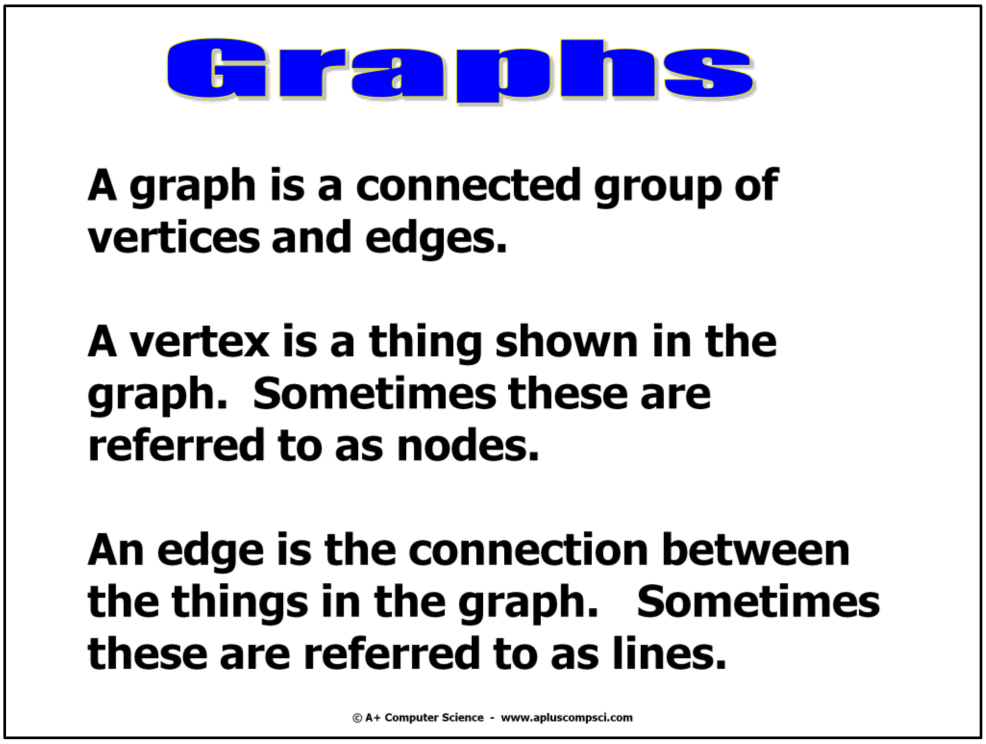
Graphs

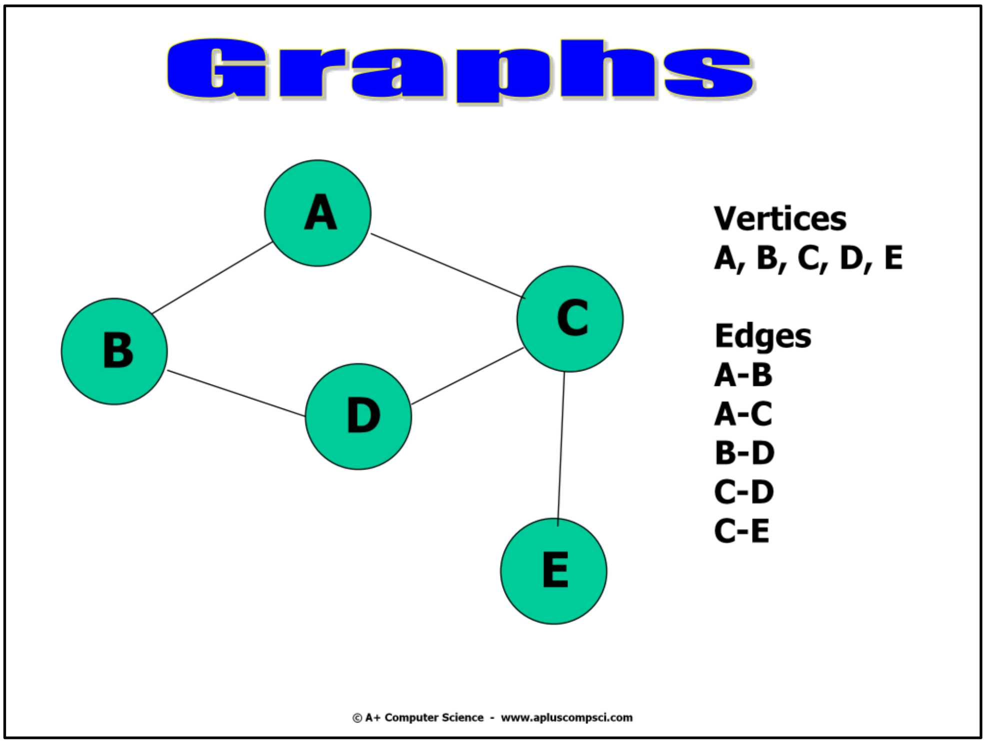


Graphs

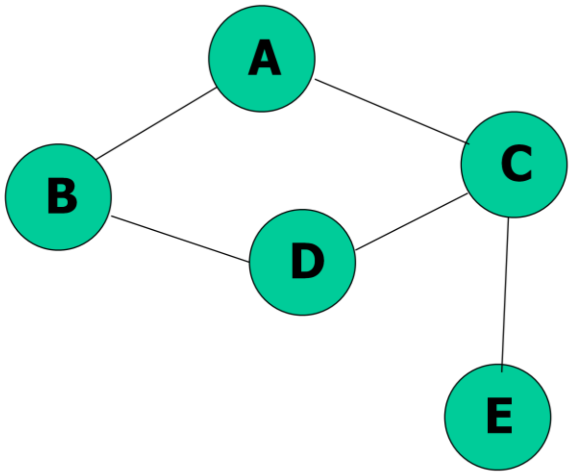
A graph is a connected group of vertices and edges.

A vertex is a thing shown in the graph. Sometimes these are referred to as nodes.

An edge is the connection between the things in the graph.   Sometimes these are referred to as lines.

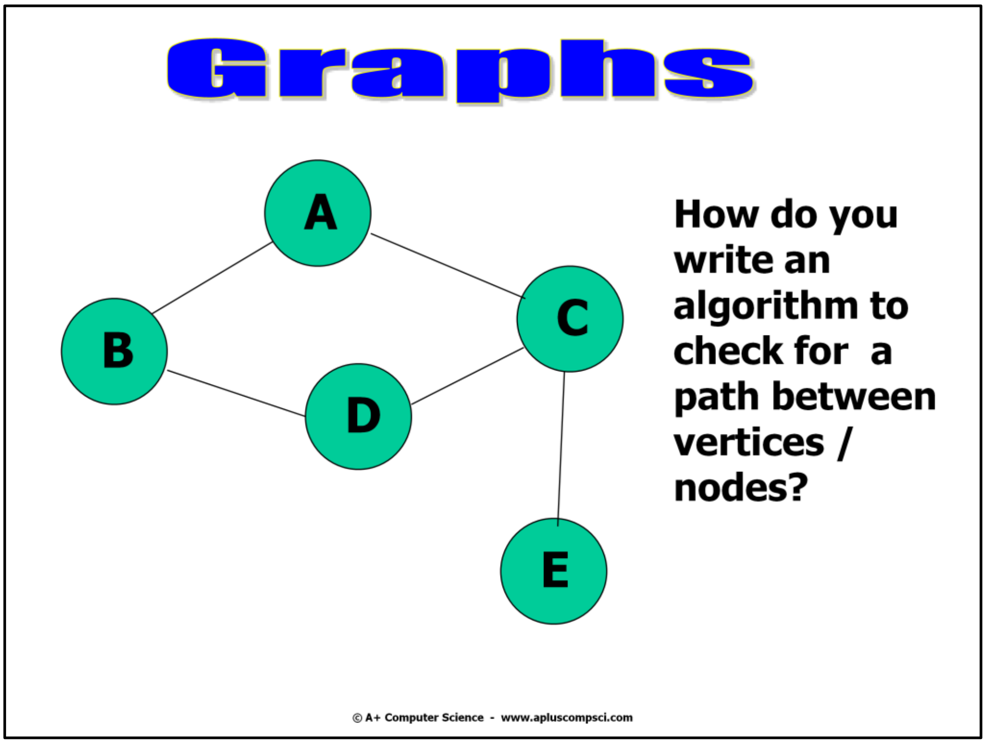


Graphs

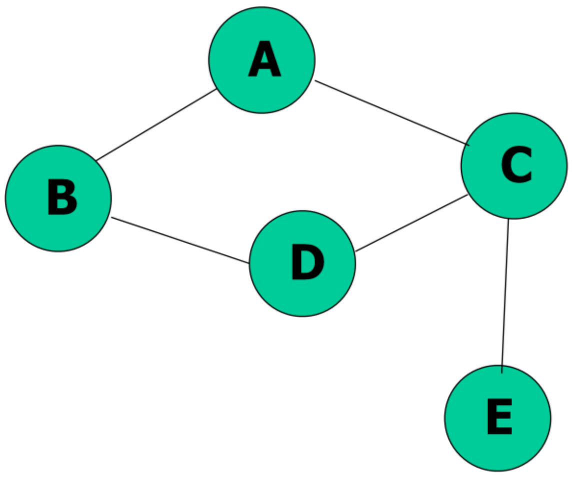


Vertices
A, B, C, D, E

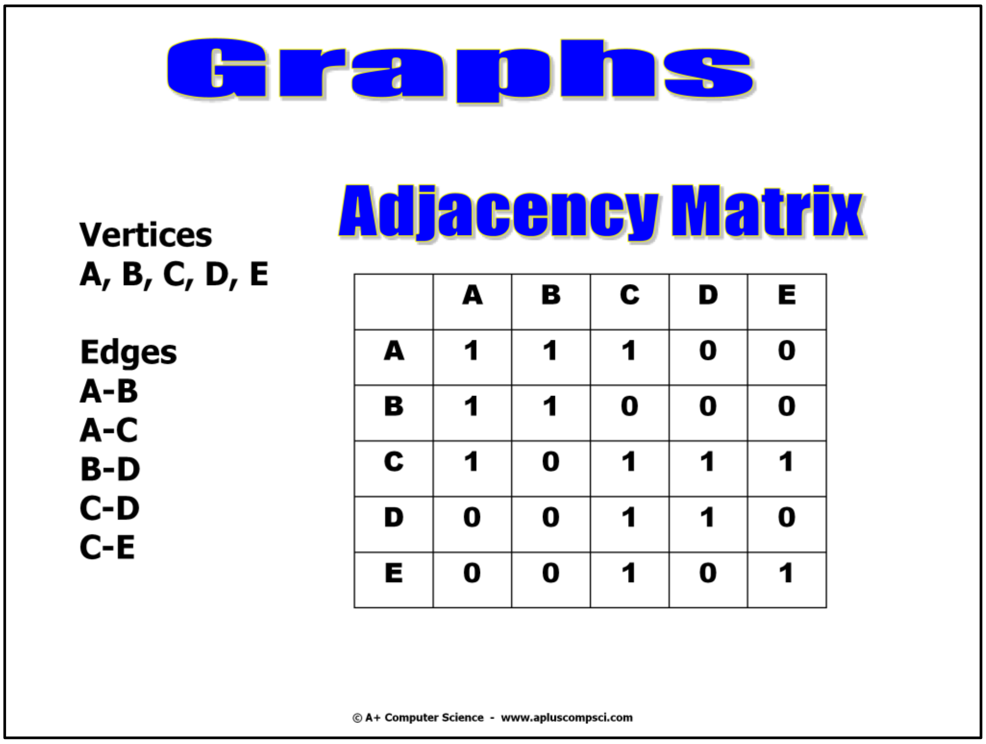
Edges
A-B
A-C
B-D
C-D
C-E



Graphs



How do you write an algorithm to check for a path between vertices / nodes?



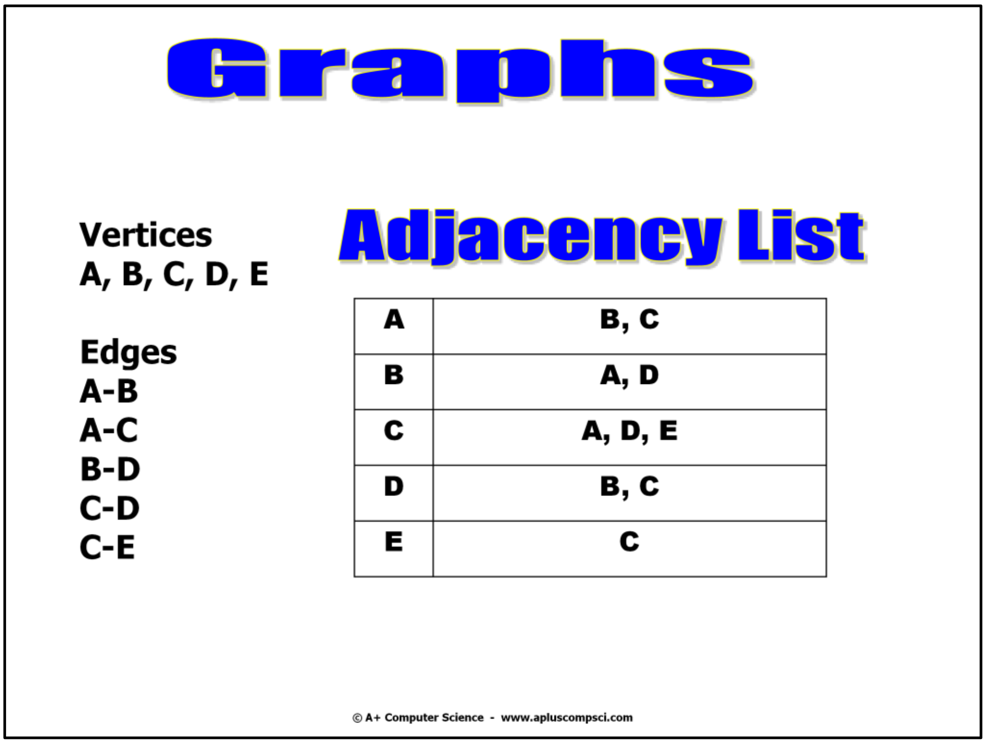
Graphs

Adjacency Matrix

Vertices
A, B, C, D, E

Edges
A-B
A-C
B-D
C-D
C-E

	A	B	C	D	E
A	1	1	1	0	0
B	1	1	0	0	0
C	1	0	1	1	1
D	0	0	1	1	0
E	0	0	1	0	1



Graphs

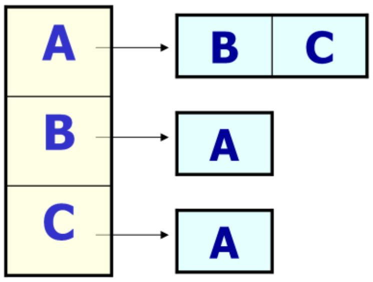
Vertices
A, B, C, D, E

Edges
A-B
A-C
B-D
C-D
C-E

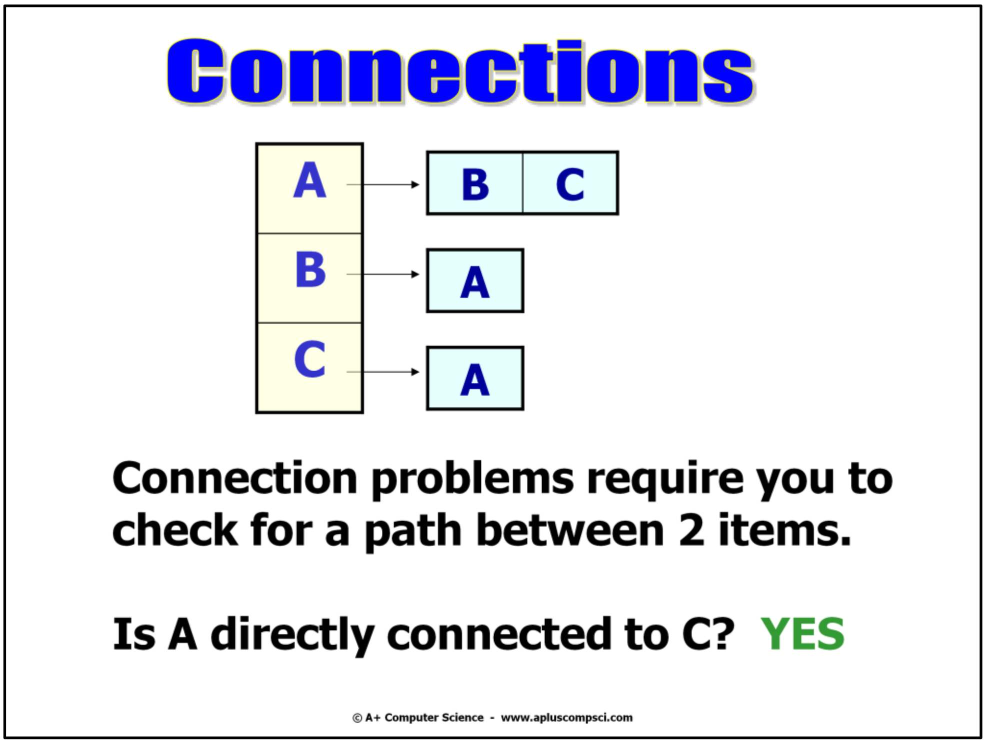
Adjacency List

A	B, C
B	A, D
C	A, D, E
D	B, C
E	C

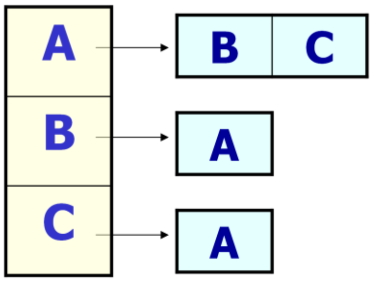
Connections



Connection problems are also graph problems as you check vertices and search edges to see if a path exists between two vertices.

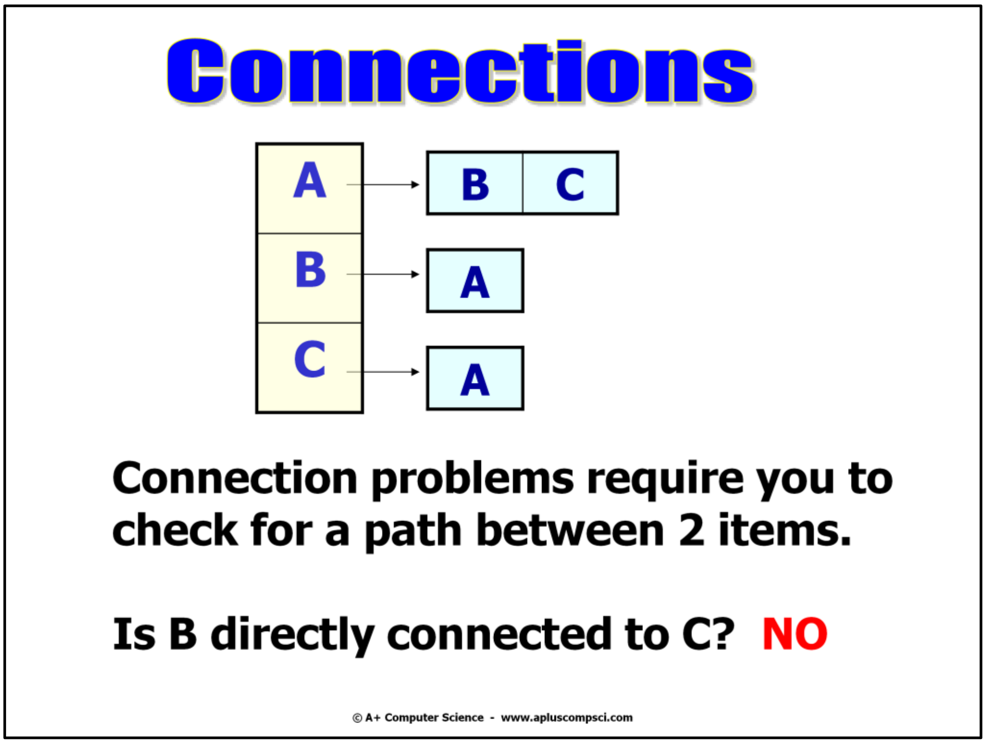


Connections

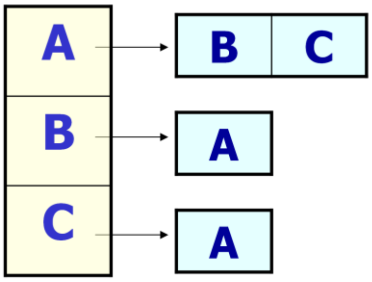


Connection problems require you to check for a path between 2 items.

Is A directly connected to C? YES



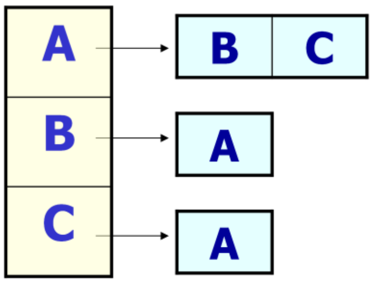
Connections



Connection problems require you to check for a path between 2 items.

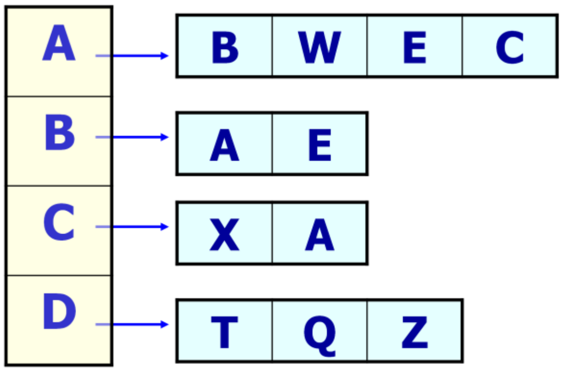
Is B directly connected to C? NO

Connections

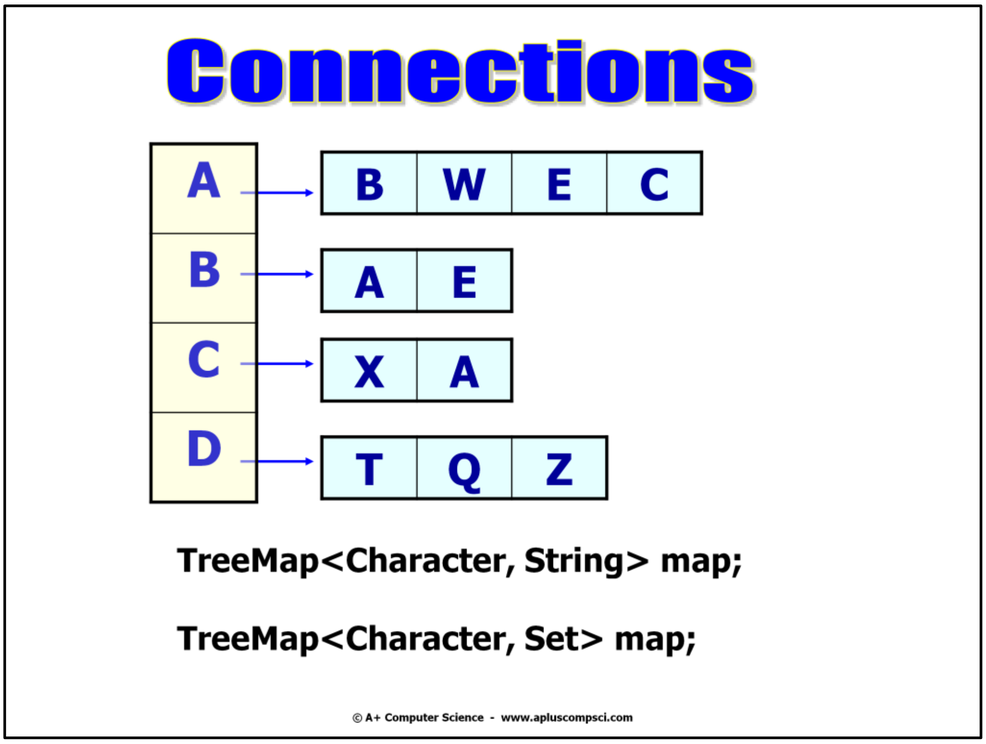


A is directly connected to B and C.
B is directly connected to A.
C is directly connected to A.
Is B connected to C? YES

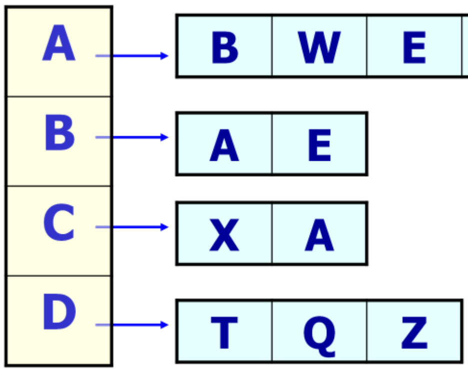
Connections



Is A connected to X? YES
Is A connected to Q? NO

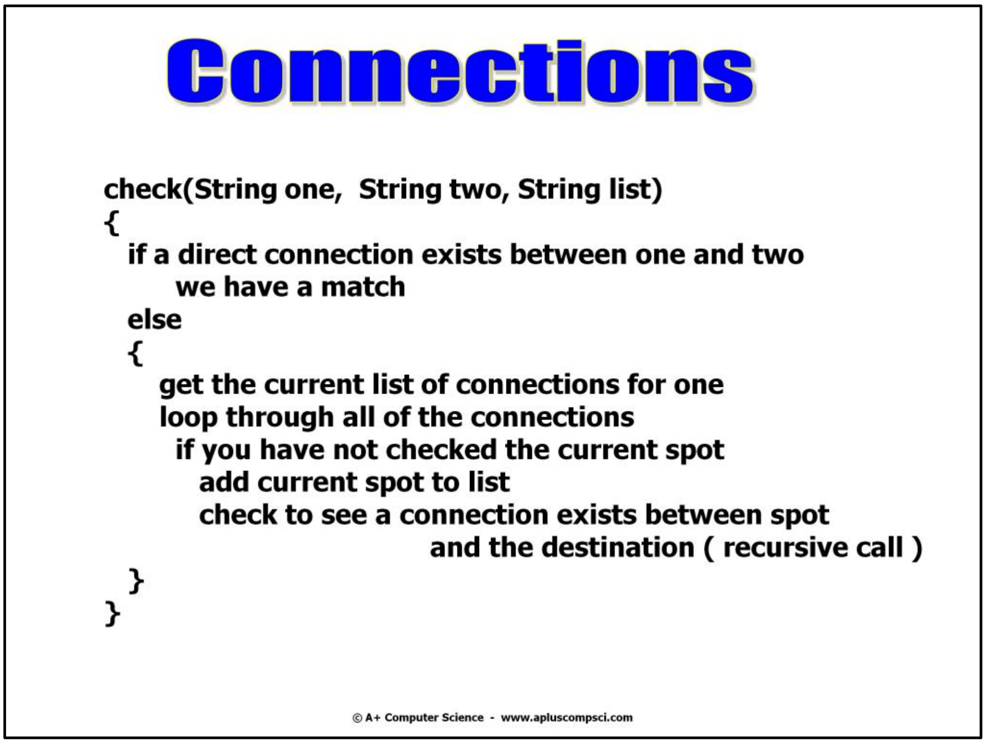


Connections



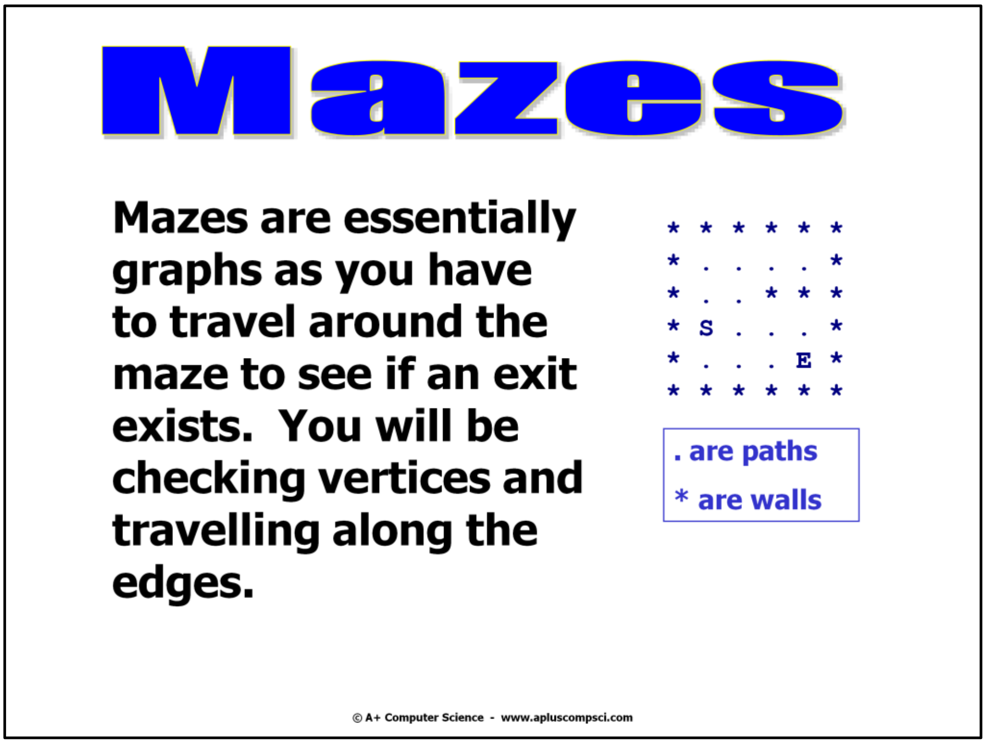
```
TreeMap<Character, String> map;

TreeMap<Character, Set> map;
```



Connections

```
check(String one,  String two, String list)
{
  if a direct connection exists between one and two
      we have a match
  else
  {
     get the current list of connections for one
     loop through all of the connections
       if you have not checked the current spot
         add current spot to list
         check to see a connection exists between spot
                               and the destination ( recursive call )
  }
}
```



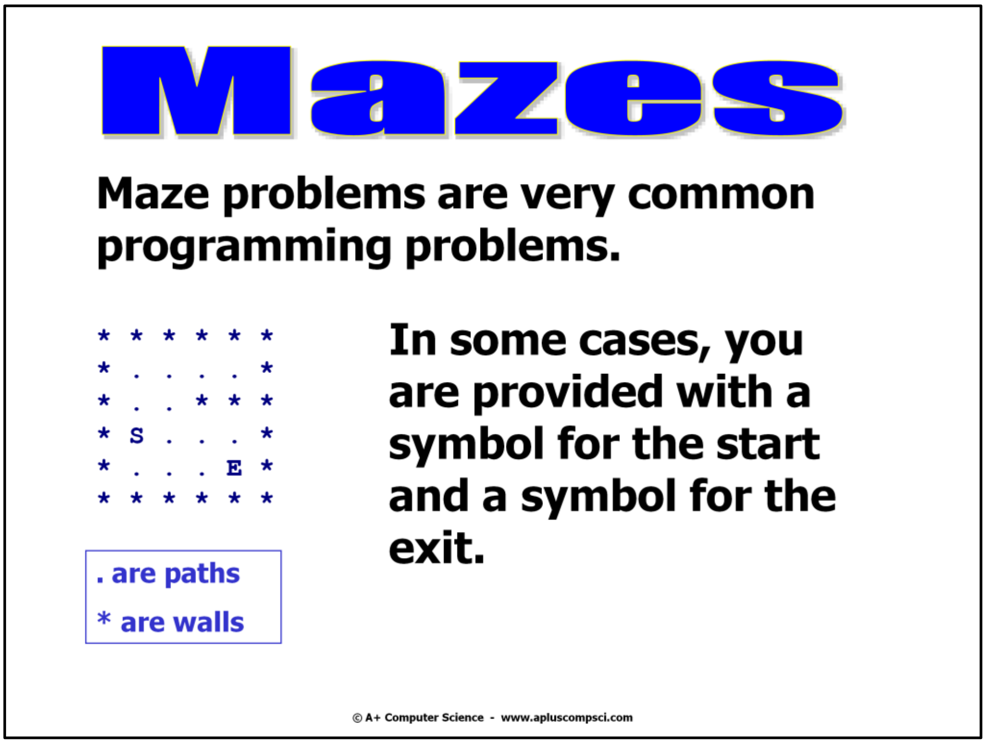
Mazes

Mazes are essentially graphs as you have to travel around the maze to see if an exit exists. You will be checking vertices and travelling along the edges.

```
* * * * * *
* . . . . *
* . . * * *
* S . . . *
* . . . E *
* * * * * *
```

. are paths

* are walls



Mazes

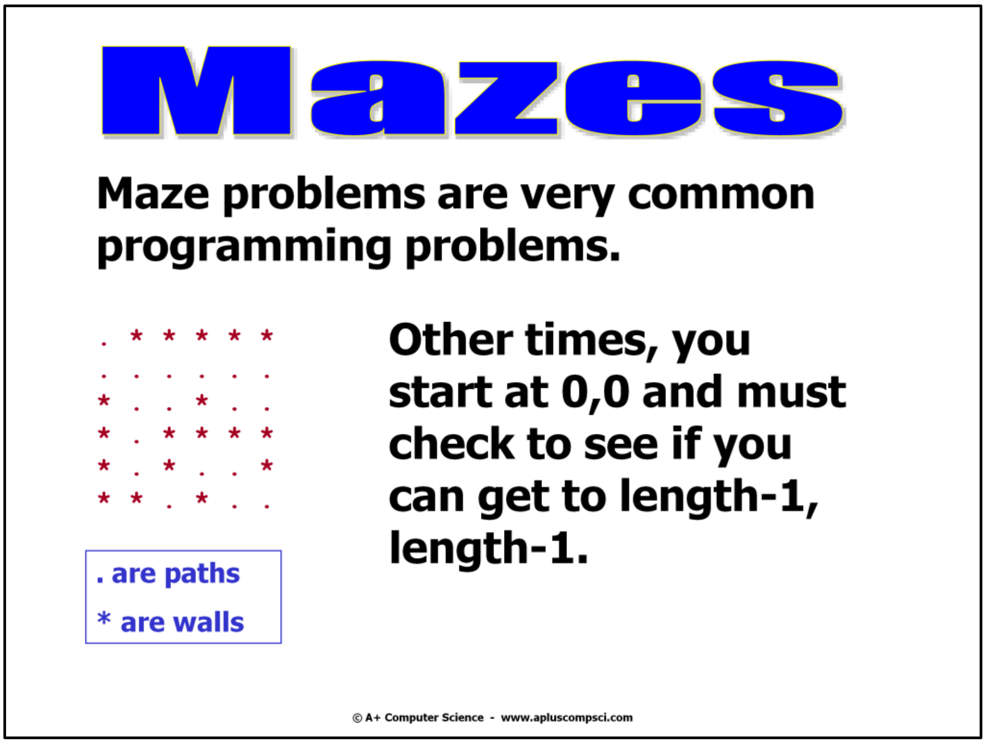
Maze problems are very common programming problems.

```
* * * * * *
* . . . . *
* . . * * *
* S . . . *
* . . . E *
* * * * * *
```

. are paths

* are walls

In some cases, you are provided with a symbol for the start and a symbol for the exit.



Mazes

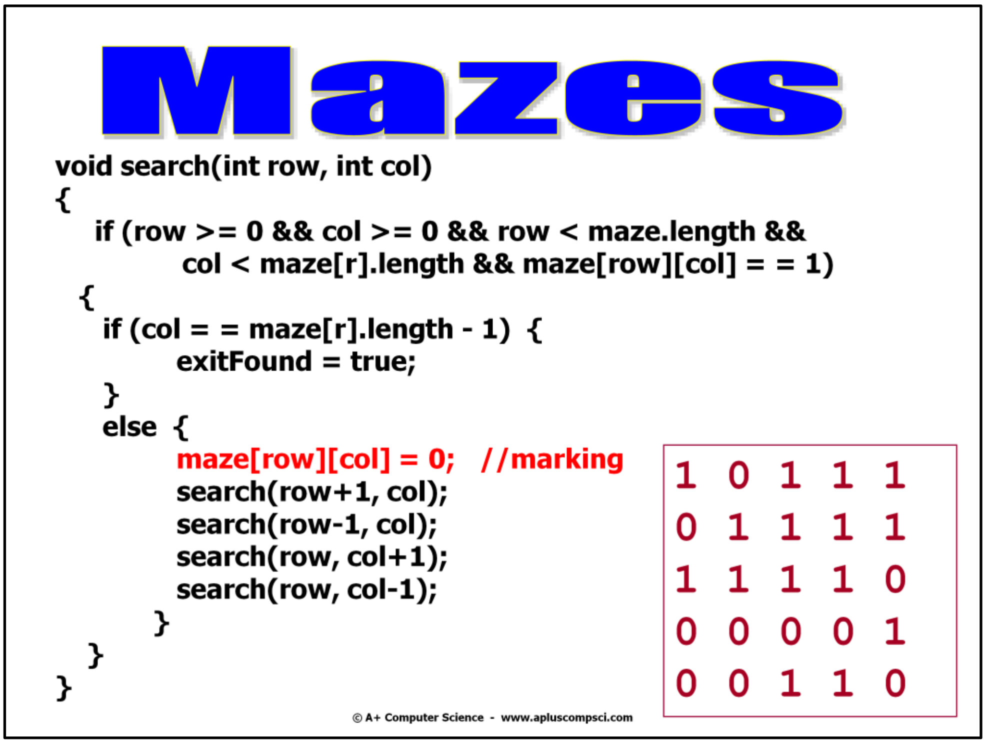
Maze problems are very common programming problems.

```
. * * * * *
. . . . . .
* . . * . .
* . * * * *
* . * . . *
* * . * . .
```

. are paths

* are walls

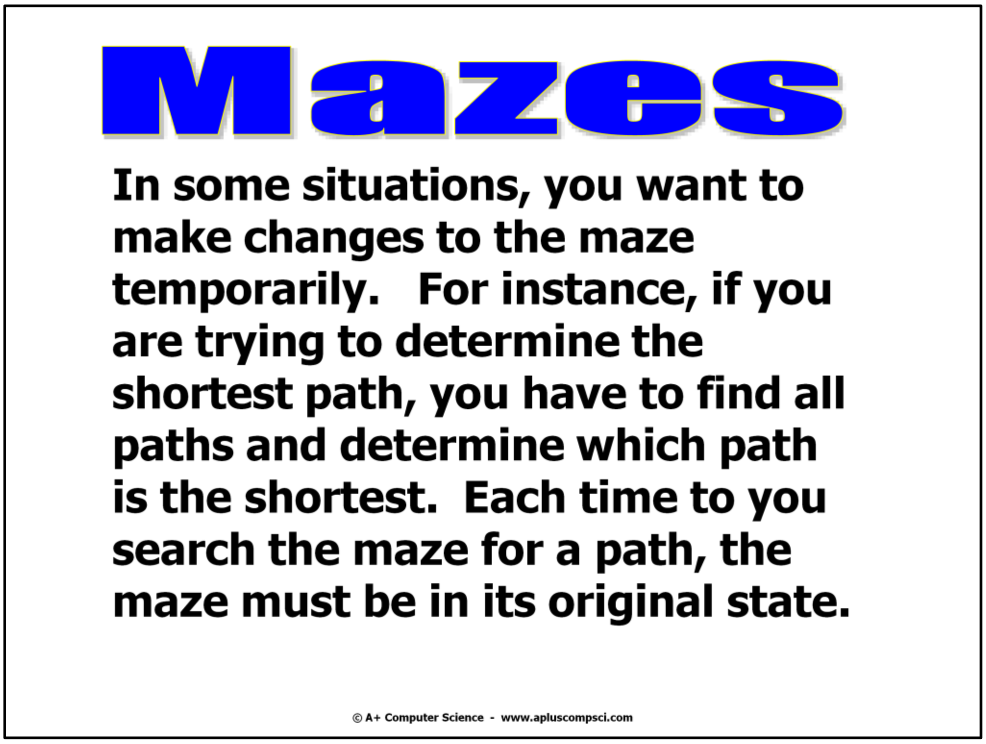
Other times, you start at 0,0 and must check to see if you can get to length-1, length-1.



Mazes

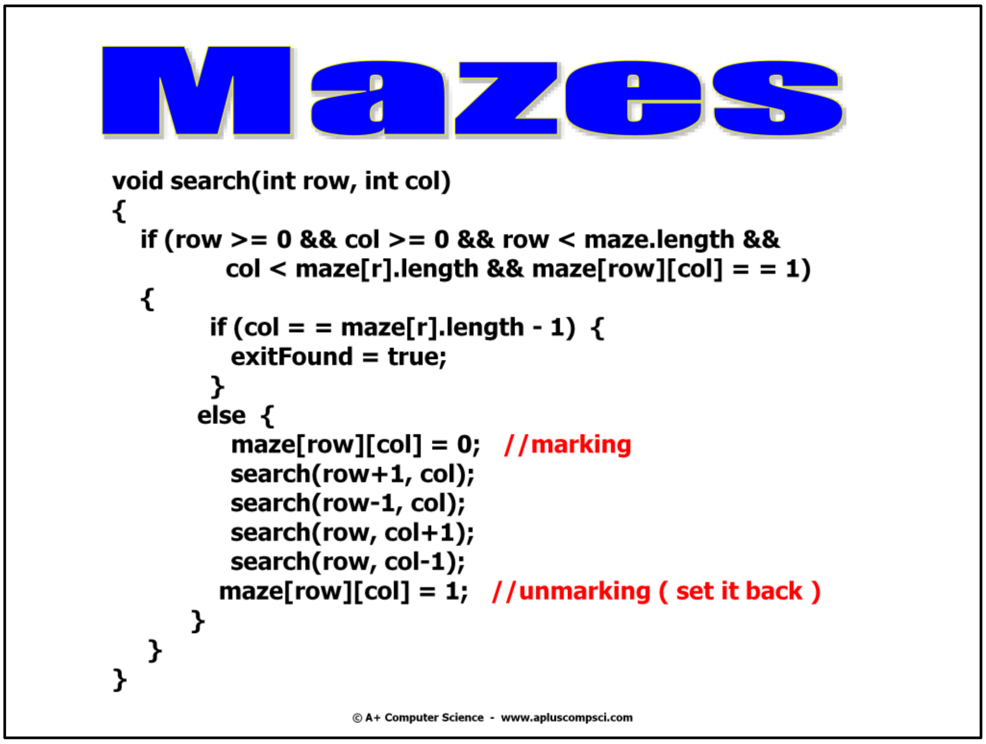
```
void search(int row, int col)
{
    if (row >= 0 && col >= 0 && row < maze.length &&
            col < maze[r].length && maze[row][col] = = 1)
  {
     if (col = = maze[r].length - 1) {
            exitFound = true;
     }
     else  {
            maze[row][col] = 0;   //marking
            search(row+1, col);
            search(row-1, col);
            search(row, col+1);
            search(row, col-1);
         }
   }
}
```

```
1  0  1  1  1
0  1  1  1  1
1  1  1  1  0
0  0  0  0  1
0  0  1  1  0
```



Mazes

In some situations, you want to make changes to the maze temporarily.   For instance, if you are trying to determine the shortest path, you have to find all paths and determine which path is the shortest. Each time to you search the maze for a path, the maze must be in its original state.



Mazes

```
void search(int row, int col)
{
   if (row >= 0 && col >= 0 && row < maze.length &&
            col < maze[r].length && maze[row][col] = = 1)
   {
          if (col = = maze[r].length - 1) {
            exitFound = true;
          }
         else {
            maze[row][col] = 0;   //marking
            search(row+1, col);
            search(row-1, col);
            search(row, col+1);
            search(row, col-1);
           maze[row][col] = 1;   //unmarking ( set it back )
         }
    }
}
```