3. Plumbing (Sequence)

"Lights-Camera-Action"

If I am to successfully do something following a set of instructions, I must know two different kinds of things. I must of course know what the individual instructions are. Less obviously I must also know what order I must follow them in. Algorithms, which are just written-out sets of instructions to do something, therefore consist of two things:

1. the **actions** that must be taken, and
2. the **order** they must be done in.

If movies about movies are to believed, when a film is being made, the director shouts "Lights-Cameras-Action" to start a new take. They are giving instructions to the film crew. First switch on the lights to light up the set, then start filming, then and only then should the actors and actresses start acting. The crew are not only being told what to do, they are being told the order to do it in. If the actors started acting before the lights were on, or the cameras were rolling then part of their potentially Oscar-winning performance would be missed! The order that the instructions should be followed is just as important as the things to be done.

When I was learning to drive, I was taught the mantra "Mirrors, Signal, Manoeuvre". What did this mean? Whenever I was about to make a turn, overtake, or in fact do anything other than follow the road ahead, I should first check my mirrors, then signal, then finally do whatever manoeuvre I was intending. To start with I tended to signal then check my mirrors then manoeuvre, for which my driving instructor gave me lots of grief. Sometimes I would start to manoeuvre and only then check my mirrors and signal. Had I still been doing it in the wrong order when I took the test I would have failed. Why? Because the order matters. It is safer if you check what is behind you and what its doing first and only then indicate your intention, and only once you have done that start to do something yourself. Test Inspectors are therefore finicky about whether you were following the algorithm properly – not only doing all the steps but doing them in the correct **order**. (As it happens I did fail my first test but not for failing to follow an algorithm, but because the instructions I was given were not precise enough. The examiner asked me to stop somewhere convenient on the left as we were going up a hill. I decided the most convenient place was away from the hill as hill-starts are a nightmare – unfortunately she wanted me to stop specifically so she could see me do a hill start. After that I hadn't a hope of passing – all because the instructions I was given were ambiguous.)

We will now spend some time looking at what we mean by the order things are done in. This is the plumbing of the algorithm: how are the instructions connected together. It turns out that only three kinds of plumbing are needed to write any algorithm. These are known as **sequence**, **selection** and **iteration**. A style of programming known as **Structured Programming** involves writing instructions only using these three forms of plumbing. It is as though a plumber only had some fixed shaped pipes with common connections, rather than being able to bend pipes into any shape. Sticking to a few well-thought out forms of connection helps keep things neat and tidy and thus easier to understand. It prevents the plumber from connecting pipes so as to look like

spaghetti. Imagine trying to work out which pipes in the attic were connected to the mains if there were pipes going everywhere. It would be a little like solving one of those children's puzzles where 3 lines are intertwined, and you have to work out which one connects the picture of the dog to which person holding the end of a leash. The fact that it is a puzzle means it is intentionally harder than it need be. In fact that is exactly the problem I have trying to work out which plug is connected to the hairdryer and which to my alarm clock the cables of which are usually in a tangled mess, so that I do not unplug the wrong thing.

We will look at sequence here. We will look at selection and iteration in more detail in later chapters.

Most simple algorithms found in everyday life are short and involve doing a series of things one after another. Many of the algorithms we have looked at so far are like this. The algorithm is given as a list of instructions. Given a series of instructions we need to know what order they are to be followed in. A common way to indicate this is to number the steps. For example we saw earlier the instructions for opening the emergency door on an airliner:

Emergency opening

1. Pull cover aside
2. Push lever to open position and release.
3. Push door outwards.

The first line of this is not part of the algorithm, but rather the "title". It tells us what following this algorithm will achieve: it will lead to the door being open. The actual algorithm consists of three lines. Each is an instruction to do something, the numbering tells us the order to do them in. First we must pull the cover aside, then push the lever and release it and finally push the door outwards. If we try to do the instructions in a different order to that given we will not manage to open the door and die a horribly painful death. If we miss a step out then we also will fail to open the door.

The instructions to open the door on some old-fashioned high-speed trains that require you to lean out of the window to open the door are similar:

1. Wait until light comes on.
2. Open Window.
3. Lean out of window and turn outside handle.

Again numbers are used to emphasise the order the instructions are to be followed in. However, they are also placed one after another on consecutive lines, so even without the numbers most people would follow the instructions in the intended order.

Numbers might similarly be used on a ticket machine, perhaps for buying rail tickets:

1. Select destination.
2. Select ticket type.
3. Insert correct money.
4. Take change.
5. Take ticket.

A well-designed machine might allow several different orderings. For example, it might allow you to select the ticket type first and then the destination ("I want a return to Cardiff please"). That just means it allows several different algorithms to be followed. There are often many algorithms that achieve the same result. Such

flexibility is needed because humans are not that good at following algorithms: “I am a human not an Automaton”.

The following instructions given by the BMA (BMA, 1990) to prepare for an emergency childbirth. It is a sequence of actions to take, specifying both what should be done and the order it should be done in.

1. Summon medical help.
2. Reassure the mother, staying clam yourself.
3. Wash hands and scrub nails under running water without drying them.
4. Make sure everything you will use is clean.
5. Prepare a flat surface using a clean sheet.
6. Prop the mother up with pillows, with legs bent and apart and feet flat..

Numbers for steps are not always used to indicate the order actions should be done in. All that is needed is a convention. The most common convention in programs and in everyday life is that we follow the instructions, starting at the top and working down the list, doing the instructions in the order given (unless of course we are from a culture such as Chinese where writing is not left to right, top to bottom). For example the instructions for using a tube of glue might go something like:

Ensure both surfaces are clean and free of grease.
Apply a thin coat of glue to both surfaces.
Wait for 5 minutes.
Push the two surfaces together.
Leave for 30 minutes.

This is a **sequence** of instructions to be followed in order. The next instruction is only started on completion of the one before. The order the instructions are to be followed is just the order they appear on the page – top to bottom. Programs often use the same convention. Start with the instruction at the top of the program and follow them one at a time until you reach the bottom.

The 17th century Scientist, Sir Isaac Newton was a great lover of algorithms. In part his meticulousness over instructions was what made him a great scientist. When doing scientific experiments he was very careful to write down the steps he followed precisely – so that others could then repeat his experiments and so replicate his results. This repeatability of experiments is the backbone of the scientific method, though it was unusual then. His love of such recipes grew as a child. He collected recipes for making various brews and potions. Michael White’s biography of him quotes an early recipe that he wrote down in his notebook in 1659 for making a crimson dye (Newton 1659)

“Take some of the clearest blood of a sheep &
put it into a bladder &
with a needle prick holes into the bottom of it then
hang it up to dry in the sun &
dissolve it in alum water according as you have need”

This slightly ghastly algorithm is just a sequence of instructions to be followed one after the other. Newton used the “&” symbol to indicate where one instruction ends and the next starts. In the language he was using to write his recipes “&” means something like “and then do” – it indicates sequencing. The instructions are to be done one after another. Notice though at one point he switches to using “then” to

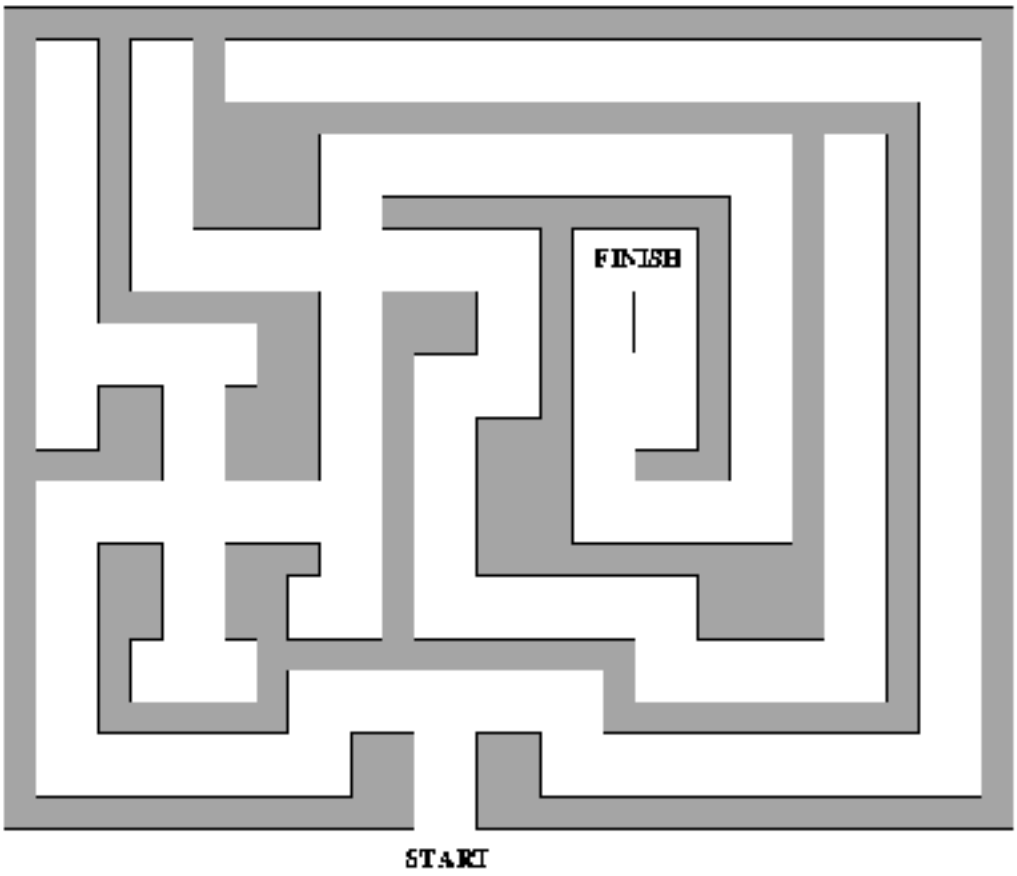
mean the same thing – not as meticulous as he ought to have been! Programming languages also usually have a symbol used to indicate when one instruction in a sequence ends and the next starts. For example a semicolon is often used for this.

The scripts of plays (and film scripts) are, in a simple sense, algorithms. They are a series of instructions that the actors are supposed to follow – either movements to make or things to say. The same things should be done and said in the 1000th performance as in the first (just as with a computer programme, it should do the same on the 1000th time it is run as the first). A script is a very simple form of algorithm as it uses only one method of giving the order of actions – that of doing things one after another. For example, Macbeth starts with the following sequence of actions:

Thunder and Lightning.
Enter three witches.
Witch 1: When shall we three meet again, In thunder, Lightning or in rain?
Witch 2: When the hurlyburly's done, When the Battle's lost and won.
etc.
Witches vanish.

Shakespeare was an early programmer!

When solving pencil and paper mazes like the one below, children normally draw a pencil line to show the solution. Have a go at solving this one.



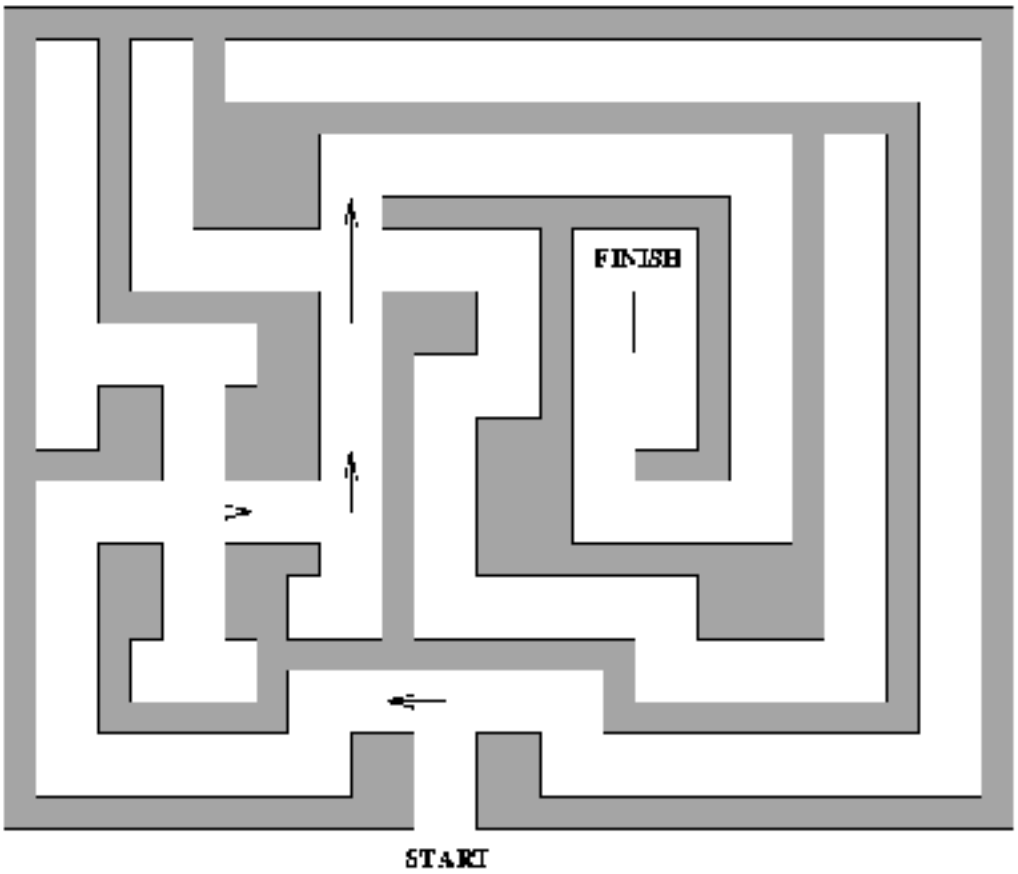
However, now suppose it was the plan of a real maze made of hedges, and you were studying the plan for a race to the centre. During the race you cannot take your plan. Once you have worked out the route to the centre of a maze, you must memorise an

algorithm that you must follow to get to the centre. What you would need to do is work out what to do at each junction. Write an algorithm using the following actions:

Enter the maze by the gate following the path ahead.
Turn left following the path ahead.
Turn right following the path ahead.
Go straight across the junction following the path ahead.
Stop at the cross.

You must put the actions in a sequence (not necessarily in the order given) so that if someone followed them they would get to the centre of the maze. We assume that anyone following the instructions will stop and look for the next instruction whenever they get to a junction.You can use each of the above actions as many times as you like in the algorithm and you may not need to use some at all. Your algorithm should be the quickest route. Try this before reading on.

The following is the same maze with arrows drawn to show what to do at each junction to make it easier.



Here is my answer using the actions given.

To get to the centre of the maze:

1. Enter the maze by the gate following the path ahead.
2. Turn left following the path ahead.
3. Go straight across the junction following the path ahead.
4. Turn left following the path ahead.
5. Go straight across the junction following the path ahead.

6. Stop at the cross.

If you can memorise this algorithm then you stand a chance of winning the race to the centre as you will take the most direct route. Taking these instructions would also almost certainly be easier than taking your plan with you and trying to work out where you are from that. With an algorithm you can follow the instructions blindly without thinking.

Here is another algorithmic puzzle that is a variation of one invented by Alcuin who was a medieval mathematician born in the 8th century (Stewart, 1992).

> A man has been to the market and bought a new hen and a sack of corn. He also has with him his Pit Bull Terrier. To get home however he must cross a river in a corracle (a very small boat). This was not a problem on his way out, but when he gets to the river he realises that the corracle is only large enough to take him and one of the sack, the hen and the dog at once. This would not matter except that if he leaves the hen and the corn alone, the hen will eat the corn. If he leaves the Pit Bull and the hen alone, the Pit Bull will kill the hen. How does the farmer get them all safely across the river in the corracle?

This puzzle illustrates sequencing because to solve the puzzle you need to come up with a sequence of river crossings that the farmer can do sometimes crossing on his own, sometimes with one of the corn, hen and dog. You need to come up with an algorithm that the farmer can follow. Try and work out a solution and write down the algorithm as a sequence of instructions before you carry on reading. You will probably find it easier if you use different counters (eg coins) for each of the farmer, corn, hen and dog.

Here is one solution algorithm:

To cross the river:

1. The farmer crosses with the hen.
2. The farmer returns alone.
3. The farmer crosses with the corn.
4. The farmer returns with the hen.
5. The farmer crosses with the dog.
6. The farmer returns alone.
7. The farmer crosses with the hen.

This algorithm would be easier to understand if we added some explanation. Why for example does the hen cross three times? The following version has comments in italics as explanation.

To cross the river:

1. The farmer crosses with the hen.
   As otherwise the dog will be left with the hen and eat it OR the hen will be left with the corn and eat it.
2. The farmer returns alone.
3. The farmer crosses with the corn.
4. The farmer returns with the hen.
   As otherwise the hen will be left with the corn and eat it.
5. The farmer crosses with the dog.

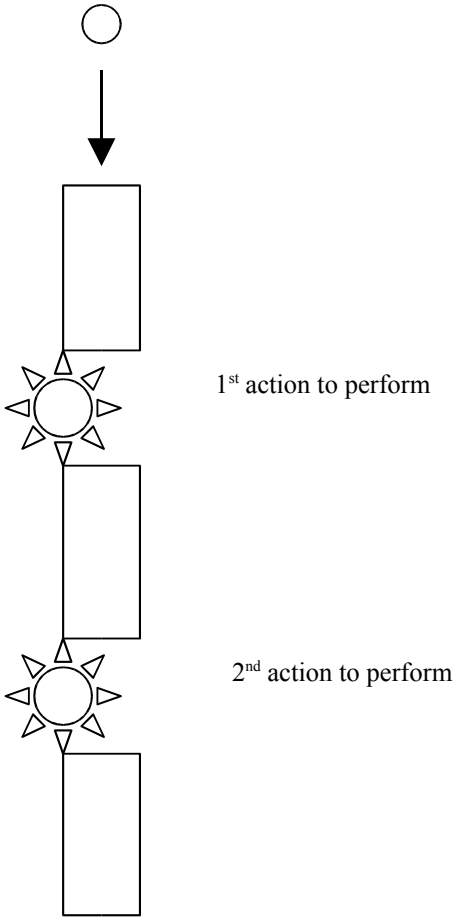
The dog is left with the corn, which is not a problem.

6. The farmer returns alone.
7. The farmer crosses with the hen.

The extra explanations are not part of the algorithm (they are not instructions to take actions) – they just explain the algorithm, helping the person following the algorithm to understand it. Even with the first version the farmer could blindly follow the instructions without understanding why he was taking each step and still get the job done. We will return to the importance of such **comments** later.

The algorithm given above is not the only one that would work. In fact there are lots, but only one other one that is as fast. All the others involve the farmer crossing the river more times than necessary. Work out and write down the other algorithm that is as fast: it is a sequence of 7 steps (that is actions) too but some of the steps are different.

These examples use the **sequence** form of plumbing (i.e. connection). The separate actions that must be performed are connected together in a line. By following the plumbing you do the actions in the correct order.



Imagine a children's marble construction toy consisting of a series of wheels connected together by pipes as in the above picture. Suppose the parts of our construction kit are each a single pipe connected to a wheel. Each pipe-wheel part can be connected to another pipe-wheel part in a tower. Put a marble in the top pipe and the first wheel will turn. The marble then passes on. The first wheel stops and the second wheel turns. The marble passes out the bottom and everything stops. Only one wheel turns at a time and they do so in order from top to bottom of the tower. Imagine each wheel is an action to be performed and the marble run represents an algorithm. The actions are executed when a marble arrives and only finish when the marble moves on. We do not need to just connect two wheel-pipe parts. We can fasten any number together.

In computer terms, a form of plumbing is known as a **control structure**. The control structure here is the **sequencing** control structure. The plumbing ensures the wheels spin one after another in a fixed order from top to bottom. The marble represents something known as the **flow of control**. It moves from wheel to wheel, its position determining which wheel is active. In a program the flow of control moves from one instruction to the next, each instruction being executed only when the flow of control reaches it.

Another way of thinking of the flow of control is that it is like a baton being passed in a relay race. Only one person holds the baton at once. All the other members of the team stand around waiting for their turn. Think of the action commands in the relay as the individual runners. Only one action is being followed at a time just as only one team member is running at a time. Once that action is completed, "the baton" i.e. the flow of control is passed to the next action, allowing it to be followed. Once it is completed it too passes control to the next action. Sequencing is like a relay on a long straight track, with the runners lined up in a straight line.

Programming is not just about working out the actions to be performed but also requires the order to be given – that is the plumbing must be indicated.

In our Imp computer we have so far glossed over how the Imps know when to do a particular action – we have only looked at the storage Imps rather than the Instruction Imps themselves. Let us return to an earlier example of instructions to swap round the values two Imps were storing. The three instructions required were:

Trevor gets Alf's value.
Alf gets Bridgit's value.
Bridgit gets Trevor's value.

To do the swap they had to be followed in the order given. Try following them in a different order to see what happens!

How do we ensure they are followed in the correct order? We introduce the Instruction Imps. These are a bunch of nameless, faceless middle-manager Imps. Each such Imp is responsible for a given instruction. For example one would be given the instruction "Trevor gets Alf's value". It would be her job to tell Trevor and Alf to do this at the appropriate time. Similarly another Imp would be responsible for the instruction "Alf gets Bridgit's value." and a third for the last instruction.

At compile time the upper management would tell these three nameless Imps what there instruction today was, and also tell them who came next. At compile time all the storage Imps would also be gathered together according to the declarations. They would all then sit around doing nothing until "run time". Run time would start when the person operating the computer pressed the appropriate button. This would indicate to the Chief Executive Imp (normally referred to in hushed tones by other Imps as 'M') of the computer that action was needed. She would then take a special baton out of its box and pass it to the Instruction Imp who had been given the first instruction to follow. She would then go back to practising her putting whilst she waited for the baton to return. As the instruction of each instruction Imp was completed they would pass the baton on to the next Imp. A final instruction Imp, holding the instruction "return" would eventually get the baton, and return it to 'M'. She would put it back in its box, open the hatch in the roof of her office and shout to the computer operator, that the program had terminated.

Lets look at what happens when we execute our "swap" program. We assume Alf and Bridget have been already initialised with the values that are to be swapped (by some earlier instructions). Note also that we have added an extra instruction on the end to Return the baton to 'M':

Trevor gets Alf's value.
Alf gets Bridgit's value.
Bridgit gets Alf's value.
Return.

The baton is passed to the Instruction Imp with the first instruction. On getting the baton he knows it is his turn to do some work. He pulls out the instruction he holds and follows it. It tells him that he must get a copy of Alf's value and pass it to Trevor. When this is done he passes on the baton to the next Instruction Imp – who holds the second instruction. The first Instruction Imp goes back to his room and goes to sleep – his job is over for the day. The second instruction Imp now has the baton so must do some work. She follows her instruction, which is to get a copy of Bridgit's value and give it to Alf, then pass the baton on. She returns to the Pub. The third instruction Imp now has the baton so copies Trevor's new value to Bridgit and passes on the baton and returns to building a giant model of Leaning Tower of Pisa out of matches. The final Imp who has the baton has one job only, to return the baton to 'M' to indicate that the program has completed. Each Instruction Imp knows who to pass the baton to next because they were told it during the compilation process. Which it is is just determined by the order of the instructions down the page. Because in this case we are using sequencing, each time the program was run, the Imp passes the baton to the same Imp – there is no choice. In a later chapter we will look at situations where there is a choice of where the baton is passed next.

We saw in the introduction how the solution to a solitaire peg-jumping puzzle was an algorithm. Look at it again and you will see our answer was just a sequence of actions to perform. The traditional game of peg solitaire where a cross shaped board starts full of pegs, apart from an empty hole in the centre, and ends with a single peg in the centre is similar. Its solution is a sequence of jump instructions. If you have a solitaire board, you may wish to see if you can work it out. Solitaire is difficult, however. Here is another solitaire puzzle given by Ian Stewart (Stewart, 1991). It is played on a

triangular board, with 10 pieces in the central triangle, but the outer edges empty. The very central piece is red. Any piece can jump over any other into the space directly beyond, with the jumped piece removed as in normal solitaire. The aim is to end with the red piece in the centre and all other pieces removed. Work out and write down the algorithm to do this.

Here is one solution. Follow it to check it works.

1. Jump piece 13 to 11.
2. Jump piece 5 to 12.
3. Jump piece 11 to 13.
4. Jump piece 13 to 6.
5. Jump piece 20 to 9.
6. Jump piece 6 to 13.
7. Jump piece 13 to 26.
8. Jump piece 17 to 19.
9. Jump piece 26 to 13.

Here is a variation that has slightly more complicated rules but with the same start and end position. As before any piece can jump any other into the space beyond. However, the jumped piece is only removed if it was jumped by the central red piece – the King. The above algorithm no longer works as the actions, though superficially similar now have a different effect – they have a different **semantics**. The semantics of the sequencing operation is unchanged, however. You need a new algorithm – a new sequence of moves. See if you can work one out (you may find this harder than the last puzzle – the algorithm needs to be slightly longer). HINT: remember other pieces can still jump the King.

Here is one solution algorithm. Follow it to see if it works.

1. Jump piece 8 to 10.
2. Jump piece 13 to 6 (capture).
3. Jump piece 6 to 15 (capture).
4. Jump piece 15 to 26 (capture).
5. Jump piece 26 to 13 (capture).
6. Jump piece 18 to 9.
7. Jump piece 17 to 8.
8. Jump piece 13 to 11 (capture).
9. Jump piece 5 to 12.
10. Jump piece 11 to 13 (capture).
11. Jump piece 14 to 5.
12. Jump piece 13 to 6 (capture).
13. Jump piece 6 to 4 (capture).
14. Jump piece 4 to 13 (capture).