University of Hertfordshire

MEng Computer Science Final Year Project

Information Based Model Exploration

Daniel Barry

danbarry16@googlemail.com

supervised by
Prof. Daniel Polani

September, 2016

Abstract

In this project we look at information gain as an effective approach to model exploration. We tackle two main problems, the fake coin problem and a camera angle limit search problem. Previous work towards self-modelling agents has failed to address the use of an information-theoretic driven search. In order to achieve this, recursively averaging information gain was used to update an internal Bayesian model to the agent. To test the effectiveness of this idea, two pieces of software were written for each problem with the information gain analysed at different depths and compared with a random action search agent. From the experiments explained in this project, we see that an information-theoretic approach is effective, with the recursive benefit depending on whether there are local maximums in the search.

Contents

1 Introduction

The aim of this project is to explore how model exploration may be achieved through a series of simpler problems, which should demonstrate how spatial exploration is feasible given a small amount of information.

1.1 Motivation

The general research direction is towards the particular issue posed by modelling humanoid robotics in RoboCup - as the problem is considered it to be sufficiently difficult in order to further the rate of progress in robotics and artificial intelligence [1] [2].

Specifically, the interest in RoboCup comes from the University team, Bold Hearts [17]. With the newest challenge of 3D printing larger robots to help overcome the issues associated with the addition of artificial grass to the problem, the team now looks towards an effective way to control such a large structure.

Figure 1, generated from table 1, shows there has been a general push towards larger robots in the competition with the introduction of artificial grass [7], posing a problem for simple walk gaits, which typically relied on predictable surfaces and reduced motor noise. More dynamic walking algorithms such as Zero Moment Point (ZMP) rely on accurate model descriptions in order to infer data such as centre of mass [8].

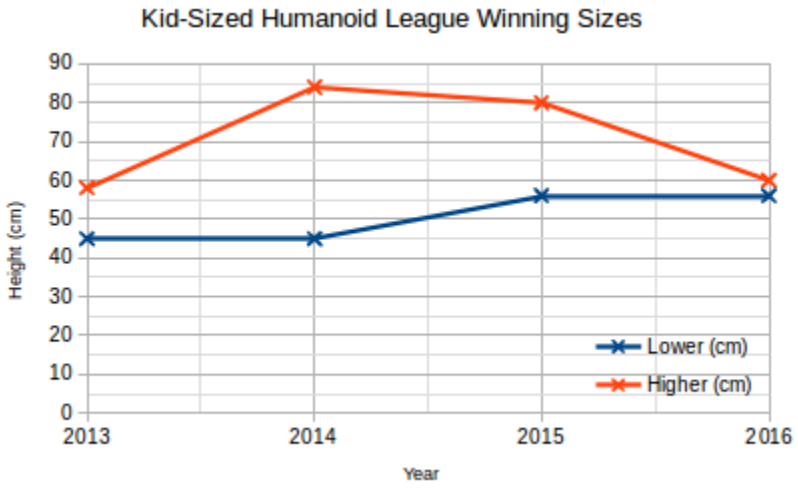

Figure 1: For the 2015 RoboCup World Cup the astro-turf was introduced to the game as part of many incremental changes towards the 2050 goal.

The general areas for research in the generalised topic of self-modelling agents are considered as the following:

- Efficient agent model exploration – *Given the current agent model, what experiment can be done to maximise the amount the agent learns about itself?*
- Storage of abstract robotic models – *Given a noisy agent model, how can an agent store just the abstract information that best describes what has be learned?*

These have been identified as the initial problems to be solved in order to have self-modelling agents and will be updated as more areas of research are discovered. The focus of this project is research in effective agent model exploration and the multiple challenges described in the "Problem Statement" section. In this project we look to validate the use of Information Theory as an effective method of agent model exploration.

Year	Placement	Team	Height
2016	1st	Rhoban Football Club	56cm
2016	2nd	ZJUDancer	58cm
2016	3rd	CIT Brains	60cm
2015	1st	CIT Brains Kid	60cm
2015	2nd	ZJUDancer	80cm
2015	3rd	Rhoban Football Club	56cm
2014	1st	CIT Brains	60cm
2014	2nd	Bold Hearts	45cm
2014	3rd	Baset Kid-Size	84cm
2013	1st	Team DARwIn	45cm
2013	2nd	AUTMan	58cm
2013	3rd	ZJUDancer	58cm

Table 1: The top three winning places in the humanoid kid-sized league for four years. Source: Robot specifications from [3], [4], [5] & [6].

1.2 Challenges

The challenges of this project can be expressed to be the following:

- Finding interesting but simple problems that exploit underlying features of search, that may show it's weaknesses and strengths as an effective model explorer.
- Comparing the effectiveness of the search in a useful way so that search performance is clear.
- Writing code that is as true to the model as possible.

1.3 Report Structure

The "Background" section we review the various existing works, as well as a review of the methods used in this project.

In the "Problem Statement" section we discuss the overall problem description, difficulties, assumptions and constraints of the project.

Moving onto the "Literature Review" section, we look at current efforts for self-modelling agents and demonstrate the difference between those and the direction in which this project takes.

Next we have the "Methodology" section, describing the experiment setup, limitations and what we aimed to learn from conducting them.

For the "Results" section we show a high-level, processed version of the data, as well as information about how it was generated.

Furthermore, the data is studied in the "Discussion" section, where we discuss what has been observed from the data and what we can learn from our observations.

The "Conclusion" section should be relatively self-evident as we collate and condense what has been learned from the series of experiments.

Looking at the "Evaluation" section offers a critical review of the project and the future direction of the next projects.

The "References" offer sources for the information cited, whilst the "Appendix" offers the source code used in the project, as well as the raw data used to draw conclusions from.

1.4 Project Statement

The purpose of this project is to evaluate information driven exploration as an effective method for model exploration, exploring the benefits of recursive search.

2 Background

2.1 Literature Review

A method for self-modelling agents used by Josh Bongard et al, currently use a Genetic Algorithm (GA) for exploration, where the best solution acts as a baseline for the next generation [9]. In this work they show how the robot can recover from simulated damage and continue to move under it's own power in a given direction which is a nice result. The exploration relies on randomly produced models with the idea that a closely matching model is picked, which essentially involves internally simulating many models. With larger degrees of freedom, the probability of randomly producing a partially acceptable model decreases significantly. The interactions between limbs is relatively low with their hardware and the different structures that the GA explores is relatively low, making the exploration using a GA easier.

This approach is formally described as a estimation-exploration algorithm (EEA) which relies on the simulation of many internal models as previously described, where fitness is based on how well matched perceived data is with input [10]. For the estimation phase a GA is used to generate 100 models, that vary in terms of masses and sensor lag (time delay for a given sensor). The fittest 50 models are used to cross breed and generate a further 50 models, which then replace the 50 lowest fitness models. This is continued over 30 generations until a model is produced.

In the exploration phase they also have 100 genomes for 30 generations, but in this phase the genomes map-to neural networks. These neural networks act as controllers, which are tested against the model produced in the estimation phase. They are then rated on the distance achieved over 1000 time steps. The result of this phase is output to the robot.

The results of this work is a robot that successfully constructs itself from a large number of assumptions about what is allowed to be a target model, including the number of parts, angle limitations, masses of parts and where sensors are located. The amount of error in this modelling is relatively low, which is one of the many gains from switching to an information theory approach to exploration.

A novel approach to this subject area is to look at how humans explore, specifically in the act of browsing for media in the paper: "A Model For Information Exploration" [11]. Here they attempt to quantify user interaction with a media system that not too dissimilar to the navigation capabilities offered by modern internet. What's interesting here is that it is not too different from the methodology that will be described later. An experiment is performed, in this case in the form of selecting media, the content is evaluated and a decision is made from what has been learned until a goal has been reached. It is interesting to see how our approach may be relatable to the way in which humans search to gain information.

In "Information Based Adaptive Robotic Exploration" we see that a successful information driven robot exploration of the world, where the robot is tasked with mapping out an indoor environment [12]. The exploration proposed in this approach uses an Occupancy Grid, thus relying on accurate location data in order to correctly associate input data with the correct grid in order to build the model. Here information density is used to improve the accuracy of the model to be stored using feature-based Simultaneous Localisation And Mapping (SLAM).

Moving towards work closely related to this project, we have "Model based Bayesian Exploration", which is focused on delivering a short sighted approximation to information related to each action in the current state when exploring a world model [13]. The result of this paper is a method to balance exploration and exploitation when choosing actions, as well as how to update Bayesian belief states using Markov Decision Processes.

Essentially what we ultimately plan to do is search a large space for non-linear relationships in high dimensions, something that kernels, deep-learning and other such practices are capable of but obscure the actual relationships. What currently isn't clear is how these relationships can be optimally explored given that we know there is a relationship based on a fundamental set of rules - even if we don't know what those rules are.

The plan for this project is to experiment with model exploration in terms of information theory, using the entropy of the internal model to identify areas worthy of exploration [14]. The reason to prefer information

theory over a GA as previously discussed is that in future projects we will have to deal with channel noise and in preparation for this, it makes sense to use methods already prepared with theory on how to tackle these issues.

With this method of exploration, we will look to reduce the entropy and increase the information in the model, where information gained is defined by $-\log(p)$ and expected information is defined by $-p \cdot \log(p)$ The problem is changed within this project in order to reflect the benefits of an information driven approach for model exploration.

2.2 Background of Bayesian Inference

Bayes' Theorem, named after Thomas Bayes who proved a special case [18], is a method in which we can deduce the correctness of several hypothesis through a method of storing the probability of them being correct. In the case of this project, we store the possible model states, referred to as world states in the context of the experiment. We keep going until it is clear that only one world state is possible, therefore we know the model.

For the simplicity of the project, we take a naive approach to Bayesian inference as seen in equation 1, we have our possible model states, W, sense states, s and action states, a.

$$p(s|a) = \sum_{W} p(s|W, a) \cdot p(W) \tag{1}$$

For each problem, we shall give the following description:

$$\begin{aligned} W &\in \{W_1, W_2, .., W_n\} \\ A &\in \{A_1, A_2, .., A_n\} \\ S &\in \{S_1, S_2, .., S_n\} \end{aligned} \tag{2}$$

Which shows which model states are possible for the given problem, which action states an agent may take and the possible sense states an agent may make.

2.3 Background of Information Theory

Information theory, or communication theory as first proposed by Claude Shannon in 1948, is where the key concept of entropy is outlined as a measurement of information [14].

An initial use of information theory based work was conducted at Bell Labs, the work place of Shannon, for the purpose of increasing the speed at which telegraphs could be sent over a noisy channel. This work was conducted by Harry Nyquist in 1924, specifically looking at the rate in which "intelligence" could be transmitted, something which was later formalised by Shannon [15].

The concept of entropy is the amount of information contained in a message transmitted from transmitter to receiver. This allows us to formally look at the information in the original message, the information lost through transmission, the compression of the information in the message, the redundancy, etc. Information theory is of course much further reaching than telegraph lines and has extended to compression, cryptography, general communication, physics and much more.

In this project, we're specifically interested in knowing how many bits of information are required to represent a state. We measure the information with equation 3, where S is the Bays state and o is an observation.

$$H(S|o) = -\sum_{S} p(s|o) \cdot \log_2 p(s|o) \tag{3}$$

The state S, represents the probability of a corresponding world state W (as seen in equation 2), as a correct hypothesis.

This of course is not enough to to determine which is better, we must know what is gained by our potential experiment given our current model. The information gain is described by:

$$I(S|o) = h(s) - h(s|o) \tag{4}$$

3 Problem Statement

3.1 Problem Description

Formally, the problem is how to explore a discrete, unknown state space effectively and show that this data is suitable for building a model.

3.2 Difficulties

The following are considered to be the difficulties of the project:

- Demonstrating the performance gain on an arbitrary problem.
- Validating whether the approach is effective.

3.3 Assumptions

In our approach we assume the following in our experiments:

- Actuators do not have noise. An example may be a motor having noise to the position it's set.
- Sensors do not have noise. Sensors are usually not perfect, particularly if they offer any kind of resolution.
- World does not have noise, such as movement or change of some type that for a static agent could be considered as noise.
- The internal model does not have noise, such as randomness or unreliability in the storage and calculation methods.
- We know the structure of the model.
- We are able to test for the model.

3.4 Constraints

In general, the only real constraint is the scope of the project, which in turn is set by the resources allocated to the project. The problem of humanoid model exploration is a complex and lengthy pursuit, something that cannot be tackled without first investing time into the individual issues posed by the problem. This project tackles a simplified version of the issue posed by exploring a large, unknown state space.

Additional limitations will be discussed in the "Critical Review" section of this project.

4 Methodology

Here we describe multiple experiments, how they are conducted and what we aim to learn from them. In the "Evaluation" section we look at what has been learned, ultimately driving the next experiment in this section. When reading, it may be useful for the purpose of order to read back and forth between the sections.

4.1 Coin Problem

With our first problem, we want to explore information driven exploration with a problem that is simple to implement and easy to understand practically. With this problem we can validate how an information driven approach would be developed and whether there is any advantage over random exploration. The problem selected is an adapted version of the fake coin problem as this is a problem we practically validate to ensure it's working correctly [16].

Our adapted version of the problem is as follows:

- There are four coins and one of them is fake, where the fake one is either lighter or heavier than the others and this is unknown. The model is aware of the number of coins and fake coins.
- The model may read the current state of the scales, as either weighted to the left, to the right or even. The state of the coins can be on the scales (left or right) and off the scales.

In figure 2 we see a visualisation of the coin problem where a coin action, A, can be in the state left, L right, R, or off, O. This real world model can be expressed as:

$$A = \{L, R, O\} \tag{5}$$

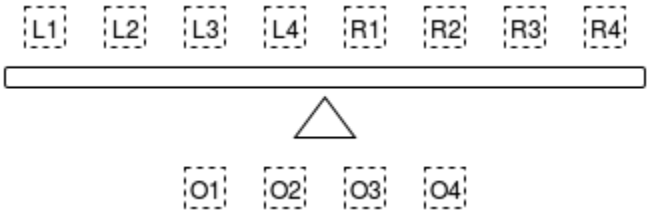

Figure 2: The possible positions of the coin in our version of the problem.

In figure 2 we can see that setting all the coins to the left side of the scales would produce a left bias, L and the opposite for a right hand bias, R. Additionally, setting all the coins off the scales or having equal weight either side of the scale will mean there is no bias and therefore is balanced, B. This sense, S, can be represented as:

$$S = \{L, R, B\} \tag{6}$$

Each coin has an additional state associated with it, in the form of whether it is the fake coin. A fake coin can be either light, L, or heavy, H. A coin may also be normal, N, which gives us the following:

$$C = \{L, H, N\} \tag{7}$$

Lastly, the Bayes model, W, is set to represent our hypothesis about the coins using the terms in equation 7:

$$\begin{aligned} W_1 &= [L, N, N, N] \\ W_2 &= [N, L, N, N] \\ W_3 &= [N, N, L, N] \\ W_4 &= [N, N, N, L] \\ W_5 &= [H, N, N, N] \\ W_6 &= [N, H, N, N] \\ W_7 &= [N, N, H, N] \\ W_8 &= [N, N, N, H] \end{aligned} \tag{8}$$

4.1.1 Random Exploration

Here we set the bench mark with random exploration to set a performance benchmark. Here the Bayesian model is updated using the random actions taken by the action.

4.1.2 1 Layer Information Gain Exploration

The purpose of this experiment is to test whether evaluating the possible information gain of taking a given action gives us an advantage when compared with taking random actions.

4.1.3 2 Layer Information Gain Exploration

Given the results of the previous test, we look to see whether there is gain in looking one layer down in the search tree.

4.1.4 3 Layer Information Gain Exploration

Lastly, we look to see whether any information can be gained in addition to one layer search for the problem by looking through all of the layers the problem should be searching.

4.2 Camera Problem

The camera problem is an attempt to bring a spatial element to the search problem given the analysis from the previous problem's experiments, where the agent is a fixed position camera with control over it's angle. The rules applied to the agent is as follows:

- An agent is only able to move left or right one space.
- The world is of fixed size (determined at the start of the simulation) and the agent has access to this information.
- Requesting left or right at a limit will not allow the agent to move.
- The agent can only sense the current position it is in and not further out. For example, an agent can detect a limit only when on that limit.

In figure 3 each possible camera angle maps to a discrete position, in this case also representing a location in which we hypothesize that a limit may be. This is used in our Bayes probabilities, where there are two distinct features that define a possible mode, a left limit and a right limit. The thick arrow represents the current camera direction which can be in any of the world positions. The camera can only gain information from the state it's currently in, meaning that in state 3 it is unaware of states 2 or 4, unless it has previously been in those states and processed them internally.

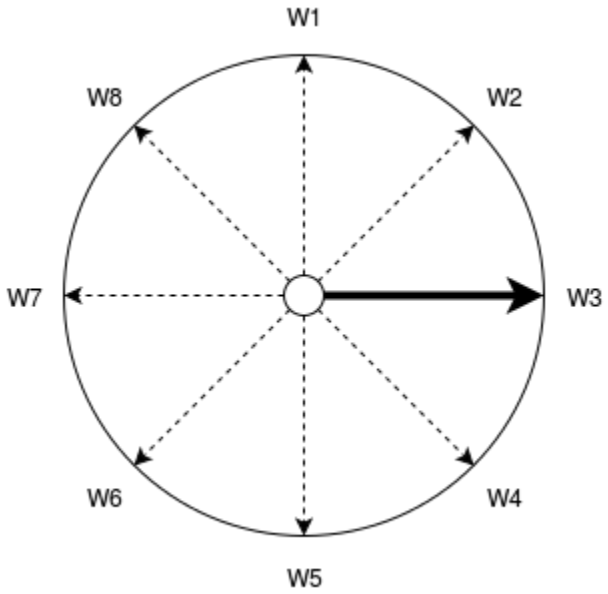

Figure 3: A basic version of the camera problem, outlining how the agent can interact with the environment.

A simple four position problem would look like the following:

$$\begin{aligned} W_1 &= [L, R, N, N] \\ W_2 &= [L, N, R, N] \\ W_3 &= [L, N, N, R] \\ W_4 &= [N, L, R, N] \\ W_5 &= [N, L, N, R] \\ W_6 &= [R, L, N, N] \\ W_7 &= [N, N, L, R] \\ W_8 &= [R, N, L, N] \\ W_9 &= [N, R, L, N] \\ W_{10} &= [R, N, N, L] \\ W_{11} &= [N, R, N, L] \\ W_{12} &= [N, N, R, L] \end{aligned} \tag{9}$$

For the equation 9, the possible world states are described by the left limit, L, the right limit, R, and no limit, N.

Actions that may be taken are described as either left, L, or right, R. This means an action state is defined as:

$$A = \{L, R\} \tag{10}$$

The sense states are defined as left limit, L, right limit, R, and N - as defined in equation 9:

$$S = \{L, R, N\} \tag{11}$$

In figure 4 we see the visual implementation of what have been discussed above, with the blue line representing the left limit, green line the right limit and the red line the current position of the camera.

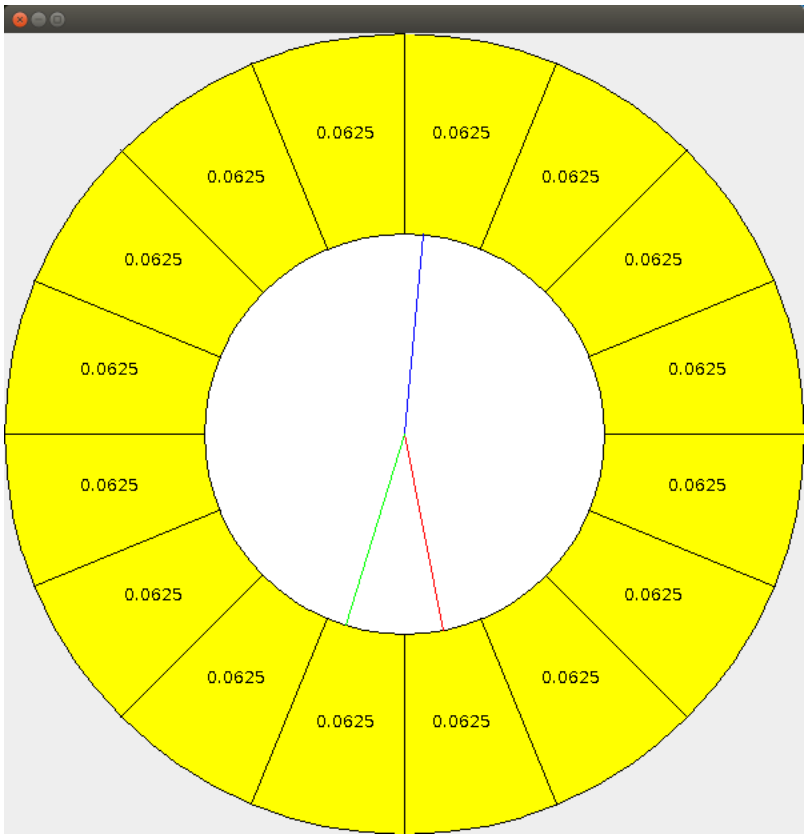

Figure 4: The GUI representation of the problem for the purpose of debugging and demoing the program.

Additionally, in this problem it has been necessary to set limits as to how long we allow the problem to process. This limit was set to 100 as initial experimentation suggested most random searches had completed by this point.

4.2.1 Random Exploration

Here we want a baseline comparison for the performance of the information driven search method. Our method should in theory be better than random actions being taken. For the purpose of the experiments having a reasonably timed end, the problems have been kept small.

4.2.2 1 Layer Information Gain Exploration

Like the coin problem, we now look at the advantages of exploring the problem based on the expected information gain.

4.2.3 2 - 6 Layer Information Gain Exploration

From the previous results, we want to see whether there is any increase in benefit from exploring further layers.

4.2.4 7 Layer Information Gain Exploration

A further experiment to see whether a limitation has been met in the amount of increase seen in depth exploration.

4.2.5 8 Layer Information Gain Exploration

To understand whether there is any change in behaviour when the depth explored is the same size of the environment.

4.2.6 Open-Loop Exploration

In these experiments, all previous experiments on this problem are repeated in order to explore how the problems behave on an open-loop problem, where the limits are removed. The environment has been set to permanently open so that the specific behaviour can be isolated, although the agent still has to determine that all previous world states are not valid.

5 Results

5.1 Coin Problem

Below is a comparison of the different types of experiments as explained in the "Methodology" section, where the average result is from raw data in the Appendix ("Coin Problem Raw Results").

Exploration Type	Average
Random	15.3546453546
1 Layer Information Gain	3.2427572428
2 Layer Information Gain	3.2457542458
3 Layer Information Gain	3.2577422577

Table 2: A comparison of the different search experiments performed.

5.2 Camera Problem

Here is a comparison of the of the different experiments outlined in the "Methodology" section, where we have the average result with the limit results excluded and the percentage of experiments that failed to reach a result. The raw results can again be seen in the Appendix ("Camera Problem Raw Results").

Exploration Type	Average	Failure (%)
Random	21.7446373851	2.1978021978
1 Layer Information Gain	5.5788667688	34.7652347652
2 Layer Information Gain	5.6780551905	23.976023976
3 Layer Information Gain	6.3220747889	17.1828171828
4 Layer Information Gain	6.6230769231	9.0909090909
5 Layer Information Gain	7.0671487603	3.2967032967
6 Layer Information Gain	7.3436563437	0
7 Layer Information Gain	7.4805194805	0
8 Layer Information Gain	7.2957042957	0

Table 3: A comparison of the different search experiments performed.

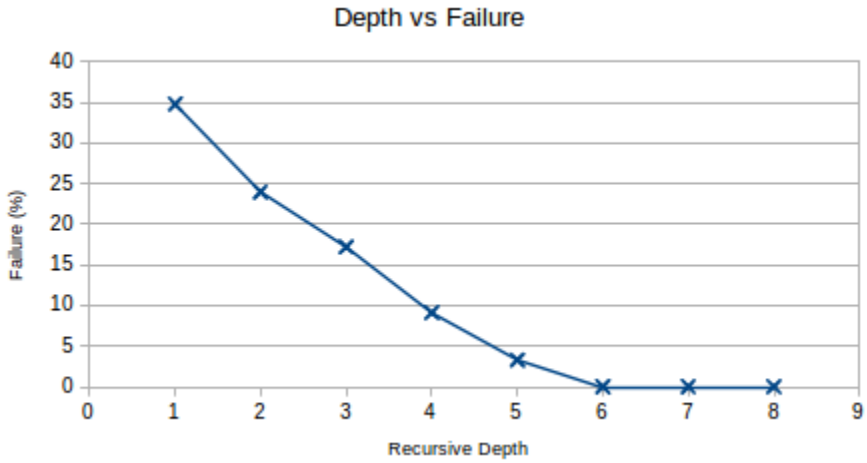

Figure 5: A graph showing the failure rate.

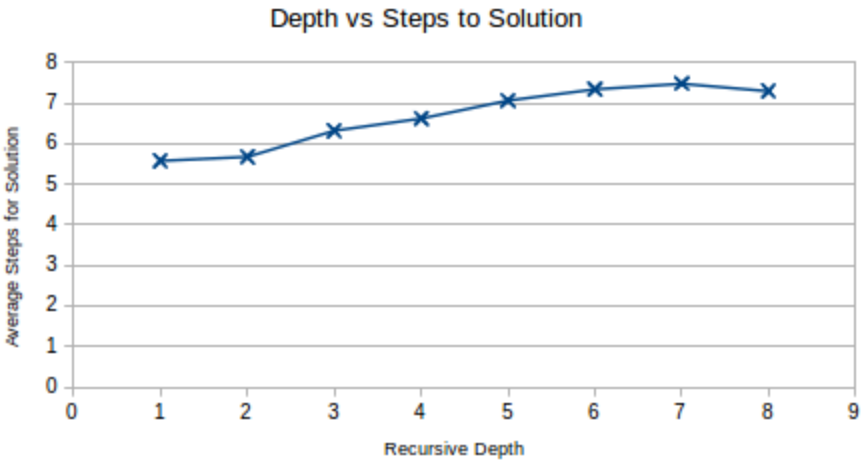

Figure 6: A graph showing the steps till solution.

Table 4 contains the results collected from the open-loop version of this problem.

Exploration Type	**Average**	**Failure (%)**
Random	21.5994005994	0
1 Layer Information Gain	7	0
2 Layer Information Gain	7	0
3 Layer Information Gain	7	0
4 Layer Information Gain	7	0
5 Layer Information Gain	7	0
6 Layer Information Gain	7	0
7 Layer Information Gain	7	0
8 Layer Information Gain	7	0

Table 4: A comparison of the different search experiments performed in the open-loop scenario of the camera problem.

6 Discussion

6.1 Coin Problem

These experiments refer to table 2.

6.1.1 Random Exploration

In table 2, for the random exploration we see that the problem takes a significant number of moves to solve. This is because a step towards progress is equally matched by step away from progress. There are three possible positions for the coin to be in. The probability of a coin being placed in a given position is $\frac{1}{3}$ and the probability of all the coins being in a given state is $\frac{1}{2}^4$.

It is expected that only a given number of states infer new information and therefore help solve the problem, but as the particular states change per randomly generated problem it is difficult avenue to explore. It's worth noting that no limit was required on the problem as the number of distinct possible states is small, meaning that randomness can easily explore most in little time and therefore infer the fake coin.

6.1.2 1 Layer Information Gain Exploration

This experiment gave us a significant gain on random exploration, at about 4.7 times better it shows itself as an affective method for exploration as expected. At just over 3 comparisons on average, this matched the best solution that could be achieved as a human too.

From this experiment, it appears it may be useful to look specifically at whether there is continued gain by looking at other depths.

6.1.3 2 Layer Information Gain Exploration

Unfortunately, this experiment didn't show any gain, something which was expected. The reason for this seems to be that one layer look ahead seems to be ample to distinguish any future gain there is to have. Doing those most optimal move for the current state is also the most optimal move for any future moves too. There's no need to do something counter-intuitive for gain.

Moving forward, we shall look at three layer gain exploration encase there is any gain to be had by seeing the entire tree when making decisions.

6.1.4 3 Layer Information Gain Exploration

As expected, there was no gain from performing this experiment. For this particular problem there doesn't appear to be any benefit from sacrificing a current gain in order for future gain.

6.1.5 Evaluation

This appears to be a special case of exploration where a greedy search implementation is just as affective as an information gain driven one, due to there not being cases where a local sacrifice in information is required in order to gain globally.

With this, we want to design a problem that is more likely to require a local loss of information in order to model the world in less steps. In order to get this demonstration, we need a problem with the concept of a local maximum and global maximums. The simplest example we could think of to demonstrate this was the "Camera Problem"", as seen in the "Methodology" section.

6.2 Camera Problem

These experiments refer to table 3.

6.2.1 Random Exploration

Firstly, with the expectation of this experiment not being a simple as the last and that the agent may get stuck at a local minima and unable to complete the problem, a limit was set on how long it was deemed reasonable to wait for an agent to find a solution. The failure percentage is the number of incidents where we see that hit the set limit, potentially taking infinite time to complete.

The random exploration set a pretty low benchmark to beat, being much lower than simply turning left 8 times and then right 8 times, which would mean 16 moves using an inefficient algorithm. That said, the number of failure cases is low, at about 2.2% it's able to find a solution in a reasonable amount of time.

6.2.2 1 Layer Information Gain Exploration

With the failure cases removed, the one layer exploration method looks suspiciously efficient. After some manual debugging, it appears that the it's because the agent can only solve simple cases where not much movement is required. What raises the number is the bias to move left or right when the information gain is even and no preference exists.

On average we can assume that the limits are set opposite to one another, meaning there is half the distance between them. Assuming a bias to the right hand side, we automatically gain the benefit of a lookahead for 7 positions. Then we consider the agent's capability, which is the information they currently have and one step left or right. This gives us:

$$((1L + 0 + 1R)/3) + ((2R + 3R + 4R + 5R + 6R + 7R + 8R)/7) = 5.\overline{6} \tag{12}$$

The letter after the number is an indication to which side the value is associated with. This agrees approximately what we have seen.

6.2.3 2 - 5 Layer Information Gain Exploration

Following these experiments, we see the failure rate deduce as it becomes less and less likely the agent will become stuck at a local maximum as the agent can effectively see further and further, as seen in figure 5. Given more lookahead capability, the agent is able to take a short-term sacrifice when given the possibility of gaining information in a future state. The average also comes up as it starts to better represent all possible states and not dismissing the cases where the agent is stuck. In figure 6 we it level off as the increase in search depth no longer yields any benefits, with infinite values (, steps greater than or equal to 100,) no longer having to be removed.

6.2.4 Limit Of Information Gain Exploration

Exploring down to layers 6, 7 and 8 we clearly see no gain from further exploration. This of course makes sense as once you have found a limit, no further exploration as you don't need to explore a limit you're on and the other limit. $8states - 2limits$ gives us the 6 lookahead states required to find a solution.

6.2.5 Open-Loop Exploration

For this environment, looking at the random search implementation we see a drop in failure rate for the open-loop version of this problem and a slight decrease in the number of steps required to find a solution. For the other layers, there is no information gain from going left or right, so it defaults to one side after seeing no preference. Of course, the agent needs to explore the entirety of the environment in order to discover that there is in fact no limits in the environment.

The last aspect of the results worthy of note is the failure rate, which is 0. This does not come as a surprise as agent is not able to get stuck and therefore reach a state where it cannot escape a local maximum. This does validate that the implementation works correctly and our previous theory about the failure rate being due to the agent becoming stuck on a limit.

6.2.6 Evaluation

It is evident we have been able to demonstrate what we set out to achieve with this problem, which is temporary non-obvious movement in order to gain information.

7 Conclusion

In this project, we want to know whether information driven exploration is an effective method for model exploration and if recursive search adds any additional advantage when compared to search at a single depth. This was set as an initial goal in order to validate information theory as a valid approach for future works.

Looking at the coin problem, we see that random exploration for small problems is relatively effective and sets a good benchmark for any algorithm to compete with it. The random search always returned a solution in the end, taking on average approximately 15 moves before randomly stumbling taking the correct actions that allow the Bayes model to come to a conclusion.

Next we investigated the implementation of searching one layer ahead using an information driven approach. This gave us a significant improvement when compared with random search as expected. What wasn't initially expected was the that this produced the most optimal solution. Recursive search didn't give us any increased advantage, meaning that a greedy algorithm is fully able to solve the coin problem optimally.

This result did mean that the coin problem wasn't able to demonstrate whether a recursive search is able to enhance the speed in which we can derive a solution. It was decided that a simple problem was needed in order to determine whether a recursive search could navigate itself out of a scenario where it is possible to get stuck at a local maxima.

The problem that seemed suitable was the camera problem, where there are left and right limits to explore. With this problem it was possible to get stuck at one of the limits, where there was zero information gain from an immediate decision to turn left or right. This result was later confirmed by a further set of experiments with no limits given to the agent.

The random implementation again set a good benchmark for what is to be bettered upon, with the more layers the better up until the search depth was six layers down, at which point there was no further performance gain. Interestingly we also saw that percentage of time the agent was stuck in it's exploration also decreased drastically as the search depth increased.

From these results, we see that information-driven exploration could be considered a replacement for the GA used by Bongard et al's work on self-modelling agents, potentially allowing the complexity of the problem to be increased also. As the potential candidates for correct models decreases, the speed at which correct models can be isolated should increase.

The advantage with this approach is that a further reduction in physical experimentation is required, as the information gain from each experiment should be significantly increased from that of semi-random actions. This of course is an important feature when hardware as a resource is both costly and time consuming to maintain, whereas computation becomes increasingly cheap in comparison.

8 Evaluation

8.1 Critical Review

8.1.1 Choice of Language

Unfortunately, Java as a choice of language for exploring mathematical ideas was not ideal. Several times there were problems that extended through the language, such as:

- Accuracy issues with dealing with small numbers. Normalising on the same array of `double` values can return different results, despite not changing the numbers. Doing the same experiment twice introduces minor discrepancies. As the search depths grew greater, the more noticeable this issue was. The values returned typically ranged from -0.0 to $< \pm 1 \times 10^{14}$ after normalisation. This is because the normalisation method sums the set of values, divides 1 by the sum and times each value in the set by that number. A world of size N has a Bays model of size $n.(n-1)$, meaning if it is size 16 (as it was during testing), there were 240 unique states. Over several calls it's possible to see this error propagate in the normalisation process.
- Visualisation of concepts isn't intuitive in Java and actually requires a lot of time to working on how the back end and front end will work together, as the display thread is blocked by the worker thread, causing the GUI to appear unresponsive to the Operating System and crashing the program. On top of this, for arbitrary diagrams you're also required to manually figure out how to draw the display through basic shapes, hence why the visualisation of the camera problem is so basic. Other languages such as C# do appear to offer a more intuitive ability to interact with the display.

There are some positive points with Java, though:

- Clear structures for primitives and core classes means that it's possible to make some assumptions about how the code will operate. If an array of `double` values is created, you can make guarantees that the variable won't suddenly be replaced with a `String` for example.
- Speed is more than good, with on-the-fly speed improvements from Just-In Time (JIT) execution, making experiments fast to run. Some experiments took a number of hours to run, which may have been much more if the language wasn't as fast as it is.

It is currently unclear whether there is a language that encapsulates all of the requirements this project has had, but is certainly worth an investment of time to prevent the same teething problems seen in this project.

8.1.2 Project Scope

In hindsight, it's better to start off with a small idea and expand on it. At the start of the project there was a lot of time and resources wasted on ideas that were too broad to write about, let alone to consider experimenting with. As a result a lot of initial experimentation was not included in this report due to either being incomplete or not yielding any results.

8.1.3 Performance Measures

Ideally, it would have been good to compare the results from the model exploration as discussed in the "Background" section. As part of their experimentation, Bongard et al were using a relatively complex star-shaped robot with real hardware, making a direct comparison difficult. Exploring whether their solution was strictly an optimisation to a robotics platform or more general ended up outside of project due to the complexity.

8.1.4 Knowledge of Area

It was clear that after a good section of the way through the project that there was a gap knowledge of the mathematics behind the project. Admittedly this did hinder the writing of the code for the experiments, meaning at one particular point it was not worth while correcting the problem programmed given the option

of starting again.

With future projects there is certainly benefit to prototyping a basic idea to check that it is both understood and feasible to be included as part of the project. Validating the feasibility of a method as part of a solution also validates the understanding on that problem.

8.2 Future Work

8.2.1 Use of Information

In our exploration search, we choose the best average information gain from our searching. It may be interesting to explore the effects of using a MaxMax, MinMax or MaxMin algorithm for determining the next action for an agent to take.

This may in turn lead to interesting questions, such as "Is it more important to retrieve some information from an experiment, rather than trying to retrieve the maximum information from an experiment?". "What exploration behaviour does MinMax inhibit?"

8.2.2 Model Storage

Particularly for the problem of exploring robotic structures, we have a compact form in which power is delivered to a joint which can be modelled as a series of wheels with limbs attached. In theory, there may be infinite structures in which are reasonably possible for a robot to be made and a discrete approach may not be appropriate. It's not yet clear how information could be used to infer a specific model in a possibly infinite space.

8.2.3 Optimisation of Search

One line of discussion whilst conducting the experiment was how to skip having to search every possible state in the space, by looking at where linear relationships might occur in the Bayes model. This in turn could allow for faster searching over large search spaces. The idea is to have a probability associated with an unexplored Bayes world state that indicates the how likely it is to inherit a linear property set by local values.

9 References

[1] Kitano, H. "RoboCup-97: Robot Soccer World Cup I", *Springer*, Vol. 1, 1997

[2] RoboCup Federation. "Objective - The Dream", URL: `http://www.robocup.org/about-robocup/objective/`, 2016

[3] RoboCup Federation. "Qualified teams for RoboCup 2016", URL: `https://www.robocuphumanoid.org/hl-2016/teams/`, 2016

[4] RoboCup Federation. "Qualified teams for RoboCup 2015", URL: `https://www.robocuphumanoid.org/hl-2015/teams/`, 2015

[5] RoboCup Federation. "Qualified teams for RoboCup 2014", URL: `https://www.robocuphumanoid.org/hl-2014/teams/`, 2014

[6] RoboCup Federation. "Qualified teams for RoboCup 2013", URL: `https://www.robocuphumanoid.org/hl-2013/teams/`, 2013

[7] RoboCup Federation. "Rules", URL: `https://www.robocuphumanoid.org/materials/rules/`, 2015

[8] Vukobratović, M., Borovac, B. "Zero-Moment Point - Thirty Five Years of it's Life", *International Journal of Humanoid Robotics*, Vol. 1, No. 1, pp. 157-173, 2004.

[9] Bongard, J., Zykov, V., Lipson, H. "Resilient Machines Through Continuous Self-Modeling", *Science*, Vol. 314, No. 5802, pp. 1118-1121, 2006.

[10] Bongard, J., Lipson, H. "Automatic Synthesis of Multiple Internal Models Through Active Exploration", *AAAI Fall Symposium: From Reactive to Anticipatory Cognitive Embodied Systems*, 2005.

[11] Waterworth, J., Chignell, M. "A Model of Information Exploration", *Hypermedia*, Vol. 3, pp. 35-58, 1991.

[12] Bourgault, F., Makarenko, A., Williams, S., Grocholsky, B., Durrant-Whyte, H. "Information Based Adaptive Robotic Exploration", *IEEE/RSJ International Conference: Intelligent Robots and Systems*, Vol. 1, 2002.

[13] Dearden, R., Friedman, N., Andre, D. "Model based Bayesian Exploration", *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, Vol. 15, pp. 150-159, 1999.

[14] Shannon, C. "A Mathematical Theory of Communication", *The Bell System Technical Journal*, Vol. 27, pp. 379-423, 623-656, 1948.

[15] Nyquist, H. "Certain Factors Affecting Telegraph Speed", *The Bell System Technical Journal*, Vol. 3, pp. 324-346, 1924.

[16] Levitin, A., Levitin, M. "Algorithmic Puzzles", *Oxford University Press*, Vol. 10, 2011.

[17] Bold Hearts. "Bold Hearts - The University of Hertfordshire RoboCup Team", URL: `http://robocup.herts.ac.uk/`, 2015

[18] Bayes, M. "An Essay towards Solving a Problem in the Doctrine of Chances. By the Late Rev. Mr. Bayes, F. R. S. Communicated by Mr. Price, in a Letter to John Canton, A. M. F. R. S.", *Philosophical Transactions*, Vol. 53, pp. 370-418, 1763.

10 Appendix

In this section of report we see the source code to various problems discussed in the project (titled “Coin Problem Source Code” and “Camera Problem Source Code”), as well as the raw results (“Coin Problem Raw Results” and “Camera Problem Raw Results”), for the purpose of transparency and accountability.

10.1 Coin Problem Source Code

In this section is the source code for the coin problem. Please contact `danbarry16@googlemail.com` for read access to:

- `https://bitbucket.org/danielbarry/final-year-project`
- `git@bitbucket.org:danielbarry/final-year-project.git`

10.1.1 run.sh

```
#!/bin/bash

function test {
  cd dist
    result=$(java -jar coin.jar $1)
    readarray -t y <<<"$result"
    for i in "${y[@]}"
    do
      echo -e $i | grep "##"
    done
  cd ..
}

function main {
  for x in {0..1000}
  do
    test $1
  done
}

main $1
```

10.1.2 build.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="coin" default="jar" basedir=".">
  <description>An experiment to test for fake coins.</description>

  <property name="source.dir" location="src/" />
  <property name="build.dir" location="bin/" />
  <property name="jar.dir" location="dist/" />
  <property name="doc.dir" location="doc/" />
  <property name="main.class" value="barray.coin.Main" />
  <property name="main.jar" value="coin.jar" />

  <target name="clean" description="Clean Binaries">
    <delete dir="${build.dir}" />
    <delete dir="${jar.dir}" />
  </target>

  <target name="doc" description="Create Documentation">
    <delete dir="${doc.dir}" />
    <mkdir dir="${doc.dir}" />
    <javadoc destdir="${doc.dir}">
      <fileset dir="${source.dir}" />
    </javadoc>
  </target>

  <target name="compile" depends="clean" description="Compile Java Code">
    <mkdir dir="${build.dir}" />
    <javac srcdir="${source.dir}" destdir="${build.dir}" debug="true" includeantruntime="
        false" />
  </target>

  <target name="jar" depends="compile" description="Create Jar File">
    <mkdir dir="${jar.dir}" />
    <jar destfile="${jar.dir}/${main.jar}" basedir="${build.dir}">
      <manifest>
        <attribute name="Main-Class" value="${main.class}" />
      </manifest>
    </jar>
  </target>

  <target name="run" depends="jar" description="Run Java App">
    <java jar="${jar.dir}/${main.jar}" fork="true" />
  </target>
</project>
```

10.1.3 Main.java

```
package barray.coin;

/**
 * Main.java
 *
 * The main entry class where input from the user is to be handled and the
 * program is to be initialised.
 **/
public class Main{
  /**
   * main()
   *
   * The main entry method for the program where user input is passed and
   * analysed before the program instance is generated and ran.
   *
   * @param args The arguments from the terminal.
   **/
  public static void main(String[] args){
    /* Create new problem */
    Problem problem = new Problem();
    /* Ask for the problem to be solved */
    int selection = Integer.parseInt(args[0]);
    switch(selection){
      case 0 :
        problem.solve(PROBLEM_SOLVER.RANDOM);
        break;
      case 1 :
        problem.solve(PROBLEM_SOLVER.INFO_GAIN_1_DPTH);
        break;
      case 2 :
        problem.solve(PROBLEM_SOLVER.INFO_GAIN_2_DPTH);
        break;
      case 3 :
        problem.solve(PROBLEM_SOLVER.INFO_GAIN_3_DPTH);
        break;
      case 4 :
        problem.solve(PROBLEM_SOLVER.INFO_GAIN_4_DPTH);
        break;
      case 5 :
        problem.solve(PROBLEM_SOLVER.INFO_GAIN_5_DPTH);
        break;
    }
  }

  /**
   * log()
   *
   * Outputs a log message to the standard output. This can then be grepped
   * from the output and loaded into a data processor as a CSV file.
   *
   * @param msgs An array of String messages to be output.
   **/
  public static void log(String[] msgs){
    /* Display message on standard stream */
    System.out.print(">>,");
    /* Iterate through messages */
    for(String str : msgs){
      System.out.print(str + ",");
    }
    /* Print new line */
    System.out.println("");
  }

  /**
   * error()
   *
```

```
   * Displays an error then exits the program. It is assumed that the error is
   * so bad that it's unsafe for the program to continue.
   *
   * @param msg The message to be displayed to the user before exiting as a
   * reason for doing so.
   **/
  public static void error(String msg){
    /* Display message on error stream */
    System.err.println("COIN [!!] " + msg);
    /* Now exit */
    exit();
  }

  /**
   * exit()
   *
   * Exits from the program with prejudice. Anybody can call but resources are
   * not handled before death.
   **/
  public static void exit(){
    /* NOTE: Assume good exit. */
    System.exit(0);
  }
}
```

10.1.4 Problem.java

```
package barray.coin;

import java.util.Random;

/**
 * Problem.java
 *
 * The problem fully expressed.
 **/
public class Problem{
  /**
   * Problem()
   *
   * The initialiser for the problem.
   **/
  public Problem(){
    /* TODO: Initialise the problem correctly. */
  }

  /**
   * solve()
   *
   * This method is responsible for solving the problem based on the requested
   * solver and blocks until the problem is solved.
   *
   * @param solver The type of the solver to be used to solve the problem.
   **/
  public void solve(PROBLEM_SOLVER solver){
    /* Make sure a solver is provided */
    if(solver == null){
      Main.error("Solver should not be NULL");
    }
    /* Randomly produce problem problem */
    int pos = (new Random()).nextInt(WORLD_STATE.values().length);
    COIN[] prob = WORLD_STATE.values()[pos].getWorldState();
    /* Select the correct solver */
    switch(solver){
      case RANDOM :
        (new RandomSearch()).process(prob);
        break;
      case INFO_GAIN_1_DPTH:
        (new InformationGain(1)).process(prob);
        break;
      case INFO_GAIN_2_DPTH:
        (new InformationGain(2)).process(prob);
        break;
      case INFO_GAIN_3_DPTH:
        (new InformationGain(3)).process(prob);
        break;
      case INFO_GAIN_4_DPTH:
        (new InformationGain(4)).process(prob);
        break;
      case INFO_GAIN_5_DPTH:
        (new InformationGain(5)).process(prob);
        break;
      default :
        Main.error("Solver not implemented");
        break;
    }
  }
}
```

10.1.5 RandomSearch.java

```
package barray.coin;

import java.lang.Math;
import java.util.Random;

/**
 * RandomSearch.java
 *
 * This class is responsible for doing a random action based search.
 **/
public class RandomSearch{
  /**
   * Random()
   *
   * The initialiser method for solving the problem.
   **/
  public RandomSearch(){
    /* Do nothing */
  }

  /**
   * process()
   *
   * Solve the problem given, block until complete.
   *
   * @param goal The solution to the problem.
   **/
  public void process(COIN[] goal){
    /* Bayesian model */
    double[] values = new double[WORLD_STATE.values().length];
    /* Setup Bayesian model */
    for(int x = 0; x < WORLD_STATE.values().length; x++){
      /* Setup the probability of each as equal */
      values[x] = 1.0 / WORLD_STATE.values().length;
    }
    /* Normalise out Bayesian model */
    values = normalise(values);
    /* Keep searching until one conclusion is found */
    int count = 0;
    for(;;){
      count++;
      ///* Log what we currently know */
      //logData(values);
      /* Calculate current information */
      double w = getEntropy(values);
      Main.log(new String[]{"world_entropy", "" + w});
      /* If there is no world entropy, we have solved the problem */
      if(w == 0){
        break;
      }
      /* Calculate the best action to be taken at a given depth */
      ACTION_STATE[] bestCoins = getBestAction(goal.length);
      /* Action on the state */
      SENSE_STATE measurement = getMeasurement(bestCoins, goal);
      /* Update information we gained in our world from the real experiment */
      values = updateBayesian(values, bestCoins, measurement);
    }
    /* Find out which solution we came to */
    int solution = -1;
    for(int x = 0; x < values.length; x++){
      if(values[x] == 1){
        solution = x;
      }
    }
    /* Print the solution */
    COIN[] solvedCoins = WORLD_STATE.values()[solution].getWorldState();
```

```
    for(int x = 0; x < solvedCoins.length; x++){
      System.out.println("solvedCoins[" + x + "] -> " + solvedCoins[x].toString());
    }
    /* Display the number of iterations required */
    System.out.println("##," + count);
  }

  /**
   * getBestAction()
   *
   * Randomly looks for the best action to be taken.
   *
   * @param size The size of the problem.
   *
   * @return The best action state that was found.
   **/
  private ACTION_STATE[] getBestAction(int size){
    /* Hard-coded values */
    int numFake = 1;
    /* The action state coins */
    ACTION_STATE[] bestCoins = new ACTION_STATE[size];
    /* Generate next action state */
    Random rand = new Random();
    for(int x = 0; x < size; x++){
      bestCoins[x] = ACTION_STATE.values()[rand.nextInt(ACTION_STATE.values().length)];
    }
    /* Return the action state coins */
    return bestCoins;
  }

  /**
   * getAvgInfo()
   *
   * Calculates the average information gained from a move by searching the
   * tree recursively to a given depth.
   *
   * @param currentDepth The current depth of the recursion.
   * @param currentWorldBay The current world Bayesian given the recursive
   * depth.
   * @param coins The proposed arrangement of the coins to be tested.
   * @return Average information gained from action at a given depth.
   **/
  private double getAvgInfo(int currentDepth, double[] currentWorldBay, ACTION_STATE[] coins
      ){
    /* Iterate through different sensor options for proposed sensor states */
    double avgInfo = 0;
    int avgCnt = 0;
    for(SENSE_STATE ss : SENSE_STATE.values()){
      /* Calculate Bayesian model based on new information */
      double[] bay = updateBayesian(currentWorldBay, coins, ss);
      /* Check whether the predicted state is impossible */
      double baySum = 0;
      for(int x = 0; x < bay.length; x++){
        baySum += bay[x];
      }
      if(baySum > 0){
        if(currentDepth > 1){
          /* All possible action states */
          int wLen = WORLD_STATE.values()[0].getWorldState().length;
          int aLen = ACTION_STATE.values().length;
          int aCombo = (int)(Math.pow(aLen, wLen));
          double avgLayerInfo = 0;
          for(int z = 0; z < aCombo; z++){
            ACTION_STATE[] nCoins = new ACTION_STATE[wLen];
            int action = z;
            /* Figure out where each coin would be in each state */
            for(int y = 0; y < wLen; y++){
              nCoins[y] = ACTION_STATE.values()[action % aLen];
```

```
            action /= aLen;
          }
          avgInfo += getAvgInfo(currentDepth - 1, bay, nCoins);
          /* Add to average information count */
          avgCnt++;
        }
      }else{
        /* Calculate the information */
        avgInfo += getEntropy(bay);
        /* Add to average information count */
        avgCnt++;
      }
    }
  }
  /* Average the information gained */
  if(avgCnt > 0){
    /* Calculate the average information for gained in each state */
    avgInfo /= avgCnt;
  }else{
    /* TODO: Possibly troubling case where no states were possible and we are
             forced to take an average due to recursion. */
    Main.error("Unhandled entropy case hit.");
  }
  /* Return the current average information as we have reached the end */
  return avgInfo;
}

/**
 * updateBayesian()
 *
 * Calculate the new values of the Bayesian model based on the current
 * Bayesian model and the given new information.
 *
 * @param bay The old Bayesian model.
 * @param pos The position of the coins as we prepared them for the experiment.
 * @param ss The new sensed state.
 * @return The new Bayesian model.
 **/
private static double[] updateBayesian(double[] bay, ACTION_STATE[] pos, SENSE_STATE ss){
  /* Make a copy of previous array with new data */
  double[] newBay = new double[bay.length];
  for(int x = 0; x < bay.length; x++){
    /* Store old value */
    newBay[x] = bay[x];
    /* Get the coins for the proposed world state */
    COIN[] coins = WORLD_STATE.values()[x].getWorldState();
    /* Calculate the sense state we would predict to see */
    SENSE_STATE predicted = getMeasurement(pos, coins);
    /* Check if that matches what we actually saw */
    if(predicted != ss){
      /* Can't be a correct hypothesis - wipe it */
      newBay[x] = 0;
    }
  }
  return normalise(newBay);
}

/**
 * getMeasurement()
 *
 * Gets the weighing of the coins.
 *
 * @param pCoins The abstract positions of the coins that we propose to measure.
 * @param aCoins The abstract positions of the coins that we actually measure.
 * @return The abstract bias to the scales.
 **/
private static SENSE_STATE getMeasurement(ACTION_STATE[] pCoins, COIN[] aCoins){
  /* NOTE: Left is negative, right is positive. */
```

```
    double bias = 0;
    for(int x = 0; x < pCoins.length; x++){
      /* Find out the amount of weight to be added */
      double coinWeight = 0;
      switch(aCoins[x]){
        case NORM :
          coinWeight = 1.0;
          break;
        case LITE :
          coinWeight = 0.5;
          break;
        case HEVY :
          coinWeight = 1.5;
          break;
      }
      /* Assign the coin weight in the correct position */
      switch(pCoins[x]){
        case LEFT :
          bias -= coinWeight;
          break;
        case OFF :
          /* None, not weighed */
          break;
        case RIGHT :
          bias += coinWeight;
          break;
      }
    }
    /* Figure out what we sensed from the calculated bias */
    if(bias < 0){
      return SENSE_STATE.LEFT;
    }
    if(bias > 0){
      return SENSE_STATE.RIGHT;
    }
    return SENSE_STATE.MIDDLE;
  }

  /**
   * logData()
   *
   * Log the data to the terminal for later reading.
   *
   * @param values The values to be displayed.
   **/
  private static void logData(double[] values){
    /* Create a string array of the data we want to print */
    String[] data = new String[values.length];
    for(int x = 0; x < values.length; x++){
      data[x] = "" + values[x];
    }
    /* Output to the terminal */
    Main.log(data);
  }

  /**
   * getEntropy()
   *
   * Gets the entropy of a given set of probabilities.
   *
   * @param set The normalised set of probabilities to be checked.
   * @return The entropy of the given set.
   **/
  private static double getEntropy(double[] set){
    /* Sum all the entropy probabilities */
    double entropy = 0;
    for(int x = 0; x < set.length; x++){
      /* Make sure we're not calculating a zero case */
```

```
      if(set[x] > 0){
        /* Calculate entropy */
        entropy += set[x] * (Math.log(set[x]) / Math.log(2));
      }
    }
    return -entropy;
  }

  /**
   * normalise()
   *
   * Normalise a set of data.
   *
   * NOTE: This method is capable of normalising a zeroed set too.
   *
   * @param set The set to be normalised.
   * @return A copy of the normalised set.
   **/
  private static double[] normalise(double[] set){
    /* Create new set */
    double[] nSet = new double[set.length];
    /* Count value of existing data */
    double c = 0;
    for(int x = 0; x < set.length; x++){
      c += set[x];
    }
    /* Check whether C is zero */
    if(c != 0){
      /* Calculate normalisation factor */
      double n = 1 / c;
      /* Normalise the data using normalisation factor */
      for(int x = 0; x < set.length; x++){
        nSet[x] = set[x] * n;
      }
    }else{
      /* Impossible state, do nothing */
    }
    /* Return the normalised set */
    return nSet;
  }
}
```

10.1.6 InformationGain.java

```
package barray.coin;

import java.lang.Math;

/**
 * InformationGain.java
 *
 * This class is responsible for doing a simple information gain based search
 * with the option of search depth level.
 **/
public class InformationGain{
  private int depth;

  /**
   * InformationGain()
   *
   * The initialiser method for solving the problem.
   *
   * @param depth The depth of the tree to search.
   **/
  public InformationGain(int depth){
    if(depth < 1){
      Main.error("Search depth not possible");
    }
    this.depth = depth;
  }

  /**
   * process()
   *
   * Solve the problem given, block until complete.
   *
   * @param goal The solution to the problem.
   **/
  public void process(COIN[] goal){
    /* Bayesian model */
    double[] values = new double[WORLD_STATE.values().length];
    /* Setup Bayesian model */
    for(int x = 0; x < WORLD_STATE.values().length; x++){
      /* Setup the probability of each as equal */
      values[x] = 1.0 / WORLD_STATE.values().length;
    }
    /* Normalise out Bayesian model */
    values = normalise(values);
    /* Keep searching until one conclusion is found */
    int count = 0;
    for(;;){
      count++;
      ///* Log what we currently know */
      //logData(values);
      /* Calculate current information */
      double w = getEntropy(values);
      Main.log(new String[]{"world_entropy", "" + w});
      /* If there is no world entropy, we have solved the problem */
      if(w == 0){
        break;
      }
      /* Calculate the best action to be taken at a given depth */
      ACTION_STATE[] bestCoins = getBestAction(depth, values, w);
      /* Action on the state */
      SENSE_STATE measurement = getMeasurement(bestCoins, goal);
      /* Update information we gained in our world from the real experiment */
      values = updateBayesian(values, bestCoins, measurement);
    }
    /* Find out which solution we came to */
    int solution = -1;
```

```
    for(int x = 0; x < values.length; x++){
      if(values[x] == 1){
        solution = x;
      }
    }
    /* Print the solution */
    COIN[] solvedCoins = WORLD_STATE.values()[solution].getWorldState();
    for(int x = 0; x < solvedCoins.length; x++){
      System.out.println("solvedCoins[" + x + "] -> " + solvedCoins[x].toString());
    }
    /* Display the number of iterations required */
    System.out.println("##," + count);
  }

  /**
   * getBestAction()
   *
   * Recursively looks for the best action to be taken.
   *
   * @param searchDepth The depth of the recursion to search for the information gained.
   * @param worldBayesian The world Bayes model.
   * @param worldEntropy The entropy of the world.
   * @return The best action state that was found.
   **/
  private ACTION_STATE[] getBestAction(int searchDepth, double[] worldBayesian, double
      worldEntropy){
    /* Best next steps possible */
    double bestInfoGain = 0;
    ACTION_STATE[] bestCoins = null;
    /* All possible action states */
    int wLen = WORLD_STATE.values()[0].getWorldState().length;
    int aLen = ACTION_STATE.values().length;
    int aCombo = (int)(Math.pow(aLen, wLen));
    for(int z = 0; z < aCombo; z++){
      ACTION_STATE[] coins = new ACTION_STATE[wLen];
      int action = z;
      /* Figure out where each coin would be in each state */
      for(int y = 0; y < wLen; y++){
        coins[y] = ACTION_STATE.values()[action % aLen];
        action /= aLen;
      }
      double avgInfo = getAvgInfo(searchDepth, worldBayesian, coins); // NOTE: Temporary.
      /* If better than our current predicted information gain, store the state */
      if(worldEntropy - avgInfo > bestInfoGain){
        bestInfoGain = worldEntropy - avgInfo;
        bestCoins = coins;
      }
    }
    return bestCoins;
  }

  /**
   * getAvgInfo()
   *
   * Calculates the average information gained from a move by searching the
   * tree recursively to a given depth.
   *
   * @param currentDepth The current depth of the recursion.
   * @param currentWorldBay The current world Bayesian given the recursive
   * depth.
   * @param coins The proposed arrangement of the coins to be tested.
   * @return Average information gained from action at a given depth.
   **/
  private double getAvgInfo(int currentDepth, double[] currentWorldBay, ACTION_STATE[] coins
      ){
    /* Iterate through different sensor options for proposed sensor states */
    double avgInfo = 0;
    int avgCnt = 0;
```

```
    for(SENSE_STATE ss : SENSE_STATE.values()){
      /* Calculate Bayesian model based on new information */
      double[] bay = updateBayesian(currentWorldBay, coins, ss);
      /* Check whether the predicted state is impossible */
      double baySum = 0;
      for(int x = 0; x < bay.length; x++){
        baySum += bay[x];
      }
      if(baySum > 0){
        if(currentDepth > 1){
          /* All possible action states */
          int wLen = WORLD_STATE.values()[0].getWorldState().length;
          int aLen = ACTION_STATE.values().length;
          int aCombo = (int)(Math.pow(aLen, wLen));
          double avgLayerInfo = 0;
          for(int z = 0; z < aCombo; z++){
            ACTION_STATE[] nCoins = new ACTION_STATE[wLen];
            int action = z;
            /* Figure out where each coin would be in each state */
            for(int y = 0; y < wLen; y++){
              nCoins[y] = ACTION_STATE.values()[action % aLen];
              action /= aLen;
            }
            avgInfo += getAvgInfo(currentDepth - 1, bay, nCoins);
            /* Add to average information count */
            avgCnt++;
          }
        }else{
          /* Calculate the information */
          avgInfo += getEntropy(bay);
          /* Add to average information count */
          avgCnt++;
        }
      }
    }
    /* Average the information gained */
    if(avgCnt > 0){
      /* Calculate the average information for gained in each state */
      avgInfo /= avgCnt;
    }else{
      /* TODO: Possibly troubling case where no states were possible and we are
               forced to take an average due to recursion. */
      Main.error("Unhandled entropy case hit.");
    }
    /* Return the current average information as we have reached the end */
    return avgInfo;
  }

  /**
   * updateBayesian()
   *
   * Calculate the new values of the Bayesian model based on the current
   * Bayesian model and the given new information.
   *
   * @param bay The old Bayesian model.
   * @param pos The position of the coins as we prepared them for the experiment.
   * @param ss The new sensed state.
   * @return The new Bayesian model.
   **/
  private static double[] updateBayesian(double[] bay, ACTION_STATE[] pos, SENSE_STATE ss){
    /* Make a copy of previous array with new data */
    double[] newBay = new double[bay.length];
    for(int x = 0; x < bay.length; x++){
      /* Store old value */
      newBay[x] = bay[x];
      /* Get the coins for the proposed world state */
      COIN[] coins = WORLD_STATE.values()[x].getWorldState();
      /* Calculate the sense state we would predict to see */
```

```
      SENSE_STATE predicted = getMeasurement(pos, coins);
      /* Check if that matches what we actually saw */
      if(predicted != ss){
        /* Can't be a correct hypothesis - wipe it */
        newBay[x] = 0;
      }
    }
    return normalise(newBay);
  }

  /**
   * getMeasurement()
   *
   * Gets the weighing of the coins.
   *
   * @param pCoins The abstract positions of the coins that we propose to measure.
   * @param aCoins The abstract positions of the coins that we actually measure.
   * @return The abstract bias to the scales.
   **/
  private static SENSE_STATE getMeasurement(ACTION_STATE[] pCoins, COIN[] aCoins){
    /* NOTE: Left is negative, right is positive. */
    double bias = 0;
    for(int x = 0; x < pCoins.length; x++){
      /* Find out the amount of weight to be added */
      double coinWeight = 0;
      switch(aCoins[x]){
        case NORM :
          coinWeight = 1.0;
          break;
        case LITE :
          coinWeight = 0.5;
          break;
        case HEVY :
          coinWeight = 1.5;
          break;
      }
      /* Assign the coin weight in the correct position */
      switch(pCoins[x]){
        case LEFT :
          bias -= coinWeight;
          break;
        case OFF :
          /* None, not weighed */
          break;
        case RIGHT :
          bias += coinWeight;
          break;
      }
    }
    /* Figure out what we sensed from the calculated bias */
    if(bias < 0){
      return SENSE_STATE.LEFT;
    }
    if(bias > 0){
      return SENSE_STATE.RIGHT;
    }
    return SENSE_STATE.MIDDLE;
  }

  /**
   * logData()
   *
   * Log the data to the terminal for later reading.
   *
   * @param values The values to be displayed.
   **/
  private static void logData(double[] values){
    /* Create a string array of the data we want to print */
```

```
    String[] data = new String[values.length];
    for(int x = 0; x < values.length; x++){
      data[x] = "" + values[x];
    }
    /* Output to the terminal */
    Main.log(data);
  }

  /**
   * getEntropy()
   *
   * Gets the entropy of a given set of probabilities.
   *
   * @param set The normalised set of probabilities to be checked.
   * @return The entropy of the given set.
   **/
  private static double getEntropy(double[] set){
    /* Sum all the entropy probabilities */
    double entropy = 0;
    for(int x = 0; x < set.length; x++){
      /* Make sure we're not calculating a zero case */
      if(set[x] > 0){
        /* Calculate entropy */
        entropy += set[x] * (Math.log(set[x]) / Math.log(2));
      }
    }
    return -entropy;
  }

  /**
   * normalise()
   *
   * Normalise a set of data.
   *
   * NOTE: This method is capable of normalising a zeroed set too.
   *
   * @param set The set to be normalised.
   * @return A copy of the normalised set.
   **/
  private static double[] normalise(double[] set){
    /* Create new set */
    double[] nSet = new double[set.length];
    /* Count value of existing data */
    double c = 0;
    for(int x = 0; x < set.length; x++){
      c += set[x];
    }
    /* Check whether C is zero */
    if(c != 0){
      /* Calculate normalisation factor */
      double n = 1 / c;
      /* Normalise the data using normalisation factor */
      for(int x = 0; x < set.length; x++){
        nSet[x] = set[x] * n;
      }
    }else{
      /* Impossible state, do nothing */
    }
    /* Return the normalised set */
    return nSet;
  }
}
```

10.1.7 PROBLEM_SOLVER.java

```
package barray.coin;

/**
 * PROBLEM_SOLVER.java
 *
 * The different types of solver for the problem.
 **/
public enum PROBLEM_SOLVER{
  RANDOM,
  INFO_GAIN_1_DPTH,
  INFO_GAIN_2_DPTH,
  INFO_GAIN_3_DPTH,
  INFO_GAIN_4_DPTH,
  INFO_GAIN_5_DPTH;
}
```

10.1.8 ACTION_STATE.java

```
package barray.coin;

/**
 * ACTION_STATE.java
 *
 * The state that can be actioned on a coin.
 **/
public enum ACTION_STATE{
  LEFT,
  OFF,
  RIGHT;
}
```

10.1.9 SENSE_STATE.java

```
package barray.coin;

/**
 * SENSE_STATE.java
 *
 * The sensed state of the balance scale.
 **/
public enum SENSE_STATE{
  LEFT,
  MIDDLE,
  RIGHT;
}
```

10.1.10 WORLD_STATE.java

```
package barray.coin;

/**
 * WORLD_STATE.java
 *
 * A collection of states the world could possibly be in when the problem is
 * complete.
 **/
public enum WORLD_STATE{
  W1(new COIN[]{COIN.LITE, COIN.NORM, COIN.NORM, COIN.NORM}),
  W2(new COIN[]{COIN.NORM, COIN.LITE, COIN.NORM, COIN.NORM}),
  W3(new COIN[]{COIN.NORM, COIN.NORM, COIN.LITE, COIN.NORM}),
  W4(new COIN[]{COIN.NORM, COIN.NORM, COIN.NORM, COIN.LITE}),
  W5(new COIN[]{COIN.HEVY, COIN.NORM, COIN.NORM, COIN.NORM}),
  W6(new COIN[]{COIN.NORM, COIN.HEVY, COIN.NORM, COIN.NORM}),
  W7(new COIN[]{COIN.NORM, COIN.NORM, COIN.HEVY, COIN.NORM}),
  W8(new COIN[]{COIN.NORM, COIN.NORM, COIN.NORM, COIN.HEVY});

  private final COIN[] coins;

  /**
   * WORLD_STATE()
   *
   * Initialise the world state.
   **/
  WORLD_STATE(COIN[] coins){
    this.coins = coins;
  }

  /**
   * getWorldState()
   *
   * A goal state of the world.
   *
   * @return The goal state of the world.
   **/
  COIN[] getWorldState(){
    return coins;
  }
}
```

10.1.11 COIN.java

```
package barray.coin;

/**
 * COIN.java
 *
 * The property to be associated with a coin.
 **/
public enum COIN{
  NORM,
  LITE,
  HEVY;
}
```

10.2 Camera Problem Source Code

In this section is the source code for the camera problem. Please contact `danbarry16@googlemail.com` for read access to:

- `https://bitbucket.org/danielbarry/final-year-project`
- `git@bitbucket.org:danielbarry/final-year-project.git`

10.2.1 run.sh

```
#!/bin/bash

function test {
  cd dist
    result=$(java -jar cam.jar $1 100 $2)
    readarray -t y <<<"$result"
    for i in "${y[@]}"
    do
      echo -e $i | grep "##"
    done
  cd ..
}

function main {
  for x in {0..1000}
  do
    test $1 $2
  done
}

main $1 $2
```

10.2.2 build.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="coin" default="jar" basedir=".">
  <description>An experiment to test for fake coins.</description>

  <property name="source.dir" location="src/" />
  <property name="build.dir" location="bin/" />
  <property name="jar.dir" location="dist/" />
  <property name="doc.dir" location="doc/" />
  <property name="main.class" value="barray.cam.Main" />
  <property name="main.jar" value="cam.jar" />

  <target name="clean" description="Clean Binaries">
    <delete dir="${build.dir}" />
    <delete dir="${jar.dir}" />
  </target>

  <target name="doc" description="Create Documentation">
    <delete dir="${doc.dir}" />
    <mkdir dir="${doc.dir}" />
    <javadoc destdir="${doc.dir}">
      <fileset dir="${source.dir}" />
    </javadoc>
  </target>

  <target name="compile" depends="clean" description="Compile Java Code">
    <mkdir dir="${build.dir}" />
    <javac srcdir="${source.dir}" destdir="${build.dir}" debug="true" includeantruntime="
        false" />
  </target>

  <target name="jar" depends="compile" description="Create Jar File">
    <mkdir dir="${jar.dir}" />
    <jar destfile="${jar.dir}/${main.jar}" basedir="${build.dir}">
      <manifest>
        <attribute name="Main-Class" value="${main.class}" />
      </manifest>
    </jar>
  </target>

  <target name="run" depends="jar" description="Run Java App">
    <java jar="${jar.dir}/${main.jar}" fork="true" />
  </target>
</project>
```

10.2.3 Main.java

```
package barray.cam;

/**
 * Main.java
 *
 * The main entry point into the program, handling the command line arguments
 * for the program. The job of this class is to find out what the user wants
 * and prepare execution of the main program if required.
 **/
public class Main{
  public static Interface out;

  /**
   * main()
   *
   * The main entry point of the program, breaking down the command line
   * arguments before processing them in other methods.
   *
   * @param args The arguments supplied the main program.
   **/
  public static void main(String[] args){
    /* Set the default interface to the terminal */
    /* TODO: Switch to GUI for demo. */
    out = new GUI();
    /* Display message about the expected input of program */
    Main.out.debug(
      "\ncam.jar <MODE> <LIMIT> <ENVIRONMENT>" +
      "\n" +
      "\n  MODE" +
      "\n    0   Random" +
      "\n    >0  Info Gain Depth" +
      "\n" +
      "\n  LIMIT" +
      "\n    Number of steps before bailout" +
      "\n" +
      "\n  ENVIRONMENT" +
      "\n    0   Basic (with limits)" +
      "\n    1   Loop (possibility of no limits)"
    );
    /* Program started */
    out.debug("Program started");
    /* Construct the environment */
    int limit = Integer.parseInt(args[1]);
    Environment env = null;
    if(Integer.parseInt(args[2]) == 0){
      env = new BasicEnv(8, limit);
    }else{
      env = new LoopEnv(8, limit);
    }
    /* Construct the agent */
    Agent agent = null;
    int mode = Integer.parseInt(args[0]);
    if(mode <= 0){
      agent = new RandomAgent(env, env.getWorldStateCount());
    }else{
      agent = new BasicAgent(env, env.getWorldStateCount(), mode);
    }
    /* Add the agent to the environment */
    env.addAgent(agent);
    /* Run the experiment */
    env.run();
    /* Program ended */
    out.debug("Program ended");
  }
}
```

10.2.4 Environment.java

```
package barray.cam;

/**
 * Environment.java
 *
 * A standard interface for the environment that the agent has to explore and
 * learn.
 **/
public interface Environment{
  /**
   * SENSE_STATE
   *
   * All possible states an agent can sense.
   **/
  public enum SENSE_STATE{
    LEFT,
    RIGHT,
    NONE
  }

  /**
   * ACT_STATE
   *
   * All possible states an agent can act.
   **/
  public enum ACT_STATE{
    LEFT,
    RIGHT
  }

  /**
   * addAgent()
   *
   * Add the agent to the environment.
   *
   * @param a The agent to be added to the environment.
   **/
  public void addAgent(Agent a);

  /**
   * run()
   *
   * The main simulation loop to be run once the environment and agent have
   * been setup correctly.
   **/
  public void run();

  /**
   * agentSense()
   *
   * Allow the agent to sense the environment.
   *
   * @param i The current position of the agent.
   * @return Proposed position of the agent.
   **/
  public SENSE_STATE agentSense(int i);

  /**
   * agentAct()
   *
   * Allows the agent to act in the environment.
   *
   * @param i The current position of the agent.
   * @param a The current action state.
   * @return The calculated position of the agent.
   **/
```

```
    public int agentAct(int i, ACT_STATE a);

    /**
     * getWorldStateCount()
     *
     * Gets the number of world states possible.
     *
     * @return The number of possible world states.
     **/
    public int getWorldStateCount();

    /**
     * getWorldStates()
     *
     * Gets the world states.
     *
     * NOTE: These must not be changed.
     *
     * @return The world states.
     **/
    public SENSE_STATE[][] getWorldStates();

    /**
     * getWorldSize()
     *
     * Gets the size of the world.
     *
     * @return The size of the world.
     **/
    public int getWorldSize();

    /**
     * getAgentPosition()
     *
     * Gets the current position of the agent.
     **/
    public int getAgentPosition();

    /**
     * getLimitA()
     *
     * Gets the limit of the environment.
     **/
    public int getLimitA();

    /**
     * getLimitB()
     *
     * Gets the limit of the environment.
     **/
    public int getLimitB();
}
```

10.2.5 BasicEnv.java

```
package barray.cam;

import java.util.Random;

/**
 * BasicEnv.java
 *
 * A simple environment setup to test the basic system.
 **/
public class BasicEnv implements Environment{
  private int n;
  private int l;
  private int limA;
  private int limB;
  private int aPos;
  private static SENSE_STATE[][] worldStates;
  private SENSE_STATE[] solution;
  private Agent agent;

  /**
   * BasicEnv()
   *
   * The basic environment initialisation.
   *
   * @param num The number of possible state.
   * @param limit The limit to the number of loops till solution is found.
   **/
  public BasicEnv(int num, int limit){
    n = num;
    l = limit;
    Random rand = new Random();
    aPos = rand.nextInt(n);
    /* Generate world state */
    int l =  n * (n - 1);
    worldStates = new SENSE_STATE[l][];
    for(int y = 0; y < n - 1; y++){ /* Right position */
      for(int x = 0; x < n; x++){ /* Left position */
        int i = (y * n) + x;
        worldStates[i] = new SENSE_STATE[n];
        /* Wipe current sense states */
        for(int z = 0; z < worldStates[i].length; z++){
          worldStates[i][z] = SENSE_STATE.NONE;
        }
        worldStates[i][x] = SENSE_STATE.LEFT;
        worldStates[i][y < x ? y : y + 1] = SENSE_STATE.RIGHT;
      }
    }
    /* Generate solution */
    solution = worldStates[rand.nextInt(worldStates.length)];
    /* Get the limits */
    for(int x = 0; x < solution.length; x++){
      if(solution[x] == SENSE_STATE.LEFT){
        limA = x;
      }
      if(solution[x] == SENSE_STATE.RIGHT){
        limB = x;
      }
    }
    /* Debug what has been created */
    String buff = "Solution: ";
    for(int x = 0; x < solution.length; x++){
      buff += "[" + solution[x] + "],";
    }
    Main.out.debug(buff);
  }
```

```
public void addAgent(Agent a){
  agent = a;
}

public void run(){
  /* Display the current state of the agent */
  Main.out.displayCurrentState(this, agent);
  /* Loop until a solution is found to the problem */
  boolean solved = false;
  int loops = 0;
  while(!solved && loops < 1){
    loops++;
    /* Pause here until user says to continue */
    Main.out.getKeyPress();
    /* Allow the agent to sense */
    agent.sense(agentSense(aPos));
    /* Allow the agent to act */
    ACT_STATE a = agent.act();
    aPos = agentAct(aPos, a);
    /* Display the current state of the agent */
    Main.out.displayCurrentState(this, agent);
    /* Check whether the problem is complete */
    double[] tW = agent.getSimpleW();
    int oneCnt = 0;
    for(double pos : tW){
      if(pos == 1){
        oneCnt++;
      }
    }
    if(oneCnt == 2){
      solved = true;
    }
  }
  /* Output number of loops */
  Main.out.debug("##," + loops);
}

public SENSE_STATE agentSense(int i){
  if(i == limA){
    return SENSE_STATE.LEFT;
  }else if(i == limB){
    return SENSE_STATE.RIGHT;
  }else{
    return SENSE_STATE.NONE;
  }
}

public int agentAct(int i, ACT_STATE a){
  int tPos = i;
  if(a == ACT_STATE.LEFT){
    if(tPos != limA){
      --tPos;
      if(tPos < 0){
        tPos += n;
      }
    }
  }else{
    if(tPos != limB){
      ++tPos;
      if(tPos >= n){
        tPos -= n;
      }
    }
  }
  return tPos;
}

public int getWorldStateCount(){
```

```
    return worldStates.length;
  }

  public SENSE_STATE[][] getWorldStates(){
    return worldStates;
  }

  public int getWorldSize(){
    return n;
  }

  public int getAgentPosition(){
    return aPos;
  }

  public int getLimitA(){
    return limA;
  }

  public int getLimitB(){
    return limB;
  }
}
```

10.2.6 LoopEnv.java

```
package barray.cam;

import java.util.Random;

/**
 * LoopEnv.java
 *
 * A simple environment setup to test the basic system.
 **/
public class LoopEnv implements Environment{
  private int n;
  private int l;
  private int limA;
  private int limB;
  private int aPos;
  private static SENSE_STATE[][] worldStates;
  private SENSE_STATE[] solution;
  private Agent agent;

  /**
   * LoopEnv()
   *
   * The basic environment initialisation.
   *
   * @param num The number of possible state.
   * @param limit The limit to the number of loops till solution is found.
   **/
  public LoopEnv(int num, int limit){
    n = num;
    l = limit;
    Random rand = new Random();
    aPos = rand.nextInt(n);
    /* Generate world state */
    int l =  n * (n - 1);
    /* NOTE: Add additional space for there being no limits */
    worldStates = new SENSE_STATE[l + 1][];
    for(int y = 0; y < n - 1; y++){ /* Right position */
      for(int x = 0; x < n; x++){ /* Left position */
        int i = (y * n) + x;
        worldStates[i] = new SENSE_STATE[n];
        /* Wipe current sense states */
        for(int z = 0; z < worldStates[i].length; z++){
          worldStates[i][z] = SENSE_STATE.NONE;
        }
        worldStates[i][x] = SENSE_STATE.LEFT;
        worldStates[i][y < x ? y : y + 1] = SENSE_STATE.RIGHT;
      }
    }
    /* Fill out the blank state */
    worldStates[l] = new SENSE_STATE[n];
    for(int x = 0; x < n; x++){
      worldStates[l][x] = SENSE_STATE.NONE;
    }
    /* Generate solution */
    //solution = worldStates[rand.nextInt(worldStates.length)];
    solution = worldStates[worldStates.length - 1];
    /* Get the limits if they exist */
    /* NOTE: Set the limits to where the agent cannot exist. */
    limA = -1;
    limB = -1;
    for(int x = 0; x < solution.length; x++){
      if(solution[x] == SENSE_STATE.LEFT){
        limA = x;
      }
      if(solution[x] == SENSE_STATE.RIGHT){
        limB = x;
```

```
      }
    }
    /* Debug what has been created */
    String buff = "Solution: ";
    for(int x = 0; x < solution.length; x++){
      buff += "[" + solution[x] + "],";
    }
    Main.out.debug(buff);
  }

  public void addAgent(Agent a){
    agent = a;
  }

  public void run(){
    /* Display the current state of the agent */
    Main.out.displayCurrentState(this, agent);
    /* Loop until a solution is found to the problem */
    boolean solved = false;
    int loops = 0;
    while(!solved && loops < l){
      loops++;
      /* Pause here until user says to continue */
      Main.out.getKeyPress();
      /* Allow the agent to sense */
      agent.sense(agentSense(aPos));
      /* Allow the agent to act */
      ACT_STATE a = agent.act();
      aPos = agentAct(aPos, a);
      /* Display the current state of the agent */
      Main.out.displayCurrentState(this, agent);
      /* Check whether the problem is complete */
      double[] tW = agent.getW();
      for(double pos : tW){
        if(pos == 1){
          solved = true;
        }
      }
    }
    /* Output number of loops */
    Main.out.debug("##," + loops);
  }

  public SENSE_STATE agentSense(int i){
    if(i == limA){
      return SENSE_STATE.LEFT;
    }else if(i == limB){
      return SENSE_STATE.RIGHT;
    }else{
      return SENSE_STATE.NONE;
    }
  }

  public int agentAct(int i, ACT_STATE a){
    int tPos = i;
    if(a == ACT_STATE.LEFT){
      if(tPos != limA){
        --tPos;
        if(tPos < 0){
          tPos += n;
        }
      }
    }else{
      if(tPos != limB){
        ++tPos;
        if(tPos >= n){
          tPos -= n;
        }
```

```
      }
    }
    return tPos;
  }

  public int getWorldStateCount(){
    return worldStates.length;
  }

  public SENSE_STATE[][] getWorldStates(){
    return worldStates;
  }

  public int getWorldSize(){
    return n;
  }

  public int getAgentPosition(){
    return aPos;
  }

  public int getLimitA(){
    return limA;
  }

  public int getLimitB(){
    return limB;
  }
}
```

10.2.7 Agent.java

```
package barray.cam;

/**
 * Agent.java
 *
 * A standard interface for an agent set to explore the environment.
 **/
public interface Agent{
  /**
   * sense()
   *
   * Called to allow the agent to sense the current world.
   *
   * @param sense The sensed stated of the world.
   **/
  public void sense(Environment.SENSE_STATE sense);

  /**
   * act()
   *
   * Allows the agent to make an action on itself in the world.
   *
   * @return The action to be taken on behalf of the agent.
   **/
  public Environment.ACT_STATE act();

  /**
   * getSimpleW()
   *
   * Gets the simplified world state, giving the probability of a limit
   * existing in a position.
   *
   * @return The simplified current world state.
   **/
  public double[] getSimpleW();

  /**
   * getW()
   *
   * Returns the full uncompressed internal Bayesian model.
   *
   * @return The internal Bayesian model.
   **/
  public double[] getW();

  /**
   * toString()
   *
   * Converts this class to a string.
   *
   * @return A string representation of this class.
   **/
  public String toString();
}
```

10.2.8 BasicAgent.java

```
package barray.cam;

/**
 * BasicAgent.java
 *
 * A basic implementation of an agent who's job is to explore a given
 * environment.
 **/
public class BasicAgent implements Agent{
  private Environment e;
  private int n;
  private double[] W;
  private int d;

  /**
   * BasicAgent()
   *
   * The initialiser for a basic agent.
   *
   * @param env The environment the agent is in.
   * @param num The number of possible state.
   * @param depth The depth to search the problem.
   **/
  public BasicAgent(Environment env, int num, int depth){
    /* Initialise the variables */
    e = env;
    n = num;
    d = depth;
    W = new double[n];
    /* Setup and normalise the data */
    for(int x = 0; x < n; x++){
      W[x] = 1;
    }
    W = InfoUtil.normalise(W);
  }

  public void sense(Environment.SENSE_STATE sense){
    W = InfoUtil.naiveBayesUpdate(W, sense, e.getAgentPosition(), e.getWorldStates());
  }

  public Environment.ACT_STATE act(){
    Environment.ACT_STATE besta = null;
    double bestI = -1;
    /* Iterate over possible actions */
    for(Environment.ACT_STATE a : Environment.ACT_STATE.values()){
      /* Iterate over possible senses */
      double avgI = 0;
      double avgC = 0;
      for(Environment.SENSE_STATE s: Environment.SENSE_STATE.values()){
        int newPos = e.agentAct(e.getAgentPosition(), a);
        double[] tW = InfoUtil.naiveBayesUpdate(W, s, newPos, e.getWorldStates());
        if(InfoUtil.sumArray(tW) > 0){
          avgI += calculateBestAvgInfo(d, tW, newPos);
          avgC++;
        }else{
          /* Do nothing for impossible scenario */
        }
      }
      avgI /= avgC;
      double avgIGain = InfoUtil.H(W) - avgI;
      /* Check to see if it beats the current best */
      if(avgIGain >= bestI){
        bestI = avgIGain;
        besta = a;
      }
    }
```

```
    /* If we hit a NULL case, record it */
    if(besta == null){
      Main.out.debug("ERROR: 'besta' should not be NULL.");
    }
    /* Select outcome */
    return besta;
  }

  /**
   * calculateAvgInfo()
   *
   * Calculates the best average info given the information relevant to it's
   * search depth.
   *
   * @param dpth The current depth of the search problem.
   * @param w The world model to do the calculations from.
   * @param i The position of the agent.
   * @return The average information for that problem.
   **/
  private double calculateBestAvgInfo(int dpth, double[] w, int i){
    /* Use the information found here, or go down another layer if not at the end */
    if(dpth > 1){
      /* Iterate over possible senses */
      double avgI = 0;
      double avgC = 0;
      /* Iterate over possible actions */
      for(Environment.ACT_STATE a : Environment.ACT_STATE.values()){
        for(Environment.SENSE_STATE s: Environment.SENSE_STATE.values()){
          int newPos = e.agentAct(i, a);
          double[] tW = InfoUtil.naiveBayesUpdate(w, s, newPos, e.getWorldStates());
          if(InfoUtil.sumArray(tW) > 0){
            avgI += calculateBestAvgInfo(dpth - 1, tW, newPos);
            avgC++;
          }else{
            /* Do nothing for impossible scenario */
          }
        }
      }
      /* Select outcome */
      return avgI / avgC;
    }else{
      return InfoUtil.H(w);
    }
  }

  public double[] getSimpleW(){
    double[] simpleW  = new double[e.getWorldSize()];
    for(int i = 0; i < e.getWorldSize(); i++){ /* World positions */
      simpleW[i] = 0;
      for(int x = 0; x < n; x++){ /* World states */
        if(e.getWorldStates()[x][i] == Environment.SENSE_STATE.LEFT ||
           e.getWorldStates()[x][i] == Environment.SENSE_STATE.RIGHT){
          simpleW[i] += W[x];
        }
      }
    }
    /* NOTE: We know this needs normalising as we are compacting (n*(n-1))
             states into n positions. */
    simpleW = InfoUtil.normalise(simpleW);
    /* Double each number because we have two limits */
    for(int x = 0; x < simpleW.length; x++){
      simpleW[x] *= 2;
    }
    return simpleW;
  }

  public double[] getW(){
    return W;
```

```
  }

  public String toString(){
    String r = "{";
    for(int x = 0; x < W.length; x++){
      r += W[x];
      if(x + 1 < W.length){
        r += ",";
      }
    }
    return r + "}";
  }
}
```

10.2.9 InfoUtil.java

```
package barray.cam;

/**
 * InfoUtil.java
 *
 * Information Theory utilities to be used for calculations.
 **/
public abstract class InfoUtil{
  /**
   * H()
   *
   * Gets the entropy of a given set of probabilities.
   *
   * @param set The normalised set of probabilities to be checked.
   * @return The entropy of the given set.
   **/
  public static double H(double[] W){
    /* Sum all the entropy probabilities */
    double entropy = 0;
    for(int x = 0; x < W.length; x++){
      double p = W[x];
      /* Make sure we're not calculating a zero case */
      if(p > 0){
        /* Calculate entropy SUM(p*log_2(p)) */
        /* NOTE: Calculation done in base e due to a Java bug. */
        entropy += p * (Math.log(p) / Math.log(2));
      }
    }
    return -entropy;
  }

  /**
   * naiveBayesUpdate()
   *
   * A naive Bayesian update on the world state given a new piece of
   * information.
   *
   * @param w An array of probabilities for the world states.
   * @param s The sense state.
   * @param i The position the update is to be performed to.
   * @param S All possible states to be considered in the Bayes model.
   * @return The new probabilities for world states.
   **/
  public static double[] naiveBayesUpdate(double[] w, Environment.SENSE_STATE s, int i,
      Environment.SENSE_STATE[][] S){
    double[] W = clone(w);
    /* The probabilities for it being left, right and none for i */
    double pL = s == Environment.SENSE_STATE.LEFT ? 1 : 0;
    double pR = s == Environment.SENSE_STATE.RIGHT ? 1 : 0;
    double pN = s == Environment.SENSE_STATE.NONE ? 1 : 0;
    /* Iterate over the Bayes model and find out how that effects it */
    for(int x = 0; x < W.length; x++){
      switch(S[x][i]){
        case LEFT :
          W[x] *= pL;
          break;
        case RIGHT :
          W[x] *= pR;
          break;
        case NONE :
          W[x] *= pN;
          break;
      }
    }
    return InfoUtil.normalise(W);
  }
```

```
  /**
   * clone()
   *
   * Clones an array of double values.
   *
   * @param a The array to be cloned.
   * @return A clone on the array, a.
   **/
  public static double[] clone(double[] a){
    double[] b = new double[a.length];
    for(int x = 0; x < a.length; x++){
      b[x] = a[x];
    }
    return b;
  }

  /**
   * normalise()
   *
   * Normalise a set of data.
   *
   * NOTE: This method is capable of normalising a zeroed set too.
   *
   * @param set The set to be normalised.
   * @return A copy of the normalised set.
   **/
  public static double[] normalise(double[] set){
    /* Create new set */
    double[] nSet = new double[set.length];
    /* Count value of existing data */
    double c = sumArray(set);
    /* Check whether C is zero */
    if(c > 0){
      /* Calculate normalisation factor */
      double k = 1 / c;
      /* Normalise the data using normalisation factor */
      for(int x = 0; x < set.length; x++){
        nSet[x] = set[x] * k;
      }
    }else{
      /* Impossible state, do nothing */
    }
    /* Return the normalised set */
    return nSet;
  }

  /**
   * sumArray()
   *
   * Sums the array and returns the result.
   *
   * @param set The data set to sum.
   * @return The sum of the array.
   **/
  public static double sumArray(double[] set){
    double val = 0;
    for(int x = 0; x < set.length; x++){
      val += set[x];
    }
    return val;
  }
}
```

10.2.10 Interface.java

```
package barray.cam;

/**
 * Interface.java
 *
 * A standard interface class for the human interface, implemented by both the
 * Terminal and GUI classes.
 **/
public interface Interface{
  /**
   * displayCurrentState()
   *
   * Displays the world problem given an environment and agent.
   *
   * @param env The environment.
   * @param agent The agent.
   **/
  public void displayCurrentState(Environment env, Agent agent);

  /**
   * debug()
   *
   * Allows a string to debugged.
   *
   * @param msg The message to be displayed.
   **/
  public void debug(String msg);

  /**
   * getKeyPress()
   *
   * Get the user to press any key in order to continue.
   **/
  public void getKeyPress();
}
```

10.2.11 Terminal.java

```
package barray.cam;

import java.util.Scanner;

/**
 * Terminal.java
 *
 * A basic text representation of what is happening.
 **/
public class Terminal implements Interface{
  public void displayCurrentState(Environment env, Agent agent){
    System.out.println(agent.toString());
  }

  public void debug(String msg){
    System.out.println(msg);
  }

  public void getKeyPress(){
    Scanner scan = new Scanner(System.in);
    debug("Press enter to continue.");
    scan.nextLine();
  }
}
```

10.2.12 GUI.java

```
package barray.cam;

import java.awt.Container;
import java.awt.Color;
import java.awt.Graphics;
import javax.swing.JFrame;

/**
 * GUI.java
 *
 * A basic graphical user interface implementation class that represents the
 * problem.
 **/
public class GUI extends JFrame implements Interface{
  private static final int WIN_SIZE = 600;

  private Terminal term;
  private double[][] lines;
  private String[] labels;
  private double[] aPos;
  private double[] limA;
  private double[] limB;

  /**
   * GUI()
   *
   * Creates a new instance of the GUI class and initialises the window to be
   * used.
   **/
  public GUI(){
    term = new Terminal();
    lines = new double[0][];
    labels = new String[]{};
    aPos = new double[2];
    limA = new double[2];
    limB = new double[2];
    setSize(WIN_SIZE, WIN_SIZE);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setContentPane(
      new Container(){
        @Override
        public void paint(final Graphics g){
          super.paint(g);
          /* Calculate dimensions */
          int a = getWidth() > getHeight() ? getHeight() : getWidth();
          int al = (int)(a / 4);
          int ak = (int)(a * 0.625) - al;
          int ah = (int)(a * 0.75) - al;
          int m = a / 2;
          /* Draw outer circle */
          g.setColor(Color.YELLOW);
          g.fillOval(0, 0, a, a);
          g.setColor(Color.BLACK);
          g.drawOval(0, 0, a, a);
          /* Draw inner circle */
          g.setColor(Color.WHITE);
          g.fillOval(al, al, ah, ah);
          g.setColor(Color.BLACK);
          g.drawOval(al, al, ah, ah);
          /* Draw lines */
          synchronized(lines){
            for(int x = 0; x < lines.length; x++){
              g.drawLine(
                (int)(m + (al * lines[x][0])),
                (int)(m + (al * lines[x][1])),
                (int)(m + (ah * lines[x][2])),
```

```
              (int)(m + (ah * lines[x][3]))
            );
            g.drawString(
              labels[x],
              (int)(
                m + (ak * lines[x][4]) -
                /* Source: http://bit.ly/2bU1Imi */
                (g.getFontMetrics().stringWidth(labels[x]) / 2)
              ),
              (int)(m + (ak * lines[x][5]))
            );
          }
        }
        g.setColor(Color.RED);
        g.drawLine(
          (int)(m),
          (int)(m),
          (int)(m + (al * aPos[0])),
          (int)(m + (al * aPos[1]))
        );
        if(limA != null){
          g.setColor(Color.BLUE);
          g.drawLine(
            (int)(m),
            (int)(m),
            (int)(m + (al * limA[0])),
            (int)(m + (al * limA[1]))
          );
        }
        if(limB != null){
          g.setColor(Color.GREEN);
          g.drawLine(
            (int)(m),
            (int)(m),
            (int)(m + (al * limB[0])),
            (int)(m + (al * limB[1]))
          );
        }
      }
    }
  );
  setVisible(true);
}

public void displayCurrentState(Environment env, Agent agent){
  double[] w = agent.getSimpleW();
  double arc = (Math.PI * 2) / w.length;
  synchronized(lines){
    lines = new double[w.length][];
    labels = new String[w.length];
    for(int x = 0; x < w.length; x++){
      /* NOTE: The last two points are for text position. */
      lines[x] = new double[6];
      lines[x][0] = Math.sin(arc * x);
      lines[x][1] = -Math.cos(arc * x);
      lines[x][2] = Math.sin(arc * x);
      lines[x][3] = -Math.cos(arc * x);
      lines[x][4] = Math.sin(arc * (x + 0.5));
      lines[x][5] = -Math.cos(arc * (x + 0.5));
      /* Make label */
      labels[x] = "" + w[x];
    }
  }
  /* Calculate the current position of the agent to draw */
  aPos[0] = Math.sin(arc * (env.getAgentPosition() + 0.5));
  aPos[1] = -Math.cos(arc * (env.getAgentPosition() + 0.5));
  /* Calculate the limits of the environment if possible */
  int lA = env.getLimitA();
```

```
    int lB = env.getLimitB();
    if(lA >= 0){
      limA[0] = Math.sin(arc * (lA + 0.25));
      limA[1] = -Math.cos(arc * (lA + 0.25));
    }else{
      limA = null;
    }
    if(lB >= 0){
      limB[0] = Math.sin(arc * (lB + 0.75));
      limB[1] = -Math.cos(arc * (lB + 0.75));
    }else{
      limB = null;
    }
    /* Make sure our update is draw */
    repaint();
  }

  public void debug(String msg){
    term.debug(msg);
  }

  public void getKeyPress(){
    term.getKeyPress();
  }
}
```

10.3 Coin Problem Raw Results

Random	First	Second	Third
3	3	3	3
3	3	3	3
3	3	3	3
3	3	3	3
3	3	3	3
3	3	3	3
3	3	3	3
3	3	3	3
3	3	3	3
3	3	3	3
3	3	3	3
4	3	3	3
4	3	3	3
4	3	3	3
4	3	3	3
4	3	3	3
4	3	3	3
4	3	3	3
4	3	3	3
4	3	3	3
4	3	3	3
4	3	3	3
4	3	3	3
4	3	3	3
4	3	3	3
4	3	3	3
4	3	3	3
4	3	3	3
4	3	3	3
4	3	3	3
4	3	3	3
4	3	3	3
4	3	3	3
4	3	3	3
4	3	3	3
4	3	3	3
4	3	3	3
4	3	3	3
4	3	3	3
4	3	3	3
4	3	3	3
4	3	3	3
4	3	3	3
4	3	3	3
4	3	3	3
5	3	3	3
5	3	3	3
5	3	3	3
5	3	3	3
5	3	3	3
5	3	3	3
5	3	3	3
5	3	3	3
5	3	3	3
5	3	3	3
5	3	3	3
5	3	3	3
5	3	3	3
5	3	3	3
5	3	3	3
5	3	3	3
5	3	3	3
5	3	3	3
5	3	3	3
5	3	3	3

5	3	3	3
5	3	3	3
5	3	3	3
5	3	3	3
5	3	3	3
5	3	3	3
5	3	3	3
5	3	3	3
5	3	3	3
5	3	3	3
6	3	3	3
6	3	3	3
6	3	3	3
6	3	3	3
6	3	3	3
6	3	3	3
6	3	3	3
6	3	3	3
6	3	3	3
6	3	3	3
6	3	3	3
6	3	3	3
6	3	3	3
6	3	3	3
6	3	3	3
6	3	3	3
6	3	3	3
6	3	3	3
6	3	3	3
6	3	3	3
6	3	3	3
6	3	3	3
6	3	3	3
6	3	3	3
6	3	3	3
6	3	3	3
6	3	3	3
6	3	3	3
6	3	3	3
6	3	3	3
6	3	3	3
6	3	3	3
6	3	3	3
6	3	3	3
6	3	3	3
6	3	3	3
6	3	3	3
6	3	3	3
6	3	3	3
6	3	3	3
6	3	3	3
6	3	3	3
6	3	3	3
6	3	3	3
6	3	3	3
7	3	3	3
7	3	3	3
7	3	3	3
7	3	3	3
7	3	3	3
7	3	3	3
7	3	3	3
7	3	3	3
7	3	3	3
7	3	3	3
7	3	3	3
7	3	3	3
7	3	3	3

7	3	3	3
7	3	3	3
7	3	3	3
7	3	3	3
7	3	3	3
7	3	3	3
7	3	3	3
7	3	3	3
7	3	3	3
7	3	3	3
7	3	3	3
7	3	3	3
7	3	3	3
7	3	3	3
7	3	3	3
7	3	3	3
7	3	3	3
7	3	3	3
7	3	3	3
7	3	3	3
7	3	3	3
7	3	3	3
7	3	3	3
7	3	3	3
7	3	3	3
7	3	3	3
7	3	3	3
7	3	3	3
7	3	3	3
7	3	3	3
7	3	3	3
7	3	3	3
7	3	3	3
7	3	3	3
8	3	3	3
8	3	3	3
8	3	3	3
8	3	3	3
8	3	3	3
8	3	3	3
8	3	3	3
8	3	3	3
8	3	3	3
8	3	3	3
8	3	3	3
8	3	3	3
8	3	3	3
8	3	3	3
8	3	3	3
8	3	3	3
8	3	3	3
8	3	3	3
8	3	3	3
8	3	3	3
8	3	3	3
8	3	3	3
8	3	3	3
8	3	3	3
8	3	3	3
8	3	3	3
8	3	3	3
8	3	3	3
8	3	3	3
8	3	3	3
8	3	3	3
8	3	3	3
8	3	3	3
8	3	3	3

8	3	3	3
8	3	3	3
8	3	3	3
8	3	3	3
8	3	3	3
8	3	3	3
8	3	3	3
8	3	3	3
8	3	3	3
8	3	3	3
8	3	3	3
8	3	3	3
8	3	3	3
8	3	3	3
8	3	3	3
8	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3

9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
9	3	3	3
10	3	3	3
10	3	3	3
10	3	3	3
10	3	3	3
10	3	3	3
10	3	3	3
10	3	3	3
10	3	3	3
10	3	3	3
10	3	3	3
10	3	3	3
10	3	3	3
10	3	3	3
10	3	3	3
10	3	3	3
10	3	3	3
10	3	3	3
10	3	3	3
10	3	3	3
10	3	3	3
10	3	3	3
10	3	3	3
10	3	3	3
10	3	3	3
10	3	3	3
10	3	3	3
10	3	3	3
10	3	3	3
10	3	3	3
10	3	3	3
10	3	3	3
10	3	3	3
10	3	3	3
10	3	3	3
10	3	3	3
10	3	3	3
10	3	3	3
10	3	3	3
10	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3

11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
11	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3

12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
12	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3

13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
13	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3

14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
14	3	3	3
15	3	3	3
15	3	3	3
15	3	3	3
15	3	3	3
15	3	3	3
15	3	3	3
15	3	3	3
15	3	3	3
15	3	3	3
15	3	3	3
15	3	3	3
15	3	3	3
15	3	3	3
15	3	3	3
15	3	3	3
15	3	3	3
15	3	3	3
15	3	3	3
15	3	3	3
15	3	3	3
15	3	3	3
15	3	3	3
15	3	3	3
15	3	3	3
15	3	3	3
15	3	3	3
15	3	3	3
15	3	3	3
15	3	3	3
15	3	3	3
15	3	3	3
15	3	3	3
15	3	3	3
15	3	3	3
15	3	3	3
15	3	3	3
15	3	3	3
15	3	3	3
15	3	3	3
15	3	3	3
15	3	3	3
15	3	3	3
15	3	3	3
16	3	3	3
16	3	3	3
16	3	3	3
16	3	3	3
16	3	3	3
16	3	3	3
16	3	3	3
16	3	3	3
16	3	3	3

16	3	3	3
16	3	3	3
16	3	3	3
16	3	3	3
16	3	3	3
16	3	3	3
16	3	3	3
16	3	3	3
16	3	3	3
16	3	3	3
16	3	3	3
16	3	3	3
16	3	3	3
16	3	3	3
16	3	3	3
16	3	3	3
16	3	3	3
16	3	3	3
16	3	3	3
16	3	3	3
16	3	3	3
16	3	3	3
16	3	3	3
16	3	3	3
16	3	3	3
16	3	3	3
16	3	3	3
16	3	3	3
16	3	3	3
17	3	3	3
17	3	3	3
17	3	3	3
17	3	3	3
17	3	3	3
17	3	3	3
17	3	3	3
17	3	3	3
17	3	3	3
17	3	3	3
17	3	3	3
17	3	3	3
17	3	3	3
17	3	3	3
17	3	3	3
17	3	3	3
17	3	3	3
17	3	3	3
17	3	3	3
17	3	3	3
17	3	3	3
17	3	3	3
17	3	3	3
17	3	3	3
17	3	3	3
17	3	3	3
17	3	3	3
17	3	3	3
17	3	3	3
17	3	3	3
17	3	3	3
17	3	3	3
17	3	3	3
17	3	3	3
17	3	3	3
18	3	3	3
18	3	3	3
18	3	3	3

18	3	3	3
18	3	3	3
18	3	3	3
18	3	3	3
18	3	3	3
18	3	3	3
18	3	3	3
18	3	3	3
18	3	3	3
18	3	3	3
18	3	3	3
18	3	3	3
18	3	3	3
18	3	3	3
18	3	3	3
18	3	3	3
18	3	3	3
18	3	3	3
18	3	3	3
18	3	3	3
18	3	3	3
18	3	3	3
18	3	3	3
18	3	3	3
18	3	3	3
18	3	3	3
18	3	3	3
18	3	3	3
18	3	3	3
18	3	3	3
18	3	3	3
18	3	3	3
19	3	3	3
19	3	3	3
19	3	3	3
19	3	3	3
19	3	3	3
19	3	3	3
19	3	3	3
19	3	3	3
19	3	3	3
19	3	3	3
19	3	3	3
19	3	3	3
19	3	3	3
19	3	3	3
19	3	3	3
19	3	3	3
19	3	3	3
19	3	3	3
19	3	3	3
19	3	3	3
19	3	3	3
19	3	3	3
19	3	3	3
19	3	3	3
19	3	3	3
19	3	3	3
19	3	3	3
19	3	3	3
19	3	3	3
19	3	3	3
19	3	3	3
19	3	3	3
19	3	3	3
19	3	3	3
19	3	3	4
19	3	3	4

19	3	3	4
20	3	3	4
20	3	3	4
20	3	3	4
20	3	3	4
20	3	3	4
20	3	3	4
20	3	3	4
20	3	3	4
20	3	3	4
20	3	4	4
20	3	4	4
20	3	4	4
20	4	4	4
20	4	4	4
20	4	4	4
20	4	4	4
20	4	4	4
20	4	4	4
20	4	4	4
20	4	4	4
20	4	4	4
20	4	4	4
20	4	4	4
20	4	4	4
20	4	4	4
20	4	4	4
20	4	4	4
20	4	4	4
20	4	4	4
20	4	4	4
20	4	4	4
21	4	4	4
21	4	4	4
21	4	4	4
21	4	4	4
21	4	4	4
21	4	4	4
21	4	4	4
21	4	4	4
21	4	4	4
21	4	4	4
21	4	4	4
21	4	4	4
21	4	4	4
21	4	4	4
21	4	4	4
21	4	4	4
21	4	4	4
21	4	4	4
21	4	4	4
21	4	4	4
21	4	4	4
21	4	4	4
21	4	4	4
21	4	4	4
21	4	4	4
22	4	4	4
22	4	4	4
22	4	4	4
22	4	4	4
22	4	4	4
22	4	4	4
22	4	4	4
22	4	4	4
22	4	4	4
22	4	4	4
22	4	4	4

22	4	4	4
22	4	4	4
22	4	4	4
22	4	4	4
22	4	4	4
22	4	4	4
22	4	4	4
22	4	4	4
22	4	4	4
22	4	4	4
23	4	4	4
23	4	4	4
23	4	4	4
23	4	4	4
23	4	4	4
23	4	4	4
23	4	4	4
23	4	4	4
23	4	4	4
23	4	4	4
23	4	4	4
23	4	4	4
23	4	4	4
23	4	4	4
23	4	4	4
23	4	4	4
23	4	4	4
23	4	4	4
23	4	4	4
23	4	4	4
23	4	4	4
24	4	4	4
24	4	4	4
24	4	4	4
24	4	4	4
24	4	4	4
24	4	4	4
24	4	4	4
24	4	4	4
24	4	4	4
24	4	4	4
24	4	4	4
24	4	4	4
24	4	4	4
24	4	4	4
24	4	4	4
24	4	4	4
24	4	4	4
24	4	4	4
24	4	4	4
24	4	4	4
24	4	4	4
24	4	4	4
25	4	4	4
25	4	4	4
25	4	4	4
25	4	4	4
25	4	4	4
25	4	4	4
25	4	4	4
25	4	4	4
25	4	4	4
25	4	4	4
25	4	4	4
26	4	4	4
26	4	4	4
26	4	4	4
26	4	4	4

26	4	4	4
26	4	4	4
26	4	4	4
26	4	4	4
26	4	4	4
26	4	4	4
26	4	4	4
26	4	4	4
26	4	4	4
26	4	4	4
26	4	4	4
26	4	4	4
26	4	4	4
26	4	4	4
27	4	4	4
27	4	4	4
27	4	4	4
27	4	4	4
27	4	4	4
27	4	4	4
27	4	4	4
27	4	4	4
28	4	4	4
28	4	4	4
28	4	4	4
28	4	4	4
28	4	4	4
28	4	4	4
28	4	4	4
28	4	4	4
28	4	4	4
28	4	4	4
28	4	4	4
28	4	4	4
29	4	4	4
29	4	4	4
29	4	4	4
29	4	4	4
29	4	4	4
29	4	4	4
29	4	4	4
29	4	4	4
29	4	4	4
29	4	4	4
29	4	4	4
30	4	4	4
30	4	4	4
30	4	4	4
30	4	4	4
30	4	4	4
30	4	4	4
30	4	4	4
30	4	4	4
30	4	4	4
30	4	4	4
31	4	4	4
31	4	4	4
31	4	4	4
31	4	4	4
31	4	4	4
31	4	4	4
31	4	4	4
31	4	4	4
32	4	4	4
32	4	4	4
32	4	4	4

32	4	4	4
32	4	4	4
32	4	4	4
32	4	4	4
32	4	4	4
32	4	4	4
32	4	4	4
32	4	4	4
33	4	4	4
33	4	4	4
33	4	4	4
33	4	4	4
34	4	4	4
34	4	4	4
34	4	4	4
34	4	4	4
34	4	4	4
34	4	4	4
34	4	4	4
35	4	4	4
35	4	4	4
35	4	4	4
35	4	4	4
35	4	4	4
35	4	4	4
35	4	4	4
36	4	4	4
36	4	4	4
36	4	4	4
36	4	4	4
37	4	4	4
37	4	4	4
38	4	4	4
39	4	4	4
39	4	4	4
39	4	4	4
40	4	4	4
40	4	4	4
40	4	4	4
41	4	4	4
41	4	4	4
43	4	4	4
44	4	4	4
44	4	4	4
46	4	4	4
46	4	4	4
49	4	4	4
50	4	4	4
50	4	4	4
52	4	4	4
53	4	4	4
58	4	4	4

10.4 Camera Problem Raw Results

Random	1 Layer	2 Layer	3 Layer	4 Layer	5 Layer	6 Layer	7 Layer	8 Layer
38	5	5	10	100	12	7	12	7
91	3	100	6	4	2	2	3	6
17	7	100	2	6	6	10	6	7
56	6	100	7	6	7	8	7	7
39	4	100	2	7	5	7	6	10
9	6	7	5	7	7	3	6	10
10	4	8	6	6	12	12	12	9
8	6	5	4	100	11	7	7	12
68	6	5	6	7	7	5	10	11
23	7	100	100	7	7	11	11	6
38	7	100	10	6	12	12	11	7
42	7	3	6	5	5	12	7	7
16	100	3	7	9	7	12	7	6
33	100	100	100	4	7	10	6	7
24	100	6	7	6	10	5	7	5
11	100	100	4	7	7	12	11	7
46	6	7	100	7	2	8	6	4
17	100	3	6	5	4	8	6	7
14	5	6	5	7	11	7	7	6
4	5	6	100	9	6	6	7	6
43	100	2	100	6	7	7	7	7
8	100	5	7	6	11	3	12	2
28	7	6	7	7	5	9	7	7
52	3	5	7	9	6	4	7	3
9	6	10	7	8	4	6	7	7
11	100	12	100	10	6	5	10	7
27	7	7	6	100	7	10	6	7
22	7	3	6	100	7	12	6	7
3	6	7	7	4	3	7	12	2
8	100	100	100	11	6	6	7	10
10	100	7	12	3	2	6	7	5
58	5	5	6	7	8	4	3	7
25	100	6	100	6	4	11	6	7
15	7	100	6	9	6	12	8	12
7	3	4	6	7	5	10	6	7
13	4	2	3	7	6	5	7	7
18	7	100	5	5	6	7	7	7
19	3	100	7	9	12	3	12	7
8	4	2	100	7	6	6	7	7
33	3	100	7	7	7	7	6	6
23	6	3	100	6	7	6	8	3
23	7	100	100	100	5	6	10	11
12	7	100	7	7	2	7	11	12
8	4	5	12	7	6	12	7	7
9	100	6	7	5	7	10	2	5
23	6	7	6	6	5	6	6	7
20	100	100	100	6	11	6	7	5
33	6	7	6	6	2	7	8	7
10	100	6	6	6	3	6	11	2
8	2	2	7	7	6	11	12	11
67	100	5	100	6	6	6	7	12
11	6	5	2	6	12	9	7	3
16	7	6	100	8	11	2	4	12
13	7	6	7	7	10	12	7	2
55	100	100	6	9	4	7	6	9
5	100	7	6	2	6	4	12	9
11	6	4	12	5	12	7	12	7
25	100	6	6	2	3	5	7	6
14	7	100	12	7	6	12	12	7
7	100	7	5	11	11	5	7	9
21	2	7	5	6	6	11	2	7
10	100	6	9	100	7	6	6	10
3	4	100	7	6	7	12	7	9
28	100	5	2	6	12	7	6	8
75	100	100	7	6	11	6	11	9

15	7	6	8	3	8	6	12	7
27	7	100	5	5	100	7	6	12
83	6	100	7	8	100	12	7	7
16	3	5	11	2	7	12	6	7
8	100	3	3	7	5	7	7	12
7	7	2	6	4	4	8	3	9
10	7	6	100	100	6	7	7	5
6	100	100	6	6	6	12	10	5
6	100	100	5	6	10	7	11	12
29	100	6	4	6	5	5	11	9
15	100	3	5	6	7	6	7	6
24	100	5	5	11	8	6	5	6
20	3	6	5	6	7	7	12	6
9	3	100	5	8	6	12	6	5
22	2	4	100	4	11	6	7	7
16	7	100	6	7	11	6	7	7
23	100	100	6	6	7	7	7	12
5	6	7	3	4	6	7	7	7
67	6	5	100	3	6	7	6	5
13	6	7	4	11	3	6	7	6
4	100	100	6	7	7	7	11	6
6	7	100	7	8	6	12	12	7
36	7	6	100	7	7	12	7	2
36	5	7	7	7	12	7	7	7
16	6	100	7	5	6	7	3	5
9	7	6	5	6	3	2	12	10
13	100	6	6	3	7	7	6	10
31	6	7	8	8	4	6	7	6
22	100	7	100	10	8	9	7	9
7	100	7	7	9	4	7	7	7
31	100	100	100	5	5	11	7	10
19	7	6	4	3	7	6	7	9
8	100	2	6	2	11	7	2	5
26	7	6	6	100	4	4	2	8
15	4	100	7	3	6	5	8	6
11	7	4	100	6	6	6	11	2
18	7	100	7	6	11	11	4	6
5	100	100	7	5	7	6	5	7
32	3	100	6	3	2	12	7	3
35	5	6	5	4	12	7	11	12
50	100	6	6	11	11	7	2	6
10	6	5	6	6	12	9	12	6
27	6	3	7	6	10	5	7	6
9	7	7	6	6	7	6	4	7
18	2	6	2	7	7	7	7	7
20	100	6	7	5	5	8	12	7
26	100	5	7	4	7	5	4	7
25	6	5	7	6	6	11	8	6
7	5	5	6	11	7	11	6	7
12	4	5	5	6	12	6	8	7
16	100	5	7	4	6	6	11	5
52	5	4	7	5	8	11	6	7
10	100	7	100	4	6	11	7	12
28	7	4	7	7	7	7	12	7
49	100	5	11	6	6	7	11	7
11	7	100	7	3	11	6	7	7
100	6	100	100	12	12	7	7	6
36	6	4	5	5	12	11	11	6
73	4	100	4	7	7	6	7	7
4	2	12	6	6	7	6	12	6
42	100	100	100	9	4	8	6	4
77	7	5	6	6	11	12	12	7
13	100	6	6	7	7	7	6	6
31	100	6	7	8	7	2	7	7
16	2	6	5	6	7	6	4	7
12	7	5	4	6	3	7	12	12
17	100	5	5	6	7	7	7	7
100	6	7	100	2	6	6	6	6

25	7	100	7	100	6	6	7	7
20	6	6	6	10	6	7	7	7
34	100	100	7	7	6	12	11	7
100	100	7	100	3	6	2	6	7
58	100	100	12	6	7	4	6	9
9	100	5	7	12	100	7	6	11
24	100	7	6	5	5	6	6	12
43	6	6	7	5	7	12	7	11
42	100	4	6	100	7	4	6	7
15	7	2	7	8	3	12	6	7
18	7	5	5	7	11	7	12	3
12	7	6	100	11	7	6	10	5
23	5	100	6	7	8	7	4	5
65	100	100	6	6	2	7	7	6
7	7	4	4	6	10	12	7	9
19	7	100	7	7	12	7	7	7
7	6	8	7	6	10	10	6	7
81	6	3	7	2	7	6	12	7
7	6	100	7	12	5	7	6	4
5	7	100	7	7	9	11	6	6
4	5	6	4	6	5	6	2	6
4	2	6	6	7	7	11	7	4
11	100	6	7	4	5	2	7	9
7	100	6	100	7	11	7	11	7
34	7	5	3	6	7	11	8	7
33	100	100	6	7	2	7	6	7
14	7	7	5	6	6	7	11	5
21	4	100	7	6	6	6	7	4
14	100	100	100	6	6	6	11	5
6	7	7	6	4	6	7	7	7
7	100	7	6	9	7	5	11	7
17	4	7	5	4	4	11	7	6
43	7	3	6	9	5	7	12	6
26	4	100	5	5	6	11	6	7
92	100	100	6	9	11	9	3	7
6	7	7	7	12	7	12	6	12
17	100	6	9	6	7	6	7	5
17	100	100	6	5	4	6	6	7
5	7	100	7	7	10	6	7	6
7	6	7	100	7	10	6	7	4
18	2	6	12	5	7	6	7	11
8	7	6	6	7	6	9	7	12
78	6	100	7	100	5	7	7	3
16	3	6	3	7	7	12	11	7
19	4	7	11	4	6	7	7	7
10	7	7	7	6	8	12	6	12
76	6	100	5	8	12	7	6	6
28	5	100	7	5	7	5	11	9
17	100	6	12	7	6	2	11	6
35	100	7	7	7	6	5	8	7
18	6	12	100	10	7	7	12	4
21	7	10	7	11	3	7	7	12
13	5	2	6	5	2	7	7	11
21	6	6	7	7	3	7	7	7
14	7	100	4	2	7	12	6	2
26	3	100	4	7	12	6	6	5
34	6	8	5	6	6	2	7	12
13	100	100	7	100	7	6	7	5
19	2	6	7	12	7	8	7	6
14	3	5	100	6	7	12	11	5
28	6	2	100	7	11	12	7	6
19	6	100	7	7	8	11	10	7
20	6	100	4	12	6	12	5	7
17	7	7	6	7	2	7	11	5
17	7	4	4	6	7	7	6	6
14	7	6	4	3	6	4	7	7
4	7	7	7	7	7	7	7	6
10	100	100	7	7	7	6	12	7

11	100	7	6	4	6	7	7	7
6	100	5	6	5	100	6	7	6
21	5	7	100	4	9	6	7	6
3	5	6	3	6	7	3	12	12
31	4	4	6	5	7	5	7	7
44	7	100	6	6	6	6	7	9
18	7	7	12	6	7	2	10	2
28	5	4	5	5	7	7	2	8
33	7	6	10	5	6	7	2	7
6	7	7	100	6	8	7	6	7
21	7	7	100	6	12	6	6	12
5	5	7	100	7	7	7	11	8
18	3	100	4	8	2	6	7	7
26	7	7	100	3	11	6	7	7
38	100	100	5	100	4	11	2	6
100	6	7	12	7	9	6	7	12
25	4	100	2	6	11	7	7	7
7	100	6	100	6	8	7	8	7
13	7	6	7	6	7	12	12	7
48	7	6	100	6	4	11	11	12
5	100	4	100	9	6	7	6	12
15	100	7	8	7	4	9	7	12
14	100	4	4	7	7	7	6	6
12	7	7	100	6	9	10	5	4
11	5	3	100	2	5	12	8	12
18	100	4	4	7	6	3	10	9
41	2	6	9	9	6	6	2	12
84	7	100	9	11	100	11	7	2
10	5	7	7	12	6	10	4	2
32	2	6	6	6	3	8	7	6
9	100	6	4	4	2	7	6	12
43	5	7	100	6	2	10	6	6
4	6	7	5	4	8	12	6	6
16	100	100	9	6	6	7	12	10
2	7	6	4	11	7	7	7	7
16	2	6	6	6	12	12	6	12
14	7	4	7	7	11	6	12	7
16	6	10	100	5	11	11	6	6
52	6	6	6	8	11	11	6	2
8	4	3	7	8	7	7	6	7
9	100	3	5	7	100	6	6	2
17	4	10	6	6	2	5	7	7
43	3	8	6	10	6	7	7	6
4	7	6	5	5	11	2	6	7
10	6	5	3	11	7	6	3	2
12	100	100	7	8	7	6	6	7
19	100	100	7	100	12	6	4	6
33	7	6	8	6	12	9	5	11
4	100	4	6	6	6	5	6	11
8	100	5	7	100	12	7	10	4
21	100	100	7	6	5	3	7	7
30	7	7	7	4	7	12	11	10
11	7	100	12	9	7	7	7	3
12	7	100	6	8	7	7	12	7
8	7	7	100	5	7	7	7	7
28	7	100	4	5	100	5	7	12
3	100	6	9	11	8	6	6	10
19	7	3	100	7	12	7	7	7
7	7	5	7	5	5	9	8	7
6	100	3	3	12	7	10	12	6
82	7	2	100	6	6	7	10	7
14	100	6	7	100	4	7	7	6
19	4	6	100	7	7	12	7	7
4	7	100	7	8	7	3	4	6
8	100	6	7	5	12	7	7	7
9	100	6	6	100	4	9	11	7
13	6	7	2	7	100	7	11	7
31	100	3	11	9	6	6	12	12

16	3	7	11	6	3	10	8	6
21	100	100	7	6	100	10	7	5
58	5	6	4	7	7	10	6	11
3	7	3	8	5	4	11	7	5
6	6	6	6	6	6	7	6	7
12	7	100	7	5	9	6	7	7
36	7	4	7	7	7	2	9	8
8	3	5	7	8	11	7	12	9
8	5	4	2	11	2	7	7	7
12	100	7	100	7	100	8	12	10
25	100	7	6	6	6	6	6	7
3	5	5	5	5	7	6	7	5
16	100	2	6	100	7	11	10	7
19	7	100	7	2	8	7	6	12
8	100	100	5	6	6	7	12	2
21	6	3	9	9	5	7	7	7
29	6	6	4	6	6	7	3	7
17	100	4	7	10	7	7	6	6
18	6	7	4	100	7	6	7	12
35	100	100	100	100	6	7	7	12
6	6	7	7	7	6	6	7	3
20	100	5	9	8	7	7	8	12
24	100	100	9	7	6	3	7	6
10	100	7	100	4	4	11	11	5
2	2	6	100	100	4	6	11	6
26	6	7	7	7	12	6	7	7
33	6	100	6	6	7	12	7	12
19	6	7	100	9	7	12	7	7
5	100	6	5	7	5	7	7	7
11	100	100	100	3	100	4	4	8
21	6	5	7	100	7	9	11	9
29	6	7	100	6	12	7	12	6
26	7	100	12	4	12	6	7	5
28	100	6	100	3	11	7	10	5
100	5	6	100	7	3	12	12	7
27	7	8	4	6	9	7	6	5
5	7	5	8	6	7	6	6	7
23	7	100	5	7	11	7	11	7
11	100	3	4	7	12	6	11	6
10	6	8	5	6	10	8	4	6
12	100	7	12	12	7	7	3	7
41	100	100	4	4	7	7	3	7
14	6	100	6	6	100	6	7	9
51	100	100	6	10	12	11	7	7
11	4	5	100	7	4	6	4	6
11	7	7	6	5	7	12	12	5
14	100	4	7	7	7	6	6	6
4	5	7	7	5	7	7	7	12
12	6	7	5	6	6	3	7	7
100	7	7	5	9	6	6	5	7
26	100	5	11	7	7	6	7	6
10	5	4	7	5	7	4	7	7
83	100	6	6	3	6	6	6	7
4	7	100	7	100	5	6	6	6
60	100	100	5	11	7	12	7	3
35	7	100	6	7	7	7	12	6
14	4	4	100	9	6	7	12	9
22	100	3	6	9	6	6	6	12
36	7	7	100	100	7	11	7	6
19	6	5	7	6	6	11	12	5
6	100	2	7	3	7	7	5	5
32	100	6	6	100	3	7	7	6
6	7	3	7	6	6	4	7	6
32	7	7	7	100	4	7	7	10
10	100	6	4	7	7	7	5	7
8	100	6	3	6	10	6	7	4
19	7	100	8	10	6	6	12	12
18	5	100	100	6	7	6	7	7

27	3	5	3	10	7	4	6	7
30	100	6	7	6	11	8	8	12
25	6	7	6	6	6	7	7	6
11	100	5	4	8	11	7	7	5
96	7	100	7	2	4	6	7	6
9	100	100	4	100	7	6	9	7
15	7	6	100	4	2	4	7	7
54	3	12	7	6	7	7	7	7
26	4	4	100	6	12	7	7	10
48	100	4	100	4	4	6	7	7
23	7	7	5	6	6	7	8	3
19	7	100	5	10	7	6	7	7
15	100	100	11	9	7	7	8	7
21	6	5	100	9	7	11	5	9
13	4	5	7	100	7	11	6	3
7	7	5	6	6	6	7	6	7
51	100	4	6	2	6	9	12	5
23	7	5	7	5	6	6	6	7
10	7	7	4	7	100	7	6	6
46	100	7	3	7	6	10	12	4
14	100	100	4	7	8	7	6	7
5	100	6	6	100	6	7	5	12
34	100	3	7	6	6	7	6	7
17	100	5	9	3	11	7	6	7
49	3	100	7	5	6	7	6	7
10	2	5	6	100	100	6	12	7
29	100	6	6	8	7	6	12	9
95	7	4	100	8	12	6	12	6
13	7	7	4	7	5	5	4	6
15	2	2	3	2	8	4	6	12
50	5	7	10	6	12	8	7	7
3	6	6	100	100	7	6	3	7
22	7	5	3	7	5	7	12	12
22	100	8	6	6	11	6	3	6
84	7	100	6	6	7	4	6	10
16	4	6	6	5	7	8	6	6
15	7	100	6	100	11	7	7	7
17	5	7	7	5	7	6	8	6
24	100	12	10	9	6	6	9	7
3	100	100	12	6	5	8	7	9
3	7	3	7	7	7	7	7	6
7	100	7	7	7	7	12	12	9
27	100	6	5	6	12	6	7	7
12	4	7	100	11	11	12	7	9
2	6	4	100	7	4	12	10	8
54	3	4	6	7	12	10	6	7
7	100	4	7	2	7	12	6	7
9	100	5	100	10	6	6	6	6
22	100	7	4	7	11	7	6	8
46	7	6	4	7	8	4	8	7
5	100	100	6	7	4	7	7	5
82	7	7	7	10	12	7	12	6
10	7	7	8	100	12	7	3	7
19	7	100	6	6	6	7	12	5
11	100	100	5	10	7	7	2	10
11	3	7	7	12	7	7	6	7
25	7	4	6	100	4	7	6	5
10	7	6	7	7	4	10	6	6
26	7	4	100	6	7	6	11	7
17	6	7	6	9	8	7	3	6
59	7	100	7	7	6	7	6	7
7	7	7	7	7	6	7	8	5
8	2	3	7	7	7	12	7	7
2	5	6	10	8	5	7	6	9
24	100	100	7	3	6	6	7	6
18	100	7	6	5	7	5	6	6
18	2	7	6	5	10	7	12	7
16	100	3	6	7	6	8	5	7

39	7	7	100	5	7	7	12	7
30	6	100	5	6	11	12	11	7
19	4	100	5	12	5	11	4	7
33	7	12	7	5	6	7	7	12
35	6	6	4	6	7	7	6	7
9	7	7	12	9	7	7	6	12
13	4	3	4	7	10	7	12	12
13	6	6	7	5	6	10	6	10
3	7	7	2	7	6	2	8	5
10	5	4	7	5	7	4	7	11
4	7	100	5	5	7	11	7	7
6	100	6	6	6	7	7	12	7
13	3	7	100	6	6	2	12	7
8	100	6	8	7	7	8	7	7
39	3	7	100	5	100	7	7	6
100	100	6	100	6	5	4	7	8
18	100	2	6	9	7	7	10	8
12	3	5	6	5	7	6	7	5
12	100	5	7	11	7	11	7	7
5	7	3	7	100	7	5	6	4
100	100	7	4	11	11	7	12	6
5	4	4	5	7	7	12	9	9
38	3	7	100	10	6	12	7	12
10	100	5	9	4	4	4	6	12
26	100	100	5	100	10	6	7	8
100	7	100	10	7	6	11	10	7
30	100	100	100	5	7	12	10	7
12	7	5	2	9	12	6	12	6
17	100	6	12	6	12	11	12	4
10	100	7	7	6	2	8	12	7
30	100	100	4	6	7	7	8	7
21	100	4	100	6	3	7	8	7
17	7	4	5	100	11	12	12	7
13	7	7	6	6	7	2	12	7
27	5	100	7	7	5	12	6	6
11	100	3	6	6	9	6	5	4
8	7	6	6	11	11	7	8	12
11	7	5	5	7	11	2	11	6
6	100	7	6	6	7	6	7	7
17	7	4	7	2	11	7	6	12
8	100	4	4	5	7	6	6	7
71	6	7	100	8	6	2	6	9
13	7	100	7	4	6	6	12	7
40	100	5	3	6	6	6	6	7
76	100	2	100	6	5	7	8	6
28	7	5	6	7	10	6	6	8
30	4	100	100	7	6	12	12	7
14	7	100	100	6	6	2	7	7
23	4	7	100	7	7	6	12	7
43	100	4	7	6	6	7	7	5
26	6	4	7	100	6	4	4	5
27	100	6	6	7	6	3	7	7
10	4	100	6	7	6	7	7	7
21	7	12	7	9	7	12	6	9
30	7	100	100	8	6	7	7	7
15	100	6	5	7	7	8	7	7
20	3	5	6	7	7	6	5	8
10	6	100	7	8	8	7	6	10
4	100	4	7	7	6	7	2	7
8	4	7	6	3	6	6	6	12
14	7	100	7	100	8	7	6	11
8	100	3	5	4	11	7	7	12
8	100	100	4	6	11	7	4	5
19	5	5	100	3	7	7	7	6
7	6	7	100	8	8	6	7	6
25	4	7	9	7	12	6	8	12
16	100	7	6	6	6	12	6	10
20	100	100	7	5	7	3	7	6

37	5	4	4	8	7	7	8	6
13	7	2	12	6	7	10	12	10
13	5	7	7	6	10	12	7	2
18	3	10	6	8	6	2	7	7
24	3	3	7	4	7	7	6	12
3	6	6	2	7	11	7	7	7
11	3	3	12	6	4	5	6	7
8	100	3	7	5	12	2	7	4
36	100	100	6	6	6	7	8	11
93	100	6	6	4	7	12	11	7
5	5	2	7	12	7	6	6	7
13	100	7	11	5	10	8	6	7
14	100	100	4	9	4	5	12	9
11	100	7	6	6	7	3	7	6
22	100	6	100	6	5	4	7	10
10	7	100	7	6	3	7	7	4
3	6	4	6	12	100	7	12	11
11	3	7	4	7	11	5	4	5
16	3	2	100	100	10	6	7	12
20	2	6	7	7	2	12	7	3
8	100	2	100	100	7	7	7	7
9	100	7	6	6	100	8	4	8
25	5	100	7	10	2	7	7	7
30	100	8	7	4	5	7	7	12
16	100	4	10	100	7	6	7	5
11	100	4	100	7	6	7	7	12
12	100	6	100	9	8	3	11	7
100	4	100	7	7	9	12	10	10
3	6	100	2	100	3	6	12	10
18	4	4	6	6	7	12	7	7
17	5	7	7	6	5	7	5	7
56	7	4	7	6	4	5	7	7
35	7	3	6	7	7	11	7	12
55	100	2	100	6	6	12	7	7
36	100	7	6	4	5	7	8	6
17	3	100	100	7	6	7	6	12
9	3	4	3	6	7	7	12	9
13	100	6	7	100	2	7	12	12
19	100	2	6	9	7	12	7	7
24	4	4	7	10	12	11	6	5
43	5	100	7	5	7	7	6	12
7	100	7	7	4	6	6	11	6
8	7	10	7	4	7	3	4	6
19	2	4	7	100	7	10	8	11
14	6	12	6	4	6	12	11	7
14	7	3	12	10	7	6	2	8
12	100	4	4	6	4	6	6	12
6	100	6	100	100	6	10	11	12
100	100	5	3	12	7	11	7	6
13	100	100	7	6	8	3	7	6
26	6	100	4	7	12	11	4	7
58	7	7	6	11	12	7	12	11
11	7	7	9	4	7	10	12	6
7	2	100	8	100	7	11	8	3
22	7	2	5	7	6	5	6	7
6	4	100	6	6	5	12	6	5
9	100	100	4	7	6	7	2	7
20	7	100	7	11	7	6	7	12
15	5	100	7	7	7	8	6	8
44	7	6	6	6	7	7	7	7
21	100	7	6	7	7	5	11	12
31	100	5	100	6	7	2	7	8
64	3	6	100	100	6	7	10	7
11	100	7	7	100	100	6	7	6
8	100	4	6	4	6	6	7	6
10	4	3	100	7	7	6	6	5
17	7	10	4	5	8	9	7	7
17	7	7	7	6	100	6	6	7

7	4	3	3	6	100	7	11	6
17	4	6	7	4	7	2	8	9
100	5	4	4	7	12	5	12	7
20	6	2	12	8	12	6	11	9
9	6	100	3	7	5	6	5	4
28	6	7	6	5	9	7	2	7
2	100	4	6	7	12	7	5	3
4	7	6	7	6	11	12	7	7
8	4	6	4	5	11	11	7	6
43	100	5	6	5	2	6	7	7
24	3	100	11	6	11	12	7	7
34	5	100	6	4	100	12	11	7
7	7	7	11	11	4	12	7	10
2	7	7	2	5	2	6	7	7
28	5	12	7	100	8	12	7	6
64	6	3	7	6	4	6	6	5
55	4	6	2	11	7	11	11	3
5	100	6	7	7	6	11	9	7
49	2	7	2	6	4	12	6	7
16	6	100	9	7	8	7	11	7
9	100	100	10	6	7	4	8	4
37	6	100	6	6	4	7	11	8
62	5	2	12	6	7	7	12	7
12	4	4	7	4	6	3	2	3
9	100	6	2	5	7	6	5	5
9	100	7	12	6	7	6	6	12
52	100	100	100	8	7	6	6	3
14	4	4	6	7	5	2	4	7
5	7	100	6	6	7	10	7	7
22	7	100	5	7	7	7	12	7
3	7	100	7	11	12	7	11	3
26	6	7	12	6	8	6	12	7
14	100	7	100	4	6	7	4	7
4	100	7	6	5	11	5	7	6
10	6	4	6	10	11	10	10	10
13	100	6	5	9	12	9	7	7
10	7	7	7	6	2	7	6	7
16	100	100	7	100	7	11	12	6
2	5	100	7	4	9	7	12	9
17	7	6	7	9	6	5	9	9
16	6	100	7	5	7	7	8	5
12	100	6	6	100	7	7	3	7
25	4	3	9	5	7	6	6	7
18	100	7	12	7	11	7	7	7
5	6	7	5	5	10	7	6	11
7	7	7	6	7	4	7	7	9
11	100	7	6	7	3	8	12	12
100	100	7	6	7	11	4	7	12
36	7	6	3	7	6	7	7	7
29	4	5	100	6	8	12	11	6
18	100	7	100	6	8	6	10	7
5	100	7	4	5	6	5	7	11
41	100	100	100	6	10	6	6	6
14	7	6	9	7	12	6	7	7
45	100	7	6	2	6	7	6	7
43	6	4	12	6	6	6	11	9
6	3	7	7	7	11	7	5	5
73	6	7	100	100	7	7	5	7
23	7	7	6	3	5	7	7	7
7	100	100	4	6	7	7	6	10
23	7	100	7	11	6	6	7	7
10	7	7	9	100	6	7	7	6
6	6	6	5	5	11	6	3	7
11	7	6	7	3	6	6	7	7
20	2	6	6	6	12	12	2	6
7	4	7	4	9	6	3	8	7
24	6	5	5	3	6	2	7	11
100	7	3	100	100	7	5	6	7

4	7	5	7	5	8	11	6	6
20	3	4	9	100	7	6	7	9
9	6	7	100	7	8	7	7	6
8	7	4	6	7	3	7	6	7
6	100	7	5	6	12	7	7	7
45	7	7	7	12	5	12	7	7
40	7	4	2	12	7	8	10	12
37	4	100	7	11	7	7	6	7
100	4	5	12	2	7	12	2	7
25	4	7	100	7	7	11	7	7
16	4	6	100	6	7	11	12	7
8	7	100	7	10	6	6	6	7
20	5	7	8	6	7	4	12	2
28	100	100	6	6	5	6	5	10
15	100	7	7	12	2	2	10	7
21	100	6	6	6	12	7	4	11
13	7	7	100	11	8	6	9	7
17	100	2	6	6	10	7	7	7
9	7	5	2	3	9	11	7	7
69	2	5	7	6	7	12	7	5
11	100	7	7	6	7	12	10	10
20	7	7	6	4	7	12	6	6
9	4	3	12	11	7	4	10	7
17	2	6	7	7	4	10	7	7
18	7	3	8	6	7	12	11	7
28	100	100	6	7	3	2	9	7
2	3	10	4	7	3	6	7	7
6	3	100	10	5	6	7	2	7
5	4	7	6	4	6	7	6	11
11	6	8	100	7	7	12	11	9
12	2	100	6	2	2	7	7	7
35	2	100	5	5	5	9	4	5
7	100	5	4	6	6	6	7	6
13	4	6	3	7	7	7	7	12
14	5	4	100	9	11	12	7	7
17	4	4	9	8	100	7	6	5
100	100	100	100	10	7	12	7	5
3	5	100	6	7	4	7	8	12
32	7	100	7	12	2	7	12	12
21	100	6	7	10	7	2	7	8
25	5	6	4	6	7	7	12	12
7	7	2	100	6	11	12	6	7
5	100	6	5	100	9	12	7	12
8	4	7	6	6	6	7	8	7
36	4	5	100	100	11	7	7	7
8	5	7	7	5	12	5	12	7
12	100	100	7	3	7	10	4	9
34	100	100	12	7	6	7	6	12
18	100	3	6	9	6	10	6	6
23	100	5	100	11	7	7	7	5
43	7	7	7	10	7	7	6	7
59	7	6	6	4	6	11	5	7
16	100	4	5	6	5	7	10	7
13	5	4	100	7	6	6	10	10
17	4	100	9	10	7	6	7	7
20	6	7	7	7	11	6	6	12
18	7	6	7	7	6	4	6	7
22	5	4	4	7	12	7	6	12
34	100	100	100	2	6	6	6	5
10	7	7	6	5	7	7	7	12
73	100	8	2	4	8	12	7	8
3	6	2	4	5	7	6	7	7
9	7	6	6	7	4	6	5	6
24	5	3	6	7	12	6	7	5
34	100	2	100	12	10	12	2	8
15	5	7	4	9	6	12	4	12
5	4	7	6	7	7	6	7	2
97	100	6	7	5	12	2	2	6

4	7	7	5	6	6	12	7	7
8	5	6	100	10	6	7	12	2
25	100	7	7	7	5	2	12	7
40	5	2	100	11	7	7	7	7
11	7	10	3	7	4	12	6	6
85	7	7	6	100	100	6	6	7
10	100	6	4	12	6	7	5	12
6	100	6	11	5	11	5	7	11
8	100	5	100	5	6	6	7	7
16	7	6	7	7	6	11	7	7
21	5	6	6	3	7	12	7	5
14	7	5	6	7	5	3	12	7
6	100	6	7	100	7	4	2	5
18	100	7	7	11	7	7	3	7
73	7	3	11	100	7	7	7	12
5	7	4	7	6	6	7	7	7
13	5	7	7	4	7	3	11	7
6	100	100	12	7	2	5	7	12
16	7	6	11	6	7	7	2	7
9	100	100	8	6	7	7	7	7
7	6	7	100	5	2	7	12	7
11	100	7	3	4	4	4	6	7
5	5	4	5	7	12	12	11	9
78	100	4	6	9	12	11	12	7
6	7	6	6	100	6	7	7	8
26	6	7	5	100	7	7	6	6
30	4	3	7	100	2	11	11	9
25	7	7	6	6	7	7	7	7
100	6	100	5	12	7	3	11	7
60	100	10	12	100	6	12	12	5
78	6	100	7	6	100	12	6	7
10	4	100	9	7	3	6	4	6
27	100	6	2	11	7	11	11	7
31	7	100	6	7	10	7	7	12
73	7	7	7	100	2	6	7	7
4	6	7	5	11	4	3	6	7
6	6	100	6	10	6	2	7	7
100	4	100	100	6	8	7	11	3
37	7	7	7	100	6	3	11	5
30	7	100	6	5	11	7	7	6
50	100	100	3	6	7	12	7	7
2	100	100	6	3	8	6	6	5
4	5	100	7	6	6	4	6	5
16	6	2	4	6	6	11	6	7
32	100	10	7	5	4	6	8	6
11	3	5	6	12	12	6	12	12
27	6	12	5	6	6	7	11	7
20	6	100	2	7	7	11	6	12
29	7	6	7	6	7	12	7	6
21	7	6	100	7	7	10	3	7
13	100	5	4	7	6	6	11	7
21	3	100	9	9	8	6	6	7
4	100	7	6	7	12	12	12	9
20	100	5	12	6	6	6	11	7
8	7	100	5	10	6	6	11	7
100	3	2	6	6	4	5	12	6
24	100	4	5	12	7	3	2	10
18	100	100	6	7	11	8	11	7
4	100	5	100	10	7	3	7	5
10	100	5	7	7	6	7	6	6
9	5	6	4	6	2	2	6	7
17	100	7	6	2	7	7	2	8
15	2	10	5	6	6	7	10	12
3	6	6	4	6	12	7	7	7
32	7	3	7	8	7	12	12	7
18	7	6	8	9	10	8	11	10
16	100	7	6	5	4	6	12	2
11	100	3	100	6	8	6	2	10

37	7	3	100	8	7	12	10	7
10	100	6	6	7	12	7	8	10
6	4	100	100	7	7	7	7	7
16	7	4	4	5	6	11	4	3
8	3	6	11	7	7	10	7	6
5	100	7	5	7	7	6	7	7
58	7	7	8	6	7	11	7	5
12	2	7	6	7	6	6	6	5
3	3	7	6	11	6	6	6	7
13	100	3	7	8	100	7	7	7
25	5	5	8	9	7	7	6	6
14	6	7	100	7	11	9	11	12
12	3	100	7	8	6	6	7	6
60	7	100	8	9	6	7	9	7
23	100	7	12	11	7	5	12	2
4	4	100	6	2	8	8	6	8
12	7	5	5	3	7	8	5	5
35	100	7	6	4	12	4	4	12
30	7	4	100	6	5	12	7	7
17	100	7	6	6	6	7	8	7
55	6	5	6	6	11	6	7	9
12	6	6	100	100	7	6	5	6
13	100	2	6	4	7	6	2	6
53	7	7	7	4	6	4	7	12
21	100	7	7	6	5	11	2	8
12	100	7	6	10	2	7	7	6
13	7	5	6	7	7	12	7	8
29	5	6	6	5	12	7	11	7
56	100	100	7	7	100	8	7	7
16	7	6	7	100	7	12	11	5
39	100	100	6	6	7	4	10	7
51	6	3	100	11	7	6	11	7
73	7	6	100	6	4	7	10	7
7	4	7	5	9	7	6	5	5
13	100	7	6	7	12	6	4	7
4	100	6	9	4	100	11	8	5
100	5	7	100	5	6	7	11	7
19	100	2	5	100	5	6	3	7
30	100	4	7	6	7	7	7	7
74	5	4	6	10	6	5	2	12
24	100	6	6	5	6	12	12	12
11	7	100	5	7	7	5	6	7
53	7	6	12	5	6	12	7	7
3	100	7	100	9	7	11	6	6
24	4	100	7	7	12	7	7	9
18	100	4	6	11	7	11	7	7
10	2	100	6	5	3	12	6	7
8	5	100	7	4	10	4	11	7
26	7	3	100	6	5	6	5	2
20	7	6	100	12	12	12	6	2
2	5	7	6	6	7	12	7	7
10	7	6	7	4	6	7	11	7
9	100	100	6	7	3	7	6	7
17	7	5	2	100	6	5	7	4
14	5	7	7	10	6	12	12	12
33	3	100	3	11	6	3	7	6
8	100	8	100	100	10	6	7	7
7	100	10	7	11	10	7	10	6
7	100	4	6	6	12	6	7	11
22	6	6	7	2	8	7	7	8
36	100	10	5	8	11	7	6	6
61	100	4	5	6	7	12	12	7
18	7	5	7	5	6	6	6	9
19	100	10	11	7	12	12	11	8
84	100	100	6	100	7	6	6	7
13	6	5	7	4	100	3	6	10
16	100	100	6	100	6	6	6	12
11	100	5	7	7	6	11	11	6

23	3	7	7	7	2	7	6	12
38	7	4	2	5	2	6	11	7
22	100	7	6	7	7	6	11	5
14	5	3	7	7	4	8	7	7
16	100	6	4	6	100	3	11	7
13	7	7	4	7	6	4	7	9
67	100	4	5	4	7	7	3	7
21	7	3	100	100	7	12	6	9
2	100	6	100	9	100	6	7	9
37	7	100	5	100	11	7	7	5
21	7	100	6	7	2	2	3	3
10	4	6	10	7	3	7	8	6
8	100	7	7	7	7	7	7	7
14	7	7	2	5	5	12	5	6
11	100	4	6	6	6	11	4	12
12	2	8	11	10	8	2	7	10
48	100	12	100	4	7	8	6	11
26	6	10	6	6	3	3	7	9
42	4	6	9	8	9	2	4	7
32	4	3	11	5	5	7	7	7
35	2	7	5	7	6	11	7	6
11	100	100	7	6	7	6	4	6
27	100	100	7	7	6	8	4	9
10	100	100	2	5	6	12	7	6
12	100	10	12	6	12	7	5	4
6	100	5	100	10	11	7	5	7
11	100	100	100	100	8	7	6	7
14	100	4	6	5	7	7	7	7
24	5	2	7	5	11	7	8	7
14	100	7	6	4	6	7	12	6
93	7	7	6	7	7	12	8	4
90	100	3	4	100	7	4	7	4
18	2	100	4	5	5	11	7	2
4	6	5	11	100	100	12	7	7
67	3	100	100	6	4	7	6	12
29	7	100	5	4	9	4	6	3
32	100	5	4	6	7	11	7	7
54	7	100	5	9	6	4	2	7
5	2	7	100	10	12	2	2	7
7	4	5	6	100	6	6	11	7
57	3	4	7	2	9	12	11	6
12	7	100	7	4	6	6	7	4
7	5	100	6	6	8	2	7	6
12	6	100	7	10	6	4	11	7
11	7	100	6	7	6	8	6	7
9	100	7	12	6	6	7	11	6
22	7	7	3	7	5	7	5	4
8	100	3	7	12	2	6	7	8
21	2	6	6	7	7	7	11	7
79	100	100	6	7	6	7	6	7
36	100	100	7	6	6	7	8	5
51	5	3	8	2	12	7	6	7
39	100	7	6	6	7	4	5	9
11	6	7	100	7	12	2	12	11
26	7	100	7	8	12	7	12	11
23	100	7	7	9	7	6	7	6
14	4	100	4	10	5	12	6	12
94	100	7	4	3	7	8	7	7
35	7	3	5	6	7	5	8	6
21	100	7	11	10	6	12	7	2
51	100	7	7	4	6	3	11	11
15	7	7	100	6	6	11	7	7
4	3	4	5	6	7	2	7	5
9	6	7	100	6	7	6	7	10
17	6	7	6	5	7	7	7	5
5	7	7	5	6	10	7	7	10
14	7	4	100	3	2	6	11	7
13	7	2	6	6	7	12	7	7

14	7	100	100	100	11	6	12	6
3	5	4	6	11	7	7	2	7
8	100	100	7	100	6	11	7	7
65	100	100	7	2	7	7	9	12
100	100	2	5	7	6	10	11	6
22	100	6	4	6	6	6	6	6
7	100	100	7	2	7	5	12	7
13	7	7	7	7	12	7	7	10
41	5	5	6	7	7	7	6	6
4	7	3	100	7	6	7	7	7
36	6	4	7	6	2	12	7	6
8	6	100	7	5	5	2	8	7
10	6	7	2	9	3	6	4	7
15	7	8	6	7	3	6	11	7
7	3	7	100	7	6	6	7	10
27	6	7	11	6	6	11	11	6
26	7	12	6	100	9	7	4	7
5	100	12	4	5	7	6	11	5
20	6	6	4	7	7	7	7	12
100	100	5	7	7	6	6	7	12
4	6	5	7	6	7	7	12	7
53	7	100	6	5	6	6	7	5
6	100	6	100	5	7	6	7	8
3	3	4	100	8	7	8	11	9
13	5	5	100	5	7	6	6	7
13	4	100	7	9	2	11	7	12
10	3	10	2	5	7	4	9	7
43	100	7	100	100	6	6	7	3
30	7	100	6	6	11	4	4	9
19	7	6	5	5	6	12	9	5
12	100	3	6	4	7	6	7	5
9	7	6	4	7	5	6	11	12
13	5	5	7	9	7	12	4	9
10	100	6	100	100	12	6	7	6
5	7	2	3	9	4	7	7	7
17	7	6	7	7	12	6	7	7
16	100	100	3	7	7	7	7	4
65	7	7	6	5	7	7	12	12
10	5	100	100	12	5	6	7	7
15	7	7	100	12	6	8	7	7
17	5	7	7	9	6	10	7	7
15	6	4	6	4	100	6	7	7
40	7	7	7	6	11	12	12	3
11	100	7	6	11	2	4	7	7
4	100	100	100	6	6	7	11	6
18	7	6	6	6	3	6	7	7
13	7	6	7	8	6	6	7	12
10	5	7	4	10	6	11	7	9
33	7	7	4	6	11	11	3	7
16	3	100	7	6	3	4	11	12
3	100	3	7	6	2	6	4	7
52	100	7	6	7	7	7	8	6
20	100	100	7	12	6	12	7	9
8	7	100	6	100	12	11	4	7
8	6	6	7	5	7	4	6	7
21	5	5	6	6	5	6	6	12
12	100	6	9	7	11	7	7	12
9	6	7	6	11	2	6	10	7
66	100	3	7	6	6	6	11	6
7	7	7	3	10	11	6	7	8
41	6	4	7	6	7	6	12	6
64	100	100	5	9	12	8	7	6
29	2	5	6	2	5	9	12	7
9	7	7	6	6	11	6	7	6
3	6	7	2	7	11	7	6	7
50	100	6	6	4	7	10	7	5
77	6	100	2	3	5	7	10	7
13	5	4	7	7	6	12	6	12

10	5	6	7	3	7	7	7	6
25	7	7	7	5	2	7	7	7
12	6	6	6	5	6	7	6	2
5	100	3	7	9	7	4	7	12
31	7	100	6	7	6	7	12	11
12	6	7	7	6	11	7	6	7
41	100	7	8	6	6	3	7	12
11	7	3	100	5	5	12	2	3
13	100	6	7	3	7	12	7	7
28	6	100	7	7	7	10	6	7
4	7	7	100	100	12	7	7	7
17	3	3	6	7	5	7	6	7
8	6	6	3	11	8	6	12	7
14	7	5	7	7	7	6	7	2
70	100	7	6	7	11	6	6	7
16	100	100	100	12	6	7	6	4
27	5	7	5	100	11	7	7	6
18	7	6	4	12	6	6	11	5
17	5	7	100	4	100	12	7	7
13	100	4	7	7	7	10	7	4
3	7	7	6	11	7	12	10	7
9	7	100	100	6	5	7	5	7
23	100	3	7	6	6	11	10	12
33	2	4	7	11	12	7	7	9
23	7	6	100	4	12	7	2	7
53	100	4	10	5	11	2	6	7
17	4	7	100	6	6	6	6	12
27	100	4	100	7	7	4	6	6
14	5	7	2	5	7	6	6	7
7	100	100	9	6	6	6	12	7
26	6	7	4	4	6	6	6	7
24	7	4	7	7	7	7	7	9
23	7	7	2	6	7	10	10	7
18	6	7	7	7	11	6	8	4
55	7	100	7	2	12	12	6	7
8	100	3	100	3	6	5	7	6
6	5	7	7	7	11	6	7	7
38	5	7	5	100	12	7	7	7
3	100	6	100	4	6	6	8	7
7	7	4	4	11	7	6	6	2
6	100	5	8	100	3	7	12	9
64	100	7	7	4	6	11	7	6
14	100	6	100	6	7	4	12	12
7	100	5	4	4	7	8	5	11
3	4	7	7	5	11	7	6	7
6	7	2	7	6	6	7	7	6
39	6	6	5	6	10	7	6	7
11	6	3	100	6	9	6	6	7
40	7	2	5	12	7	6	2	6
17	100	2	7	6	6	10	7	7
21	7	6	4	100	7	7	7	7
32	5	4	9	6	11	7	7	7

10.5 Camera Problem Open-Loop Raw Results

Random	1 Layer	2 Layer	3 Layer	4 Layer	5 Layer	6 Layer	7 Layer	8 Layer
40	7	7	7	7	7	7	7	7
16	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
12	7	7	7	7	7	7	7	7
20	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
34	7	7	7	7	7	7	7	7
20	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
32	7	7	7	7	7	7	7	7
23	7	7	7	7	7	7	7	7
43	7	7	7	7	7	7	7	7
57	7	7	7	7	7	7	7	7
39	7	7	7	7	7	7	7	7
36	7	7	7	7	7	7	7	7
18	7	7	7	7	7	7	7	7
37	7	7	7	7	7	7	7	7
29	7	7	7	7	7	7	7	7
18	7	7	7	7	7	7	7	7
21	7	7	7	7	7	7	7	7
48	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
25	7	7	7	7	7	7	7	7
23	7	7	7	7	7	7	7	7
14	7	7	7	7	7	7	7	7
25	7	7	7	7	7	7	7	7
29	7	7	7	7	7	7	7	7
31	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
24	7	7	7	7	7	7	7	7
25	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
12	7	7	7	7	7	7	7	7
14	7	7	7	7	7	7	7	7
20	7	7	7	7	7	7	7	7
32	7	7	7	7	7	7	7	7
18	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
29	7	7	7	7	7	7	7	7
50	7	7	7	7	7	7	7	7
19	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
40	7	7	7	7	7	7	7	7
36	7	7	7	7	7	7	7	7
32	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
39	7	7	7	7	7	7	7	7
19	7	7	7	7	7	7	7	7
29	7	7	7	7	7	7	7	7
12	7	7	7	7	7	7	7	7
16	7	7	7	7	7	7	7	7
21	7	7	7	7	7	7	7	7
29	7	7	7	7	7	7	7	7
12	7	7	7	7	7	7	7	7
35	7	7	7	7	7	7	7	7
24	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7

9	7	7	7	7	7	7	7	7
12	7	7	7	7	7	7	7	7
28	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
24	7	7	7	7	7	7	7	7
21	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
18	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
16	7	7	7	7	7	7	7	7
14	7	7	7	7	7	7	7	7
20	7	7	7	7	7	7	7	7
16	7	7	7	7	7	7	7	7
20	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
16	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
27	7	7	7	7	7	7	7	7
48	7	7	7	7	7	7	7	7
38	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
33	7	7	7	7	7	7	7	7
46	7	7	7	7	7	7	7	7
12	7	7	7	7	7	7	7	7
23	7	7	7	7	7	7	7	7
29	7	7	7	7	7	7	7	7
16	7	7	7	7	7	7	7	7
14	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
18	7	7	7	7	7	7	7	7
14	7	7	7	7	7	7	7	7
31	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
25	7	7	7	7	7	7	7	7
35	7	7	7	7	7	7	7	7
25	7	7	7	7	7	7	7	7
16	7	7	7	7	7	7	7	7
21	7	7	7	7	7	7	7	7
19	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
16	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
7	7	7	7	7	7	7	7	7
10	7	7	7	7	7	7	7	7
53	7	7	7	7	7	7	7	7
30	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
18	7	7	7	7	7	7	7	7
12	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
31	7	7	7	7	7	7	7	7
30	7	7	7	7	7	7	7	7
20	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
7	7	7	7	7	7	7	7	7
23	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
42	7	7	7	7	7	7	7	7
18	7	7	7	7	7	7	7	7
12	7	7	7	7	7	7	7	7
38	7	7	7	7	7	7	7	7

22	7	7	7	7	7	7	7	7
23	7	7	7	7	7	7	7	7
18	7	7	7	7	7	7	7	7
16	7	7	7	7	7	7	7	7
14	7	7	7	7	7	7	7	7
37	7	7	7	7	7	7	7	7
14	7	7	7	7	7	7	7	7
18	7	7	7	7	7	7	7	7
19	7	7	7	7	7	7	7	7
29	7	7	7	7	7	7	7	7
26	7	7	7	7	7	7	7	7
12	7	7	7	7	7	7	7	7
8	7	7	7	7	7	7	7	7
24	7	7	7	7	7	7	7	7
18	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
12	7	7	7	7	7	7	7	7
8	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
19	7	7	7	7	7	7	7	7
10	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
31	7	7	7	7	7	7	7	7
16	7	7	7	7	7	7	7	7
38	7	7	7	7	7	7	7	7
48	7	7	7	7	7	7	7	7
28	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
16	7	7	7	7	7	7	7	7
24	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
41	7	7	7	7	7	7	7	7
51	7	7	7	7	7	7	7	7
8	7	7	7	7	7	7	7	7
14	7	7	7	7	7	7	7	7
26	7	7	7	7	7	7	7	7
36	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
21	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
22	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
12	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
14	7	7	7	7	7	7	7	7
32	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
29	7	7	7	7	7	7	7	7
56	7	7	7	7	7	7	7	7
18	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
32	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
20	7	7	7	7	7	7	7	7
30	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
30	7	7	7	7	7	7	7	7
25	7	7	7	7	7	7	7	7
7	7	7	7	7	7	7	7	7
28	7	7	7	7	7	7	7	7
36	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
27	7	7	7	7	7	7	7	7
21	7	7	7	7	7	7	7	7
24	7	7	7	7	7	7	7	7

26	7	7	7	7	7	7	7	7
10	7	7	7	7	7	7	7	7
16	7	7	7	7	7	7	7	7
27	7	7	7	7	7	7	7	7
16	7	7	7	7	7	7	7	7
63	7	7	7	7	7	7	7	7
14	7	7	7	7	7	7	7	7
28	7	7	7	7	7	7	7	7
27	7	7	7	7	7	7	7	7
41	7	7	7	7	7	7	7	7
12	7	7	7	7	7	7	7	7
21	7	7	7	7	7	7	7	7
37	7	7	7	7	7	7	7	7
20	7	7	7	7	7	7	7	7
14	7	7	7	7	7	7	7	7
18	7	7	7	7	7	7	7	7
43	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
31	7	7	7	7	7	7	7	7
24	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
14	7	7	7	7	7	7	7	7
28	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
20	7	7	7	7	7	7	7	7
27	7	7	7	7	7	7	7	7
39	7	7	7	7	7	7	7	7
32	7	7	7	7	7	7	7	7
20	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
16	7	7	7	7	7	7	7	7
21	7	7	7	7	7	7	7	7
30	7	7	7	7	7	7	7	7
31	7	7	7	7	7	7	7	7
20	7	7	7	7	7	7	7	7
16	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
21	7	7	7	7	7	7	7	7
27	7	7	7	7	7	7	7	7
7	7	7	7	7	7	7	7	7
37	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
67	7	7	7	7	7	7	7	7
16	7	7	7	7	7	7	7	7
19	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
14	7	7	7	7	7	7	7	7
42	7	7	7	7	7	7	7	7
21	7	7	7	7	7	7	7	7
22	7	7	7	7	7	7	7	7
30	7	7	7	7	7	7	7	7
20	7	7	7	7	7	7	7	7
38	7	7	7	7	7	7	7	7
10	7	7	7	7	7	7	7	7
16	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
7	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
57	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
25	7	7	7	7	7	7	7	7
35	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
27	7	7	7	7	7	7	7	7

32	7	7	7	7	7	7	7	7
20	7	7	7	7	7	7	7	7
25	7	7	7	7	7	7	7	7
23	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
45	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
24	7	7	7	7	7	7	7	7
8	7	7	7	7	7	7	7	7
8	7	7	7	7	7	7	7	7
24	7	7	7	7	7	7	7	7
38	7	7	7	7	7	7	7	7
18	7	7	7	7	7	7	7	7
16	7	7	7	7	7	7	7	7
53	7	7	7	7	7	7	7	7
26	7	7	7	7	7	7	7	7
27	7	7	7	7	7	7	7	7
34	7	7	7	7	7	7	7	7
33	7	7	7	7	7	7	7	7
23	7	7	7	7	7	7	7	7
20	7	7	7	7	7	7	7	7
29	7	7	7	7	7	7	7	7
53	7	7	7	7	7	7	7	7
24	7	7	7	7	7	7	7	7
16	7	7	7	7	7	7	7	7
23	7	7	7	7	7	7	7	7
23	7	7	7	7	7	7	7	7
27	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
34	7	7	7	7	7	7	7	7
12	7	7	7	7	7	7	7	7
28	7	7	7	7	7	7	7	7
12	7	7	7	7	7	7	7	7
19	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
10	7	7	7	7	7	7	7	7
21	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
36	7	7	7	7	7	7	7	7
14	7	7	7	7	7	7	7	7
28	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
14	7	7	7	7	7	7	7	7
10	7	7	7	7	7	7	7	7
36	7	7	7	7	7	7	7	7
29	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
34	7	7	7	7	7	7	7	7
18	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
20	7	7	7	7	7	7	7	7
51	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
35	7	7	7	7	7	7	7	7
33	7	7	7	7	7	7	7	7
23	7	7	7	7	7	7	7	7
30	7	7	7	7	7	7	7	7
28	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
14	7	7	7	7	7	7	7	7
7	7	7	7	7	7	7	7	7
29	7	7	7	7	7	7	7	7
35	7	7	7	7	7	7	7	7
8	7	7	7	7	7	7	7	7
21	7	7	7	7	7	7	7	7
26	7	7	7	7	7	7	7	7
23	7	7	7	7	7	7	7	7

17	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
27	7	7	7	7	7	7	7	7
16	7	7	7	7	7	7	7	7
20	7	7	7	7	7	7	7	7
8	7	7	7	7	7	7	7	7
36	7	7	7	7	7	7	7	7
32	7	7	7	7	7	7	7	7
33	7	7	7	7	7	7	7	7
16	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
16	7	7	7	7	7	7	7	7
37	7	7	7	7	7	7	7	7
7	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
44	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
28	7	7	7	7	7	7	7	7
12	7	7	7	7	7	7	7	7
21	7	7	7	7	7	7	7	7
30	7	7	7	7	7	7	7	7
37	7	7	7	7	7	7	7	7
16	7	7	7	7	7	7	7	7
20	7	7	7	7	7	7	7	7
31	7	7	7	7	7	7	7	7
19	7	7	7	7	7	7	7	7
32	7	7	7	7	7	7	7	7
10	7	7	7	7	7	7	7	7
21	7	7	7	7	7	7	7	7
25	7	7	7	7	7	7	7	7
30	7	7	7	7	7	7	7	7
23	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
12	7	7	7	7	7	7	7	7
14	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
48	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
16	7	7	7	7	7	7	7	7
30	7	7	7	7	7	7	7	7
27	7	7	7	7	7	7	7	7
55	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
46	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
21	7	7	7	7	7	7	7	7
21	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
35	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
18	7	7	7	7	7	7	7	7
27	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
7	7	7	7	7	7	7	7	7
22	7	7	7	7	7	7	7	7
21	7	7	7	7	7	7	7	7
19	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
10	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
7	7	7	7	7	7	7	7	7
19	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7

44	7	7	7	7	7	7	7	7
14	7	7	7	7	7	7	7	7
16	7	7	7	7	7	7	7	7
16	7	7	7	7	7	7	7	7
12	7	7	7	7	7	7	7	7
34	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
16	7	7	7	7	7	7	7	7
41	7	7	7	7	7	7	7	7
18	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
19	7	7	7	7	7	7	7	7
19	7	7	7	7	7	7	7	7
24	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
33	7	7	7	7	7	7	7	7
38	7	7	7	7	7	7	7	7
12	7	7	7	7	7	7	7	7
20	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
10	7	7	7	7	7	7	7	7
57	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
33	7	7	7	7	7	7	7	7
21	7	7	7	7	7	7	7	7
41	7	7	7	7	7	7	7	7
18	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
21	7	7	7	7	7	7	7	7
27	7	7	7	7	7	7	7	7
7	7	7	7	7	7	7	7	7
28	7	7	7	7	7	7	7	7
66	7	7	7	7	7	7	7	7
45	7	7	7	7	7	7	7	7
48	7	7	7	7	7	7	7	7
27	7	7	7	7	7	7	7	7
30	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
23	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
19	7	7	7	7	7	7	7	7
18	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
32	7	7	7	7	7	7	7	7
19	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
19	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
14	7	7	7	7	7	7	7	7
30	7	7	7	7	7	7	7	7
39	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
28	7	7	7	7	7	7	7	7
12	7	7	7	7	7	7	7	7
28	7	7	7	7	7	7	7	7
20	7	7	7	7	7	7	7	7
34	7	7	7	7	7	7	7	7
16	7	7	7	7	7	7	7	7
22	7	7	7	7	7	7	7	7
37	7	7	7	7	7	7	7	7
21	7	7	7	7	7	7	7	7

51	7	7	7	7	7	7	7	7
22	7	7	7	7	7	7	7	7
25	7	7	7	7	7	7	7	7
23	7	7	7	7	7	7	7	7
19	7	7	7	7	7	7	7	7
22	7	7	7	7	7	7	7	7
45	7	7	7	7	7	7	7	7
24	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
10	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
50	7	7	7	7	7	7	7	7
33	7	7	7	7	7	7	7	7
44	7	7	7	7	7	7	7	7
35	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
58	7	7	7	7	7	7	7	7
18	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
35	7	7	7	7	7	7	7	7
24	7	7	7	7	7	7	7	7
12	7	7	7	7	7	7	7	7
8	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
7	7	7	7	7	7	7	7	7
8	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
23	7	7	7	7	7	7	7	7
24	7	7	7	7	7	7	7	7
37	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
23	7	7	7	7	7	7	7	7
28	7	7	7	7	7	7	7	7
19	7	7	7	7	7	7	7	7
34	7	7	7	7	7	7	7	7
10	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
16	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
21	7	7	7	7	7	7	7	7
10	7	7	7	7	7	7	7	7
35	7	7	7	7	7	7	7	7
7	7	7	7	7	7	7	7	7
28	7	7	7	7	7	7	7	7
42	7	7	7	7	7	7	7	7
27	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
8	7	7	7	7	7	7	7	7
24	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
16	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
32	7	7	7	7	7	7	7	7
50	7	7	7	7	7	7	7	7
16	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
26	7	7	7	7	7	7	7	7
31	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
27	7	7	7	7	7	7	7	7
42	7	7	7	7	7	7	7	7
76	7	7	7	7	7	7	7	7

9	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
34	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
18	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
21	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
12	7	7	7	7	7	7	7	7
14	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
7	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
24	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
32	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
10	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
27	7	7	7	7	7	7	7	7
41	7	7	7	7	7	7	7	7
25	7	7	7	7	7	7	7	7
25	7	7	7	7	7	7	7	7
25	7	7	7	7	7	7	7	7
23	7	7	7	7	7	7	7	7
28	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
19	7	7	7	7	7	7	7	7
18	7	7	7	7	7	7	7	7
25	7	7	7	7	7	7	7	7
10	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
16	7	7	7	7	7	7	7	7
22	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
14	7	7	7	7	7	7	7	7
47	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
30	7	7	7	7	7	7	7	7
32	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
27	7	7	7	7	7	7	7	7
22	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
24	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
14	7	7	7	7	7	7	7	7
18	7	7	7	7	7	7	7	7
35	7	7	7	7	7	7	7	7
22	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
21	7	7	7	7	7	7	7	7
19	7	7	7	7	7	7	7	7
25	7	7	7	7	7	7	7	7
20	7	7	7	7	7	7	7	7
22	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
37	7	7	7	7	7	7	7	7
31	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
19	7	7	7	7	7	7	7	7
12	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
16	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
40	7	7	7	7	7	7	7	7

19	7	7	7	7	7	7	7	7
32	7	7	7	7	7	7	7	7
10	7	7	7	7	7	7	7	7
34	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
19	7	7	7	7	7	7	7	7
24	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
8	7	7	7	7	7	7	7	7
21	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
29	7	7	7	7	7	7	7	7
14	7	7	7	7	7	7	7	7
24	7	7	7	7	7	7	7	7
28	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
31	7	7	7	7	7	7	7	7
28	7	7	7	7	7	7	7	7
24	7	7	7	7	7	7	7	7
20	7	7	7	7	7	7	7	7
12	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
12	7	7	7	7	7	7	7	7
26	7	7	7	7	7	7	7	7
33	7	7	7	7	7	7	7	7
24	7	7	7	7	7	7	7	7
19	7	7	7	7	7	7	7	7
24	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
26	7	7	7	7	7	7	7	7
18	7	7	7	7	7	7	7	7
24	7	7	7	7	7	7	7	7
26	7	7	7	7	7	7	7	7
22	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
12	7	7	7	7	7	7	7	7
20	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
31	7	7	7	7	7	7	7	7
30	7	7	7	7	7	7	7	7
29	7	7	7	7	7	7	7	7
38	7	7	7	7	7	7	7	7
12	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
30	7	7	7	7	7	7	7	7
19	7	7	7	7	7	7	7	7
12	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
27	7	7	7	7	7	7	7	7
27	7	7	7	7	7	7	7	7
18	7	7	7	7	7	7	7	7
38	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
20	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
24	7	7	7	7	7	7	7	7
12	7	7	7	7	7	7	7	7
14	7	7	7	7	7	7	7	7
19	7	7	7	7	7	7	7	7

15	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
22	7	7	7	7	7	7	7	7
24	7	7	7	7	7	7	7	7
18	7	7	7	7	7	7	7	7
37	7	7	7	7	7	7	7	7
35	7	7	7	7	7	7	7	7
25	7	7	7	7	7	7	7	7
30	7	7	7	7	7	7	7	7
57	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
25	7	7	7	7	7	7	7	7
50	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
35	7	7	7	7	7	7	7	7
26	7	7	7	7	7	7	7	7
14	7	7	7	7	7	7	7	7
33	7	7	7	7	7	7	7	7
18	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
21	7	7	7	7	7	7	7	7
39	7	7	7	7	7	7	7	7
33	7	7	7	7	7	7	7	7
31	7	7	7	7	7	7	7	7
20	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
12	7	7	7	7	7	7	7	7
38	7	7	7	7	7	7	7	7
19	7	7	7	7	7	7	7	7
19	7	7	7	7	7	7	7	7
16	7	7	7	7	7	7	7	7
21	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
22	7	7	7	7	7	7	7	7
29	7	7	7	7	7	7	7	7
48	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
41	7	7	7	7	7	7	7	7
29	7	7	7	7	7	7	7	7
33	7	7	7	7	7	7	7	7
16	7	7	7	7	7	7	7	7
14	7	7	7	7	7	7	7	7
19	7	7	7	7	7	7	7	7
23	7	7	7	7	7	7	7	7
10	7	7	7	7	7	7	7	7
19	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
18	7	7	7	7	7	7	7	7
25	7	7	7	7	7	7	7	7
62	7	7	7	7	7	7	7	7
21	7	7	7	7	7	7	7	7
32	7	7	7	7	7	7	7	7
21	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
19	7	7	7	7	7	7	7	7
23	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
35	7	7	7	7	7	7	7	7
23	7	7	7	7	7	7	7	7
19	7	7	7	7	7	7	7	7
44	7	7	7	7	7	7	7	7
21	7	7	7	7	7	7	7	7
18	7	7	7	7	7	7	7	7

29	7	7	7	7	7	7	7	7
21	7	7	7	7	7	7	7	7
16	7	7	7	7	7	7	7	7
7	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
12	7	7	7	7	7	7	7	7
23	7	7	7	7	7	7	7	7
8	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
21	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
7	7	7	7	7	7	7	7	7
25	7	7	7	7	7	7	7	7
27	7	7	7	7	7	7	7	7
36	7	7	7	7	7	7	7	7
18	7	7	7	7	7	7	7	7
44	7	7	7	7	7	7	7	7
19	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
23	7	7	7	7	7	7	7	7
18	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
39	7	7	7	7	7	7	7	7
18	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
22	7	7	7	7	7	7	7	7
21	7	7	7	7	7	7	7	7
61	7	7	7	7	7	7	7	7
24	7	7	7	7	7	7	7	7
14	7	7	7	7	7	7	7	7
10	7	7	7	7	7	7	7	7
8	7	7	7	7	7	7	7	7
7	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
20	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
24	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
23	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
19	7	7	7	7	7	7	7	7
20	7	7	7	7	7	7	7	7
26	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
18	7	7	7	7	7	7	7	7
25	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
16	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
36	7	7	7	7	7	7	7	7
45	7	7	7	7	7	7	7	7
29	7	7	7	7	7	7	7	7
19	7	7	7	7	7	7	7	7
12	7	7	7	7	7	7	7	7
23	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
14	7	7	7	7	7	7	7	7
10	7	7	7	7	7	7	7	7
23	7	7	7	7	7	7	7	7
33	7	7	7	7	7	7	7	7
21	7	7	7	7	7	7	7	7
23	7	7	7	7	7	7	7	7
23	7	7	7	7	7	7	7	7
16	7	7	7	7	7	7	7	7
10	7	7	7	7	7	7	7	7
24	7	7	7	7	7	7	7	7

13	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
16	7	7	7	7	7	7	7	7
12	7	7	7	7	7	7	7	7
42	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
21	7	7	7	7	7	7	7	7
31	7	7	7	7	7	7	7	7
14	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
25	7	7	7	7	7	7	7	7
35	7	7	7	7	7	7	7	7
21	7	7	7	7	7	7	7	7
32	7	7	7	7	7	7	7	7
23	7	7	7	7	7	7	7	7
50	7	7	7	7	7	7	7	7
27	7	7	7	7	7	7	7	7
14	7	7	7	7	7	7	7	7
19	7	7	7	7	7	7	7	7
29	7	7	7	7	7	7	7	7
31	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
48	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
29	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
35	7	7	7	7	7	7	7	7
20	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
12	7	7	7	7	7	7	7	7
39	7	7	7	7	7	7	7	7
48	7	7	7	7	7	7	7	7
29	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
35	7	7	7	7	7	7	7	7
22	7	7	7	7	7	7	7	7
43	7	7	7	7	7	7	7	7
28	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
43	7	7	7	7	7	7	7	7
29	7	7	7	7	7	7	7	7
45	7	7	7	7	7	7	7	7
25	7	7	7	7	7	7	7	7
19	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
30	7	7	7	7	7	7	7	7
23	7	7	7	7	7	7	7	7
34	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
20	7	7	7	7	7	7	7	7
28	7	7	7	7	7	7	7	7
25	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
22	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
36	7	7	7	7	7	7	7	7
10	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
28	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
23	7	7	7	7	7	7	7	7

19	7	7	7	7	7	7	7	7
25	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
30	7	7	7	7	7	7	7	7
16	7	7	7	7	7	7	7	7
27	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
8	7	7	7	7	7	7	7	7
21	7	7	7	7	7	7	7	7
20	7	7	7	7	7	7	7	7
14	7	7	7	7	7	7	7	7
10	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
39	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
40	7	7	7	7	7	7	7	7
38	7	7	7	7	7	7	7	7
24	7	7	7	7	7	7	7	7
10	7	7	7	7	7	7	7	7
32	7	7	7	7	7	7	7	7
55	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
7	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
30	7	7	7	7	7	7	7	7
10	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
18	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
25	7	7	7	7	7	7	7	7
18	7	7	7	7	7	7	7	7
21	7	7	7	7	7	7	7	7
27	7	7	7	7	7	7	7	7
18	7	7	7	7	7	7	7	7
22	7	7	7	7	7	7	7	7
16	7	7	7	7	7	7	7	7
22	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
20	7	7	7	7	7	7	7	7
37	7	7	7	7	7	7	7	7
21	7	7	7	7	7	7	7	7
24	7	7	7	7	7	7	7	7
10	7	7	7	7	7	7	7	7
20	7	7	7	7	7	7	7	7
21	7	7	7	7	7	7	7	7
7	7	7	7	7	7	7	7	7
22	7	7	7	7	7	7	7	7
21	7	7	7	7	7	7	7	7
29	7	7	7	7	7	7	7	7
14	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
35	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
45	7	7	7	7	7	7	7	7
18	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
13	7	7	7	7	7	7	7	7
42	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
22	7	7	7	7	7	7	7	7
20	7	7	7	7	7	7	7	7
31	7	7	7	7	7	7	7	7
34	7	7	7	7	7	7	7	7
16	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
19	7	7	7	7	7	7	7	7

12	7	7	7	7	7	7	7	7
16	7	7	7	7	7	7	7	7
24	7	7	7	7	7	7	7	7
7	7	7	7	7	7	7	7	7
14	7	7	7	7	7	7	7	7
30	7	7	7	7	7	7	7	7
56	7	7	7	7	7	7	7	7
16	7	7	7	7	7	7	7	7
12	7	7	7	7	7	7	7	7
28	7	7	7	7	7	7	7	7
23	7	7	7	7	7	7	7	7
29	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
46	7	7	7	7	7	7	7	7
32	7	7	7	7	7	7	7	7
18	7	7	7	7	7	7	7	7
22	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
19	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
27	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
26	7	7	7	7	7	7	7	7
48	7	7	7	7	7	7	7	7
31	7	7	7	7	7	7	7	7
21	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
15	7	7	7	7	7	7	7	7
31	7	7	7	7	7	7	7	7
19	7	7	7	7	7	7	7	7
42	7	7	7	7	7	7	7	7
10	7	7	7	7	7	7	7	7
19	7	7	7	7	7	7	7	7
9	7	7	7	7	7	7	7	7
22	7	7	7	7	7	7	7	7
10	7	7	7	7	7	7	7	7
25	7	7	7	7	7	7	7	7
18	7	7	7	7	7	7	7	7
21	7	7	7	7	7	7	7	7
17	7	7	7	7	7	7	7	7
21	7	7	7	7	7	7	7	7
19	7	7	7	7	7	7	7	7
25	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7
24	7	7	7	7	7	7	7	7
8	7	7	7	7	7	7	7	7
21	7	7	7	7	7	7	7	7
25	7	7	7	7	7	7	7	7
26	7	7	7	7	7	7	7	7
11	7	7	7	7	7	7	7	7