CS280 Notes 2/1

by Christian Sagel

Questions

Linear search for unsorted, binary search for sorted
There's some extra tests that are verbose to help students figure out issues with their algorithm.

We try moving the queen every unit in the row until we find a non-collision.
If there are no collisions, we backtrack to the last queen.

It's like a maze, where you backtrack after trying all paths to the last fork.
Recursive backtracking technique.

Knight's tour.
A perfect tour is hitting every square once before returning to the starting position.
When the knight has lots of options, it should choose the square with the least amount of moves available to it. The intuition is to go for the edge cases.

Assignment

For this assignment, we will be allowed to use anything from the STL.
We will be solving sudoku with a simple recursive algorithm.
We find numbers that work, then moving backwards once we detect a collision.
Helper functions, my friend
A function given a number and a position on the board, find if a collision
Given a square, find all numbers in the neighborhood.

Once we have all the support routines.
Validate board. A sanity check. As we write code, we should have this debugging as we run; we can turn it off after.
Make sure there's no duplicates in the board.

Mead has a valid sudoku solver for our friends.
The driver uses getopt.
We have to tell the driver what collisions to detect.

The Sudoku code is completely independent of graphics systems or anything else. The code should be kept clean.
The interface is clear.

No hardcoding by sizes.
The entire assignment can be done algorithmically, no need hard-code anything.
We can do whatever we want internally, as long as we convert whatever we have into what the driver wants.
Call the SUDOKU_CALLBACK function on each move.

On the CALL_BACK, we check if single-step on boolean is on. It has a return value. If if it returns abort, it stops the algorithm and cleans up what it was doing.
Don't go looking up long_jump.

Do not use exceptions to back out.
Abort (stop)
Historically the easiest assignment out of the memory manager and BLIST.

Collections

Collections of Data
Collections of ADT.
The stack is one of the most powerful data structures in data programming. (LIFO) Last in, first out.
push back complexity; O(k) // constant
shrink: O(n)

Freeing memory is constant time.
Complexity of destructor for array-based: constant, unless it's an UDT. For linked lists, it is O(n)

Postfix notation has the operators after the operands. (34+). This is also called RPN for Reverse Polish Notation. Many calculators were made this way.
A stack is the perfect data structure to implement this paradigm.
The algorithm:
When we see an operand, we push it on the stack.
When we see an operator we pop the 2 top items (operands), perform the arithmetic, then push the result of the arithmetic.