TWTG

- What work is still to come
  - APT, Huff, APT
- What work has been done and feedback to come
  - Where are we, where will we be

- Two d[][] arrays, backtracking, graphs
  - Data structures, algorithms, analysis
- Toward Huffman Coding
  - Priority Queues and Data compression

Backtracking by image search

Searching with no guarantees

- Search for best move in automated game play
  - Can we explore every move?
  - Candidate moves ranked by “goodness”?
  - Can we explore entire tree of possible moves?

- Search with partial information
  - Predictive texting with T9 or iPhone or ...
  - What numbers fit in Sudoku suare

- Try something, if at first you don't succeed ....

Search, Backtracking, Heuristics

- How do you find a needle in a haystack?
  - How does a computer play chess?
  - How does a computer play Go? Jeopardy?

- How does Bing/Googlemap find routes from one place to another?
  - Shortest path algorithms
  - Longest path algorithms

- Optimal algorithms and heuristic algorithms
  - Is close good enough? Measuring “closeness”?
  - Is optimality important, how much does it cost?
Exhaustive Search/Heuristics

- We can probably explore entire game tree for tic-tac-toe, but not for chess
  - How many tic-tac-toe boards are there?
  - How many chess boards are there?
- What do we do when the search space is huge?
  - Brute-force/exhaustive won't work, need heuristics?
  - What about google-maps/Garmin finding routes?
- Backtracking can use both concepts

Classic problem: N queens

- Can queens be placed on a chess board so that no queens attack each other?
  - Easily place two queens
  - What about 8 queens?
- Make the board NxN, this is the N queens problem
  - Place one queen/column
  - Horiz/Vert/Diag attacks
- Backtracking
  - Tentative placement
  - Recurse, if ok done!
  - If fail, undo tentative, retry
- wikipedia-n-queens

Backtracking idea with N queens

- For each column C, tentatively place a queen
  - Try each row in column C, if [r][c] ok? Put "Q"
  - Typically "move on" is recursive
  - If done at [r][c], DONE! Else try next row in C
    - Must remove 'Q' if failure, when unwinding recursion
- Each column C "knows" what row R being used
  - At first? that's row zero, but might be an attack
  - Unwind recursion/backtrack, try "next" location
- Backtracking: record an attempt go forward
  - Move must be "undoable" on backtracking

N queens backtracking:

```java
public boolean solve(int col){
    if (col == mySize) return true;
    // try each row until all are tried
    for(int r=0; r < mySize; r++){
        if (myBoard.safeToPlace(r,col)){
            myBoard.setQueen(r,col,true);
            if (solve(col+1)){
                return true;
            }
            myBoard.setQueen(r,col,false);
        }
    }
    return false;
}
```

[https://git.cs.duke.edu/201fall16/nqueens/tree/master](https://git.cs.duke.edu/201fall16/nqueens/tree/master)
Queens Details

- How do we know when it's safe to place a queen?
  - No queen in same row, or diagonal
  - For each column, store the row that a queen is in
  - See QBoard.java for details

- How do we set a Queen in column C?
  - Store row at which Queen placed, simulate [r][c]
  - Must store something, use INFINITY if no queen
  - See Qboard.setQueen for details

Basic ideas in backtracking search

- Enumerate all possible choices/moves
  - Try choices in order, committing to a choice
  - If the choice doesn't work? Must undo, try next
    - Backtracking step, choices must be undoable

- Can a move be made? Try it and use move/board
  - If move worked? Yes!!!
  - If move did not work? Undo and try again

- Board holds state of game: make move, undo move

GridGame APT


- Why is this a backtracking problem?
  - Try a move, if it's a win? Count it.
  - If it's not a win? Undo it and try next move

- What does winning move mean?
  - Opponent has no winning move!
  - Assume plays perfectly, same code as me!!!!!
What can we do with a board?

- Can you determine if \([r][c]\) is legal?
  - \([1][0]\) is legal, why?
  - \([3][1]\) is NOT legal, why?

- Suppose there are no legal moves? Answer: Zero/0

- Suppose I place an 'X' and then ask
  - How many ways to win does opponent have?
  - If answer is Zero/0, what does placing 'X' do?

- This leads to backtracking, believe the code!!!

GridGame backtracking, count wins

```java
private int countWinners(char[][] board) {
    int wins = 0;
    for(int r=0; r < 4; r++) {
        for(int c=0; c < 4; c++) {
            if (canMove(board, r, c)) {
                board[r][c] = 'X';
                int opponentWins = countWinners(board);
                if (opponentWins == 0) {
                    wins += 1;
                }
                board[r][c] = '.';
            }
        }
    }
    return wins;
}
```

Two-dimensional Arrays

- In Java this is really an array of arrays
  - What does int[][] x = new int[4][4] look like?
  - See javarepl.com

- What does this do:

```java
String[] strs = {"X.X", "X.X", "X.X", .....", "X.X", "X.X", "X.X", .....", ....."};
char[][] board = new char[4][4];
for(int k=0; k < 4; k++) {
    board[k] = strs[k].toCharArray();
}
```
Toward understanding rats and cheese

http://www.cs.duke.edu/csed/newapt/ratroute.html

Key Ideas here

- Create a two-d char[][] array for the board
  - Find cheese and rat in doing this, use standard recursion rather than backtracking

```java
int cheeseRow, cheeseCol; // instance vars
public int numRoutes(String[] enc) {
    int ratRow = 0, ratCol = 0;
    board = new char[enc.length][enc[0].length()];
    // initialize all state, instance vars/local
    int currentDistance = cheeseDistance(ratRow, ratCol);
    return countUp(ratRow, ratCol, currentDistance, board);
}
```

How does countUp work?

- First check row, col parameters
  - If out of bounds? Return ...
  - If board[row][col] == 'X'? Return ...
  - If board[row][col] == 'C'? Return ...

- Calculate cheeseDistance(row,col)
  - If moving away, i.e., greater than parameter?

- Make recursive call for each of four neighbors
  - Accumulate sum, return after making four calls

WOTO


Creating two-d arrays
Visiting all 4 or 8 neighbors using offset arrays
BoggleScore revisited

- Given an n-letter word W, find how many ways W can be found on a 4x4 board
  - Letters are indexed: 0, 1, 2, ..., n-1

- Suppose for each b[r][c] we knew how many ways W ended at b[r][c]?
  - Then we could add up 16 numbers and be done!
  - How many ways does "AAAAH" end at [0][0], at [0][1], and so on

Find the word 'ATP' on a board

- Create a 4x4 char[][] grid
- Parallel 4x4 long[][] counts
  - We will make one for each letter in word

<table>
<thead>
<tr>
<th></th>
<th>T</th>
<th>A</th>
<th>S</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

So final answer is ... 12

Steps for each word

- Initial long[][]. 1's and 0's, we can call this $G_0$
- Construct $G_n$ from $G_{n-1}$ with board and n$^{th}$ char
  - Neighbor-sum if n$^{th}$ letter found on board
- char and long[][] and board make new long[][]
  - Once for each char
  - Base case? No chars: sum

<table>
<thead>
<tr>
<th></th>
<th>T</th>
<th>A</th>
<th>A</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

So final answer is ... 12

Computer v. Human in Games

- Computers can explore a large search space of moves quickly
  - How many moves possible in chess, for example?
- Computers cannot explore every move (why) so must use heuristics
  - Rules of thumb about position, strategy, board evaluation
  - Try a move, undo it and try another, track the best move
- What do humans do well in these games? What about computers?
  - What about at Duke?
**Games at Duke**

- **Alan Biermann**
  - Natural language processing
  - Compsci 1: Great Ideas
  - Duchess, checkers, chess

- **Tom Truscott**
  - Duke undergraduate working with/for Biermann
  - Usenet: online community
  - Second EFF Pioneer Award (with Vint Cerf!)

**Heuristics**

- A heuristic is a rule of thumb, doesn't always work, isn't guaranteed to work, but useful in many/most cases
  - Search problems that are "big" often can be approximated or solved with the right heuristics

- What heuristic is good for Sudoku?
  - Is there always a no-reasoning move, e.g., 5 goes here?
  - What about "if I put a 5 here, then..."?

- What other optimizations/improvements can we make?
  - For chess, checkers: good heuristics, good data structures

**Huffman Coding**

- Understand Huffman Coding
  - Data compression
  - Priority Queue
  - Bits and Bytes
  - Greedy Algorithm

- Many compression algorithms
  - Huffman is optimal, per-character compression
  - Still used, e.g., basis of Burrows-Wheeler
  - Other compression 'better', sometimes slower?
  - LZW, GZIP, BW, ...

**Compression and Coding**

- What gets compressed?
  - Save on storage, why is this a good idea?
  - Save on data transmission, how and why?

- What is information, how is it compressible?
  - Exploit redundancy, without that, hard to compress

- Represent information: code (Morse cf. Huffman)
  - Dots and dashes or 0's and 1's
  - How to construct code?
PQ Application: Data Compression

- Compression is a high-profile application
  - .zip, .mp3, .jpg, .gif, .gz, ...
  - What property of MP3 was a significant factor in what made Napster work (why did Napster ultimately fail?)

- Why do we care?
  - Secondary storage capacity doubles every year
  - Disk space fills up there is more data to compress than ever before
  - Ever need to stop worrying about storage?

More on Compression

- Different compression techniques
  - .mp3 files and .zip files?
  - .gif and .jpg?
- Impossible to compress/lossless everything: Why?
- Lossy methods
  - pictures, video, and audio (JPEG, MPEG, etc.)
- Lossless methods
  - Run-length encoding, Huffman

Coding/Compression/Concepts

- For ASCII we use 8 bits, for Unicode 16 bits
  - Minimum number of bits to represent N values?
  - Representation of genomic data (a, c, g, t)?
  - What about noisy genomic data?

- We can use a variable-length encoding, e.g., Huffman
  - How do we decide on lengths? How do we decode?
  - Values for Morse code encodings, why?

Huffman Coding

- D.A Huffman in early 1950's: story of invention
  - Analyze and process data before compression
  - Not developed to compress data "on-the-fly"
- Represent data using variable length codes
  - Each letter/chunk assigned a codeword/bitstring
  - Codeword for letter/chunk is produced by traversing the Huffman tree
  - Property: No codeword produced is the prefix of another
  - Frequent letters/chunk have short encoding, while those that appear rarely have longer ones
- Huffman coding is optimal per-character coding method
Mary Shaw

● Software engineering and software architecture
  ▶ Tools for constructing large software systems
  ▶ Development is a small piece of total cost, maintenance is larger, depends on well-designed and developed techniques

● Interested in computer science, programming, curricula, and canoeing, health-care costs

● ACM Fellow, Alan Perlis
  Professor of Compsci at CMU

Huffman coding: go go gophers

<table>
<thead>
<tr>
<th>ASCII</th>
<th>3 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>g</td>
<td>010</td>
</tr>
<tr>
<td>o</td>
<td>011</td>
</tr>
<tr>
<td>p</td>
<td>012</td>
</tr>
<tr>
<td>h</td>
<td>100</td>
</tr>
<tr>
<td>e</td>
<td>101</td>
</tr>
<tr>
<td>r</td>
<td>110</td>
</tr>
<tr>
<td>s</td>
<td>111</td>
</tr>
<tr>
<td>sp.</td>
<td>1110</td>
</tr>
</tbody>
</table>

Encoding uses tree/trie:
- combine two smallest weights
- repeat
- priority queue?

Building a Huffman tree

- Begin with a forest of single-node trees/tries (leaves)
  ▶ Each node/tree/leaf is weighted with character count
  ▶ Node stores two values: character and count

- Repeat until there is only one node left: root of tree
  ▶ Remove two minimally weighted trees from forest
  ▶ Create new tree/internal node with minimal trees as children,
    - Weight is sum of children’s weight (no char)
- How does process terminate? Finding minimum?
  ▶ Remove minimal trees, hummm….
How do we create Huffman Tree/Trie?

- Insert weighted values into priority queue
  - What are initial weights? Why?
- Remove minimal nodes, weight by sums, re-insert
  - Total number of nodes?

```java
PriorityQueue<TreeNode> pq = new PriorityQueue<>();
for (int k = 0; k < freq.length; k++){
    pq.add(new TreeNode(k, freq[k], null, null));
}
while (pq.size() > 1){
    TreeNode left = pq.remove();
    TreeNode right = pq.remove();
    pq.add(new TreeNode(0, left.weight+right.weight, left, right));
} 
TreeNode root = pq.remove();
```

Creating compressed file

- Once we have new encodings, read every character
- Write encoding, not the character, to compressed file
- Why does this save bits?
- What other information needed in compressed file?

- How do we uncompress?
  - How do we know foo.hf represents compressed file?
  - Is suffix sufficient? Alternatives?

- Why is Huffman coding a two-pass method?
  - Alternatives?

Uncompression with Huffman

- We need the trie to uncompress
  - 000100100010011001101111
- As we read a bit, what do we do?
  - Go left on 0, go right on 1
  - When do we stop? What to do?

- How do we get the trie?
  - How did we get it originally? Store 256 int/counts
    - How do we read counts?
  - How do we store a trie? 20 Questions relevance?
    - Reading a trie? Leaf indicator? Node values?

Other Huffman Issues

- What do we need to decode?
  - How did we encode? How will we decode?
  - What information needed for decoding?

- Reading and writing bits: chunks and stopping
  - Can you write 3 bits? Why not? Why?
  - PSEUDO_EOF
  - BitInputStream and BitOutputStream: API

- What should happen when the file won’t compress?
  - Silently compress bigger? Warn user? Alternatives?
Huffman Complexities

- How do we measure? Size of input file, size of alphabet
  - Which is typically bigger?

- Accumulating character counts: ______
  - How can we do this in O(1) time, though not really

- Building the heap/priority queue from counts _____
  - Initializing heap guaranteed

- Building Huffman tree _____
  - Why?

- Create table of encodings from tree _____
  - Why?

- Write tree and compressed file _____