Intractability

Today

- Intractability
- Search problems
- P vs. NP
- Classifying problems
- NP-completeness

Questions about computation

Q. What is a general-purpose computer?
Q. Are there limits on the power of digital computers?
Q. Are there limits on the power of machines we can build?

A simple model of computation: DFAs

Tape.
- Stores input.
- One arbitrarily long strip, divided into cells.
- Finite alphabet of symbols.

Tape head.
- Points to one cell of tape.
- Reads a symbol from active cell.
- Moves one cell at a time.

Q. Is there a more powerful model of computation?
A. Yes.
A universal model of computation: Turing machines

Tape.
- Stores input, output, and intermediate results.
- One arbitrarily long strip, divided into cells.
- Finite alphabet of symbols.

Tape head.
- Points to one cell of tape.
- Reads a symbol from active cell.
- Writes a symbol to active cell.
- Moves one cell at a time.

Q. Is there a more powerful model of computation?
A. No!

Church-Turing thesis (1936)

Turing machines can compute any function that can be computed by a physically harnessable process of the natural world.

Remark. "Thesis" and not a mathematical theorem because it’s a statement about the physical world and not subject to proof.

Use simulation to prove models equivalent.
- Android simulator on iPhone.
- iPhone simulator on Android.

Implications.
- No need to seek more powerful machines or languages.
- Enables rigorous study of computation (in this universe).

Bottom line. Turing machine is a simple and universal model of computation.

Church-Turing thesis: evidence

- 8 decades without a counterexample.
- Many, many models of computation that turned out to be equivalent.

<table>
<thead>
<tr>
<th>model of computation</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enhanced Turing machines</td>
<td>multiple heads, multiple tapes, 2D tape, nondeterminism</td>
</tr>
<tr>
<td>untyped lambda calculus</td>
<td>method to define and manipulate functions</td>
</tr>
<tr>
<td>recursive functions</td>
<td>functions dealing with computation on integers</td>
</tr>
<tr>
<td>unrestricted grammars</td>
<td>iterative string replacement rules used by linguists</td>
</tr>
<tr>
<td>extended L-systems</td>
<td>parallel string replacement rules that model plant growth</td>
</tr>
<tr>
<td>programming languages</td>
<td>Java, C, C++, Perl, Python, PHP, Lisp, PostScript, Excel</td>
</tr>
<tr>
<td>random access machines</td>
<td>registers plus main memory, e.g., TOY, Pentium</td>
</tr>
<tr>
<td>cellular automata</td>
<td>cells which change state based on local interactions</td>
</tr>
<tr>
<td>quantum computer</td>
<td>compute using superposition of quantum states</td>
</tr>
<tr>
<td>DNA computer</td>
<td>compute using biological operations on DNA</td>
</tr>
</tbody>
</table>

A question about algorithms

Q. Which algorithms are useful in practice?
- Measure running time as a function of input size $N$.
- Useful in practice ("efficient") = polynomial time for all inputs.

Ex 1. Sorting $N$ items takes $N \log N$ compares using mergesort.
Ex 2. Finding best TSP tour on $N$ points takes $N!$ steps using brute search.

Theory. Definition is broad and robust.
Practice. Poly-time algorithms scale to huge problems.
**Exponential growth**

Exponential growth dwarfs technological change.

- Suppose you have a giant parallel computing device...
- With as many processors as electrons in the universe...
- And each processor has power of today’s supercomputers...
- And each processor works for the life of the universe...

![Graph: $1000! \gg 10^{79} \times 10^{13} \times 10^{17}$](image)

- Will not help solve 1,000 city TSP problem via brute force.

**Questions about problems**

Q. Which problems can we solve in practice?
A. Those with poly-time algorithms.

Q. Which problems have poly-time algorithms?
A. Not so easy to know. Focus of today’s lecture.


**Bird’s-eye view**

**Def.** A problem is intractable if it can’t be solved in polynomial time.

**Desiderata.** Prove that a problem is intractable.

**Two problems that provably require exponential time.**

- Given a constant-size program, does it halt in at most $K$ steps?
- Given $N$-by-$N$ checkers board position, can the first player force a win?

**INTRACTABILITY**

- Search problems
- $P$ vs. $NP$  
- Classifying problems  
- $NP$-completeness
Four fundamental problems

**LSOLVE.** Given a system of linear equations, find a solution.

\[
\begin{align*}
0x_0 + 1x_1 + 1x_2 &= 4 \\
2x_0 + 4x_1 - 2x_2 &= 2 \\
0x_0 + 3x_1 + 15x_2 &= 36
\end{align*}
\]

\[\begin{array}{c}
x_0 = -1 \\
x_1 = 2 \\
x_2 = 2
\end{array}\]

variables are real numbers

**LP.** Given a system of linear inequalities, find a solution.

\[
\begin{align*}
48x_0 + 16x_1 + 19x_2 &\leq 88 \\
5x_0 + 4x_1 + 35x_2 &\geq 13 \\
15x_0 + 4x_1 + 20x_2 &\geq 23
\end{align*}
\]

\[\begin{array}{c}
x_0 = 1 \\
x_1 = 1 \\
x_2 = 0
\end{array}\]

variables are real numbers

**ILP.** Given a system of linear inequalities, find a 0-1 solution.

\[
\begin{align*}
x_0 + x_1 &\geq 1 \\
x_0 + x_2 &\geq 1 \\
x_0 + x_1 + x_2 &\leq 2
\end{align*}
\]

\[\begin{array}{c}
x_0 = 0 \\
x_1 = 1 \\
x_2 = 1
\end{array}\]

variables are 0 or 1

**SAT.** Given a system of boolean equations, find a binary solution.

\[
\begin{align*}
(x_0 \lor x_1) \land (x_0 \lor x_2) &= \text{true} \\
(x_1 \land x_2) \land (x_0 \lor x_1) &= \text{false} \\
(x_0 \land x_2) \land (x_0 \lor x_1) &= \text{false}
\end{align*}
\]

\[\begin{array}{c}
x_0 = \text{false} \\
x_1 = \text{false} \\
x_2 = \text{true}
\end{array}\]

variables are true or false

Search problems

**Search problem.** Given an instance \(I\) of a problem, find a solution \(S\).

**Requirement.** Must be able to efficiently check that \(S\) is a solution.

or report none exists

poly-time in size of instance \(I\)

LSOLVE. Given a system of linear equations, find a solution.

\[
\begin{align*}
0x_0 + 1x_1 + 1x_2 &= 4 \\
2x_0 + 4x_1 - 2x_2 &= 2 \\
0x_0 + 3x_1 + 15x_2 &= 36
\end{align*}
\]

\[\begin{array}{c}
x_0 = -1 \\
x_1 = 2 \\
x_2 = 2
\end{array}\]

To check solution \(S\), plug in values and verify each equation.
Search problems


Requirement. Must be able to efficiently check that $S$ is a solution.

LP. Given a system of linear inequalities, find a solution.

$$
\begin{align*}
48x_0 + 16x_1 + 119x_2 & \leq 88 \\
5x_0 + 4x_1 + 35x_2 & \geq 13 \\
15x_0 + 4x_1 + 20x_2 & \geq 23 \\
x_0, x_1, x_2 & \geq 0
\end{align*}
$$

instance $I$  \hspace{1cm} solution $S$

To check solution $S$, plug in values and verify each inequality.

ILP. Given a system of linear inequalities, find a binary solution.

$$
\begin{align*}
x_1 + x_2 & \geq 1 \\
x_0 + x_1 & \geq 1 \\
x_0 + x_1 + x_2 & \leq 2
\end{align*}
$$

instance $I$  \hspace{1cm} solution $S$

To check solution $S$, plug in values and verify each inequality.

SAT. Given a system of boolean equations, find a boolean solution.

$$(x'_1 \lor x'_2) \land (x_0 \lor x_2) = \text{true}$$

instance $I$  \hspace{1cm} solution $S$

To check solution $S$, plug in values and verify each equation.

FACTOR. Given an $n$-bit integer $x$, find a nontrivial factor.

$$\text{input size = number of bits}$$

instance $I$  \hspace{1cm} solution $S$

To check solution $S$, long divide 193707721 into 147573952589676412927.
Intractability

- Search problems
- P vs. NP
- Classifying problems
- NP-completeness

Def. P is the class of search problems solvable in poly-time.

<table>
<thead>
<tr>
<th>problem</th>
<th>description</th>
<th>poly-time algorithm</th>
<th>instance I</th>
<th>solution S</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSOLVE</td>
<td>Find a vector that satisfies (Ax = b)</td>
<td>Gaussian elimination</td>
<td>(x_0 + x_1 + x_2 = 4) (2x_0 + x_1 + x_2 = 2) (x_0 + x_1 + x_2 = 4)</td>
<td>(x_0 = 2) (x_1 = 2) (x_2 = 2)</td>
</tr>
<tr>
<td>LP</td>
<td>Find a vector that satisfies (Ax \leq b)</td>
<td>ellipsoid</td>
<td>(b_{ii}x_{j1} + b_{j2}x_{j2} &gt; b_{j3}) (b_{ii}x_{j1} + b_{j2}x_{j2} &gt; b_{j3}) (b_{ii}x_{j1} + b_{j2}x_{j2} &gt; b_{j3})</td>
<td>(x_i = 1) (x_i = 1) (x_i = 1)</td>
</tr>
<tr>
<td>SAT</td>
<td>Find a boolean vector (x) that satisfies (\Phi(x) = b)</td>
<td></td>
<td>(x_0 \lor x_1 \lor x_2 = \text{true}) (x_0 \lor x_1 \lor x_2 = \text{true}) (x_0 \lor x_1 \lor x_2 = \text{false})</td>
<td>(x_0 = 0) (x_1 = 0) (x_2 = 1)</td>
</tr>
<tr>
<td>FACTOR</td>
<td>Find a nontrivial factor of the integer (x)</td>
<td></td>
<td>(147573952589676412927193707721)</td>
<td>(191707721)</td>
</tr>
</tbody>
</table>

Significance. What scientists and engineers do compute feasibly.

NP

Def. NP is the class of all search problems.

<table>
<thead>
<tr>
<th>problem</th>
<th>description</th>
<th>poly-time algorithm</th>
<th>instance I</th>
<th>solution S</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSOLVE</td>
<td>Find a vector that satisfies (Ax = b)</td>
<td>Gaussian elimination</td>
<td>(x_0 + x_1 + x_2 = 4) (2x_0 + x_1 + x_2 = 2) (x_0 + x_1 + x_2 = 4)</td>
<td>(x_0 = 2) (x_1 = 2) (x_2 = 2)</td>
</tr>
<tr>
<td>LP</td>
<td>Find a vector that satisfies (Ax \leq b)</td>
<td>ellipsoid</td>
<td>(b_{ii}x_{j1} + b_{j2}x_{j2} &gt; b_{j3}) (b_{ii}x_{j1} + b_{j2}x_{j2} &gt; b_{j3}) (b_{ii}x_{j1} + b_{j2}x_{j2} &gt; b_{j3})</td>
<td>(x_i = 1) (x_i = 1) (x_i = 1)</td>
</tr>
<tr>
<td>SAT</td>
<td>Find a boolean vector (x) that satisfies (\Phi(x) = b)</td>
<td></td>
<td>(x_0 \lor x_1 \lor x_2 = \text{true}) (x_0 \lor x_1 \lor x_2 = \text{true}) (x_0 \lor x_1 \lor x_2 = \text{false})</td>
<td>(x_0 = 0) (x_1 = 0) (x_2 = 1)</td>
</tr>
<tr>
<td>FACTOR</td>
<td>Find a nontrivial factor of the integer (x)</td>
<td></td>
<td>(147573952589676412927193707721)</td>
<td>(191707721)</td>
</tr>
</tbody>
</table>

Significance. What scientists and engineers aspire to compute feasibly.

Nondeterminism

Nondeterministic machine can guess the desired solution.

Ex. \(\int()\ a = \text{new int[N]};\)
- Java initializes entries to 0.
- Nondeterministic machine initializes entries to the solution!

ILP. Given a system of linear inequalities, guess a 0-1 solution.

Ex. Turing machine.
- Deterministic: state, input determines next state.
- Nondeterministic: more than one possible next state.

NP. Search problems solvable in poly time on a nondeterministic TM.
Extended Church-Turing thesis

\[ P = \text{search problems solvable in poly-time in the natural world.} \]

Evidence supporting thesis. True for all physical computers.

Natural computers? No successful attempts (yet).

Ex. Computing Steiner trees with soap bubbles

STEINER: Find set of lines of minimal length connecting \( N \) given points

doesn't work for large \( N \)

Implication. To make future computers more efficient, suffices to focus on improving implementation of existing designs.

Automating creativity

**Q.** Being creative vs. appreciating creativity?

**Ex.** Mozart composes a piece of music; our neurons appreciate it.

**Ex.** Wiles proves a deep theorem; a colleague referees it.

**Ex.** Boeing designs an efficient airfoil; a simulator verifies it.

**Ex.** Einstein proposes a theory; an experimentalist validates it.

Computational analog. Does \( P = NP \)?

P vs. NP

Does \( P = NP \)?

\[ P = \text{search problems solvable in poly-time.} \]

\[ NP = \text{class of all search problems.} \]

Overwhelming consensus. \( P \neq NP \).

The central question

\[ P = NP \]

\[ P \neq NP \]

Two worlds.

If \( P = NP \)... Poly-time algorithms for SAT, ILP, TSP, FACTOR, ...

If \( P \neq NP \)... Would learn something fundamental about our universe.

Creative

ordinary

Does \( P = NP \)? [Can you always avoid brute-force searching and do better]
The central question

**P.** Class of search problems solvable in poly-time.
**NP.** Class of all search problems.

Does P = NP? [Can you always avoid brute-force searching and do better?]

Millennium prize. $1 million for resolution of P = NP problem.

A key problem: satisfiability

**SAT.** Given a system of boolean equations, find a solution.

\[
\begin{align*}
x_1' \lor x_2 \lor x_3 &= \text{true} \\
x_1 \lor x_2' \lor x_3 &= \text{true} \\
x_1 \lor x_2 \lor x_3' &= \text{true} \\
x_1' \lor x_2' \lor x_4 &= \text{true}
\end{align*}
\]

Key applications.
- Automatic verification systems for software.
- Electronic design automation (EDA) for hardware.
- Mean field diluted spin glass model in physics.
- ...

Exhaustive search

Q. How to solve an instance of SAT with \(n\) variables?
A. Exhaustive search: try all \(2^n\) truth assignments.

Q. Can we do anything substantially more clever?
Conjecture. No poly-time algorithm for SAT.

"intractable"
Classifying problems

Q. Which search problems are in P?
A. No easy answers (we don’t even know whether P = NP).

Problem X poly-time reduces to problem Y if X can be solved with:
- Polynomial number of standard computational steps.
- Polynomial number of calls to Y.

Consequence. If SAT poly-time reduces to Y, then we conclude that Y is (probably) intractable.

SAT poly-time reduces to ILP

SAT. Given a system of boolean equations, find a solution.

ILP. Given a system of linear inequalities, find a 0-1 solution.

More poly-time reductions from boolean satisfiability

Still more reductions from SAT

Conjecture. SAT is intractable.
Implication. All of these problems are intractable.

Aerospace engineering. Optimal mesh partitioning for finite elements.
Biology. Phylogeny reconstruction.
Chemical engineering. Heat exchanger network synthesis.
Chemistry. Protein folding.
Civil engineering. Equilibrium of urban traffic flows.
Economics. Computation of arbitrage in financial markets with friction.
Electrical engineering. VLSI layout.
Environmental engineering. Optimal placement of contaminant sensors.
Game theory. Nash equilibrium that maximizes social welfare.
Mathematics. Given integer coefficients, compute

\[ \sum_{i=0}^{n} (a_{i} \times x_{i}) \]

Mechanical engineering. Structure of turbulence in shear flows.
Medicine. Reconstructing 3D shape from biplane angiogram.
Operations research. Traveling salesperson problem.
Physics. Partition function of 3d Ising model.
Politics. Shapley-Shubik voting power.
Recreation. Versions of Sudoku, Checkers, Minesweeper, Tetris.
Statistics. Optimal experimental design.

plus over 6,000 scientific papers per year
**Intractability**

- Search problems
- P vs. NP
- Classifying problems
- NP-completeness

---

**NP-completeness**

*Definition.* An NP problem is **NP-complete** if every problem in NP poly-time reduce to it.


**Extremely brief proof sketch:**
- Convert non-deterministic TM notation to SAT notation.
- If you can solve SAT, you can solve any problem in NP.

*Corollary.* Poly-time algorithm for SAT iff P = NP.

---

**Implications of Cook-Levin theorem**

All of these problems (and many, many more) poly-time reduce to SAT.
Implications of Karp + Cook-Levin

All of these problems are NP-complete; they are manifestations of the same really hard problem.

Implications of NP-Completeness

Implication. [SAT captures difficulty of whole class NP]
- Poly-time algorithm for SAT iff $P = NP$.
- No poly-time algorithm for some NP problem $\Rightarrow$ none for SAT.

Remark. Can replace SAT with any of Karp’s problems.

Proving a problem NP-complete guides scientific inquiry.
- 1926: Ising introduces simple model for phase transitions.
- 1944: Onsager finds closed form solution to 2D version in tour de force.
- 19xx: Feynman and other top minds seek 3D solution.
- 2000: 3D-ISING proved NP-complete.

Two worlds (more detail)

Overwhelming consensus (still). $P \neq NP$.

Why we believe $P \neq NP$.

“*We admire Wiles’ proof of Fermat’s last theorem, the scientific theories of Newton, Einstein, Darwin, Watson and Crick, the design of the Golden Gate bridge and the Pyramids, precisely because they seem to require a leap which cannot be made by everyone, let alone a by simple mechanical device.*” — Avi Wigderson

Summary

$P$. Class of search problems solvable in poly-time.

$NP$. Class of all search problems, some of which seem wickedly hard.

NP-complete. Hardest problems in NP.

Intractable. Problem with no poly-time algorithm.

Many fundamental problems are NP-complete.
- SAT, ILP, HAMILTON-PATH, …
- 3D-ISING, …

Use theory a guide:
- A poly-time algorithm for an NP-complete problem would be a stunning breakthrough (a proof that $P = NP$).
- You will confront NP-complete problems in your career.
- Safe to assume that $P \neq NP$ and that such problems are intractable.
- Identify these situations and proceed accordingly.
Exploiting intractability

Modern cryptography.
- Ex. Send your credit card to Amazon.
- Ex. Digitally sign an e-document.
- Enables freedom of privacy, speech, press, political association.

RSA cryptosystem.
- To use: multiply two \( n \)-bit integers. \([\text{poly-time}]\)
- To break: factor a \( 2n \)-bit integer. \([\text{unlikely poly-time}]\)

\[
\begin{align*}
\text{Multiply} & = \text{EASY} \\
23 \times 67 & \rightarrow 1,541 \\
\text{Factor} & = \text{HARD}
\end{align*}
\]

Challenge. Factor this number.
Can’t do it? Create a company based on the difficulty of factoring.

RSA-704 \((\$30,000 \text{ prize if you can factor})\)

RSA algorithm

RSA sold for $2.1 billion
or design a t-shirt

Coping with intractability

Relax one of desired features.
- Solve arbitrary instances of the problem.
- Solve the problem to optimality.
- Solve the problem in poly-time.

Special cases may be tractable.
- Ex: Linear time algorithm for 2-SAT. \(\text{at most two variables per equation}\)
- Ex: Linear time algorithm for Horn-SAT. \(\text{at most one un-negated variable per equation}\)

Proposition. [Shor 1994] Can factor an \( n \)-bit integer in \( n^3 \) steps on a "quantum computer."

Q. Do we still believe the extended Church-Turing thesis???
Coping with intractability

Relax one of desired features.
- Solve arbitrary instances of the problem.
- Solve the problem to optimality.
- Solve the problem in poly-time.

Develop a heuristic, and hope it produces a good solution.
- No guarantees on quality of solution.
- Ex. TSP assignment heuristics.
- Ex. Metropolis algorithm, simulating annealing, genetic algorithms.

Approximation algorithm. Find solution of provably good quality.
- Ex. MAX-3SAT: provably satisfy 87.5% as many clauses as possible.
  but if you can guarantee to satisfy 87.51% as many clauses as possible in poly-time, then P = NP!

Caveat.

Combinatorial search

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

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

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

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

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?

![Image of rook placement problem]

$int\[a\[4\] = 6 \text{ means the rook from row 4 is in column 6.}$

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)

Initial permutation

N = 2

N = 3

N = 4

N = 5

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;
        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);
    }
}

4-rooks search tree

Solutions
N-rooks problem: back-of-envelope running time estimate

Slow way to compute $N!$.

Hypothesis. Running time is about $2 (N! / 8!)$ seconds.

<table>
<thead>
<tr>
<th>java Rooks 7</th>
<th>wc -l</th>
</tr>
</thead>
<tbody>
<tr>
<td>5040</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>java Rooks 8</th>
<th>wc -l</th>
</tr>
</thead>
<tbody>
<tr>
<td>40320</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>java Rooks 9</th>
<th>wc -l</th>
</tr>
</thead>
<tbody>
<tr>
<td>362880</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>java Rooks 10</th>
<th>wc -l</th>
</tr>
</thead>
<tbody>
<tr>
<td>3628800</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>java Rooks 25</th>
<th>wc -l</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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?

Represent. No two queens in the same row or column $\Rightarrow$ permutation.

Additional constraint. No diagonal attack is possible.

Challenge. Enumerate (or even count) the solutions.

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: effectiveness of backtracking

Pruning the search tree leads to enormous time savings.

<table>
<thead>
<tr>
<th>N</th>
<th>Q(N)</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>24</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>120</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>720</td>
</tr>
<tr>
<td>7</td>
<td>40</td>
<td>5,040</td>
</tr>
<tr>
<td>8</td>
<td>92</td>
<td>40,320</td>
</tr>
<tr>
<td>9</td>
<td>352</td>
<td>362,880</td>
</tr>
<tr>
<td>10</td>
<td>724</td>
<td>3,628,800</td>
</tr>
<tr>
<td>11</td>
<td>2,680</td>
<td>39,916,800</td>
</tr>
<tr>
<td>12</td>
<td>14,200</td>
<td>479,001,600</td>
</tr>
<tr>
<td>13</td>
<td>73,712</td>
<td>6,227,020,800</td>
</tr>
<tr>
<td>14</td>
<td>365,596</td>
<td>87,178,291,200</td>
</tr>
</tbody>
</table>

Hamilton path

**Goal.** Find a simple path that visits every vertex exactly once.

**Remark.** Euler path easy, but Hamilton path is NP-complete.
Hamilton path: backtracking solution

**Backtracking solution.** To find Hamilton path starting at $v$:
- Add $v$ to current path.
- For each vertex $w$ adjacent to $v$ - find a simple path starting at $w$ using all remaining vertices
- Clean up: remove $v$ from current path.

Q. How to implement?
A. Add cleanup to DFS (!)