adjacent empty squares.

Output: All (simple) paths from square \((x,y)\) to square \((h,s)\) that use

squares at \((h,f,x,f)\) and \((h,s,x,s)\) are empty.

Input: a lattice of squares, some "filled in" and the rest empty. Assume the

maze.

Make the computer find all simple paths through a planar square lattice based.
Maze Image

Array View: Represent filled squares with 1, empties with 0 in a C++ 2-dim array, choice! Note what the first 5 numbers are for.

You can prepare the input in a grid or not, your

skip over and ignore spaces and newlines.

"cin >> " for input makes the program

"cin >> ..“ for input makes the program

\[\begin{array}{cccc}
\text{Partially Filled:} & 0 & 0 & 0 & 1 \\
\text{Array View: Represent Filled Squares with 1, Empties with 0 in a C++ 2-dim Array,} & 0 & 1 & 0 & 1 \\
\text{Maze Image View} & 0 & 0 & 0 & 0 \\
\end{array}\]
Induced subgraph

Example of an
containing 0's.

Vertices: Array positions

of this Maze:

Graph Representation

Maze

Image of a Different

University at Albany Computer Science Dept.
Here is the rule for telling which array positions are neighbors (form an edge):

\[(i, j), (i+1, j), (i, j+1), (i-1, j), (i, j-1), (i+1, j)\]
\[(\zeta', \tau + \tau) \nrightarrow (\zeta', \tau) \nrightarrow (\zeta', \tau - \tau)\]

For one of the \( q = 0, 1, 2, \text{ or } 3 \).

\[
\lambda x. [\text{Rule} 4] \quad \lambda y. [\text{Rule} 9] \quad \lambda x. \text{dx AND Rule} + \tau = x \quad (3) \quad \text{and} \quad \lambda x. \text{dy + Rule} + \tau = x \quad (3) \quad \text{and} \quad 0 = \lambda y \quad (3) \quad \text{and} \quad u > \lambda \quad \Rightarrow \quad 0 \quad \text{AND} \quad u > \lambda \quad \Rightarrow \quad 0 \quad \text{AND} \quad u > \lambda \quad \Rightarrow \quad 0 \quad (4) \quad \text{and only if}\]

\[(\lambda x) (\zeta, \tau) \text{ is adjacent to}\]
\[\{ \{0, 1\}, \{1, 1\}, \{1, 0\}, \{0, 0\}, \{1, 0\}, \{0, 1\} \} = \]
\[\text{static cost structure for input dxy, input dy Rule} \quad [\text{Rule} 4] \quad \text{Table based coding of our graph adjacency Rule}\]
Here is the rule for telling which array positions are neighbors and forming an edge:

- \((i-1, j+1)\)
- \((i, j)\)
- \((i+1, j-1)\)
- \((i+1, j)\)
- \((i+1, j+1)\)
- \((i+1, j-1)\)
- \((i-1, j)\)
- \((i-1, j+1)\)
- \((i-1, j-1)\)

This idea is applicable to a hexagonal maze.
\begin{align*}
(\&, \&') & \quad (\&-\&', \&+\&') \\
(\&+\&', \&') & \quad (\&', \&') \\
(\&+\&', \&' -\&') & \quad (\&', \&' -\&')
\end{align*}

for one of the \( q=0, 1, 2, 3, 4, \text{ or } 5. \) //

\begin{align*}
\forall [b] \& + \&' & = x (3) \text{ and} //
\forall [b] \& + \&' & = 0 (2) \text{ and} //
\& u > \&' \Rightarrow 0 \text{ and } \& u > x \Rightarrow 0, \& u > \&' \Rightarrow 0, \& u > \&' \Rightarrow 0 (1) //
\end{align*}

\begin{align*}
\forall' (\&') & \text{ is adjacent to} //
\{\{0, 1\}, \{1, 0\}, \{-1, 0\}, \{0, -1\}, \{1, -1\}, \{-1, 1\}\} =
\text{static constant structure} \text{ in the } \& \text{ dy} \text{ Rule} [6] \text{ adjacency rule} //
\end{align*}
(2,1) (1,2) (2,2) (3,2) (2,3) (3,3) (1,4) (0,4) (0,0) (1,1) (0,1) (1,0)

Adjacent empty squares.

Output: All (simple) paths from square \((y_1, x_1)\) to square \((y_2, x_2)\) that use squares at \((y,f, x,f)\) and \((y,s, x,s)\) are empty.

Input: \(u\) by \(n\) lattice of squares, some filled in and the rest empty. Assume the maze.

Make the computer find all simple paths through a planar square lattice based problem of project 5:

University at Albany Computer Science Dept.
Sample output (first, reprint input maze):

0 0 0 0 1
0 1 0 1 0
0 0 0 0 0
0 0 1 0 1
1 1 1 0 0

Sample input:

0 0 0 0 1
0 1 0 1 0
0 0 0 0 0
0 0 1 0 1
1 1 1 0 0

5 4 0 0 5
Number of solutions = 3

| > > 0 1 |
| 0 1 1 0 |
| 0 0 > 0 |
| 0 0 1 1 |
| 1 1 1 > x |

The solution path printed by a direction symbol (> <) of a 0: > 0 0 1 |
| 0 1 0 1 |
| > > 0 |
| 0 0 1 1 |
| 1 1 1 > x |

Then, print each solution proceeded by a blank line with each empty square in the matrix.
Let's name it **Prin$\text{st}$$\text{itions}$ ($\ldots$).

6. Apply the recursion/divide and conquer pattern: Make the key procedure recursive.

5. What data does the key procedure need? It needs access to the whole maze.

4. What should the key procedure do? **Print each path that ends at the goal square** $(h,f,fx,f)$. with a parameter that indicates the first square $(sx, sy)$.

3. Every journey begins with the first step. **The key procedure will be called print the paths.**

2. **Print the maze**: to verify it was formed correctly and to help develop code to squares of the maze. (DONE)

1. **Input the maze size** $n$, row and column coordinates of the start and destination squares, and then $n^2$ or $1$s to represent the filled and empty cells.

**Solution Strategy**
There is a better way—Maintain TEMPO\(\text{RARY MARKS\ in\ an\ array.}\)

So, let's try making the PATH FOUN\(\text{D\ so\ FAR\ }}\) by a parameter to
path found SO FAR. This data varies with where we are in doing the work.

\[ \text{row} (i + 1) \]

... SOME \(Q\)UENES RESTRICT WHERE THE \( (i + 1)\)ST QUEEN MAY BE PLACED IN
\(R^n\)ow \(S\)quare TWICE. OTHERWISE, THERE MIGHT BE AN IN\(F\)INITELY NUMBER OF SOLUTIONS.

Each solution path must be simple, which means it must not use the same
a square outside the $\$n\$ by $\$n\$ maze.

or path found so far, nor does it use a square.

Each such continuation path does not use any square in the path and continuing to the goal square $(x, y)$ are printed.

```
// POSITION: All paths beginning with the given path so far.

// PRECONDITION: The list holding all the squares in the node starts a linked list holding all the squares in the path found so far. Thus, most currently found square in the path found so far.

// POSITION: Plast is the address of a node holding the
```

9. Direct algorithm pseudo-code for
{ else clause done} / return;

*/for loop finish*/ { { 

printSolutions (pnext );

pnext->data = this square;

}

AND NOT in the path so far )

is in the maze, and is empty,

according to Rule [q]

if (the square adjacent to plast)->data

}

for (b=0; b<4; b++)

pnext->link = plast;

PathListNode *pnext = new PathListNode;

int q; // AUTOMATIC VARIABLE

}
else

{ print the path found so far; return;

if (plast->data is the goal square )

... PathListNode *plast )
can access the (private) the 2-dim array and the size variable.

empty square. Those must be member functions of class Maze so they

an index pair is within the 0...n-1, 0...n-1 range it corresponds to an

Test if a given index pair (denoting a square) is in a linked list. (B) Test if
design helper methods to help implement its operations. For example, (A)
design the pseudo-code in the outline of the PrintSolutions() method to

A. Analyze the pseudo-code in the outline of the PrintSolutions() method to

so far.

pathPrintNode to implement the linked list of squares that is the path found

3. The Maze class uses some kind of linked list node structure of class type

PrintSolutions()

may be accessed by any methods that need them. Such methods include

private data members are private data members so they

2. The 2-dim array A array and size variable are private data members so they

reading it, printing it, printing paths, and printing all the solution paths.

1. Design a class named Maze to model one maze, and have methods for

Design Ideas
4. Write a skeleton implementation file maze.cpp which must include maze.h.

You have in your mind,
depicts.

correct any inconsistencies and syntax errors the compiler
and postconditions are consistent with each other and with the intentions

3. Check that the function declarations, data member declarations and pre-
document what you want each function member to do.

2. Start writing the file maze.h, and try to write pre- and postconditions to

1. Invent (or use our given) names for the data and function members.

Complete the Design of the Maze Class
1. Code the main module; edit, compile and try to link it until all compilation warnings, errors and linking errors go away.

2. You might want to write test drivers for some of the helper functions.

3. Finish coding, testing and debugging the skeleton functions, beginning with those that read in the maze data and print it out.