Algorithms for Part 2 and Part 3.

Data structure and design for Project.

Graph Search and Traversal by Labeling/Backtrack Algorithms.

CSI 310: Lecture 24
Graphs.

Search and traversal of a maze of squares ABSTRACTIONS to problems on
2. Traverse: Find all squares that are reachable from the START square.

1. Search: Find one path from START to GOAL

Solve Sec. 15.3 (and 15.4) detail Graph search and Traverse algorithm ideas that can
More precisely: \( V_1, V_2 \) is an arc/edge/tunnel in the graph.

can go to \( V_2 \) from \( V_1 \) in a SINGLE STEP.

Neighboring vertices (or squares): Vertex \( V_1 \) is a neighbor of vertex \( V_2 \) if you

Directed Graphs: Each arc is a one-way tunnel.

DIRE 3 pertains to

Reference: DS0 Sec. 15.3.
two-dimensional ARRAY (of char).

Since the vertices are named by (i,j), 0 ≤ i, j ≤ 0, it's cool to use a

   \[ u > i > j > 0 \text{ or } A \text{ or } \text{ if the vertex is marked,} \]

   \[ \text{say 0, if unmarked, one of } <, >, \text{ or } \text{if the marking value;} \]

Data structure needed: A variable for each vertex to hold the marking value.

- Once a vertex is processed, it is not ever processed again.
- So: or processed.

I. Mark (or label) each vertex (i.e. "maze square") the first time it is visited

Big Ideas:
I like to say here: “Δ has been fully explored.”

(c) After the above loop finishes, REMOVE Δ from S.

Coding idea: USE a LOOP!

are marked. If not, the neighbor is marked.

The computer systematically accesses all the neighbors of Δ, to check if they

This idea implies these algorithm steps:

We don’t KNOW if every neighbor of Δ is marked.

We KEEP track, in some data structure S, of every vertex Δ such that:

2. Δ is marked.
class Maze {
    // Data Members
    int A[24][24];  int n;
    int sx, sy, gs, gy;
    // 2-dim partially filled array
    char Marks[24][24];  // See Lecture Notes.

    Data Members
    } maze

    class Maze
    
    // Data Structure for the Maze and the Marks:

    University at Albany Computer Science Dept.
So, the stack of activation records holds $S$. Choice 1 gives recursive DEFT USE LOCAL EXTANT VARIABLE $V$ in

Depth First Traversal is done when data structure for $S$ (container for all

\[ S \quad \text{stack} \]

Is

the marked vertices that have NOT been FULLY EXPLORED

University at Albany Computer Science Dept.
Choice 2 gives NON-RECURSIVE DFTR: Use an explicit STACT, such as the linked-list of squares you used for Part 1.

a STACT

is

the marked vertices that have NOT been FULLY EXPLORRED.

Depth First Traversal is done when Data Structure for S (container for all

University at Albany Computer Science Dept.
Depth-first Labelling Traversal
Depth-First Labelling Traversal
Depth-first Labelling Traversal
Depth-first Labeling Traversal
Recursively BFS is unnatural (to me, at least) use an explicit queue, discussed in DO 15.3.

Breadth First Traversal is done when Data Structure for S (container for all marked vertices that have not been fully explored)
Breadth-First Labelling Search

Here are the squares in the order they are inserted in the queue:

0, 0, 1, 0, 2, 0, 3, 0, 3, 1, 4, 0, 3, 2, 5, 0

5, 4, 0, 4, 3, 5, 5, 5 DONE!

2, 2, 4, 2, 1, 2, 4, 3, 5, 2, 1, 3, 4, 4, 5, 3, 0, 3, 4, 5

Start

Goal