Table of Contents

Day 5 Adventure: Back to the Future

“We have accomplished our missions in this past world, “ Dark Knight exclaims, “now we need to go back to the future!”

With her photographic memory, Bubble Gum recites their science teacher's words before their departure, "enjoy the 4 days camping and see you all in my class on Monday!"

Banana Split added, "let's quickly figure out the launch point of the time tunnel! I miss our family and friends."

The launch point is hidden in the Polymorphic Woods. In Grandma's story, the landscape is changing every season. When our Earth is strong, the trees are lined up nicely.

When our Earth is under weather, the plants grow in all directions signaling the urgent need for attention.

The BestFour gathers all the souvenirs from this past world and takes off to the Woods in their vessel. To their surprise, the Polymorphic Woods is indeed a complicated maze now. Since the Gate of Traceback has been sealed. It will self heal gradually. But they can not afford to wait that long. Remember, 1 time unit in this world is equivalent to 1000 time units in the future world (1 second = 1000 milli-seconds). Here everything goes much SLOWERRRRRR.

Without further ado, the BestFour starts figuring out the shortest path to traverse through the maze of Polymorphic Woods.

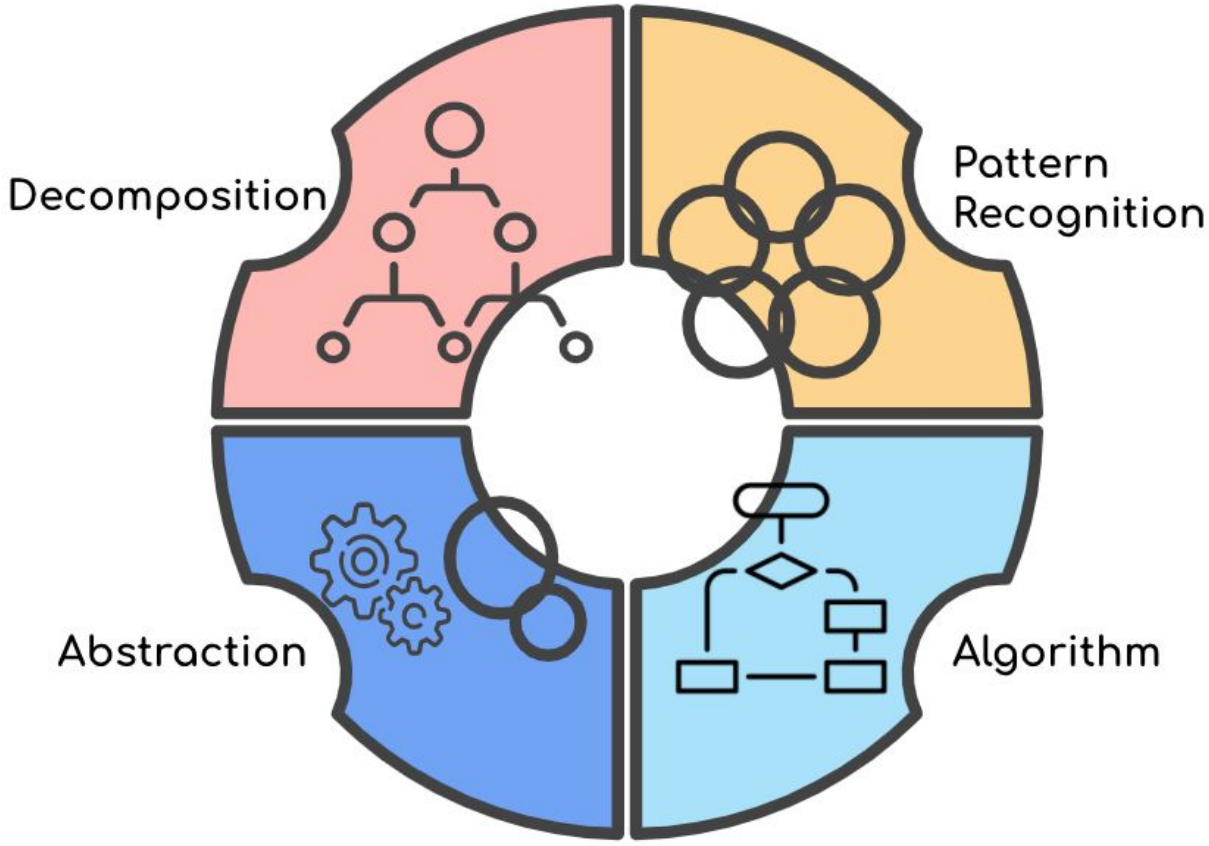
Story of Maze

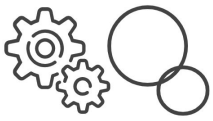
The name labyrinth came from the Greek myth. The Minotaur was a monster with the body of a man and the head and tail of a bull. It was the offspring of the Cretan Queen and a majestic bull. King Minos of Crete had people build a huge maze called Labyrinth to house the monster. Every nine years (some contradiction in the sources regarding this year), they threw seven youths and seven maidens into Labyrinth to feed the Minotaur. Eventually Athenian hero Theseus went into the maze and slaughtered the monster. He unwound a thread as he went into the Labyrinth and found his way back successfully using the thread.

Movie The Shining by Stephen King presented a labyrinth of hedges. The maze symbolizes the barriers in family members' connection. The little boy Danny has a strong connection with his mother, and they were able to navigate to the center successfully, whereas Danny's father was blocked by the maze and disoriented and trapped.

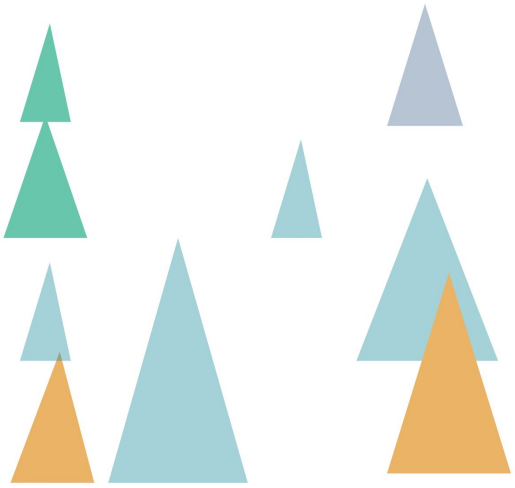
Maze is a complex branching puzzle with many choices of paths and directions. Labyrinth is a special type of maze which has only a single path to the center. Our life is a maze. We constantly face decisions, and frequently experience surprises and reliefs. If we keep an open and growing mind, many paths will eventually lead to our goals.

Wearing the CalliLens (COmputational Thinking)

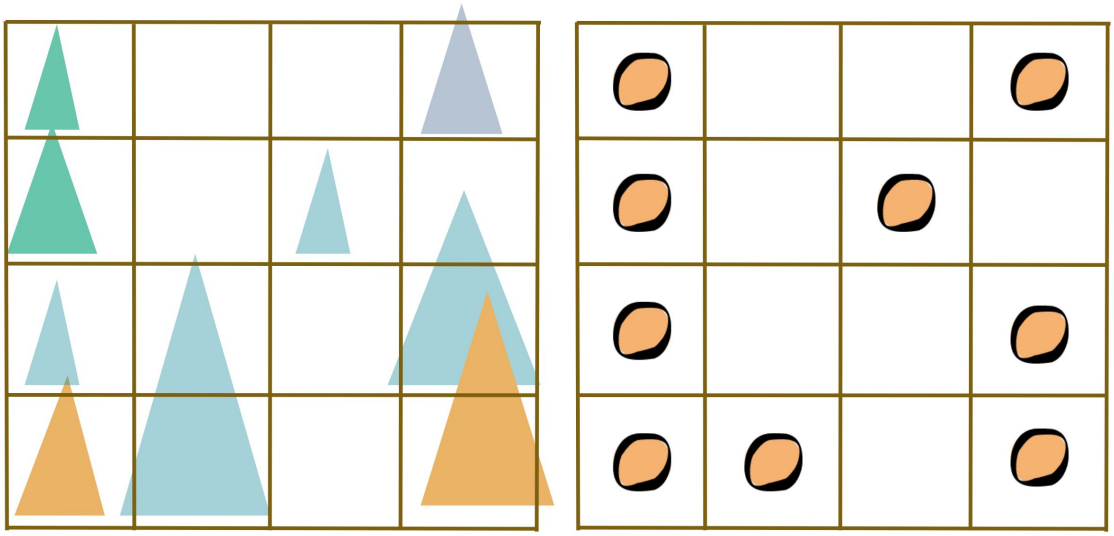




The woods have overwhelming details: oak, pine, walnut, cottonwood, buckeye, etc.



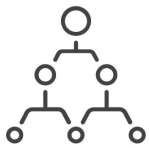
They all act as obstacles in the maze. CalliLens are able to filter out the unnecessary details and keep only the essential information.



2D Array (list of list) to represent maze

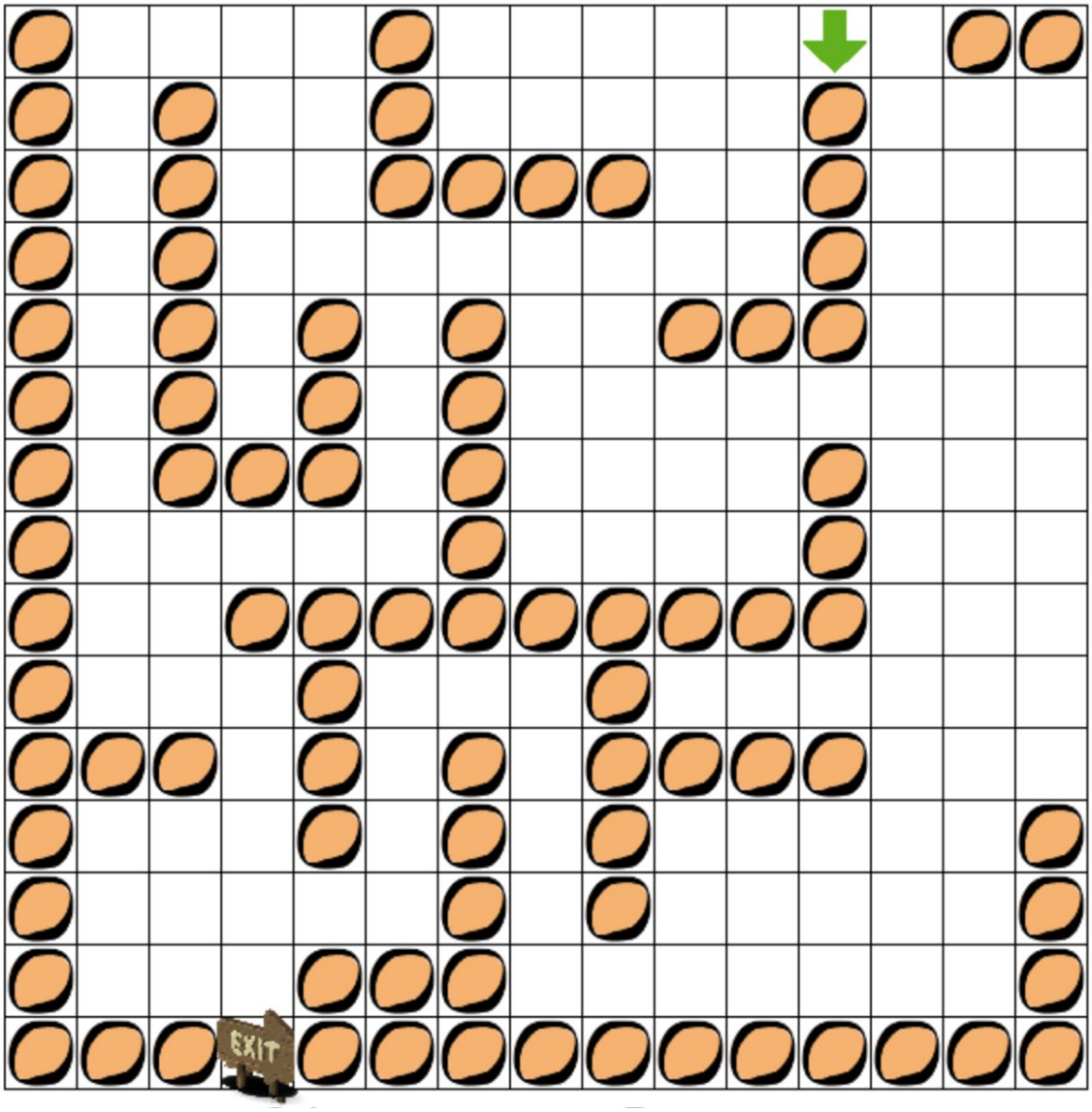
```
def init_maze():
    maze = []
    maze.append(["#", " ", "O", "#"])
    maze.append(["#", " ", "#", " "])
    maze.append(["#", " ", " ", "#"])
    maze.append(["#", "#", "X", "#"])

    return maze
```



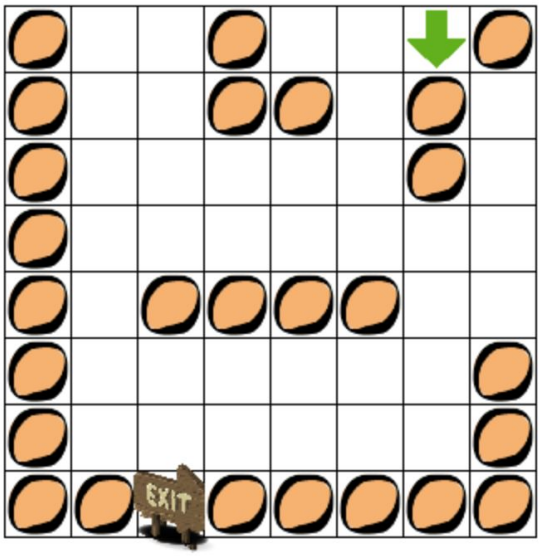
Decomposition

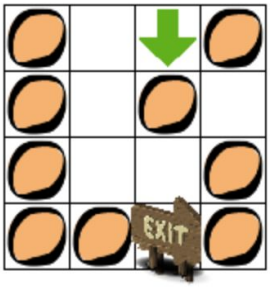
OMG!



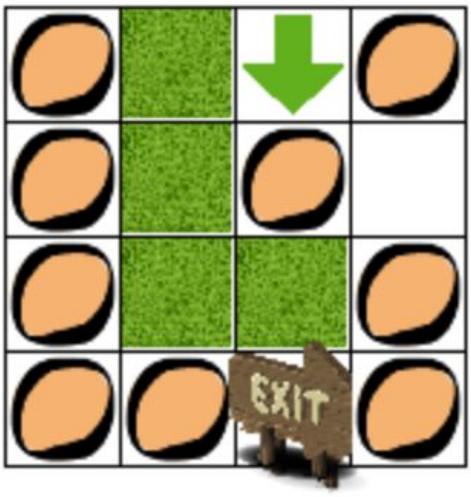
Monster Problem

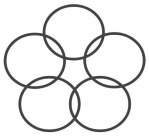
We can decompose it to some miniature manageable problem.





Now we can solve it so easily!

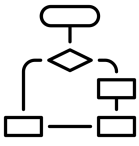




Pattern Recognition

Banana Split finds this question very intriguing.

When we look for the shortest path to the launch point in the Polymorphic Woods maze, we orient on the starting location and look around the neighbourhood (left, right, front, back) which is one step away. If not found, we move to the neighbour locations one by one and further explore its neighbours which is two steps away. We repeat this process until finding the launch point. Since every time we move one step further from the starting point, it guarantees that the first path found is the shortest path.



Algorithm Design

Here is the algorithm (solution steps):

A. Begin at the starting point, explore its adjacent locations
B. At each next step, always expand one step further from the already explored locations
C. Eventually reach the exit with the shortest path

Maze Solutions

Breadth First Search (BFS) Algorithm

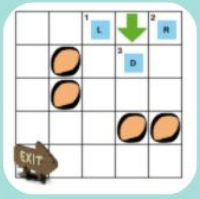
Our algorithm here is to search wide first rather than deep. This search algorithm is called **Breadth-First Search**, or **BFS**.

BFS works here because it doesn't consider a single path at once. It considers all the paths starting from the source and moves ahead one unit in all those paths at the same time which makes sure that the first time when the destination is visited, it is the shortest path.

In order to keep track of the already searched positions in order, we need to store them and maintain their sequence. This requirement reminds us of some daily experience [analogy] : we sometimes wait in a queue (or line) to be attended, e.g. supermarket checkout line. We walk to the tail of the queue and get served when we move up to the front.

> Black cell: the location oriented on which the neighbours are being explored
> Blue cell: locations in the queue
> Grey cell: locations removed from the queue and its neighbours have already been explored
> The small number in the top left corner shows the sequence of the visits.
>
> Valid neighbours (within boundary, no block, and not visited yet) are added to the end of the queue waiting for their neighbours to be further explored.

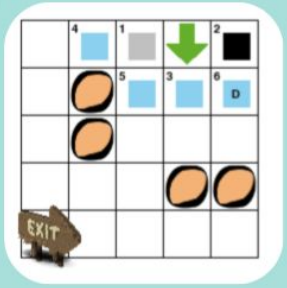
[ToDo: BFS left page, DFS right page]



1. Start exploring neighbours (LRUD)
Queue: (0,2) (0,4) (1,3)

therefore called Breadth First Search

2. Retrieve from Queue
explore its neighbours
Queue: (0,4) (1,3) (0,1) (1,2)



3. Repeat step 2
Retrieve from Queue
explore its neighbours
Queue: (1,3) (0,1) (1,2) (1,4)

4. When retrieving (3,0) from the Queue and exploring its neighbour, find EXIT

Mission accomplished!

Queue

In the computer, when we need to maintain the order of some data items and retrieve them one by one based on their arrival order, we use a **queue**: **first in first out (FIFO)**!

Voilà, the problem is solved! We store the already searched locations into a queue. The newly searched location is added to the tail of the queue. Each time we retrieve one location from the head of the queue and expand to its surrounding area, we then add the expanded area to the queue until we find the target.

[queue diagram - (queue diagram, head, tail, FIFO)]

Queue operations

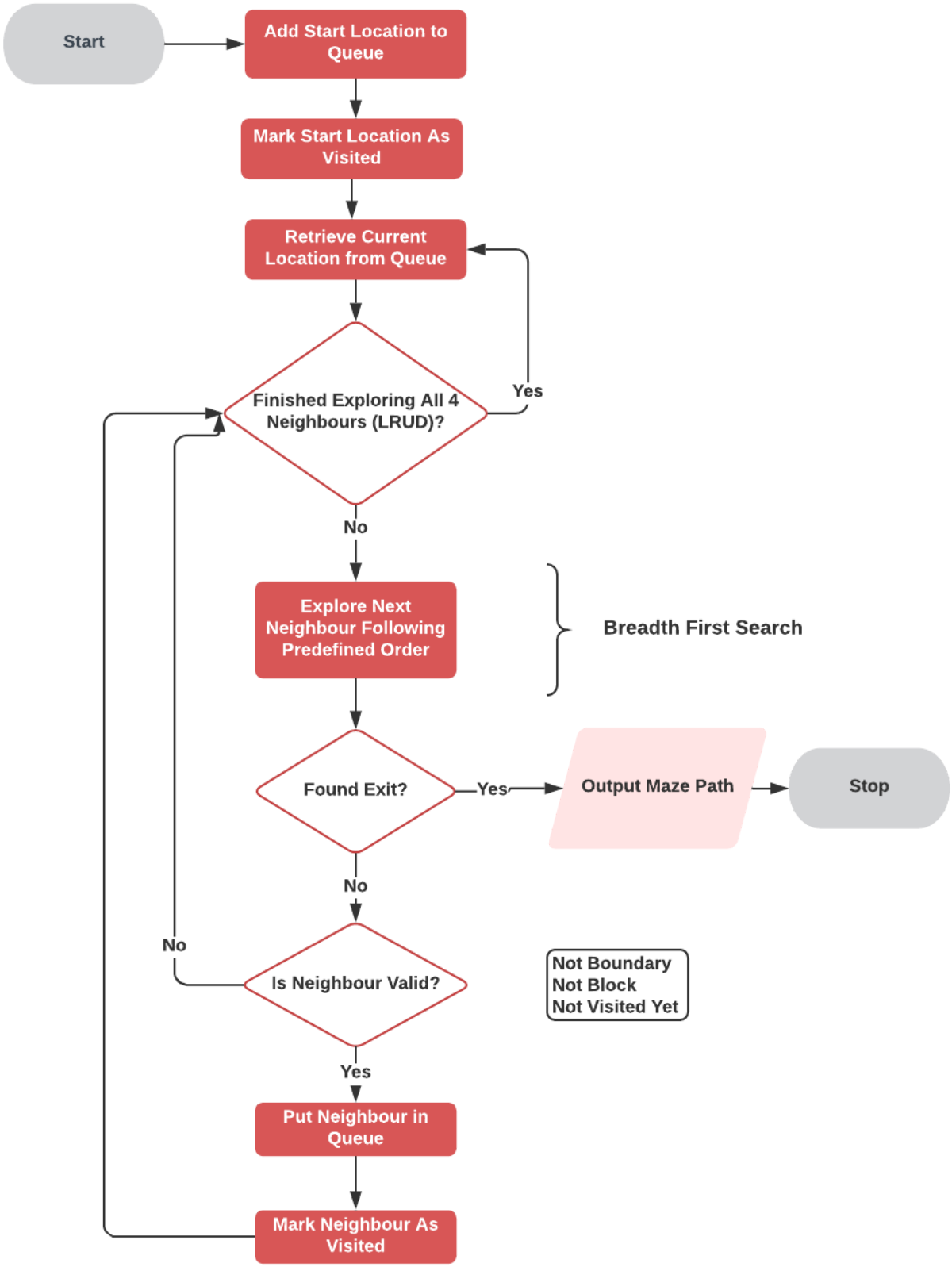
- CreateQueue(): creates an empty queue
- Enqueue(queue, item): adds item to the tail of the queue
- Dequeue(queue): removes an item from the head of the queue and makes the next item the new head. If the queue is empty, Dequeue() generates an error.
- IsQueueEmpty(queue): returns True if the queue is empty, False otherwise.

Optimization to Prevent Repeated Paths

With suboptimal memory, we might repeat the same route and delay the quest. Remember that computer, our friend, has the best memory. We just need to create a **visited** data storage to mark the visited locations so that we will not repeat the same location.

[ToDo: need to explain visited data structure for optimization]

Diagram Illustration


Start
Add Start Location to Queue
Mark Start Location As Visited
Retrieve Current Location from Queue
Finished Exploring All 4 Neighbours (LRUD)?
Yes
No
Explore Next Neighbour Following Predefined Order
Breadth First Search
Found Exit?
Yes
Output Maze Path
Stop
No
No
Is Neighbour Valid?
Not Boundary
Not Block
Not Visited Yet
Yes
Put Neighbour in Queue
Mark Neighbour As Visited

Mighty Python

Maze Representation

Maze.py at https://github.com/applepiinc/arithmaticthinking/tree/main/maze/2Darray

BFS Implementation

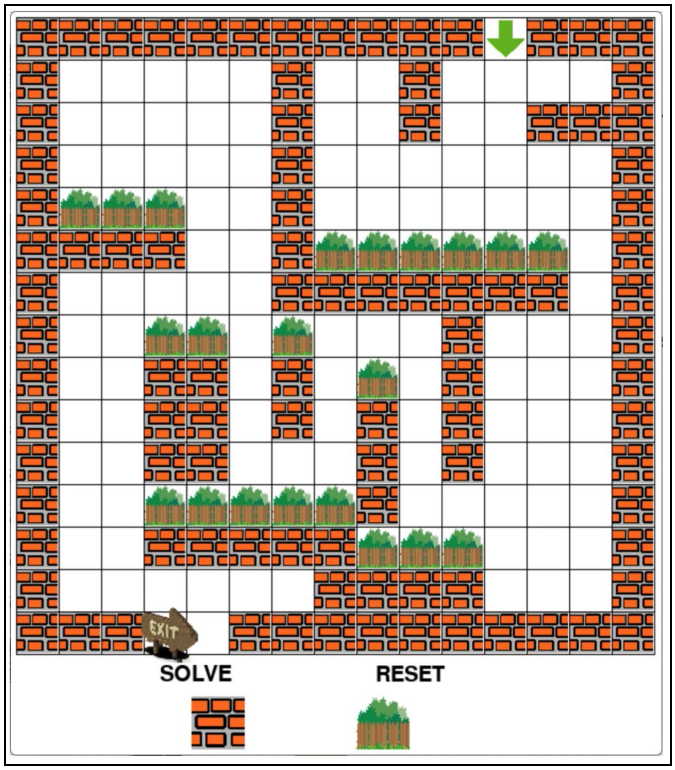
mazesolver_BFS.py at
https://github.com/applepiinc/arithmaticthinking/tree/main/maze

Maze Pygame

pygame_mazesolver_BFS.py and related 5 images (start, end, wall, grass, path) at
https://github.com/applepiinc/arithmaticthinking/tree/main/maze

Maze Demo

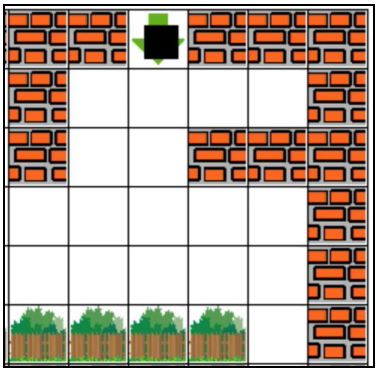
pygame_mazesolver_BFS_DemoPlayModes.py
path) at
https://github.com/applepiinc/arithmaticthinking/tree/main/maze
Let's say, we want to solve the following maze with the BFS algorithm.

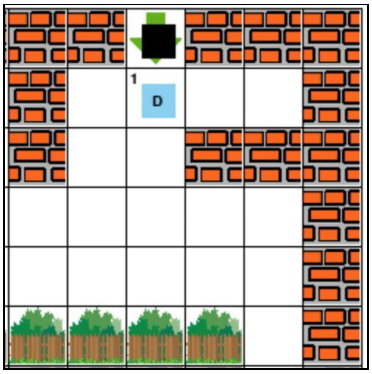


In the following diagram:

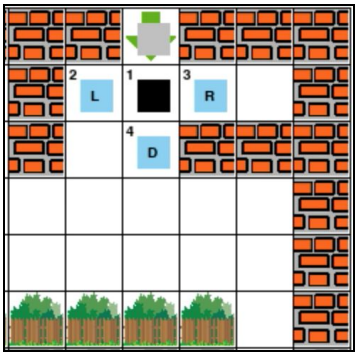
- Number: step sequence number (note, computer can teleport between non-adjacent locations using queue, e.g. from step # to step ##) [Fun fact]
- Black: represent the location which the neighbourhood (breadth) is being explored
- Light blue: represent the queue which stores the visited locations, but their neighbourhood (breadth) is not explored yet.
- Grey: represent the locations that have been removed from the head of the queue and we have explored their neighbourhood (breadth)

Program will explore from the starting location, explore its visitable neighbours. Because there are barriers to the left and right, and the maze boundary to the top, the only visitable neighbour is location 1 by going down. Computer puts location 1 to the tail of the queue so that it remembers to explore location 1's neighbourhood later.

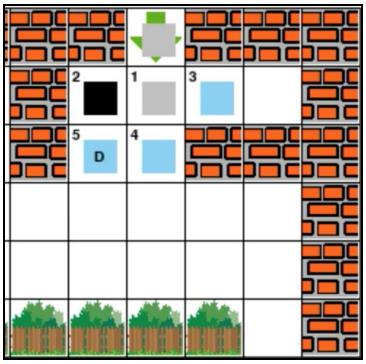




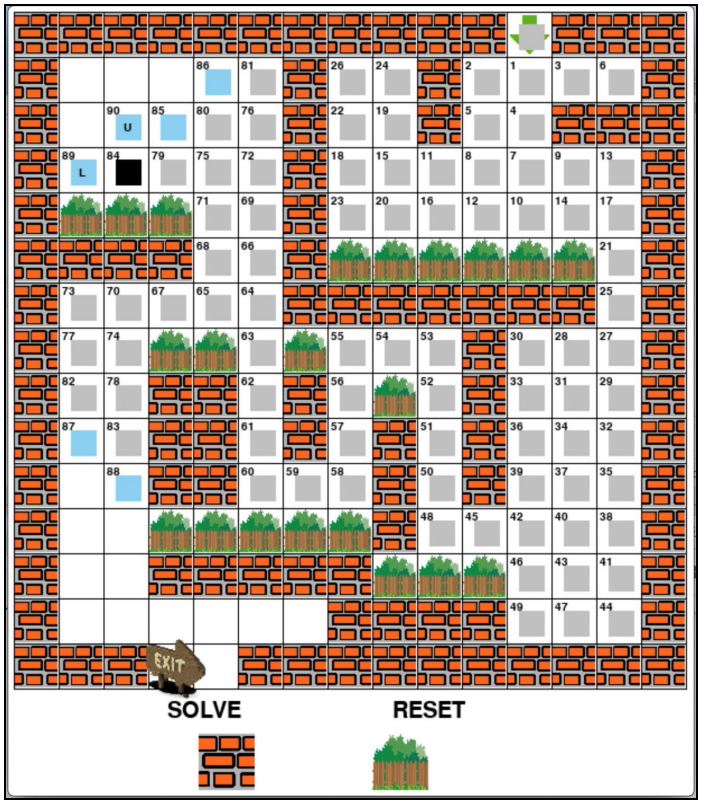
Computer retrieves location 1 from the head of the queue, and further explores its visitable neighbours. In this case, location 2 (left), 3 (right) and 4 (down) are visitable. Computer puts location 2, 3 and 4 sequentially to the tail of the queue for further neighbourhood exploration.

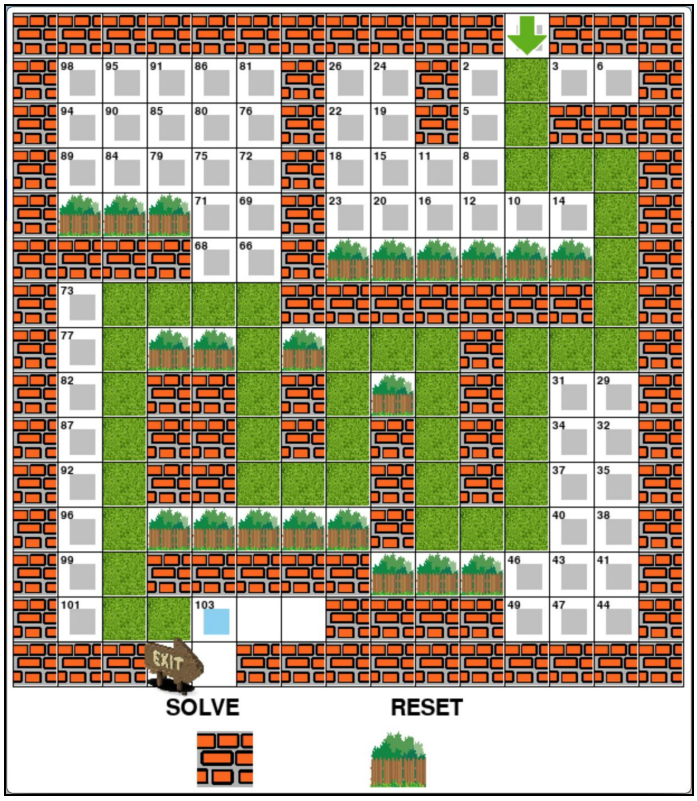


Computer retrieves location 2 from the head of the queue, and further explores its visitable neighbours, in this case location 5 (down). Computer puts location 5 to the tail of the queue for further neighbourhood exploration.



Now you get the point. After repeating above steps, here we go:



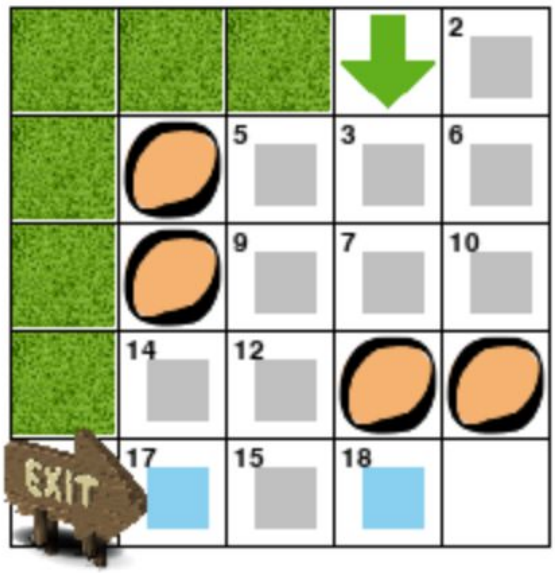


Depth First Search

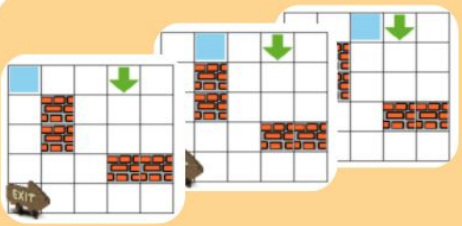
Well DFS isn't a great idea here because you're going to constantly revisit the same sub-paths, and also because you're going to have to explore ALL possible paths to find what's the shortest. Generally speaking when you have a recursive problem with duplication of work going on, you should think about dynamic programming. In this specific case though, you can use DFS, which is in fact pretty similar to what you would do with a standard DP solution for this problem.

DFS runs a much longer time to find the shortest path.

Let us use the same maze as the one solved by BFS.
[ToDo: refer to the BFS maze figure number instead of copying here]



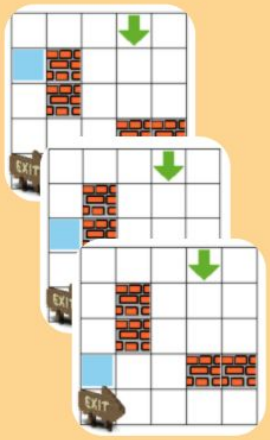
We will use wall block instead of stone to differentiate the illustrations for the two algorithms.
[ToDo: to put the illustration side by side for comparison]



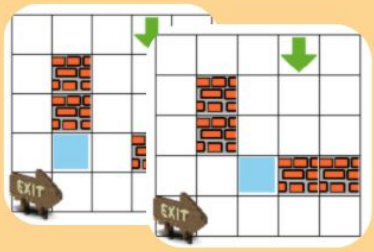
1. Start exploring Left

2. Continue Left until reaching end

<u>therefore called Depth First Search</u>



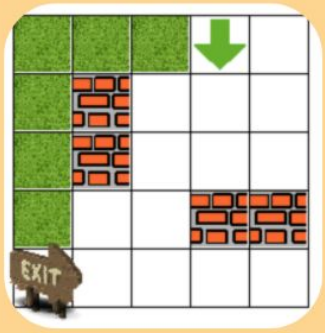
3. Continue Down
Left? invalid
Right? invalid
Up? invalid
Down? valid



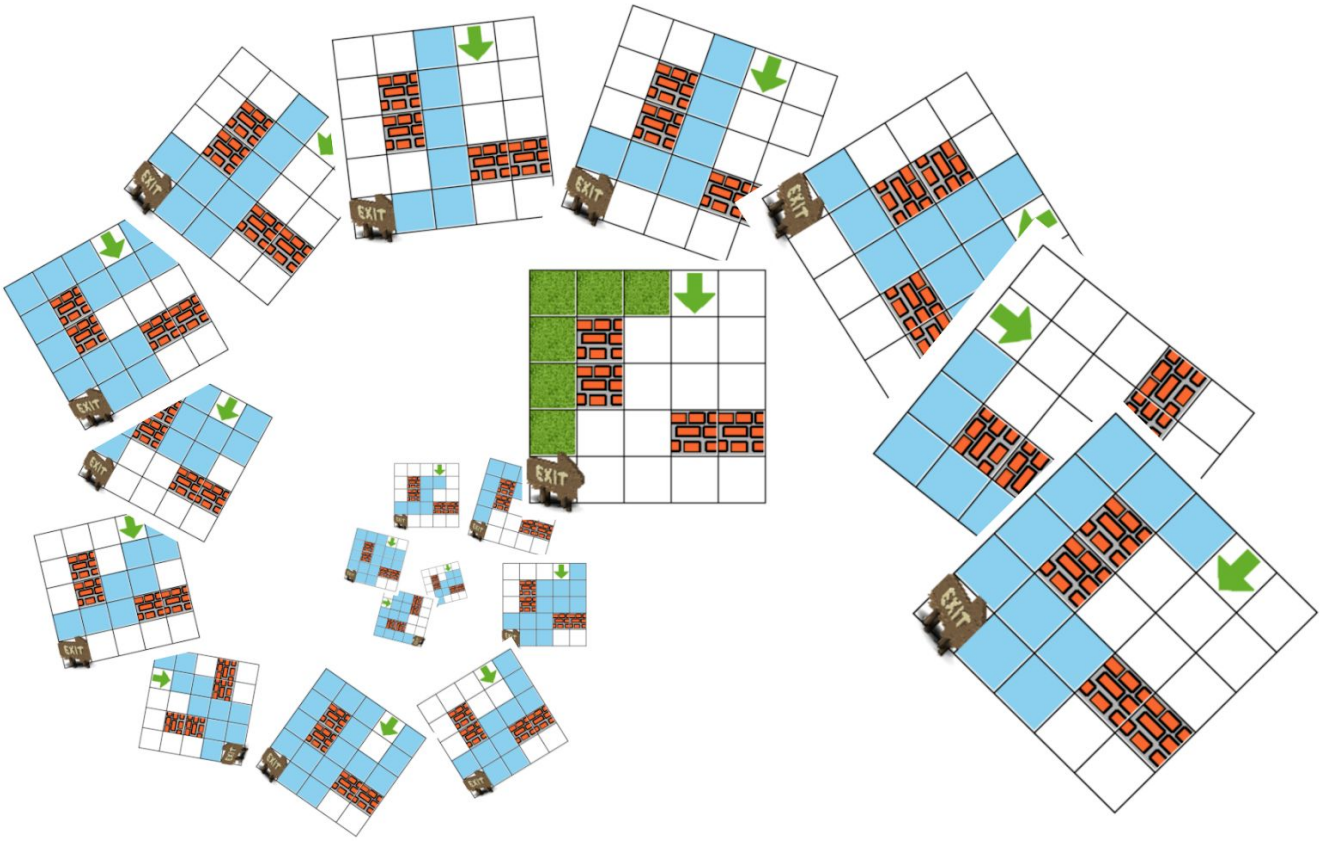
4. Continue Right
Left? invalid
Right? valid

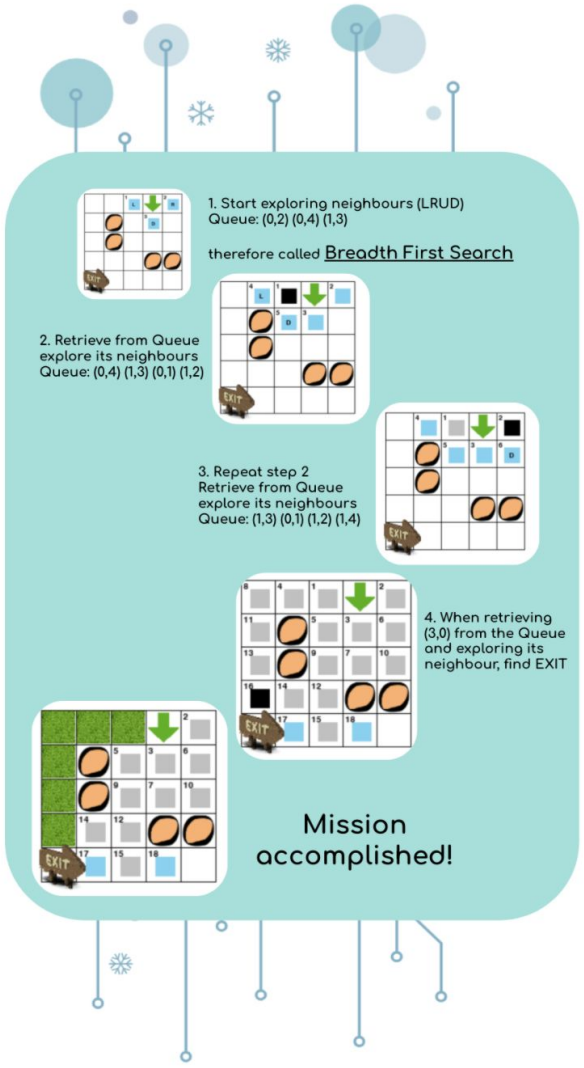
After exploring all the possible paths, select the shortest one.

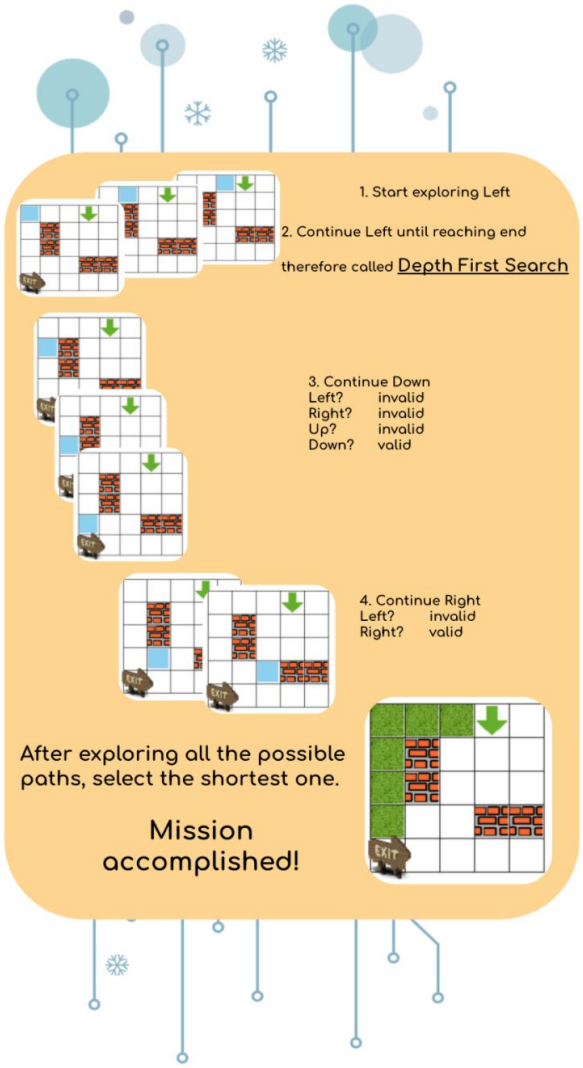
Mission accomplished!



Can you believe it? DFS needs to exhaust 64 paths to determine the shortest one! [headache]

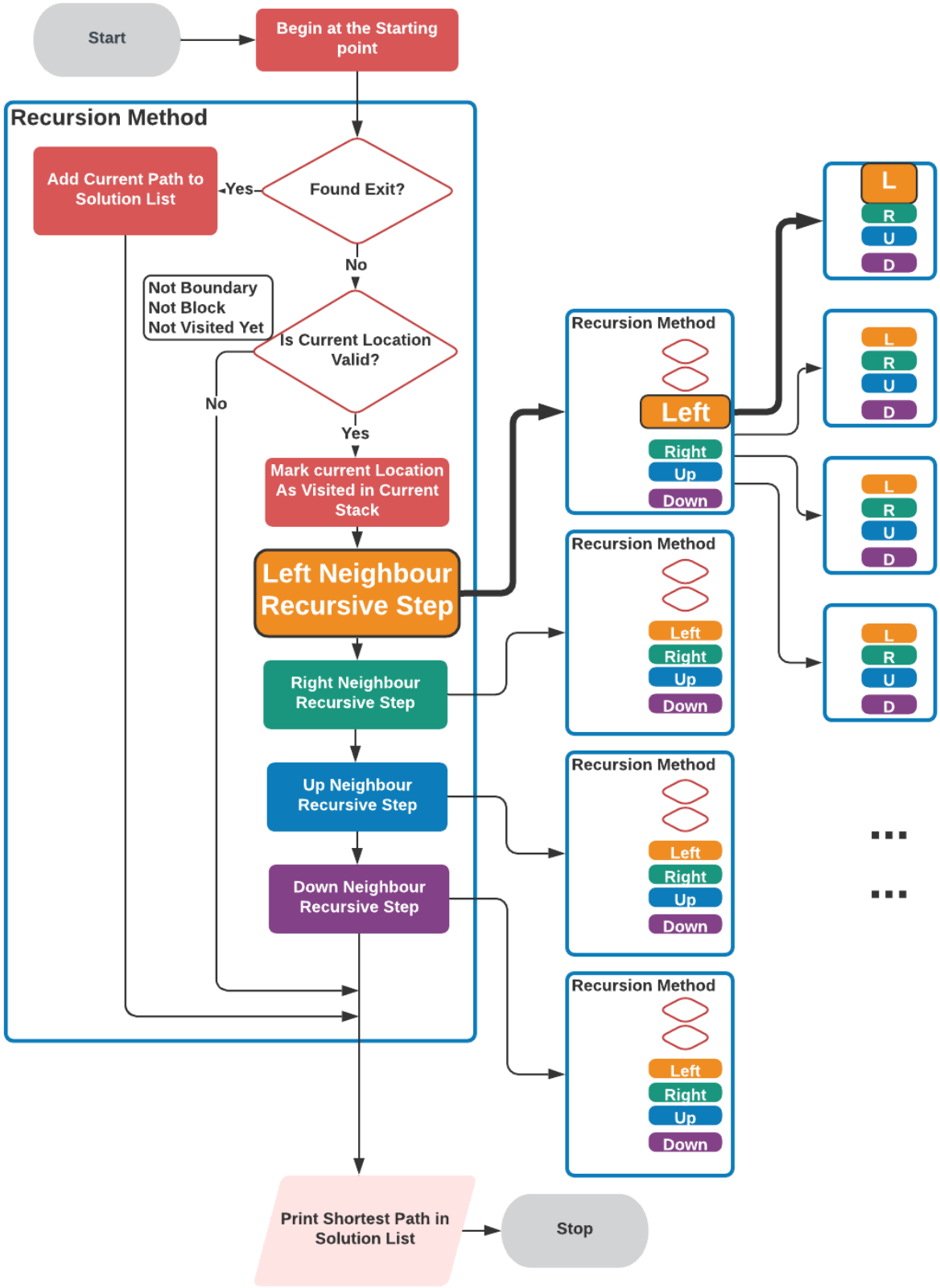






Stack and Recursion

Diagram Illustration


Start
Begin at the Starting point
Recursion Method
Add Current Path to Solution List
Yes
Found Exit?
No
Not Boundary
Not Block
Not Visited Yet
Is Current Location Valid?
No
Yes
Mark current Location As Visited in Current Stack
Left Neighbour Recursive Step
Right Neighbour Recursive Step
Up Neighbour Recursive Step
Down Neighbour Recursive Step
Print Shortest Path in Solution List
Stop
Recursion Method
Left
Right
Up
Down
L
R
U
D
...
...

Following this diagram's flow, the program will explore the left neighbour first, and then the left neighbour's left, and then left's left's left… It goes down one direction till the end of the route before exploring other directions. Therefore, this algorithm is called **Depth** First Search (DFS).

Mighty Python

DFS Implementation

mazesolver_DFS.py at
https://github.com/applepiinc/arithmaticthinking/tree/main/maze

Maze Pygame

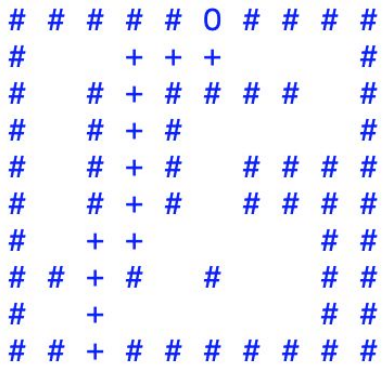
pygame_mazesolver_DFS.py and related 5 images (start, end, wall, grass, path) at
https://github.com/applepiinc/arithmaticthinking/tree/main/maze

Maze Demo

pygame_mazesolver_DFS_DemoPlayModes.py
path) at
https://github.com/applepiinc/arithmaticthinking/tree/main/maze

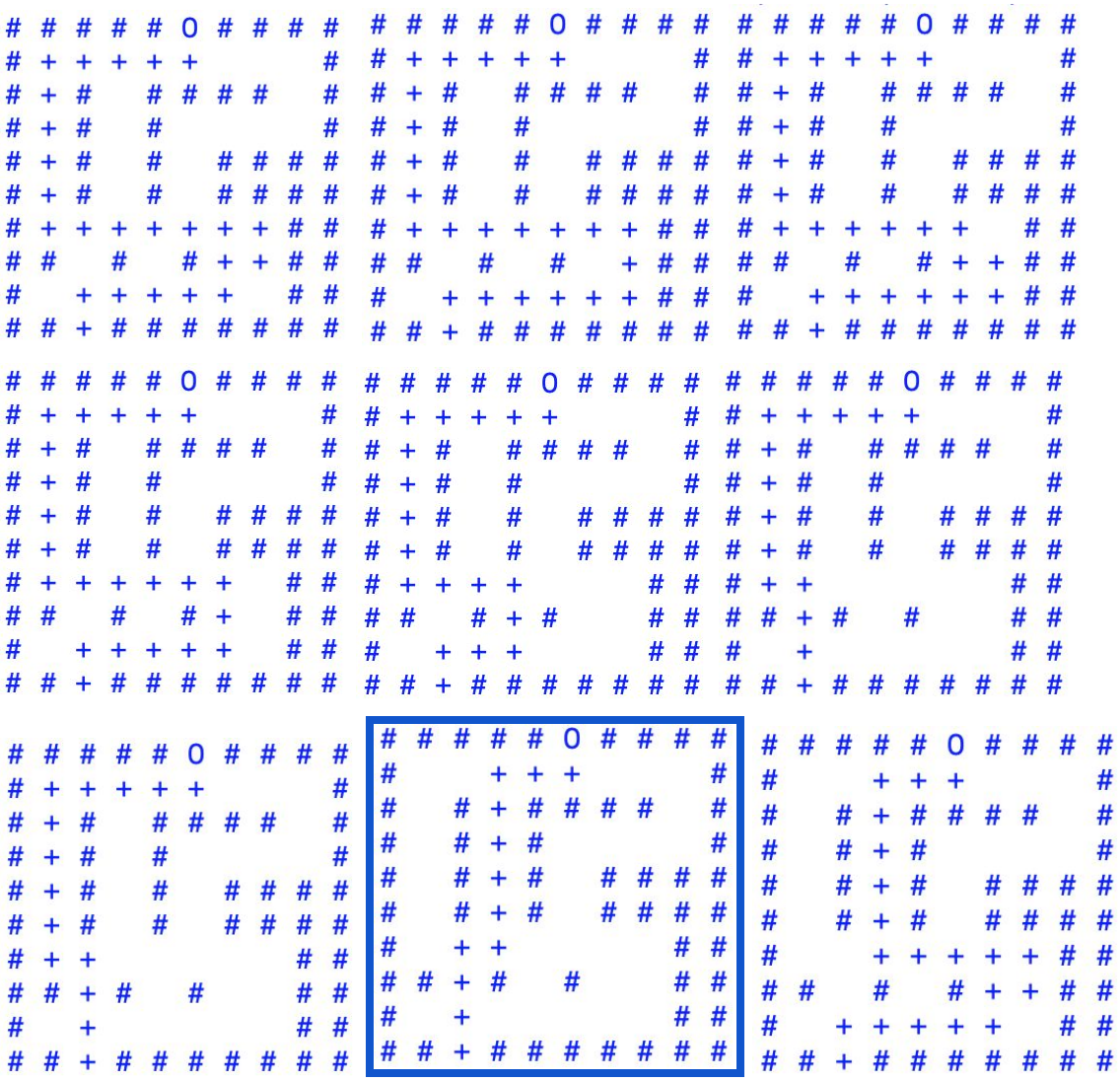
Comparison of Breadth First Search and Depth First Search

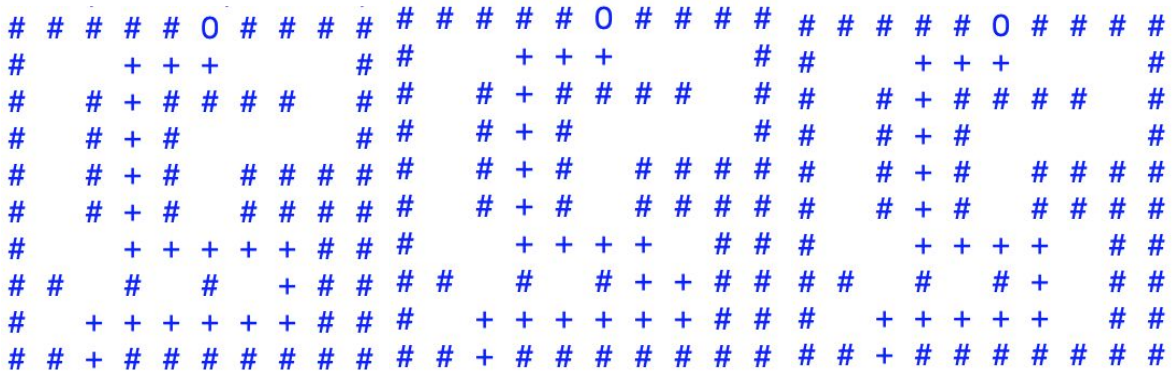
BFS



DFS

Have to exhaust all possible paths and select the shortest one.





Etc. etc. Total of 27 paths to exit.

We can not determine the shortest path until going through all the possibilities.

BFS Famous Use Cases

https://stackoverflow.com/questions/3332947/when-is-it-practical-to-use-depth-first-search-dfs-vs-breadth-first-search-bf?rq=1

- Shortest path, the shortest steps show up early in the queue. The steps can only grow further.

- **use BFS** - when you want to find the shortest path from a certain source node to a certain destination. ...
- **use DFS** - when you want to exhaust all possibilities, and check which one is the best/count the number of all possible ways.
    - Compute the shortest path to visit each city exactly once on a weighted graph. [Note: There is a cleverer Dynamic Programming (DP) solution with an exponential number of states.]
    - The N-Queue problem (if you do competitive programming). [Note: Poly-time constructive solution exists.]
- *use either **BFS or DFS*** - when you just want to check ***connectedness*** between two nodes on a given graph. (Or more generally, whether you could reach a given state to another.)

DFS - advanced applications:

- Cycle detection
- Strongly connected component / Biconnected component
- Eulerian path

Where you should not:

As you know, DFS is a recursive algorithm, you can write it using a stack, but it does not change its recursive nature, so DO NOT USE it on infinitely deep graphs. DFS is not a complete algorithm for infinitely deep graphs (it does not guarantee to reach the goal if there is any).

Even if your graph is very deep but you have the prior knowledge that your goal is a shallow one, using DFS is not a very good idea.

Also BFS needs to keep all the current nodes in the memory, so DO NOT USE it on the graphs with high branching factor, or your computer will run out of memory very quickly.

If you are facing an infinitely deep graph with a high branching factor, you can use the Iterative Deepening algorithm.

What is the advantage of DFS over BFS?
DFS uses stack data structure to process the nodes while **BFS** uses Queue data structure. **DFS** is more memory efficient since it stores a number of nodes at max the height of the **DFS** tree in the stack while **BFS** stores every adjacent node it processes in the queue.

References

Inspiration sources:
https://www.ancient.eu/Minotaur/#:~:text=In%20Greek%20mythology%2C%20the%20Minotaur%20was%20a%20monster,known%20as%20the%20Labyrinth%20to%20house%20the%20beast.

https://www.theosophical.org/publications/quest-magazine/42-publications/quest-magazine/1276-the-layrinth-a-brief-introduction-to-its-history-meaning-and-use?gclid=Cj0KCQjwyJn5BRDrARIsADZ9ykHxcSmuMcctxwq6WCucjZO4UlWWgsDZdbCw2YGTv1HHsXe8ok4kBh4aAkC1EALw_wcB

Competitive book and discount:
http://inventwithpython.com/blog/2011/08/11/recursion-explained-with-the-flood-fill-algorithm-and-zombies-and-cats/