there yet remained one camel.

"I am the poor man standing by." Said the crafty Bedouin, and gaily mounting the camel, he rode away, with the ten pieces of silver in his turban.

Now, how many camels were in the Sheik's herd?

Solution: The camels could be divided exactly according to the Sheik's will only if it were a multiple of 2, 3 and 9 i.e. 18, 36, 54, 72,...

But according to the brothers, they needed one more camel to divide it according to the Sheik's will; so the Sheik had 17, 35, 52, 71,... camels.

But "only 18" (18/2=9, 18/3=6, 18/9=2 so that 9+6+2=17) leaves remainder one (18-17=1). So there were 17 camels in the Sheik's herd.

But, the problem is that the number of camels 9, 6, 2 that the brothers received are not yet in the proportion of one-half, one-third and one-ninth. This is the paradoxical situation. The Camels could be divided exactly according to the Sheik's will only if he had 18+..., 36+..., 54+..., 72+...,... camels.

Probably the Sheik was not good at mathematics. Therefore, this puzzle has no solution (it is a paradox).

Solution by Mace4:

Listing 2.9: Fragment from *samLoydPuzzle3.out*

```
Exiting with failure.

Process 596 exit (exhausted) Sat Dec  9 13:44:38 2023
The process finished Sat Dec  9 13:44:38 2023
```

2.4 Math Puzzles

2.4.1 Math Puzzle 1 - Weighted Boxes in Crates

Problem: I have ten boxes, with a total weight of 75kg:

15 kg, 13 kg, 11 kg, 10 kg, 9 kg, 8 kg, 4 kg, 2 kg, 2 kg, 1 kg

I want to pack the boxes into 3 crates, but each crate can carry a maximum of 25 kg.

How can I pack the boxes into the crates?

Solution: There are ten possible ways:

{Crate 1}, {Crate 2}, {Crate 3}
{15,10}, {13,8,4}, {11,9,2,2,1}
{15,10}, {13,11,1}, {9,8,4,2,2}
{15,10}, {11,8,4,2}, {13,9,2,1}
{15,10}, {11,9,4,1}, {13,8,2,2}
{11,10,4}, {15,8,2}, {13,9,2,1}
{11,10,4}, {15,9,1}, {13,8,2,2}
{13,8,4}, {15,9,1}, {11,10,2,2}
{13,10,2}, {15,8,2}, {11,9,4,1}
{13,10,2}, {15,9,1}, {11,8,4,2}
{13,11,1}, {15,8,2}, {10,9,4,2}

Solution by Mace4:

Looking at the solution, I decided to divide the problem into three subproblems:

a) The first crate has exactly 2 boxes, the second crate has exactly 3 boxes and the third crate has exactly 5 boxes;
b) The first crate has exactly 2 boxes, the second crate has exactly 4 boxes and the third crate also has exactly 4 boxes;
c) The first crate has exactly 3 boxes, the second crate also has exactly three boxes and the third crate has exactly 4 boxes;

Listing 2.10: Fragment from *mathPuzzle1_a.out*

```
interpretation( 76, [number=1, seconds=0], [

```

```
        function(B1, [ 1 ]),
        function(B2, [ 4 ]),
        function(B3, [ 2 ]),
        function(B4, [ 3 ]),
        function(B5, [10 ]),
        function(B6, [ 5 ]),
        function(B7, [ 6 ]),
        function(B8, [ 7 ]),
        function(B9, [ 8 ]),
        function(B10, [ 9 ]),

        function(c1, [25 ]),
        function(c2, [25 ]),
        function(c3, [25 ]),

        function(b(_), [ 0,15,13,11,10, 9, 8, 4, 2, 2, 1, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0, 0, 0 ])
]).

interpretation( 76, [number=2, seconds=0], [

        function(B1, [ 1 ]),
        function(B2, [ 4 ]),
        function(B3, [ 2 ]),
        function(B4, [ 6 ]),
        function(B5, [ 7 ]),
        function(B6, [ 3 ]),
        function(B7, [ 5 ]),
        function(B8, [ 8 ]),
        function(B9, [ 9 ]),
        function(B10, [10 ]),

        function(c1, [25 ]),
        function(c2, [25 ]),
        function(c3, [25 ]),

        function(b(_), [ 0,15,13,11,10, 9, 8, 4, 2, 2, 1, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0, 0, 0 ])
]).
```

Listing 2.11: Fragment from *mathPuzzle1_b.out*

```
interpretation( 76, [number=1, seconds=0], [

        function(B1, [ 1 ]),
        function(B2, [ 4 ]),
        function(B3, [ 2 ]),
        function(B4, [ 5 ]),
```

```
        function(B5, [ 8 ]),
        function(B6, [10 ]),
        function(B7, [ 3 ]),
        function(B8, [ 6 ]),
        function(B9, [ 7 ]),
        function(B10, [ 9 ]),

        function(c1, [25 ]),
        function(c2, [25 ]),
        function(c3, [25 ]),

        function(b(_), [ 0,15,13,11,10, 9, 8, 4, 2, 2, 1, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0, 0, 0 ])
]).

interpretation( 76, [number=2, seconds=0], [

        function(B1, [ 1 ]),
        function(B2, [ 4 ]),
        function(B3, [ 2 ]),
        function(B4, [ 6 ]),
        function(B5, [ 8 ]),
        function(B6, [ 9 ]),
        function(B7, [ 3 ]),
        function(B8, [ 5 ]),
        function(B9, [ 7 ]),
        function(B10, [10 ]),

        function(c1, [25 ]),
        function(c2, [25 ]),
        function(c3, [25 ]),

        function(b(_), [ 0,15,13,11,10, 9, 8, 4, 2, 2, 1, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0, 0, 0 ])
]).
```

Listing 2.12: Fragment from *mathPuzzle1_c.out*

```
interpretation( 76, [number=1, seconds=0], [

        function(B1, [ 1 ]),
        function(B2, [ 5 ]),
        function(B3, [10 ]),
        function(B4, [ 2 ]),
        function(B5, [ 4 ]),
        function(B6, [ 9 ]),
        function(B7, [ 3 ]),
        function(B8, [ 6 ]),
```

```
        function(B9, [ 7 ]),
        function(B10, [ 8 ]),

        function(c1, [25 ]),
        function(c2, [25 ]),
        function(c3, [25 ]),

        function(b(_), [ 0,15,13,11,10, 9, 8, 4, 2, 2, 1, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0, 0, 0 ])
]).

interpretation( 76, [number=2, seconds=0], [

        function(B1, [ 1 ]),
        function(B2, [ 5 ]),
        function(B3, [10 ]),
        function(B4, [ 2 ]),
        function(B5, [ 4 ]),
        function(B6, [ 8 ]),
        function(B7, [ 3 ]),
        function(B8, [ 6 ]),
        function(B9, [ 7 ]),
        function(B10, [ 9 ]),

        function(c1, [25 ]),
        function(c2, [25 ]),
        function(c3, [25 ]),

        function(b(_), [ 0,15,13,11,10, 9, 8, 4, 2, 2, 1, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0, 0, 0 ])
]).

interpretation( 76, [number=3, seconds=0], [

        function(B1, [ 1 ]),
        function(B2, [ 5 ]),
        function(B3, [10 ]),
        function(B4, [ 2 ]),
        function(B5, [ 6 ]),
        function(B6, [ 7 ]),
        function(B7, [ 3 ]),
        function(B8, [ 4 ]),
        function(B9, [ 8 ]),
        function(B10, [ 9 ]),

        function(c1, [25 ]),
        function(c2, [25 ]),
```

```
        function(c3, [25 ]),

        function(b(_), [ 0,15,13,11,10, 9, 8, 4, 2, 2, 1, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0, 0, 0 ])
]).

interpretation( 76, [number=4, seconds=0], [

        function(B1, [ 1 ]),
        function(B2, [ 5 ]),
        function(B3, [10 ]),
        function(B4, [ 3 ]),
        function(B5, [ 4 ]),
        function(B6, [ 7 ]),
        function(B7, [ 2 ]),
        function(B8, [ 6 ]),
        function(B9, [ 8 ]),
        function(B10, [ 9 ]),

        function(c1, [25 ]),
        function(c2, [25 ]),
        function(c3, [25 ]),

        function(b(_), [ 0,15,13,11,10, 9, 8, 4, 2, 2, 1, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0, 0, 0 ])
]).

interpretation( 76, [number=5, seconds=0], [

        function(B1, [ 1 ]),
        function(B2, [ 6 ]),
        function(B3, [ 8 ]),
        function(B4, [ 2 ]),
        function(B5, [ 3 ]),
        function(B6, [10 ]),
        function(B7, [ 4 ]),
        function(B8, [ 5 ]),
        function(B9, [ 7 ]),
        function(B10, [ 9 ]),

        function(c1, [25 ]),
        function(c2, [25 ]),
        function(c3, [25 ]),

        function(b(_), [ 0,15,13,11,10, 9, 8, 4, 2, 2, 1, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
            0, 0, 0, 0 ])
]).

interpretation( 76, [number=6, seconds=0], [

        function(B1, [ 1 ]),
        function(B2, [ 6 ]),
        function(B3, [ 8 ]),
        function(B4, [ 2 ]),
        function(B5, [ 4 ]),
        function(B6, [ 9 ]),
        function(B7, [ 3 ]),
        function(B8, [ 5 ]),
        function(B9, [ 7 ]),
        function(B10, [10 ]),

        function(c1, [25 ]),
        function(c2, [25 ]),
        function(c3, [25 ]),

        function(b(_), [ 0,15,13,11,10, 9, 8, 4, 2, 2, 1, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0, 0, 0 ])
]).
```

2.4.2 Math Puzzle 2 - Bread for Workers

Problem: About 2000 years BC there lived Ahmes, a royal secretary and mathematician of the Pharaoh Amenemhat III. One of his papyruses was found in 1853 by an Englishman called Rhind near the temple of Ramses II in Thebes. It has many mathematical puzzles and here is one:

120 loaves of bread must be divided among five workers.

Each worker in line must get more than the previous: the same amount more in each case (an arithmetical progression).

And the first two workers shall get seven times less than the three others.

How many loaves does each worker get?

Solution: Let us say the middle worker (worker 3) gets "w" loaves. And that "d" is the common difference between workers.

So the workers get:
w-2d
w-d
w

w+d
w+2d

The middle worker gets a perfect average, so 120/5 = 24 loaves

The first two workers get seven times less than the three others: 7*[(24-2d) + (24-d)] = 24 + (24+d) + (24+2d)

From this: d = 264/24=11

And this is the solution:

1st worker = 2 loaves
2nd worker = 13 loaves
3rd worker = 24 loaves
4th worker = 35 loaves
5th worker = 46 loaves

Solution by Mace4:

Listing 2.13: Fragment from *mathPuzzle2.out*

```
interpretation( 121, [number=1, seconds=0], [
        function(Worker, [24 ]),

        function(Worker1, [ 2 ]),
        function(Worker2, [13 ]),
        function(Worker3, [24 ]),
        function(Worker4, [35 ]),
        function(Worker5, [46 ]),

        function(d, [11 ]),
        function(loaves, [120 ]),
        function(numberOfWorkers, [ 5 ])
]).
```

2.4.3 Math Puzzle 3 - Diminishing Coconuts

Problem: After being marooned on an island, a group of five people spent some time obtaining a lot of coconuts. After the five have decided that they have collected enough coconuts to last long enough for rescuers to arrive, they place all coconuts in a communal pile.

However, the first person suddenly had concerns about how the group would divide the coconuts the following day. In the dead of the night, the first survivor divided the pile into five equal piles of whole coconuts, gave one remaining coconut to a passing monkey, hid their share of the coconuts (one-fifth of the pile), and mixed the four other piles to cover his tracks before going back to sleep.

The second survivor had the same exact train of thought, and proceeded to divide the pile into five equal piles of whole coconuts, give one remaining coconut to a passing monkey, hide

their share of the coconuts (one-fifth of the pile), and mix back together the other four piles before going back to sleep.

In an ironic twist of fate, the other three survivors also had the same line of reasoning as the other two survivors and proceeded to do exactly the same thing (divide the pile, give one remaining coconut to a passing monkey, etc.). In a twist of fate, none of the survivors woke up to the other survivors taking their share from the pile.

When the group woke up in the morning, everyone could see that the pile was substantially reduced, but since every survivor took from the pile, no one said anything to incriminate themselves. Nonetheless, the survivors divided the reduced pile of coconuts into five equal shares of whole coconuts one last time, this time without any remaining coconuts to give to any monkeys.

What is the least number of whole coconuts the pile can have before the night?

Solution: First, we start with 5^5 coconuts. This is the smallest number that can be divided evenly into fifths, have a fifth removed and the process repeated five times, with no coconuts going to the monkey.

Four of the 5^5 coconuts are rotten and placed aside. When the remaining supply of coconuts is divided into fifths, there will of course be one non-rotten coconut left over to give to the monkey.

After the first person has taken his share, and the monkey has his coconut, we put the four rotten coconuts back with the others to make a pile of 5^4 coconuts. This can be evenly divided by 5. Before making this next division, however, we again put the four rotten coconuts aside so that the division will leave an extra coconut for the monkey.

This procedure - borrowing the rotten coconuts only long enough to see that an even division into fifths can be made, then putting them aside again - is repeated at each division. After the sixth and last division, the rotten coconuts remain on the side, the property of no one.

The rotten coconuts aren't actual coconuts, since they end up the property of no one, but since we see that 5^5 coconuts can be divided neatly with the use of the four rotten coconuts, the answer is 5^5 - 4, or 3121 coconuts.

Solution by Mace4:

Listing 2.14: Fragment from *mathPuzzle3.out*

```
interpretation( 3200, [number=1, seconds=115], [

        function(a1, [3121 ]),

        function(a2, [624 ]),
        function(a3, [499 ]),
        function(a4, [399 ]),
        function(a5, [319 ]),
        function(a6, [255 ]),
        function(a7, [204 ])
]).
```

2.4.4 Math Puzzle 4 - Riddle of Ages

Problem: I was talking to a friend Tammy, the middle child of five. She has a younger sister Tracey, and three brothers Tommy (oldest child), Timmy and Tony (youngest child). I asked her how old she was, but she hardly ever gives a straight answer. The following is the conversation we had. "How old are you Tammy?" "I am three times as old as Timmy was when Tony was born. Actually, it's funny, but Timmy's, Tracey's, Tommy's and my ages were all factors of Mum's age when Tony was born. The only other year where the ages of more than two of us were factors of Mum's age was when Tracey was half as old as I am now." "How old is your Mum now?" "There's another funny thing. When Mum turns 50, Tony will be the same age Mum was when she gave birth to Tommy."

Assuming Tammy was talking in terms of whole years, and with my mathematical and reasoning abilities I was able to quickly deduce not just Tammy's age, but all of their ages. So how old are Tommy, Timmy, Tammy, Tracey, Tony and Mum now?

Solution: Tommy is 23, Timmy is 19, Tammy is 18, Tracey is 16, Tony is 13 and Mum is 43.

x = Mum's age when she gave birth to Tommy = Tony's age when Mum turns 50
y = Tommy's age when Tony was born
z = Mum's age when Tony was born

From Tammy's last statement, 2x + y = 50 and it can be deduced that x = z - y, where y is a factor of z. By combining the two equations you get 2z = 50 + y The only possible solutions for that equation so that y is also a factor of z are (z = 26, y = 2) and (z = 30, y = 10). Only 30 has four or more factors, so Mum was 30 and Tommy was 10 when Tony was born.

This means that of the remaining factors of 30, Timmy was 6, 5 or 3, Tammy was 5, 3 or 2 and Tracey was 3, 2 or 1. This makes Tammy 18, 15 or 9 now. The only other time where the ages of more than two of the children were factors of Mum's age was when Tracey was half what Tammy is now, so Tammy must be 18 now.

In the second multiple-factors-of-Mum's-age year, Tracey was therefore 9. When Tony was born, Tracey was either 3, 2 or 1, giving possible time differences of 6, 7 or 8 years, which correspond to Tony's age in the second multiple-factors-of-Mum's-age year. It also means Mum was 36, 37 or 38 that year. 37 is prime and 38 has factors of 1, 2 and 19, and as Tony was at least 6, it means that Mum must have been 36 that year.

Going back to when Tony was born we have that Mum was 30, Tommy was 10, Timmy was 6, Tracey was 3, and as the middle child Tammy must have been 5. If Tammy is 18 now then Tony is 13, Tracey is 16, Timmy is 19, Tommy is 23 and Mum is 43.

Solution by Mace4:

Listing 2.15: Fragment from *mathPuzzle4.out*

```
interpretation( 51, [number=1, seconds=13], [

        function(Mother, [43 ]),
```

```
        function(Tammy, [18 ]),
        function(Timmy, [19 ]),
        function(Tommy, [23 ]),
        function(Tony, [13 ]),
        function(Tracey, [16 ]),

        function(n1, [ 7 ]),
        function(n2, [ 7 ])
]).
```

2.5 Cypher Puzzles

2.5.1 Cypher Puzzle 1 - Mirrored Number Multiplication

Problem: ABCD × E = DCBA

(Replace letters with digits and have the answer be true. A,B,C,D and E are all different digits.)

Solution: 2178 × 4 = 8712

Solution by Mace4:

Listing 2.16: Fragment from *cypherPuzzle1.out*

```
interpretation( 1001, [number=1, seconds=1], [

        function(A, [ 2 ]),
        function(B, [ 1 ]),
        function(C, [ 7 ]),
        function(D, [ 8 ]),
        function(E, [ 4 ])
]).
```

2.5.2 Cypher Puzzle 2 - Raising to a Certain Power

Problem: Solve the following (D and E are two different digits):

$(DD)^E$= DEED

Solution: $11^3 = 1331$

Solution by Mace4:

Listing 2.17: Fragment from *cypherPuzzle2.out*

```
interpretation( 1400, [number=1, seconds=0], [

        function(D, [ 1 ]),
        function(E, [ 3 ]),
```

```
5
6           function(n1, [11 ]),
7           function(n2, [1331 ])
8   ]).
```

2.5.3 Cypher Puzzle 3 - Crack the Code (Easy)

Problem: Find the code!

- It has 6 different digits
- Even and odd digits alternate (note: zero is an even number)
- Digits next to each other have a difference greater than one
- When we break the code into three sets of two-digit numbers, then the first and middle are a multiple of the last

What is the code? (more than 1 solution)

Solution: With even and odd digits alternating, and difference greater than one, then the last 2 digits can only be:

03, 05, 07, 09, 14, 16, 18, 25, 27, 29, 30, 36, 38, 41, 47, 49, ... any higher and we can't make multiples for the first or second pair

Let's check for multiples where even and odd digits alternate and have difference greater than one:

03 has 27, 63, 69, 81
07 has 49, 63
09 has 27, 63, 81
18 has 36, 72, 90
25 has none
27 has none ...

No others work following the rules

When we try to make full numbers that follow all the rules we find only these:

692703
816903
496307
816309
903618

Solution by Mace4:

Listing 2.18: Fragment from *cypherPuzzle3.out*

```
1   interpretation( 30, [number=1, seconds=0], [
2
3           function(D1, [ 4 ]),
```

```
4           function(D2, [ 9 ]),
5           function(D3, [ 6 ]),
6           function(D4, [ 3 ]),
7           function(D5, [ 0 ]),
8           function(D6, [ 7 ]),
9
10          relation(isDigit(_), [ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]),
11
12          relation(even(_), [ 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]),
13
14          relation(odd(_), [ 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]),
15
16          relation(neighbour(_,_), [
17                              0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                   1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                   1, 1, 1,
18                              0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                   1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                   1, 1, 1,
19                              1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                   1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                   1, 1, 1,
20                              1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                   1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                   1, 1, 1,
21                              1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
                                   1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                   1, 1, 1,
22                              1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
                                   1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                   1, 1, 1,
23                              1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1,
                                   1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                   1, 1, 1,
24                              1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1,
                                   1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                   1, 1, 1,
25                              1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1,
                                   1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                   1, 1, 1,
26                              1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1,
                                   1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                   1, 1, 1,
27                              1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1,
                                   1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                   1, 1, 1,
28                              1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1,
                                   1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                   1, 1, 1,
29                              1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1,
30          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
                0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1,
31          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
                0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1,
32          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1,
33          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1,
34          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1,
35          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1,
36          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1,
                1, 1, 1,
37          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1,
                1, 1, 1,
38          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1,
                1, 1, 1,
39          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1,
                1, 1, 1,
40          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1,
                1, 1, 1,
41          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1,
                1, 1, 1,
42          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
                1, 1, 1,
43          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
                0, 1, 1,
44          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
                0, 0, 1,
45          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                0, 0, 0,
46          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 0, 0 ])
```

```
]).

interpretation( 30, [number=2, seconds=0], [

        function(D1, [ 6 ]),
        function(D2, [ 9 ]),
        function(D3, [ 2 ]),
        function(D4, [ 7 ]),
        function(D5, [ 0 ]),
        function(D6, [ 3 ]),

        relation(isDigit(_), [ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]),

        relation(even(_), [ 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]),

        relation(odd(_), [ 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]),

        relation(neighbour(_,_), [
                              0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                 1, 1, 1,
                              0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                 1, 1, 1,
                              1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                 1, 1, 1,
                              1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                 1, 1, 1,
                              1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
                                 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                 1, 1, 1,
                              1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
                                 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                 1, 1, 1,
                              1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1,
                                 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                 1, 1, 1,
                              1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1,
                                 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                 1, 1, 1,
                              1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1,
                                 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                 1, 1, 1,
                              1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1,
                                 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                 1, 1, 1,
                              1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1,
                                 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
```

```
    1, 1, 1,
76  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1,
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1,
77  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1,
78  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
    0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1,
79  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
    0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1,
80  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1,
81  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1,
82  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1,
83  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1,
84  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1,
85  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1,
86  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1,
    1, 1, 1,
87  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1,
    1, 1, 1,
88  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1,
    1, 1, 1,
89  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1,
    1, 1, 1,
90  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1,
    1, 1, 1,
91  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1,
    0, 1, 1,
92  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1,
    0, 0, 1,
93  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
```

```
                                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                0, 0, 0,
                            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                1, 0, 0 ])
]).

interpretation( 30, [number=3, seconds=0], [

        function(D1, [ 8 ]),
        function(D2, [ 1 ]),
        function(D3, [ 6 ]),
        function(D4, [ 3 ]),
        function(D5, [ 0 ]),
        function(D6, [ 9 ]),

        relation(isDigit(_), [ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]),

        relation(even(_), [ 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]),

        relation(odd(_), [ 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]),

        relation(neighbour(_,_), [
                            0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                1, 1, 1,
                            0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                1, 1, 1,
                            1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                1, 1, 1,
                            1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                1, 1, 1,
                            1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
                                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                1, 1, 1,
                            1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
                                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                1, 1, 1,
                            1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1,
                                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                1, 1, 1,
                            1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1,
                                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                1, 1, 1,
                            1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1,
                                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                1, 1, 1,
```

```
122 |   1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1,
    |      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    |      1, 1, 1,
123 |   1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1,
    |      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    |      1, 1, 1,
124 |   1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1,
    |      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    |      1, 1, 1,
125 |   1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
    |      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    |      1, 1, 1,
126 |   1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
    |      0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    |      1, 1, 1,
127 |   1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
    |      0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    |      1, 1, 1,
128 |   1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    |      0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    |      1, 1, 1,
129 |   1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    |      1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    |      1, 1, 1,
130 |   1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    |      1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
    |      1, 1, 1,
131 |   1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    |      1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
    |      1, 1, 1,
132 |   1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    |      1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1,
    |      1, 1, 1,
133 |   1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    |      1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1,
    |      1, 1, 1,
134 |   1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    |      1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1,
    |      1, 1, 1,
135 |   1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    |      1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1,
    |      1, 1, 1,
136 |   1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    |      1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1,
    |      1, 1, 1,
137 |   1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    |      1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1,
    |      1, 1, 1,
138 |   1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    |      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
    |      1, 1, 1,
139 |   1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    |      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
```

```
                                0, 1, 1,
                            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
                                0, 0, 1,
                            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                0, 0, 0,
                            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                1, 0, 0 ])
]).

interpretation( 30, [number=4, seconds=0], [

        function(D1, [ 8 ]),
        function(D2, [ 1 ]),
        function(D3, [ 6 ]),
        function(D4, [ 9 ]),
        function(D5, [ 0 ]),
        function(D6, [ 3 ]),

        relation(isDigit(_), [ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]),

        relation(even(_), [ 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]),

        relation(odd(_), [ 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]),

        relation(neighbour(_,_), [
                            0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                1, 1, 1,
                            0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                1, 1, 1,
                            1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                1, 1, 1,
                            1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                1, 1, 1,
                            1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
                                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                1, 1, 1,
                            1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
                                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                1, 1, 1,
                            1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1,
                                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                1, 1, 1,
                            1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1,
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1,
169             1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1,
170             1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1,
171             1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1,
172             1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1,
173             1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1,
174             1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
                0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1,
175             1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
                0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1,
176             1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1,
177             1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1,
178             1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1,
179             1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1,
180             1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1,
                1, 1, 1,
181             1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1,
                1, 1, 1,
182             1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1,
                1, 1, 1,
183             1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1,
                1, 1, 1,
184             1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1,
                1, 1, 1,
185             1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1,
                1, 1, 1,
```

```
186                             1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                   1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
                                   1, 1, 1,
187                             1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                   1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
                                   0, 1, 1,
188                             1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                   1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
                                   0, 0, 1,
189                             1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                   1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                   0, 0, 0,
190                             1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                   1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                   1, 0, 0 ])
191 ]).
192
193 interpretation( 30, [number=5, seconds=0], [
194
195         function(D1, [ 9 ]),
196         function(D2, [ 0 ]),
197         function(D3, [ 3 ]),
198         function(D4, [ 6 ]),
199         function(D5, [ 1 ]),
200         function(D6, [ 8 ]),
201
202         relation(isDigit(_), [ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]),
203
204         relation(even(_), [ 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]),
205
206         relation(odd(_), [ 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]),
207
208         relation(neighbour(_,_), [
209                             0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                   1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                   1, 1, 1,
210                             0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                   1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                   1, 1, 1,
211                             1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                   1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                   1, 1, 1,
212                             1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                   1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                   1, 1, 1,
213                             1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
                                   1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                   1, 1, 1,
214                             1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
                                   1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
```

```
    1, 1, 1,
215 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1,
216 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1,
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1,
217 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1,
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1,
218 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1,
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1,
219 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1,
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1,
220 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1,
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1,
221 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1,
222 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
    0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1,
223 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
    0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1,
224 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1,
225 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1,
226 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1,
227 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1,
228 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1,
    1, 1, 1,
229 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1,
    1, 1, 1,
230 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1,
    1, 1, 1,
231 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1,
    1, 1, 1,
232 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
```

```
                                    1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1,
                                    1, 1, 1,
233                             1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                    1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1,
                                    1, 1, 1,
234                             1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
                                    1, 1, 1,
235                             1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
                                    0, 1, 1,
236                             1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
                                    0, 0, 1,
237                             1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                    0, 0, 0,
238                             1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                    1, 0, 0 ])
239 ]).
```

2.5.4 Cypher Puzzle 4 - Digits with Properties

Problem: There is a ten-digit mystery number (no leading 0), represented by ABCDEFGHIJ, where each numeral, 0 through 9, is used once. Given the following clues, what is the number?

1) Digit A is either a square number or a triangle number, but not both.
2) Digit B is either an even number or a cube number, but not both.
3) Digit C is either a cube number or a triangle number, but not both.
4) Digit D is either an odd number or a square number, but not both.
5) Digit E is either an odd number or a cube number, but not both.
6) Digit F is either an odd number or a triangle number, but not both.
7) Digit G is either an odd number or a prime number, but not both.
8) Digit H is either an even number or a square number, but not both.
9) Digit I is either a square number or a cube number, but not both.
10) Digit J is either a prime number or a triangle number, but not both.
11) $A < B$, $C < D$, $E < F$, $G < H$, $I < J$
12) $A + B + C + D + E < F + G + H + I + J$

Solution: 3605891247

A = 3, B = 6, C = 0, D = 5, E = 8, F = 9, G = 1, H = 2, I = 4, J = 7

1) 3 is a triangle number, but not a square number.
2) 6 is an even number, but not a cube number.
3) 0 is a cube number, but not a triangle number.
4) 5 is an odd number, but not a square number.
5) 8 is a cube number, but not an odd number.

6) 9 is an odd number, but not a triangle number.
7) 1 is an odd number, but not a prime number.
8) 2 is an even number, but not a square number.
9) 4 is a square number, but not a cube number.
10) 7 is a prime number, but not a triangle number.
11) $3 < 6$, $0 < 5$, $8 < 9$, $1 < 2$, $4 < 7$
12) $3 + 6 + 0 + 5 + 8 = 22$, $9 + 1 + 2 + 4 + 7 = 23$, $22 < 23$

A) First, determine what two subsets of numbers qualify for each digit. However, each digit can only be in one of the subsets, not both. B) Digit A can not be 0 (a leading zero is not allowed). C) Next, eliminate those numbers that do not satisfy Condition 11. Digit I can only be 4. Digit A can only be 3. Digit B can only be 6. Digit C can only be 0. D) Digit D and Digit J must be some combination of 5 and 7. Digit E can only be 8. Digit F can only be 9. Digit G can only be 1. Digit H can only be 2. E) To satisfy Condition 12, Digit D is 5 and Digit J is 7.

Solution by Mace4:

Listing 2.19: Fragment from *cypherPuzzle4.out*

```
interpretation( 30, [number=1, seconds=0], [

        function(A, [ 3 ]),
        function(B, [ 6 ]),
        function(C, [ 0 ]),
        function(D, [ 5 ]),
        function(E, [ 8 ]),
        function(F, [ 9 ]),
        function(G, [ 1 ]),
        function(H, [ 2 ]),
        function(I, [ 4 ]),
        function(J, [ 7 ]),

        relation(cube(_), [ 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]),

        relation(even(_), [ 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]),

        relation(odd(_), [ 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]),

        relation(prime(_), [ 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]),

        relation(square(_), [ 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]),

        relation(triangle(_), [ 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ])
]).
```

2.5.5 Cypher Puzzle 5 - Interconnected Digits

Problem: There is a ten-digit mystery number (no leading 0), represented by ABCDEFGHIJ, where each numeral, 0 through 9, is used once. Given the following clues, what is the number?

1) Either A = B / 3 or A = G + 3.
2) Either B = I - 4 or B = E + 4.
3) Either C = J + 2 or C = F * 3.
4) Either D = G * 4 or D = E / 3.
5) Either E = J - 1 or E = D / 4.
6) Either F = B * 2 or F = A - 4.
7) Either G = F + 1 or G = I - 3.
8) Either H = A / 2 or H = C * 3.
9) Either I = H + 3 or I = D / 2.
10) Either J = H - 2 or J = C * 2.

Solution: 5038612947

A = 5, B = 0, C = 3, D = 8, E = 6, F = 1, G = 2, H = 9, I = 4, J = 7

1) A = G + 3 = 2 + 3 = 5.
2) B = I - 4 = 4 - 4 = 0.
3) C = F * 3 = 1 * 3 = 3.
4) D = G * 4 = 2 * 4 = 8.
5) E = J - 1 = 7 - 1 = 6.
6) F = A - 4 = 5 - 4 = 1.
7) G = F + 1 = 1 + 1 = 2.
8) H = C * 3 = 3 * 3 = 9.
9) I = D / 2 = 8 / 2 = 4.
10) J = H - 2 = 9 - 2 = 7.

A) Check each letter for any value that it can not be on the left side of an equation. For example, A can not be 0, as B would also have to be 0 or G would have to be negative. On the first pass through the letters, the following can be determined: A not 0, C not 0, C not 1, D not 0, D not 5, D not 6, D not 7, D not 9, E not 9, F not 7, F not 9, G not 8 (because F not 7), H not 0, H not 5, H not 7, H not 8, I not 0, I not 3 (because H not 0 and D not 6), I not 8 (because H not 5), J not 3 (because H not 5), J not 5 (because H not 7), and J not 9.

B) Make a second pass through the letters. The following can be determined: C not 5, C not 7, D not 3, E not 4, E not 8, and G not 0.

C) Make a third pass. The following can be determined: B not 8.

D) For D, only four numbers are still valid: 1, 2, 4, and 8. Plug each of these numbers into D and check the resulting flow of equations. If D = 1, then the resulting flow will lead to J having to be both 4 and 0. If D = 2, then the resulting flow will lead to 2 also being assigned to G. If D = 4, then the resulting flow will lead to 4 also being assigned to either A or I. If D = 8, then G = 2, F = 1, and A = 5.

E) For E, four numbers are still valid: 0, 3, 6, and 7. Plug each number into E and check the resulting flow. If E = 0, then the flow will lead to J being assigned 1, which has already been assigned to F. If E = 3, then the flow will lead to C being assigned 2, which has already been assigned to G. If E = 7, then the flow will lead to J being assigned 8, which has already been assigned to D. If E = 6, then J = 7, H = 9, and C = 3.

F) The only remaining value for I is 4, which leaves 0 being assigned to B.

Solution by Mace4:

Listing 2.20: Fragment from *cypherPuzzle5.out*

```
interpretation( 37, [number=1, seconds=0], [

        function(A, [ 5 ]),
        function(B, [ 0 ]),
        function(C, [ 3 ]),
        function(D, [ 8 ]),
        function(E, [ 6 ]),
        function(F, [ 1 ]),
        function(G, [ 2 ]),
        function(H, [ 9 ]),
        function(I, [ 4 ]),
        function(J, [ 7 ])
]).
```

2.5.6 Cypher Puzzle 6 - Crack the Code (Medium)

Problem: Find a 10-digit number where the first digit is how many zeros are in the number, the second digit is how many 1s are in the number etc. until the tenth digit which is how many 9s are in the number.

Solution: 6210001000

Solution by Mace4:

Listing 2.21: Fragment from *cypherPuzzle6.out*

```
interpretation( 10, [number=1, seconds=10451], [

        function(c(_), [ 6, 2, 1, 0, 0, 0, 1, 0, 0, 0 ]),
        function(h(_), [ 6, 2, 1, 0, 0, 0, 1, 0, 0, 0 ]),

        function(equal(_,_), [
                              0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                              0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
                              0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                              0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                              0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                              0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                              0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                              0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                              0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                              0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]),

        function(f(_,_), [
                              0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
                              0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                              0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
                              1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
                            1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                            1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                            0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
                            1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                            1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                            1, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]),

        function(greater(_,_), [
                            1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                            1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
                            1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
                            1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
                            1, 1, 1, 1, 1, 0, 0, 0, 0, 0,
                            1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
                            1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
                            1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
                            1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
                            1, 1, 1, 1, 1, 1, 1, 1, 1, 1 ]),

        function(less(_,_), [
                            1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                            0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                            0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
                            0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
                            0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
                            0, 0, 0, 0, 0, 1, 1, 1, 1, 1,
                            0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
                            0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
                            0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
                            0, 0, 0, 0, 0, 0, 0, 0, 0, 1 ])
]).
```

2.5.7 Cypher Puzzle 7 - Crack the Code (Hard)

Problem: Can you crack the code?

6 8 2 - One digit is correct and well placed
6 1 4 - One digit is correct but wrong place
2 0 6 - Two digits are correct but wrong places
7 3 8 - Nothing is correct
8 7 0 - One digit is correct but wrong place

What is the code?

Solution: The code is 0 4 2.

Let the statements be A, B, C, D and E.

Now from statements A, B, D we know that one of the digit is 2 and well placed (third place).

From statements C and E we know that another digit is 0 and its place is first in the code.

Finally from statement B we now know that 4 is the remaining digit of the code and its place is second in the code.

Hence, the code is 0 4 2.

Solution by Mace4:

Listing 2.22: Fragment from *cypherPuzzle7.out*

```
interpretation( 10, [number=1, seconds=0], [

        function(A, [ 0 ]),
        function(B, [ 4 ]),
        function(C, [ 2 ]),

        relation(orAB(_), [ 1, 0, 0, 0, 1, 0, 0, 0, 0, 0 ]),
        relation(orAC(_), [ 1, 0, 1, 0, 0, 0, 0, 0, 0, 0 ]),
        relation(orBC(_), [ 0, 0, 1, 0, 1, 0, 0, 0, 0, 0 ]),

        relation(notTwoDigitsAB(_,_), [
                        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                        0, 1, 1, 1, 0, 1, 1, 1, 1, 1,
                        0, 1, 1, 1, 0, 1, 1, 1, 1, 1,
                        0, 1, 1, 1, 0, 1, 1, 1, 1, 1,
                        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                        0, 1, 1, 1, 0, 1, 1, 1, 1, 1,
                        0, 1, 1, 1, 0, 1, 1, 1, 1, 1,
                        0, 1, 1, 1, 0, 1, 1, 1, 1, 1,
                        0, 1, 1, 1, 0, 1, 1, 1, 1, 1,
                        0, 1, 1, 1, 0, 1, 1, 1, 1, 1 ]),
        relation(notTwoDigitsAC(_,_), [
                        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                        0, 1, 0, 1, 1, 1, 1, 1, 1, 1,
                        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                        0, 1, 0, 1, 1, 1, 1, 1, 1, 1,
                        0, 1, 0, 1, 1, 1, 1, 1, 1, 1,
                        0, 1, 0, 1, 1, 1, 1, 1, 1, 1,
                        0, 1, 0, 1, 1, 1, 1, 1, 1, 1,
                        0, 1, 0, 1, 1, 1, 1, 1, 1, 1,
                        0, 1, 0, 1, 1, 1, 1, 1, 1, 1,
                        0, 1, 0, 1, 1, 1, 1, 1, 1, 1 ]),
        relation(notTwoDigitsBC(_,_), [
                        1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                        1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                        1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                        1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                        1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                        1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                        1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                        1, 1, 0, 1, 0, 1, 1, 1, 1, 1 ]),

        relation(nothingIsCorrect(_,_,_), [
```

```
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
                        0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                        0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                        0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                        0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                        0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                        0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                        0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                        0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                        0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                        0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                        0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                        0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                        0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                        0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                        0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                        0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                        0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                        0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                        0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                        0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                        0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                        0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                        0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                        0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                        0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                        0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                        0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                        0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                        0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                        0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                        0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                        0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                        0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                        0, 1, 0, 1, 0, 1, 1, 1, 1, 1 ]),

    relation(oneDigitIsCorrectAndWellPlaced(_,_,_), [
                        1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                        1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                        1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
```

```
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 1, 1, 1, 1, 1, 1, 1,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 1, 1, 1, 1, 1, 1, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 1, 1, 1, 1, 1, 1, 1,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
```

```
                        0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                        0, 1, 0, 1, 1, 1, 1, 1, 1, 1,
                        0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                        0, 1, 0, 1, 1, 1, 1, 1, 1, 1,
                        0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                        0, 1, 0, 1, 1, 1, 1, 1, 1, 1,
                        0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 1, 0, 0, 0, 0, 0, 0, 0 ]),

    relation(oneDigitIsCorrectAndWronglyPlaced(_,_,_), [
                        1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                        1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                        1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                        1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                        1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                        1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 1, 1, 1, 0, 1, 1, 1, 1, 1,
1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 1, 1, 0, 1, 1, 1, 1, 1,
0, 1, 1, 1, 0, 1, 1, 1, 1, 1,
0, 1, 1, 1, 0, 1, 1, 1, 1, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 1, 1, 0, 1, 1, 1, 1, 1,
0, 1, 1, 1, 0, 1, 1, 1, 1, 1,
0, 1, 1, 1, 0, 1, 1, 1, 1, 1,
0, 1, 1, 1, 0, 1, 1, 1, 1, 1,
0, 1, 1, 1, 0, 1, 1, 1, 1, 1,
1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 1, 1, 1, 0, 1, 1, 1, 1, 1,
1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 1, 1, 1, 1, 1, 1, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 1, 1, 1, 1, 1, 1, 1,
0, 1, 0, 1, 1, 1, 1, 1, 1, 1,
0, 1, 0, 1, 1, 1, 1, 1, 1, 1,
0, 1, 0, 1, 1, 1, 1, 1, 1, 1,
0, 1, 0, 1, 1, 1, 1, 1, 1, 1,
0, 1, 0, 1, 1, 1, 1, 1, 1, 1,
0, 1, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 1, 1, 1, 0, 1, 1, 1, 1, 1,
1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
```

```
                        1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                        0, 1, 1, 1, 0, 1, 1, 1, 1, 1,
                        1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                        0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                        1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                        1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                        1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                        1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                        1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                        1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                        1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                        0, 1, 1, 1, 0, 1, 1, 1, 1, 1,
                        1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                        0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                        1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                        1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                        1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                        1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                        1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                        1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                        1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                        1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                        0, 1, 1, 1, 0, 1, 1, 1, 1, 1,
                        1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                        0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                        1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                        1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                        1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                        1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                        1, 0, 0, 0, 1, 0, 0, 0, 0, 0 ]),

    relation(twoDigitsAreCorrectAndWellPlaced(_,_,_), [
                        0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                        1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
                        0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                        0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
```

```
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]),

         relation(twoDigitsAreCorrectAndWronglyPlaced(_,_,_), [
                                1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ])
]).
```

2.6 Logic Puzzles

2.6.1 Logic Puzzle 1 - Lies and Ages

Problem: Alex, Brook, Cody, Dusty, and Erin recently found out that all of their birthdays were on the same day, though they are different ages.

On their mutual birthday, they were jabbering away, flapping their gums about their recent discovery. And, lucky me, I was there. Some of the things that I overheard were...

- Dusty said to Brook: "I'm nine years older than Erin."
- Erin said to Brook: "I'm seven years older than Alex."
- Alex said to Brook: "Your age is exactly 70% greater than mine."
- Brook said to Cody: "Erin is younger than you."
- Cody said to Dusty: "The difference between our ages is six years."
- Cody said to Alex: "I'm ten years older than you."
- Cody said to Alex: "Brook is younger than Dusty."

• Brook said to Cody: "The difference between your age and Dusty's is the same as the difference between Dusty's and Erin's."

Since I knew these people – and how old they were, I knew that they were not telling the whole truth.

After thinking about it, I realized that when one of them spoke to someone older, everything they said was true, but when speaking to someone younger, everything they said was false.

How old is each person?

(Replace letters with digits and have the answer be true. A,B,C,D and E are all different digits.)

Solution:

Alex is 30
Brook is 51
Cody is 55
Dusty is 46
Erin is 37

REASONING

Let the ages and names of Alex, Brook, Cody, Dusty and Erin be A, B, C, D and E.
C says to A, that C = A + 10. If C were younger than A, that would be lying, so C must be older than A. (But still lying.)
We have A < C.
C says to A, that B < D. As C > A, C is lying, so B > D.
We have A < C, D < B.
D says to B, that D = E + 9. As D < B, D is telling the truth, so D > E.
We have A < C, E < D < B, D = E + 9.
E says to B, that E = A + 7. As E < B, E is telling the truth, so E > A.
We have A < C, A < E < D < B, D = E + 9, E = A + 7.
Since D = E + 9 and E = A + 7, D = A + 7 + 9 = A + 16.
We have A < C, A < E < D < B, D = E + 9 = A + 16, E = A + 7.
B says to C, that E < C. If B > C then B would be lying, so then E > C, and then A < C < E < D < B. However, C says to D, that C = D ± 6; since C < D, this gives C = D - 6. However, we have E = D - 9, which would make E < C, giving a contradiction. The assumption that B > C is therefore false, so B < C.
We have A < E < D < B < C, D = E + 9 = A + 16, E = A + 7.
A says to B, that B = (17/10)*A. As A < B, A is telling the truth.
We have A < E < D < B < C, B = (17/10)*A, D = E + 9 = A + 16, E = A + 7.
B says to C, that $|C - D| = |D - E|$? $|C - D| = 9$. As B < C, B is telling the truth, so C = D + 9. As D = A + 16, C = A + 16 + 9 ? C = A + 25.
We have A ¡ E ¡ D ¡ B ¡ C, B = (17/10)A, C = A + 25, D = A + 16, E = A + 7.
Using D < B < C, we have A + 16 < (17/10)*A < A + 25 ? 16 < (7/10)*A < 25 ? 160/7 < A < 250/7 ? 22 + 6/7 < A < 35 + 5/7. Since B and A must both be whole numbers, and B = (17/10)*A ? B - A = (7/10)A, (7/10)*A must be a whole number. Hence A must be divisible by 10. The only whole number fitting 22 + 6/7 < A < 35 + 5/7 is A = 30.

We have A = 30, B = (17/10)*A, C = A + 25, D = A + 16, E = A + 7.
Hence A = 30, B = 51, C = 55, D = 46, E = 37.

A VERBAL DESCRIPTION OF THE REASONING

Cody tells Alex she's older than her by 10 years. If Cody is younger, she's lying, and that's impossible, so Cody must be older than Alex, just not by 10 years.

FACT: Cody is older than Alex (but not by 10 years).

Cody also lies to (younger) Alex that Brook is younger than Dusty.

FACT: Dusty is older than Brook.

Dusty tells the truth to (older) Brook that she's 9 years older than Erin.

FACT: Dusty is 9 years older than Erin.

Erin tells the truth to (older) Brook that she's 7 years older than Alex.

FACT: Erin is 7 years older than Alex.

Alex tells the truth to (older) Brook that Brook's age is 70% greater than her own. For Brook's age to be a whole number, Alex's age must be a multiple of 10. Since Brook is older than Dusty, and Dusty is 7 + 9 = 16 years older than Alex, that means Brook has to be more than 16 years older than Alex. The lowest multiple of 7 greater than 16 is 21.

FACT: Alex is at least 30 years old (and definitely a multiple of 10).

At this point, Brook appears to be the oldest, lying lady. Let's assume that, and see if it works.

In that case, Cody is lying to Dusty that the difference in their ages is 6 years, but Brook tells the truth to (older) Cody that the difference between Cody's age and Dusty's is the same as the difference between Dusty's and Erin's, namely, 9 years. Let's test this scenario, assuming Alex's age is 30. Then we get, from youngest to oldest:

TESTING: Alex = 30, Erin = 37, Dusty = 46, Brook = 51, Cody = 55

Checking all statements and the age relations shows that this is an answer. Is this the only answer?

If Alex's age was 40, then Brook's age would be 68, and Cody's age would be 65, so Cody would not be the oldest, and that would be a fatal flaw. If Alex is older than 30, Brook is older than Cody, and Cody is not the oldest. Hence, it must have been the only answer.

Solution by Mace4:

Listing 2.23: Fragment from *logicPuzzle1.out*

```
interpretation( 56, [number=1, seconds=2], [

        function(Alex, [30 ]),
        function(Brook, [51 ]),
        function(Cody, [55 ]),
        function(Dusty, [46 ]),
        function(Erin, [37 ])
]).
```

2.6.2 Logic Puzzle 2 - Groups of Schoolgirls

Problem: In a boarding school there are fifteen schoolgirls who always take their daily walks in groups of three.

How can it be arranged so that each schoolgirl walks in a group with two different companions every day for a week (7 days)?

Solution:
Give the girls letters A to O - one possible schedule is then:

Week Day	Group 1	Group 2	Group 3	Group 4	Group 5
Monday	ABE	CDG	HIL	JKN	MOF
Tuesday	BCF	DEH	IJM	KLO	NAG
Wednesday	EFI	GHK	LMA	NOC	BDJ
Thursday	CEK	DFL	GIO	HJA	MNB
Friday	EGM	FHN	IKB	JLC	OAD
Saturday	KMD	LNE	OBH	ACI	FGJ
Sunday	AFK	BGL	CHM	DIN	EJO

The famouse Schoolgirls Problem was first posed by Reverend Thomas Kirkman in 1857. It led to a new branch of mathematics called Combinatorics.

Solution by Mace4:
Mace4 found a lot of solutions (I stopped it after a few seconds), and I am showing here only one of them (I don't think it's the same as the one above), but in the *.out* file, there are over 2500 correct, distinct models (and there would have been even more If I hadn't stopped the execution of the program).

Listing 2.24: Fragment from *logicPuzzle2.out*

```
1
2  interpretation( 15, [number=1, seconds=0], [
3
4          function(A, [ 0 ]),
5          function(B, [ 1 ]),
6          function(C, [ 2 ]),
7          function(D, [ 3 ]),
8          function(E, [ 4 ]),
9          function(F, [ 5 ]),
10         function(G, [ 6 ]),
11         function(H, [ 7 ]),
12         function(I, [ 8 ]),
13         function(J, [ 9 ]),
14         function(K, [10 ]),
15         function(L, [11 ]),
16         function(M, [12 ]),
17         function(N, [13 ]),
18         function(O, [14 ]),
```

```
19
20          relation(walkTogetherFri(_,_), [
21                                  0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0,
                                       0,
22                                  0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
                                       0,
23                                  0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
                                       0,
24                                  0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0,
                                       0,
25                                  0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
                                       0,
26                                  1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                                       0,
27                                  0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
                                       0,
28                                  0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
                                       0,
29                                  0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0,
                                       0,
30                                  0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                                       1,
31                                  0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                                       1,
32                                  0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                       0,
33                                  1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                       0,
34                                  0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0,
                                       0,
35                                  0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
                                       0 ]),
36
37          relation(walkTogetherMon(_,_), [
38                                  0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                                       0,
39                                  1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                       0,
40                                  0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                       0,
41                                  0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                       0,
42                                  0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
                                       0,
43                                  0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                                       0,
44                                  0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0,
                                       0,
45                                  0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
                                       0,
46                                  1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
                                       0,
47                                  0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
```

```
                                0,
48                          0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
                                0,
49                          0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
                                0,
50                          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
                                1,
51                          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
                                1,
52                          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
                                0 ]),
53
54          relation(walkTogetherSat(_,_), [
55                          0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1,
                                0,
56                          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
                                0,
57                          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
                                0,
58                          0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
                                0,
59                          0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
                                0,
60                          0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0,
                                0,
61                          1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                                0,
62                          0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                1,
63                          0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                                0,
64                          0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
                                0,
65                          0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0,
66                          0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
                                1,
67                          0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0,
68                          1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                0,
69                          0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
                                0 ]),
70
71          relation(walkTogetherSun(_,_), [
72                          0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
                                1,
73                          0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
                                0,
74                          0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0,
                                0,
75                          0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                                1,
```

```
                        0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
                           0,
                        0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
                           0,
                        0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
                           0,
                        1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
                           0,
                        0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
                           0,
                        0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                           0,
                        0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1,
                           0,
                        0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0,
                           0,
                        0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0,
                           0,
                        0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
                           0,
                        1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                           0 ]),

    relation(walkTogetherThu(_,_), [
                        0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
                           0,
                        0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                           1,
                        0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,
                           0,
                        0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0,
                           0,
                        1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                           0,
                        0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                           1,
                        0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
                           0,
                        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
                           0,
                        0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                           0,
                        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
                           0,
                        0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0,
                           0,
                        1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                           0,
                        0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0,
                           0,
                        0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                           0,
                        0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
                        0 ]),

    relation(walkTogetherTue(_,_), [
                    0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
                        0,
                    0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
                        0,
                    1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                        0,
                    0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
                        0,
                    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
                        0,
                    0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
                        0,
                    0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                        1,
                    0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
                        0,
                    0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
                        1,
                    1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                        0,
                    0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
                        0,
                    0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
                        0,
                    0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
                        0,
                    0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0,
                        0,
                    0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
                        0 ]),

    relation(walkTogetherWed(_,_), [
                    0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
                        0,
                    0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
                        0,
                    0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
                        1,
                    1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
                        0,
                    0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
                        1,
                    0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
                        0,
                    0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0,
                        0,
                    0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                        0,
                    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
                        0,
```

```
                                0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1,
                                   0,
                                1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                   0,
                                0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
                                   0,
                                0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0,
                                   0,
                                0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
                                   0,
                                0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                   0 ])
]).
```

Chapter 3

A3: Planning

Bibliography

Berkeley's version of the Pac-Man game
AI course slides
Video Tutorials on YouTube
Various tutorials found on the internet
Additional information about heuristics
Tricky Heights 1, 2 & 3
Einstein Puzzle 1
Einstein Puzzle 2
Einstein Puzzle 3
Sam Loyd Puzzle 1
Sam Loyd Puzzle 2
Sam Loyd Puzzle 3
Math Puzzle 1
Math Puzzle 2
Math Puzzle 3
Math Puzzle 4
Cypher Puzzle 1
Cypher Puzzle 2
Cypher Puzzle 3
Cypher Puzzle 4
Cypher Puzzle 5
Cypher Puzzle 6
Cypher Puzzle 7
Logic Puzzle 1
Logic Puzzle 2

Appendix A

Your original code

Don't be a cheater! Cheating affects your colleagues, scholarships and a lot more. This section should contain only code developed by you, without any line re-used from other sources. This section helps me to correctly evaluate your amount of work and results obtained.

A.1 Search

A.1.1 Search Agent Problem

Random Search

Listing A.1: Random Search Algorithm

```
def randomSearch(problem):
    current_state = problem.getStartState()
    list1 = []
    while (problem.isGoalState(current_state) == False):
        successors = problem.getSuccessors(current_state)
        random_successor = random.choice(successors)
        current_state = random_successor[0]
        list1.append(random_successor[1])

    return list1
```

Depth First Search

Listing A.2: Node Class

```
class Node:
    def __init__ (self, triplet, parent):
        self.triplet = triplet
        self.parent = parent
    def getTriplet(self):
        return self.triplet
    def getParent(self):
        return self.parent
    def __cmp__(self, that):
        return cmp(self.triplet, that.getTriplet())
```

Listing A.3: DFS Algorithm

```
def depthFirstSearch(problem):
    start_position = problem.getStartState()

    current_node = Node((start_position, None, 0), None)
    stack = [current_node]
    visited = []

    while stack:
        current_node = stack.pop()
        current_position = current_node.getTriplet()[0]

        if (problem.isGoalState(current_position) == True):
            break

        if (current_node not in visited):
            visited.append(current_node)
            successors = problem.getSuccessors2(current_position)
            for successor in successors:
                successorNode = Node(successor, current_node)
                if (successorNode not in visited):
                    stack.append(successorNode)

    list2 = []
    while current_node.getParent():
        list2.append(current_node.getTriplet()[1])
        current_node = current_node.getParent()

    list2.reverse()
    return list2
```

Breadth First Search

Listing A.4: BFS Algorithm

```
def breadthFirstSearch(problem):
    start_position = problem.getStartState()

    current_node = Node((start_position, None, 0), None)
    queue = [current_node]
    visited = []

    while queue:
        current_node = queue.pop(0)
        current_position = current_node.getTriplet()[0]

        if (problem.isGoalState(current_position) == True):
            break

        if (current_node not in visited):
```

```
            visited.append(current_node)
            successors = problem.getSuccessors2(current_position)
            for successor in successors:
                successorNode = Node(successor, current_node)
                if (successorNode not in visited):
                    queue.append(successorNode)

    list3 = []
    while current_node.getParent():
        list3.append(current_node.getTriplet()[1])
        current_node = current_node.getParent()

    list3.reverse()
    return list3
```

Iterative Deepening Search

Listing A.5: Depth Limited Search

```
def depthLimitedSearch(problem, start_position, depth):
    """Variant of DFS, but has a certain depth until which is
       allowed to go"""
    """This is only a helper function for the IDS algorithm"""

    current_node = Node((start_position, None, 0), None)
    stack = [current_node]
    visited = []
    foundGoal = False

    while stack:
        current_node = stack.pop()
        current_position = current_node.getTriplet()[0]

        if (problem.isGoalState(current_position) == True):
            foundGoal = True
            break

        if (current_node not in visited):
            visited.append(current_node)
            depth -= 1
            if (depth < 0):
                return None
            successors = problem.getSuccessors2(current_position)
            for successor in successors:
                successorNode = Node(successor, current_node)
                if (successorNode not in visited):
                    stack.append(successorNode)

    list3 = []
    if (foundGoal == True):
```

```
        while current_node.getParent():
            list3.append(current_node.getTriplet()[1])
            current_node = current_node.getParent()

        list3.reverse()

    return list3
```

Listing A.6: IDS Algorithm

```
def iterativeDeepeningSearch(problem):
    start_position = problem.getStartState()

    depth = 1
    list3 = []
    while not list3:
        list3 = depthLimitedSearch(problem, start_position, depth
            )
        depth += 1

    return list3
```

Uniform Cost Search

Listing A.7: Node2 Class

```
class Node2:
    def __init__ (self, triplet, parent, cost):
        self.triplet = triplet
        self.parent = parent
        self.cost = cost
    def getTriplet(self):
        return self.triplet
    def getParent(self):
        return self.parent
    def getCost(self):
        return self.cost
    def __cmp__(self, other):
        return cmp(self.triplet, other.getTriplet())
    def __eq__(self, other):
        return self.triplet == other.getTriplet()
```

Listing A.8: UCS Algorithm

```
def uniformCostSearch(problem):
    start_position = problem.getStartState()

    current_node = Node2((start_position, None, 0), None, 0)
    priority_queue = [current_node]
    visited = []

```

```
    while priority_queue:

        priority_queue = sorted(priority_queue, key=lambda x: x.
           getCost())
        current_node = priority_queue.pop(0)
        current_position = current_node.getTriplet()[0]

        if (problem.isGoalState(current_position) == True):
            break

        if (current_node not in visited):
            visited.append(current_node)
            successors = problem.getSuccessors2(current_position)
            for successor in successors:
                successorNode2 = Node2(successor, current_node,
                   current_node.getCost() + successor[2])
                if (successorNode2 not in visited):
                    priority_queue.append(successorNode2)

    list4 = []
    while current_node.getParent():
        list4.append(current_node.getTriplet()[1])
        current_node = current_node.getParent()

    list4.reverse()
    return list4
```

A* Search

Listing A.9: Node3 Class

```
class Node3:
    def __init__ (self, triplet, parent, cost, costWithHeuristic)
       :
        self.triplet = triplet
        self.parent = parent
        self.cost = cost
        self.costWithHeuristic = costWithHeuristic
    def getTriplet(self):
        return self.triplet
    def getParent(self):
        return self.parent
    def getCost(self):
        return self.cost
    def getCostWithHeuristic(self):
        return self.costWithHeuristic
    def __cmp__(self, that):
        return cmp(self.triplet, that.getTriplet())
    def __eq__(self, other):
        return self.triplet == other.getTriplet()
```

Listing A.10: A* Search Algorithm

```
def aStarSearch(problem, heuristic=nullHeuristic):
    start_position = problem.getStartState()

    current_node = Node3((start_position, None, 0), None, 0, 0)
    priority_queue = [current_node]
    visited = []

    while priority_queue:

        priority_queue = sorted(priority_queue, key=lambda x: x.
           getCostWithHeuristic())
        current_node = priority_queue.pop(0)
        current_position = current_node.getTriplet()[0]

        if (problem.isGoalState(current_position) == True):
            break

        if (current_node not in visited):
            visited.append(current_node)
            successors = problem.getSuccessors2(current_position)
            for successor in successors:
                successorNode3 = Node3(successor, current_node,
                   current_node.getCost() + successor[2],
                   current_node.getCost() + successor[2] +
                   heuristic(successor[0], problem))
                if (successorNode3 not in visited):
                    priority_queue.append(successorNode3)

    list5 = []
    while current_node.getParent():
        list5.append(current_node.getTriplet()[1])
        current_node = current_node.getParent()

    list5.reverse()
    return list5
```

Heuristics (besides Manhattan and Euclidean)

Listing A.11: Chebyshev Heuristic

```
def chebyshevHeuristic(position, problem, info={}):
    xy1 = position
    xy2 = problem.goal
    dx = abs(xy1[0] - xy2[0])
    dy = abs(xy1[1] - xy2[1])
    return max(dx, dy)
```

Listing A.12: Octile Heuristic

```
def octileHeuristic(position, problem, info={}):
    xy1 = position
    xy2 = problem.goal
    dx = abs(xy1[0] - xy2[0])
    dy = abs(xy1[1] - xy2[1])
    return (dx + dy) - ((2 ** 0.5) - 2) * min(dx, dy)
```

Listing A.13: Bidirectional Manhattan Heuristic

```
def bidirectionalManhattanHeuristic(position, problem, info={}):
    xy1 = position
    xy2 = problem.goal
    xy3 = problem.getStartState()
    h1 = abs(xy1[0] - xy2[0]) + abs(xy1[1] - xy2[1])
    h2 = abs(xy1[0] - xy3[0]) + abs(xy1[1] - xy3[1])
    return min(h1, h2)
```

Listing A.14: Bidirectional Euclidean Heuristic

```
def bidirectionalEuclideanHeuristic(position, problem, info={}):
    xy1 = position
    xy2 = problem.goal
    xy3 = problem.getStartState()
    h1 = ( (xy1[0] - xy2[0]) ** 2 + (xy1[1] - xy2[1]) ** 2 ) **
       0.5
    h2 = ( (xy1[0] - xy3[0]) ** 2 + (xy1[1] - xy3[1]) ** 2 ) **
       0.5
    return min(h1, h2)
```

Listing A.15: Bidirectional Chebyshev Heuristic

```
def bidirectionalChebyshevHeuristic(position, problem, info={}):
    xy1 = position
    xy2 = problem.goal
    xy3 = problem.getStartState()
    dx1 = abs(xy1[0] - xy2[0])
    dy1 = abs(xy1[1] - xy2[1])
    dx2 = abs(xy1[0] - xy3[0])
    dy2 = abs(xy1[1] - xy3[1])
    h1 = max(dx1, dy1)
    h2 = max(dx2, dy2)
    return min(h1, h2)
```

Listing A.16: Bidirectional Octile Heuristic

```
def bidirectionalOctileHeuristic(position, problem, info={}):
    xy1 = position
    xy2 = problem.goal
    xy3 = problem.getStartState()
    dx1 = abs(xy1[0] - xy2[0])
    dy1 = abs(xy1[1] - xy2[1])
    dx2 = abs(xy1[0] - xy3[0])
```

```
    dy2 = abs(xy1[1] - xy3[1])
    h1 = (dx1 + dy1) - ((2 ** 0.5) - 2) * min(dx1, dy1)
    h2 = (dx2 + dy2) - ((2 ** 0.5) - 2) * min(dx2, dy2)
    return min(h1, h2)
```

Weighted A*

Listing A.17: Weighted A* Search Algorithm

```
def weightedAStarSearch(problem, heuristic=nullHeuristic,
   cost_weight=1.0, heuristic_weight=2.0):
    start_position = problem.getStartState()

    current_node = Node3((start_position, None, 0), None, 0, 0)
    priority_queue = [current_node]
    visited = []

    while priority_queue:

        priority_queue = sorted(priority_queue, key=lambda x: x.
           getCostWithHeuristic())
        current_node = priority_queue.pop(0)
        current_position = current_node.getTriplet()[0]

        if (problem.isGoalState(current_position) == True):
            break

        if (current_node not in visited):
            visited.append(current_node)
            successors = problem.getSuccessors2(current_position)
            for successor in successors:
                successorNode3 = Node3(successor, current_node,
                   current_node.getCost() + successor[2],
                   cost_weight * (current_node.getCost() +
                   successor[2]) + heuristic_weight * heuristic(
                   successor[0], problem))
                if (successorNode3 not in visited):
                    priority_queue.append(successorNode3)

    list5 = []
    while current_node.getParent():
        list5.append(current_node.getTriplet()[1])
        current_node = current_node.getParent()

    list5.reverse()
    return list5
```

BEAM Search

Listing A.18: BEAM Search Algorithm

```
def beamSearch(problem, heuristic=nullHeuristic, beta=11):
    start_position = problem.getStartState()

    current_node = Node3((start_position, None, 0), None, 0, 0)
    priority_queue = [current_node]
    visited = []

    found_goal = False
    while priority_queue and not found_goal:

        priority_queue = sorted(priority_queue, key=lambda x: x.
           getCostWithHeuristic())
        while len(priority_queue) > beta:
            priority_queue.pop()
        current_node = priority_queue.pop(0)
        current_position = current_node.getTriplet()[0]

        if (problem.isGoalState(current_position) == True):
            found_goal = True

        if (current_node not in visited):
            visited.append(current_node)
            successors = problem.getSuccessors2(current_position)
            for successor in successors:
                successorNode3 = Node3(successor, current_node,
                   current_node.getCost() + successor[2],
                   current_node.getCost() + successor[2] +
                   heuristic(successor[0], problem))
                if (successorNode3 not in visited):
                    priority_queue.append(successorNode3)

    list6 = []
    while current_node.getParent():
        list6.append(current_node.getTriplet()[1])
        current_node = current_node.getParent()

    list6.reverse()
    return list6
```

Hill Climbing (Random Restart)

Listing A.19: Fitness Function

```
def fitnessFunction(current_node):
    #in this problem, it is just the cost
    return current_node.getCost()
```

Listing A.20: Hill Climbing Search Algorithm (Random Restart)

```
def hillClimbingSearch(problem):
    start_position = problem.getStartState()
    wallList = problem.getAllWalls().asList()
    maximum_coordinate_x = max(wallList)[0]
    maximum_coordinate_y = max(wallList)[1]
    number_of_iterations = 0

    forbidden_positions = [(1, 1)] #add the goal position to the forbidden list of positions from which the hill-climbing algorithm cannot restart
    list7 = []
    foundGoal = False
    while (not foundGoal):

        number_of_iterations += 1
        while list7:
            list7.pop()

        current_node = None
        if number_of_iterations > 1:
            new_starting_position = None
            while True:
                random_x = random.randint(1, maximum_coordinate_x)
                random_y = random.randint(1, maximum_coordinate_y)
                new_starting_position = (random_x, random_y)
                if new_starting_position not in forbidden_positions and new_starting_position not in wallList:
                    break

            current_node = Node2((new_starting_position, None, 0), None, 0)
        else:
            current_node = Node2((start_position, None, 0), None, 0)

        visited = []
        forbidden_positions.append(current_node.getTriplet()[0])

        while True:
            current_position = current_node.getTriplet()[0]
            visited.append(current_node)
            current_fitness = fitnessFunction(current_node)

            if (problem.isGoalState(current_position) == True):
                foundGoal = True
                break

```

```
            best_fitness = 0x3f3f3f3f
            best_successor = None

            successors = problem.getSuccessors2(current_position)
            for successor in successors:
                successorNode = Node2(successor, current_node,
                    current_fitness + successor[2])
                neighbour_fitness = fitnessFunction(successorNode
                    )
                if (neighbour_fitness < best_fitness and
                    neighbour_fitness >= current_fitness and
                    successorNode not in visited):
                     best_fitness = neighbour_fitness
                     best_successor = successorNode
            if not best_successor:
                break
            else:
                current_node = best_successor
                list7.append(current_node.getTriplet()[1])

    return list7
```

Genetic Algorithm

Listing A.21: Population Class

```
class Population:
    def __init__ (self, listOfCoordinates, fitnessScore):
        self.listOfCoordinates = listOfCoordinates
        self.fitnessScore = fitnessScore
    def getListOfCoordinates(self):
        return self.listOfCoordinates
    def setListOfCoordinates(self, listOfCoordinates):
        self.listOfCoordinates = listOfCoordinates
    def getFitnessScore(self):
        return self.fitnessScore
    def setFitnessScore(self, fitnessScore):
        self.fitnessScore = fitnessScore
    def __cmp__(self, that):
        return cmp(self.listOfCoordinates, that.
            getListOfCoordinates())
    def __eq__(self, other):
        return self.listOfCoordinates == other.
            getListOfCoordinates()
```

Listing A.22: Initialize Population Function (creates randomly generated valid paths)

```
def initializePopulation(problem):
    current_position = problem.getStartState()
    list1 = [current_position]
    while (problem.isGoalState(current_position) == False):
```

```
        successors = problem.getSuccessors(current_position)
        random_successor = random.choice(successors)
        current_position = random_successor[0]
        list1.append(current_position)

    population = Population(list1, 0)
    print list1
    return population
```

Listing A.23: Evaluate Population Function (computes the fitness score of a path)

```
def evaluatePopulation(population, goal_position):
    fitness_score = 0
    for individual in population:
        if isinstance(individual, tuple) and len(individual) ==
           2:
            dx = abs(individual[0] - goal_position[0])
            dy = abs(individual[1] - goal_position[1])
            fitness_score += (dx + dy) - ((2 ** 0.5) - 2) * min(
               dx, dy)
        else:
            return 0x3f3f3f3f

    return fitness_score
```

Listing A.24: Common Neighbour Function (returns a common neighbour of two nodes)

```
def commonNeighbour(problem, element1, element2):
    successors1 = problem.getSuccessors(element1)
    successors2 = problem.getSuccessors(element2)
    for successor1 in successors1:
        for successor2 in successors2:
            if successor1[0] == successor2[0]:
                return successor1[0]

    return None
```

Listing A.25: Mutation Function

```
def mutation(child):
    n = len(child)
    for i in range(0, n - 3):
        if child[i] == child[i + 2]:
            new_child = []
            new_child.append(child[:i])
            new_child.append(child[(i + 2):])
            return new_child

    return child
```

Listing A.26: Crossover Function

```
def crossover(problem, parent1, parent2):
    childFinal = Population([], 0x3f3f3f3f)
    child = []

    coordinates1 = parent1.getListOfCoordinates()
    coordinates2 = parent2.getListOfCoordinates()
    n1 = len(coordinates1)
    n2 = len(coordinates2)
    middle1 = n1 // 2
    middle2 = n2 // 2
    index1 = middle1
    index2 = middle2
    restart = False
    second_batch = False

    best_parent_fitness = min(parent1.getFitnessScore(), parent2.
       getFitnessScore())

    while True:
        child = []
        if restart:
            index1 = middle1
            index2 = middle2
            restart = False
            second_batch = True

        element1 = coordinates1[index1]
        element2 = coordinates2[index2]
        while True:
            if not second_batch:
                if (index1 > 0):
                    index1 -= 1
                if (index2 < n2 - 1):
                    index2 += 1
                if index1 == 0 and index2 == n2 - 1:
                    break
            else:
                if (index1 < n1 - 1):
                    index1 += 1
                if (index2 > 0):
                    index2 -= 1
                if index1 == n1 - 1 and index2 == 0:
                    break
            element1 = coordinates1[index1]
            element2 = coordinates2[index2]
            if element1 == element2 or commonNeighbour(problem,
               element1, element2) is not None:
                break

        if element1 == element2:
```

```
            if not second_batch:
                child.append(coordinates1[:index1])
                child.append(coordinates2[index2:])
            else:
                child.append(coordinates1[:index2])
                child.append(coordinates2[index1:])
        else:
            neighbour = commonNeighbour(problem, element1,
               element2)
            if neighbour is not None:
                if not second_batch:
                    child.append(coordinates1[:index1])
                    child.append(neighbour)
                    child.append(coordinates2[index2:])
                else:
                    child.append(coordinates1[:index2])
                    child.append(neighbour)
                    child.append(coordinates2[index1:])
            else:
                if second_batch:
                    restart = True
                else:
                    return []

        if child:
            child = [item for sublist in child for item in
               sublist]
            if len(child) >= 3:
                child = mutation(child)
                child = [item for sublist in child for item in
                   sublist]
                childFinal.setListOfCoordinates(child)
                child_fitness = evaluatePopulation(child, problem
                   .goal)
                childFinal.setFitnessScore(child_fitness)
                if (child_fitness < best_parent_fitness):
                    break

    return childFinal
```

Listing A.27: Genetic Search Algorithm

```
def geneticSearch(problem):
    start_position = problem.getStartState()

    populations = []
    for i in range(1,6):
        populations.append(initializePopulation(problem))

    current_generation = 1
    maximum_number_of_generations = 500
```

```
    while (current_generation <= maximum_number_of_generations):

        for population in populations:
            population.setFitnessScore(evaluatePopulation(
               population.getListOfCoordinates(), problem.goal))

        number_of_populations = len(populations)
        if number_of_populations == 1:
            break
        maximum_number_of_combinations = number_of_populations *
           (number_of_populations - 1) / 2
        new_populations = []
        parent_combinations = []
        while len(new_populations) < number_of_populations:
            random_int1 = 0
            random_int2 = 0
            while True:
                random_int1 = random.randint(0,
                   number_of_populations - 1)
                random_int2 = random.randint(0,
                   number_of_populations - 1)
                if (random_int1 < random_int2 and (random_int1,
                   random_int2) not in parent_combinations) or
                   len(parent_combinations) ==
                   maximum_number_of_combinations:
                    break

            if len(parent_combinations) !=
               maximum_number_of_combinations:
                parent_combinations.append((random_int1,
                   random_int2))
                parent1 = populations[random_int1]
                parent2 = populations[random_int2]
                child = crossover(problem, parent1, parent2)
                if child:
                    new_populations.append(child)
            else:
                break

        if new_populations:
            populations = new_populations
            if len(populations) == 1:
                break
        else:
            break
        current_generation += 1

    list8 = []
    populations = sorted(populations, key=lambda x: x.
       getFitnessScore())
```

```
    population_list = populations[0].getListOfCoordinates()
    n = len(population_list)
    for i in range(0, n - 1):
        if problem.isGoalState(population_list[i]):
            break
        successors = problem.getSuccessors(population_list[i])
        for successor in successors:
            if successor[0] == population_list[i + 1]:
                list8.append(successor[1])

    return list8
```

A.1.2 Corners Problem

Modified Functions For The New Environment

Listing A.28: In the *searchAgent.py* file

```
def getStartState(self):
    return self.startingPosition

def isGoalState(self, state):
    return state[1] == [1, 1, 1, 1]

def getSuccessors(self, state):
    successors = []
    for action in [Directions.NORTH, Directions.SOUTH, Directions
       .EAST, Directions.WEST]:
        x, y = state[0]
        dx, dy = Actions.directionToVector(action)
        nextx, nexty = int(x + dx), int(y + dy)
        hitsWall = self.walls[nextx][nexty]
        if not hitsWall:
            nextState = (nextx, nexty)
            cost = self.costFn(nextState)
            new_state = list(state[1])
            if nextState in self.corners:
                new_state[self.corners.index(nextState)] = 1
            successors.append(((nextState, new_state), action,
               cost))

    self._expanded += 1 # DO NOT CHANGE
    return successors
```

Listing A.29: Corners Heuristic

```
def cornersHeuristic(state, problem):
    result = 0
    x1, y1 = state[0]
    goals = state[1]
    number_of_unvisited_corners = 0
```

```
    for corner in corners:
        if (goals[corners.index(corner)] == 0):
            number_of_unvisited_corners += 1
            dx = abs(x1 - corner[0])
            dy = abs(y1 - corner[1])
            octileDistance = (dx + dy) - ((2 ** 0.5) - 2) * min(
               dx, dy)
            result = max(result, octileDistance)

    return result + number_of_unvisited_corners
```

A.1.3 All Food Search Problem

Listing A.30: All Food Heuristic

```
def foodHeuristic(state, problem):
    result = 0
    x1, y1 = position
    foodList = foodGrid.asList()
    for food in foodList:
        dx = abs(x1 - food[0])
        dy = abs(y1 - food[1])
        octileDistance = (dx + dy) - ((2 ** 0.5) - 2) * min(dx,
           dy)
        manhattanDistance = dx + dy
        euclideanDistance = (dx ** 2 + dy ** 2) ** 0.5
        chebyshevDistance = max(dx, dy)
        result += (4 * octileDistance - (manhattanDistance +
           euclideanDistance + chebyshevDistance))

    return 500 * (result + len(foodList))
```

A.2 Logics

A.2.1 Warm-Up Logic Puzzles

Tricky Heights 1

Listing A.31: *heightPuzzle1.in*

```
set(arithmetic).
assign(domain_size, 5).
assign(max_models, -1).

list(distinct).
        [James, Carly, Kate, Sammy, Natalie].
end_of_list.

formulas(assumptions).

```

```
        shorter(x,y) <-> x < y.
        taller(x,y) <-> x > y.

        shorter(x,y) -> taller(y,x).
        taller(x,y) -> shorter(y,x).

        %James is taller than Kate and Carly
        taller(James, Carly) & taller(James, Kate).

        %Sammy is shorter than Kate
        shorter(Sammy, Kate).

        %Natalie is shorter than Kate and Sammy
        shorter(Natalie, Kate) & shorter(Natalie, Sammy).

        %Sammy is shorter than Carly
        shorter(Sammy, Carly).

end_of_list.
```

Tricky Heights 2

Listing A.32: *heightPuzzle2.in*

```
set(arithmetic).
assign(domain_size, 6).
assign(max_models, -1).

list(distinct).
        [Rachel, Jessica, John, Maria, Lara, Sally].
end_of_list.

formulas(assumptions).

        shorter(x,y) <-> x < y.
        taller(x,y) <-> x > y.

        shorter(x,y) -> taller(y,x).
        taller(x,y) -> shorter(y,x).

        %Rachel and Jessica are taller than John
        taller(Rachel,John) & taller(Jessica,John).

        %Maria is shorter than Lara and Lara is taller than Sally
        shorter(Maria,Lara) & taller(Lara,Sally).

        %John is taller than Lara
        taller(John,Lara).

end_of_list.
```

Tricky Heights 3

Listing A.33: *heightPuzzle3.in*

```
set(arithmetic).
assign(domain_size, 5).
assign(max_models, -1).

list(distinct).
        [Joy, Cassie, Geoff, Val, Brian].
end_of_list.

formulas(assumptions).

        shorter(x,y) <-> x < y.
        taller(x,y) <-> x > y.

        shorter(x,y) -> taller(y,x).
        taller(x,y) -> shorter(y,x).

        %Joy is taller than Cassie
        taller(Joy,Cassie).

        %Joy is taller than Brian who is taller than Cassie and Geoff
        taller(Joy,Brian) & taller(Brian,Cassie) &
           taller(Brian,Geoff).

        %Geoff is shorter than Val who is shorter than Cassie
        shorter(Geoff,Val) & shorter(Val,Cassie).

end_of_list.
```

A.2.2 Einstein Puzzles

Einstein Puzzle 1

Listing A.34: *einsteinPuzzle1.in*

```
set(arithmetic).
assign(domain_size, 8).
assign(max_models, 1).

list(distinct).
        [Daniella, Victoria, Hannah, Jenny, Monica, Irene, Pamela,
           Veronica].
        [Matthew, Owen, Stan, Robert, Alexander, Oto, Paul, Rick].
        [ShopAssistant, Doctor, Agriculturist, WarehouseManager,
           TicketCollector, Accountant, Shopper, Teacher].
        [Trabant, Skoda, Moskvic, Wartburg, Dacia, Fiat, Renault,
           Ziguli].
        [Pink, Brown, White, Yellow, Violet, Red, Green, Blue].
        [MulatkaGabriela, TheModernComedy, DameCommissar, WeWereFive,
           ShedStoat, TheSeadog, GrandfatherJoseph, SlovackoJudge].
end_of_list.

formulas(assumptions).
```

```
15
16          Black=0.
17          Horricks=1.
18          Smith=2.
19          Kuril=3.
20          Cermak=4.
21          Zajac=5.
22          Swain=6.
23          Dvorak=7.
24
25          brought(x,y) & brought(z,y) -> x=z.
26          brought(x,y) & brought(x,z) -> y=z.
27          brought(x,y) -> -borrowed(x,y).
28
29          borrowed(x,y) & borrowed(z,y) -> x=z.
30          borrowed(x,y) & borrowed(x,z) -> y=z.
31          borrowed(x,y) -> -brought(x,y).
32
33          %Clue 1:
34          Daniella = Black & Daniella = ShopAssistant.
35
36          %Clue 2:
37          exists x (brought(x,TheSeadog) & x = Fiat & x = Red).
38
39          %Clue 3:
40          Owen = Victoria & Owen = Brown.
41
42          %Clue 4:
43          Stan = Horricks & Stan = Hannah & Stan = White.
44
45          %Clue 5:
46          Jenny = Smith & Jenny = WarehouseManager & Jenny = Wartburg.
47
48          %Clue 6:
49          Monica = Alexander & borrowed(Monica,GrandfatherJoseph).
50
51          %Clue 7:
52          Matthew = Pink & brought(Matthew,MulatkaGabriela).
53
54          %Clue 8:
55          Irene = Oto & Irene = Accountant.
56
57          %Clue 9:
58          exists x (borrowed(x,WeWereFive) & x=Trabant) &
               brought(Smith,WeWereFive).
59
60          %Clue 10:
61          Cermak = TicketCollector & brought(Cermak,ShedStoat) &
               borrowed(Zajac,ShedStoat).
62
63          %Clue 11:
64          Kuril = Doctor & borrowed(Kuril,SlovackoJudge).
65
```

```
        %Clue 12:
        Paul = Green.

        %Clue 13:
        Veronica = Dvorak & Veronica = Blue.

        %Clue 14:
        brought(Rick,SlovackoJudge) & Rick = Ziguli.

        %Clue 15:
        exists x (brought(x,DameCommissar) &
           borrowed(x,MulatkaGabriela)).

        %Clue 16:
        Dacia = Violet.

        %Clue 17:
        exists x (x=Teacher & borrowed(x,DameCommissar)).

        %Clue 18:
        Agriculturist = Moskvic.

        %Clue 19:
        Pamela = Renault & brought(Pamela,GrandfatherJoseph).

        %Clue 20:
        exists y (brought(Zajac,y) & borrowed(Pamela,y)).

        %Clue 21:
        Robert = Yellow & borrowed(Robert,TheModernComedy).

        %Clue 22:
        Swain = Shopper.

        %Clue 23:
        exists x (brought(x,TheModernComedy) & x=Skoda).

end_of_list.
```

Einstein Puzzle 2

Listing A.35: *einsteinPuzzle2.in*

```
set(arithmetic).
assign(domain_size, 8).
assign(max_models, 1).

list(distinct).
        [Footbal, Cricket, Volleyball, Badminton, LawnTennis,
           Basketball, Hockey, TableTennis].
end_of_list.

formulas(assumptions).
```

```
10
11          Alex=0.
12          Betty=1.
13          Carol=2.
14          Dan=3.
15          Earl=4.
16          Fay=5.
17          George=6.
18          Harry=7.
19
20          Personnel=0.
21          Administration=1.
22          Marketing=2.
23
24          Department(x) <-> x=Personnel | x=Administration |
               x=Marketing.
25          -Department(y) -> WorksIn(x,y)=0.
26
27          %No more than three people can work in any department
28          WorksIn(0,y) + WorksIn(1,y) + WorksIn(2,y) + WorksIn(3,y) +
               WorksIn(4,y) + WorksIn(5,y) + WorksIn(6,y) + WorksIn(7,y)
               <= 3.
29
30          -WorksWith(x,x).
31          WorksWith(x,y) <-> exists z (x!=y & WorksIn(x,z)=1 &
               WorksIn(y,z)=1).
32          WorksWith(x,y) <-> WorksWith(y,x).
33
34          WorksIn(x,y) < 2.
35
36          %A person can only work in one department
37          WorksIn(x,0) + WorksIn(x,1) + WorksIn(x,2) = 1.
38
39          %There need to be 8 people working in all 3 departments
40          WorksIn(0,0) + WorksIn(1,0) + WorksIn(2,0) + WorksIn(3,0) +
               WorksIn(4,0) + WorksIn(5,0) + WorksIn(6,0) + WorksIn(7,0) +
41          WorksIn(0,1) + WorksIn(1,1) + WorksIn(2,1) + WorksIn(3,1) +
               WorksIn(4,1) + WorksIn(5,1) + WorksIn(6,1) + WorksIn(7,1) +
42          WorksIn(0,2) + WorksIn(1,2) + WorksIn(2,2) + WorksIn(3,2) +
               WorksIn(4,2) + WorksIn(5,2) + WorksIn(6,2) + WorksIn(7,2)
               = 8.
43
44          %Clue 1:
45          WorksIn(Dan,Administration)=1 & Dan!=Footbal & Dan!=Cricket.
46
47          %Clue 2:
48          WorksIn(Fay,Personnel)=1 & WorksWith(Fay,Alex) &
               Alex=TableTennis.
49          x!=Fay & x!=Alex -> WorksIn(x,Personnel)=0.
50
51          %Clue 3:
52          -WorksWith(Earl,Dan) & -WorksWith(Harry,Dan).
53
```

```
54          %Clue 4:
55          Carol=Hockey & WorksIn(Carol,Marketing)=0.
56
57          %Clue 5:
58          WorksIn(George,Administration)=0 & George!=Cricket &
               George!=Badminton.
59
60          %Clue 6:
61          exists x (WorksIn(x,Administration)=1 & x=Footbal).
62
63          %Clue 7:
64          exists x (x=Volleyball & WorksIn(x,Personnel)=1).
65
66          %Clue 8:
67          WorksIn(x,Administration)=1 -> (x!=Badminton & x!=LawnTennis).
68
69          %Clue 9:
70          Harry!=Cricket.
71
72  end_of_list.
```

Einstein Puzzle 3

Listing A.36: *einsteinPuzzle3.in*

```
1   set(arithmetic).
2   assign(domain_size, 12).
3   assign(max_models, 1).
4
5   list(distinct).
6           [Paul, Hank, Luke, Sam, Zick].
7   end_of_list.
8
9   formulas(assumptions).
10
11          %Clue 1:
12          Apple = 0.
13          Pear = 1.
14          Nut = 2.
15          Cherry = 3.
16          Carrot = 4.
17          Parsley = 5.
18          Gourd = 6.
19          Onion = 7.
20          Aster = 8.
21          Rose = 9.
22          Tulip = 10.
23          Lily = 11.
24
25          inGarden(y,x) < 2.
26
27          (y=0 | y>5) -> inGarden(y,x)=0.
28
```

```
29          y>=1 & y<=5 -> inGarden(y,0) + inGarden(y,1) + inGarden(y,2)
               + inGarden(y,3) + inGarden(y,4) + inGarden(y,5) +
               inGarden(y,6) + inGarden(y,7) + inGarden(y,8) +
               inGarden(y,9) + inGarden(y,10) + inGarden(y,11) = 4.
30
31          fruit(x) <-> (x=Apple | x=Pear | x=Nut | x=Cherry).
32          vegetable(x) <-> (x=Carrot | x=Parsley | x=Gourd | x=Onion).
33          flower(x) <-> (x=Aster | x=Rose | x=Tulip | x=Lily).
34
35          validGarden(y) <-> (y>=1 & y<=5).
36
37          validGarden(Paul) & validGarden(Hank) & validGarden(Luke) &
               validGarden(Sam) & validGarden(Zick).
38
39          %Clue 2:
40          validGarden(y) -> inGarden(y,0) + inGarden(y,1) +
               inGarden(y,2) + inGarden(y,3) + inGarden(y,4) +
               inGarden(y,5) + inGarden(y,6) + inGarden(y,7) +
               inGarden(y,8) + inGarden(y,9) + inGarden(y,10) +
               inGarden(y,11) = 4.
41
42          %Clue 3:
43          inGarden(1,x) + inGarden(2,x) + inGarden(3,x) + inGarden(4,x)
               + inGarden(5,x) >= 1.
44          inGarden(1,x) + inGarden(2,x) + inGarden(3,x) + inGarden(4,x)
               + inGarden(5,x) < 5.
45
46          %Clue 4:
47          exists x (inGarden(1,x) + inGarden(2,x) + inGarden(3,x) +
               inGarden(4,x) + inGarden(5,x) = 4).
48          inGarden(1,x) + inGarden(2,x) + inGarden(3,x) + inGarden(4,x)
               + inGarden(5,x) = 4 & inGarden(1,y) + inGarden(2,y) +
               inGarden(3,y) + inGarden(4,y) + inGarden(5,y) = 4 -> y=x.
49
50          %Clue 5:
51          %exists x exists y exists z (fruit(x) & vegetable(y) &
               flower(z) &
52      %((inGarden(1,x) + inGarden(1,y) + inGarden(1,z) = 3) |
53          %(inGarden(2,x) + inGarden(2,y) + inGarden(2,z) = 3) |
54          %(inGarden(3,x) + inGarden(3,y) + inGarden(3,z) = 3) |
55          %(inGarden(4,x) + inGarden(4,y) + inGarden(4,z) = 3) |
56          %(inGarden(5,x) + inGarden(5,y) + inGarden(5,z) = 3))).
57
58          exists x (fruit(x) & inGarden(Sam,x)=1).
59          exists y (vegetable(y) & inGarden(Sam,y)=1).
60          exists z1 exists z2 (z1<z2 & flower(z1) & flower(z2) &
               inGarden(Sam,z1)+inGarden(Sam,z2)=2).
61
62          %Clue 6:
63          ((fruit(x) -> inGarden(1,x)=1) | (vegetable(y) ->
               inGarden(1,y)=1) | (flower(z) -> inGarden(1,z)=1)) |
64          ((fruit(x) -> inGarden(2,x)=1) | (vegetable(y) ->
               inGarden(2,y)=1) | (flower(z) -> inGarden(2,z)=1)) |
```

```
65          ((fruit(x) -> inGarden(3,x)=1) | (vegetable(y) ->
               inGarden(3,y)=1) | (flower(z) -> inGarden(3,z)=1)) |
66          ((fruit(x) -> inGarden(4,x)=1) | (vegetable(y) ->
               inGarden(4,y)=1) | (flower(z) -> inGarden(4,z)=1)) |
67          ((fruit(x) -> inGarden(5,x)=1) | (vegetable(y) ->
               inGarden(5,y)=1) | (flower(z) -> inGarden(5,z)=1)).
68
69          %Clue 7:
70          inGarden(1,Pear)=1.
71          inGarden(2,Pear)=0.
72          inGarden(3,Pear)=0.
73          inGarden(4,Pear)=0.
74          inGarden(5,Pear)=1.
75
76          %Clue 8:
77          Paul=3.
78          inGarden(3,Lily)=0.
79
80          %Clue 9:
81          inGarden(1,Aster) + inGarden(2,Aster) + inGarden(3,Aster) +
               inGarden(4,Aster) + inGarden(5,Aster) = 1.
82          (validGarden(y) & vegetable(x) & inGarden(y,x)=1) ->
               inGarden(y,Aster)=0.
83
84          %Clue 10:
85          (validGarden(y) & inGarden(y,Rose)=1) ->
               inGarden(y,Parsley)=0.
86
87          %Clue 11:
88          inGarden(1,Nut) + inGarden(2,Nut) + inGarden(3,Nut) +
               inGarden(4,Nut) + inGarden(5,Nut) = 1.
89          (validGarden(y) & inGarden(y,Nut)=1) -> (inGarden(y,Gourd)=1
               & inGarden(y,Parsley)=1).
90
91          %Clue 12:
92          inGarden(1,Apple)=1 & inGarden(1,Cherry)=1.
93
94          %Clue 13:
95          inGarden(1,Cherry) + inGarden(2,Cherry) + inGarden(3,Cherry)
               + inGarden(4,Cherry) + inGarden(5,Cherry) = 2.
96
97          %Clue 14:
98          inGarden(Sam,Cherry)=1 & inGarden(Sam,Onion)=1.
99
100         %Clue 15:
101         inGarden(Luke,0) + inGarden(Luke,1) + inGarden(Luke,2) +
               inGarden(Luke,3) = 2.
102
103         %Clue 16:
104         inGarden(1,Tulip) + inGarden(2,Tulip) + inGarden(3,Tulip) +
               inGarden(4,Tulip) + inGarden(5,Tulip) = 2.
105
106         %Clue 17:
```

```
107         inGarden(1,Apple) + inGarden(2,Apple) + inGarden(3,Apple) +
               inGarden(4,Apple) + inGarden(5,Apple) = 1.
108
109         %Clue 18:
110         inGarden(1,Parsley) + inGarden(2,Parsley) +
               inGarden(3,Parsley) + inGarden(4,Parsley) +
               inGarden(5,Parsley) = 1.
111         inGarden(Zick,Parsley)=0.
112         Zick=1 -> inGarden(2,Parsley)=1.
113         Zick=2 -> (inGarden(1,Parsley)=1 | inGarden(3,Parsley)=1).
114         Zick=4 -> (inGarden(3,Parsley)=1 | inGarden(5,Parsley)=1).
115         Zick=5 -> inGarden(4,Parsley)=1.
116
117         %Clue 19:
118         Sam!=1 & Sam!=5.
119
120         %Clue 20:
121         vegetable(x) -> inGarden(Hank,x)=0.
122         inGarden(Hank,Aster)=0.
123
124         %Clue 21:
125         exists x1 exists x2 exists x3 exists x4 (x1<x2 & x2<x3 &
               x4!=x1 & x4!=x2 & x4!=x3 & vegetable(x1) & vegetable(x2) &
               vegetable(x3) & -vegetable(x4) & inGarden(Paul,x1)=1 &
               inGarden(Paul,x2)=1 & inGarden(Paul,x3)=1 &
               inGarden(Paul,x4)=1).
126
127 end_of_list.
```

A.2.3 Sam Loyd Puzzles

Sam Loyd Puzzle 1

Listing A.37: *samLoydPuzzle1.in*

```
1  set(arithmetic).
2  assign(domain_size, 901).
3  assign(max_models, -1).
4  assign(max_seconds, 15).
5
6  formulas(assumptions).
7
8          Wagon0>0 & People0>0 & Wagon1>0 & People1>0 & Wagon2>0 &
              People2>0.
9
10         Wagon1 = Wagon0 + -10.
11         People1 = People0 + 1.
12
13         Wagon2 = Wagon1 + -15.
14         People2 = People0 + 3.
15
16         Wagon0*People0=Wagon1*People1 & Wagon0*People0=Wagon2*People2.
17
```

```
end_of_list.
```

Sam Loyd Puzzle 2

Listing A.38: *samLoydPuzzle2.in*

```
set(arithmetic).
assign(domain_size, 30).
assign(max_models, 1).

formulas(assumptions).

        childrenWithoutSammy/3 + (3*childrenWithoutSammy)/4 =
           childrenWithoutSammy+1.
        childrenWithSammy = childrenWithoutSammy + 1.

end_of_list.
```

Sam Loyd Puzzle 3

Listing A.39: *samLoydPuzzle2.in*

```
set(arithmetic).
assign(domain_size, 73).
assign(max_models, 1).

formulas(assumptions).

        herd = 17 | herd = 35 | herd = 53 | herd = 71.

        (herd+1)/2 + (herd+1)/3 + (herd+1)/9 = herd.

        son1 * 2 = herd.
        son2 * 3 = herd.
        son3 * 9 = herd.

end_of_list.
```

A.2.4 Math Puzzles

Math Puzzle 1

Listing A.40: *mathPuzzle1_a.in*

```
set(arithmetic).
assign(domain_size, 76).
assign(max_models, -1).

list(distinct).
        [0,B1,B2,B3,B4,B5,B6,B7,B8,B9,B10].
end_of_list.

```

```
formulas(assumptions).

        B1<=10 & B2<=10 & B3<=10 & B4<=10 & B5<=10 & B6<=10 & B7<=10
           & B8<=10 & B9<=10 & B10<=10.

        B1 < B2.
        B3 < B4 & B4 < B5.
        B6 < B7 & B7 < B8 & B8 < B9 & B9 < B10.

        b(1)=15.
        b(2)=13.
        b(3)=11.
        b(4)=10.
        b(5)=9.
        b(6)=8.
        b(7)=4.
        b(8)=2.
        b(9)=2.
        b(10)=1.
        (x>=11 | x=0) -> b(x)=0.

        c1=25.
        c2=25.
        c3=25.

        c1=b(B1)+b(B2).
        c2=b(B3)+b(B4)+b(B5).
        c3=b(B6)+b(B7)+b(B8)+b(B9)+b(B10).

        c1+c2+c3=75.

end_of_list.
```

Listing A.41: *mathPuzzle1_b.in*

```
set(arithmetic).
assign(domain_size, 76).
assign(max_models, -1).

list(distinct).
        [0,B1,B2,B3,B4,B5,B6,B7,B8,B9,B10].
end_of_list.

formulas(assumptions).

        B1<=10 & B2<=10 & B3<=10 & B4<=10 & B5<=10 & B6<=10 & B7<=10
           & B8<=10 & B9<=10 & B10<=10.

        B1 < B2.
        B3 < B4 & B4 < B5 & B5 < B6.
        B7 < B8 & B8 < B9 & B9 < B10.

        B1 < B3 & B3 < B7. % avoid isomorphisms
        B10 != 8. %avoid isomorphisms
```

```
19
20          b(1)=15.
21          b(2)=13.
22          b(3)=11.
23          b(4)=10.
24          b(5)=9.
25          b(6)=8.
26          b(7)=4.
27          b(8)=2.
28          b(9)=2.
29          b(10)=1.
30          (x>=11 | x=0) -> b(x)=0.
31
32          c1=25.
33          c2=25.
34          c3=25.
35
36          c1=b(B1)+b(B2).
37          c2=b(B3)+b(B4)+b(B5)+b(B6).
38          c3=b(B7)+b(B8)+b(B9)+b(B10).
39
40          c1+c2+c3=75.
41
42  end_of_list.
```

Listing A.42: *mathPuzzle1_c.in*

```
1   set(arithmetic).
2   assign(domain_size, 76).
3   assign(max_models, -1).
4
5   list(distinct).
6           [0,B1,B2,B3,B4,B5,B6,B7,B8,B9,B10].
7   end_of_list.
8
9   formulas(assumptions).
10
11          B1<=10 & B2<=10 & B3<=10 & B4<=10 & B5<=10 & B6<=10 & B7<=10
               & B8<=10 & B9<=10 & B10<=10.
12
13          B1 < B2 & B2 < B3.
14          B4 < B5 & B5 < B6.
15          B7 < B8 & B8 < B9 & B9 < B10.
16
17          B1 < B4. %avoid isomorphisms
18          B3 = 8 -> B3 < B6. %avoid isomorphisms
19          B3 = 9 -> B3 < B10 & B3 < B9. %avoid isomorphisms
20
21          b(1)=15.
22          b(2)=13.
23          b(3)=11.
24          b(4)=10.
25          b(5)=9.
26          b(6)=8.
```

```
        b(7)=4.
        b(8)=2.
        b(9)=2.
        b(10)=1.
        (x>=11 | x=0) -> b(x)=0.

        c1=25.
        c2=25.
        c3=25.

        c1=b(B1)+b(B2)+b(B3).
        c2=b(B4)+b(B5)+b(B6).
        c3=b(B7)+b(B8)+b(B9)+b(B10).

        c1+c2+c3=75.

end_of_list.
```

Math Puzzle 2

Listing A.43: *mathPuzzle2.in*

```
set(arithmetic).
assign(domain_size, 121).
assign(max_models, 1).

formulas(assumptions).

        loaves=120.
        numberOfWorkers=5.

        Worker1 = Worker + -(2*d).
        Worker2 = Worker + -(d).
        Worker3 = Worker.
        Worker4 = Worker + d.
        Worker5 = Worker + 2*d.

        Worker=loaves/numberOfWorkers.

        7*(Worker1+Worker2)=Worker3+Worker4+Worker5.

end_of_list.
```

Math Puzzle 3

Listing A.44: *mathPuzzle3.in*

```
set(arithmetic).
assign(domain_size, 3200).
assign(max_models, -1).

list(distinct).
        [0,a1,a2,a3,a4,a5,a6,a7].
```

```
end_of_list.

formulas(assumptions).

        a7<a6 & a6<a5 & a5<a4 & a4<a3 & a3<a2 & a2<a1.

        a1 = 5 * a2 + 1.
        4 * a2 = 5 * a3 + 1.
        4 * a3 = 5 * a4 + 1.
        4 * a4 = 5 * a5 + 1.
        4 * a5 = 5 * a6 + 1.
        4 * a6 = 5 * a7.

end_of_list.
```

Math Puzzle 4

Listing A.45: *mathPuzzle4.in*

```
set(arithmetic).
assign(domain_size, 51).
assign(max_models, -1).

list(distinct).
        [0, Tammy, Tracey, Tommy, Timmy, Tony, Mother].
end_of_list.

formulas(assumptions).

        Tony<Tammy & Tony<Tracey & Tony<Tommy & Tony<Timmy &
           Tony<Mother.
        Mother>Tammy & Mother>Tracey & Mother>Tommy & Mother>Timmy.
        Tommy>Tammy & Tommy>Tracey & Tommy>Timmy.

        Tammy=3*(Timmy + -Tony).

        Mother + -Tony = 30.

        % The factors of 30, excluding 1 and 30, are: 2, 3, 5, 6, 10,
           15
        (Tammy + -Tony = 2) | (Tammy + -Tony = 3) | (Tammy + -Tony =
           5) | (Tammy + -Tony = 6) | (Tammy + -Tony = 10) | (Tammy +
           -Tony = 15).
        (Tracey + -Tony = 2) | (Tracey + -Tony = 3) | (Tracey + -Tony
           = 5) | (Tracey + -Tony = 6) | (Tracey + -Tony = 10) |
           (Tracey + -Tony = 15).
        (Tommy + -Tony = 2) | (Tommy + -Tony = 3) | (Tommy + -Tony =
           5) | (Tommy + -Tony = 6) | (Tommy + -Tony = 10) | (Tommy +
           -Tony = 15).
        (Timmy + -Tony = 2) | (Timmy + -Tony = 3) | (Timmy + -Tony =
           5) | (Timmy + -Tony = 6) | (Timmy + -Tony = 10) | (Timmy +
           -Tony = 15).

```

```
25          2*n1 = 2*Tracey + -Tammy.
26          % It will result that n1=7; therefore Mother-7=36
27          % The factors of 36, excluding 1 and 36 are: 2, 3, 4, 6, 9,
               12, 18
28
29          (Tracey + -n1 = 2) | (Tracey + -n1 = 3) | (Tracey + -n1 = 4)
               | (Tracey + -n1 = 6) | (Tracey + -n1 = 9) | (Tracey + -n1
               = 12) | (Tracey + -n1 = 18).
30          (Timmy + -n1 = 2) | (Timmy + -n1 = 3) | (Timmy + -n1 = 4) |
               (Timmy + -n1 = 6) | (Timmy + -n1 = 9) | (Timmy + -n1 = 12)
               | (Timmy + -n1 = 18).
31          (Tony + -n1 = 2) | (Tony + -n1 = 3) | (Tony + -n1 = 4) |
               (Tony + -n1 = 6) | (Tony + -n1 = 9) | (Tony + -n1 = 12) |
               (Tony + -n1 = 18).
32
33          n2 = 50 + -Mother.
34          Tony + n2 = Mother + -Tommy.
35
36          n1=n2.
37
38  end_of_list.
```

A.2.5 Cypher Puzzles

Cypher Puzzle 1

Listing A.46: *cypherPuzzle1.in*

```
1   set(arithmetic).
2   assign(domain_size, 1001).
3   assign(max_models, 1).
4
5   list(distinct).
6           [A,B,C,D,E].
7           [0,A].
8   end_of_list.
9
10  formulas(assumptions).
11
12          A<=9 & B<=9 & C<=9 & D<=9 & E<=9.
13
14          (A*1000 + B*100 + C*10 + D) * E = D*1000 + C*100 + B*10 + A.
15
16  end_of_list.
```

Cypher Puzzle 2

Listing A.47: *cypherPuzzle2.in*

```
1   set(arithmetic).
2   assign(domain_size, 1400).
3   assign(max_models, -1).
4
```

```
list(distinct).
        [D,E].
        [0,D].
end_of_list.

formulas(assumptions).

        D<=9 & E<=9.

        n1 = D*10 + D.
        n2 = D*1000 + E*100 + E*10 + D.

        E=0 -> 1=n2.
        E=1 -> n1=n2.
        E=2 -> n1*n1=n2.
        E=3 -> n1*n1*n1=n2.
        E=4 -> n1*n1*n1*n1=n2.
        E=5 -> n1*n1*n1*n1*n1=n2.
        E=6 -> n1*n1*n1*n1*n1*n1=n2.
        E=7 -> n1*n1*n1*n1*n1*n1*n1=n2.
        E=8 -> n1*n1*n1*n1*n1*n1*n1*n1=n2.
        E=9 -> n1*n1*n1*n1*n1*n1*n1*n1*n1=n2.

end_of_list.
```

Cypher Puzzle 3

Listing A.48: *cypherPuzzle3.in*

```
set(arithmetic).
assign(domain_size, 30).
assign(max_models, -1).

list(distinct).
        [D1,D2,D3,D4,D5,D6].
end_of_list.

formulas(assumptions).

        isDigit(x) <-> x<=9.

        isDigit(D1) & isDigit(D2) & isDigit(D3) & isDigit(D4) &
           isDigit(D5) & isDigit(D6).

        even(x) <-> x=0 | x=2 | x=4 | x=6 | x=8.
        odd(x) <-> x=1 | x=3 | x=5 | x=7 | x=9.

        (even(D1) & odd(D2) & even(D3) & odd(D4) & even(D5) &
           odd(D6)) |
        (odd(D1) & even(D2) & odd(D3) & even(D4) & odd(D5) &
           even(D6)).

        neighbour(x,y) <-> y + -x > 1 | x + -y > 1.
```

```
        neighbour(x,y) -> neighbour(y,x).

        neighbour(D1,D2) & neighbour(D2,D3) & neighbour(D3,D4) &
           neighbour(D4,D5) & neighbour(D5,D6).

        exists x exists y ((D5*10+D6)*x=(D3*10+D4) &
           (D5*10+D6)*y=(D1*10+D2)).

end_of_list.
```

Cypher Puzzle 4

Listing A.49: *cypherPuzzle4.in*

```
set(arithmetic).
assign(domain_size, 30).
assign(max_models, -1).

list(distinct).
        [A,B,C,D,E,F,G,H,I,J].
end_of_list.

formulas(assumptions).

        square(x) <-> x=0 | x=1 | x=4 | x=9.
        triangle(x) <-> x=1 | x=3 | x=6.
        even(x) <-> x=0 | x=2 | x=4 | x=6 | x=8.
        odd(x) <-> x=1 | x=3 | x=5 | x=7 | x=9.
        cube(x) <-> x=0 | x=1 | x=8.
        prime(x) <-> x=2 | x=3 | x=5 | x=7.

        A<=9 & B<=9 & C<=9 & D<=9 & E<=9 & F<=9 & G<=9 & H<=9 & I<=9
           & J<=9.

        A>0.

        %Clue 1:
        square(A) | triangle(A).
        -(square(A) & triangle(A)).

        %Clue 2:
        even(B) | cube(B).
        -(even(B) & cube(B)).

        %Clue 3:
        cube(C) | triangle(C).
        -(cube(C) & triangle(C)).

        %Clue 4:
        odd(D) | square(D).
        -(odd(D) & square(D)).

        %Clue 5:
```

```
        odd(E) | cube(E).
        -(odd(E) & cube(E)).

        %Clue 6:
        odd(F) | triangle(F).
        -(odd(F) & triangle(F)).

        %Clue 7:
        odd(G) | prime(G).
        -(odd(G) & prime(G)).

        %Clue 8:
        even(H) | square(H).
        -(even(H) & square(H)).

        %Clue 9:
        square(I) | cube(I).
        -(square(I) & cube(I)).

        %Clue 10:
        prime(J) | triangle(J).
        -(prime(J) & triangle(J)).

        %Clue 11:
        A<B & C<D & E<F & G<H & I<J.

        %Clue 12:
        A+B+C+D+E < F+G+H+I+J.

end_of_list.
```

Cypher Puzzle 5

Listing A.50: *cypherPuzzle5.in*

```
set(arithmetic).
assign(domain_size, 37).
assign(max_models, -1).

list(distinct).
        [A,B,C,D,E,F,G,H,I,J].
end_of_list.

formulas(assumptions).

        A<=9 & B<=9 & C<=9 & D<=9 & E<=9 & F<=9 & G<=9 & H<=9 & I<=9
           & J<=9.

        A>0.

        %Clue 1:
        B=A*3 | A=G+3.

```

```
        %Clue 2:
        B+4=I | B=E+4.

        %Clue 3:
        C=J+2 | C=F*3.

        %Clue 4:
        D=G*4 | E=D*3.

        %Clue 5:
        J=E+1 | D=E*4.

        %Clue 6:
        F=B*2 | A=F+4.

        %Clue 7:
        G=F+1 | I=G+3.

        %Clue 8:
        A=H*2 | H=C*3.

        %Clue 9:
        I=H+3 | D=I*2.

        %Clue 10:
        H=J+2 | J=C*2.

end_of_list.
```

Cypher Puzzle 6

Listing A.51: *cypherPuzzle6.in*

```
set(arithmetic).
assign(domain_size, 10).
assign(max_models, -1).

formulas(assumptions).

        less(x,y) < 2 & greater(x,y) < 2 & equal(x,y) < 2.
        less(x,y) = 1 -> (x <= y).
        less(x,y) = 0 -> (x > y).

        greater(x,y) = 1 -> (x >= y).
        greater(x,y) = 0 -> (x < y).

        equal(x,y) = 1 -> (x = 1 & y = 1).
        equal(x,y) = 0 -> (x != 1 | y != 1).

        f(x,y) = equal(less(c(x),y),greater(c(x),y)).
        h(x) = f(0,x) + f(1,x) + f(2,x) + f(3,x) + f(4,x) + f(5,x) +
           f(6,x) + f(7,x) + f(8,x) + f(9,x).

```

```
20          %First digit of a number cannot be 0
21          c(0) > 0.
22
23          c(x) = h(x).
24
25  end_of_list.
```

Cypher Puzzle 7

Listing A.52: *cypherPuzzle6.in*

```
1   set(arithmetic).
2   assign(domain_size, 10).
3   assign(max_models, -1).
4
5   formulas(assumptions).
6
7           orAB(x) <-> A=x | B=x.
8           orBC(x) <-> B=x | C=x.
9           orAC(x) <-> A=x | C=x.
10
11          notTwoDigitsAB(x,y) <-> A!=x & A!=y & B!=x & B!=y.
12          notTwoDigitsBC(x,y) <-> B!=x & B!=y & C!=x & C!=y.
13          notTwoDigitsAC(x,y) <-> A!=x & A!=y & C!=x & C!=y.
14
15          oneDigitIsCorrectAndWronglyPlaced(x,y,z) <-> (B=x &
               notTwoDigitsAC(y,z)) | (C=x & notTwoDigitsAB(y,z)) | (A=y
               & notTwoDigitsBC(x,z)) | (C=y & notTwoDigitsAB(x,z)) |
               (A=z & notTwoDigitsBC(x,y)) | (B=z & notTwoDigitsAC(x,y)).
16
17          twoDigitsAreCorrectAndWronglyPlaced(x,y,z) <-> (A!=x &
               orBC(x) & B!=y & orAC(y)) | (A!=x & orBC(x) & C!=z &
               orAB(z)) | (B!=y & orAC(y) & C!=z & orAB(z)).
18
19          nothingIsCorrect(x,y,z) <-> A!=x & A!=y & A!=z & B!=x & B!=y
               & B!=z & C!=x & C!=y & C!=z.
20
21          oneDigitIsCorrectAndWellPlaced(x,y,z) <-> (A=x &
               notTwoDigitsBC(y,z)) | (B=y & notTwoDigitsAC(x,z)) | (C=z
               & notTwoDigitsAB(x,y)).
22
23          twoDigitsAreCorrectAndWellPlaced(x,y,z) <-> (A=x & B=y &
               C!=z) | (A=x & B!=y & C=z) | (A!=x & B=y & C=z).
24
25
26          oneDigitIsCorrectAndWellPlaced(6,8,2).
27          oneDigitIsCorrectAndWronglyPlaced(6,1,4).
28          twoDigitsAreCorrectAndWronglyPlaced(2,0,6).
29          nothingIsCorrect(7,3,8).
30          oneDigitIsCorrectAndWronglyPlaced(8,3,0).
31
32  end_of_list.
```

A.2.6 Logic Puzzles

Logic Puzzle 1

Listing A.53: *logicPuzzle1.in*

```
set(arithmetic).
assign(domain_size, 56).
assign(max_models, -1).

list(distinct).
        [Alex,Brook,Cody,Dusty,Erin].
end_of_list.

formulas(assumptions).

        Dusty<Brook <-> Dusty=Erin+9.
        Erin<Brook <-> Erin=Alex+7.
        Alex<Brook <-> Alex*17=Brook*10.
        Brook<Cody <-> Erin<Cody.
        Brook<Cody <-> abs(Cody + -Dusty) = abs(Dusty + -Erin).
        Cody<Dusty <-> Cody=Dusty+6 | Dusty=Cody+6.
        Cody<Alex <-> Cody=Alex+10.
        Cody<Alex <-> Brook<Dusty.

end_of_list.
```

Logic Puzzle 2

Listing A.54: *logicPuzzle2.in*

```
set(arithmetic).
assign(domain_size, 15).
assign(max_models, -1).
assign(max_seconds, 10).

formulas(assumptions).

        A=0.
        B=1.
        C=2.
        D=3.
        E=4.
        F=5.
        G=6.
        H=7.
        I=8.
        J=9.
        K=10.
        L=11.
        M=12.
        N=13.
        O=14.

```

```
24          walkTogetherMon(x,y) -> walkTogetherMon(y,x).
25          walkTogetherTue(x,y) -> walkTogetherTue(y,x).
26          walkTogetherWed(x,y) -> walkTogetherWed(y,x).
27          walkTogetherThu(x,y) -> walkTogetherThu(y,x).
28          walkTogetherFri(x,y) -> walkTogetherFri(y,x).
29          walkTogetherSat(x,y) -> walkTogetherSat(y,x).
30          walkTogetherSun(x,y) -> walkTogetherSun(y,x).
31
32          % A schoolgirl cannot walk to school with herself
33          -walkTogetherMon(x,x) & -walkTogetherTue(x,x) &
               -walkTogetherWed(x,x) & -walkTogetherThu(x,x) &
               -walkTogetherFri(x,x) & -walkTogetherSat(x,x) &
               -walkTogetherSun(x,x).
34
35          exists y1 exists y2 (y1<y2 & walkTogetherMon(x,y1) &
               walkTogetherMon(x,y2)).
36          y1<y2 & walkTogetherMon(x,y1) & walkTogetherMon(x,y2) &
               walkTogetherMon(x,y3) -> y3=y1 | y3=y2.
37
38          exists y1 exists y2 (y1<y2 & walkTogetherTue(x,y1) &
               walkTogetherTue(x,y2)).
39          y1<y2 & walkTogetherTue(x,y1) & walkTogetherTue(x,y2) &
               walkTogetherTue(x,y3) -> y3=y1 | y3=y2.
40
41          exists y1 exists y2 (y1<y2 & walkTogetherWed(x,y1) &
               walkTogetherWed(x,y2)).
42          y1<y2 & walkTogetherWed(x,y1) & walkTogetherWed(x,y2) &
               walkTogetherWed(x,y3) -> y3=y1 | y3=y2.
43
44          exists y1 exists y2 (y1<y2 & walkTogetherThu(x,y1) &
               walkTogetherThu(x,y2)).
45          y1<y2 & walkTogetherThu(x,y1) & walkTogetherThu(x,y2) &
               walkTogetherThu(x,y3) -> y3=y1 | y3=y2.
46
47          exists y1 exists y2 (y1<y2 & walkTogetherFri(x,y1) &
               walkTogetherFri(x,y2)).
48          y1<y2 & walkTogetherFri(x,y1) & walkTogetherFri(x,y2) &
               walkTogetherFri(x,y3) -> y3=y1 | y3=y2.
49
50          exists y1 exists y2 (y1<y2 & walkTogetherSat(x,y1) &
               walkTogetherSat(x,y2)).
51          y1<y2 & walkTogetherSat(x,y1) & walkTogetherSat(x,y2) &
               walkTogetherSat(x,y3) -> y3=y1 | y3=y2.
52
53          exists y1 exists y2 (y1<y2 & walkTogetherSun(x,y1) &
               walkTogetherSun(x,y2)).
54          y1<y2 & walkTogetherSun(x,y1) & walkTogetherSun(x,y2) &
               walkTogetherSun(x,y3) -> y3=y1 | y3=y2.
55
56
57          walkTogetherMon(x,y) -> -walkTogetherTue(x,y) &
               -walkTogetherWed(x,y) & -walkTogetherThu(x,y) &
               -walkTogetherFri(x,y) & -walkTogetherSat(x,y) &
```

```
            -walkTogetherSun(x,y).
58          walkTogetherTue(x,y) -> -walkTogetherMon(x,y) &
            -walkTogetherWed(x,y) & -walkTogetherThu(x,y) &
            -walkTogetherFri(x,y) & -walkTogetherSat(x,y) &
            -walkTogetherSun(x,y).
59          walkTogetherWed(x,y) -> -walkTogetherTue(x,y) &
            -walkTogetherMon(x,y) & -walkTogetherThu(x,y) &
            -walkTogetherFri(x,y) & -walkTogetherSat(x,y) &
            -walkTogetherSun(x,y).
60          walkTogetherThu(x,y) -> -walkTogetherTue(x,y) &
            -walkTogetherWed(x,y) & -walkTogetherMon(x,y) &
            -walkTogetherFri(x,y) & -walkTogetherSat(x,y) &
            -walkTogetherSun(x,y).
61          walkTogetherFri(x,y) -> -walkTogetherTue(x,y) &
            -walkTogetherWed(x,y) & -walkTogetherThu(x,y) &
            -walkTogetherMon(x,y) & -walkTogetherSat(x,y) &
            -walkTogetherSun(x,y).
62          walkTogetherSat(x,y) -> -walkTogetherTue(x,y) &
            -walkTogetherWed(x,y) & -walkTogetherThu(x,y) &
            -walkTogetherFri(x,y) & -walkTogetherMon(x,y) &
            -walkTogetherSun(x,y).
63          walkTogetherSun(x,y) -> -walkTogetherTue(x,y) &
            -walkTogetherWed(x,y) & -walkTogetherThu(x,y) &
            -walkTogetherFri(x,y) & -walkTogetherSat(x,y) &
            -walkTogetherMon(x,y).
64
65  end_of_list.
```