The Power of Recursion: Brute force

- Consider the TypingJob APT problem: What is minimum number of minutes needed to type n term papers given page counts and three typists typing one page/minute? (assign papers to typists to minimize minutes to completion)
  - Example: \{3, 3, 3, 5, 9, 10, 10\} as page counts

- How can we solve this in general? Suppose we're told that there are no more than 10 papers on a given day.
  - How does this constraint help us?
  - What is complexity of using brute-force?

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?

Backtracking with Boggle

- Boggle™ played on 4x4 board
  - Other sizes possible
  - Form words by connecting letters horizontally, vertically, diagonally
  - Cannot re-use letters (normally)

- Two approaches
  - Build words from connections, find partial words in dictionary
  - Look up every word in the dictionary on the board
- Which is better? How is backtracking used?

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

- 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 all board locations tried in boggle
  - When every possible moved 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

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?

Recasting the problem

- Instead of writing this function, write another and call it

```java
// @return min minutes to type papers in pages
int bestTime(int[] pages)
{
    return best(pages, 0, 0, 0);
}
```
- What cases do we consider in function below?

```java
int best(int[] pages, int index,
    int t1, int t2, int t3)
// returns min minutes to type papers in pages
// starting with index-th paper and given
// minutes assigned to typists, t1, t2, t3
{
}
```

Loop Invariants

- Want to reason about the correctness of a proposed iterative solution
- Loop invariants provide a means to effectively about the correctness of code

```java
while !done do
    // what is true at every step
    // Update/iterate
    // maintain invariant
od
```

Bean Can game

- Can contains N black beans and M white beans initially
- emptied according the following repeated process
  - Select two beans from the can
  - If the beans are:
    - same color: put a black bean back in the can
    - different colors: put a white bean back in the can
  - Player who chooses the color of the remaining bean wins the game
- Analyze the link between the initial state and the final state
- Identify a property that is preserved as beans are removed from the can
  - Invariant that characterizes the removal process
**Bean Can Algorithm**

```plaintext
while (num-beans-in-can > 1) do
    pick 2 beans randomly
    if bean1-color == bean2-color then
        put-back black bean
    else
        put-back white bean
    od
```

**Bean Can Analysis**

- What happens each turn?
  - Number of beans in can is decreased by one
  - Number of white beans is either reduced by 2 or 0
  - Number of black beans is either reduced by 1 or 0
- Examine the final states for 2 bean and 3 bean initial states
- Any guesses for the correct strategy?
  - What is the process invariant?

---

**The Game of Nim**

- Two Piles of counters with N and M counters in each pile
- 2 players take turns:
  - Remove some number of counters (≥ 1) from one pile
  - Player who removes last counter wins
- Properties
  - Complete information: could exhaustively search for winning solution
  - Impartial: same moves are available for each player

**Nim Analysis**

- Denote state by (x,y): number of counters in each pile
- What about simple case of (1,1)?
- For whom is (1,1) a “safe” state?
- How about (1,2) or (1,3)?
- How about (2,2)?
- What is the invariant to be preserved by the winning player?
**Numbers from Ends**

- Game begins with some even number of numbers on a line
  
  10  5  7  9  6  12
- Players take turns removing numbers from the ends while keeping running sum of numbers collected so far
- Player with largest sum wins
- Complete information but how to win without search?

**Patterns**

"Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice”

- Alexander et. al, 1977
- A text on architecture!

- What is a programming or design pattern?
- Why are patterns important?

**What is a pattern?**

- “... a three part rule, which expresses a relation between a certain context, a problem, and a solution. The pattern is, in short, at the same time a thing, ..., and the rule which tells us how to create that thing, and when we must create it.”
  
  Christopher Alexander

- name factory, aka virtual constructor
- problem delegate creation responsibility: expression tree nodes
- solution createFoo() method returns aFoo, bFoo,...
- consequences potentially lots of subclassing, ...

- more a recipe than a plan, micro-architecture, frameworks, language idioms made abstract, less than a principle but more than a heuristic
- patterns capture important practice in a form that makes the practice accessible

**Patterns are discovered, not invented**

- You encounter the same “pattern” in developing solutions to programming or design problems
  
  ➢ develop the pattern into an appropriate form that makes it accessible to others
  ➢ fit the pattern into a language of other, related patterns

- Patterns transcend programming languages, but not (always) programming paradigms
  
  ➢ OO folk started the patterns movement
  ➢ language idioms, programming templates, programming patterns, case studies
Programming Problems

- Microsoft interview question (1998)
- Dutch National Flag problem (1976)
- Remove Zeros (AP 1987)
- Quicksort partition (1961, 1986)
- Run-length encoding (SIGCSE 1998)

Removing Duplicates

```java
void crunch(ArrayList<String> list) {
    int lastUniqueIndex = 0;
    String lastUnique = list.get(0);
    for (int k = 1; k < list.size(); k++) {
        String current = list.get(k);
        if (current != lastUnique) {
            list.set(++lastUniqueIndex, current);
            lastUnique = current;
        }
        for (int k = list.size() - 1; k > lastUniqueIndex; k--) {
            list.remove(k);
        }
    }
}
```

One loop for linear structures

- Algorithmically, a problem may seem to call for multiple loops to match intuition on how control structures are used to program a solution to the problem, but data is stored sequentially, e.g., in an array or file. Programming based on control leads to more problems than programming based on structure. Therefore, use the structure of the data to guide the programmed solution: one loop for sequential data with appropriately guarded conditionals to implement the control.

Consequences: one loop really means loop according to structure, do not add loops for control: what does the code look like for run-length encoding example?

What about efficiency?

Coding Pattern

- Name:
  - one loop for linear structures
- Problem:
  - Sequential data, e.g., in an array or a file, must be processed to perform some algorithmic task. At first it may seem that multiple (nested) loops are needed, but developing such loops correctly is often hard in practice.
- Solution:
  - Let the structure of the data guide the coding solution. Use one loop with guarded/if statements when processing one-dimensional, linear/sequential data
- Consequences:
  - Code is simpler to reason about, facilitates develop of loop invariants, possibly leads to (slightly?) less efficient code