Graphs and Graph Search, for Project 6

Binary Search

Tree Application Review

Choosing Your Variables

CSI 310: Lecture 24
The gdb/ddd debugger's `print` or `display` commands reveal an object's value:

```
Foo = 42
```

as the corresponding argument: `Foo`, `cin`>`mychar`?

Functions with a non-constant reference parameter: Called with the object

Assignment operators: With the object on the LHS. Eg, `mychar`=`s`;

The value is set and modified by:

```
value = contents = state
```

Each object has a runtime (dynamic)

```
OBJECT = VARIABLE = INSTANCE
```

In C++ terminology, these words are synonyms:

University at Albany Computer Science Dept.
array (Could be an array of anything).

definitions or declarations.

struct or class: Programmer or Library defined by struct/class type

scalar: ALL pointers, char, int, double, bool, enum, etc.

There are 3 categories of type:

characters.

type that set has 256 elements in it. Only about 1/3 of it is printable

for the 8-bit char

Each object has a compile time (static) type, which determines what operations
object after the lifetime of that object has expired.

DANGER: A dangling pointer value is an address when used to access an OR, that memory might remain unused. (You cannot predict)
jotstream or other library.

in an activation record, free store_mainframe data structures, buffer for means, be used for some other object or a hidden system purpose (return address
After the lifetime expires, that memory resource might be recycled, which
like a lease which give you the right to occupy an apartment).
impelments the object (variable) is dedicated or reserved for that usage. (It is
The lifetime is the period of time during which the memory resource that
Each object (synonym variable) has a LIFETIME.
A constructor is called when execution flows into the definition of an automatic variable.

A ++ constructor is called when execution flows into the definition of an
same variable name.

allocate in the execution stack, so multiple variables exist (are alive) with the

New for CS1310: When a function recurses, multiple activation records for it are

when the corresponding function activation activation

The lifetime ENDS when execution leaves the containing block; in particular,

Taught first in CS201. Most common and elementary.

{ ... int heresanotherautomaticarray[1039];
main( ) { int *internationalObject; int anotherAuto; makeSOMETHING;
Automatic:

In C++, there are 3 basic kinds of lifetimes (which came from C):
You need an ADDITIONAL pointer type variable to hold the address returned by new, so you can access the dynamic variable that that new operation allocated.


Example of code that allocates one useless instance of a Maze and never deletes.

people, who prefer a confusing terminology.

Dynamic variables are allocated in the "free store" (called "heap" by some).

returned by the corresponding new.

If begins when new is run, and ends when delete is called on the address

we use library functions malloc() and free() instead.

This lifetime is controlled by explicitly coded new and delete operations. (In C,'

The 2nd basic kind of lifetime is DYNAMIC.
have LOCAL SCOPE except those declared externally.

All variables declared inside functions have static lifetime. All variables declared inside a function body makes the
function's name the variable's name have the SCOPE instead of GLOBAL SCOPE.

The variable keyword of variable declarations OUTSIDE of function bodies is the
static keyword of classical programs. Like Linus Torvalds, is the
another favorite feature of classical programmers.

Classical C style code often has static variables shared by several functions.

Defining a variable declared OUTSIDE of all function bodies makes that variable
have static lifetime (logically) spans the entire run of the program.

The 3rd basic kind of lifetime is STATIC.
instance (containing them). So, all 3 kinds are possible.

All other data members have the same lifetime as the particular object (ie,

(only one instance for that class).

static data members have STATIC lifetime. They are called „class variables„.

What about data members of classes/structs?
void printPath(int node) {
    if (node == 0)
        return;
    if (maze[node][1] == 0 && maze[node][size-1] == 0)
        postReturn true iff
    bool is-Empty (int i, int j) {
    post: Return true iff
        int size; // Invariant: Maze[0..size-1][0..size-1] hold our maze.
    int maze[64][64];
    private:
    } // class Maze
}

"Maze.h"

"Node.h"

#include "Node.h"

define Maze-Included
#define Maze-Defined

"Main.h"

IMPLEMENT VARIABLE belonging to main(), and use function members to
The choice Main/Savitch illustrated most was: Make the key class instance be an

This is a software design choice.

So, what lifetime should I choose for the Maze?
The assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.

By the assignment.
```c
{
    return 0;
}

void solve( pstart );  // prints all solutions and their count...
pstart->link = NULL;
pstart->f = 0;
pstart->t = 0;
pstart = new Node;
node *pstart;
void print();
Maze Maze();    // reads in and constructs the maze.
}

int main()
{
    #include "Maze.h"
    #include "node.h"
    Maze Maze();
}
...See below...

used by the key function's six-way loop.

Answer: The execution stack which contains the automatic variables named $g$, $h$ should then try an UPPER-RIGHT; then LEFT; then RIGHT; ... all six? How does the computer figure out that after trying an UPPER-LEFT step, it

Where is the stack of trial decisions?

The search algorithm follows the backtracking strategy of the n queens solution.
{ }  
*else clause done*/ return;  
}  
*for loop finished*  
{  
    solve (next);  
	pnext->data = this hexagon;  
}  

AND NOT in the path so far  
AND in the maze, AND is empty,

[Rule] according to Rule [Rules]  
it (the hexagon adjacent to path->data
})  

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

pnext->link = plast;  

node *pnext = new node;  

int q;  

// AUTOMATIC VARIABLE  
}

else  

{ }  

print the path found so far; return;  

if (plast->data is the goal hexagon)  

to guess: solve (node *plast);
study sec. 15.2 through for this subject.

For project 6, only implicitly represented graphs are involved. You will have to
The computer sometimes computes from the data in Mypos the representation of the current same position Mypos.

Graph vertex, such as THE.

The data or value in a data structure (i.e., object or variable) represents ONE

**Implicit**

עורר set<

 3. Edge Sets: Like edge lists, except a Set abstract data type (such as STL

vertex i.

The **ith** linked list contains all the numbers of the neighbors of number. The **ith** linked list contains all the numbers of the neighbors of

2. Edge Lists: Each linked list node pointer is used. Each node holds one vertex

edge by a[i][j] == false.

An edge from vertex i to vertex j is signified by A[i][j] == true, no such

1. Adjacency Matrix: An n by n array A[i][j] is used.

\[ \text{...} \]

\( (n - 1) \).

**Explicit**

Graph Implementations (See Sec. 15.2 in brief)

University at Albany Computer Science Dept.
tree or not.

(3) A search process to tell if a given key is in the structure of a search
process which can be compared with > and $.

(2) A decision tree of a decision process to find an answer.

(1) An expression tree expresses the structure of an expression.

Some applications of trees:
else search (left subtree of I, k)
     return false;
}

} else
{
    else search (left subtree of I, k)
     return false;
}

if (k > (root of I))
{
    if (k is in the root of I) return true;
    else search (right subtree I, key k)
}

Think of one explicit example: one binary search tree containing the keys 10, 20, 30, 40, 50, 60, 70 and 80. The demo program will build and print such a tree when you input these numbers, in any order.

Think of one explicit example: one binary search tree containing the keys 10, 20, 30, 40, 50, 60, 70 and 80. The demo program will build and print such a tree when you input these numbers, in any order.

Think of one explicit example: one binary search tree containing the keys 10, 20, 30, 40, 50, 60, 70 and 80. The demo program will build and print such a tree when you input these numbers, in any order.

Think of one explicit example: one binary search tree containing the keys 10, 20, 30, 40, 50, 60, 70 and 80. The demo program will build and print such a tree when you input these numbers, in any order. Each node.

Think of one explicit example: one binary search tree containing the keys 10, 20, 30, 40, 50, 60, 70 and 80. The demo program will build and print such a tree when you input these numbers, in any order.

Think of one explicit example: one binary search tree containing the keys 10, 20, 30, 40, 50, 60, 70 and 80. The demo program will build and print such a tree when you input these numbers, in any order.

Think of one explicit example: one binary search tree containing the keys 10, 20, 30, 40, 50, 60, 70 and 80. The demo program will build and print such a tree when you input these numbers, in any order.

Think of one explicit example: one binary search tree containing the keys 10, 20, 30, 40, 50, 60, 70 and 80. The demo program will build and print such a tree when you input these numbers, in any order.
count ++ "No..."
    }
    // precondition: A[0..n-1] MUST BE SORTED.
    Sequential Search (not best):

array elements and output k's index, if k is in the array.

increasing order, and an input key k, tell whether or not k is in one of the

The sorted array search problem: Given an array A[] of n keys sorted in
count >> "No..." end;
{
    if (bound = k + 1)
    else
    {
        if (bound = k - 1)

            ( [mid > 1]

                if (mid == 1)

                    // rounds down

                ( [bound + bound]/2

                    MI[ ] = (bound + bound)/2;

                    ( (bound > bound)

                        [Then K must be in A[bound...bound].]

                        [If K is in A[0...n-1]

                            Loop Invariant:

                            int mid;

                            if (int bound = n-1; // on page 563.

                            then int bound = n-1; // Note the pitfalls

                            ( precondition A[0...n-1] MUST BE SORTED.}

                            Binary search in a sorted array:

                            University at Albany Computer Science Dept.)
THINK: What if $u$ is a GIC, about 1 billion, which is $2^{30}$?

1 + \log_2 (u) \geq \text{previous length!}

So, the number of comparison steps is reduced to half or less of its
the subarray that remains to be searched is reduced to half or less of its
Why binary array search is efficient: After each < or > comparison operation,

If $k < a[3]$, we restrict search to $a[0..2]$.  

If $k > a[3]$, we restrict search to $a[4..7]$.  

<table>
<thead>
<tr>
<th>77</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>70</td>
<td>60</td>
<td>50</td>
<td>40</td>
<td>30</td>
<td>20</td>
<td>10</td>
</tr>
</tbody>
</table>

TRY it with $u=8$ on this array:
This binary search tree expresses the structure of the search done by the pseudo-code.
used by mortals to store.

The game state graphs of games like chess or Go are too big for any computer.

The game state graph of the configuration of the second vertex
by the first vertex into the configuration of the second vertex called an edge (when one legal move can change the configuration represented
position of configuration of a game. Two vertices are connected by an arc (also
represented some board

Game State Graph: Each node (also called vertex) represents some board.

Write solution software.

It’s like applying matching: Often a problem can be represented as an equation, and
problem on the corresponding graph.

problem in Problem Solving: Often, a problem can be represented as
corresponding graph.

4. A graph, like a tree, is a nonlinear data structure consisting of nodes and

4. 716-726 for Project 6.

We skip to Chapter 15 on Graphs, now? But only 695-702 and
Problem: Can you get from the start position HTH to the goal position THT?

\[ \text{\( X X A \leftrightarrow \text{XXA and } \Lambda X X \leftrightarrow \text{AXX}\)}\]

as each other. (c) You may flip one of the end coins only if the other two coins are the same.

2. You may flip the middle coin whenever you want to.

\[ \text{\( Z X X \leftrightarrow \text{ZX X}\)}\]

I. You may flip the middle coin whenever you want to.

Rules:

Following:

Each move consists of turning over one of the 3 coins according to these.

<table>
<thead>
<tr>
<th>Tail</th>
<th>Head up in that order (denoted HTH)</th>
</tr>
</thead>
</table>

The board is 3 coins placed in a row. At the start, the coins are turned Head, Tail, and Head up in that order (denoted HTH).

Game Board and What is a Move

Example of p. 697-9:
Let's draw the game state graph:

Rule 1: HXH $\leftrightarrow$ HX'H
Rule 2

\[ HHT \xleftarrow{\text{Rule 2}} HHX' \xrightarrow{\text{HHX}} HHH \xrightarrow{\text{HHX}} HTH \]

\[ X'HH \xleftarrow{\text{HHX}} HHH \xrightarrow{\text{HHX}} HTH \]
There were two ways to get to the same position!
Graph tell us about how many moves it takes to win?

What does this

The same game state graph drawn differently, p. 698-699.
The computer sometimes computes from the data in Wpos the representation of

Graph vertex such as THE CURRENT SAME POSITION Wpos.

The data or value in a data structure, (i.e., object or variable) represents one

Implicit:

(set<int>);

3. Edge Sets: Like edge lists, except a SET abstract data type (such as STL

vertex i.

The i-th linked list contains all the numbers of the neighbors of

number. The i-th linked list contains all the numbers of the neighbors of

vertex.

2. Edge Lists: n linked list node pointers are used. Each node holds one vertex

edge by A[i][j] = true.

An edge from vertex i to vertex j is signified by A[i][j] = true, no such

I. Adjacency Matrix: An n by n array A[i][j] is used.

n – 1).

Explicit: (Suppose the graph has n vertices, The vertices are numbered 0, 1, …)

Graph Implementations (Sec. 15.2 in brief)
Problem 6: Make the computer find all simple paths through a planar hexagonal lattice.

Input: A lattice of hexagons, some filled in and the rest empty. Assume the hexagons at (0,0) and (n-1,n) are empty.

Output: All (simple) paths from hexagon (0,0) to hexagon (n-1,n) that use adjacent empty hexagons only.

University at Albany Computer Science Dept.
Maze Image

Array View: Represent filled cells with 1, empties with 0 in a C++ 2-dim. array:

```cpp
int Maze[64][64];
for(int i=0;i<n;i++)
    for(int j=0;j<n;j++)
        cin >> Maze[i][j];
```

View

View

Maze Image
Image of a Different Maze

Graph Representation

Vertices: Array positions containing 0's.

Example of an induced subgraph of this Maze:
Here is the rule for

edge

forming neighbors which array

telling which array

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

(0, 1)
(1, 1)
(2, 1)
(3, 1)
(4, 1)
(0, 2)
(1, 2)
(2, 2)
(3, 2)
(4, 2)
(0, 3)
(1, 3)
(2, 3)
(3, 3)
(4, 3)
(0, 4)
(1, 4)
(2, 4)
(3, 4)
(ζ', ι+ι) (ι', ι+ι)
(ι+ζ', ι) (ζ', ι) (ι-ζ', ι)
(ι+ζ', ι-ι) (ζ', ι-ι)

for one of the b = 0, 1, 2, 3, 4, or 5. //

 δy [δy] + δx [δx] = y [y] and Rule (3) (3) and //
0 = [δy] and [δx] = 0 AND Maze (2) (2) and //
\( u > \lambda \Rightarrow 0 \) AND \( u > x \Rightarrow 0 \) AND \( u > f \Rightarrow 0 \) AND \( u > i \Rightarrow 0 \) (1) //

: if AND only if //

(δy, δx) is adjacent to //

\{\{0\}, \{1\}, \{0, 0\}, \{0, 1, 1\}, \{1, 1\}\} = //

Table based Coding of our Graph Adjacency Rule://

"University at Albany Computer Science Dept."
Problem of Project 6:

Output: All (simple) paths from hexagon (0, 0) to hexagon (u, u - 1) that use adjacent empty hexagons only.

Input: A lattice of hexagons, some "filled in", and the rest empty. Assume that hexagons at (0, 0), (0, 1), (1, 0), (2, 0), (3, 0), (3, 1), (3, 2), (2, 3), (1, 3), (1, 4), (2, 4), (3, 4), (4, 4).

(0, 0), (0, 1), (1, 0), (2, 0), (3, 0), (3, 1), (3, 2), (2, 3), (1, 3), (1, 4), (2, 4), (3, 4), (4, 4).
Sample input (first, reprint input maze, indented):

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

Sample output (first, reprint output, indented):

```
0 0 0 1 1
0 1 0 0 0
0 0 1 1 0
0 0 0 1 0
0 1 1 0 0
```
Where, of course, $\exists \exists \exists$

Number of solutions found, with:

Then, finally, report the

A few more...

| p | p | 0 | 1 | 1 | p | p | p | p | p | p | p | p | p | p | p | p | p | p | p |

in the solution path printed by a $p$ instead of a $0$.

Then, print each solution, proceeded by a blank line, with each empty hexadecimal.
Let’s name it `printSolutions()`.

recursive.

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

at least.

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

**Goal hexagon** \((u, u - 1, u - 1)\)

Print each path that ends at the

4. What should the key procedure do?

print the paths.

3. Every journey begins with the first step: The key procedure will be called

print the maze! to verify it was formed correctly and to help develop code to

(DONE)

empty hexagons of the maze.

I. Input the maze size \(n\) and then \(n^20s\) or \(1s\) to represent the filled and

Solution Strategy
So, let's try making the path found so far be a parameter to
which data varies with where we are in doing the work.

7. Each solution path must be simple, which means it must not use the same
hexagon twice. Otherwise, there might be an infinite number of solutions.

8. So, present solutions (***).

This condition has a flavor from the n-Queens problem: After some queens
have been tentatively placed in rows 1, 2, ..., there might be an
infinite number of solutions.

Otherwise, these are in which hexagons are in the

the (?? + 1)st queen may be placed in row (?? + 1).

So, let's try making the path found so far be a parameter to
which data varies with where we are in doing the work.

7. Each solution path must be simple, which means it must not use the same
hexagon twice. Otherwise, there might be an infinite number of solutions.

This condition has a flavor from the n-Queens problem: After some queens
have been tentatively placed in rows 1, 2, ..., there might be an
infinite number of solutions.

Otherwise, these are in which hexagons are in the
a hexagon outside the $\$ by \$ maze.

path found so far, nor does it use a "tile"ed" hexagon.

Each such continuation path does not use any hexagon in the

tar and continuing to the goal hexagon $(v-1, u-1)$ are printed.

// POSITION_condition: All paths beginning with the given path so

path found so far.

node starts a (tinked) list holding all the hexagons in the

most current" found hexagon in the path found so far. This

// PRECONDITION: plase is the address of a node holding the

( )

9. Direct Algorithm Pseudo-code for
{ else clause done

for (i = 0; i < height; i++)
  for (j = 0; j < width; j++)
    if (path->next->data = this hexagon)
      { else if (not in the path so far)
        if (is in the maze, and is empty,
          according to rule[j][g]
            // (the hexagon adjacent to path->last->data

          for (b = 0; b < g; b++)
            path = path->next->next;
          path = new PathListNode;
          // AUTOMATIC VARIABLE!

          for
          else
            { print the path found so far; return;
              if (path->goal = the goal hexagon) } } 10. 2.2. PrintSolutions (PathListNode *path)
can access the (private) the 2-dim array and the size variable.

"empty" hexagons. These must be member functions of class Maze so they

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

Test if a given index pair (denoting a hexagon) 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

4. Analyze the pseudo-code in the outline of the printSolutions() method to

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

pathTraversal to implement the linked list of hexagons that is the path

#include printSolutions()

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

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

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

41
determine.

Compile it and correct any inconsistencies and syntax errors the compiler

'4. Write a skeleton implementation file Maze.cpp which must include Maze.h'

you have in your mind.

and postconditions are consistent with each other and with the intentions

3. Check that the function declarations, data member declarations and pre-

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.