

SHORTEST PATH ALGORITHM

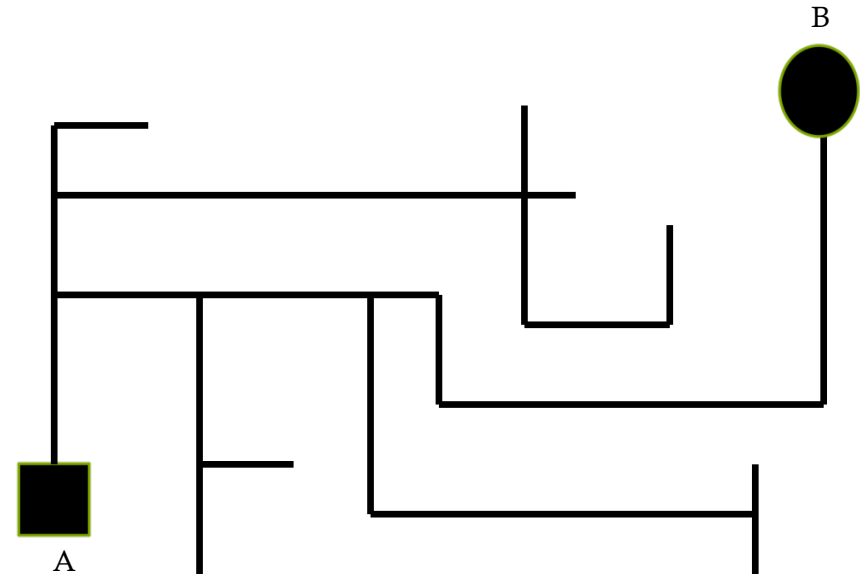
Contributions from Deepesh

WHY FIND THE SHORTEST PATH?

Imagine if your robot has to start from A and go to B – repeating this for 100 times !

How much time could it save if it just finds and remembers the shortest path and avoids all the dead-ends and unwanted turns in the junction !

This optimized robot will be the right choice in warehouse and other industry applications.



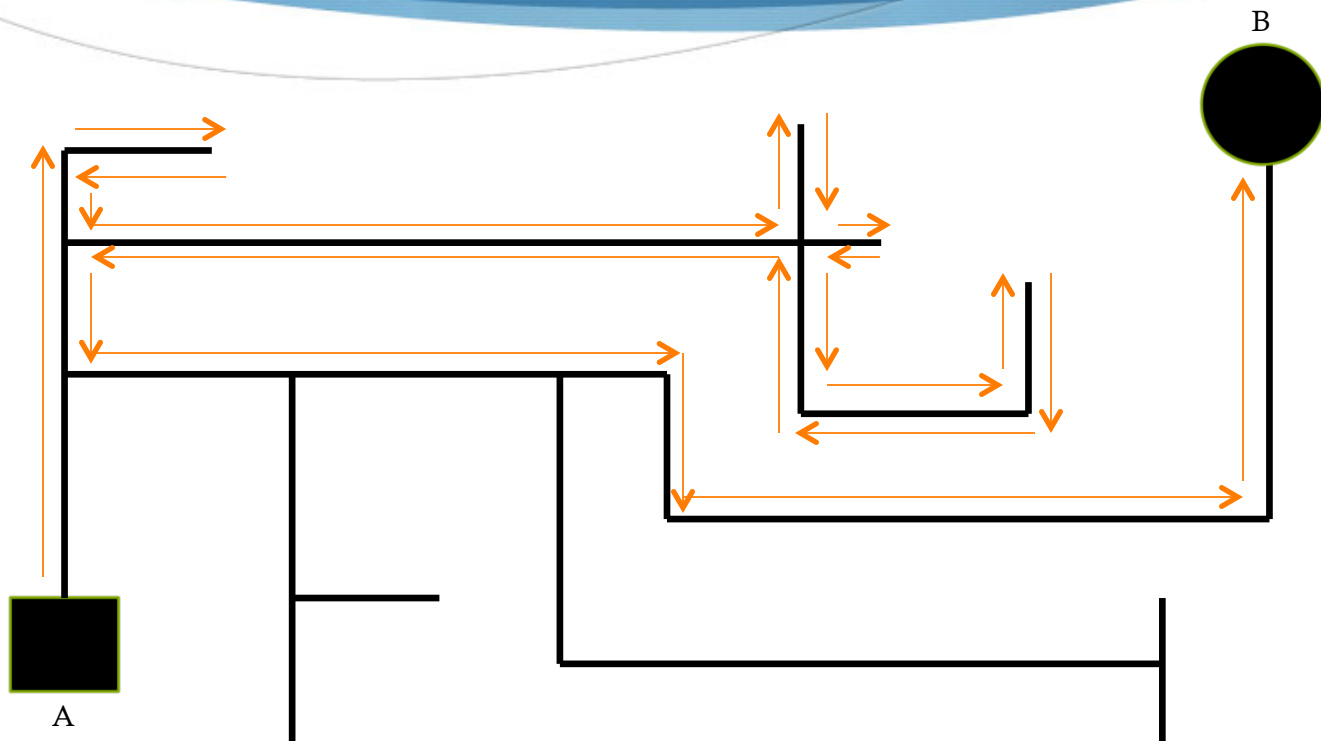
UNDERSTANDING WHY

To understand the importance of the shortest path, let us see how your robot moves in the given arena for the most popular algorithms

1. Left Algorithm

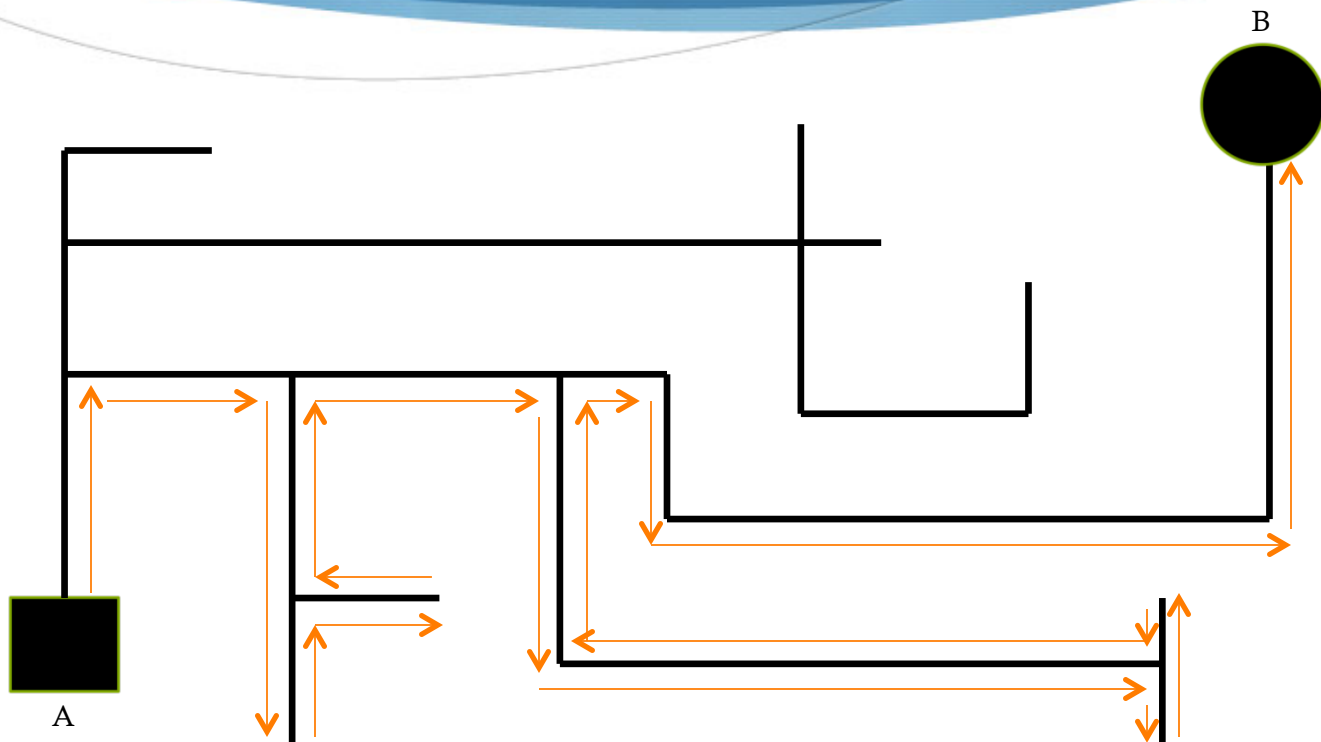
2. Right Algorithm

LEFT ALGORITHM



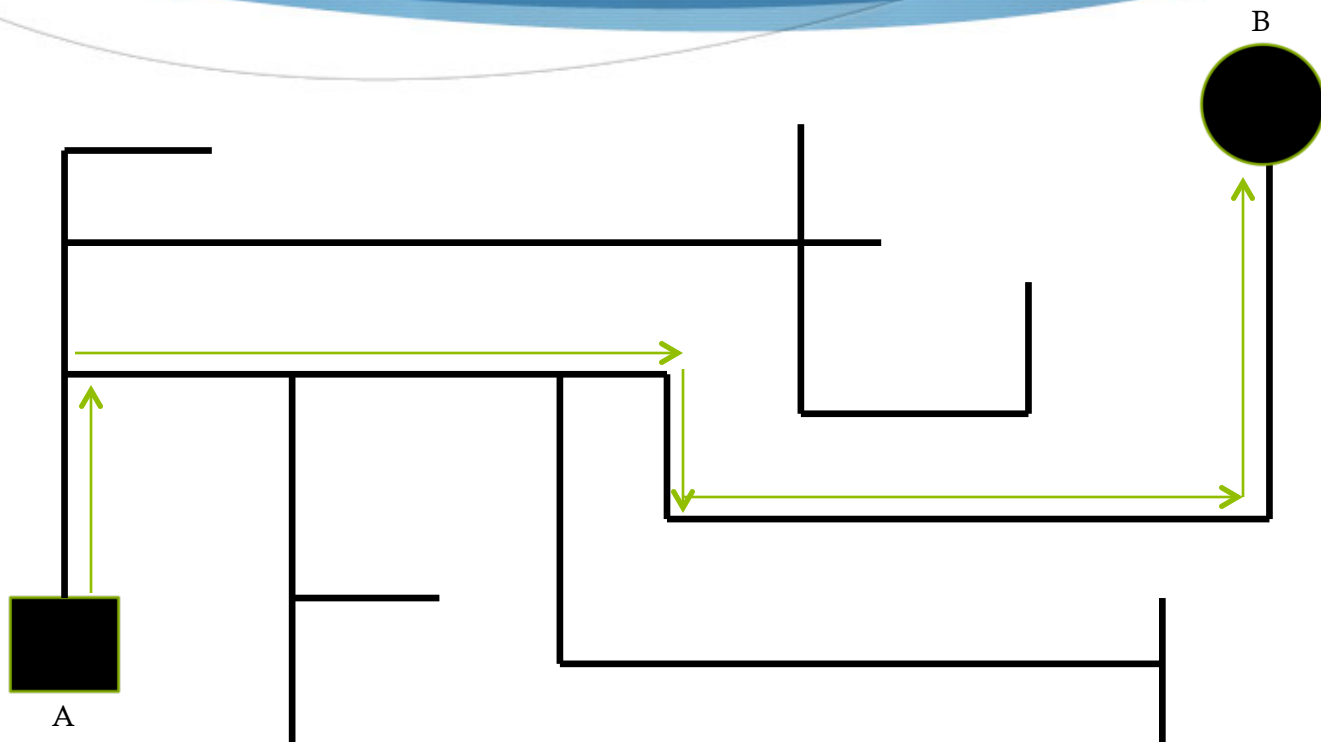
The arrows show the path in which your robot moves if it is programmed to perform the left algorithm

RIGHT ALGORITHM



The arrows show the path in which your robot moves if it is programmed to perform the right algorithm

SHORTEST PATH



This is the shortest path for this arena which the robot should find and trace continuously ! Just imagine how much time could be saved !

HOW TO DO?

To Achieve this you need to help your robot remember its choices in every junction, and eliminate the bad choices – thereby, deriving the shortest path. To understand this, you first have to understand the difference between

a turn, and

a junction.

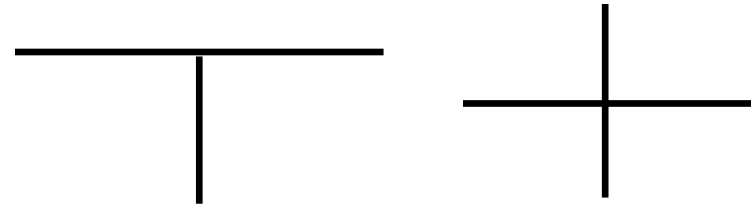
TURN AND JUNCTIONS

URNS



A Turn is when a robot has no other option other than making that turn – for your shortest path algorithm, your robot need not remember such turns

JUNCTIONS



A Junction is when a robot has more than 1 option to move – eg: it can either take a left or go straight. In other words, junction is a point where more than one line intersects. These are very important for your shortest path algorithm

THINGS TO REMEMBER FOR YOUR ROBOT

These are the two most important actions your robot has to remember:

1. Its choice in every junction
2. When it encounters a dead end

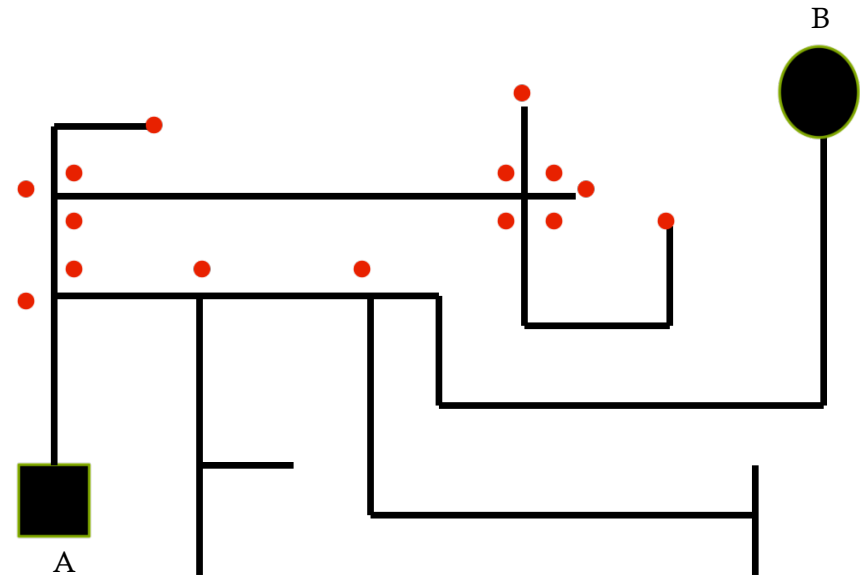
How will it remember this?

Your robot has to store the above two actions every time in an array called “Direction”.

IMPLEMENTATION

Suppose your robot is programmed for left algorithm – and finds the shortest path, we shall see how it remembers its actions and how it eliminates its bad choices

The red points indicate the junctions/ dead-ends which the robot will encounter and remember.

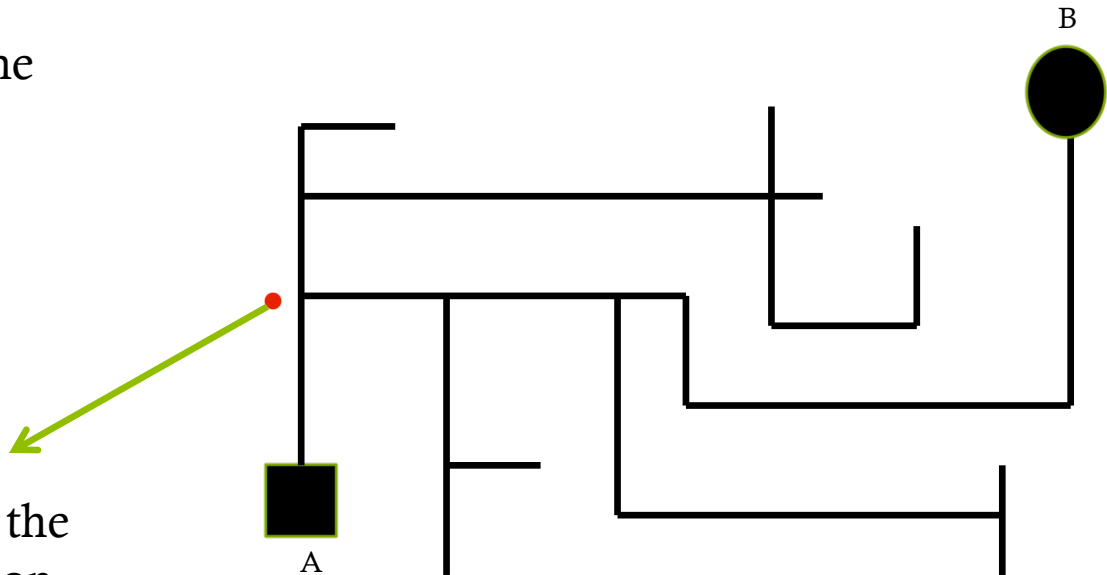


STEP 1

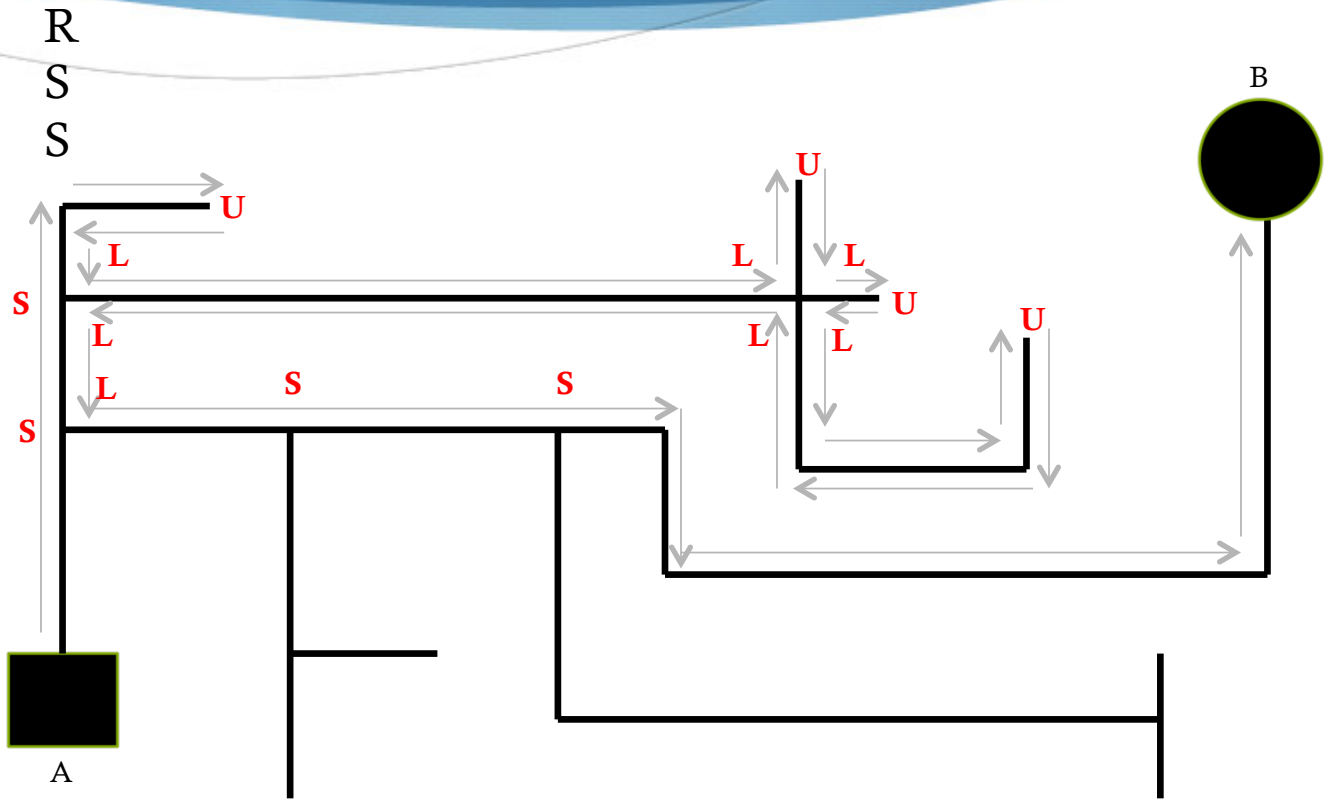
Let us give a name to every turn the robot makes:

1. Left \rightarrow L
2. Right \rightarrow R
3. Straight \rightarrow S
4. U-Turn \rightarrow U

So, for the first junction, the robot makes a choice of Straight - thus, the robot saves its choice in the direction array
 $\text{dir}[0] = \text{S};$



COMPLETING THE DIRECTION ARRAY

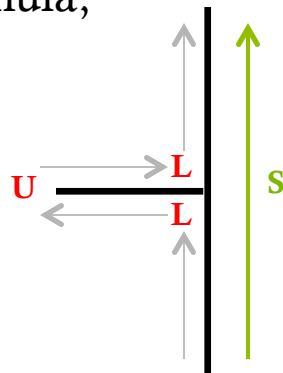
$$\begin{aligned} \text{Dir}[0] &= \text{S}; \\ \text{Dir}[1] &= \text{S}; \\ \text{Dir}[2] &= \text{U}; \\ \text{Dir}[3] &= \text{L}; \\ \text{Dir}[4] &= \text{L}; \\ \text{Dir}[5] &= \text{U}; \\ \text{Dir}[6] &= \text{L}; \\ \text{Dir}[7] &= \text{U}; \\ \text{Dir}[8] &= \text{L}; \\ \text{Dir}[9] &= \text{U}; \\ \text{Dir}[10] &= \text{L}; \\ \text{Dir}[11] &= \text{L}; \\ \text{Dir}[12] &= \text{L}; \\ \text{Dir}[13] &= \text{S}; \\ \text{Dir}[14] &= \text{S}; \end{aligned}$$


FORMULAS

To eliminate the bad choices, we have some formulas which you need to understand.

1. When a robot takes a left, encounters a dead-end and again takes a left, then the robot should have actually chosen Straight path. Hence the formula,


$$LUL = S$$



Similarly,

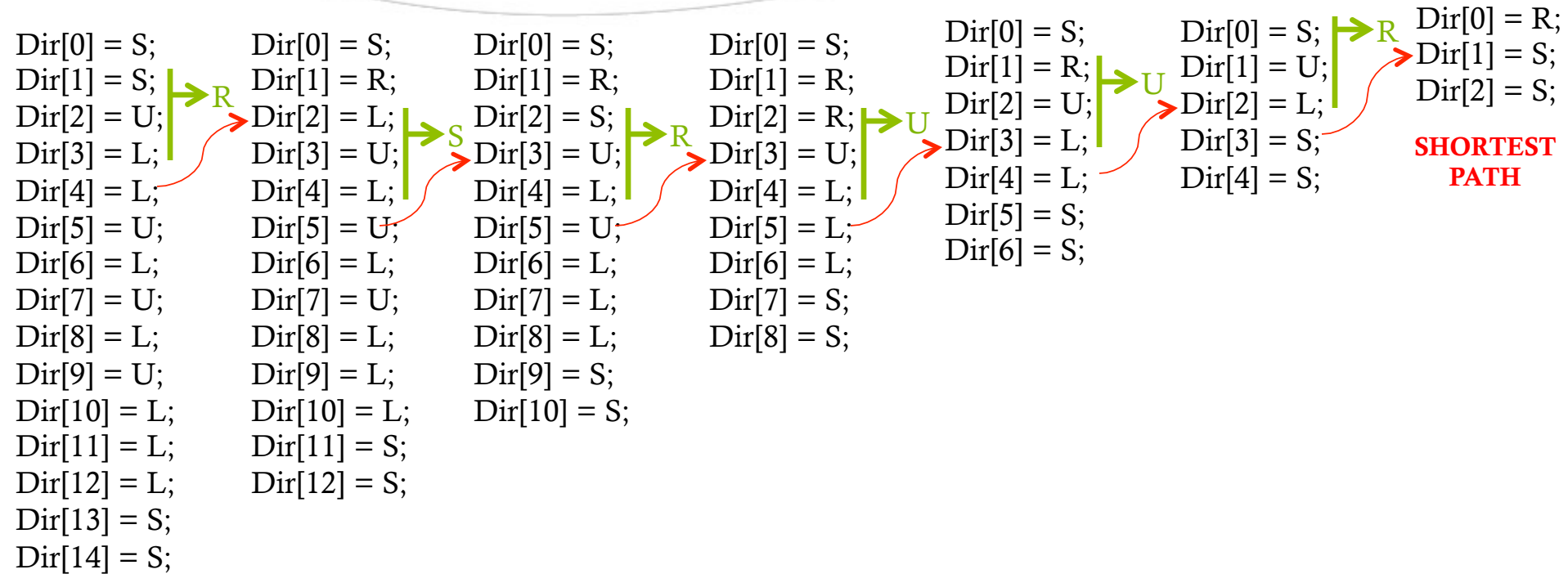
2. $RUR = S$
3. $SUL/LUS = R$
4. $SUR/RUS = L$
5. $RUL/LUR = U$
6. $LSUSL = S$
7. $RSUSR = S$

IMPLEMENTING THE FORMULAS







Dir[0] = S;
Dir[1] = S;
Dir[2] = U;  R
Dir[3] = L;
Dir[4] = L;
Dir[5] = U;
Dir[6] = L;
Dir[7] = U;
Dir[8] = L;
Dir[9] = U;
Dir[10] = L;
Dir[11] = L;
Dir[12] = L;
Dir[13] = S;
Dir[14] = S;

- Take the first element of the array, and check if it matches with any of the formula,
- If not, go to the next element
- If yes, then replace it with the formula result
- And eliminate the unwanted entries

STEP BY STEP INSIGHT



ELIMINATION PROGRAM

```
for (i=0; i<n;i++)  
{  
  change=0;  
  if(dir[i]==S && dir[i+1]==U && dir[i+2]==L)  To check if the entries match any of  
  {  Setting the formula's result  
    dir[i] = R;  
    for(j=i+1; (j+2)<n; j++)  Eliminating the two unwanted entries and every  
    {  Array size will decrease because of the elimination  
      dir[j] = dir[j+2];  
    }  
    n=j;  
    change=1;  
  }  
  else if.....next formula (totally 10 formula)  
  if(change==1)  If a change has occurred, then the loop must start from the beginning  
  {  If a change has occurred, then the loop must start from the beginning  
    i=(-1);  
  }  
}
```

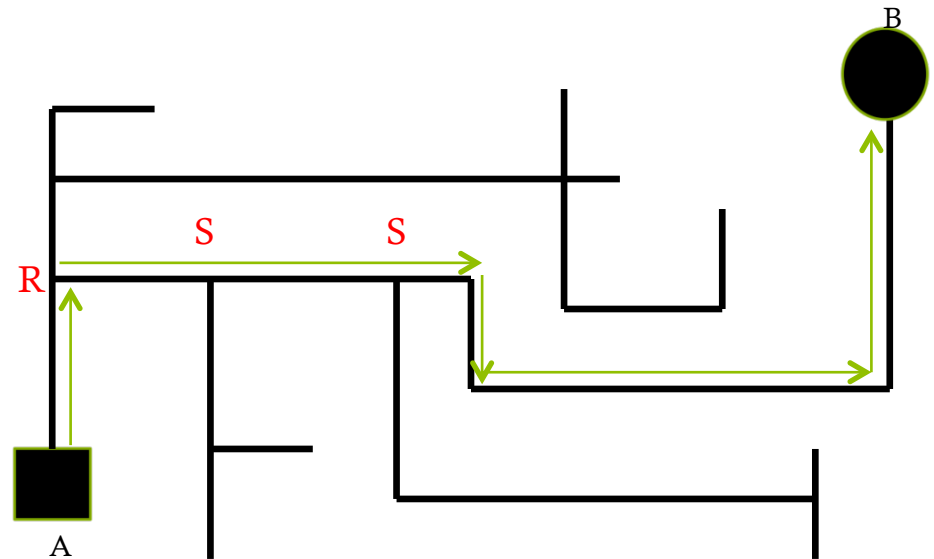
AFTER FINALIZING

After the eliminations are done, then the robot will just have to follow the array at every junction it encounters

Dir[0] = R;

Dir[1] = S;

Dir[2] = S;

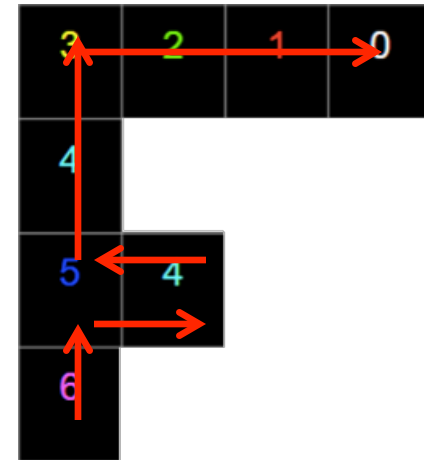


FLOODFILL WITH SHORTEST PATH

Since in the flood fill algorithm we know the row and column value at every position, we have a different and easier method to implement shortest path !

Suppose there is a row and column array which records the pathway for every step,

| Step Num. | Row Value | Column Value |
|-----------|-----------|--------------|
| 1 | 0 | 0 |
| 2 | 1 | 0 |
| 3 | 1 | 1 |
| 4 | 1 | 0 |
| 5 | 2 | 0 |
| 6 | 3 | 0 |
| 7 | 3 | 1 |
| 8 | 3 | 2 |
| 9 | 3 | 3 |



FLOODFILL WITH SHORTEST PATH

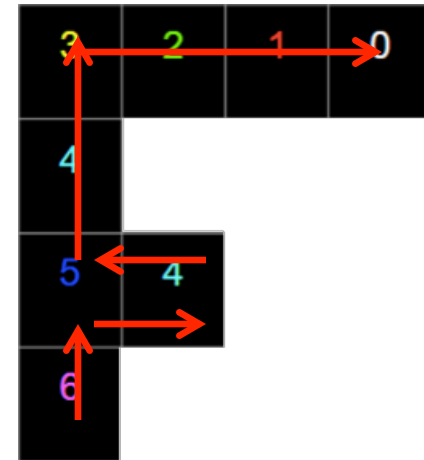
In this you can notice that,
Step 1 and Step 3 are the same

This means that the robot has
taken a wrong path and has
returned to the same spot again

So we can eliminate all the steps
after step1 till step3

How to implement this
elimination?

| Step Num. | Row Value | Column Value |
|-----------|-----------|--------------|
| 0 | 0 | 0 |
| 1 | 1 | 0 |
| 2 | 1 | 1 |
| 3 | 1 | 0 |
| 4 | 2 | 0 |
| 5 | 3 | 0 |
| 6 | 3 | 1 |
| 7 | 3 | 2 |
| 8 | 3 | 3 |



IMPLEMENTATION IN FLOOD FILL

| Step Num. | Row Value | Column Value |
|-----------|-----------|--------------|
| 0 | 0 | 0 |
| 1 | 1 | 0 |
| 2 | 1 | 1 |
| 3 | 1 | 0 |
| 4 | 2 | 0 |
| 5 | 3 | 0 |
| 6 | 3 | 1 |
| 7 | 3 | 2 |
| 8 | 3 | 3 |

```
for(i=0; i<n; i++)  
{  
    for(j=i+1; j<=n; j++)  
    {  
        if(row[i] == row[j] && col[i]==col[j])  
        {  
            for(k=i+1; j<=n; j++,k++)  
            {  
                row[k]=row[j+1];  
            }  
            n=k;  
        }  
    }  
    i=i-1;  
}
```

→ Taking every entry one by one to check for a duplicate

→ Checking if it matches with any of the entries in the entire array set

→ Eliminating the steps in between the duplicates and every other entries moves forward in the array

→ N value will reduce due to the elimination

→ Reducing 'i' so that it checks again from the duplicate entry.

WWW.KIDOBOTIKZ.COM