

CSE 211 (Theory of Computation)

Introduction

Dr. Muhammad Masroor Ali

Professor

Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology

July 2018

Version: 0.81, Last modified: October 30, 2018

Automata, Computability, and Complexity

Sipser, 0.1, p-1

- Three traditionally central areas of the theory of computation:
 - automata,
 - computability, and
 - complexity.



- They are linked by the question:

What are the fundamental capabilities and limitations of computers?



Automata, Computability, and Complexity — *continued*

Sipser, 0.1, p-1

- This question goes back to the 1930s when mathematical logicians first began to explore the meaning of computation.
- Technological advances since that time have greatly increased our ability to compute.
- Have brought this question out of the realm of theory into the world of practical concern.



Automata, Computability, and Complexity — *continued*

Sipser, 0.1, p-1

- In each of the three areas — automata, computability, and complexity — this question is interpreted differently.
- The answers vary according to the interpretation.



Automata, Computability, and Complexity

Sipser, 0.1, p-2

- Computer problems come in different varieties.
- Some are easy, and some are hard.
- For example, the sorting problem is an easy one.
- Say that you need to arrange a list of numbers in ascending order.
- Even a small computer can sort a million numbers rather quickly.



Automata, Computability, and Complexity

Sipser, 0.1, p-2

- Compare that to a scheduling problem.
- Say that you must find a schedule of classes for the entire university to satisfy some reasonable constraints.
- No two classes are to take place in the same room at the same time.



Automata, Computability, and Complexity

Sipser, 0.1, p-2

- The scheduling problem seems to be much harder than the sorting problem.
- If you have just a thousand classes, finding the best schedule may require centuries, even with a supercomputer.



Automata, Computability, and Complexity

Sipser, 0.1, p-2

What makes some problems computationally hard and others easy?



Automata, Computability, and Complexity

Sipser, 0.1, p-2

- This is the central question of complexity theory.
- Remarkably, we don't know the answer to it, though it has been intensively researched for over 40 years.
- Later, we explore this fascinating question and some of its ramifications.



Automata, Computability, and Complexity

Sipser, 0.1, p-2

- In one important achievement of complexity theory thus far, researchers have discovered an elegant scheme for classifying problems according to their computational difficulty.
- It is analogous to the periodic table for classifying elements according to their chemical properties.
- Using this scheme, we can demonstrate a method for giving evidence that certain problems are computationally hard, even if we are unable to prove that they are.



P, NP, NP-Complete and NP-Hard

<https://stackoverflow.com/questions/1857244/>

what-are-the-differences-between-np-np-complete-and-np-hard

Decision Problem: A problem with a *yes* or *no* answer.



P, NP, NP-Complete and NP-Hard

<https://stackoverflow.com/questions/1857244/>

what-are-the-differences-between-np-np-complete-and-np-hard

P

- P is a complexity class that represents the set of all decision problems that can be solved in polynomial time.
- That is, given an instance of the problem, the answer yes or no can be decided in polynomial time.



P, NP, NP-Complete and NP-Hard

<https://stackoverflow.com/questions/1857244/>

what-are-the-differences-between-np-np-complete-and-np-hard

P

- Given a connected graph G , can its vertices be coloured using two colours so that no edge is monochromatic?



P, NP, NP-Complete and NP-Hard

<https://stackoverflow.com/questions/1857244/>

what-are-the-differences-between-np-np-complete-and-np-hard

P

- Start with an arbitrary vertex.
- Color it red and all of its neighbours blue and continue.
- Stop when you run out of vertices or you are forced to make an edge have both of its endpoints be the same color.



P, NP, NP-Complete and NP-Hard — *continued*

<https://stackoverflow.com/questions/1857244/>

what-are-the-differences-between-np-np-complete-and-np-hard

NP

- NP is a complexity class that represents the set of all decision problems for which the instances where the answer is “yes” have proofs that can be verified in polynomial time.
- This means that if someone gives us an instance of the problem and a certificate (sometimes called a witness) to the answer being yes, we can check that it is correct in polynomial time.



P, NP, NP-Complete and NP-Hard — *continued*

<https://stackoverflow.com/questions/1857244/>

what-are-the-differences-between-np-np-complete-and-np-hard

NP

- Integer factorisation is in NP.
- This is the problem that given integers n and m , is there an integer f with $1 < f < m$, such that f divides n (f is a small factor of n)?



P, NP, NP-Complete and NP-Hard — *continued*

<https://stackoverflow.com/questions/1857244/>

what-are-the-differences-between-np-np-complete-and-np-hard

NP

- This is a decision problem because the answers are *yes* or *no*.



P, NP, NP-Complete and NP-Hard — *continued*

<https://stackoverflow.com/questions/1857244/>

what-are-the-differences-between-np-np-complete-and-np-hard

NP

- Someone hands us an instance of the problem (so they hand us integers n and m) and an integer f with $1 < f < m$.
- Claim that f is a factor of n (the certificate).
- We can check the answer in polynomial time by performing the division n/f .



P, NP, NP-Complete and NP-Hard — *continued*

<https://stackoverflow.com/questions/1857244/>

what-are-the-differences-between-np-np-complete-and-np-hard

NP-Complete

- NP-Complete is a complexity class which represents the set of all problems X in NP for which it is possible to reduce any other NP problem Y to X in polynomial time.



P, NP, NP-Complete and NP-Hard — *continued*

<https://stackoverflow.com/questions/1857244/>

what-are-the-differences-between-np-np-complete-and-np-hard

NP-Complete

- Intuitively this means that we can solve Y quickly if we know how to solve X quickly.



P, NP, NP-Complete and NP-Hard — *continued*

<https://stackoverflow.com/questions/1857244/>

what-are-the-differences-between-np-np-complete-and-np-hard

NP-Complete

- Precisely, Y is reducible to X .
- There is a polynomial time algorithm f to transform instances y of Y to instances $x = f(y)$ of X in polynomial time.
- With the property that the answer to y is yes, if and only if the answer to $f(y)$ is yes.



P, NP, NP-Complete and NP-Hard — *continued*

<https://stackoverflow.com/questions/1857244/>

what-are-the-differences-between-np-np-complete-and-np-hard

NP-Complete

- 3-SAT.
- We are given a conjunction (ANDs) of 3-clause disjunctions (ORs), statements of the form,

$$(x_{v11} \vee x_{v21} \vee x_{v31}) \wedge$$

$$(x_{v12} \vee x_{v22} \vee x_{v32}) \wedge$$

$$\dots \wedge$$

$$(x_{v1n} \vee x_{v2n} \vee x_{v3n})$$

- Each x_{vij} is a boolean variable or the negation of a variable from a finite predefined list (x_1, x_2, \dots, x_n) .



P, NP, NP-Complete and NP-Hard — *continued*

<https://stackoverflow.com/questions/1857244/>

what-are-the-differences-between-np-np-complete-and-np-hard

NP-Complete

- It can be shown that every NP problem can be reduced to 3-SAT.
- The proof of this requires use of the technical definition of NP (based on non-deterministic Turing machines).
- This is known as Cook's theorem.



P, NP, NP-Complete and NP-Hard — *continued*

<https://stackoverflow.com/questions/1857244/>

what-are-the-differences-between-np-np-complete-and-np-hard

NP-Complete

- What makes NP-complete problems important is that if a deterministic polynomial time algorithm can be found to solve one of them, every NP problem is solvable in polynomial time.
- One problem to rule them all.



Graph 3-colorability is NP-Complete

<https://www.ugrad.cs.ubc.ca/~cs320/2013W2/handouts/3color.html>

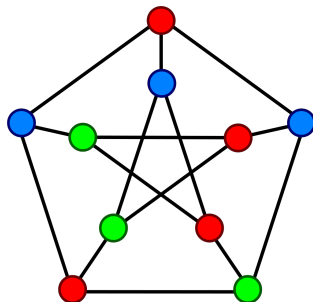
- Problem of determining whether or not a graph is 3-colorable.
- Nodes are can be colored using at most 3 colors, so that no two adjacent colors are identical.



Graph 3-colorability is NP-Complete — *continued*

<https://www.ugrad.cs.ubc.ca/~cs320/2013W2/handouts/3color.html>

- It belongs to NP.
- We can not find a solution is polynomial time.
- But if we have a solution, we can then verify that by simply looking at each edge.



Graph 3-colorability is NP-Complete — *continued*

<https://www.ugrad.cs.ubc.ca/~cs320/2013W2/handouts/3color.html>

- We need to show that, given an instance of 3-satisfiability, we can construct an *equivalent* instance of the Graph 3-colorability problem.



Graph 3-colorability is NP-Complete — *continued*

<https://www.ugrad.cs.ubc.ca/~cs320/2013W2/handouts/3color.html>

- By equivalent, we mean that:
 - We can assign true/false values to every variable in the instance of satisfiability, so that every clause evaluates to true,
- if and only if,
 - We can color the vertices of the graph using 3 colors, such that no edge has both endpoints of the same color.



Graph 3-colorability is NP-Complete — *continued*

<https://www.ugrad.cs.ubc.ca/~cs320/2013W2/handouts/3color.html>

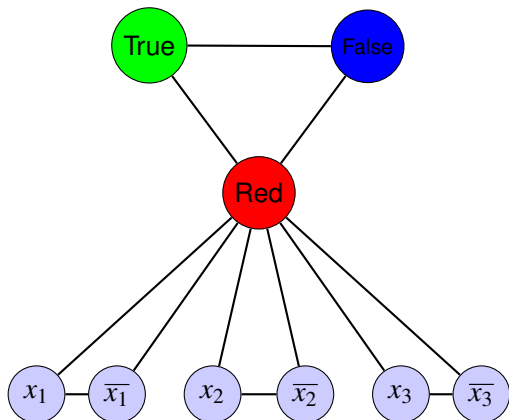
- This allows us to solve the instance of Satisfiability by constructing the graph.
- And then checking whether or not we can color it.
- If so, the instance of Satisfiability was satisfiable.
- If not, it wasn't.



Graph 3-colorability is NP-Complete — *continued*

<https://www.ugrad.cs.ubc.ca/~cs320/2013W2/handouts/3color.html>

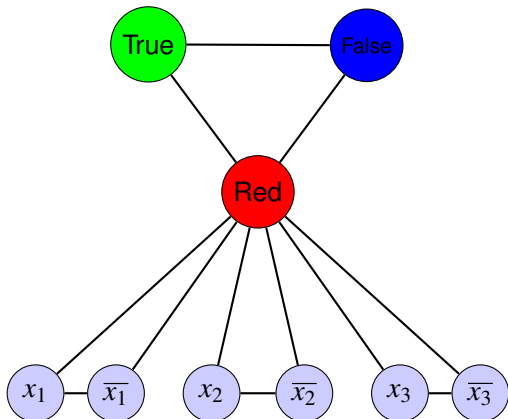
- We create a triangle.
- The vertices are called True, False and Red.
- These names will also be the names we will give to the color that gets used for these vertices when the graph is 3-colored.



Graph 3-colorability is NP-Complete — *continued*

<https://www.ugrad.cs.ubc.ca/~cs320/2013W2/handouts/3color.html>

- For every variable x_i that appears in the instance of Satisfiability, we connect a vertex x_i to a vertex \bar{x}_i .
- We connect both x_i and \bar{x}_i to red.

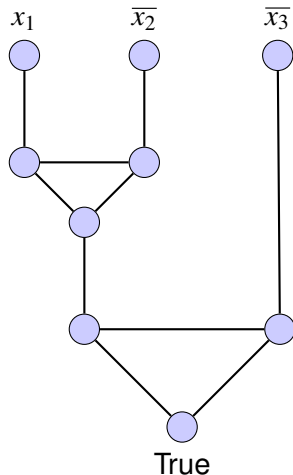


Graph 3-colorability is NP-Complete — *continued*

<https://www.ugrad.cs.ubc.ca/~cs320/2013W2/handouts/3color.html>

$$x_1 \vee \overline{x_2} \vee \overline{x_3}$$

- Graph for $x_1 \vee \overline{x_2} \vee \overline{x_3}$.

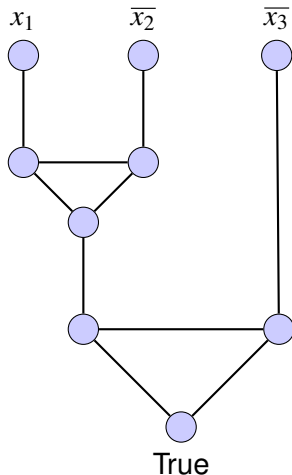


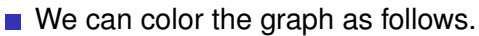
Graph 3-colorability is NP-Complete — *continued*

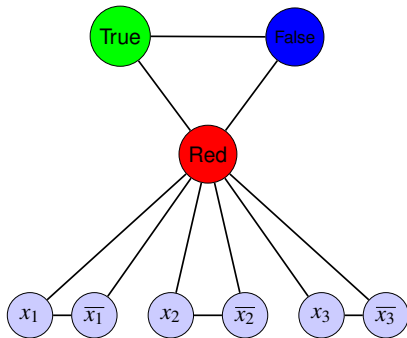
<https://www.ugrad.cs.ubc.ca/~cs320/2013W2/handouts/3color.html>

- The only ways to color it with the colors True, False and Red, without having an edge whose two endpoints have the same color, is to use the color True for at least one of the vertices at the top.
- Conversely, if we assign colors True and False to the three vertices at the top, then as long as at least one of them was colored True, then we will be able to color the rest of the graph.

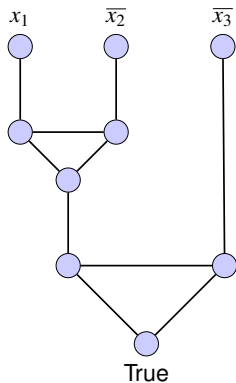
$$x_1 \vee \overline{x_2} \vee \overline{x_3}$$



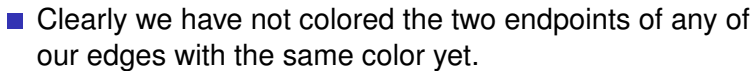
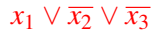


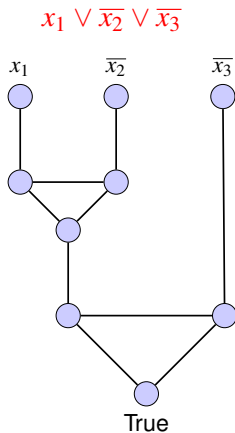
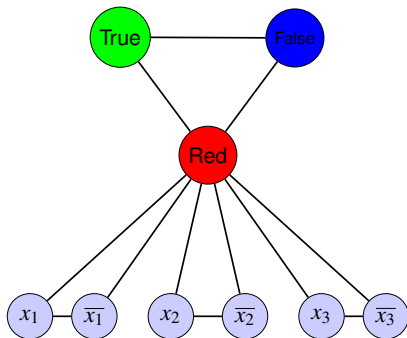


$$x_1 \vee \overline{x_2} \vee \overline{x_3}$$

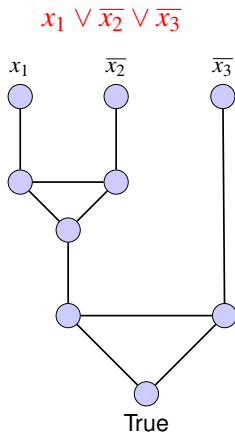
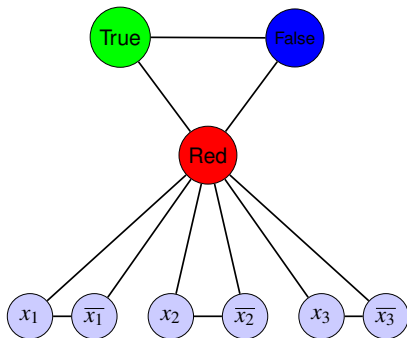


- If variable x_i was assigned the value true, then we color vertex x_i true, and vertex $\overline{x_i}$ false.
- If variable x_i was assigned the value false, then we color vertex $\overline{x_i}$ false, and vertex x_i true.

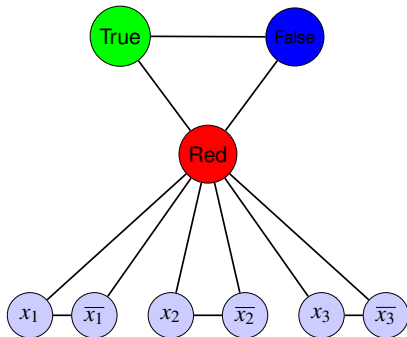




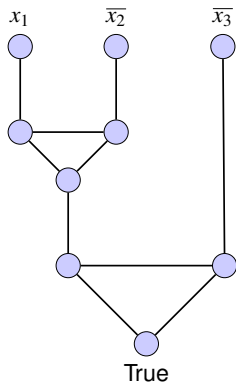
■ Hence we can color the rest of these using the three colors.



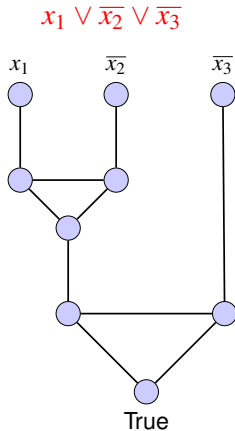
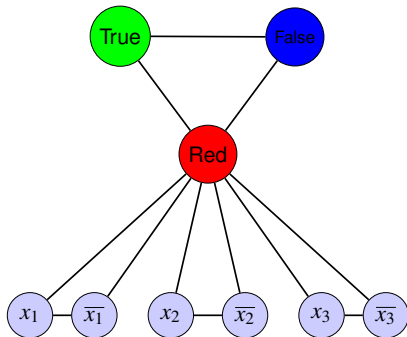
■ Now suppose that the graph can be colored using 3 colors.



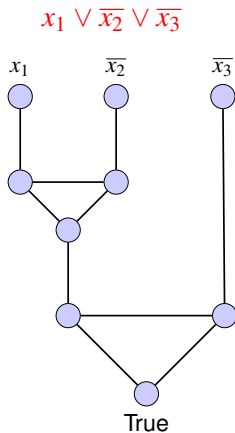
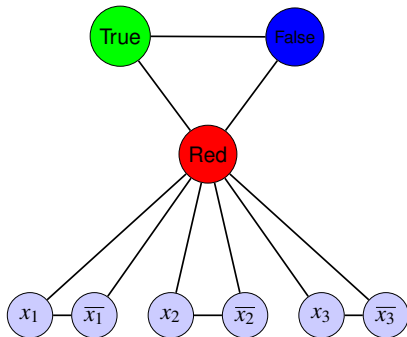
$$x_1 \vee \overline{x_2} \vee \overline{x_3}$$



- We assign the value true to a variable x_i if the vertex x_i was colored true.
- And the value false if the vertex x_i was colored false.
- This vertex can not be colored red because it is adjacent to the red vertex.



- Every clause C must be true.
- The piece of the graph used for C can only be colored with 3 colors if one or more of the vertices at the top is colored true.



■ This means that the instance of Satisfiability is satisfiable.

Graph 3-colorability is NP-Complete — *continued*

<https://www.ugrad.cs.ubc.ca/~cs320/2013W2/handouts/3color.html>

- Graph 3-colorability is NP-Complete.



P, NP, NP-Complete and NP-Hard — *continued*

<https://stackoverflow.com/questions/1857244/>

what-are-the-differences-between-np-np-complete-and-np-hard

NP-Hard

- Intuitively, these are the problems that are at least as hard as the NP-complete problems.
- Note that NP-hard problems do not have to be in NP.
- And they do not have to be decision problems.



P, NP, NP-Complete and NP-Hard — *continued*

<https://stackoverflow.com/questions/1857244/>

what-are-the-differences-between-np-np-complete-and-np-hard

NP-Hard

- The precise definition here is that a problem X is NP-hard, if there is an NP-complete problem Y , such that Y is reducible to X in polynomial time.



P, NP, NP-Complete and NP-Hard — *continued*

<https://stackoverflow.com/questions/1857244/>

what-are-the-differences-between-np-np-complete-and-np-hard

NP-Hard

- But since any NP-complete problem can be reduced to any other NP-complete problem in polynomial time, all NP-complete problems can be reduced to any NP-hard problem in polynomial time.
- Then, if there is a solution to one NP-hard problem in polynomial time, there is a solution to all NP problems in polynomial time.



P, NP, NP-Complete and NP-Hard — *continued*

<https://stackoverflow.com/questions/1857244/>

what-are-the-differences-between-np-np-complete-and-np-hard

NP-Hard

- The halting problem is an NP-hard problem.
- This is the problem that given a program P and input I , will it halt?
- This is a decision problem but it is not in NP.



P, NP, NP-Complete and NP-Hard — *continued*

<https://stackoverflow.com/questions/1857244/>

what-are-the-differences-between-np-np-complete-and-np-hard

NP-Hard

- It is clear that any NP-complete problem can be reduced to this one.
- As another example, any NP-complete problem is NP-hard.



Traveling Salesman Problem is NP-Hard

<https://cs.stackexchange.com/a/2836/7200> and https://stanford.edu/~rezab/classes/cme305/W15/Midterm/pmidtermII_soln.pdf

- Deciding whether a graph has a Hamiltonian cycle is NP-Complete.



Traveling Salesman Problem is NP-Hard — *continued*

<https://stanford.edu/~rezab/classes/cme305/W15/Midterm/pmidtermIIIsoln.pdf>

- We will use the decision version of TSP:

Travelling Salesman Problem (Decision Version): Given a weighted graph G and a target cost C , is there a Hamiltonian cycle in G whose weight is at most C ?



Traveling Salesman Problem is NP-Hard — *continued*

<https://stanford.edu/~rezab/classes/cme305/W15/Midterm/pmidtermIIIsoln.pdf>

- We will use the decision version of TSP:

Travelling Salesman Problem (Decision Version): Given a weighted graph G and a target cost C , is there a Hamiltonian cycle in G whose weight is at most C ?

- For a “yes” instance, the certificate is just some Hamiltonian cycle whose weight is at most C .



Traveling Salesman Problem is NP-Hard — *continued*

<https://stanford.edu/~rezab/classes/cme305/W15/Midterm/pmidtermIIIsoln.pdf>

- We reduce Hamiltonian cycle HC to TSP.
- Show $HC \leq_P TSP$.



Traveling Salesman Problem is NP-Hard — *continued*

<https://stanford.edu/~rezab/classes/cme305/W15/Midterm/pmidtermIIIsoln.pdf>

- You could find the cost of a minimum tour by binary search, starting with the weight of the entire network as an upper bound.
- We can do this in polynomial time.



Traveling Salesman Problem is NP-Hard — *continued*

<https://stanford.edu/~rezab/classes/cme305/W15/Midterm/pmidtermIIIsoln.pdf>

- Given that Hamiltonian Cycle is NP-Complete, the decision version of TSP is NP-Complete, hence TSP is NP-Hard.



NP-completeness

<https://en.wikipedia.org/wiki/NP-completeness>

- In computational complexity theory, an NP-complete decision problem is one which is in the NP complexity class and which is also NP-hard.
- In this context, NP stands for “nondeterministic polynomial time”.
- The set of NP-complete problems is often denoted by NP-C or NPC.



NP-completeness — *continued*

<https://en.wikipedia.org/wiki/NP-completeness>

- Any given solution to an NP-complete problem can be verified quickly (in polynomial time).
- But there is no known efficient way to locate a solution in the first place.
- The most notable characteristic of NP-complete problems is that no fast solution to them is known.
- That is, the time required to solve the problem using any currently known algorithm increases very quickly as the size of the problem grows.



NP-completeness — *continued*

<https://en.wikipedia.org/wiki/NP-completeness>

- As a consequence, determining whether it is possible to solve these problems quickly, called the P versus NP problem, is one of the principal unsolved problems in computer science today.



NP-completeness — *continued*

<https://en.wikipedia.org/wiki/NP-completeness>

- Some well-known problems that are NP-complete when expressed as decision problems:
 - Boolean satisfiability problem (SAT)
 - Knapsack problem
 - Hamiltonian path problem
 - Travelling salesman problem (decision version)
 - Subgraph isomorphism problem
 - Subset sum problem
 - Clique problem
 - Vertex cover problem
 - Independent set problem
 - Dominating set problem
 - Graph coloring problem



Boolean Satisfiability Problem

https://en.wikipedia.org/wiki/Boolean_satisfiability_problem

- SATISFIABILITY or SAT.
- Is the problem of determining if there exists an interpretation that satisfies a given Boolean formula.



Boolean Satisfiability Problem — *continued*

https://en.wikipedia.org/wiki/Boolean_satisfiability_problem

- It asks whether the variables of a given Boolean formula can be consistently replaced by the values TRUE or FALSE in such a way that the formula evaluates to TRUE.
- If this is the case, the formula is called satisfiable.
- On the other hand, if no such assignment exists, the function expressed by the formula is FALSE for all possible variable assignments and the formula is unsatisfiable.



Boolean Satisfiability Problem — *continued*

https://en.wikipedia.org/wiki/Boolean_satisfiability_problem

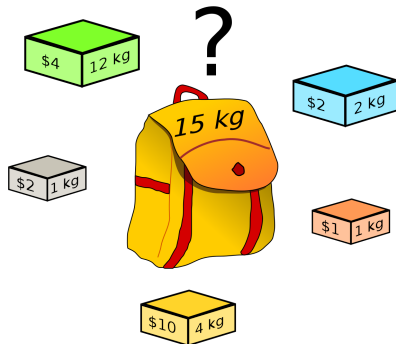
- The formula $a \wedge \neg b$ is satisfiable because one can find the values $a = \text{TRUE}$ and $b = \text{FALSE}$, which make $(a \wedge \neg b) = \text{TRUE}$.
- In contrast, $a \wedge \neg a$ is unsatisfiable.



Knapsack Problem

https://en.wikipedia.org/wiki/Knapsack_problem

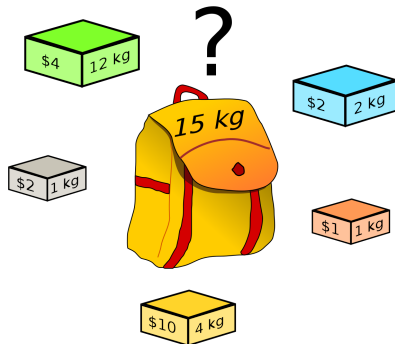
- The knapsack problem or rucksack problem is a problem in combinatorial optimization.
- Given a set of items, each with a weight and a value.
- Determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.



Knapsack Problem

https://en.wikipedia.org/wiki/Knapsack_problem

- It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most valuable items.



Knapsack Problem — *continued*

https://en.wikipedia.org/wiki/Knapsack_problem

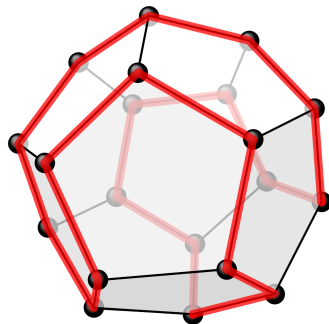
- The decision problem form of the knapsack problem
Can a value of at least V be achieved without exceeding the weight W ?
is NP-complete.
- Thus there is no known algorithm both correct and fast (polynomial-time) in all cases.



Hamiltonian Path Problem

https://en.wikipedia.org/wiki/Hamiltonian_path_problem

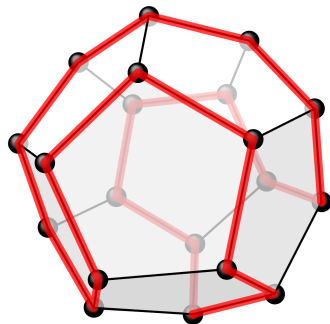
Hamiltonian path: a path in an undirected or directed graph that visits each vertex exactly once.



Hamiltonian Path Problem

https://en.wikipedia.org/wiki/Hamiltonian_path_problem

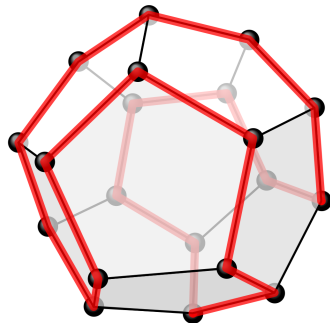
- Hamiltonian path problem and the Hamiltonian cycle problem are problems of determining whether a Hamiltonian path or a Hamiltonian cycle exists in a given graph (whether directed or undirected).
- Both problems are NP-complete.



Hamiltonian Path Problem

https://en.wikipedia.org/wiki/Hamiltonian_path_problem

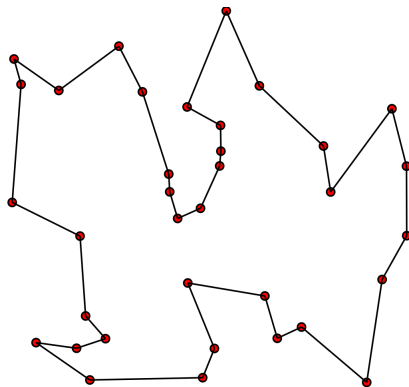
- There are $n!$ different sequences of vertices that might be Hamiltonian paths in a given n -vertex graph.



Travelling Salesman Problem

https://en.wikipedia.org/wiki/Travelling_salesman_problem

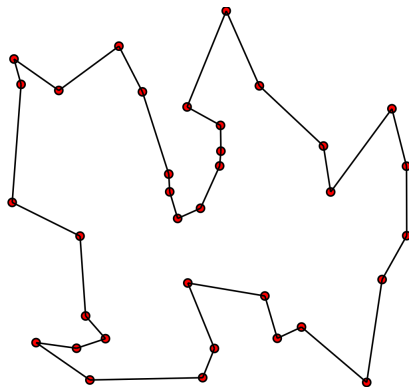
- Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city?
- It is an NP-hard problem in combinatorial optimization, important in operations research and theoretical computer science.



Travelling Salesman Problem

https://en.wikipedia.org/wiki/Travelling_salesman_problem

- The decision version of the TSP, given a length L , the task is to decide whether the graph has any tour shorter than L , belongs to the class of NP-complete problems.



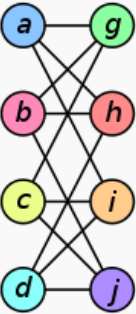
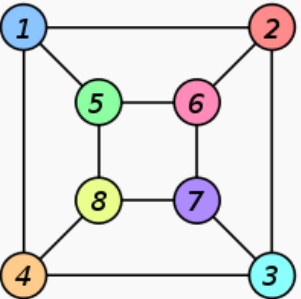
Subgraph Isomorphism Problem

https://en.wikipedia.org/wiki/Subgraph_isomorphism_problem

- Two graphs G and H are given as input.
- One must determine whether G contains a subgraph that is isomorphic to H .



Subgraph Isomorphism Problem — *continued*

| Graph G | Graph H | An isomorphism between G and H |
|---|---|---|
|  |  | $\begin{aligned}f(a) &= 1 \\f(b) &= 6 \\f(c) &= 8 \\f(d) &= 3 \\f(g) &= 5 \\f(h) &= 2 \\f(i) &= 4 \\f(j) &= 7\end{aligned}$ |



Subgraph Isomorphism Problem

https://en.wikipedia.org/wiki/Subgraph_isomorphism_problem

- Subgraph isomorphism is a generalization of both the maximum clique problem and the problem of testing whether a graph contains a Hamiltonian cycle, and is therefore NP-complete.



Subset Sum Problem

https://en.wikipedia.org/wiki/Subset_sum_problem

- Given a set (or multiset) of integers, is there a non-empty subset whose sum is zero?
- Given the set $\{-7, -3, -2, 5, 8\}$, the answer is yes because the subset $\{-3, -2, 5\}$ sums to zero.



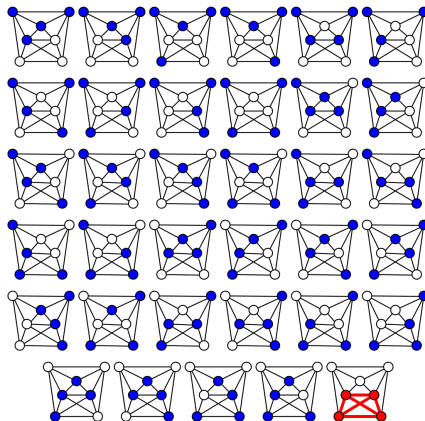
Clique Problem

https://en.wikipedia.org/wiki/Clique_problem

- Finding cliques, subsets of vertices, all adjacent to each other, also called complete subgraphs, in a graph.



Clique Problem — *continued*



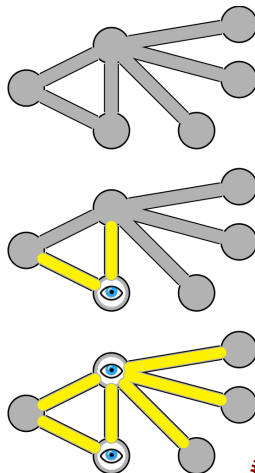
- The brute force algorithm finds a 4-clique in this 7-vertex graph by systematically checking all $C(7, 4) = 35$ 4-vertex subgraphs.



Vertex Cover Problem

https://en.wikipedia.org/wiki/Vertex_cover

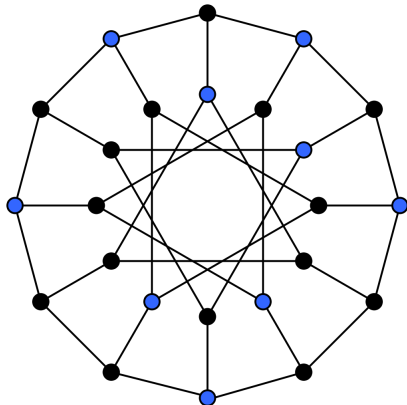
- Vertex cover (sometimes node cover) of a graph is a set of vertices such that each edge of the graph is incident to at least one vertex of the set.
- The problem of finding a minimum vertex cover is a classical optimization problem in computer science.
- It is a typical example of an NP-hard optimization problem that has an approximation algorithm.



Independent Set (Graph Theory)

[https://en.wikipedia.org/wiki/Independent_set_\(graph_theory\)](https://en.wikipedia.org/wiki/Independent_set_(graph_theory))

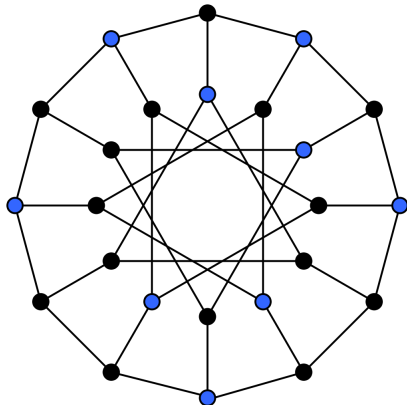
- Independent set or stable set is a set of vertices in a graph, no two of which are adjacent.
- That is, it is a set S of vertices such that for every two vertices in S , there is no edge connecting the two.
- Equivalently, each edge in the graph has at most one endpoint in S .



Independent Set (Graph Theory)

[https://en.wikipedia.org/wiki/Independent_set_\(graph_theory\)](https://en.wikipedia.org/wiki/Independent_set_(graph_theory))

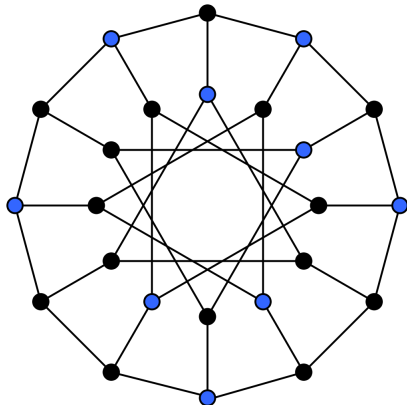
- The size of an independent set is the number of vertices it contains.
- Independent sets have also been called internally stable sets.



Independent Set (Graph Theory)

[https://en.wikipedia.org/wiki/Independent_set_\(graph_theory\)](https://en.wikipedia.org/wiki/Independent_set_(graph_theory))

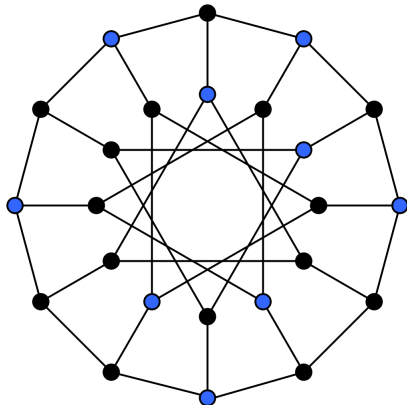
- A maximum independent set is an independent set of largest possible size for a given graph G .
- This size is called the independence number of G , and denoted $\alpha(G)$.



Independent Set (Graph Theory)

[https://en.wikipedia.org/wiki/Independent_set_\(graph_theory\)](https://en.wikipedia.org/wiki/Independent_set_(graph_theory))

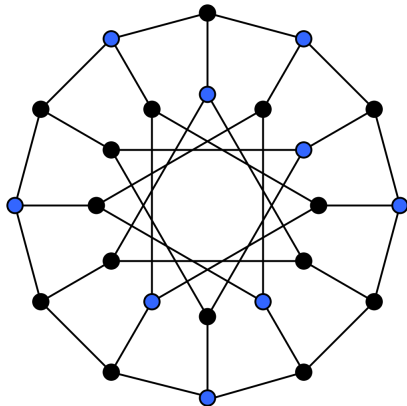
- The problem of finding such a set is called the maximum independent set problem and is an NP-hard optimization problem.



Independent Set (Graph Theory)

[https://en.wikipedia.org/wiki/Independent_set_\(graph_theory\)](https://en.wikipedia.org/wiki/Independent_set_(graph_theory))

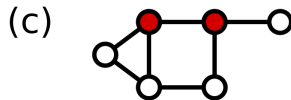
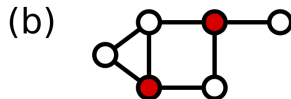
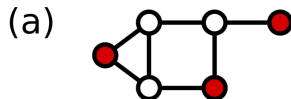
- The nine blue vertices form a maximum independent set.



Dominating Set

https://en.wikipedia.org/wiki/Dominating_set

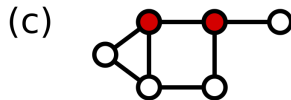
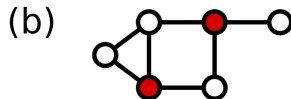
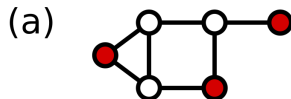
- A dominating set for a graph $G = (V, E)$ is a subset D of V such that every vertex not in D is adjacent to at least one member of D .
- The domination number $\gamma(G)$ is the number of vertices in a smallest dominating set for G .



Dominating Set

https://en.wikipedia.org/wiki/Dominating_set

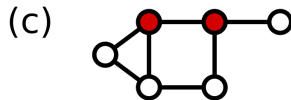
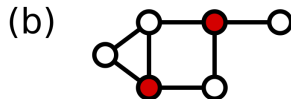
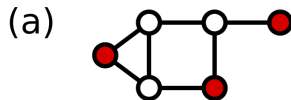
- The dominating set problem concerns testing whether $\gamma(G) \leq K$ for a given graph G and input K .
- It is a classical NP-complete decision problem.



Dominating Set

https://en.wikipedia.org/wiki/Dominating_set

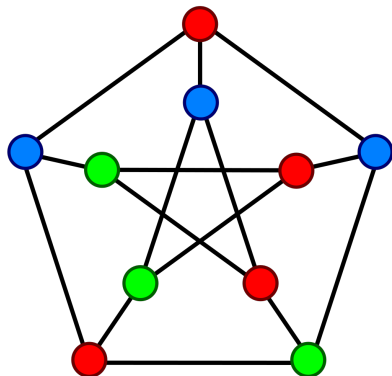
- Dominating sets are indicated by the red vertices.



Graph Coloring

https://en.wikipedia.org/wiki/Graph_coloring

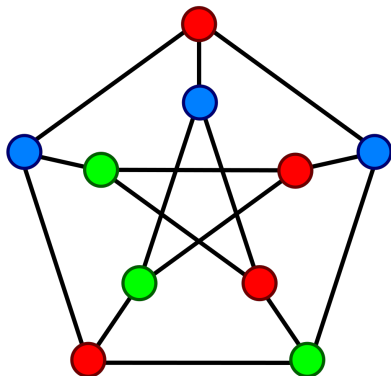
- Graph coloring is a special case of graph labeling.
- It is an assignment of labels traditionally called *colors* to elements of a graph subject to certain constraints.



Graph Coloring

https://en.wikipedia.org/wiki/Graph_coloring

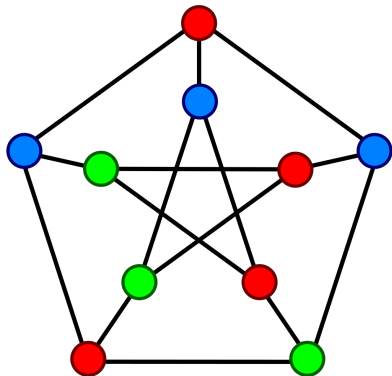
- In its simplest form, it is a way of coloring the vertices of a graph such that no two adjacent vertices are of the same color.
- This is called a vertex coloring.



Graph Coloring

https://en.wikipedia.org/wiki/Graph_coloring

- A proper vertex coloring of the graph with 3 colors, the minimum number possible.



Automata, Computability, and Complexity

Sipser, 0.1, p-2

- You have several options when you confront a problem that appears to be computationally hard.
- First, by understanding which aspect of the problem is at the root of the difficulty, you may be able to alter it so that the problem is more easily solvable.
- Second, you may be able to settle for less than a perfect solution to the problem.
- In certain cases, finding solutions that only approximate the perfect one is relatively easy.
- Third, some problems are hard only in the worst case situation, but easy most of the time.



Automata, Computability, and Complexity

Sipser, 0.1, p-2

- Depending on the application, you may be satisfied with a procedure that occasionally is slow but usually runs quickly.
- Finally, you may consider alternative types of computation, such as randomized computation, that can speed up certain tasks.



Travelling Salesman Problem — Computing a Solution

https://en.wikipedia.org/wiki/Travelling_salesman_problem

Exact Algorithms

- The most direct solution would be to try all permutations (ordered combinations) and see which one is cheapest (using brute force search).
- The running time for this approach lies within a polynomial factor of $O(n!)$, the factorial of the number of cities.
- So this solution becomes impractical even for only 20 cities.



Travelling Salesman Problem — Computing a Solution

https://en.wikipedia.org/wiki/Travelling_salesman_problem

Exact Algorithms

- One of the earliest applications of dynamic programming is the Held–Karp algorithm that solves the problem in time $O(n^2 2^n)$.



Travelling Salesman Problem — Computing a Solution

https://en.wikipedia.org/wiki/Travelling_salesman_problem

Heuristic and Approximation Algorithms

- Various heuristics and approximation algorithms, which quickly yield good solutions have been devised.
- Modern methods can find solutions for extremely large problems (millions of cities) within a reasonable time which are with a high probability just 2–3% away from the optimal solution.



Travelling Salesman Problem — Computing a Solution

https://en.wikipedia.org/wiki/Travelling_salesman_problem

Heuristic and Approximation Algorithms

- The nearest neighbour (NN) algorithm (a greedy algorithm) lets the salesman choose the nearest unvisited city as his next move.



Heuristic and Approximation Algorithms

- The Christofides algorithm follows a similar outline but combines the minimum spanning tree with a solution of another problem, minimum-weight perfect matching.
- This gives a TSP tour which is at most 1.5 times the optimal.



Travelling Salesman Problem — Computing a Solution

https://en.wikipedia.org/wiki/Travelling_salesman_problem

Heuristic and Approximation Algorithms

- Optimized Markov chain algorithms which use local searching heuristic sub-algorithms can find a route extremely close to the optimal route for 700 to 800 cities.



Automata, Computability, and Complexity

Sipser, 0.1, p-2

- One applied area that has been affected directly by complexity theory is the ancient field of cryptography.
- In most fields, an easy computational problem is preferable to a hard one because easy ones are cheaper to solve.
- Cryptography is unusual because it specifically requires computational problems that are hard, rather than easy.
- Secret codes should be hard to break without the secret key or password.
- Complexity theory has pointed cryptographers in the direction of computationally hard problems around which they have designed revolutionary new codes.



The RSA Algorithm

Cryptography and Network Security Principles and Practice, 7th Edition, William Stallings

- Developed in 1977 by Ron Rivest, Adi Shamir, and Len Adleman at MIT and first published in 1978.
- The Rivest-Shamir-Adleman (RSA) scheme has since that time reigned supreme as the most widely accepted and implemented general-purpose approach to public-key encryption.



The RSA Algorithm — *continued*

Cryptography and Network Security Principles and Practice, 7th Edition, William Stallings

- The public key is chosen using $n = pq$, where, p , q , two prime numbers are chosen privately.
- The private key is calculated using p and q .
- In order to break the algorithm, the attacker needs to factor n into its two prime factors.



Automata, Computability, and Complexity

Sipser, 0.1, p-3

- During the first half of the twentieth century, mathematicians such as Kurt Gödel, Alan Turing, and Alonzo Church discovered that certain basic problems cannot be solved by computers.
- One example of this phenomenon is the problem of determining whether a mathematical statement is true or false.
- This task is the bread and butter of mathematicians.
- It seems like a natural for solution by computer because it lies strictly within the realm of mathematics.
- But no computer algorithm can perform this task.



Mathematical Expressions, True or False?

From StackExchange Mathematics*

- Deciding whether a statement with mathematical expressions is true or false.

1 $2x - 5 = 1$ if and only if $x = 3$

2 If $x^2 \geq y^2$, then $x < 0$ or $x \geq y$

-
- For 1 we must prove the statement in both directions.
 - That is:
 - If $2x - 5 = 1$, then $x = 3$
and
 - If $x = 3$, then $2x - 5 = 1$

* <https://math.stackexchange.com/questions/925771/deciding-whether-a-statement-with-mathematical-expressions-is-true-or-false>



Mathematical Expressions, True or False?

From StackExchange Mathematics*

- Deciding whether a statement with mathematical expressions is true or false.

1 $2x - 5 = 1$ if and only if $x = 3$

2 If $x^2 \geq y^2$, then $x < 0$ or $x \geq y$

-
- How we can determine whether an expression such as $2x - 5 = 1$ is true or false?

*

[https://math.stackexchange.com/questions/925771/](https://math.stackexchange.com/questions/925771/deciding-whether-a-statement-with-mathematical-expressions-is-true-or-false)

[deciding-whether-a-statement-with-mathematical-expressions-is-true-or-false](https://math.stackexchange.com/questions/925771/deciding-whether-a-statement-with-mathematical-expressions-is-true-or-false)



Mathematical Expressions, True or False?

From StackExchange Mathematics*

- Deciding whether a statement with mathematical expressions is true or false.

1 $2x - 5 = 1$ if and only if $x = 3$

2 If $x^2 \geq y^2$, then $x < 0$ or $x \geq y$

-
- For example, logical implication in the form of “If P , then Q ” is true when both P and Q are true or P is false.

* <https://math.stackexchange.com/questions/925771/deciding-whether-a-statement-with-mathematical-expressions-is-true-or-false>



Mathematical Expressions, True or False?

From StackExchange Mathematics*

- Deciding whether a statement with mathematical expressions is true or false.

1 $2x - 5 = 1$ if and only if $x = 3$

2 If $x^2 \geq y^2$, then $x < 0$ or $x \geq y$

-
- So, in the case of $2x - 5 = 1$, is this true or false?

* <https://math.stackexchange.com/questions/925771/deciding-whether-a-statement-with-mathematical-expressions-is-true-or-false>



Mathematical Expressions, True or False?

From StackExchange Mathematics*

- Deciding whether a statement with mathematical expressions is true or false.

1 $2x - 5 = 1$ if and only if $x = 3$

2 If $x^2 \geq y^2$, then $x < 0$ or $x \geq y$

-
- In the case of $x^2 \geq y^2$, is this true or false?

* <https://math.stackexchange.com/questions/925771/deciding-whether-a-statement-with-mathematical-expressions-is-true-or-false>



Mathematical Expressions, True or False? — *continued*

- Suppose that $2x - 5 = 1$.
- Then we can rearrange it as follows:

$$2x - 5 = 1$$

$$\implies 2x = 6$$

$$\implies x = 3.$$

- Moreover, if $x = 3$, then $2x - 5 = 2 \cdot 3 - 5 = 1$.
- Therefore $2x - 5 = 1 \iff x = 3$.



Mathematical Expressions, True or False? — *continued*

- We never decided whether the statement $2x - 5 = 1$ was actually true.
- We just showed the implication, that if it is true, then the statement $x = 3$ is also true.
- This is how one proves implications in mathematics.



Mathematical Expressions, True or False? — *continued*

1 $2x - 5 = 1$ if and only if $x = 3$

2 If $x^2 \geq y^2$, then $x < 0$ or $x \geq y$

-
- It turns out the statement $2x - 5 = 1$ is only *sometimes* true.
 - what we have shown is that it is true exactly when $x = 3$ is true, and false otherwise.



Mathematical Expressions, True or False? — *continued*

1 $2x - 5 = 1$ if and only if $x = 3$

2 If $x^2 \geq y^2$, then $x < 0$ or $x \geq y$

-
- For the second statement, we do not know whether or not $x^2 \geq y^2$ is actually true.
 - It actually depends on the values of x and y .
 - What we can say though, is that if it is true, then:

$$\begin{aligned} & x^2 \geq y^2 \\ \implies & x^2 - y^2 \geq 0 \\ \implies & (x - y)(x + y) \geq 0 \end{aligned}$$



Mathematical Expressions, True or False? — *continued*

1 $2x - 5 = 1$ if and only if $x = 3$

2 If $x^2 \geq y^2$, then $x < 0$ or $x \geq y$

-
- At this point there are two possibilities.
 - Either both $(x - y)$ and $(x + y)$ are nonnegative, or one is nonpositive and the other is negative.



Mathematical Expressions, True or False? — *continued*

1 $2x - 5 = 1$ if and only if $x = 3$

2 If $x^2 \geq y^2$, then $x < 0$ or $x \geq y$

-
- In the former case, $x - y \geq 0 \implies x \geq y$.
 - In the latter case, if one of $x + y$ and $x - y$ is nonpositive, and the other negative, then their sum is strictly negative, so $(x - y) + (x + y) = 2x < 0 \implies x < 0$.



Mathematical Expressions, True or False? — *continued*

1 $2x - 5 = 1$ if and only if $x = 3$

2 If $x^2 \geq y^2$, then $x < 0$ or $x \geq y$

-
- This proves the statement, that if $x^2 \geq y^2$, then either $x < 0$ or $x \geq y$.
 - We still don't know whether the individual statements are inherently true, yet we can prove the implication regardless.



Automata, Computability, and Complexity

Sipser, 0.1, p-3

- Among the consequences of this profound result was the development of ideas concerning theoretical models of computers that eventually would help lead to the construction of actual computers.



Automata, Computability, and Complexity

Sipser, 0.1, p-3

- The theories of computability and complexity are closely related.
- In complexity theory, the objective is to classify problems as easy ones and hard ones.
- Whereas in computability theory, the classification of problems is by those that are solvable and those that are not.
- Computability theory introduces several of the concepts used in complexity theory.



Automata, Computability, and Complexity

Sipser, 0.1, p-3

- Automata theory deals with the definitions and properties of mathematical models of computation.
- These models play a role in several applied areas of computer science.
- One model, called the finite automaton, is used in text processing, compilers, and hardware design.
- Another model, called the context-free grammar, is used in programming languages and artificial intelligence.



Automata, Computability, and Complexity

Sipser, 0.1, p-3

- Automata theory is an excellent place to begin the study of the theory of computation.
- The theories of computability and complexity require a precise definition of a computer.
- Automata theory allows practice with formal definitions of computation as it introduces concepts relevant to other non-theoretical areas of computer science.



Automata, a Motivating Example

Adapted from

<http://infolab.stanford.edu/~ullman/ialc/spr10/slides/fa1.pdf>

- Recognizing strings ending in “ing”

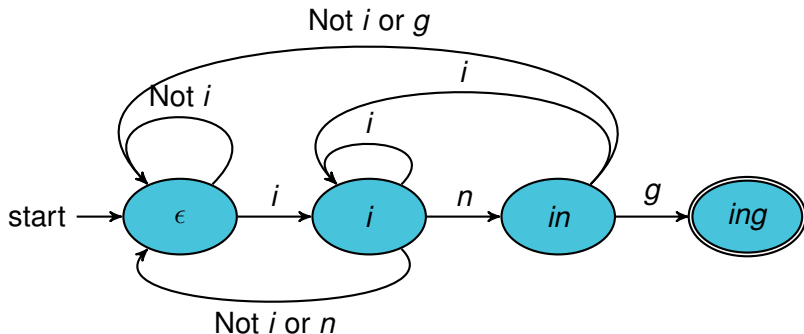


Automata, a Motivating Example

Adapted from

<http://infolab.stanford.edu/~ullman/ialc/spr10/slides/fa1.pdf>

■ Recognizing strings ending in “ing”



Automata, a Motivating Example

Adapted from

<http://infolab.stanford.edu/~ullman/ialc/spr10/slides/fa1.pdf>

- Protocol for sending data.

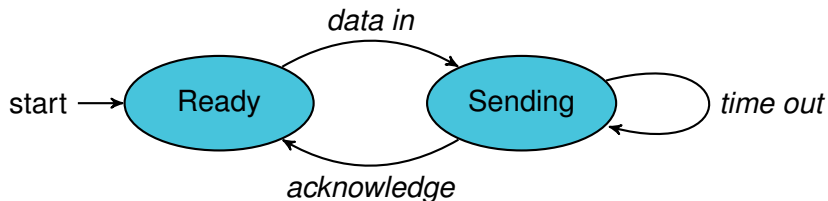


Automata, a Motivating Example

Adapted from

<http://infolab.stanford.edu/~ullman/ialc/spr10/slides/fa1.pdf>

■ Protocol for sending data.



Computability, a Motivating Example — Strange Planet

Adapted from

<http://infolab.stanford.edu/~ullman/ialc/spr10/slides/fal.pdf>

- On a distant planet, there are three colors of stones, **R** (red), **G** (green), and **B** (blue).
- When two different colored stones get in contact with each other:
 - 1 These two stones are destroyed.
 - 2 Two more stones of a third color are created.



Strange Planet — *continued*

- Observation: the number of stones never changes.
- The planet fails if at some point all stones are of the same color.
- Then, no more creation can take place.
- State = sequence of three integers — the numbers of stones of color **R** (red), **G** (green), and **B** (blue).



Strange Planet — *continued*

- In a given state, must the planet eventually fail?
- In a given state, is it possible for the planet to fail, if the wrong contact choices are made?



Strange Planet — *continued*

- In a given state, must the planet eventually fail?
- In a given state, is it possible for the planet to fail, if the wrong contact choices are made?
- These questions mirror real ones about protocols.
- “Can the planet fail?” is like asking whether a protocol can enter some undesired or error state.
- “Must the planet fail?” is like asking whether a protocol is guaranteed to terminate.
- Here, “failure” is really the good condition of termination.



Strange Planet — *continued*

Transitions

- An **R**-event occurs when stones of color **G** and **B** touch and are replaced by two **R**'s.
- Analogously: **G**-events and **B**-events.
- We represent these by symbols **R**, **G**, and **B**, respectively.



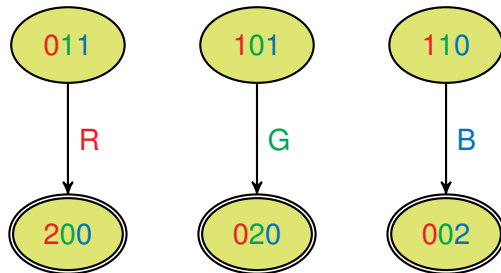
Strange Planet — *continued*

2 Stones



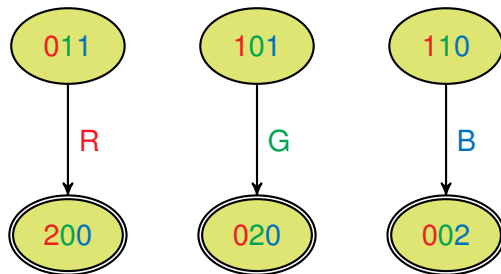
Strange Planet — *continued*

2 Stones



Strange Planet — *continued*

2 Stones



- All states are *must-fail* states.



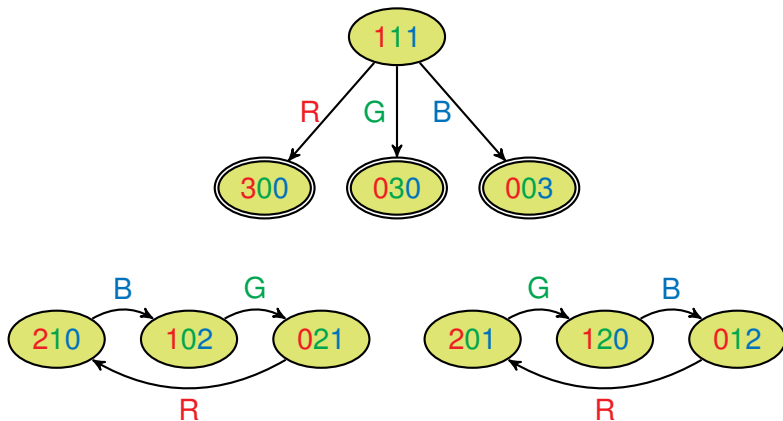
Strange Planet — *continued*

3 Stones



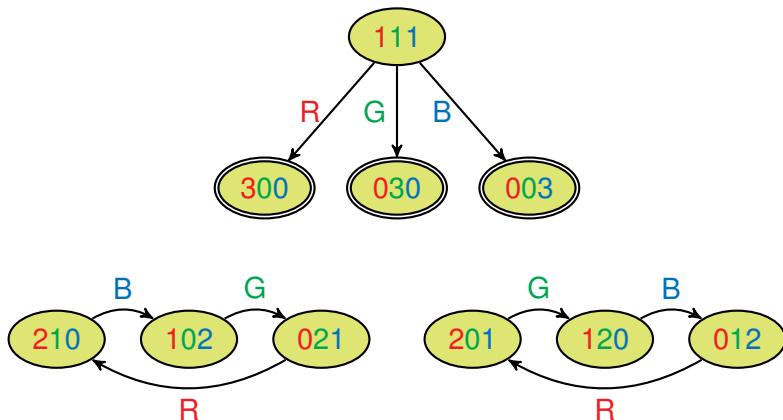
Strange Planet — *continued*

3 Stones



Strange Planet — *continued*

3 Stones



- Four states are *must-fail* states.
- The others are *can't-fail* states.
- State 111 has several transitions.



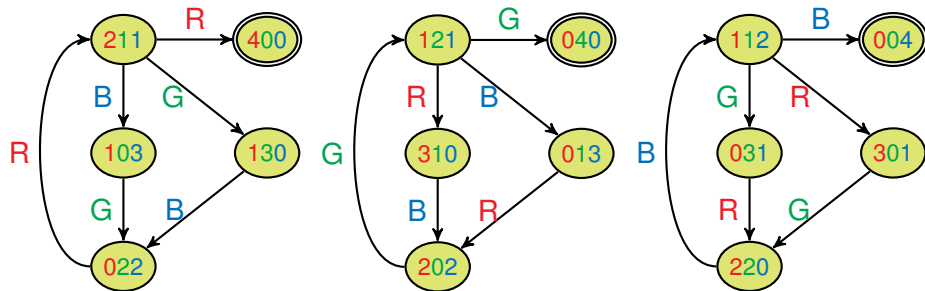
Strange Planet — *continued*

4 Stones



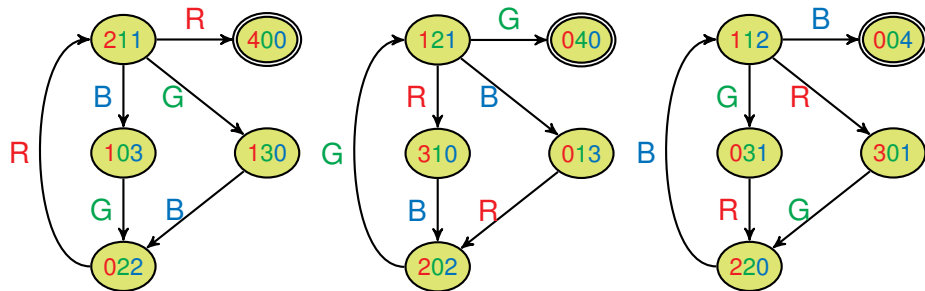
Strange Planet — *continued*

4 Stones



Strange Planet — *continued*

4 Stones



- States 400 and similar ones are *must-fail* states.
- All other states are *might-fail* states.



Strange Planet — *continued*

Taking Advantage of Symmetry

- The ability to fail depends only on the set of numbers of the three colors, not on which color has which number.
- Let's represent states by the list of counts, sorted by largest-first.
- Only one transition symbol, x .



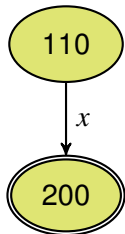
Strange Planet — *continued*

The Cases of 2, 3, 4 Stones



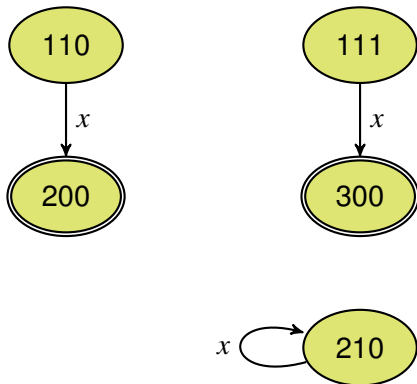
Strange Planet — *continued*

The Cases of 2, 3, 4 Stones



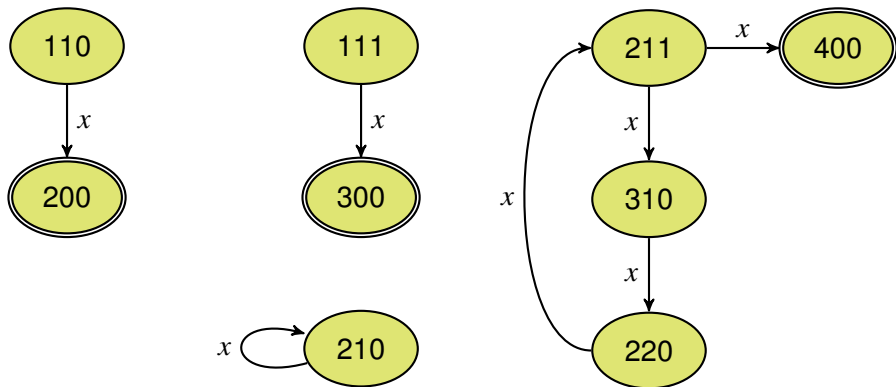
Strange Planet — *continued*

The Cases of 2, 3, 4 Stones



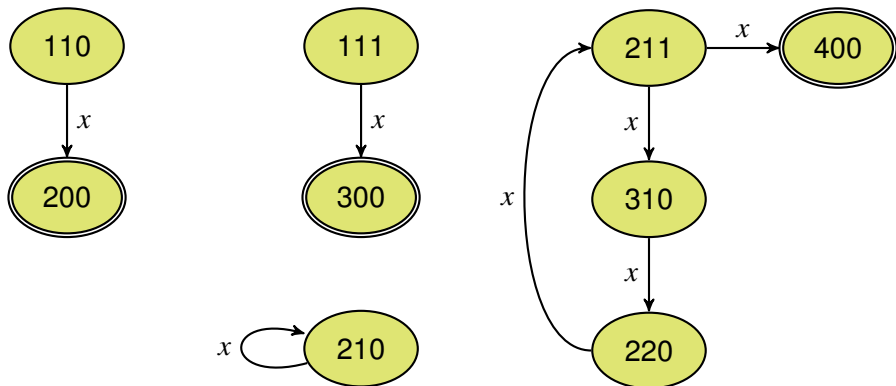
Strange Planet — *continued*

The Cases of 2, 3, 4 Stones



Strange Planet — *continued*

The Cases of 2, 3, 4 Stones

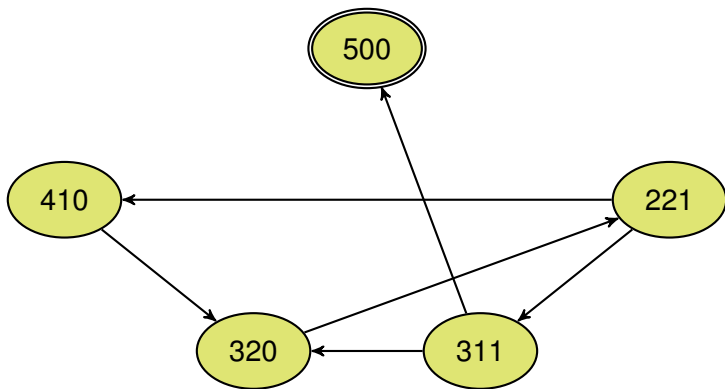


- For the case $n = 4$, there is *nondeterminism*.
- Different transitions are possible from 211 on the same input.



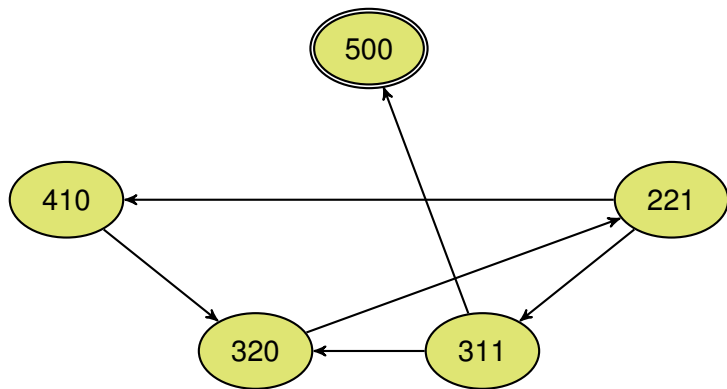
Strange Planet — *continued*

5 Stones



Strange Planet — *continued*

5 Stones

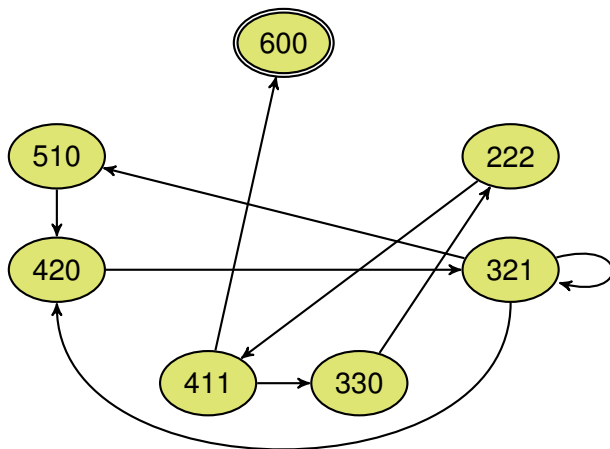


- 500 is a must-fail state.
- All others are might-fail states.

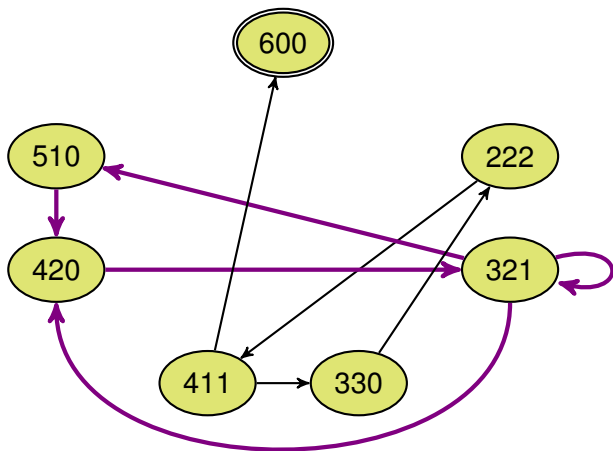


6 Stones

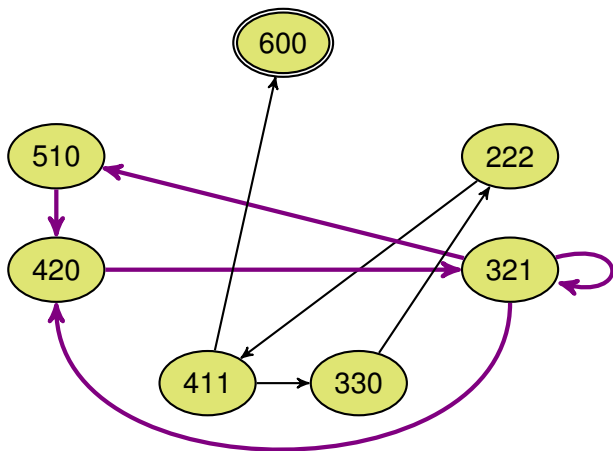
6 Stones



6 Stones

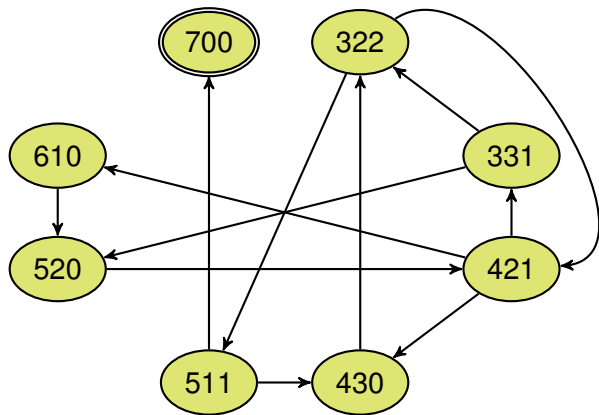


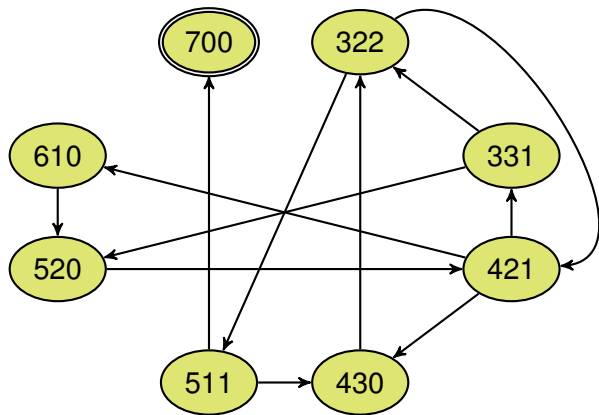
6 Stones



- 600 is a must-fail state.
- 510, 420, and 321 are can't-fail states.
- 411, 330, and 222 are might-fail states.

7 Stones





- 700 is a must-fail state.
- All others are might-fail states.

Strange Planet — *continued*

Questions for Thought

- 1 Without symmetry, how many states are there with n individuals?
- 2 What if we use symmetry?
- 3 For n individuals, how do you tell whether a state is “must-fail,” “might-fail,” or “can’t-fail”?





End of Slides

