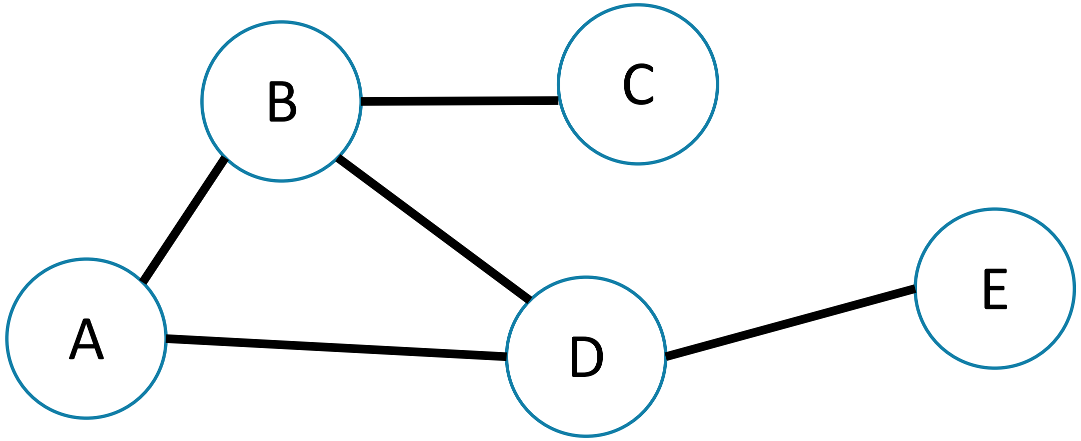
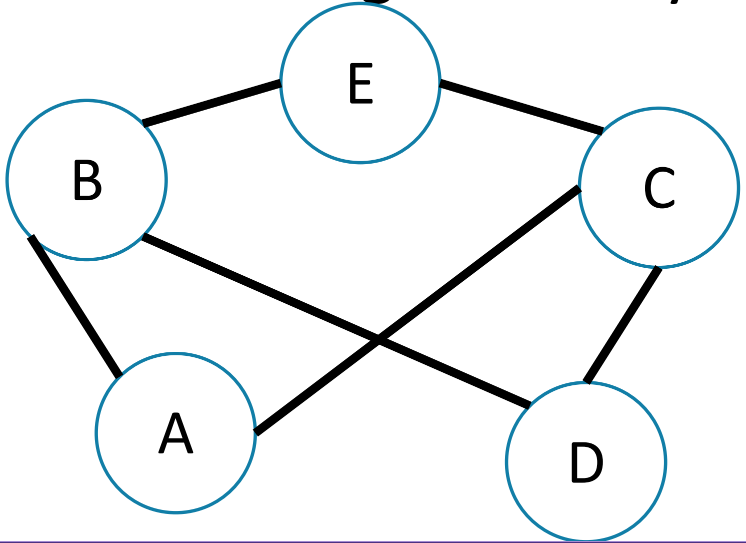
P vs. NP

Data Structures and Algorithms

Warm-up

Can these graphs be 2-colored? If so find a 2-coloring. If not try to explain why one doesn't exist.



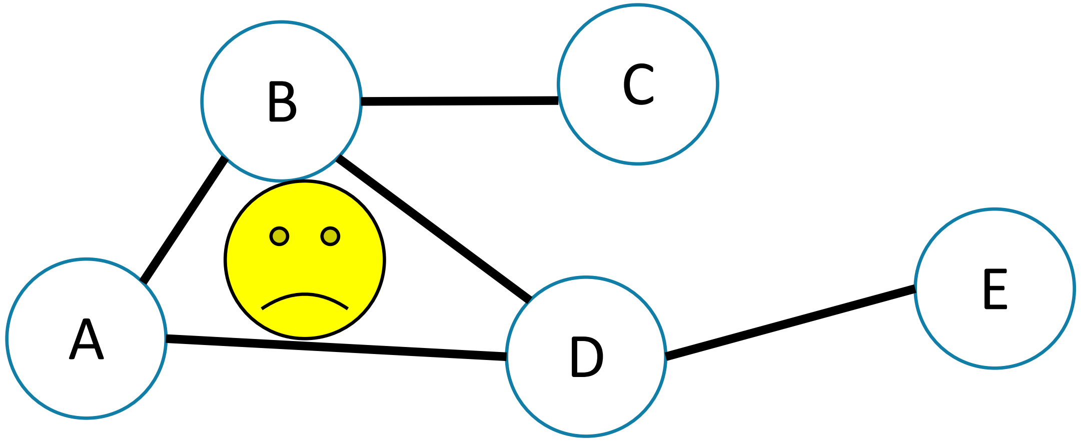


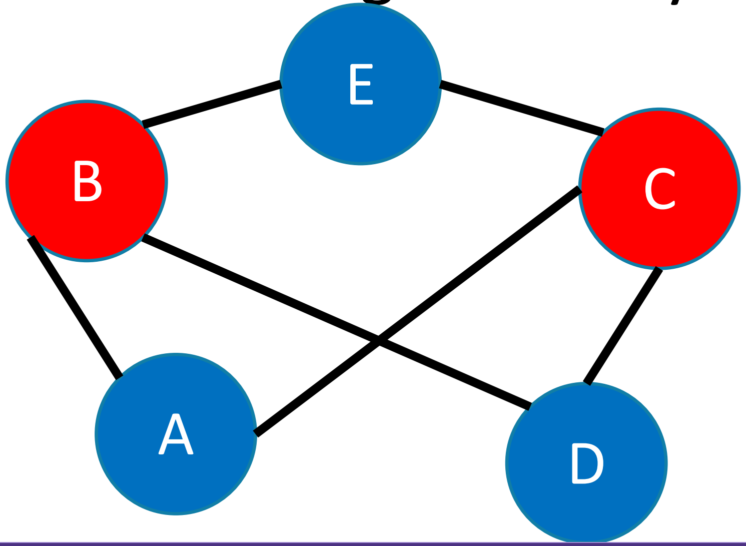
2-Coloring

Given an undirected, unweighted graph G, color each vertex "red" or "blue" such that the endpoints of every edge are different colors (or report no such coloring exists).

Warm-up

Can these graphs be 2-colored? If so find a 2-coloring. If not try to explain why one doesn't exist.





2-Coloring

Given an undirected, unweighted graph G, color each vertex "red" or "blue" such that the endpoints of every edge are different colors (or report no such coloring exists).

Last Time

We found an algorithm to solve 2-SAT, and said we'd try to reduce 2-Coloring to 2-SAT.

2-Satisfiability ("2-SAT")

Given: A set of Boolean variables, and a list of requirements, each of the form:

```
    variable1==[True/False] || variable2==[True/False]
```

Find: A setting of variables to "true" and "false" so that **all** of the requirements evaluate to "true"

2-Coloring

Given an undirected, unweighted graph G, color each vertex "red" or "blue" such that the endpoints of every edge are different colors (or report no such coloring exists).

2-Coloring

Why would we want to 2-color a graph?

- We need to divide the vertices into two sets, and edges represent vertices that **can't** be together.

You can modify BFS to come up with a 2-coloring (or determine none exists)

- This is a good exercise!

But coming up with a whole new idea sounds like **work.**

And we already came up with that cool 2-SAT algorithm.

- Maybe we can be lazy and just use that!
- Let's **reduce** 2-Coloring to 2-SAT!

Use our 2-SAT algorithm
to solve 2-Coloring

A Reduction

We need to describe 2 steps

1. How to turn a graph for a 2-color problem into an input to 2-SAT

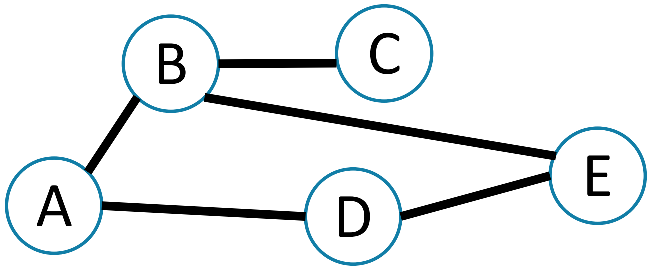
2. How to turn the ANSWER for that 2-SAT input into the answer for the original 2-coloring problem.

How can I describe a two coloring of my graph?

- Have a variable for each vertex – is it red?

How do I make sure every edge has different colors? I need one red endpoint and one blue one, so this better be true to have an edge from v1 to v2:

```
(v1IsRed || v2isRed) && (!v1IsRed || !v2IsRed)
```



Transform Input

2-SAT Algorithm

Transform Output

Taking a step back

So far this quarter, we’ve given you a bunch of problems that (with some clever tricks) you can solve really quickly.

To wrap up the quarter, we’re going to talk about problems we don’t know how to solve really quickly.

But first, what do I mean by “quickly”?

Running Times

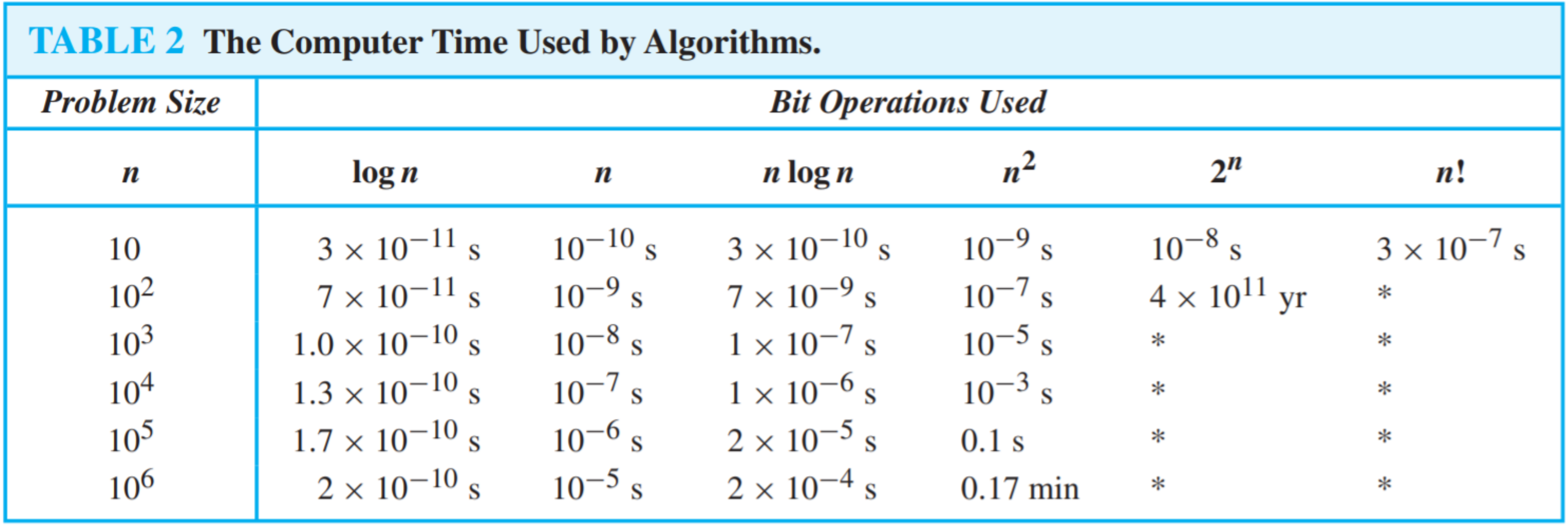


TABLE 2 The Computer Time Used by Algorithms.

Problem Size	*Bit Operations Used*					
n	$\log n$	n	$n \log n$	n^2	2^n	$n!$
10	3×10^{-11} s	10^{-10} s	3×10^{-10} s	10^{-9} s	10^{-8} s	3×10^{-7} s
10^2	7×10^{-11} s	10^{-9} s	7×10^{-9} s	10^{-7} s	4×10^{11} yr	*
10^3	1.0×10^{-10} s	10^{-8} s	1×10^{-7} s	10^{-5} s	*	*
10^4	1.3×10^{-10} s	10^{-7} s	1×10^{-6} s	10^{-3} s	*	*
10^5	1.7×10^{-10} s	10^{-6} s	2×10^{-5} s	0.1 s	*	*
10^6	2×10^{-10} s	10^{-5} s	2×10^{-4} s	0.17 min	*	*

Table from Rosen’s Discrete Mathematics textbook

How big of a problem can we solve for an algorithm with the given running times?

“*” means more than 10^{100} years.

Efficient

We'll consider a problem "efficiently solvable" if it has a polynomial time algorithm.

I.e. an algorithm that runs in time $O(n^k)$ where k is a constant.

Are these algorithms always actually efficient?

Well………no

Your n^{10000} algorithm or even your $2^{2^{2^{2^2}}} \cdot n^3$ algorithm probably aren't going to finish anytime soon.

But these edge cases are rare, and polynomial time is good as a low bar

- If we can't even find an n^{10000} algorithm, we should probably rethink our strategy

Decision Problems

For today, we’re going to talk about **decision problems**.

Problems that have a “yes” or “no” answer.

Why?

Theory reasons (ask me later).

But it’s not too bad

- most problems can be rephrased as very similar decision problems.

E.g. instead of “find the shortest path from s to t” ask
Is there a path from s to t of length at most k?

P

P (stands for “Polynomial”)

The set of all decision problems that have an algorithm that runs in time $O(n^k)$ for some constant k.

The decision version of all problems we’ve solved in this class are in P.

P is an example of a “complexity class”

A set of problems that can be solved under some limitations (e.g. with some amount of memory or in some amount of time).

I’ll know it when I see it.

Another class of problems we want to talk about.

“I’ll know it when I see it” Problems.

Decision Problems such that:

If the answer is YES, you can prove the answer is yes by

- Being given a “proof” or a “certificate”
- Verifying that certificate in polynomial time.

What certificate would be convenient for short paths?

- The path itself. Easy to check the path is really in the graph and really short.

I’ll know it when I see it.

For each of the following problems: what is your “certificate” that the answer is YES, and how do you check it?

Light Spanning Tree:
Is there a spanning tree of graph G of weight at most k?

The spanning tree itself.
Verify by checking it really connects every vertex and its weight.

2-Coloring:
Can you color vertices of a graph red and blue so every edge has differently colored endpoints?

The coloring.
Verify by checking each edge.

2-SAT:
Given a set of variables and a list of requirements:
(variable==[T/F] || variable==[T/F])
Find a setting of the variables to make every requirement true.

The assignment of variables.
Verify by checking each requirement.

If the answer is NO, is there a certificate?
There might be, or there might not be – it’s not required by the definition.

I’ll know it when I see it.

More formally,

NP (stands for “nondeterministic polynomial”)

The set of all decision problems such that if the answer is YES, there is a proof of that which can be verified in polynomial time.

It’s a common misconception that NP stands for “not polynomial”
Please never ever ever ever say that.

Please.

Every time you do a theoretical computer scientist sheds a single tear.

(That theoretical computer scientist is me)

P vs. NP

P vs. NP

Are P and NP the same complexity class?
That is, can every problem that can be verified **in polynomial time also be** solved **in polynomial time.**

No one knows the answer to this question.

In fact, it’s the biggest open problem in Computer Science.

Hard Problems

Let's say we want to prove that every problem in NP can actually be solved efficiently.

We might want to start with a really hard problem in NP.

What is the hardest problem in NP?

What does it mean to be a hard problem?

Reductions are a good definition:

- If A reduces to B then "A $\leq$ B" (in terms of difficulty)
  - Once you have an algorithm for B, you have one for A automatically from the reduction!

NP-Completeness

NP-complete

The problem B is NP-complete if B is in NP and
for all problems A in NP, A reduces to B.

An NP-complete problem is a “hardest” problem in NP.

If you have an algorithm to solve an NP-complete problem, you have an algorithm for **every** problem in NP.

An NP-complete problem is a **universal language** for encoding “I’ll know it when I see it” problems.

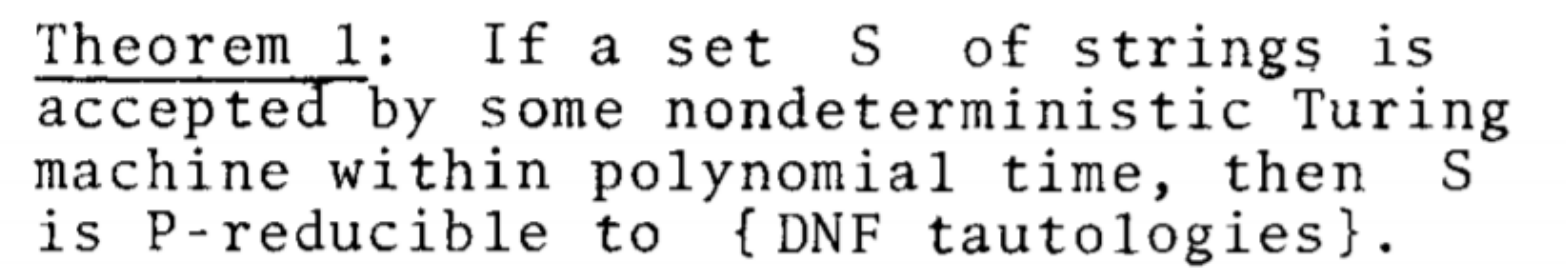
Does one of these exist?

NP-Completeness

An NP-complete problem does exist!

Cook-Levin Theorem (1971)

3-SAT is NP-complete



Theorem 1: If a set S of strings is accepted by some nondeterministic Turing machine within polynomial time, then S is P-reducible to {DNF tautologies}.

This sentence (and the proof of it) won Cook the Turing Award.

2-SAT vs. 3-SAT

2-Satisfiability (“2-SAT”)

Given: A set of Boolean variables, and a list of requirements, each of the form:

```
    variable1==[True/False] || variable2==[True/False]
```

Find: A setting of variables to “true” and “false” so that **all** of the requirements evaluate to “true”

3-Satisfiability (“3-SAT”)

Given: A set of Boolean variables, and a list of requirements, each of the form:

```
variable1==[True/False]||variable2==[True/False]||variable3==[True/False]
```

Find: A setting of variables to “true” and “false” so that **all** of the requirements evaluate to “true”

2-SAT vs. 3-SAT

2-Satisfiability ("2-SAT")

Given: A set of Boolean variables, and a list of requirements, each of the form:

```
    variable1==[True/False] || variable2==[True/False]
```

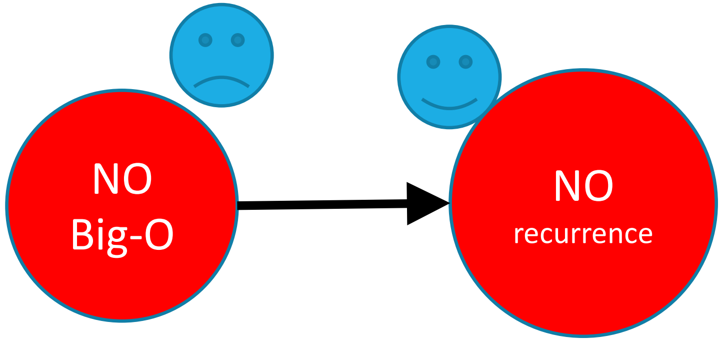
Find: A setting of variables to "true" and "false" so that **all** of the requirements evaluate to "true"

Our first try at 2-SAT (just try all variable settings) would have taken $O(2^Q S)$ time.

But we came up with a really clever graph that reduced the time to $O(Q + S)$ time.

2-SAT vs. 3-SAT

Can we do the same for 3-SAT?



For 2-SAT we thought we had 2^Q options, but we realized that we didn't have as many choices as we thought – once we made a few choices, our hand was forced and we didn't have to check all possibilities.

3-Satisfiability ("3-SAT")

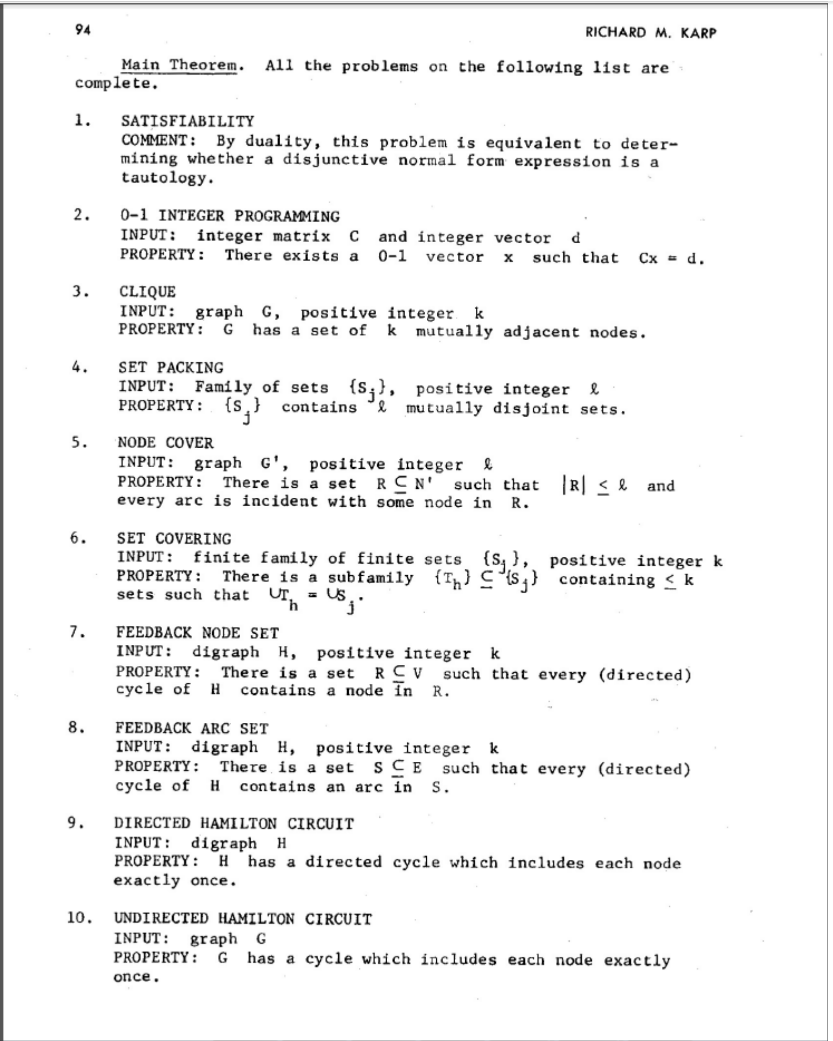
Given: A set of Boolean variables, and a list of requirements, each of the form:

```
variable1==[True/False]||variable2==[True/False]||variable3==[True/False]
```

Find: A setting of variables to "true" and "false" so that **all** of the requirements evaluate to "true"

NP-Complete Problems

But Wait! There's more!



94 RICHARD M. KARP

Main Theorem. All the problems on the following list are complete.

1. SATISFIABILITY
COMMENT: By duality, this problem is equivalent to determining whether a disjunctive normal form expression is a tautology.

2. 0-1 INTEGER PROGRAMMING
INPUT: integer matrix C and integer vector d
PROPERTY: There exists a 0-1 vector x such that $Cx = d$.

3. CLIQUE
INPUT: graph G, positive integer k
PROPERTY: G has a set of k mutually adjacent nodes.

4. SET PACKING
INPUT: Family of sets $\{S_j\}$, positive integer ℓ
PROPERTY: $\{S_j\}$ contains ℓ mutually disjoint sets.

5. NODE COVER
INPUT: graph G', positive integer ℓ
PROPERTY: There is a set $R \subseteq N'$ such that $|R| \leq \ell$ and every arc is incident with some node in R.

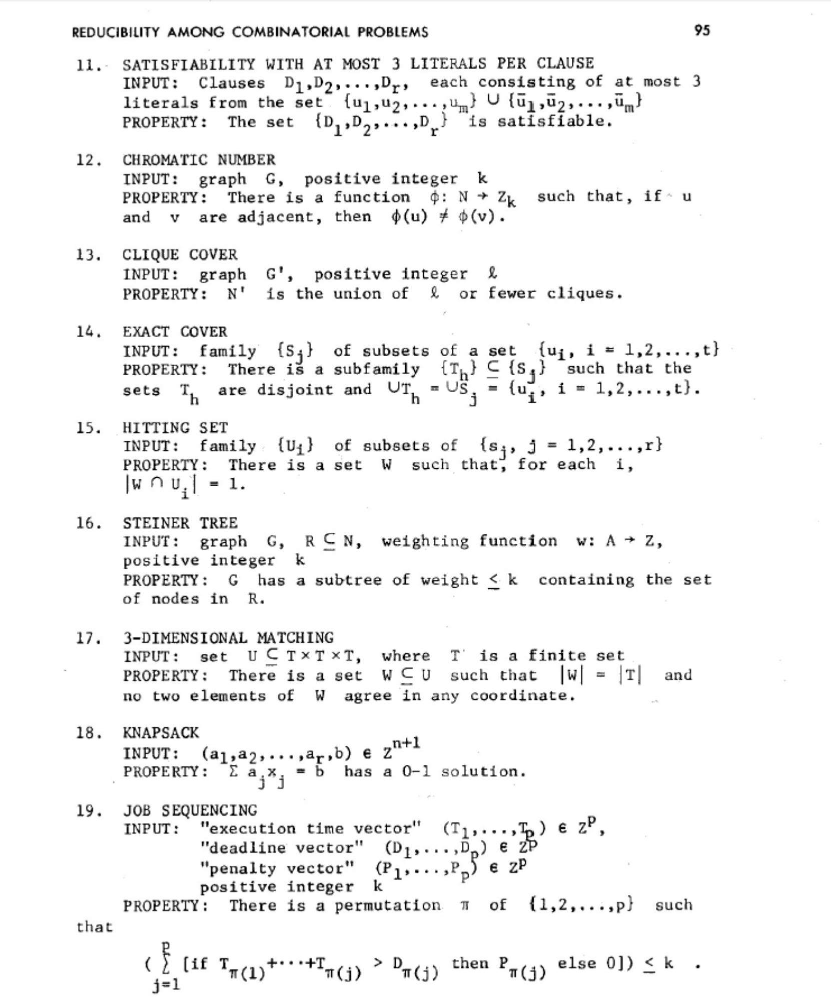
6. SET COVERING
INPUT: finite family of finite sets $\{S_j\}$, positive integer k
PROPERTY: There is a subfamily $\{T_h\} \subseteq \{S_j\}$ containing $\leq k$ sets such that $\cup T_h = \cup S_j$.

7. FEEDBACK NODE SET
INPUT: digraph H, positive integer k
PROPERTY: There is a set $R \subseteq V$ such that every (directed) cycle of H contains a node in R.

8. FEEDBACK ARC SET
INPUT: digraph H, positive integer k
PROPERTY: There is a set $S \subseteq E$ such that every (directed) cycle of H contains an arc in S.

9. DIRECTED HAMILTON CIRCUIT
INPUT: digraph H
PROPERTY: H has a directed cycle which includes each node exactly once.

10. UNDIRECTED HAMILTON CIRCUIT
INPUT: graph G
PROPERTY: G has a cycle which includes each node exactly once.



REDUCIBILITY AMONG COMBINATORIAL PROBLEMS 95

11. SATISFIABILITY WITH AT MOST 3 LITERALS PER CLAUSE
INPUT: Clauses $D_1, D_2, \ldots, D_r$, each consisting of at most 3 literals from the set $\{u_1, u_2, \ldots, u_m\} \cup \{\bar{u}_1, \bar{u}_2, \ldots, \bar{u}_m\}$
PROPERTY: The set $\{D_1, D_2, \ldots, D_r\}$ is satisfiable.

12. CHROMATIC NUMBER
INPUT: graph G, positive integer k
PROPERTY: There is a function $\phi: N \to Z_k$ such that, if u and v are adjacent, then $\phi(u) \neq \phi(v)$.

13. CLIQUE COVER
INPUT: graph G', positive integer ℓ
PROPERTY: N' is the union of ℓ or fewer cliques.

14. EXACT COVER
INPUT: family $\{S_j\}$ of subsets of a set $\{u_i, i = 1,2,\ldots,t\}$
PROPERTY: There is a subfamily $\{T_h\} \subseteq \{S_j\}$ such that the sets T_h are disjoint and $\cup T_h = \cup S_j = \{u_i, i = 1,2,\ldots,t\}$.

15. HITTING SET
INPUT: family $\{U_i\}$ of subsets of $\{s_j, j = 1,2,\ldots,r\}$
PROPERTY: There is a set W such that, for each i, $|W \cap U_i| = 1$.

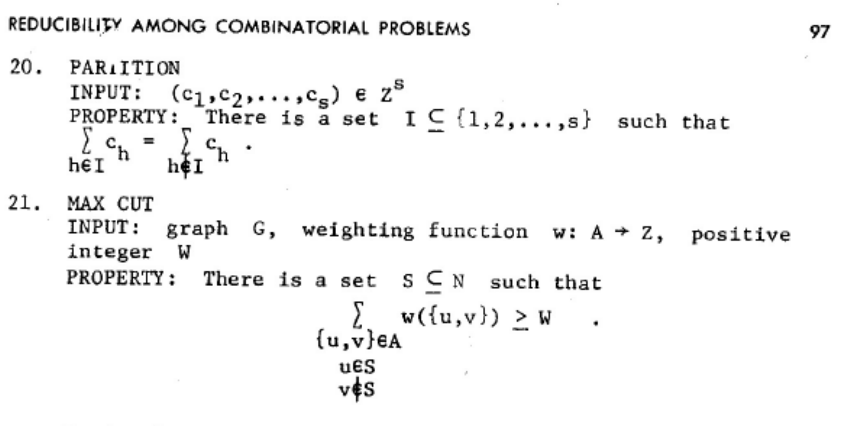
16. STEINER TREE
INPUT: graph G, $R \subseteq N$, weighting function $w: A \to Z$, positive integer k
PROPERTY: G has a subtree of weight $\leq k$ containing the set of nodes in R.

17. 3-DIMENSIONAL MATCHING
INPUT: set $U \subseteq T \times T \times T$, where T is a finite set
PROPERTY: There is a set $W \subseteq U$ such that $|W| = |T|$ and no two elements of W agree in any coordinate.

18. KNAPSACK
INPUT: $(a_1, a_2, \ldots, a_r, b) \in Z^{n+1}$
PROPERTY: $\Sigma a_j x_j = b$ has a 0-1 solution.

19. JOB SEQUENCING
INPUT: "execution time vector" $(T_1, \ldots, T_p) \in Z^p$,
"deadline vector" $(D_1, \ldots, D_p) \in Z^p$
"penalty vector" $(P_1, \ldots, P_p) \in Z^p$
positive integer k
PROPERTY: There is a permutation π of $\{1,2,\ldots,p\}$ such that

$$\left(\sum_{j=1}^{p} [\text{if } T_{\pi(1)} + \cdots + T_{\pi(j)} > D_{\pi(j)} \text{ then } P_{\pi(j)} \text{ else } 0]\right) \leq k \quad .$$



REDUCIBILITY AMONG COMBINATORIAL PROBLEMS 97

20. PARıITION
INPUT: $(c_1, c_2, \ldots, c_s) \in Z^s$
PROPERTY: There is a set $I \subseteq \{1,2,\ldots,s\}$ such that $\sum_{h \in I} c_h = \sum_{h \notin I} c_h$.

21. MAX CUT
INPUT: graph G, weighting function $w: A \to Z$, positive integer W
PROPERTY: There is a set $S \subseteq N$ such that

$$\sum_{\substack{\{u,v\} \in A \\ u \in S \\ v \notin S}} w(\{u,v\}) \geq W \quad .$$

Karp's Theorem (1972)

A lot of problems are NP-complete

NP-Complete Problems

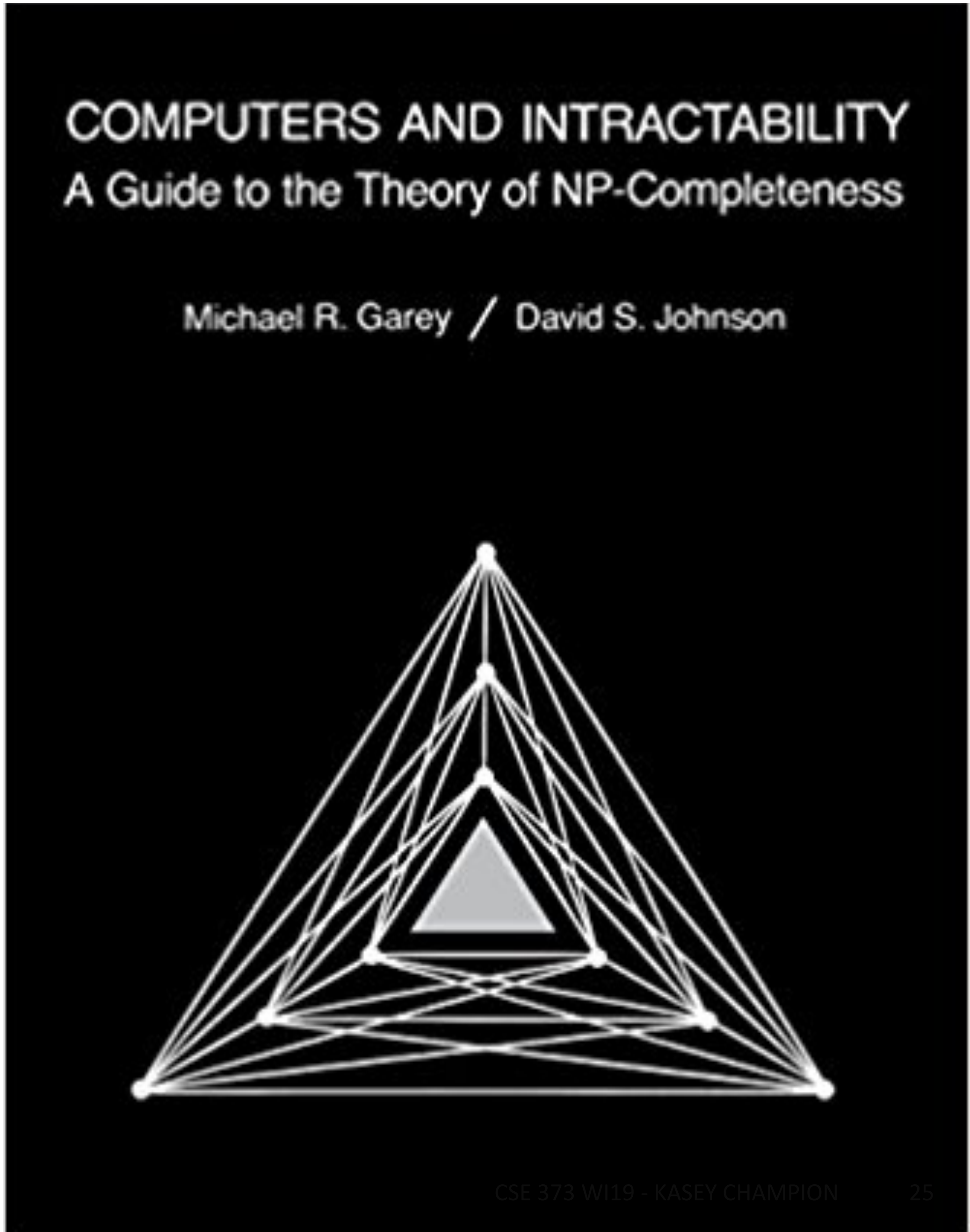
But Wait! There’s more!

By 1979, at least 300 problems had been proven NP-complete.

Garey and Johnson put a list of all the NP-complete problems they could find in this textbook.

Took almost 100 pages to just list them all.

No one has made a comprehensive list since.



COMPUTERS AND INTRACTABILITY

A Guide to the Theory of NP-Completeness

Michael R. Garey / David S. Johnson

NP-Complete Problems

But Wait! There’s more!

In the last month, mathematicians and computer scientists have put papers on the arXiv claiming to show (at least) 25 more problems are NP-complete.

There are literally thousands of NP-complete problems known.

And some of them look weirdly similar to problems we’ve already studied.

Dealing with NP-Completeness

Option 1: Maybe it’s a special case we understand

Maybe you don’t need to solve the general problem, just a special case

Option 2: Maybe it’s a special case we *don’t* understand (yet)

There are algorithms that are known to run quickly on “nice” instances. Maybe your problem has one of those.

One approach: Turn your problem into a SAT instance, find a solver and cross your fingers.

Dealing with NP-Completeness

Option 3: Approximation Algorithms

You might not be able to get an exact answer, but you might be able to get close.

Optimization version of Traveling Salesperson

Given a weighted graph, find a tour (a walk that visits every vertex and returns to its start) of minimum weight.

Algorithm:

Find a minimum spanning tree.

Have the tour follow the visitation order of a DFS of the spanning tree.

Theorem: This tour is at most twice as long as the best one.

Why should you care about P vs. NP

Most computer scientists are convinced that P≠NP.

Why should you care about this problem?

It’s your chance for:

$1,000,000. The Clay Mathematics Institute will give $1,000,000 to whoever solves P vs. NP (or any of the 5 remaining problems they listed)

To get a Turing Award

Why should you care about P vs. NP

Most computer scientists are convinced that P≠NP.

Why should you care about this problem?

It’s your chance for:

$1,000,000. The Clay Mathematics Institute will give $1,000,000 to whoever solves P vs. NP (or any of the 5 remaining problems they listed)

To get ~~a Turing Award~~ the Turing Award renamed after you.

Why Should You Care if P=NP?

Suppose P=NP.

Specifically that we found a genuinely in-practice efficient algorithm for an NP-complete problem. What would you do?

- $1,000,000 from the Clay Math Institute obviously, but what's next?

Why Should You Care if P=NP?

We found a genuinely in-practice efficient algorithm for an NP-complete problem. What would you do?

- Another $5,000,000 from the Clay Math Institute
- Put mathematicians out of work.
- Decrypt (essentially) all current internet communication.
- No more secure online shopping or online banking or online messaging…or online *anything.*

A world where P=NP is a very very different place from the world we live in now.

Why Should You Care if P≠NP?

We already expect P≠NP. Why should you care when we finally prove it?

P≠NP says something fundamental about the universe.

For some questions there is not a clever way to find the right answer

- Even though you'll know it when you see it.

There is actually a way to obscure information, so it cannot be found quickly no matter how clever you are.

Why Should You Care if P≠NP?

To prove P≠NP we need to better understand the differences between problems.

- Why do some problems allow easy solutions and others don’t?
- What is the structure of these problems?

We don’t care about P vs NP just because it has a huge effect about what the world looks like.

We will learn a lot about computation along the way.