Towers of Hanoi

- Move disks from "from" peg to "to" peg
- What is the recurrence relation in terms of numDisks?

```cpp
void Move(int from, int to, int aux, int numDisks)
// pre: numDisks on peg from,
// post: numDisks moved to peg to
{
    if (numDisks == 1)
        cout << from << " to " << to << endl;
    else
    {
        Move(from, aux, to, numDisks-1);
        Move(from, to, aux, 1);
        Move(aux, to, from, numDisks-1);
    }
}
```

Backtracking, Search, Heuristics

- Many problems require an approach similar to solving a maze
  > Certain mazes can be solved using the “right-hand” rule
  > Other mazes, e.g., with islands, require another approach
  > If you have “markers”, leave them at intersections, don’t explore the same place twice

- What happens if you try to search the web, using links on pages to explore other links, using those links to …
  > How many web pages are there?
  > What rules to webcrawlers/webspiders follow?
  - Who enforces the rules?
  > Keep track of where you’ve been don’t go there again
  - Any problems with this approach?

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
  > # different tries/column?
- N Queens Demo Applet!
- Backtracking
  > Use “current” row in a col
  > If ok, try next col
  > If fail, back-up, next row

Backtracking idea with N queens

- Try to place a queen in each column in turn
  > Try first row in column C, if ok, move onto next column
  > If solved, great, otherwise try next row in column C, place queen, move onto the next column
    - Must unplace the placed queen to keep going

- What happens when we start in a column, where to start?
  > If we fail, move back to previous column (which remembers where it is/failed)
  > When starting in a column anew, start at beginning
    - When backing up, try next location, not beginning

- Backtracking in general, record an attempt go forward
  > If going forward fails, undo the record and backup
Basic ideas in backtracking search

- We need to be able to enumerate all possible choices/moves
  - We try these choices in order, committing to a choice
  - If the choice doesn't pan out we must undo the choice
    - This is the backtracking step, choices must be undoable
- Process is inherently recursive, so we need to know when the search finishes
  - When all columns tried in N queens
  - When we have found the exit in a maze
  - When every possible move tried in Tic-tac-toe or chess?
    - Is there a difference between these games?
- Summary: enumerate choices, try a choice, undo a choice, this is brute force search: try everything

N queens backtracking: nqueens.cpp

```cpp
bool Queens::SolveAtCol(int col)
// pre: queens placed at columns 0,1,...,col-1
// post: returns true if queen can be placed in column col
// and N queen problem solved (N is square board size)
{
    int k; int rows = myBoard.numrows();
    if (col == rows) return true;
    for(k=0; k < rows; k++)
    {
        if (NoQueensAttackingAt(k,col))
        {
            myBoard[k][col] = true;  // place a queen
            if (SolveAtCol(col+1))
            {    return true;            }
            myBoard[k][col] = false;  // unplace the queen
        }
    }
    return false;
}
```

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?

Backtracking, minimax, game search

- We'll use tic-tac-toe to illustrate the idea, but it's a silly game to show the power of the method
  - What games might be better? Problems?
- Minimax idea: two players, one maximizes score, the other minimizes score, search complete/partial game tree for best possible move
  - In tic-tac-toe we can search until the end-of-the game, but this isn't possible in general, why not?
  - Use static board evaluation functions instead of searching all the way until the game ends
- Minimax leads to alpha-beta search, then to other rules and heuristics
**Minimax for tic-tac-toe**

- Players alternate, one might be computer, one human (or two computer players)
- Simple rules: win scores +10, loss scores –10, tie is zero
  - X maximizes, O minimizes
- Assume opponent plays smart
  - What happens otherwise?
- As game tree is explored is there redundant search?
  - What can we do about this?

---

**yfzhltea byveqye**

- The words above represent a simple substitution cypher
  - Each letter mapped to one other letter, no inconsistencies
  - Often used in cryptogram puzzles (newspaper, online,...)
  - How can we write a computer program to solve this?
- Ideas for solving the problem? Benchmark/ballpark idea to accept (or not)
  -
  -
  -
- Problems on the horizon?

---

**One possible solution in docrypto.cpp**

- Study this for an example of backtracking
  - Similar to N queens: make move, recurse, undo as needed
  - What’s a move in this problem?
- Illustrates a few C++ and OO concepts
  - Static variables and functions: belong to class not object
  - Also called “class variables”, don’t need object to access
  - Must be careful when initializing static variables because order of initialization can be important
- See WordSource object shared by all CryptoMap objects, how and when is the WordSource initialized?

---

**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 cryptograms?
  - Solve small words first
  - Solve large words first
  - Do something else?
- What other optimizations/improvements can we make?
  - See program, cryptomap.cpp and docrypto.cpp