Final exam

During finals period. 7:30-10:30 PM on Friday, January 15.

- McCosh 50.

Rules.

including associated readings and assignments (but no serious Java programming)

- Covers all material through this past Tuesday.
- Emphasizes post-midterm material.
- Honor code, closed book, closed note.
- 8.5-by-11 page of notes (one side, in your own handwriting).
- Electronic devices are forbidden.

Final preparation.

- Review session: Wednesday 1/13, 5-7 PM, Room TBA.
- Take old exams, but also read (and understand!) lecture notes.

http://algs4.cs.princeton.edu

Combinatorial Search and Algorithm Design

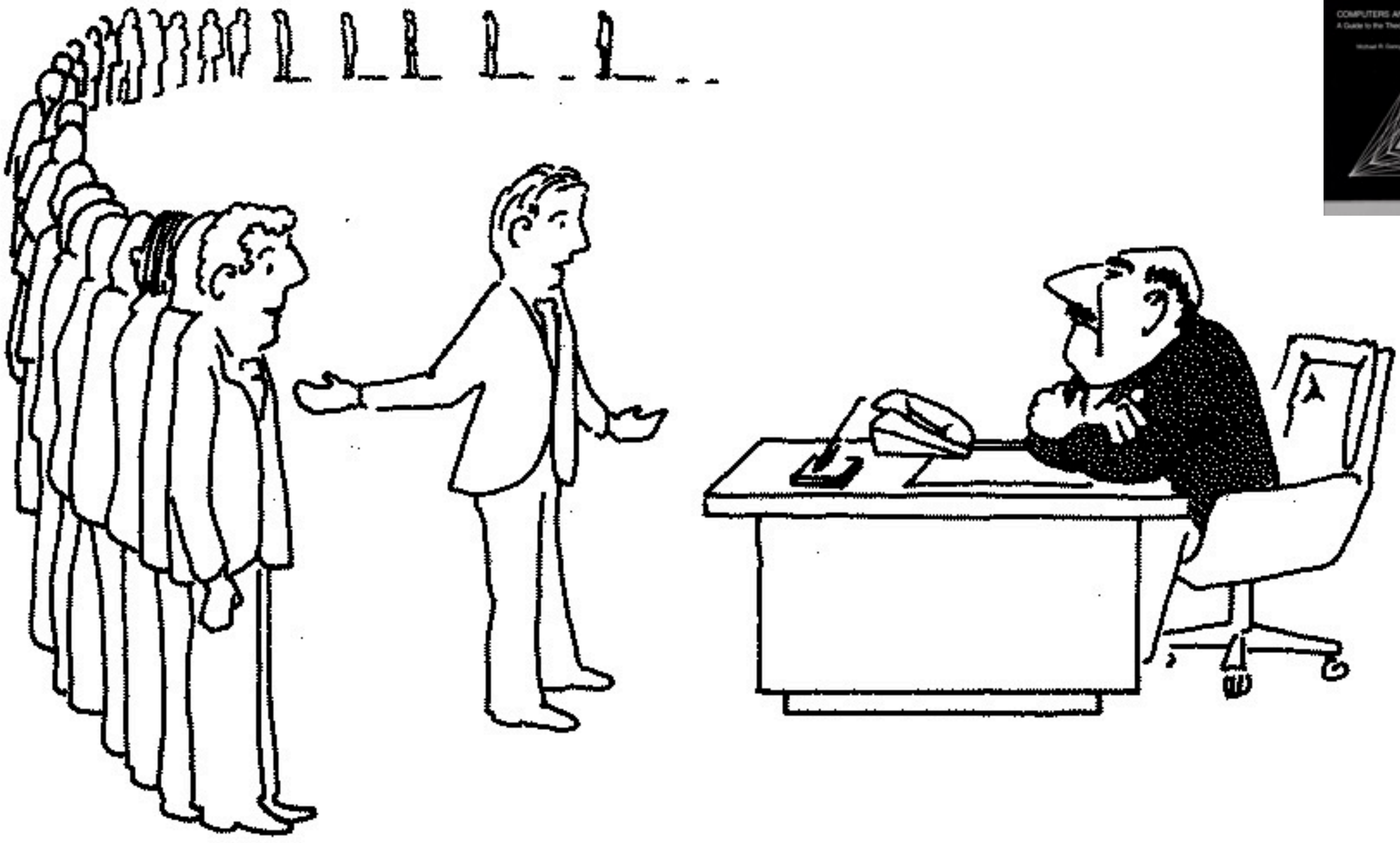
- *introduction*
- *permutations*
- *backtracking*
- *subsets*
- *algorithm design principles*
- *interview questions*

http://algs4.cs.princeton.edu

Combinatorial Search and Algorithm Design

- *introduction*
- *permutations*
- *backtracking*
- *subsets*
- *algorithm design principles*
- *interview questions*

Implications of NP-completeness



“I can’t find an efficient algorithm, but neither can all these famous people.”

Overview

Exhaustive search. Iterate through all elements of a search space.

Applicability. Huge range of problems (including intractable ones).

Caveat. Search space is typically exponential in size $\Rightarrow$ effectiveness may be limited to relatively small instances.

Backtracking. Method for examining feasible solutions to a problem, by systematically pruning infeasible ones.

Warmup: enumerate N-bit strings

Goal. Process all 2^N bit strings of length N.

- Maintain array `a[]` where `a[i]` represents bit `i`.
- Simple recursive method does the job.

```
// enumerate bits in a[k] to a[N-1]
private void enumerate(int k)
{
   if (k == N)
   {  process(); return;  }
   enumerate(k+1);
   a[k] = 1;
   enumerate(k+1);
   a[k] = 0;
}
```

clean up

N = 3

0	0	0
0	0	1
0	0	0
0	1	0
0	1	1
0	1	0
0	0	0
1	0	0
1	0	1
1	0	0
1	1	0
1	1	1
1	1	0
1	0	0
0	0	0

N = 4

0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

a[0] a[N-1]

Remark. Equivalent to counting in binary from 0 to $2^N - 1$.

Warmup: enumerate N-bit strings

```
public class BinaryCounter
{
   private int N;    // number of bits
   private int[] a; // a[i] = ith bit

   public BinaryCounter(int N)
   {
      this.N = N;
      this.a = new int[N];
      enumerate(0);
   }

   private void process()
   {
      for (int i = 0; i < N; i++)
         StdOut.print(a[i]) + " ";
      StdOut.println();
   }

   private void enumerate(int k)
   {
     if (k == N)
     {  process(); return;  }
     enumerate(k+1);
     a[k] = 1;
     enumerate(k+1);
     a[k] = 0;
   }
}
```

```
public static void main(String[] args)
{
   int N = Integer.parseInt(args[0]);
   new BinaryCounter(N);
}
```

```
% java BinaryCounter 4
0 0 0 0
0 0 0 1
0 0 1 0
0 0 1 1
0 1 0 0
0 1 0 1
0 1 1 0
0 1 1 1
1 0 0 0
1 0 0 1
1 0 1 0
1 0 1 1
1 1 0 0
1 1 0 1
1 1 1 0
1 1 1 1
```

all programs in this lecture are variations on this theme

http://algs4.cs.princeton.edu

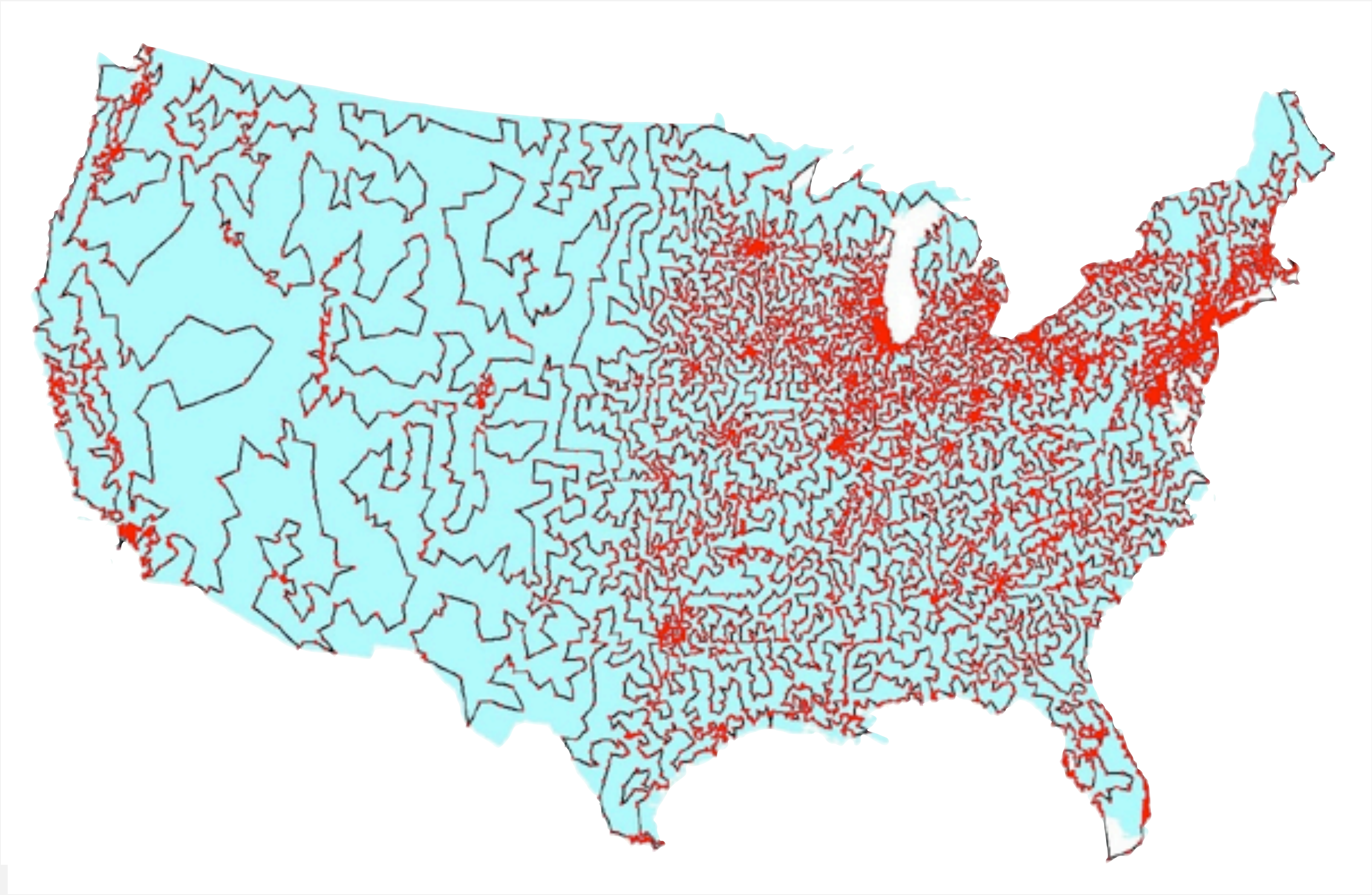
Combinatorial Search and Algorithm Design

- introduction
- *permutations*
- backtracking
- subsets
- algorithm design principles
- interview questions

Traveling salesperson problem

Euclidean TSP. Given N points in the plane, find the shortest tour.

Proposition. Euclidean TSP is NP-hard.

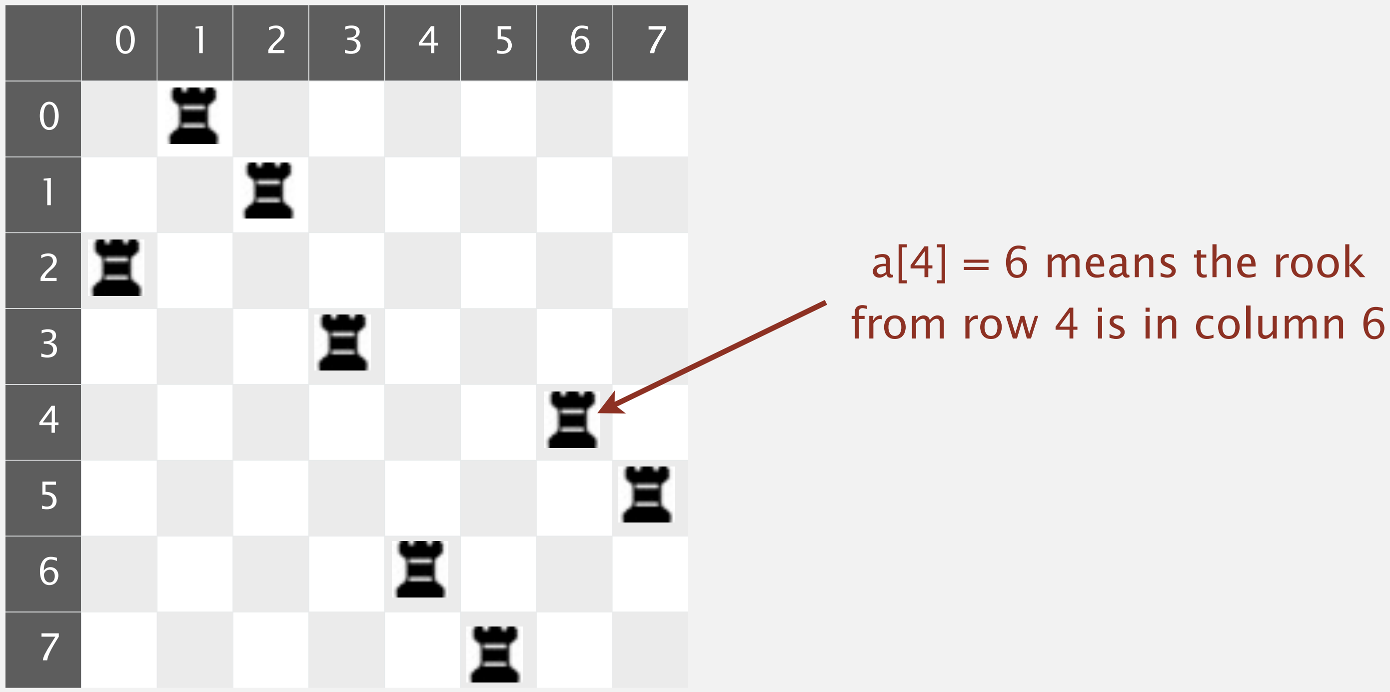


13509 cities in the USA and an optimal tour

Brute force. Design an algorithm that checks all tours by enumerating all possible orderings for cities.

N-rooks problem

Q. How many ways are there to place N rooks on an N-by-N board so that no rook can attack any other?




```
int[] a = { 2, 0, 1, 3, 6, 7, 4, 5 };
```

Representation. No two rooks in the same row or column $\Rightarrow$ permutation.

Challenge. Enumerate all $N!$ permutations of N integers 0 to $N-1$.

Enumerating permutations

Recursive algorithm to enumerate all $N\,!$ permutations of N elements.

- Start with permutation `a[0]` to `a[N-1]`.
- For each value of `i`:
  - swap `a[i]` into position 0
  - enumerate all $(N-1)\,!$ permutations of `a[1]` to `a[N-1]`
  - clean up (swap `a[i]` back to original position)

N = 3

0	1	2
0	2	1
0	1	2
1	0	2
1	2	0
1	0	2
0	1	2
2	1	0
2	0	1
2	1	0
0	1	2

cleanup swaps that bring permutation back to original

0 followed by perms of 1 2 3	1 followed by perms of 0 2 3	2 followed by perms of 1 0 3	3 followed by perms of 1 2 0
0 1 2 3	1 0 2 3	2 1 0 3	3 1 2 0
0 1 3 2	1 0 3 2	2 1 3 0	3 1 0 2
0 2 1 3	1 2 0 3	2 0 1 3	3 2 1 0
0 2 3 1	1 2 3 0	2 0 3 1	3 2 0 1
0 3 2 1	1 3 2 0	2 3 0 1	3 0 2 1
0 3 1 2	1 3 0 2	2 3 1 0	3 0 1 2

a[0] a[N-1]

Enumerating permutations

Recursive algorithm to enumerate all $N\,!$ permutations of N elements.

- Start with permutation of `a[k]` to `a[N-1]` (where initially k=0).
- For each value of `i` starting with k:
  - swap `a[i]` into position k
  - enumerate all $(N-1)\,!$ permutations of `a[k+1]` to `a[N-1]`
  - clean up (swap `a[i]` back to original position)

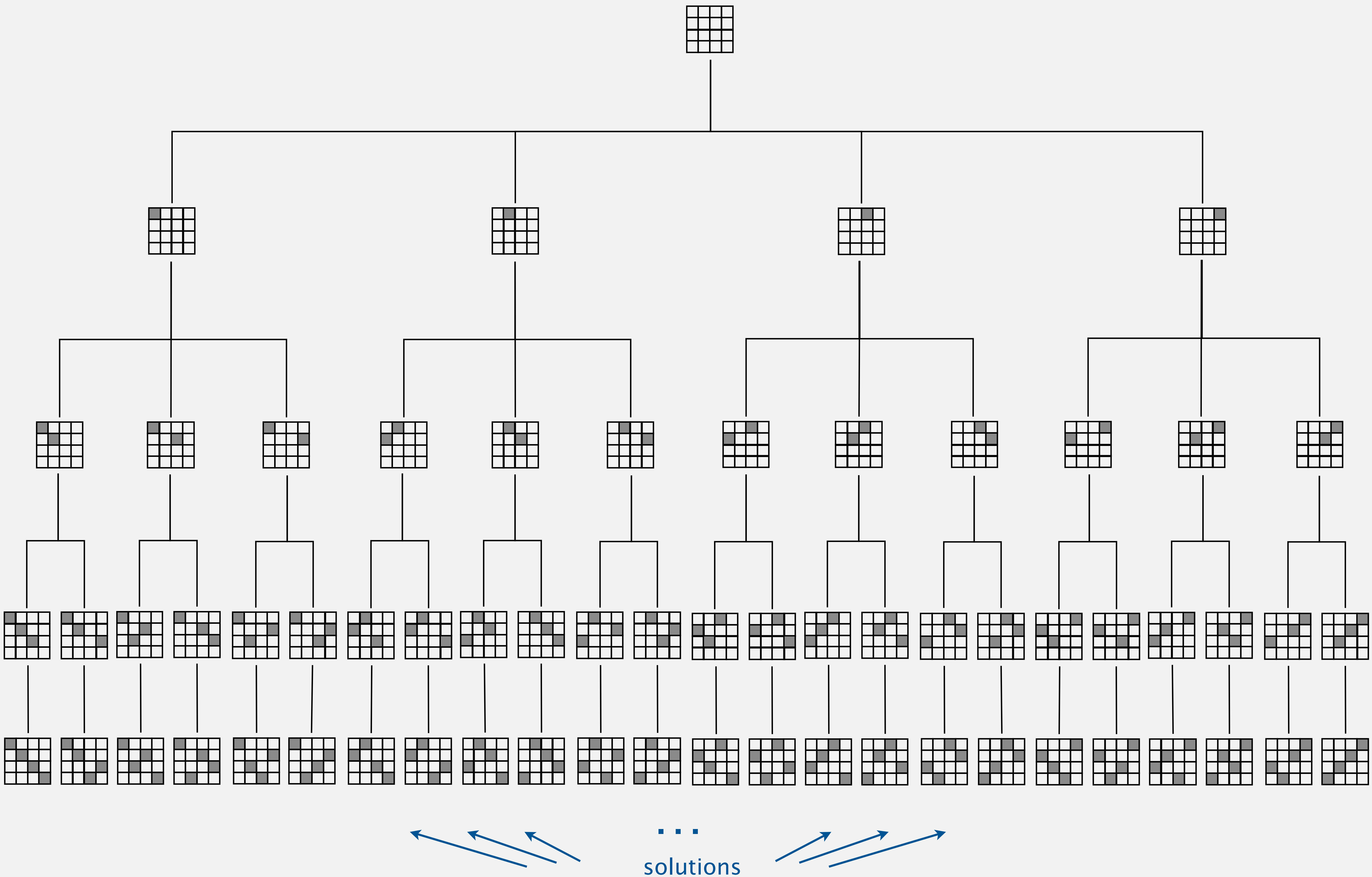
```
// place N-k rooks in a[k] to a[N-1]
private void enumerate(int k)
{
   if (k == N)
   {  process(); return;  }

   for (int i = k; i < N; i++)
   {
      exch(k, i);
      enumerate(k+1);
      exch(i, k);        ← clean up
   }
}
```

Enumerating permutations

```
public class Rooks
{
   private int N;
   private int[] a;  // bits (0 or 1)

   public Rooks(int N)
   {
      this.N = N;
      a = new int[N];
      for (int i = 0; i < N; i++)
         a[i] = i;          ← initial permutation
      enumerate(0);
   }

   private void enumerate(int k)
   { /* see previous slide */  }

   private void exch(int i, int j)
   {  int t = a[i]; a[i] = a[j]; a[j] = t;  }

   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      new Rooks(N);
   }
}
```

```
% java Rooks 2
0 1
1 0

% java Rooks 3
0 1 2
0 2 1
1 0 2
1 2 0
2 1 0
2 0 1
```

4-rooks search tree



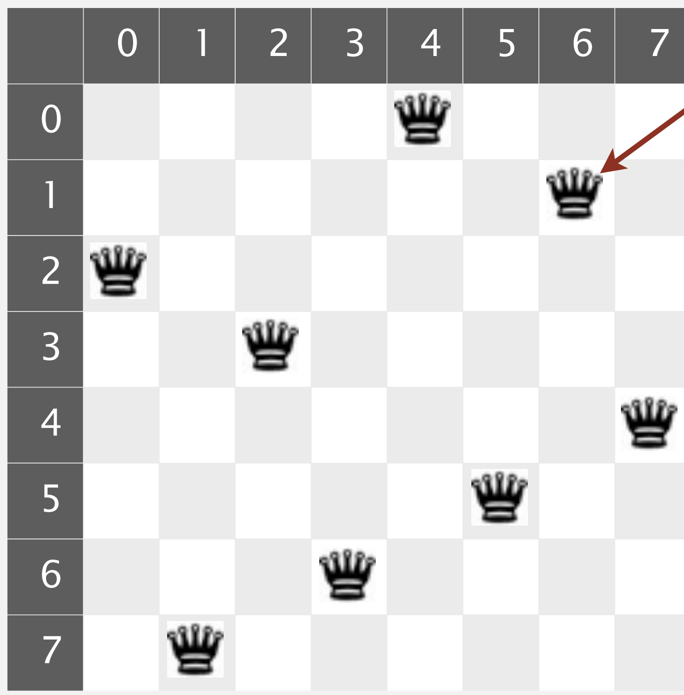
http://algs4.cs.princeton.edu

Combinatorial Search and Algorithm Design

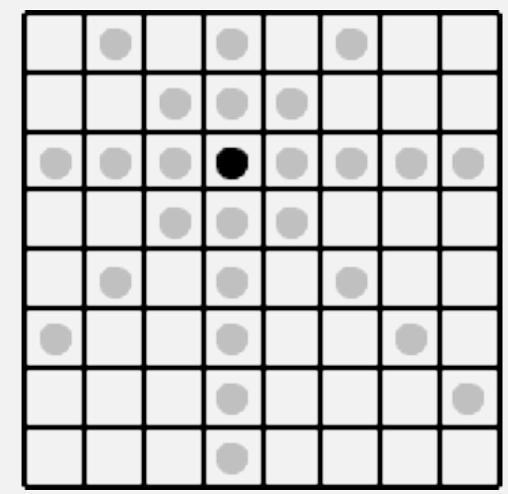
- *introduction*
- *permutations*
- ***backtracking***
- *subsets*
- *algorithm design principles*
- *interview questions*

N-queens problem

Q. How many ways are there to place N queens on an N-by-N board so that no queen can attack any other?



a[1] = 6 means the queen from row 1 is in column 6



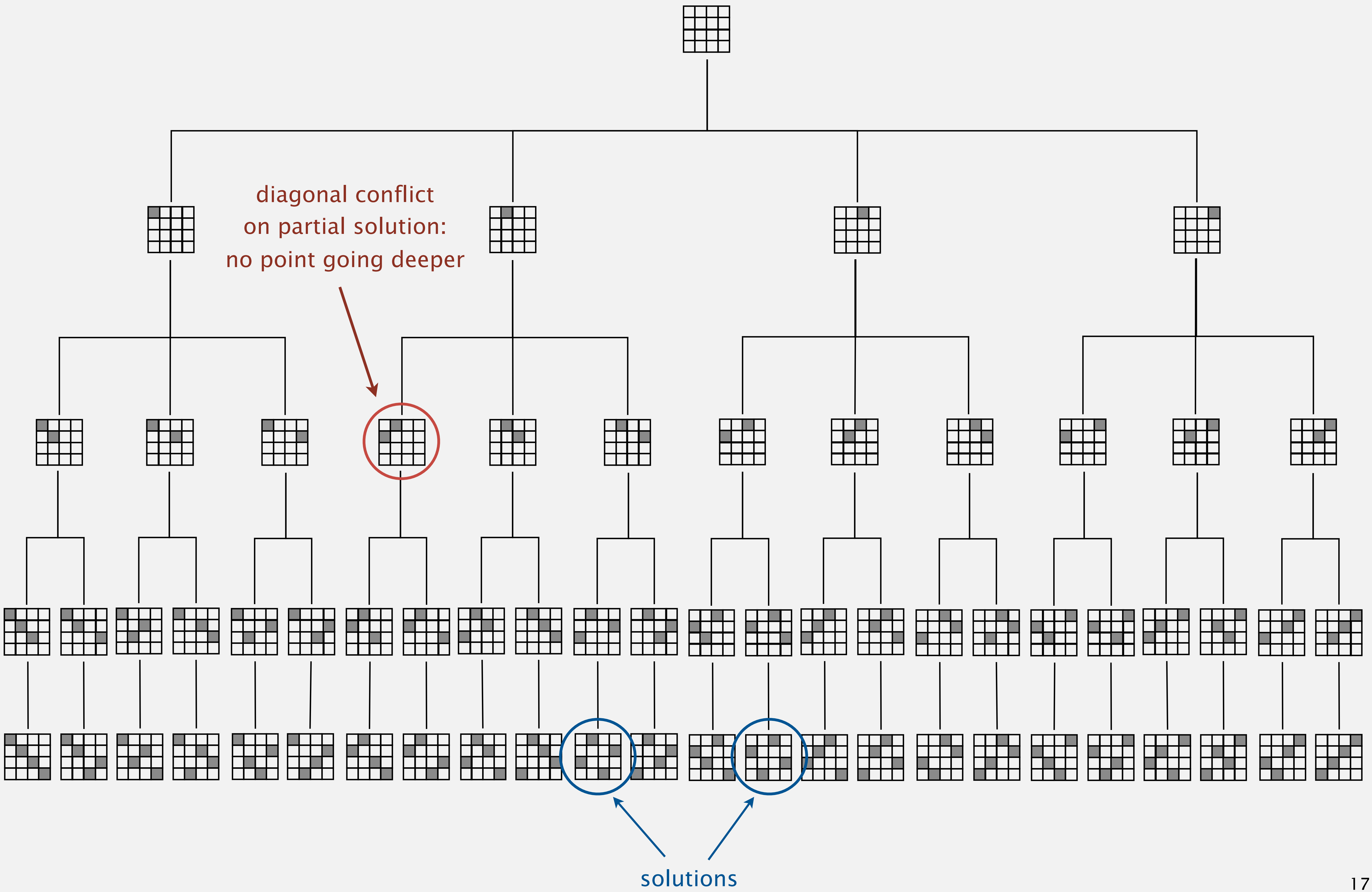
```
int[] a = { 2, 7, 3, 6, 0, 5, 1, 4 };
```

Representation. No 2 queens in the same row or column $\Rightarrow$ permutation.

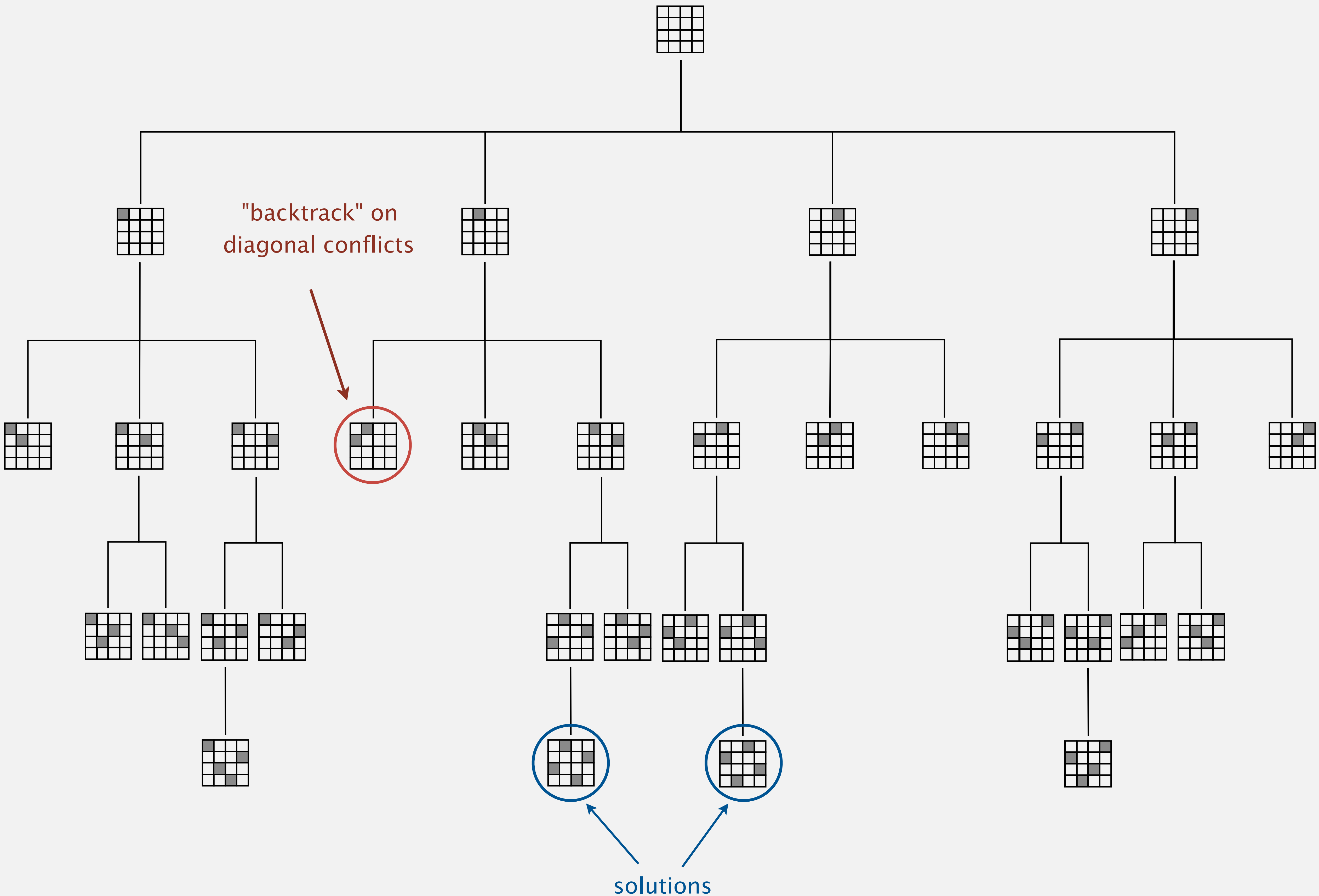
Additional constraint. No diagonal attack is possible.

Challenge. Enumerate (or even count) the solutions. ← unlike N-rooks problem, nobody knows answer for N > 30

4-queens search tree



4-queens search tree (pruned)



Backtracking

Backtracking paradigm. Iterate through elements of search space.

- When there are several possible choices, make one choice and recur.
- If the choice is a dead end, backtrack to previous choice, and make next available choice.

Benefit. Identifying dead ends allows us to prune the search tree.

Ex. [backtracking for N-queens problem]

- Dead end: a diagonal conflict.
- Pruning: backtrack and try next column when diagonal conflict found.

Applications. Puzzles, combinatorial optimization, parsing, ...

N-queens problem: backtracking solution

```
private boolean canBacktrack(int k)
{
   for (int i = 0; i < k; i++)
   {
      if ((a[i] - a[k]) == (k - i)) return true;
      if ((a[k] - a[i]) == (k - i)) return true;
   }
   return false;
}

// place N-k queens in a[k] to a[N-1]
private void enumerate(int k)
{
   if (k == N)
   {  process(); return;  }

   for (int i = k; i < N; i++)
   {
      exch(k, i);
      if (!canBacktrack(k)) enumerate(k+1);
      exch(i, k);
   }
}
```

stop enumerating if adding queen k leads to a diagonal violation

```
% java Queens 4
1 3 0 2
2 0 3 1

% java Queens 5
0 2 4 1 3
0 3 1 4 2
1 3 0 2 4
1 4 2 0 3
2 0 3 1 4
2 4 1 3 0
3 1 4 2 0
3 0 2 4 1
4 1 3 0 2
4 2 0 3 1

% java Queens 6
1 3 5 0 2 4
2 5 1 4 0 3
3 0 4 1 5 2
4 2 0 5 3 1
```

a[0] a[N-1]

N-queens problem: effectiveness of backtracking

Pruning the search tree leads to enormous time savings.

N	Q(N)	N !	time (sec)
8	92	40,320	–
9	352	362,880	–
10	724	3,628,800	–
11	2,680	39,916,800	–
12	14,200	479,001,600	1.1
13	73,712	6,227,020,800	5.4
14	365,596	87,178,291,200	29
15	2,279,184	1,307,674,368,000	210
16	14,772,512	20,922,789,888,000	1352

Conjecture. $Q(N) \sim N! \,/\, c^N$, where c is about 2.54.

Hypothesis. Running time is about $(N! \,/\, 2.5^N) \,/\, 43{,}000$ seconds.

Some backtracking success stories

TSP. Concorde solves real-world TSP instances with ~ 85K points.

- Branch-and-cut.
- Linear programming.
- ...

Combinatorial
Optimization and
Networked
Combinatorial
Optimization
Research and
Development
Environment

SAT. Chaff solves real-world instances with ~ 10K variables.

- Davis-Putnam backtracking.
- Boolean constraint propagation.
- ...

Chaff: Engineering an Efficient SAT Solver

Matthew W. Moskewicz
Department of EECS
UC Berkeley
moskewcz@alumni.princeton.edu

Conor F. Madigan
Department of EECS
MIT
cmadigan@mit.edu

Ying Zhao, Lintao Zhang, Sharad Malik
Department of Electrical Engineering
Princeton University
{yingzhao, lintaoz, sharad}@ee.princeton.edu

ABSTRACT

Boolean Satisfiability is probably the most studied of combinatorial optimization/search problems. Significant effort has been devoted to trying to provide practical solutions to this problem for problem instances encountered in a range of applications in Electronic Design Automation (EDA), as well as in Artificial Intelligence (AI). This study has culminated in the

Many publicly available SAT solvers (e.g. GRASP [8], POSIT [5], SATO [13], rel_sat [2], WalkSAT [9]) have been developed, most employing some combination of two main strategies: the Davis-Putnam (DP) backtrack search and heuristic local search. Heuristic local search techniques are not guaranteed to be complete (i.e. they are not guaranteed to find a satisfying assignment if one exists or prove unsatisfiability); as a

http://algs4.cs.princeton.edu

Combinatorial Search and Algorithm Design

- *introduction*
- *permutations*
- *backtracking*
- ***subsets***
- *algorithm design principles*
- *interview questions*

Enumerating subsets: natural binary encoding

Given N elements, enumerate all 2^N subsets.

- Count in binary from 0 to $2^N - 1$.
- Maintain array `a[]` where `a[i]` represents element `i`.
- If `1`, `a[i]` in subset; if 0, `a[i]` not in subset.

i	binary	subset
0	0 0 0 0	empty
1	0 0 0 1	0
2	0 0 1 0	1
3	0 0 1 1	1 0
4	0 1 0 0	2
5	0 1 0 1	2 0
6	0 1 1 0	2 1
7	0 1 1 1	2 1 0
8	1 0 0 0	3
9	1 0 0 1	3 0
10	1 0 1 0	3 1
11	1 0 1 1	3 1 0
12	1 1 0 0	3 2
13	1 1 0 1	3 2 0
14	1 1 1 0	3 2 1
15	1 1 1 1	3 2 1 0

Enumerating subsets: natural binary encoding

Given N elements, enumerate all 2^N subsets.

- Count in binary from 0 to $2^N - 1$.
- Maintain array `a[]` where `a[i]` represents element `i`.
- If `1`, `a[i]` in subset; if 0, `a[i]` not in subset.

Binary counter from warmup does the job.

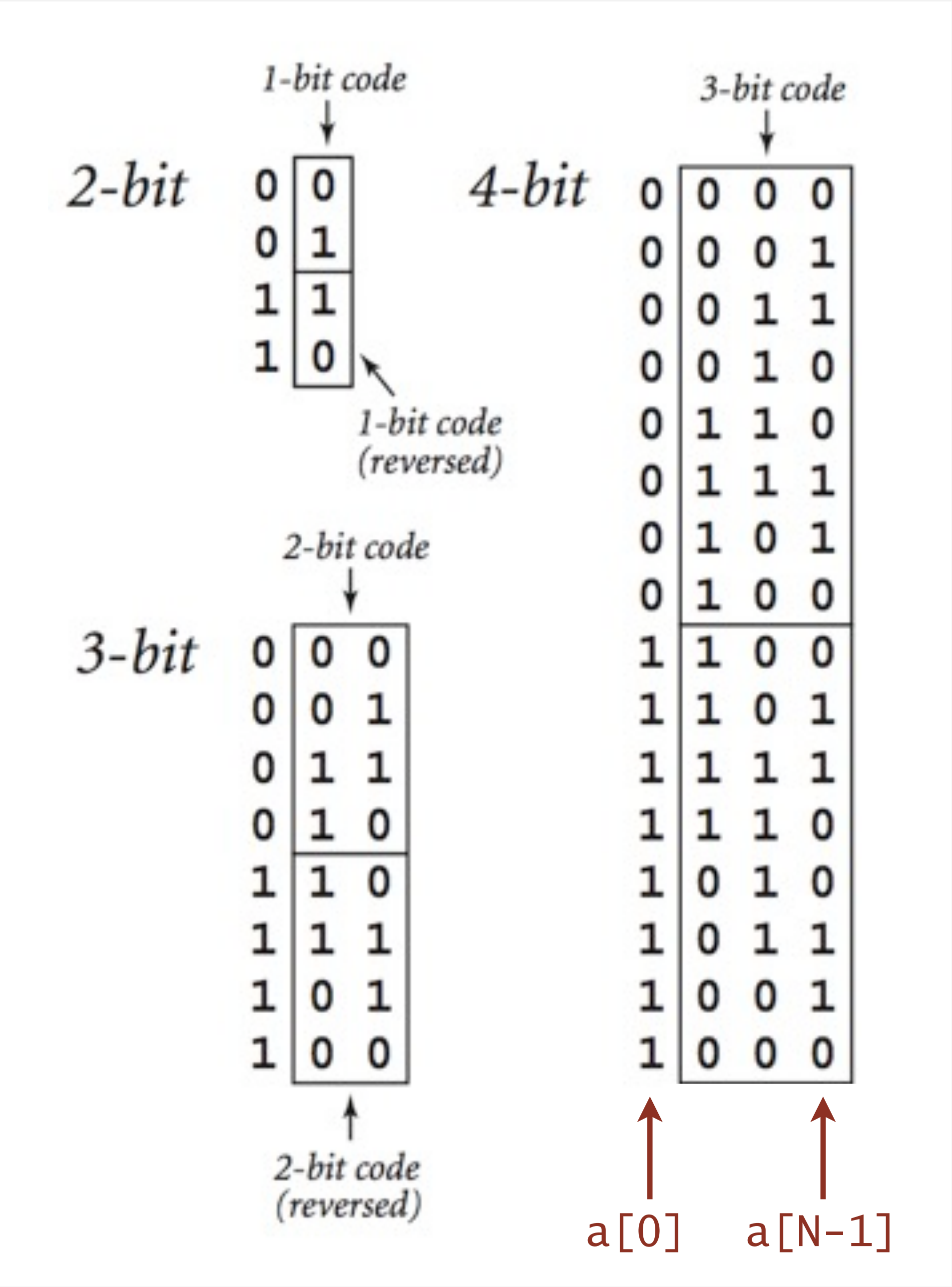
```
private void enumerate(int k)
{
  if (k == N)
  {  process(); return;  }
  enumerate(k+1);
  a[k] = 1;
  enumerate(k+1);
  a[k] = 0;
}
```

But multiple elements added / removed at once - can do better!

Binary reflected gray code

Def. The k-bit binary reflected Gray code is:

- The $(k-1)$ bit code, with a 0 prepended to each word, followed by:
- The $(k-1)$ bit code in reverse order, with a 1 prepended to each word.



Proposition. The Gray code enumerates all k-bit binary integers, while flipping only a single bit between adjacent codewords.

Pf. [By induction]

Enumerating subsets using Gray code

Two simple changes to binary counter from warmup:

- Flip `a[k]` instead of setting it to `1`.
- Eliminate cleanup.

Gray code binary counter

```
// all bit strings in a[k] to a[N-1]
private void enumerate(int k)
{
   if (k == N)
   {  process(); return;  }
   enumerate(k+1);
   a[k] = 1 - a[k];
   enumerate(k+1);
}
```

0	0	0
0	0	1
0	1	1
0	1	0
1	1	0
1	1	1
1	0	1
1	0	0

same values since no cleanup

standard binary counter (from warmup)

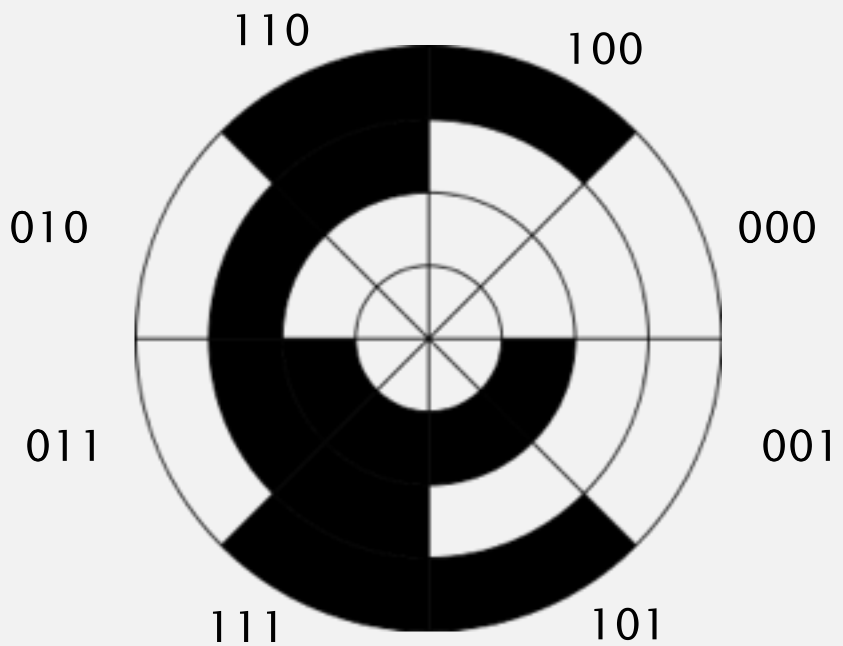
```
// all bit strings in a[k] to a[N-1]
private void enumerate(int k)
{
   if (k == N)
   {  process(); return;  }
   enumerate(k+1);
   a[k] = 1;
   enumerate(k+1);
   a[k] = 0;
}
```

0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

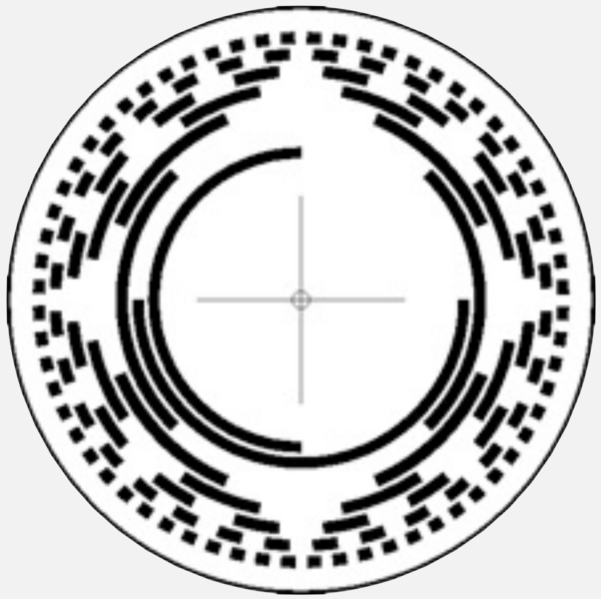
a[0]   a[N-1]

Advantage. Only one element in subset changes at a time.

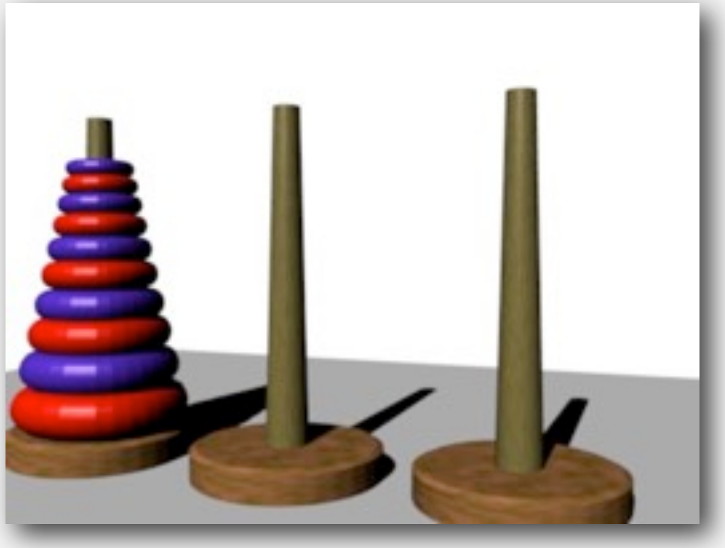
More applications of Gray codes



3-bit rotary encoder

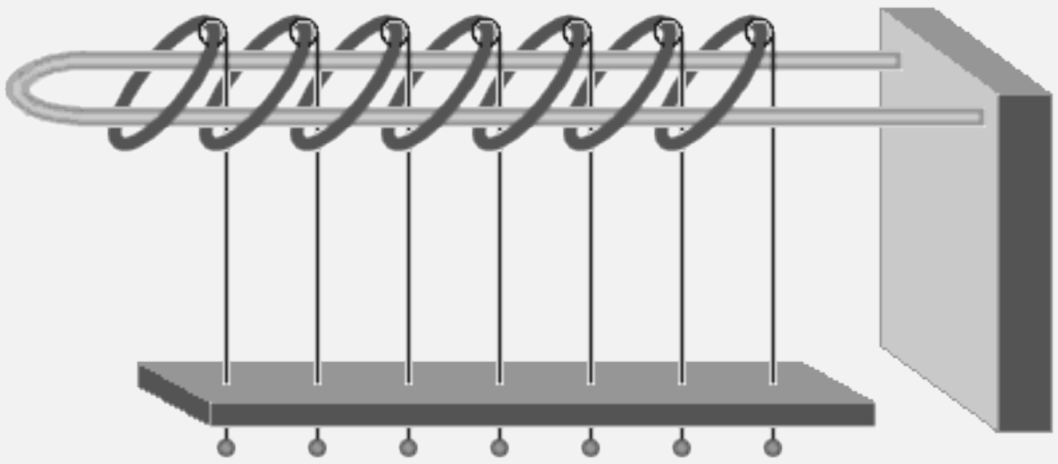


8-bit rotary encoder



Towers of Hanoi

(move ith smallest disk when bit i changes in Gray code)



Chinese ring puzzle (Baguenaudier)

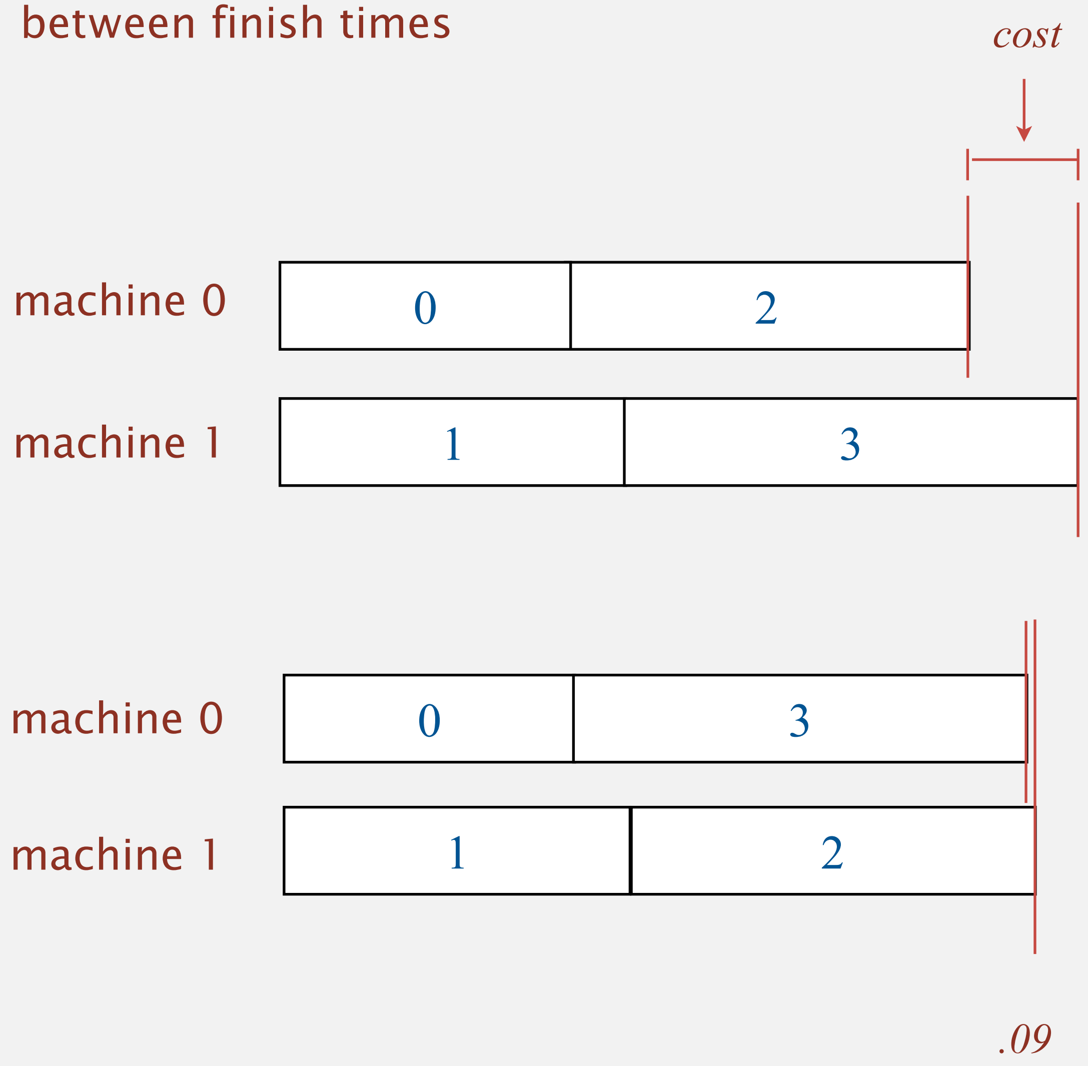
(move ith ring from right when bit i changes in Gray code)

Scheduling

Scheduling (set partitioning). Given N jobs of varying lengths, divide among two machines to minimize the time the last job finishes.

or, equivalently, difference between finish times

job	length
0	1.41
1	1.73
2	2.00
3	2.23



Remark. This scheduling problem is NP-complete.

Scheduling (Java implementation)

```
public class Scheduler
{
   private int N;           // Number of jobs.
   private int[] a;         // Subset assignments.
   private int[] b;         // Best assignment.
   private double[] jobs;   // Job lengths.

   public Scheduler(double[] jobs)
   {
      this.N = jobs.length;
      this.jobs = jobs;
      a = new int[N];
      b = new int[N];
      enumerate(N);
   }

   public int[] best()
   {  return b;  }

   private void enumerate(int k)
   {  /* Gray code enumeration. */   }

   private void process()
   {
     if (cost(a) < cost(b))
       for (int i = 0; i < N; i++)
         b[i] = a[i];
   }

   public static void main(String[] args)
   {  /* create Scheduler, print results */  }
}
```

a[]				finish times		cost
0	0	0	0	7.38	0.00	7.38
0	0	0	1	5.15	2.24	2.91
0	0	1	1	3.15	4.24	1.09
0	0	1	0	5.38	2.00	
0	1	1	0	3.65	3.73	0.08
0	1	1	1	1.41	5.97	
0	1	0	1	3.41	3.97	
0	1	0	0	5.65	1.73	
1	1	0	0	4.24	3.15	
1	1	0	1	2.00	5.38	
1	1	1	1	0.00	7.38	
1	1	1	0	2.24	5.15	
1	0	1	0	3.97	3.41	
1	0	1	1	1.73	5.65	
1	0	0	1	3.73	3.65	
1	0	0	0	5.97	1.41	

```
MACHINE 0     MACHINE 1
     1.41
                   1.73
                   2.00
     2.24
------------------------
     3.65          3.73
```

http://algs4.cs.princeton.edu

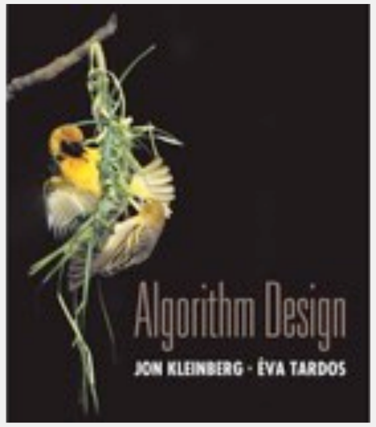
Combinatorial Search and Algorithm Design

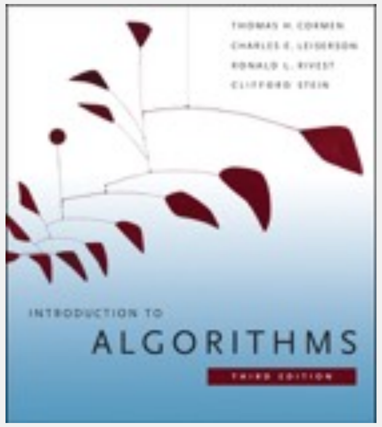
- introduction
- permutations
- backtracking
- subsets
- **algorithm design principles**
- interview questions

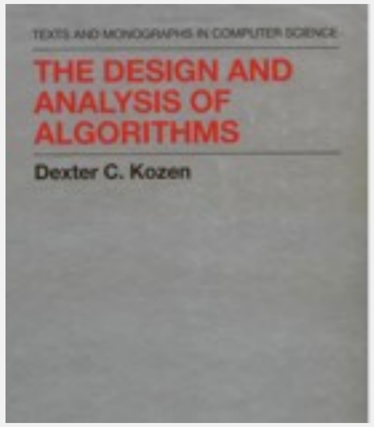
Algorithm design

Algorithm design patterns.

- Reduction to a previously solved problem. [Last time]
- Brute force + pruning. [Today]
- Greedy.
- Dynamic programming.
- Divide-and-conquer.
- Network flow.
- Randomized algorithms.







Want more? See COS 423.

Greedy algorithms

Make locally-optimal choices at each step.

Familiar examples.

- Huffman coding.
- Prim's algorithm.
- Kruskal's algorithm.
- Dijkstra's algorithm.

More classic examples.

- U.S. coin changing.
- Activity scheduling.
- Gale-Shapley stable marriage.
- ...

Caveat. Greedy algorithm only sometimes leads to optimal solution (but is often used anyway, especially for intractable problems).

Dynamic programming

"Eager" solution of overlapping subproblems.

- Build up solutions to larger and larger subproblems (caching solutions in a table for later reuse).
- Choose order so that partial results are available when you need them.

Familiar examples.

- Shortest paths in DAGs.
- Seam carving.
- Bellman-Ford.

THE THEORY OF DYNAMIC PROGRAMMING

RICHARD BELLMAN

More classic examples.

- Unix diff.
- Viterbi algorithm for hidden Markov models.
- Smith-Waterman for DNA sequence alignment.
- CKY algorithm for parsing context-free grammars.

...

Divide and conquer

Break up problem into independent subproblems.

- Solve each subproblem recursively.
- Combine solutions to subproblems to form solution to original problem.

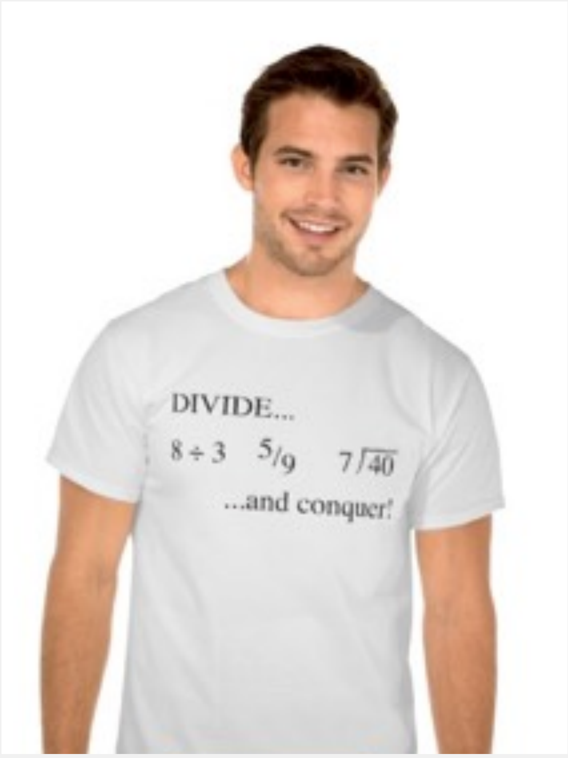
Familiar example.

- Mergesort.
- Quicksort.

More classic examples.

- Closest pair.
- Convolution and FFT.
- Matrix multiplication.
- Integer multiplication.

...



needs to take COS 226?

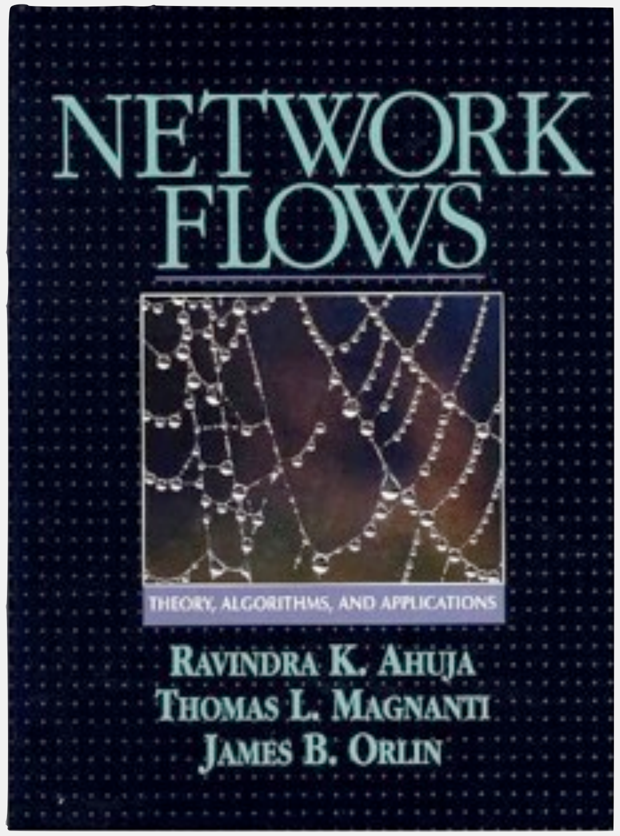
Prototypical usage. Turns brute-force N^2 algorithm into $N \log N$ algorithm.

Network flow

Classic problems on graphs with weights.

Familiar examples.

- Shortest paths.
- Bipartite matching.
- Maxflow and mincut.
- Minimum spanning tree.



Other classic examples.

- Nonbipartite matching.
- Min cost arborescence.
- Assignment problem.
- Min cost flow.
- ...

Applications. Many problems in science, engineering, and social sciences.

Randomized algorithms

Use random coin flips to guide behavior.

- Probabilistic guarantees of correctness or performance.

Familiar examples.

- Quicksort.
- Quickselect.
- Rabin-Karp substring search.

More classic examples.

- Miller-Rabin primality testing.
- Polynomial identity testing.
- Volume of convex body.
- Universal hashing.
- Global min cut.

...