What did we learned in the last class?

Flashback: Two versions of Game of Life.
Lesson from Last Class

Program Analysis and Program Design are closely interrelated. A good computer engineer must know both.

In this course we will learn a host of new powerful programming techniques. Along with we will learn more formal methods for analyzing their performance.

Technique of Recursion
Concept of Recursion

- Let us consider a set of nested subroutines...

Program Stack

In recursive program, instead of one routine calling a different routine, one routine can repeatedly calls itself.
Recursion

- Recursion is a powerful tool which can make the solution of many difficult problem astonishingly easy.
- It is a powerful tool to divide and conquer complex problems.
- However, it is also very important to carefully analyze a recursive solution.
- In this class we will see two examples of recursive solutions, and will learn techniques how to analyze recursive programs.

Tower of Hanoi

This is task which is underway at the Temple of Brahma. At the creation of the world, the priest were given a brass platform on which were 3 diamond needles. On the first needle were stacked 64 golden disks, each one slightly smaller than the one under it. The priest were assigned the task of moving all the golden disks from the first needle to the third. The end of the task will signify the end of the world.
Solution

- Solution:

**Move(64,1,3,2)**

- Meaning: Move 64 disks from tower 1 to tower 3 using tower 2 as temporary.

Solution (Divide and Conquer)

- Step 1:
  - Move (63,1,2,3)
  - printf("Move disk #64 from tower 1 to tower 3\n");
  - Move(63,2,3,1)

- Step 2?
Structure of Recursive Program

Every recursive process consists of two parts:
1. A smallest, base case that is processed without recursion; and
2. A general method that reduces a particular case to one or more of the smaller cases, thereby making progress toward eventually reducing the problem all the way to the base case.

Solution

```c
/* Move: moves count disks from start to finish using temp
for temporary storage. */
void Move(int count, int start, int finish, int temp)
{
    if (count > 0) {
        Move(count-1, start, temp, finish);
        printf("Move a disk from %d to %d.\n", start, finish);
        Move(count-1, temp, finish, start);
    }
}
```
Demonstration: Tower of Hanoi

Hanoi.exe

```
Move(2, 1, 3, 2)
```

```
Move(1, 1, 2, 3)
```

```
Move(2, 1, 3, 2)
```

```
Move(3, 1, 2, 3)
```

```
Move(3, 1, 2, 3)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
```

```
Move(2, 1, 3, 2)
Analysis

• Recursion Tree

Height & Number of Nodes

• Number of Nodes = 1 + 2 + 4 + … … .. 2^63 = 2^64 - 1

How Large is this number?

• 10^3 ≈ 2^10

• Let the priest can perform
  – one move per second then it will take:
    – 2^64 > 2^4 . 2^60 = 16 x 10^{18} secs

• There are about:
  – 3.2 x 10^7 seconds in a year.
  – The life of universe is 20 billion years.
  – It will take 25 times more to complete the task!

• Computers will fail
  – because of time.
  – How much space will be required?
A Useful Case

A Fruitful Application of Recursion:

n-queen problem

Apparenty an analytically unsolvable problem. Even C. F. Gauss, who attempted this in 1850 was perplexed by this problem. But, solution does exists. See the two shown above.
Solution Outline

```c
void AddQueen(void)
{
    for (every unguarded position p on the board) {
        Place a queen in position p;
        n++;
        if (n == 8)
            Print the configuration;
        else
            AddQueen();
        Remove the queen from position p;
        n--;
    }
}
```

Example 4-Queen

(a) Dead end
(b) Dead end
Example 4-Queen

Backtracking: Build correct partial solution and proceed. When an inconsistent state arises, the algorithm backs up to the point of last correct partial solution.

Choice of Data Structure

- Boolean array or integer array?
  - Keep a count of check, to help backtracking.
- Search later or mark ahead?
- Pigeon hole principle
  - use one row one queen to reduce search.
- Keep track of free columns
  - int col[8]
- Keep track of free diagonals
  - number of diagonal 2*boardsize - 1
  - all downdiagonal (x-y)=constant
  - all updiagonal (x+y)=constant
#include "common.h"
define BOARDSIZE 8
define DIAGONAL (2*BOARDSIZE-1)
define DONOFFSET 7

void WriteBoard(void);
void AddQueen(void);

int queencol[BOARDSIZE]; /* column with the queen */
Boolean colfree[BOARDSIZE]; /* is the column free? */
Boolean upfree[DIAGONAL]; /* is the upward diagonal free? */
Boolean downfree[DIAGONAL]; /* is the downward diagonal free? */

int queencount = -1, /* row whose queen is currently placed */
    numsol = 0; /* number of solutions found so far */

int main(void)
{
    int i;
    for (i = 0; i < BOARDSIZE; i++)
        colfree[i] = TRUE;
    for (i = 0; i < DIAGONAL; i++)
    {
        upfree[i] = TRUE;
        downfree[i] = TRUE;
    }
    AddQueen();
    return 0;
}

void AddQueen(void)
{
    int col; /* column being tried for the queen */
    queencount++;
    for (col = 0; col < BOARDSIZE; col++)
    {
        if (colfree[col] && upfree[queencount + col] &&
            downfree[queencount - col + DONOFFSET])
        {
            /* Put a queen in position (queencount, col). */
            queencol[queencount] = col;
            colfree[col] = FALSE;
            upfree[queencount + col] = FALSE;
            downfree[queencount - col + DONOFFSET] = FALSE;
        }
        else
            WriteBoard();
        if (queencount == BOARDSIZE-1) /* termination condition */
            /* Proceed recursively. */
            else
                AddQueen();
        /* Now backtrack by removing the queen. */
        colfree[col] = TRUE;
        upfree[queencount + col] = TRUE;
        downfree[queencount - col + DONOFFSET] = TRUE;
    }
    queencount--;
}
Analysis

- Naïve approach:
  - generate a random configuration and test it.
  \[
  \binom{64}{8} = 4,426,165,368
  \]
- One queen per row
  \[8^8 = 16,777,216\]
- One queen per column
  \[8! = 40,320\]