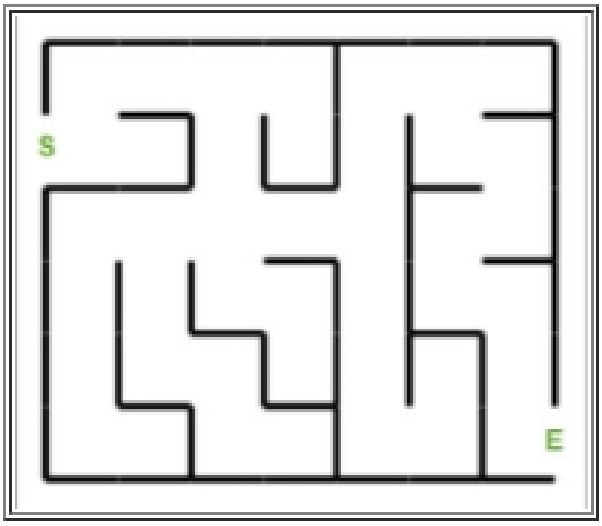
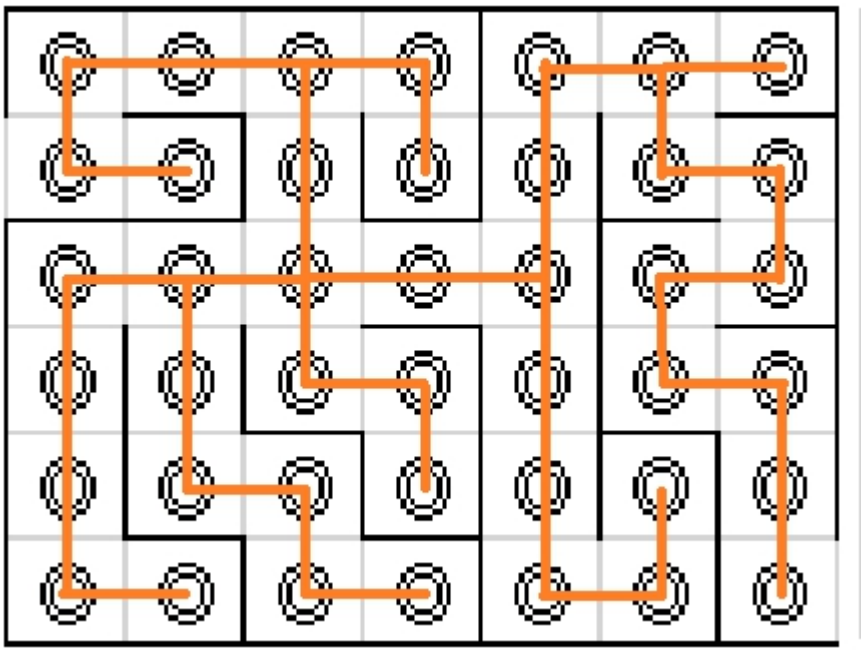
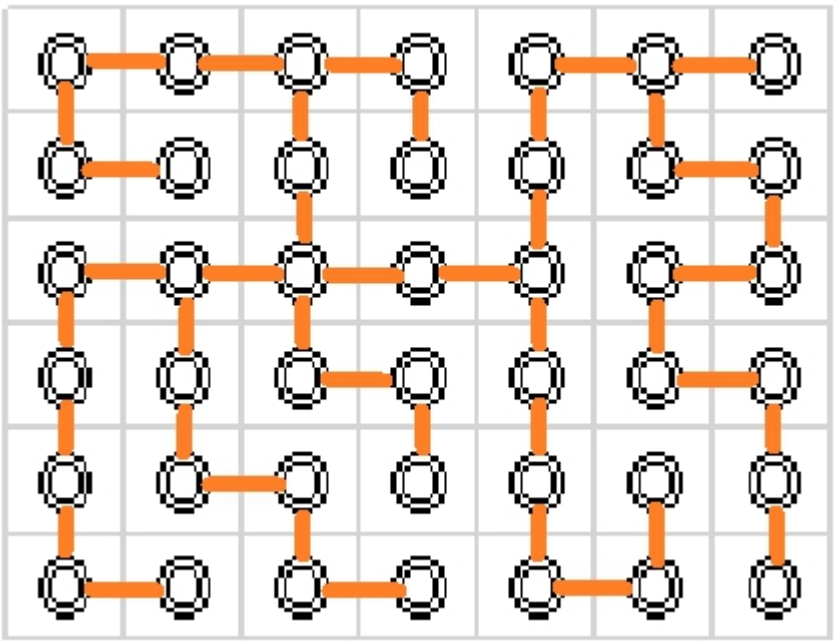
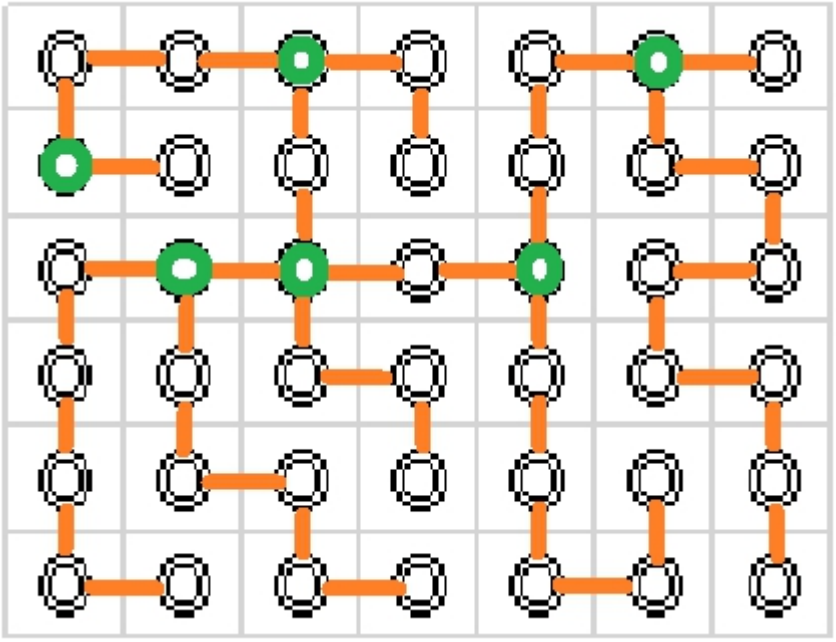
Use Bellman Ford's Algorithm to find the shortest path of a maze.

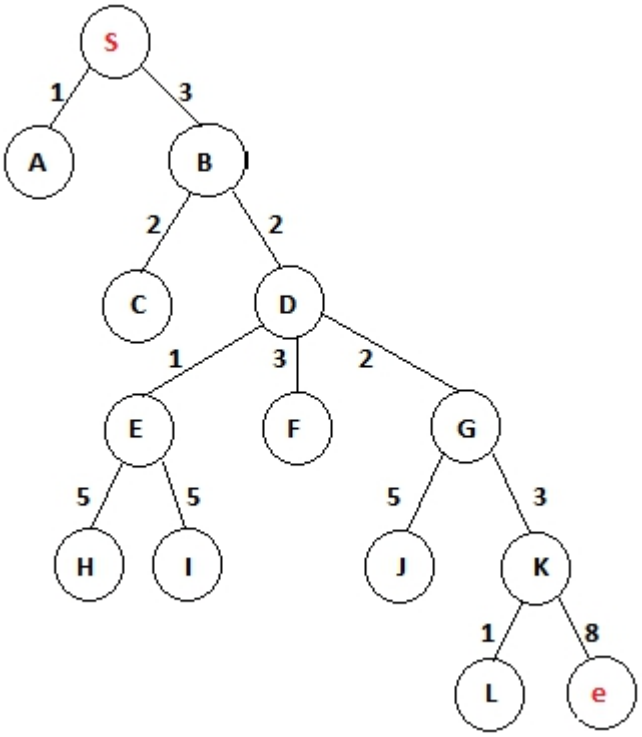
- Step 1: Similar to the previous question of finding the shortest path of the a maze. But instead of using Dijkstra's Algorithm, you will use Bellman Ford's Algorithm.











1 st iteration													
S	A	B	C	D	E	F	G	H	I	J	K	L	e
0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
0	1	3	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
0	1	3	5	5	∞	∞	∞	∞	∞	∞	∞	∞	∞
0	1	3	5	5	6	8	7	∞	∞	∞	∞	∞	∞
0	1	3	5	5	6	8	7	∞	∞	12	10	∞	∞
0	1	3	5	5	6	8	7	∞	∞	12	10	11	18
0	1	3	5	5	6	8	7	11	11	12	10	11	18

2nd iteration													
S	A	B	C	D	E	F	G	H	I	J	K	L	e
0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
0	1	3	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
0	1	3	5	5	∞	∞	∞	∞	∞	∞	∞	∞	∞
0	1	3	5	5	6	8	7	∞	∞	∞	∞	∞	∞
0	1	3	5	5	6	8	7	∞	∞	12	10	∞	∞
0	1	3	5	5	6	8	7	∞	∞	12	10	11	18
0	1	3	5	5	6	8	7	11	11	12	10	11	18

Shortest path is 18.

- Step 2: Comparing the performance of Dijkstra's Algorithm and Bellman Ford's Algorithm in solving this question by
  - Big-O comparison
  - comparing how many steps are required to find a graph that has the shortest path.
    - Note:
      - A step is defined as either comparing two numbers or replacing a number.
      - You can count how many steps for Dijkstra's Algorithm on the created table.
      - Refer this example on counting the steps for Bellman Ford's Algorithm.

```
Algorithm: Dijkstra's-Algorithm (G, w, s)
for each vertex v ∈ G.V
   v.d := ∞
   v.∏ := NIL
s.d := 0
S := Φ
Q := G.V
while Q ≠ Φ
   u := Extract-Min (Q)
   S := S U {u}
   for each vertex v ∈ G.adj[u]
      if v.d > u.d + w(u, v)
         v.d := u.d + w(u, v)
         v.∏ := u
```

Time complexity: O(E logV) 15 steps

```
Bellman-Ford-Algorithm (G, w, s)
for each vertex v ∈ G.V     ==> O(V)
   v.d := ∞
   v.∏ := NIL
s.d := 0
for i = 1 to |G.V| - 1          ==> O(V)
   for each edge (u, v) ∈ G.E ==> O(E)
      if v.d > u.d + w(u, v)
         v.d := u.d +w(u, v)
         v.∏ := u        ====>O(V)*O(E) = O(VE)
for each edge (u, v) ∈ G.E     ==>O(E)
   if v.d > u.d + w(u, v)
      return FALSE
return TRUE
```

===> O(V)+ O(VE) + O(E)

===> Time complexity: O(VE) 14 steps*2 iterations