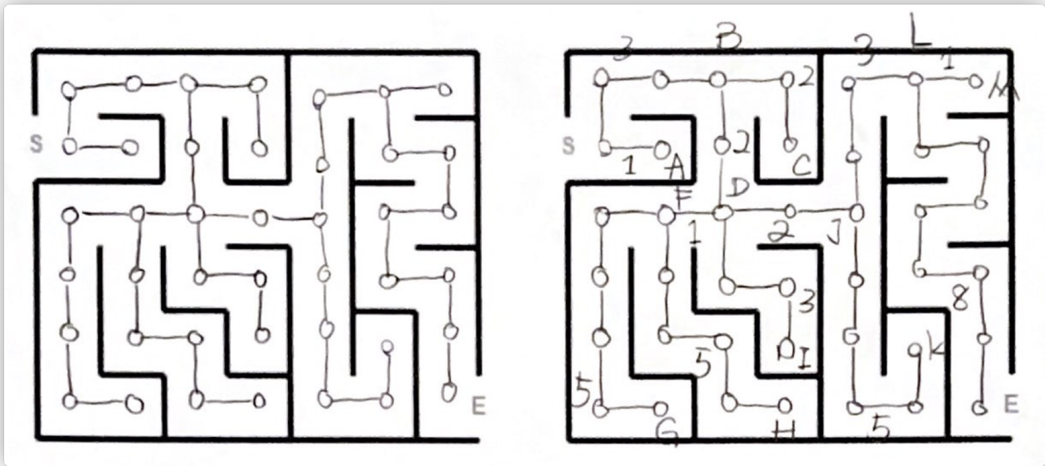
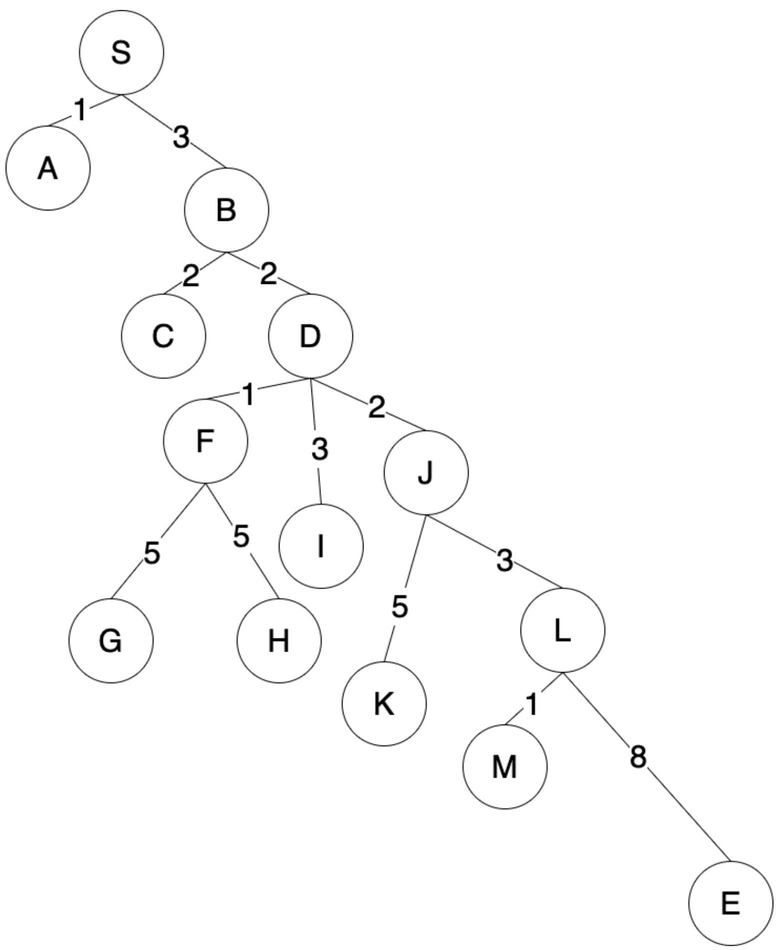
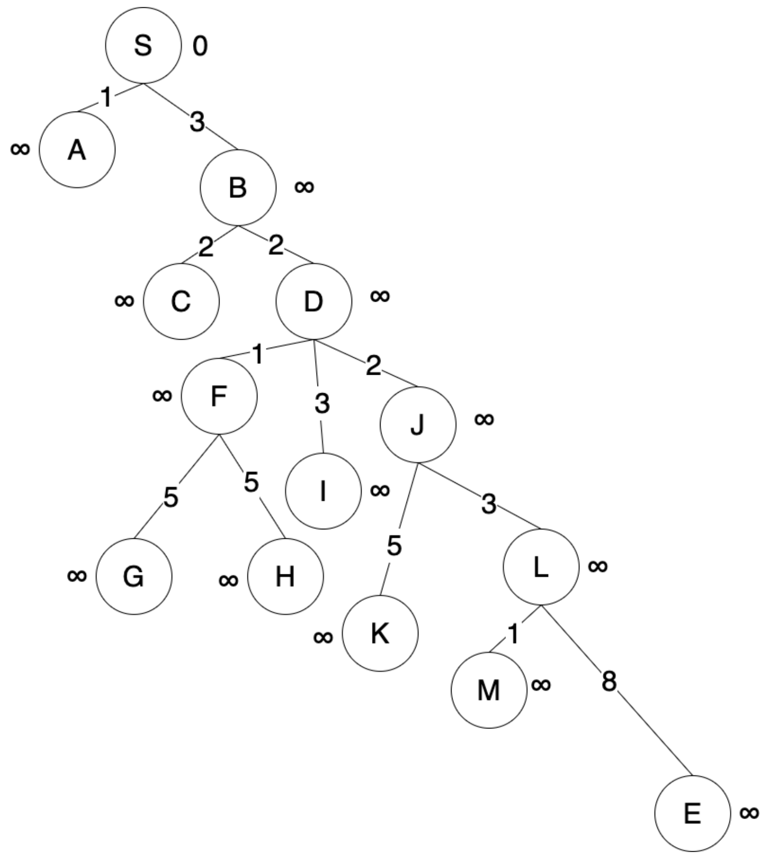
Use Bellman Ford's Algorithm to find the <u>shortest path</u> of a maze.



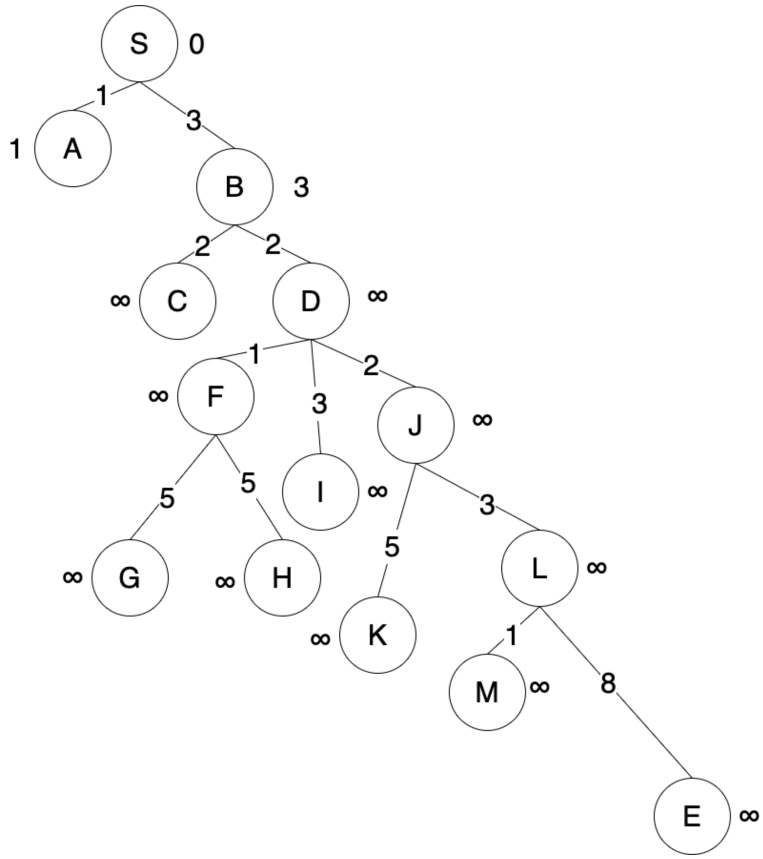


Using Bellman Ford Algorithm

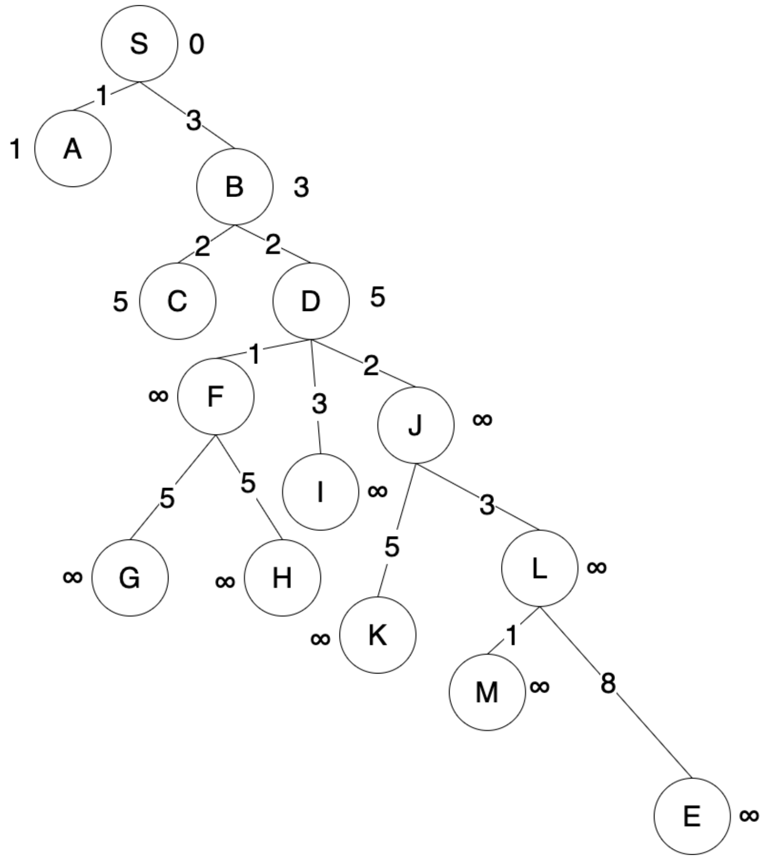
Step 0: At the time of initialization, all of the vertices except the source are marked by ∞ and the source is marked by 0.



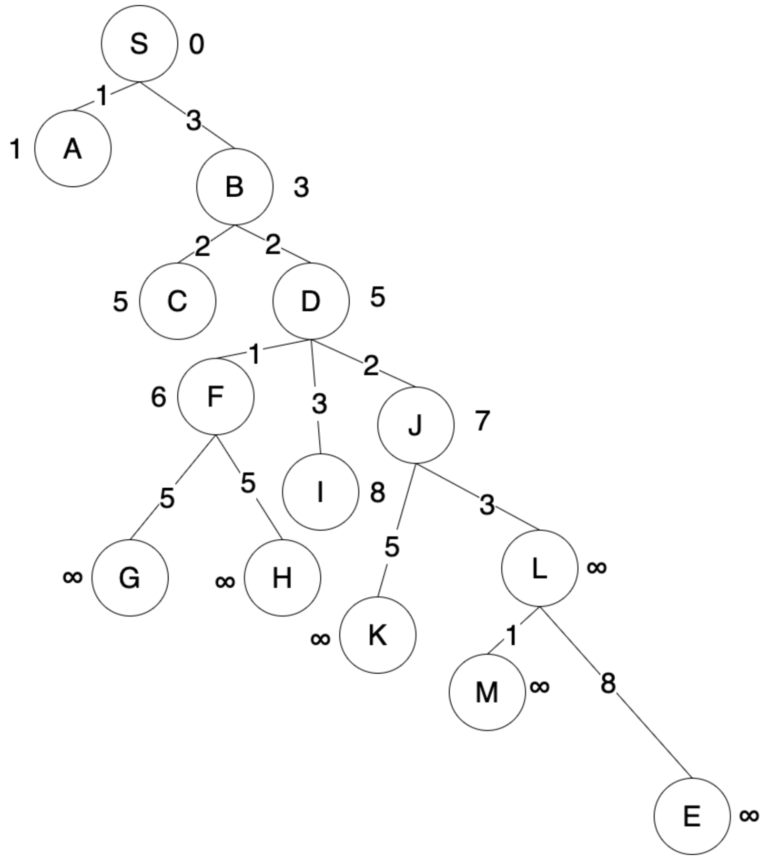
Step 1: In the first step, all the vertices which are reachable from the source S, are updated by minimum cost. Hence, vertices A and B are updated.



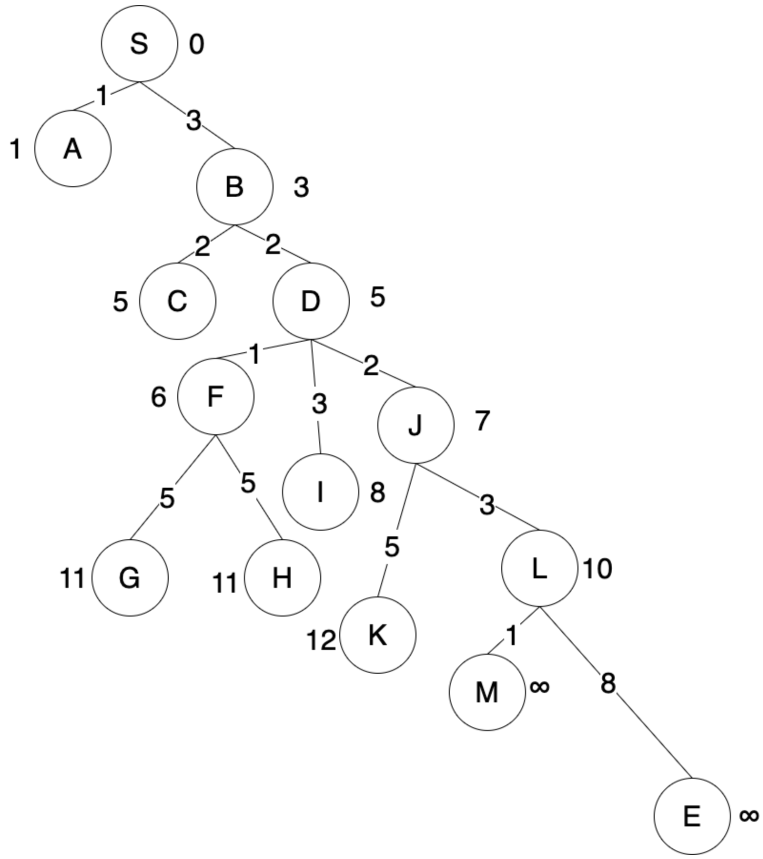
Step 2: In the next step, all the vertices which are reachable from A and B are updated by minimum cost. Thus, vertices C and D are updated.



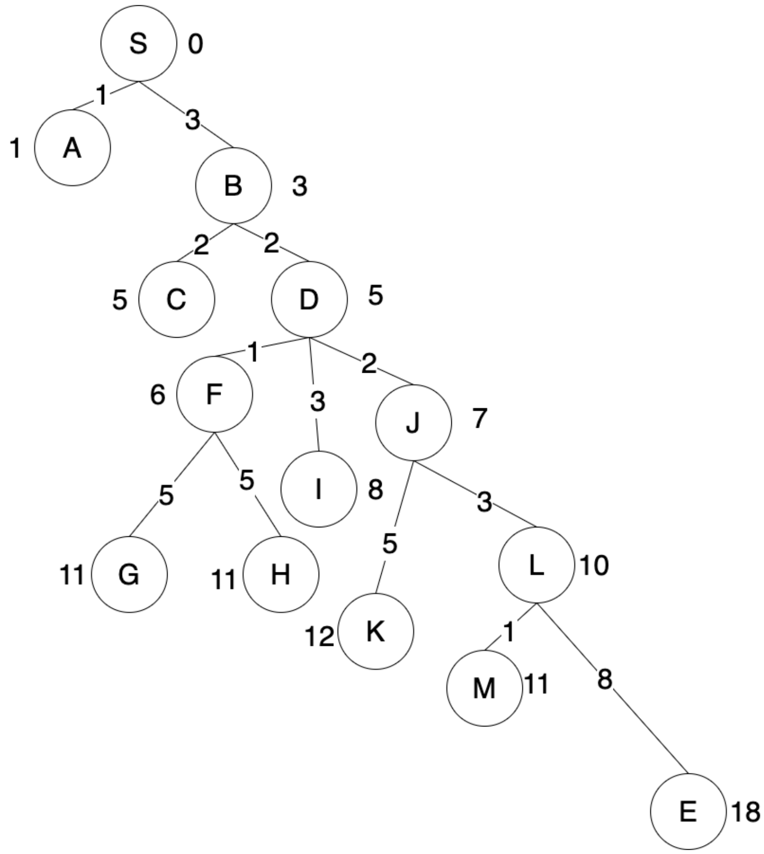
Step 3: Following the same logic, all the vertices which are reachable from C and D are updated by minimum cost. Thus, vertices F, I and J are updated.



Step 4: Following the same logic, all the vertices which are reachable from F, I and J are updated by minimum cost. Thus, vertices G, H, K and L are updated.



Step 5: Following the same logic, all the vertices which are reachable from G, H, K and L are updated by minimum cost. Thus, vertices M and E are updated.



Step 6:

Hence, the minimum distance between vertex S and vertex E is 18. Based on the predecessor information, the path is S → B → D → J → L → E.

Comparing the performance of Dijkstra's Algorithm and Bellman Ford Algorithm in solving this question

```
Algorithm: Dijkstra's-Algorithm (G, w, s)
for each vertex v Є G.V  // Initial values takes O(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 Є G.adj[u] // Through every vertex takes O(E)
      if v.d > u.d + w(u, v)
         v.d := u.d + w(u, v)
         v.∏ := u
```

Dijkstra's Algorithm:

If extract min function is implemented using linear search, the complexity of this algorithm is O(V^2 + E).

If extract min function use min-heap search, it will takes O(logV) time which makes the final time complexity become O(VlogV + E)

```
Bellman-Ford-Algorithm (G, w, s)
for each vertex v Є G.V  // Initial values takes O(V)
   v.d := ∞
   v.∏ := NIL
s.d := 0
for i = 1 to |G.V| - 1  // Nested loop takes O(VE)
   for each edge (u, v) Є G.E
      if v.d > u.d + w(u, v)
         v.d := u.d +w(u, v)
         v.∏ := u
for each edge (u, v) Є G.E // Through every edge takes O(E)
   if v.d > u.d + w(u, v)
      return FALSE
return TRUE
```

Bellman Ford Algorithm: O(V+E+VE) = O(VE)

Comparing the steps

Bellman Ford Algorithm: Needs two cycles, 26 steps.

Dijkstra's Algorithm: 11 steps

	Initial	step1(S)	step2(S,A,B)	step3(S,A,B,C)	step4(S,A,B,C,D)	step5(S,A,B,C,D,F)
Vertex	S	S→A	S→B	B→C	B→D	D→F
S	0	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~
A	∞	1	~~1~~	~~1~~	~~1~~	~~1~~
B	∞	3	3	3	3	~~3~~
C	∞	∞	∞	5	~~5~~	~~5~~
D	∞	∞	∞	5	5	5
F	∞	∞	∞	∞	∞	6
G	∞	∞	∞	∞	∞	∞
H	∞	∞	∞	∞	∞	∞
I	∞	∞	∞	∞	∞	8
J	∞	∞	∞	∞	∞	7
K	∞	∞	∞	∞	∞	∞
L	∞	∞	∞	∞	∞	∞
M	∞	∞	∞	∞	∞	∞
E	∞	∞	∞	∞	∞	∞

	step6(S,A,B,C,D,F,G)	step7(S,A,B,C,D,F,G,H)	step8(S,A,B,C,D,F,G,J)	step9(S,A,B,C,D,F,G,J,L)	step10(S,A,B,C,D,F,G,J,L,M)	step11
Vertex	F→G	F→H	D→J	J→L	L→M	L→E
S	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~
A	~~1~~	~~1~~	~~1~~	~~1~~	~~1~~	~~1~~
B	~~3~~	~~3~~	~~3~~	~~3~~	~~3~~	~~3~~
C	~~5~~	~~5~~	~~5~~	~~5~~	~~5~~	~~5~~
D	~~5~~	~~5~~	5	5	5	5
F	6	6	~~6~~	~~6~~	~~6~~	~~6~~
G	11	~~11~~	~~11~~	~~11~~	~~11~~	~~11~~
H	∞	11	~~11~~	~~11~~	~~11~~	~~11~~
I	8	8	8	8	8	8
J	7	7	7	7	~~7~~	~~7~~
K	∞	∞	∞	12	12	12

L	∞	∞	∞	10	10	10
M	∞	∞	∞	∞	11	~~11~~
E	∞	∞	∞	∞	18	18

After comparing, when data set is large, the Dijkstra's Algorithm usually is faster than the Bellman Ford Algorithm.

Google Slide: https://docs.google.com/presentation/d/1d07YLFiVSCvRhVJHAVbLu16vdYXcJHwEDWKM6qU9gp0/edit?usp=sharing

GitHub: https://github.com/blueandhack/Maze-Project