CSc 445
Algorithms

The Greatest Hits
Greatest Hits of 445

Brute Force Algorithms

“When in doubt, use brute force”
- Ken Thompson, creator of UNIX
Brute Force Algorithms

- Brute Force algorithms simply exhaust all possibilities:
  
  ```
  foreach (possible solution)
  if (solution works)
    announce it
  report “no solution”
  ```

- Sometimes, it's the best we know
- Often OK for a first draft or simple program
Brute Force Algorithms

Group Discussion:

How many different passwords of length 12 are there? (Assume no limitations to their content.)

If we checked 1 billion passwords per second (!), how long would it take to check all of them?
Brute Force Algorithms

<table>
<thead>
<tr>
<th>Password Length</th>
<th>Num Passwords</th>
<th>Time (10^9 / sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>$2^{36} \sim 6.9 \times 10^{10}$</td>
<td>1 minute</td>
</tr>
<tr>
<td>7</td>
<td>$2^{42} \sim 4.4 \times 10^{12}$</td>
<td>1 hour</td>
</tr>
<tr>
<td>8</td>
<td>$2^{48} \sim 2.8 \times 10^{14}$</td>
<td>2.5 days</td>
</tr>
<tr>
<td>9</td>
<td>$2^{54} \sim 1.8 \times 10^{16}$</td>
<td>150 days</td>
</tr>
<tr>
<td>10</td>
<td>$2^{60} \sim 1.2 \times 10^{18}$</td>
<td>30 years</td>
</tr>
<tr>
<td>11</td>
<td>$2^{66} \sim 7.4 \times 10^{19}$</td>
<td>1800 years</td>
</tr>
<tr>
<td>12</td>
<td>$2^{72} \sim 4.7 \times 10^{21}$</td>
<td>11 millenia</td>
</tr>
</tbody>
</table>

Assuming: roughly 64 allowed characters
Group Discussion:

What is the mass of the Earth, in grams? (Look it up.)

How long would your password need to be to have as many combinations as the Earth has grams?
## Brute Force Algorithms

<table>
<thead>
<tr>
<th>Password Length</th>
<th>Num Passwords</th>
<th>Time ((10^9 / \text{sec}))</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>(2^{90} \sim 1.2 \times 10^{27})</td>
<td>2.3 billion years</td>
</tr>
</tbody>
</table>

6.0 \times 10^{27} \text{ g}

https://en.wikipedia.org/wiki/Earth
Brute Force Algorithms

• Brute force is OK
  – If $n$ is small
  – If $n$ is medium, but you only run the program once
  – If you can't find a better answer

• Classic examples
  – Prime factorization
  – Various combinatoric problems
Brute Force Algorithms

Always look for better options!

- Can you solve a tiny piece of the problem quickly?
  - Greedy

- Can you break the problem in half?
  - Divide and Conquer

- Can you find commonality between possible solutions, and share labor between them?
  - Backtracking
Brute Force Algorithms

Always look for better options!

- Can you record insights, to make future checks faster?
  - Dynamic Programming

- Can you live with “good enough?”
  - Approximation

- Can you tell when you're “close?”
  - Randomization, hill climbing
Greatest Hits of 445

Greedy Algorithms

Image Source: http://hansolav.net/sql/graphs.html
Greedy Algorithms

Greedy algorithms solve the problem a little bit at a time

for (each little piece)
    solve this piece, store it away
Greedy Algorithms

Group Discussion:

What greedy algorithms have we seen
- In this class?
- In previous classes?
Greedy Algorithms

Group Discussion:
What greedy algorithms have we seen
- In this class?
- In previous classes?

Some Answers:
- Binary search
- Dijkstra's

Some Others:
- Making change
- Huffman encoding
Greedy Algorithms

Greedy algorithms are great, but...
...they often don't work!

- You need to prove them correct before you use them
- Or, you need to argue why it's “good enough”
Greatest Hits of 445

Divide and Conquer

Subdivide

Recurse

Merge
Divide and Conquer

• Divide and conquer algorithms break the problem into smaller problems
  - Solve each one
  - Join answers together

• Similar to greedy, though usually dividing into equal parts
Group Discussion:

What divide and conquer algorithms have we seen
• In this class?
• In previous classes?
• Elsewhere?
Group Discussion:

What divide and conquer algorithms have we seen
- In this class?
- In previous classes?
- Elsewhere?

Some Examples:
- Quicksort, MergeSort
- Exponentiation
- MapReduce
Greatest Hits of 445

Backtracking

Go deep

If fail, back up a little and try again
The 8 Queens Problem

- A classic logic puzzle: Can you place 8 Queens on a board such that none can attack any others?
The 8 Queens Problem

In chess, a Queen can move straight or on diagonals, any distance.

So the Queen here attacks many squares.
The 8 Queens Problem

Group Exercise:
If we used brute force, how many combinations would we need to try?

Then: Try solving the problem!
The 8 Queens Problem

This problem was solved in 1850.

How?
The 8 Queens Problem

Group Discussion:

Sketch out a backtracking algorithm for solving the 8 queens problem.

What should you choose/do at each step?

How do you know if you've failed?

What can you store, at each step, to make it cheaper to explore future options?
The 8 Queens Problem

In the 8 Queens problem, we mark off which squares in the board are attacked.
We now only consider squares which are not yet attacked as possible locations for another Queen.

The 8 Queens Problem
Backtracking

• In backtracking, we generally build a stack
  – Recursion or a data structure?

• Each element on the stack represents a “step” in the algorithm
  – Complete info, or delta?

• When we fail, we pop the top off the stack, and explore another possibility
Greatest Hits of 445

Dynamic Programming

Taking Notes
Dynamic Programming

- Dynamic programming algorithms solve problems by **recording answers for small subproblems**.
  
  Different than divide-and-conquer because d&c algos use non-overlapping subsets

- Build larger answers (mostly) by combining small pieces.

- Often, we don't know what will matter; many of the things we record might not be important.
Fibonacci

fibonacci(int n):
    if n < 0
        throw some exception
    if n == 0
        return 0
    if n == 1
        return 1
    return fibonacci(n-1) + fibonacci(n-2)

Group Exercise:
What's the runtime cost of this implementation?
Fibonacci

fibonacci(int n):
    if n < 0
        throw some exception
    if n == 0
        return 0
    if n == 1
        return 1
    return fibonacci(n-1) + fibonacci(n-2)

Answer:
Each execution doubles the number of recursive calls...so

\[ O \left( 2^n \right) \]
fibonacci(int n):

Group Exercise:

How to refactor this algorithm to make it faster?

What sort of information is used over and over?

Can we save it away and then just look it up later, when we need it?

Sketch out a new form of the algorithm.
Fibonacci

fib_cache = {}

fibonacci(int n):
    if n in fib_cache:
        return fib_cache[n]
    if n == 1 or n == 0:
        answer = 1
    else:
        answer = fibonacci(n-1)+fibonacci(n-2)
    fib_cache[n] = answer
    return answer
fibonacci = {}
fibonacci(int n):
    if n in fibonacci:
        return fibonacci[n]
    if n == 1 or n == 0:
        answer = 1
    else:
        answer = fibonacci(n-1) + fibonacci(n-2)
    fibonacci[n] = answer
return answer

Now, we mostly get our answers by reading from a dictionary.

Each time that we generate a new answer, we record it in the dictionary.

We only generate each solution once.

Total cost: $O(n)$
Dynamic Programming

- The “all to all” shortest path problem:
  “Given a graph, what is the shortest path from each node to every other node?”

![Graph with nodes and edges showing shortest paths]
In this algorithm, we might record the best known path between each pair of nodes.

We build new answers (longer paths) by joining together very short paths:

\[
\text{path from } f \rightarrow a + \text{ path from } a \rightarrow b \quad \Rightarrow \quad \text{one possible } f \rightarrow b
\]
Greatest Hits of 445

Approximation

Good enough for government work
Approximation

• Occasionally, you can find approximate answers quickly.
  – If they are good enough, live with it.

• Sometimes based on heuristics.
  – “This often works.”

• Sometimes based on lower bounds.
  – “It's not going to get much better than this answer.”
  – Best if we can quantify how much worse you are.
  – Example: traveling salesperson problem
Approximation

Example:

- Finding the shortest path through a real world transportation network
Approximation

Example:

- Finding the shortest path through a real world transportation network

Physical limitations give you good clues about how to find a path.
Approximation

Example:

- Finding the shortest path through a **real world** transportation network

Say you're looking to find a path between the red dots.

A good **heuristic** would be to use northbound lines.

Using an eastbound lines will rarely give a good solution.
Approximation

Example:

- Finding the shortest path through a real world transportation network

The lower bound on the distance traveled is the distance “as the crow flies.”

Any solution that is within 20% of that number is pretty good.

It may not be worth the trouble to find a better one.
Greatest Hits of 445

Randomization, Hill Climbing

Onwards and Upwards
Randomization, Hill Climbing

- **Problem**: Find the *optimal* solution, in a space with many variables.

- **Nice Property**: Results are *contiguous* – meaning a small change in input leads to a small change in output.

- **Difficulty**: Local maxima exist.
Randomization, Hill Climbing

• Often used to optimize a design
  - Cache
  - Engines
  - Wings
  - Roads
  - Distributed computations
Fully Random Algorithms

• What if hill climbing is not possible?
  - Inputs might be discrete
  - Outputs might not be contiguous

• Alternative: simply choose random values
  - Iterate many times
  - Choose the best
  - **Definitely** not guaranteed to find the best
  - But often does “OK”