CSCI-UA 480.4: APS
Algorithmic Problem Solving

Complete Search or Brute Force

Instructor: Joanna Klukowska

created based on materials for this class by
Bowen Yu and materials shared by the authors of
the textbook Steven and Felix Halim

Complete Search

- a.k.a., **brute force**

- solving problems by traversing the entire (or a large part of) search space in order to find the desired solution
- recursive backtracking solutions fall under this category

Challenge: Digit Quotient

Task

Find all pairs of 5-digit numbers that collectively use all 10 digits 0 - 9 once each, such that the first number divided by the second number is equal to a given integer N, where $2 <= N <= 79$.

$$abcde/fghij = N$$

(The first digit of one of the numbers could be zero.)

Example:

N = 62

79546 / 01283

94736 / 01528

Challenge: Digit Quotient

Observation

- `fghij` can range from `01234` to `98765`
  or, even better from `01234` to `(98765/N)`
- given `fghij` we can compute `abcde` as `fghij*N`

Questions

- given two values `tmp1` and `tmp2`, how can be decide if they use all ten digits

Challenge: Digit Quotient

Observation

- `fghij` can range from `01234` to `98765`
  or, even better from `01234` to `(98765/N)`

- given `fghij` we can compute `abcde` as `fghij*N`

Questions

- given two values `tmp1` and `tmp2`, how can be decide if they use all ten digits

Solution

input: N

```
for fghij in 1234 to 98765/N :
    abcde = fghij * N
    check if abcde and fghij collectively use 10 digits
    if they do
        print the values of the two numbers
```

Challenge: Digit Quotient

to check if `abcde` and `fghij` use all 10 digits

- set `used_digits` to (`fghij` < 1000) - this will set the last bit of `used_digits` to 1 if the `f` digit is zero
- set tmp to `abcde`
- while tmp > 0
  - `used_digits` |= 1 << (tmp%10)
  - tmp /= 10
- set tmp to `fghij`
- while tmp > 0
  - `used_digits` |= 1 << (tmp%10)
  - tmp /= 10
- if (used_digits == (1<<10) - 1)
  - all digits are used

Challenge: Solving Equations

Given three integers A, B, C ($1 <= A, B, C <= 10000$) find three other distict values X, Y, Z such that:

$$X + Y + Z = A$$

$$X \times Y \times Z = B$$

$$X^2 + Y^2 + Z^2 = C$$

Challenge: Solving Equations

Given three integers A, B, C ($1 <= A, B, C <= 10000$) find three other distict values X, Y, Z such that:

$$X + Y + Z = A$$

$$X \times Y \times Z = B$$

$$X^2 + Y^2 + Z^2 = C$$

Observation

the possible values of x, y and z are in $[-100, 100]$

- how do we know that?

Challenge: Solving Equations

Given three integers A, B, C ($1 <= A, B, C <= 10000$) find three other distict values X, Y, Z such that:

$$X + Y + Z = A$$

$$X \times Y \times Z = B$$

$$X^2 + Y^2 + Z^2 = C$$

Observation

the possible values of x, y and z are in $[-100, 100]$

- how do we know that?
- since in $X^2 + Y^2 + Z^2 = C$, C can be at most 10,000, none of the values can exceed 100 in absolute value

Challenge: Solving Equations

Algorithm

Give: values of A, B, C

```
for x in -100 .. 100
  for y in -100 .. 100
    for z in -100 .. 100
       if x, y and x are all different
          AND x + y + z = A
          AND x * y * z = B
          AND x*x + y*y + z*z = C
              we have a solution
```

Can we eliminate some of the steps?

Challenge: Solving Equations

Can we eliminate some of the steps?

- because of the second equation, we know that at least of of the above loops has a range of -22 to 22 only (since if all values were x=y=z=22, we would have 22^3>10,000 and 21^3 < 10,000)
- in each loop we can check if the value is not equal to already used value
- for each loop we can check if the current variables (ignoring the contributions from the next one) are able to satisfy the conditions of the three equations

```
for x in -22 .. 22
  if x * x <= C
    for y in -100 .. 100
      if y != x AND x * x + y * y <= C
        for z in -100 .. 100
           if x, y and x are all different
              AND x + y + z = A
              AND x * y * z = B
              AND x*x + y*y + z*z = C
                  we have a solution
```

Queens on a Chess Board

Task Place 8 queens on an 8x8 chess board so that no two queens attack one another.

Queens on a Chess Board

Task Place 8 queens on an 8x8 chess board so that no two queens attack one another.

Solution 1 (most naive)

- enumarate all combinations of 8 different cells out of the 64 possibilities and see which of them provide a solution
  performance: ${}_{64}C_8 \approx 4\text{billion}$

Queens on a Chess Board

Task Place 8 queens on an 8x8 chess board so that no two queens attack one another.

Solution 1 (most naive)

- enumarate all combinations of 8 different cells out of the 64 possibilities and see which of them provide a solution
  performance: ${}_{64}C_8 \approx 4\text{billion}$

Solution 2 (still naive, but a bit faster)

- observation: each queen has to be in its own column
- for each column, pick a single row position
  performance: $8^8 \approx 17\text{million}$

Queens on a Chess Board

Task Place 8 queens on an 8x8 chess board so that no two queens attack one another.

Solution 1 (most naive)

- enumarate all combinations of 8 different cells out of the 64 possibilities and see which of them provide a solution
  performance: ${}_{64}C_8 \approx 4\text{billion}$

Solution 2 (still naive, but a bit faster)

- observation: each queen has to be in its own column
- for each column, pick a single row position
  performance: $8^8 \approx 17\text{million}$

Solution 3 (a bit faster than the previous one)

- observation: each queen has to be in its own column and its own row
- for each column, pick a single row position that is not equal to any previously chosen row
  performance: $8! \approx 40\text{thousand}$

Queens on a Chess Board
Backtracking

Solution 4 (getting there)

- observation 1: each queen has to be in its own
  - column
  - row
  - diagonal
- observation 2: no point in continuing to generate solutions that violate one of the above conditions (this gives us backtracking: prune the paths that do not lead to a valid solution)

performance: $\mathrm{sub}O(n!)$ where n is the size of the chessboard (= number of queens)

See [handout] for the implementation.

Queens on a Chess Board
Backtracking

- With 8 queens on an 8x8 chessboard we have under $8! \approx 40 \mathrm{thousand}$ operations.
- If each operation takes $\approx 10^{-8}$ seconds, than this algorithm completes in under a second for 8 queens.

Queens on a Chess Board
Backtracking

- With 8 queens on an 8x8 chessboard we have under $8! \approx 40 \mathrm{thousand}$ operations.
- If each operation takes $\approx 10^{-8}$ seconds, than this algorithm completes in under a second for 8 queens.
- What happens if you use a larger chessboard (i.e. larger number of queens)? for example $n = 14$

Other problems to looke at

- 15-puzzle Game
- Bishops on a Chessboard
- Tug of War
- CD to Tape
- Squares

Making the *complete search* work

- filter vs. generate
  generate all possible solutions rather than starting with a set of all options and removing the impossible one

- prune infeasible search space areas as soon as possible

- pre-calculate and store results that may need to be used again (trade memory for time)

- optimize your source code:

  - code in C++, not Java ☹
  - use efficient i/o routines (`scanf`/`printf` over `cin`/`cout` in C++ `BufferedReader`/`BufferedWriter` over `Scanner`/`System.out` in Java)
  - use cache friendly algorithms: consecutive memory accesses in an array rather than jumping around
  - access 2D arrays in a row major order
  - use `int`/`long` or `bitset`/`Bitset` for manipulating boolean data, not arrays or vectors of boolean variables
  - use an array over `Vector`/`ArrayList`
  - declare large data structures and objects once and reuse them
  - use iterative implementation over recursive implementations (assuming same level of difficulty in coding)
  - use c-string instead of C++ `string` class; use `StringBuffer` instead of `String` class in Java