Home Work 03

Unnecessary problems.

Each labyrinth cell contains data (its weight). As result, we should get solution with minimum sum of its cells weights. When robot enters the labyrinth, it can only see weights of surrounding cells. For example, there may be next situation:

- R — Robot (current position).
- B — Enter cell.
- T — Exit cell.

Picture 1.

We have two routes. First is shorter, but weights more, then second one. The robot is now on cell (1,7), and surrounding cells are (1,6) with weight = 1 and (2,7) with weight = 6. I decided not to compare them, because robot cannot know the weight of the further cell(s). Logically, it may select cell (1,6), as it has lower weight, but in further, as we see, this way will weigh more. Even if you will compare weights of routes 1 and 2 while passing them, you will need to 'jump' between them. It will take more time.

	0	1	2	3	4	5	6	7	8
0	#	#	#	#	#	#	#	#	#
1	#	#	#	#	1	1	1	#	#
2	#	8	1	6	1	#	1	#	#
3	#	4	#	#	T	#	1	#	#
4	#	2	#	#	#	1	1	1	#
5	#	9	#	#	#	1	#	#	#
6	#	1	#	#	#	1	#	#	#
7	#	R	6	1	1	1	#	#	#
8	#	B	#	#	#	#	#	#	#

Route Picture 1

NNNNNNEEES: 32

Route NEEEENNNENNNWWSS: 19

Pictures 2 & 3.

On picture 2 robot has reached the cell (1,5), and its current route(NNN) weight = 11.

Second alternative (picture 3). Robot can discover, that second route(NEE) is shorter, and its weight will be 8. But to do this, he must go back for two steps, then go to cell (3,7). As you can see, now the first route is not a success solution. So, robot will make more steps, if it will choose shortest way every time after entering a new cell.

	0	1	2	3	4	5	6	7	8
0	#	#	#	#	#	#	#	#	#
1	#	#	#	#	1	1	1	#	#
2	#	8	1	#	1	#	1	#	#
3	#	4	#	#	T	#	1	#	#
4	#	2	#	#	#	1	1	1	#
5	#	R	#	#	#	1	#	#	#
6	#	1	#	#	#	1	#	#	#
7	#	1	6	1	1	1	#	#	#
8	#	B	#	#	#	#	#	#	#

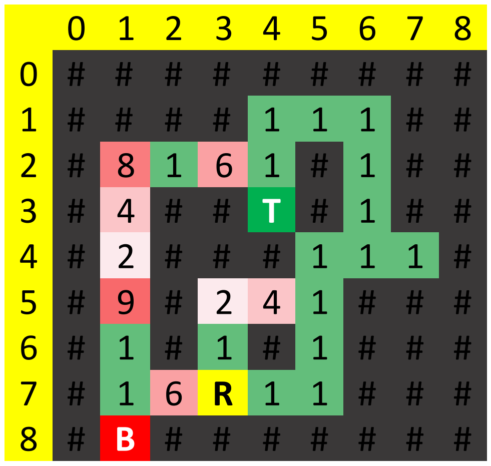
Current: 11 Picture 2

	0	1	2	3	4	5	6	7	8
0	#	#	#	#	#	#	#	#	#
1	#	#	#	#	1	1	1	#	#
2	#	8	1	#	1	#	1	#	#
3	#	4	#	#	T	#	1	#	#
4	#	2	#	#	#	1	1	1	#
5	#	9	#	#	#	1	#	#	#
6	#	1	#	#	#	1	#	#	#
7	#	1	6	R	1	1	#	#	#
8	#	B	#	#	#	#	#	#	#

Current: 8 Picture 3

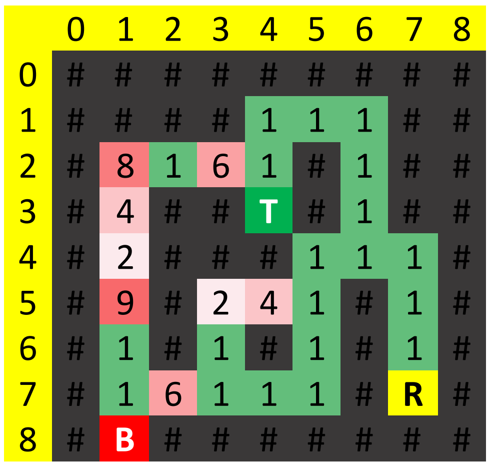
Robot’s moves.

1. When robot enters the labyrinth, program creates ‘zero’ node (position (1,8)). Every node has array of further possible directions. So, after entering he looks around and chooses first possible way to go to in order (W, S, E, N). Also, it removes selected direction from this node, so when robot will enter this cell next time, it would not have possibility to go this way one more time.
2. Every time before moving, it looks around, e.g. for picture 1 it will choose to go in direction E, because W and S has no suitable cells. For position on picture 2 it has only one direction to move (it’s N). It ignores last visited cell when it chooses next one.
3. If robot finds more than one further way possible, it adds current position to this route’s ignore case to avoid infinity loop (picture 4). Actually, there are two loops — big and small one.
4. Every time robot enters next cell he checks if this cell’s ‘current’ data equals T. If the condition passes, it adds route to route list.
5. If cell has no more possible directions (picture 5), robot will go back (every node stores its parent, except ‘zero’ node, the ‘zero’ node’s parent is ‘null’) until it finds next possible way. In picture 5 it will go back to position (6,4), then discover next possible way to go (N) to cell (6,3).
6. When robot has discovered all possible routes, it returns to ‘zero’ node position. When it discovers, that current node’s parent is null, it exits the cycle.
7. After we get all possible routes, they are sorted by weight, and as the result we get the lightest one.



	0	1	2	3	4	5	6	7	8
0	#	#	#	#	#	#	#	#	#
1	#	#	#	#	1	1	1	#	#
2	#	8	1	6	1	#	1	#	#
3	#	4	#	#	T	#	1	#	#
4	#	2	#	#	#	1	1	1	#
5	#	9	#	2	4	1	#	#	#
6	#	1	#	1	#	1	#	#	#
7	#	1	6	R	1	1	#	#	#
8	#	B	#	#	#	#	#	#	#

Picture 4



	0	1	2	3	4	5	6	7	8
0	#	#	#	#	#	#	#	#	#
1	#	#	#	#	1	1	1	#	#
2	#	8	1	6	1	#	1	#	#
3	#	4	#	#	T	#	1	#	#
4	#	2	#	#	#	1	1	1	#
5	#	9	#	2	4	1	#	1	#
6	#	1	#	1	#	1	#	1	#
7	#	1	6	1	1	1	#	R	#
8	#	B	#	#	#	#	#	#	#

Picture 5