Complete Search (aka Brute Force)

Section 3.1–3.2

Dr. Mayfield and Dr. Lam

Department of Computer Science
James Madison University

Oct 16, 2015

CS: Just do it!

Complete search = try every possibility

- Should never give you Wrong Answer
- Might result in Time Limit Exceeded

> Example: find the min/max number in an array

KISS principle http://en.wikipedia.org/wiki/KISS_principle

- Keep it simple, stupid
- Keep it short and simple
- Keep it simple and straightforward

Iterative search

Example 1: Selection sort

- For each position, find the max (of what's left)

Example 2: Naive $GCD(m, n)$

- Try all k from $min(m, n)$ to 1
- Test if each k divides m and n

Example 3: Primality test

- Try all k from 2 to $\sqrt{n}$
- Test if each k divides n

UVa 725: Division

Find and display all pairs of 5-digit numbers that:

1. Use the digits 0 through 9 once each
2. first number / other number == N

Hints: just try them all

- Search from 01234 to 98765 / N
- Use a bitset to track digits 0..9
- Other number = first number * N

UVa 11742: Social Constraints

n movie goers with m seating constraints

- e.g., a and b must be at most c seats apart
- How many possible seating arrangements?

Hint: explore all permutations

```
#include <algorithm>

int p[] = {0, 1, 2, 3, 4, 5, 6, 7};
do {
    // check each social constraint
}
while (next_permutation(p, p + 8));
```

Java permutations

```
import java.util.*;

class Permute<T>
{
    public void permute(List<T> items, int n)
    {
        if (n == items.size()-1) {
            // handle a permutation
        } else {
            for (int i=n; i<items.size(); i++) {
                Collections.swap(items, i, n);
                permute(items, n+1);
                Collections.swap(items, i, n);
            }
        }
    }
}
```

Permute examples

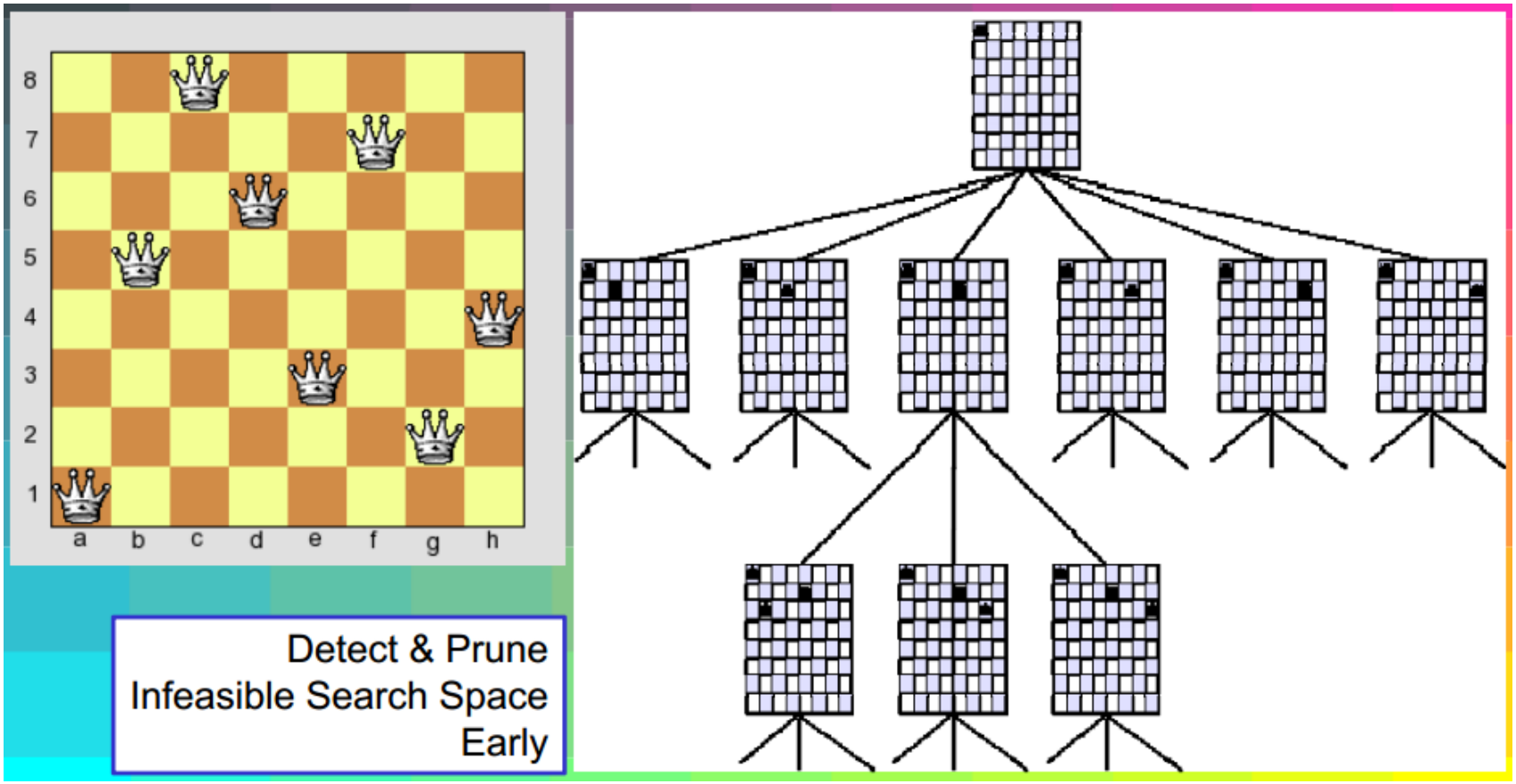
```
public static void main(String[] args)
{
    List<String> letters = Arrays.asList("A", "B", "C", "D");
    (new Permute<String>()).permute(letters, 0);

    List<Integer> nums = Arrays.asList(1, 2, 3, 4, 5);
    (new Permute<Integer>()).permute(nums, 0);
}
```

What if the search space is too big?

Don't explore infeasible/inferior solutions.

Pruning the search space



http://en.wikipedia.org/wiki/Eight_queens_puzzle

$\binom{64}{8} = 4{,}426{,}165{,}368$ possibilities

Recursive backtracking

General approach

- Break down problem into n sub-problems
- Initialize solution to empty, then Solve(1)

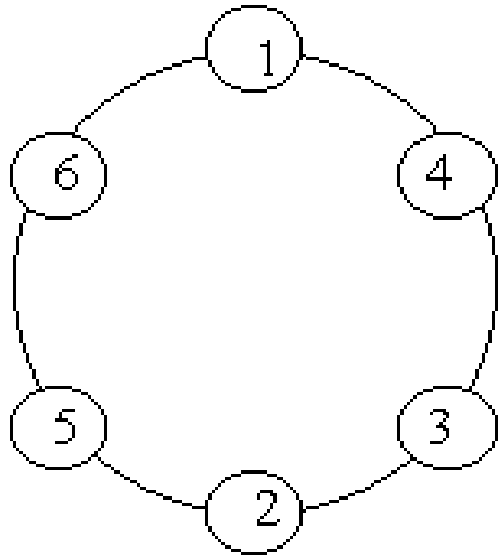
Algorithm Solve(i)

```
for each x ∈ P_i                          solution x to sub-problem i
    add x to solution
    if solution is promising              does not violate constraints
        if i == n
            output solution               solved all the sub-problems
        else
            solve(i + 1)                  recursively solve next sub-prob
    remove x from solution
```

UVa 524: Prime Ring Problem

Reminder from 2nd week:

1. Write code to read the input
   - Debug by printing the input
2. Write code to print the output
   - Double check the formatting
3. Write code to solve the problem
   - In this case, use backtracking



You'll also need to test for primes

- Since $0 < n \leq 16$, brute force is good enough
- http://en.wikipedia.org/wiki/Primality_test

Don't forget to read!

Section 3.2.3 Tips

1. Filtering versus generating
2. Prune search space early
3. Utilize symmetries
4. Pre-Computation
5. Try solving it backwards
6. Optimizing your source code
7. Use a better data structure

C++ Tips of the Week

Standard error

All processes have three I/O streams:

1. Standard in (cin / System.in)
2. Standard out (cout / System.out)
3. Standard error (cerr / System.err)

Tip: print debug output to cerr

```
while (cin >> word)
{
    cerr << "Next word: " << word << endl;
    ...
}
```

- cerr is ignored by most judges (including UVa)
- Excessive output might slow down execution
- Java: use `System.err.println`

64-bit integers

Historical compromise

- C++ `long` may be 32-bits
- C++ `long long` is 64-bits

Java fixed this problem

- `int` is always 32-bits
- `long` is always 64-bits
- `java.math.BigInteger`
  - Implemented with `int[]`
  - Lots of useful methods