Algorithms and labyrinths

A. Belov-Kanel, I. Gazhur, I. Ivanov-Pogodaev, A. Malistov

Section A. This is an introductory section. These supporting problems will be removed after intermediate finish.

In the first problems of this section we will examine a standard finite automatic machine: a robot moving on a square grid. The robot can do some actions defined by his program. These actions are:

1. Move forward (to the next cell);
2. Rotate in the current cell;
3. Drop a flag into the current cell (if the robot has a flag);
4. Look for a flag in the current cell;
5. Pick up a flag from the current cell.

Every problem defines how many flags the robot has. The program consists of numerated list with instructions for the robot. Some of these instructions are jumps to another instructions (like GOTO operator in computer languages). We will discuss the opportunities of walking through some sets of cells. By *walking through* or *investigating* the set of cells we mean that the robot using his program can walk through *every* cell in this set.

First of all we should examine some of the robot's skills

Example. Let us prove that the robot with two flags can explore the tape — an infinite tape with width 1. Let the robot be in the cell 0 with two flags. Let us write the robot's program.

1. Drop a flag.
2. Go to cell 1.
3. Drop a flag.

Now we are going to organize a shuttle-moving of the robot between these two flags. Besides, the robot will increase the distance between the flags by «pushing» them. We can obtain this by using the program below:

4. Turn around. (turning at 180 degrees)
5. Move forward.
6. Look for a flag.
7. If there is no flag then go to 5.
8. Pick up the flag and move forward.
9. Drop flag and go to 4.

It is easy to see that the robot will explore all the cells on the tape.

Could the robot explore the tape without any flags? The answer is «no». In order to prove this we should formalize construction of a robot. It is easy to see that every robot's action depends on the number of the current instruction and the presence of a flag in the current cell. Let us call these factors by *the robot's inner condition*. Obviously, there is a finite number of various inner conditions of the robot. In most cases, the robot changes his inner condition after an action. According to the Dirihlet Principle, some inner condition of the robot will happen again. So, let it take t seconds. The robot has moved right for a cells. It is easy to see that in another t seconds the robot will also move right for another a cells. It is clear that the robot cannot reach some cells placed to the left of his starting position.

A0. *Prove that the robot with one flag cannot explore the tape.*

Hint. You should consider two cases: 1) the robot does not go away from his flag at the distance more then some `N`;
2) the robot goes away from his flag at any distance.

A1. *Prove that the robot with* 4 *flags can explore the plane.*

A2. *Prove that the robot with* 3 *flags can explore the plane. Could the robot with* 3 *flags explore the three dimensional space?*

Let some borders of cells be impassable. The robot can see these barriers between the cells, when he is situated in one of these cells.

A3. *The border of the half-plane is impassable line. Prove that the robot with* 1 *flag can explore the half-plane.*

A4. *Prove that the robot with* 1 *flag cannot explore the plane with four cuts. (picture 1) Prove that the robot with* 2 *flags can do this.*

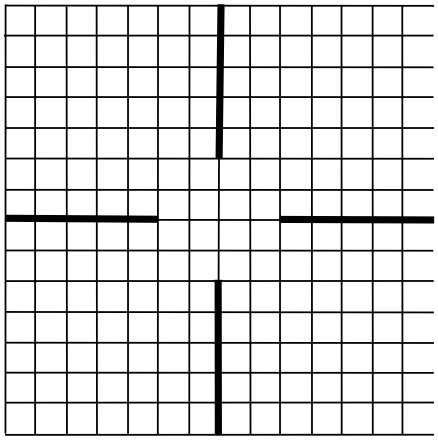

Fig. 1: Four cuts on the infinite plane

The robot is moving on the vertices of the graph. This graph may be infinite.

A5. a) *The graph is called tree if there are no cycles. Let a tree be infinite. Suppose that any vertex of the tree has degree* 3. *Prove that the robot with* 1 *flag cannot explore this tree.*

b) *The same question for the robot with* 2 *flags.*

A6. *Prove that the robot with* 2 *flags cannot explore the plane.*

Hint. Use the ideas of the problems A0 and A4.

The robot without any flags is moving on the first quarter of the plane. The borders are impassable walls. The robot can see a wall then he is in the cell with this wall.

A7. *Let the robot be in the cell with coordinates* $(2^n, 0)$. *Write a universal (not depending on* n*) program, for moving the robot into the cell* $(3^n, 0)$ *with stopping in that cell.*

Let us call the transition above as *transition from* 2^n *to* 3^n.

A8. *Write the programs for the robot to make transitions below:*

a) from 2^n *to* 6^n;

b) from 2^n *to* 2^{2^n};

c) from $2^n \cdot 3^m$ *to* $2^n \cdot 3^m \cdot 5^{mn}$;

d) from $2^n \cdot 3^m$ *to* $2^n \cdot 3^m \cdot 5^{m+n}$;

e) from 2^n *to* $2^{\lfloor\sqrt{n}\rfloor}$;

f) from 2^n *to* 2^{n^2};

g) from 2^n *to* 2^{k_n}, the k_n is the n-th decimal digit of the $\sqrt{2}$;

h) from 2^n κ 2^{k_n}, the k_n is the n-th decimal digit of the π;

A9. *Prove that the robot with* 3 *flags can explore* n*-dimensional space.*

A10. *Prove that the robot with* 3 *flags can explore a tree with degree of each vertex equal to* 3.

Villager's algorithms.

In this section we will explore a *world* – connected subset of cellular plane. Usually, we don't know its size. Every cell on the plane has one of the following types:

1. Rock. The villager could not visit this cell.

2. Ground. The villager can walk through it.

3. Lake. This cell is impassable, but the villager can fishing here from some neighboring cell. There are following parameters for each lake-call: `nibble` (the amount of fish that villager can get from lake using special «fishing» operation); and `fish` - the amount of remaining fish in the lake.

4. House. This is the unique cell in the world. Usually, villager starts here. In some problems the villager should bring home his fish.

The villager can walk thorough cell to another neighboring cell. Being in some cell he looks to the one of four possible directions.

Villager's *action area* is a set of two elements: a cell with the villager and a neighboring by villager's direction cell.

The villager can use information from his action area's cells. But he knows nothing about other cells. Also he has some number of flags (limited by specific problem's conditions). He can drop down these flags in some places and explore the world. The villager can fish on the neighboring lake-cell. Also he can bring some amount of fish limited by 10 kg.

Cells in the world can contain some objects (flags or fish). These objects include some variables. The villager can read this information. Also, he can change flag's variables or create the new ones.

The villager has limited memory. He can create self variables and change its values. The villager can use self variables and its values independently of his location.

Formally, the villager can do the following actions:

1. [Go] Go forward.
2. [Rotate] Rotate to some direction.
3. [Read] Read some information about variables on the objects in his action area.
4. [SetFlag] Set flag in the action area's cell. Or take flag from it. Also it is possible to create some variables on the flags and change its values.
5. [CheckFlag] Check out some flags in the action area.
6. [Fishing] Fishing in the neighboring cell (in the forward direction).
7. [GetPutFish] Get or put down some fish in the action area.
8. [Write] Create self variable or change its value.
9. [Math] Make arithmetic and logic operations using variables. For example, it is possible to check an equality of some variables.

Passing of time. The time in the world is a sequence of *time beats*. If there are no special instructions, each execution of Go, Fishing and GetPutFish operations interrupt the current beat (and we proceed to the next beat). Other operations Rotate, Read, SetFlag, CheckFlag, Write, Math take no time but if there are 500 such operations in the current beat, we proceed to the next beat too. Also, execution of CheckFlag second time in the same beat also interrupt this beat.

In the following problems one should find an villager's algorithms. You can construct and run your own algorithms (flowcharts) using special simulation application «VillagerLife». You will get access to this application.

There are some maximum working time for every problem. The size of maps is unknown. If there are no special conditions, the villager can check the edge of the map (if he is located on the edge cell). The villager can explore a map if he can visit every approachable cell using some algorithm.

Section B. In this section we could not use self variables.

B1. *Explore finite rectangular map (without lakes and rocks) using four flags. In this problem villager could not check an edge of a map.*

B2. *There are two flags on the map (without lakes and rocks) : one in the cell with coordinates* $(x, 0)$*, second in the cell* $(0, y)$*. The villager has a third flag and starts in origin* $(0, 0)$*. How the villager can set any flag to the cell* $(x + y, 0)$*?* x, y *– are some positive integers.*

B3. *Now our maps contain rocks and lakes. Explore a map using any number of flags. (As much as villager need). Put a flag in every approachable cell..*

B4. *Solve the previous problem using* $2n$ *time beats, there* n *is the number of approachable cells.*

Section C. In this section we can use self variables.

C1. *Find the number of approachable cells and write it to the* `SPACE` *variable. Time is limited by* $2n$ *beats, there* n*- is the number of approachable cells.*

C2. *Find the number of approachable lake-cells and write it to the* `WATER` *variable.*

C3. *Find the shortest distance from house to a flag with variable* `TARGET=THIS`*. Write this distance to the variable* $DIST$*.*

C4. *Find the approachable lake-cell with maximal nibble. Write its coordinates to the variables* `NIBBLE_X` *and* `NIBBLE_Y`*. Write the value of nibble to the variable* `NIBBLE`*.*

C5. *Let us define a usefulness of lake-cell. This is a specific maximal amount of fish that villager can bring home (ratio of maximal amount of fish can be obtained in this lake and minimal number of beats used for fishing and bring fish home). Find the lake-sell with maximal usefulness. Write its coordinates to the variables* `USEFULNESS_X` *and* `USEFULNESS_Y`*. Write the value of usefulness to the variable* `USEFULNESS`*.*

C6. *Collect* `n` *kg of fish in the house. The value of* `n` *is recorded in variable* `FishReq` *on the house.*

C7. *There is another villager with house in the map. He can fishing too but could not steal it from our house. Collect* `n` *kg of fish in the house. The value of* `n` *is recorded in variable* `FishReq` *on the house.*

C8. *There is another villager with house in the map. He can fishing too and can steal it from our house. Collect* `n` *kg of fish in the house. The value of* `n` *is recorded in variable* `FishReq` *on the house.*