Graphs and Graph Search, for Project 5

Binary Search

Tree Application Review

CSI 310: Lecture 21
Binary Search Tree: A decision tree for answering whether or not a given question, (M/S confuse this with taxonomy trees).

Binary Decision Trees: Each leaf is an answer; each non-leaf is a yes-no question. (it is like a telephone book).

Other Name Space Trees: EC, the Domain Name System of the Internet.

File Name Trees: Express a system to identify files using a sequence of directory names, plus a file name.

Taxonomy Trees: See http://www.nord.nuth.gov/ncbo/Taxonomy and expression (string, web document, program, etc.).

Expression Trees: Express the structure of the computation expressed by an
For the tree and each subtree \( T \), the root contains the largest of the numbers in the tree (of numbers) with the heap property.
worst case. Use an amount of computer work proportional to the height of the tree, in the
insertion into a heap, and
search and insertion in a binary search tree,
Main conclusion:
number into a heap ordered tree.
We then viewed Main/Savitch's page 520 and described how to insert a new
search tree.
example of how a dictionary of states (of the USA) is implemented by a binary
We viewed Main's slide set 10b on the dictionary abstract data type and the
search and insert a new number into it.
We viewed Main/Savitch's page 498 on a binary search tree and described how to
We viewed Main/Savitch's page 470 on a decision tree.
Some Applications of Trees:

1. An Expression Tree expresses the structure of an expression.
2. A Decision Tree expresses the structure of a decision process to find an answer.
3. A Search Tree (of numeric or other keys which can be compared with < and >) expresses the structure of a search process to tell if a given key is in the tree or not.
{ { return search( right subtree of T, k ); };
    if ( T has no right subtree ) return false;
}
else
{
    if ( T has no right subtree ) return false;
    if ( k > key(root of T) )
        ( )
    if ( k is in the root of T ) return true;
}

Search: BinarySearchtree, 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 of Lab 22 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 of Lab 22 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 of Lab 22 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 of Lab 22 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 of Lab 22 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 of Lab 22 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 of Lab 22 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 of Lab 22 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 of Lab 22 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 of Lab 22 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 of Lab 22 will build and print such a
tree when you input these numbers in any order.
count >> "No..."
{
    count >> "Yes!"
    if (K == [1] A)
    for (int i = 0; i < u (A));
}

//Precondition: A[0..n-1] MUST BE SORTED.
//Precondition: A[0..n-1] MUST BE SORTED.

Sequential Search (not best):
count >> "No..." )
{
    
    
bound = k + I
}
else
bound = k - I

if K > A[mid]

if K == A[mid]

MID = (bound + bound) / 2;

while (bound > bound)

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

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

Loop Invariant:

int Mid;

search int bound = n-I;  // on page 563.

search int bound = 0;  // Note the preconditions

A[0..n-I] MUsT BE SORTED.

Binary search (in a sorted array):
THINK: What if \( u \) is a Gt, about 1 billion, which is 230? 

If \( k > A[3] \), we restrict search to \( A[4] \ldots 7 \].

If \( k < A[3] \), we restrict search to \( A[0] \ldots 2 \].

[Diagram of a 2D array with values from 0 to 80]
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 graphs of games like chess or Go are connected by an arc (also called an edge) which one legal move can change the configuration represented by the first vertex into the configuration of the second vertex. Two vertices are connected by an arc (also position of configuration of a game). A game graph represents some board

Game State Graph: Each node (also called vertex) represents solution software.

You can buy, download, or write the solution is obtained by solving the equation... Often a problem can be represented as an equation, and it's like applying match. "A problem on the corresponding graph."

"Graphs in Problem Solving: Often, a problem can be represented as a graph, and the solution to the problem in obtained by solving a graph."

On the needs of an application.

Often, graph nodes may be linked in any pattern—or lack of pattern—dependent Gone. Graph nodes may be linked in any pattern or lack of pattern. If a tree is between the nodes... But in a graph, even this modicum of order is lost. A graph, like a tree, is a nonlinear data structure consisting of nodes and links.

716-726 for Project 5.

We skip to Chapter 15 on Graphs, now!! But only 696-702 and
Problem: Can you get from the start position HTH to the goal position THT?

(X, X \leftarrow X, X \text{ and } \lambda, X \leftarrow \lambda, X)

as each other.

2. You may flip one of the end coins only if the other two coins are the same.

(Z, \lambda \leftarrow Z \lambda \lambda) (\text{middle coin whenever you want to}.

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:

Tail, and Head up in that order (denoted HTH).

The board is 3 coins placed in a row. At the start, the coins are turned Head,'
Rule 1 \( HXH \leftrightarrow HX'H \)

Let's draw the game state graph.
HTH
HHH
HHX'\leftrightarrow HHX
HHH
X'HHX\leftrightarrow HHX
Rule 2
Rule 2
Rule 2
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 MP_{pos} the representation of

\textbf{Graph vertex}, such as \textbf{THE current same position MP_{pos}}.

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

Explicit:

\begin{enumerate}
\item A vertex \( i \) is adjacent to vertex \( j \) if \( \exists \) an edge \( i \rightarrow j \).
\item An adjacency matrix: \( A[i][j] = 1 \) if \( i \rightarrow j \).
\item Suppose the graph has \( n \) vertices. The vertices are numbered \( 0, 1, \ldots, n-1 \).
\end{enumerate}

\textbf{Graph Implementations (Sec. 15.2 in brief)}
Problem of Project 5:

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

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

Output: All (simple) paths from square \((h, s, x, s)\) to square \((h, f, x, f)\) that use adjacent empty squares.

() (4, 5) 
(3, 5) 
(2, 5) 
(1, 4) 
(3, 4) 
(3, 3) 
(3, 2) 
(2, 2) 
(1, 1) 
(1, 2) 
(2, 2) 
(3, 2) 
(3, 3) 
(2, 4) 
(3, 4) 
(2, 5) 
(3, 5) 
(4, 5) 

18 University at Albany Computer Science Dept.
You can prepare the input in a grid or not, your choice. Note what the first 5 numbers are for.

Skipped over and ignore spaces and newlines.

partially filled:

0 in a C++ 2-dim array, squares with 1, empties with 0 in a C++ 2-dim array, squares with 1, empties with

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

Maze Image View

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

Maze Image View
Induced subgraph (Example of an
containing 0's.
Vertices: Array positions
of this Maze:
Graph Representation
Maze
Image of a Different
Here is the rule for telling which array positions are neighbors:

\[(i,j)\] — \[(i,j+1)\] — \[(i,j-1)\] — \[(i-1,j)\] — \[(i+1,j)\]

The edge forms an array.
\((\zeta', \tau')\) \mid
\((\tau + \zeta', \tau) \rightarrow (\zeta', \tau) \rightarrow (\tau - \zeta', \tau)\)

for one of the \(q = 0, 1, 2, \) or \(3.\)

\[\forall [\varphi] \text{ dy} \cdot \forall [\psi] \text{ dx} \AND \forall [\varphi] \text{ dy} \cdot \forall [\psi] \text{ dx} \AND \forall [\varphi] \text{ dy} \cdot \forall [\psi] \text{ dx}
\]

\(u \Rightarrow \lambda \Rightarrow \iota \AND u \Rightarrow x \Rightarrow \iota \AND u \Rightarrow \zeta \Rightarrow \iota \AND u \Rightarrow \tau \Rightarrow \iota (1) //
\]

: \(\forall [\lambda] \AND \iota \AND (x, \zeta) \text{ is adjacent to position } (\iota, \tau) \)

\{ \{0, \iota\}, \{0, \iota\}, \{0, -\iota\}, \{0, -\iota\} \} = \)

static const struct \{ \text{int dx; \text{int dy} \text{ rule} } [4] \text{ Table based coding of our Graph adjacency rule} //

University at Albany Computer Science Dept.
Here is the rule for telling which array positions are neighbors:

Edge:

\[
\begin{align*}
(\tau, \tau + 1) & \quad (\tau - 1, \tau + 1) \\
\tau + 1, \tau & \quad (\tau, \tau) & \quad (\tau - 1, \tau - 1) \\
\tau + 1, \tau - 1 & \quad (\tau, \tau) & \quad (\tau + 1, \tau)
\end{align*}
\]

This idea is flexible. Let's do a hexagonal maze.
\[(i,j) \rightarrow (i-1,j-1) \quad (i,j+1) \quad (i+1,j) \quad (i+1,j+1) \quad (i-1,j) \quad (i,j-1)\]

For one of the \( a = 0, 1, 2, 3, 4 \) or 5, //
\[dx \cdot \text{rule} + \text{if} \quad \text{and (3)} \quad \text{and (2)} \quad \text{and (1)} \quad \text{if and only if} \quad (\lambda, x) \text{ is adjacent to} \quad (i,j) \quad \text{position (i,j)} \]
\[
\{ \{0,0\}, \{0,1\}, \{0,1\}, \{0,-1\}, \{0,-1\}, \{1,0\}, \{1,1\}, \{1,-1\}, \{1,-1\}\}; =
[6]
\text{static cost strcut} \{ \text{int dx}; \text{int dy}; \text{rule [6]} \text{Table based Coding of our HEXAGONAL Graph adjacency rule};
//
adjacent empty squares.

Output: All simple paths from square (his, xs) to square (hj, xf) are empty.

Input: n by n lattice of squares, some "hilled 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:
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
```

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
5 1 1 5 5
```
Number of solutions = 3

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The solution path printed by a capital letter instead of a 0:

Then, print each solution preceded by a blank line, with each empty square in

University at Albany Computer Science Dept.
Let's name it `printSolutions()`.

1. Print the maze: to verify it was formed correctly and to help develop code to print the paths.

2. Print each path that ends at the goal square \((x_f, y_f)\).

3. Every journey begins with the first step. The key procedure will be called with a parameter that indicates the first square \((s_x, s_y)\).

4. What should the key procedure do? Print the paths.

5. What data does the key procedure need? It needs access to the whole maze, \((h_f, f_x, f_y)\).

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

Solution Strategy
8. So, Print solutions (\ldots). So, let's try making the PATH FOUND SO FAR be a parameter to track found solutions with. This data varies with where we are in doing the work.

Each square twice, otherwise, there might be an infinite number of solutions.

After some queens have been tentatively placed in rows 1, 2, \ldots, \ldots THESE queens restrict where the \( (? + I) \)st queen may be placed in row \( ( ? + I) \). This condition has a flavor from the \( n \)-Queens problem (see a future lecture):
a square outside the $\text{subset}$ of the maze.

Each such continuation path does not use any square in the path found so far, nor does it use a "filled" square or far and continuing to the goal square $(f, y)$ are printed.

9. Dijkstra\'s algorithm pseudo-code for

```plaintext
// PositionCondition: All paths beginning with the given path so far.
// Path found so far.
node starts in the list holding all the squares in the paths found so far. This
most currently found square in the path found so far. Thus
PrintSolutions(PartialSolutions(PlastNode, PlastNode * plast)
```
{ } //else clause done

/** for loop finish */
{

printSolutions ( next ) ;

next->data = this square ;

}

AND NOT in the path so far

is in the maze, AND is empty,

according to Rule [9]

if (the square adjacent to plast->data

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

next->link = plast ;

pathListNode *next = new pathListNode ;

int q ;

else

{ } //AUTOMATIC VARIABLES

else

print the path found so far ; return ;

if (plast->data is the goal square )

}
can access the (private) the 2-dim array and the size variable.

"empty square. These 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.

pathFinder() to implement the linked list of squares that is the path found

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

printSolutions()

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

2. The 2-dim 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

33

University at Albany Computer Science Dept.
A. Write a skeleton implementation file `maze.cxx` which must include `maze.h`.

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

3. Check that the function declarations, data member declarations and pre-
   and postconditions are consistent with each other and with the intentions

You have in your mind.

4. Compile it and correct any inconsistencies and syntax errors the compiler
detects.
those that read in the maze data and print it out.

3. Finish coding, testing and debugging the skeleton functions, beginning with

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

1. Code the main module; edit, compile and try to link it until all compilation

Implemtentation and Test