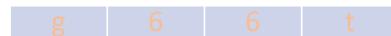


EC504 ALGORITHMS AND DATA STRUCTURES
FALL 2020 MONDAY & WEDNESDAY
2:30 PM - 4:15 PM



Prof: David Castañón, dac@bu.edu

GTF: Mert Toslali, toslali@bu.edu

Haoyang Wang: haoyangw@bu.edu

Christopher Liao: cliao25@bu.edu

Reminders:

- which time
know
- Hw 8 → M 4/26
 - Project → F 4/30
 - W 4/28 Review for Final (a little...)
 - Th. 5/6 Final Exam
 - 3 P.M.
 - F 5/7 Final Exam (++)
9 A.M.

Comment: Hw 8 is hard... /
NP hard! /

A Theory of Computation

\

- While we have introduced many problems with polynomial-time algorithms...
 - We haven't formally defined what this means
 - And not all problems enjoy fast computation
- Given a set I of problem instances, and a set S of problem solutions, an abstract problem is a binary relationship in $I \times S$. That is, a set of pairs (i,s)
 - The problem shortest path associates each instance of a graph and an origin-destination with a sequence of vertices which connect the origin and destination
- **Decision** problems have a yes or no solution. Abstract decision problem is a function which maps problem instances I into {yes, no}.
 - e.g. Is the shortest path in this graph between nodes 0 and nodes 30 have length > 10?

P and NP

- A decision problem belongs to the class **P** if there is an algorithm solving the problem with a running time on a deterministic machine that is polynomial in the input size.
- A decision problem belongs to the class **NP** (non-deterministic polynomial) if:
 - Any solution y leading to 'yes' can be encoded in polynomial space with respect to the size of the input x .
 - Checking whether a given solution leads to 'yes' can be done in polynomial time with respect to the size of x and y
 - problem is solvable in polynomial time in a non-deterministic machine
 - Can explore all possible solutions in parallel
 - Oracle selects best possible solution to check

Slightly more formal:

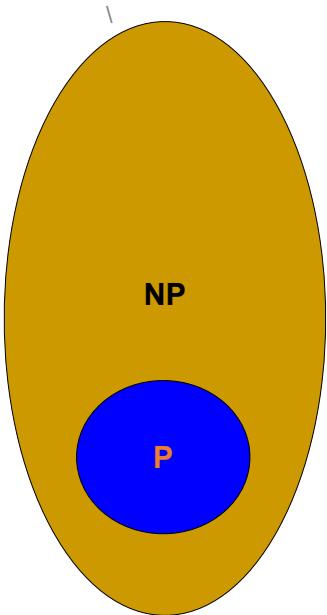
- Problem Q belongs to the class **NP**, if there exists a polynomial-time 2-argument algorithm A , such that:
 - For each instance i , i is a yes-instance to Q , if and only if, there is a polynomial-size certificate c for which $A(i,c) = \text{true}$.

Polynomially Reducible

- **Definition:** Problem A is polynomially reducible to problem B if there exists an algorithm for solving problem A in polynomial time if we could solve arbitrary instances of problem B at unit cost
 - Written as $A \leq_P B$
 - If $A \leq_P B$, and $B \leq_P A$, then we write $A =_P B$
- If $A \in \mathbf{P}$ and $B \leq_P A$, then $B \in \mathbf{P}$
 - e.g. Shortest path problem is in \mathbf{P} . Assignment problem is polynomially reducible to shortest path problem. Then Assignment problem is in \mathbf{P} .

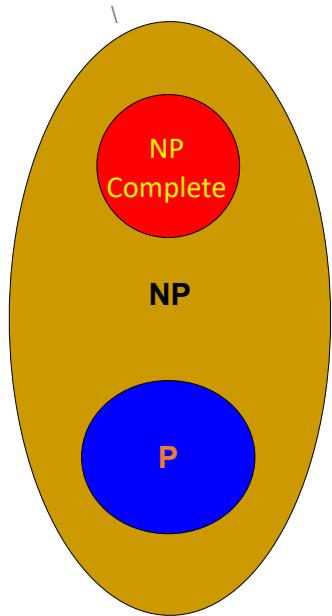
NP Complete

- We know class **P** is a subset of class **NP**
 - TSP is not known to be in class **P** but is in **NP**
 - It is a conjecture that **P** \neq **NP**, but it has not been proven
 - We do know **P** \subset **NP**
- **Definition:** a problem A is NP-complete, if $A \in \mathbf{NP}$ and every problem $B \in \mathbf{NP}$ can be polynomially reduced to A . That is,
$$B \leq_P A$$
- Do such problems exist? If so, **and** one of them was in class **P**, then every problem in class **NP** could be solved in polynomial time



Existence of NP-complete Problems

- Steven Cook and Leonid Levin (BU) proved in parallel that there exists an **NP** complete problem
 - Levin joined BU in 1980s
- Specifically: Boolean satisfiability (SAT)
 - Given a Boolean formula, is there an assignment of True and False to its variables so that the formula evaluates to True?
 - e.g. Can we find values of p, q, r, s so formula below is true?
$$(p \vee q \vee r) \wedge (p \vee \neg q \wedge s) \wedge (\neg p) \wedge (\neg q \wedge s) \wedge (\neg s)$$
- Cook-Levin Theorem
 - If a polynomial-time deterministic algorithm can solve this problem, then polynomial-time deterministic algorithms can solve all NP problems



How Do We Find Other NP-Complete Problems

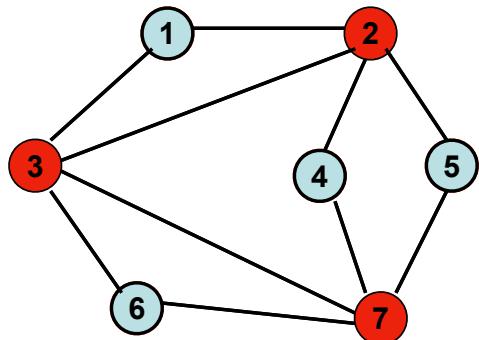
- Reduction! We want to find if problem Y in **NP** is NP-complete
- If $\text{SAT} \leq_P Y$, then Problem Y is NP-Complete! SAT is no easier than Y
- Note: to prove a problem A is in class **P**, we need to find a problem B in class **P** so that $A \leq_P B$
- To prove a problem A is NP-complete class **P**, find an NP-complete problem so that $B \leq_P A$
- Note the following: it is easy to show that $\text{SAT} =_P 3\text{-SAT}$ using standard logic transformations

NP-complete problem 1: Clique

- Clique: Given a graph G and a number k , decide whether G has a clique of size $\geq k$.
 - Clique: a complete subgraph.
- Fact: Clique is in NP.
- Theorem: If one can solve Clique in polynomial time, then one can also solve 3-SAT in polynomial time.
 - So Clique is at least as hard as 3-SAT.
- Corollary: Clique is NP-complete.

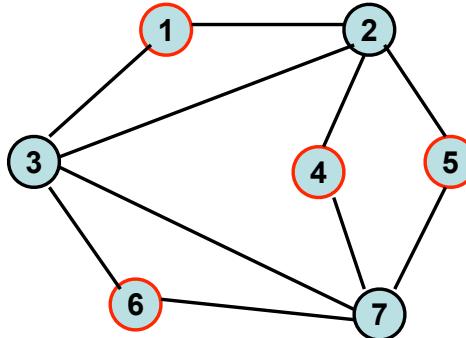
NP-complete problem 2: Vertex Cover

- **Vertex Cover**: Given a graph G and a number k , decide whether G has a **vertex cover** of size $\leq k$
 - V' is a vertex cover if all edges in G are “touched” by vertices from V'
- Vertex Cover is in NP
 - Given a candidate subset $S \subseteq V$, it is easy to check whether $|S| \leq k$ and S touches all edges in E



Another NP-Complete Problem: Independent Set

- **Independent Set**: Decide whether a given graph has an independent set of size at least k
- The above argument shows that the **Independent Set** problem is also NP-Complete because $\text{Clique} \leq_P \text{Independent set}$



NP-complete problem 3: Integer Programming (IP)

- Any 3-SAT formula can be expressed by integer programming.
- Consider a conjunctive clause, for example,
 - $(\bar{x}_1 \vee x_2 \vee x_3, x_1, x_2, x_3 \in \{0,1\})$
 - This is equivalent to $(1 - x_1) + x_2 + x_3 \geq 1, x_1, x_2, x_3 \in \{0,1\}$
- Multiple conjunctive clauses:
 - Translate to multiple integer programming constraints which must be satisfied simultaneously
 - Finding a feasible solution to the set of integer programming inequalities would yield a feasible solution to 3-SAT

Integer Programming example

- 3-SAT formula

$$\rightarrow (\bar{x}_1 \vee x_2 \vee x_3) \wedge (\bar{x}_2 \vee x_4 \vee \bar{x}_5) \wedge (x_1 \vee x_3 \vee x_5) \wedge (x_3 \vee \bar{x}_4 \vee x_5)$$

- Integer program inequalities

$$(1 - x_1) + x_2 + x_3 \geq 1,$$

$$(1 - x_2) + x_4 + (1 - x_5) \geq 1,$$

$$x_1 + x_3 + x_5 \geq 1,$$

$$x_3 + (1 - x_4) + x_5 \geq 1,$$

$$x_1, x_2, x_3, x_4, x_5 \in \{0,1\}$$

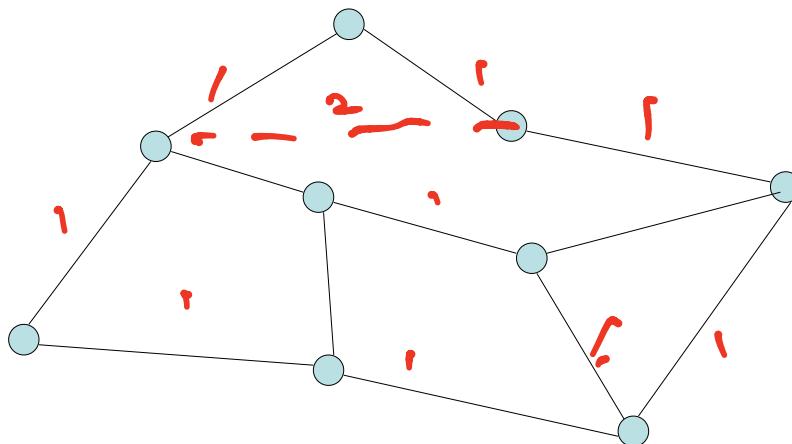
- So if one can solve IP efficiently, then one can also solve 3-SAT efficiently

- $3\text{-SAT} \leq_P \text{Integer Programming}$

$\rightarrow \text{NP-Complete}$

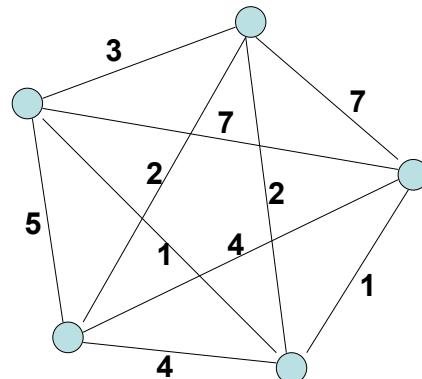
Hamiltonian Circuit Problem

- Hamiltonian Circuit – does there exist a simple cycle including all the vertices of the graph?
 - Can show Vertex Cover \leq_P Hamiltonian Circuit (very complicated construction)



Traveling Salesman Problem

- Given a complete graph with edge weights, determine the shortest tour that includes all of the vertices (visit each vertex exactly once, and get back to the starting point)
 - Already proved Hamiltonian Circuit \leq_P TSP ✓



Find the minimum cost tour

Subset Sums

The subset sum problem

- Given n natural numbers w_1, \dots, w_n and an integer W , is there a subset that adds up to exactly W ?

→ Ex. $\{ 215, 215, 275, 275, 355, 355, 420, 420, 580, 580, 655, 655 \}$, $W = \underline{1505}$

Yes. $215 + 355 + 355 + 580 = 1505.$

- Subset sum is in NP
 - Easy to verify whether a subset sums up to W ✓
- Important aspect: numbers are input in binary
 - Reduction must be polynomial in number of bits of input
 - Want to show $3\text{-SAT} \leq_P \text{Subset Sum}$

Subset Sum via Dynamic Programming

- Break up into smaller problems, build up to full problem
- $F(k, s)$ = True if there exists a subset of w_1, \dots, w_k that adds up to s , else false
- Initialization: $F(k, 0) = \text{True}$, $k = 0, \dots, n$; $F(0, s) = \text{False}$, $s > 0$
- Dynamic Programming recursion:

$$F(k, s) = \begin{cases} F(k-1, s) & \text{if } w_k > s \\ F(k-1, s) \vee F(k-1, s - w_k) & \text{if } w_k \leq s \end{cases}$$

- Can fill table to get solution: $O(n \times W) \rightarrow$ *Total value S.*
- But, this is not polynomial: W is exponential in the size of the binary description of W in bits
 - Requires exponential space and time

SAT Reduction to Subset Sum

- Given 3-SAT instance with n variables and k clauses, form $2n + 2k$ decimal integers, each having $n + k$ digits as follows
 - For each variable x_i , construct numbers t_i and f_i of $n + m$ digits: the i -th digit of t_i and f_i is equal to 1, other digits from 1 to n are 0
 - For $n+1 \leq j \leq n+m$, the j -th digit of t_i is equal to 1 if x_i is in clause c_{j-n}
 - For $n+1 \leq j \leq n+m$, the j -th digit of f_i is equal to 1 if \bar{x}_i is in clause c_{j-n}
 - All other digits of t_i and f_i are 0
- Example: $(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3)$

Number	..	i			..	j			..
	..	1	2	3	..	1	2	3	4
t_1		1	0	0		1	0	0	1
f_1		1	0	0		0	1	1	0
t_2		0	1	0		1	0	1	0
f_2		0	1	0		0	1	0	1
t_3		0	0	1		1	1	0	1
f_3		0	0	1		0	0	1	0

$$(x_1 \vee x_2 \vee x_3) = 1$$

$(\bar{x}_1 \vee \bar{x}_2 \vee x_3)$

$x_i \rightarrow$ digits ($n + m$) $n = \# x_i$
 $\rightarrow 0 + 1$ $m = \# \text{classes}$

SAT Reduction to Subset Sum - 2

- Given 3-SAT instance with n variables and k clauses, form $2n + 2k$ decimal integers, each having $n + k$ digits as follows
 - For each clause c_j , construct numbers z_j and y_j of $n + m$ digits: The $(n+j)$ -th digit of z_j and y_j is equal to 1, all other digits of z_j and y_j are 0
 - Example: $(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3)$
 - Finally, construct a sum number s of $n + m$ digits:
For $1 \leq j \leq n$, the j -th digit of s is equal to 1
For $n + 1 \leq j \leq n + m$, the j -th digit of s is equal to 3
 - e.g. 1113333
 -

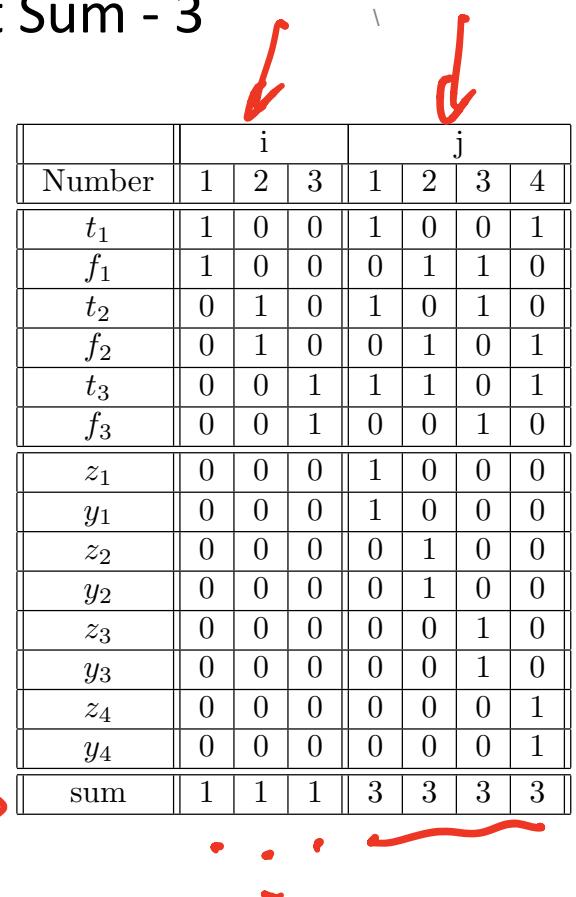
$$x_1 \vee x_2 \vee x_3$$

Find subset that sums to

Number	i		j					
	1	2	3	1	2	3	4	
t_1	1	0	0	1	0	0	1	
f_1	1	0	0	0	1	1	0	
t_2	0	1	0	1	0	1	0	
f_2	0	1	0	0	1	0	1	
t_3	0	0	1	1	1	0	1	
f_3	0	0	1	0	0	1	0	
z_1	0	0	0	1	0	0	0	
y_1	0	0	0	1	0	0	0	
z_2	0	0	0	0	1	0	0	
y_2	0	0	0	0	1	0	0	
z_3	0	0	0	0	0	1	0	
y_3	0	0	0	0	0	0	1	
z_4	0	0	0	0	0	0	1	
y_4	0	0	0	0	0	0	1	
sum	1	1	1	3	3	3	3	

SAT Reduction to Subset Sum - 3

- If we find subset of rows that sum to the total sum, then we can convert the rows to Boolean variables $x_i, i = 1, \dots, n$ such that every clause $c_j, j = 1, \dots, m$ is true
 - x_i true if t_i is in subset; otherwise, f_i must be in subset so x_i is false
 - Since the j columns sum to 3, at least one of the literals in c_j must be true because there are at most 2 in z_j, y_j
- If there exists a true assignment, then there is a set of rows that sums to sum
 - For clause c_j , add rows y_j if two or less literals are true, add z_j if only one literal is true
- Result:** 3-SAT \leq_P Subset Sum ✓



	i			j			
Number	1	2	3	1	2	3	4
t_1	1	0	0	1	0	0	1
f_1	1	0	0	0	1	1	0
t_2	0	1	0	1	0	1	0
f_2	0	1	0	0	1	0	1
t_3	0	0	1	1	1	0	1
f_3	0	0	1	0	0	1	0
z_1	0	0	0	1	0	0	0
y_1	0	0	0	1	0	0	0
z_2	0	0	0	0	1	0	0
y_2	0	0	0	0	1	0	0
z_3	0	0	0	0	0	1	0
y_3	0	0	0	0	0	1	0
z_4	0	0	0	0	0	0	1
y_4	0	0	0	0	0	0	1
sum	1	1	1	3	3	3	3

$$\rightarrow \underbrace{(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3)}_{c_1 \quad c_2}$$

$$\begin{array}{r}
 \rightarrow t_1 \ 1\ 0\ 0 \quad | \quad c_1 \quad 0 \\
 F_1 \ 1\ 0\ 0 \quad | \quad 0 \quad 1 \\
 t_2 \ 0\ 1\ 0 \quad | \quad 1 \quad 0 \\
 \rightarrow f_2 \ 0\ 1\ 0 \quad | \quad 0 \quad 1 \rightarrow \\
 t_3 \ 0\ 0\ 1 \quad | \quad 1 \quad 0 \\
 \rightarrow f_3 \ 0\ 0\ 1 \quad | \quad 0 \quad 1 \\
 \hline
 \rightarrow u_1 \ 0\ 0\ 0 \quad | \quad 1 \quad 0 \\
 \rightarrow z_1 \ 0\ 0\ 0 \quad | \quad 1 \quad 0 \\
 \rightarrow u_2 \ 0\ 0\ 0 \quad | \quad 0 \quad 1 \rightarrow \\
 z_2 \ 0\ 0\ 0 \quad | \quad 0 \quad 1 \\
 \hline
 \end{array}$$

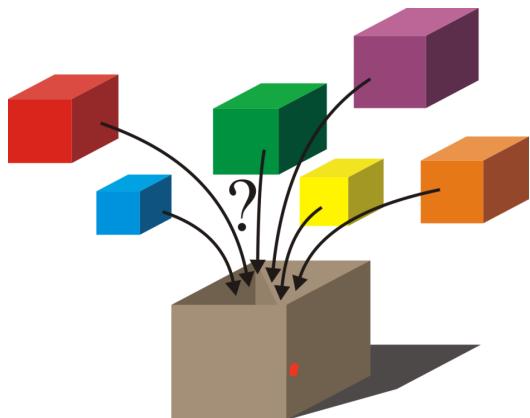
$\underbrace{1 \ 1 \ 1}_{\sim} \quad \underbrace{3 \ 3}_{\sim}.$

$$\begin{array}{ll}
 x_1 = 1 & x_3 = 0 \\
 x_2 = 0 &
 \end{array}$$

Knapsack problem

The knapsack problem

- Suppose a container can hold a maximum weight W
- Given a number of items with specified weights w_i and values v_i , is there some combination of items that has a total value greater than or equal to V without exceeding the maximum weight W ?



Knapsack Decision Problem is NP-Complete

- Show that $\text{Subset Sum} \leq_P \text{Knapsack}$ ✓
- Given subset sum problem instance with w_1, \dots, w_n and total weight W , form following Knapsack problem:
 - Let weight of object i be w_i
 - Let value v_i of object i be $v_i = w_i$
- Solve the following knapsack decision problem: Is there a subset of items S such that the value is greater than or equal to W , while the total weight is less than or equal to W ?

• $\sum_{i \in S} v_i = \sum_{i \in S} w_i \geq W; \quad \sum_{i \in S} w_i \leq W;$

- If answer is yes, then S is a subset with sum W . If no solution can be found for knapsack, then subset sum does not have a subset with sum W

NP-Hard vs NP-Complete Problems

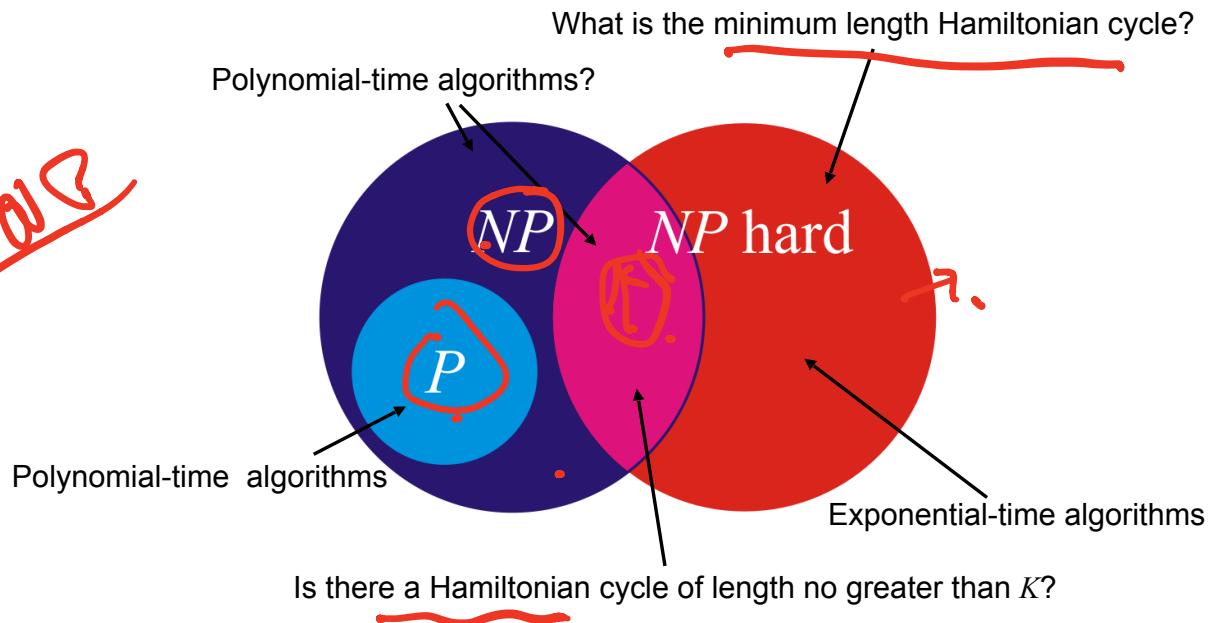
- The optimization version of NP-Complete problem is referred to as an **NP-Hard** problem
 - NP-complete problems are decision problems
 - E.g. finding the minimum length Hamiltonian circuit is NP-hard; finding whether there exists a Hamiltonian circuit with length less than L is NP-complete ...
 - Finding the maximum value that fits in a knapsack is NP-hard; finding whether we can fit value of at least V is NP-Complete
- Formally, a problem X is NP-hard, if there is an NP-complete problem Y, such that Y is reducible to X in polynomial time
 - But X does not have to be in NP, and does not have to be a decision problem

NP-Complete $Y \leq_p X$
 $X \notin NP$ $\rightarrow X$ is NP-hard

NP and NP Hard

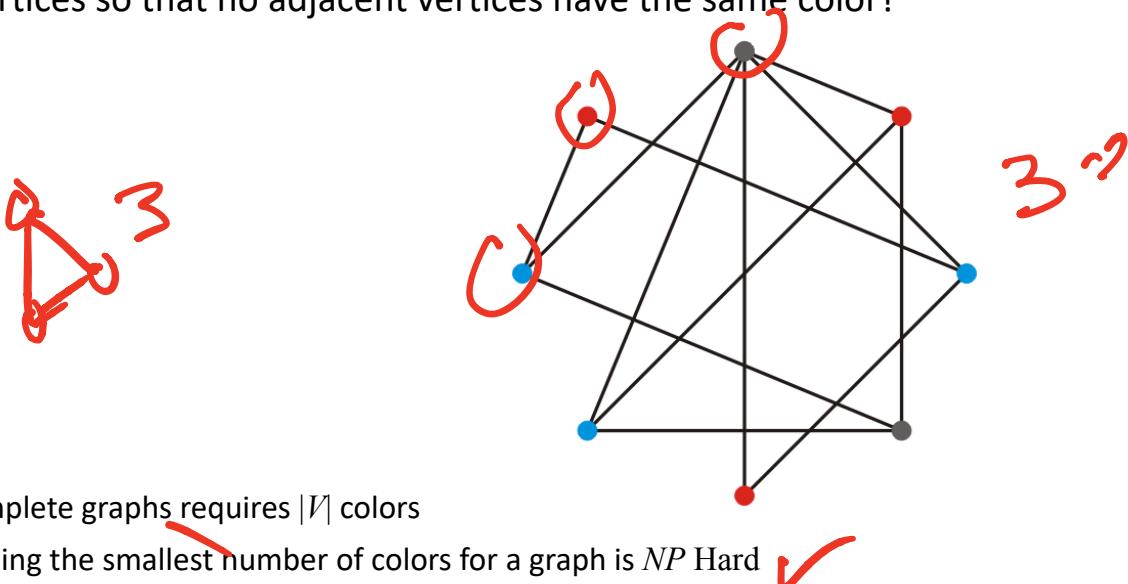
Consider this Venn diagram where $P \subseteq NP$

~~P > NP~~



NP-Hard: Graph colorings

- Given a graph and n colors, is it possible to assign one of the colors to each of the vertices so that no adjacent vertices have the same color?



NP-Complete: Graph isomorphisms

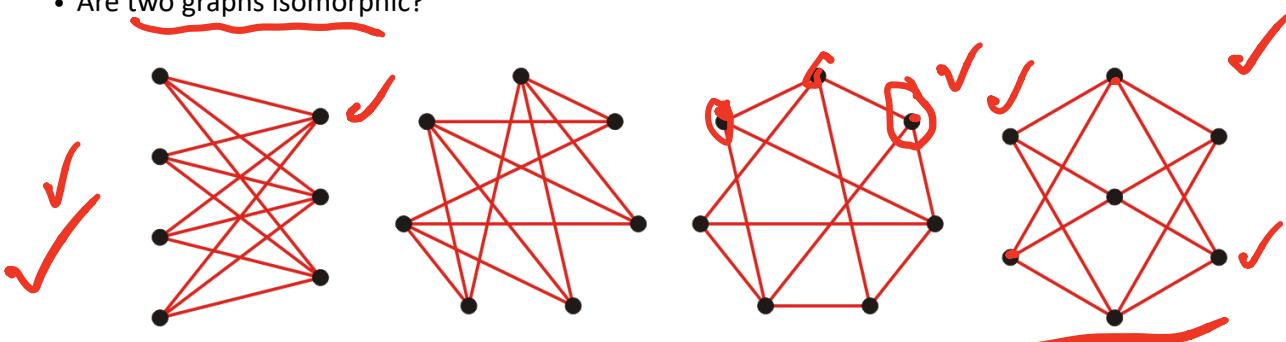
Recall the sub-graph isomorphism problem



- Given two graphs, is one graph isomorphic to a sub-graph of the other?

A possibly easier question is:

- Are two graphs isomorphic?

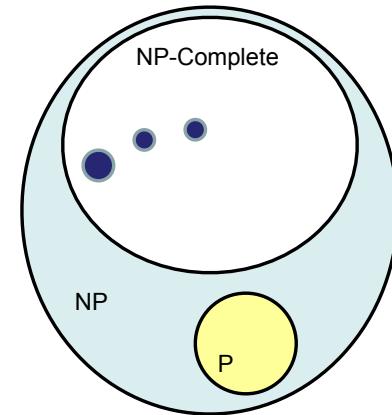


- So far, the best algorithm is $2^{O(\sqrt{n \ln(n)})}$



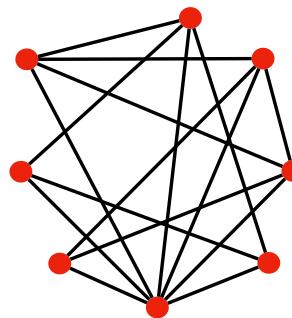
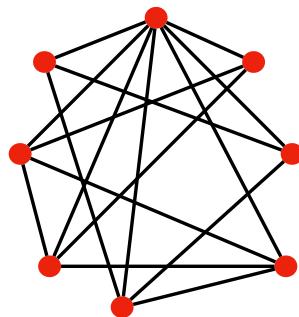
Populating the NP-Completeness Universe

- Circuit Sat $<_P$ 3-SAT
- 3-SAT $<_P$ Independent Set
- 3-SAT $<_P$ Vertex Cover
- Independent Set $<_P$ Clique
- 3-SAT $<_P$ Hamiltonian Circuit
- Hamiltonian Circuit $<_P$ Traveling Salesman
- 3-SAT $<_P$ Integer Linear Programming
- 3-SAT $<_P$ Graph Coloring
- 3-SAT $<_P$ Subset Sum
- Subset Sum $<_P$ Scheduling with Release times and deadlines



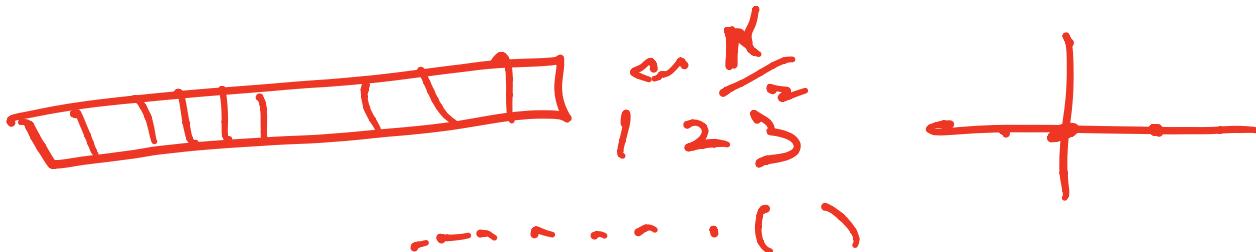
Problems in NP not in P or NP-Complete ✓

- Graph Isomorphism
 - We don't have an algorithm to show problem is in **P**, and no reduction of an np-complete problem to it



Problems in NP not in P or NP-Complete - 2

- These problems are often referred to as **NP-intermediate**
 - Circuit Minimization is the problem of obtaining the smallest logic circuit that represents a given Boolean function or truth table
 - Turnpike problem: Given $\frac{n(n - 1)}{2}$ distances, find n points on a line
 - Cutting-stock problem: cutting standard-sized pieces of stock material, such as paper rolls or sheet metal, into pieces of specified sizes while minimizing material wasted
 - Prime factorization: decomposition of a composite number into a product of prime factors
 - Numbers in boxes: If you put the numbers $1, 2, \dots, n$ into boxes, and If x, y, z are in a box then $x + y \neq z$. What is the smallest number of boxes you will need?



Exercise

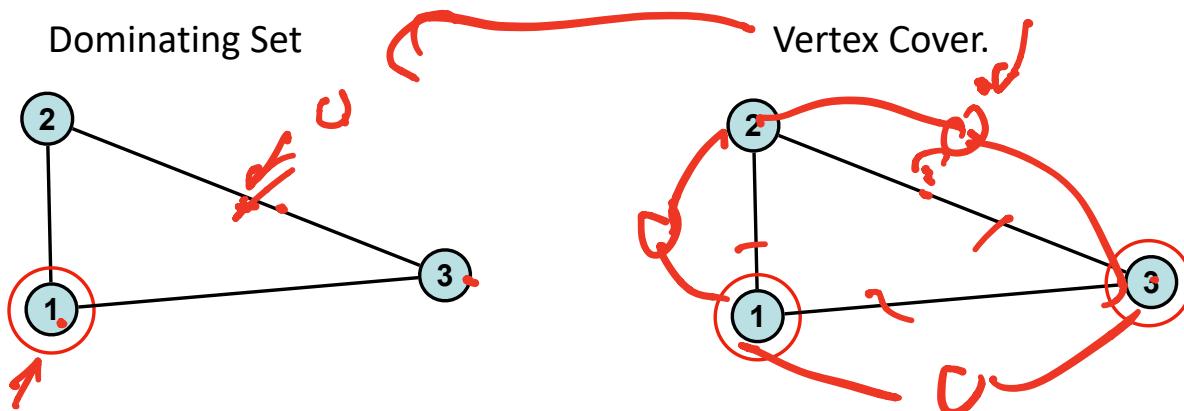
- Dominating Set problem: Given a graph $G = (V, E)$ and an integer K , decide whether G contains a dominating set of size at most K .
 - Dominating set: $S \subseteq V$ such that $\forall v \in V$, either $v \in S$ or v has a neighbor in S .
 - Namely, S and S 's neighbors cover the entire V .
- Prove that Dominating Set is NP-complete.
- Hint: Reduction from Vertex Cover.

$S, N(S)$

$S \cup N(S) = \checkmark$

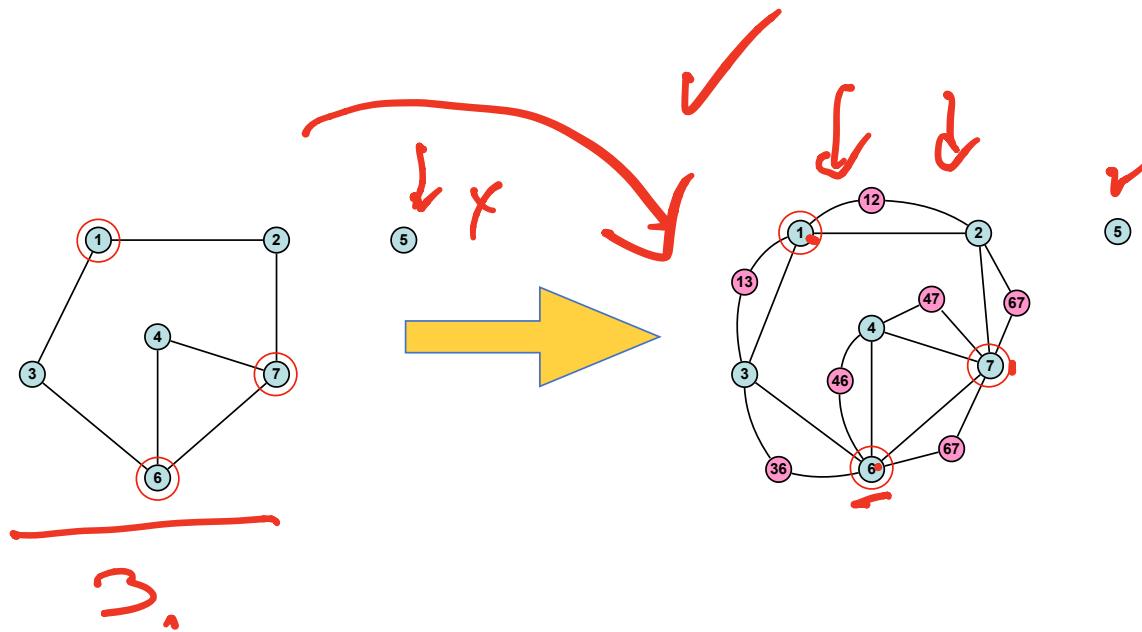
Vertex Cover vs Dominating Set

- Every vertex cover is a dominating set (if graph is connected)
- But, size of minimal vertex cover is different from size of minimal dominating set! Minimal dominating set can be smaller...



Reduction

- Add an extra vertex for every edge
 - Minimum dominating set = $k \iff$ minimum vertex cover is $k - n_I$, where n_I is number of isolated vertices



Further Concepts in Complexity

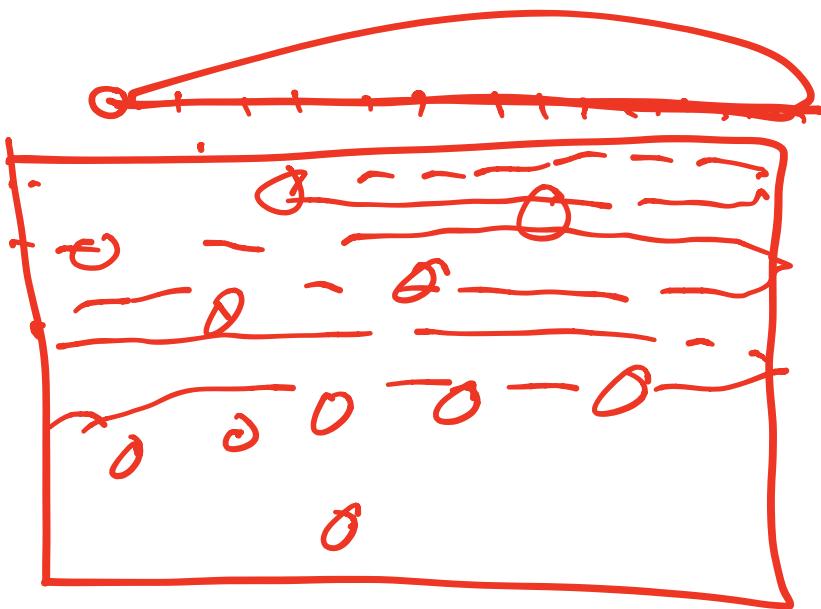
- **Pseudopolynomial** complexity: If K is the size of the largest number, and n is the size of the input, then the worst case complexity is polynomial in n and K . (e.g. Integer knapsack)
- **Strongly Polynomial** complexity: worst case complexity is polynomial in input size, independent of largest value of number in input.
- **Strongly NP-complete problems**: If one restricts the size of the largest number in the problem to K , where K is a polynomial in the input size n , then the problem is still NP-complete.
 - e.g. Clique
- **PSPACE**: solved using an amount of storage which is polynomial in the size of the problem.
 - It is clear that P is in PSPACE; more importantly, NP is also in PSPACE
 - PSPACE-complete: Super Mario Brothers, Reversi,

Visual PC PSPACE Complete Games

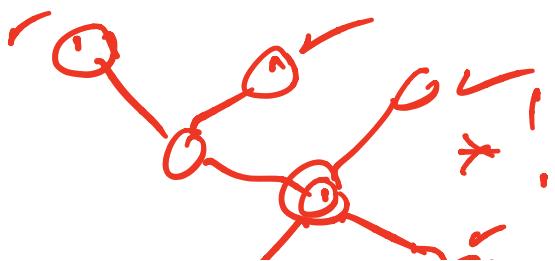
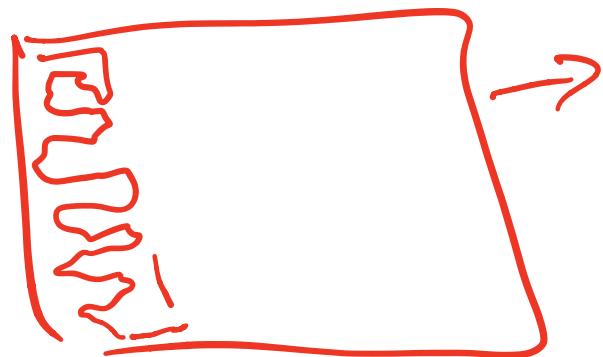
What to do if a problem is NP-complete?

- Solve on special cases where polynomial algorithms exists
 - e.g., TSP in one dimension, independent set on tree graphs, ...
- Heuristics and approximations
 - Would like some guarantee that the performance is good
 - Algorithms that are fast on average
- ✗ • Good exponential-time algorithms
 - Exact exponential-time algorithms is one of many options
 - Should be better than trivial algorithm: enumeration of all possibilities
 - e.g. TSP on dense graphs: $(n-1)!$ possible tours, but can solve in much less time

$\rightarrow \underline{1.5}^n << \underline{2^n} << \underline{n^n}$
 $\underline{2^n} << \underline{n!}$



Space-filling Curve



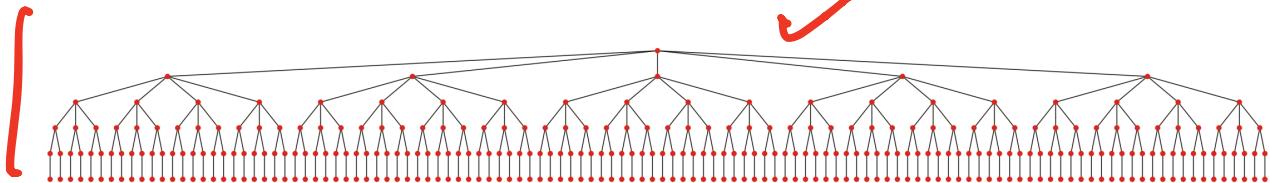
Deterministic polynomial-time algorithms

Consider the traveling salesman problem:

In a complete weighed graph, what is the least weight Hamiltonian cycle?

A deterministic algorithm to find a solution is to do tree traversal

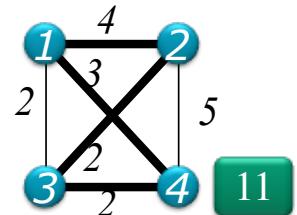
- This tries every path
- The run time is $\Theta(|V|!)$
- The Held-Karp or Bellman's algorithm runs in $\Theta(|V|^2 2^{|V|})$ time
 - It uses dynamic programming



Held-Karp Algorithm for TSP

\

- Take one starting vertex s arbitrarily. 
- For set of vertices $S \subset V - \{s\}$, and vertex $v \in S$, let $B(S, v) =$ minimum length of a path, that starts in s , visits all vertices in S (and no other vertices) and ends in v
- Dynamic Programming Recursion
-  $B(\{v\}, v) = d(s, v)$ (Initialization) 
- For $j = 2$ to $|V| - 1$:
 - For all sets S , $|S| = j$, for all $v \in S$:
 -  $B(S, v) = \min_{w \in S} B(S - \{v\}, w) + d(v, w)$ 
 - Return $\min_{v \in V, v \neq s} [B(V - \{s\}, v) + d(v, s)]$
- Complexity: $O(n^2 2^n)$. Must compute $B(S, v)$ for all subsets $S \subset V, v \in V$
 - $O(n 2^n)$ values, each of which takes $O(n)$ to compute: Much better than $n!$



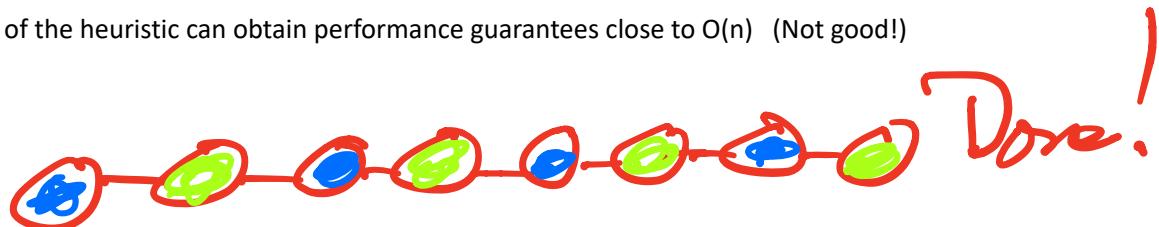
Start    

..(v) x -5'

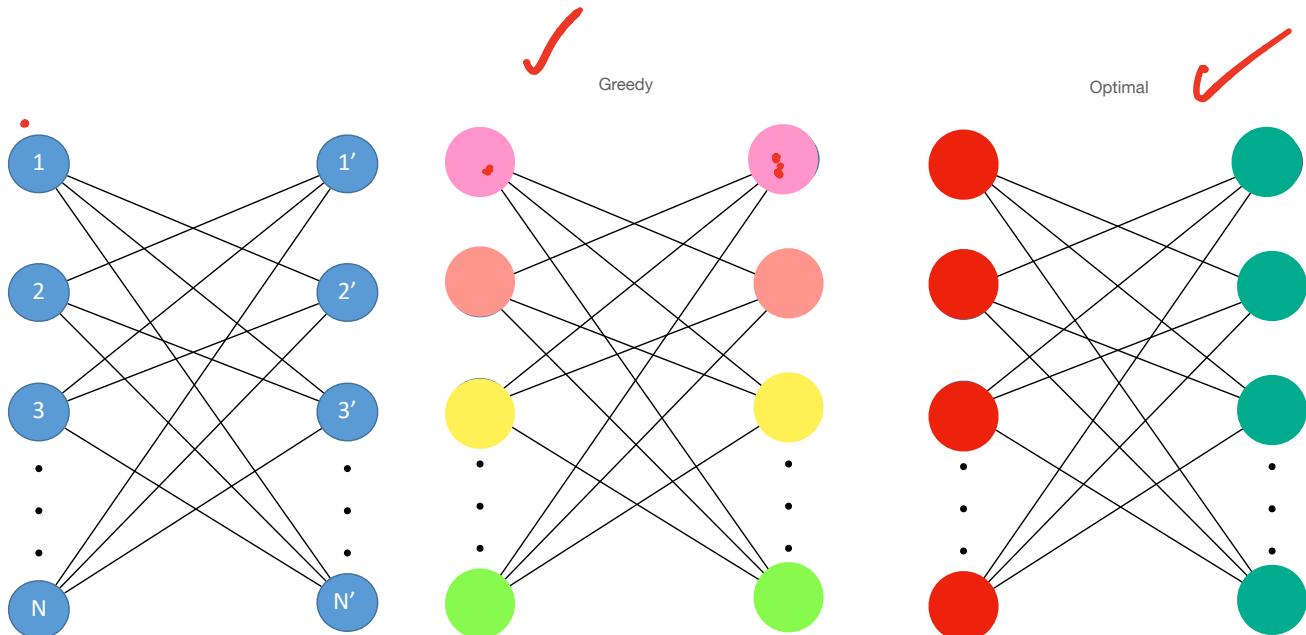
v  $S = \{v\}$

Approximate Algorithms

- Objective: Find approximate algorithms for NP-hard problems with performance guarantees
 - Solution of approximate algorithm is within a factor of optimal solution
- Example: Graph coloring: Find the minimum number of colors for nodes so that no two neighbors have the same color
 - Greedy heuristic: Choose a color and an arbitrary starting node. Consider all other nodes: if a node can be painted with this color, do so; otherwise leave nodes unpainted.
 - Now, choose a second color. Loop over all unpainted nodes and paint them with this color if possible.
 - Repeat until done.
- Can show: There is an ordering of nodes for which this is the optimal thing to do: e.g. a line graph
- Can also show: There are graphs which can be colored with just k colors, for which the heuristic will use c colors, and c/k can be made as large as possible
 - No performance guarantee!
 - Variations of the heuristic can obtain performance guarantees close to $O(n)$ (Not good!)



Bad example for heuristic



$\rightarrow N$ Colors
 $\rightarrow 2N$ Vertices

$\frac{N}{2} \rightarrow \infty$
 $N \rightarrow \infty$

Guaranteed Performance: Integer Knapsack

- Greedy: Choose j in descending order of V_j/w_j , until you exceed capacity.

Let j be last item chosen that exceeds capacity. Return $\arg \max[\sum_{k=1}^{j-1} V_k, V_j]$

- Theorem: Value of Greedy algorithm is no worse than $1/2$ value of optimal algorithm

- P = value of sequential greedy knapsack before overflow
- V_j = cost of item that overflows
- $P + V_j$ is upper bound on optimal Knapsack. (by fractional argument).

- Bad example for Greedy:

- 2 tasks. Task 1: Cap ε , value 2ε ; Task2: Cap C , value C

$$\frac{P+V_j}{\max(P, V_j)} \geq \frac{\max(P, V_j)}{P+V_j} > \frac{1}{2}$$

Increase to $\underline{W} + \underline{w_j}$

1 off. + ~~(P + V)~~ $\frac{V_1}{w_1} = 2$

$$\sqrt{w_1 = \epsilon} \quad \frac{V_1}{w_1} = 2 \\ V_1 = 2\epsilon \quad \frac{V_2}{w_2} = 1$$

$$w_2 = C$$

$$V_2 = C$$

$$\boxed{W = C} *! \quad 2\epsilon \\ wC$$

$$\frac{C}{\epsilon} \rightarrow \infty ..!$$

$$\frac{C}{C} \rightarrow !$$

Guaranteed Performance: Vertex Cover

- Greedy Algorithm: $S \leftarrow \emptyset; E' \leftarrow E; M \leftarrow \emptyset$
 - While ($E' \neq \emptyset$)
Let $\{u, v\} \in E'$ be an arbitrary edge ✓
Add $\{u, v\}$ to $M: M = M \cup \{\{u, v\}\}$. Note that M matches pairs of vertices in 1-1 manner
Update vertex cover $S = S \cup \{u\} \cup \{v\}$
Delete from E' all edges incident to either u or v
 - Return S
- Let S^* be a minimum vertex cover. Then, greedy algorithm computes a vertex cover S with $|S| \leq 2|S^*|$. It is a 2-approximation algorithm

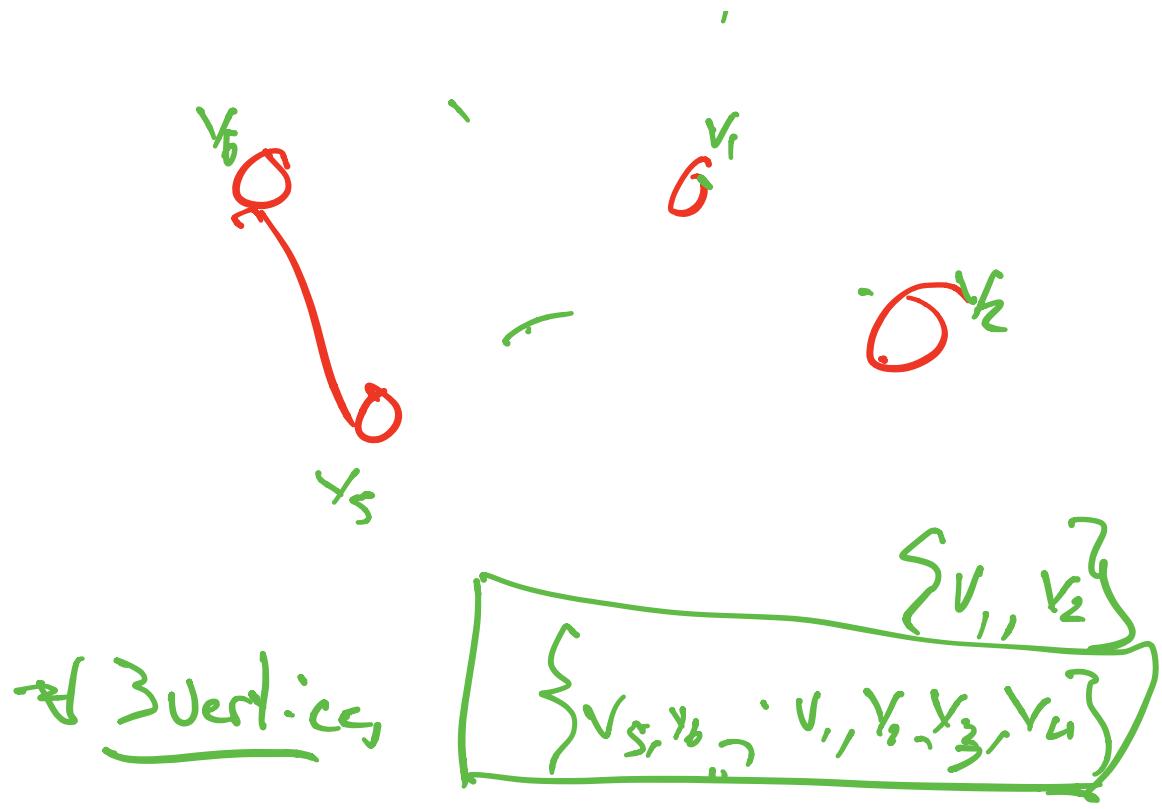
Pf.

- S is a vertex cover
- M is a matching, where no pair of edges share a vertex. Hence

$$|S^*| \geq |M|, \text{ and } |S| = 2|M|, \text{ so } |S| \leq 2|S^*|$$







Guaranteed Performance: Vertex Cover

- Theorem. [Dinur–Safra 2004] If $P \neq NP$, then no ρ -approximation for VERTEX-COVER for any $\rho < 1.3606$.

•

TSP Approximations

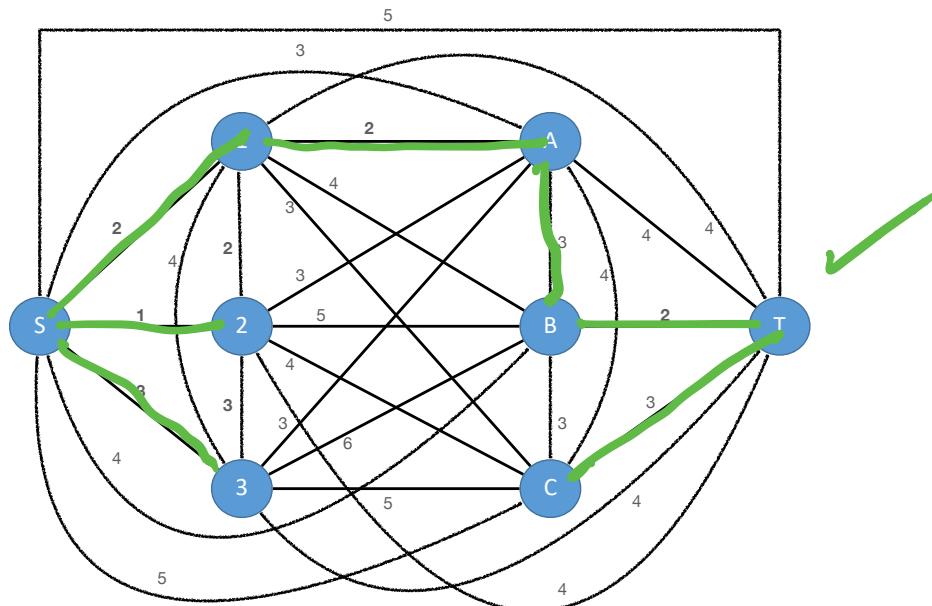
- Dynamic programming (Held-Karp, Bellman): $O(n^2 2^n)$ exact algorithm
- Restrict to Metric (or Euclidean) TSP: assume distances $d(i, j)$ satisfy the following
 - Symmetry: $d(i, j) = d(j, i)$ ✓
 - Triangle inequality: $d(i, j) \leq d(i, k) + d(k, j)$ for all i, j, k
- Approximate algorithm
 - Find a minimum spanning tree $G = (V, T)$
 - Add a duplicate edge to each edge in the minimum spanning tree T , with same distance, resulting in augmented edges T'
 - Find an Euler tour in $G' = (V, T')$. Note one exists because degree of each vertex is even.
 - List the vertices in order visited by Euler tour; skip vertices that appear more than once

TSP Approximations

- Theorem: Approximate algorithm tour has distance no longer than twice the distance D^* of the optimal TSP tour
 - Proof: Let D = weight of minimum spanning tree, D' the weight of the approximate tour
Then, $D \leq D^*$, as the minimum spanning tree reaches every vertex, and has less constraints than the TSP tour
Also, note the length of the Euler tour is $2D$, as it travels every edge of the MST twice
From the metric property, cutting the corners by deleting the duplicate vertices cannot increase the length,
Hence $D \leq D^* \leq D' \leq 2D$

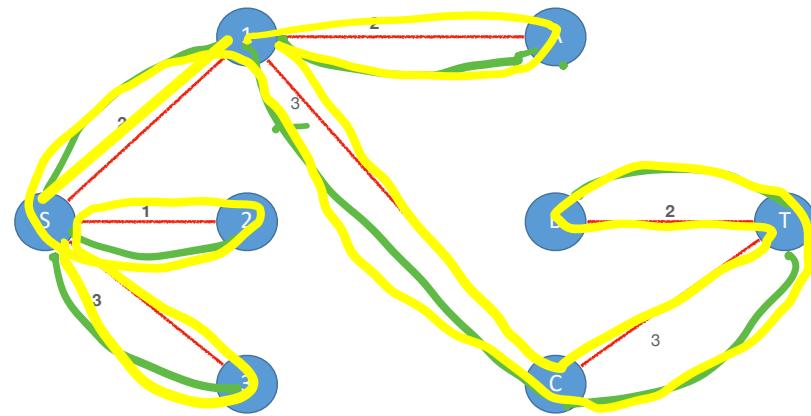
Example

- Find MST: Weight 16



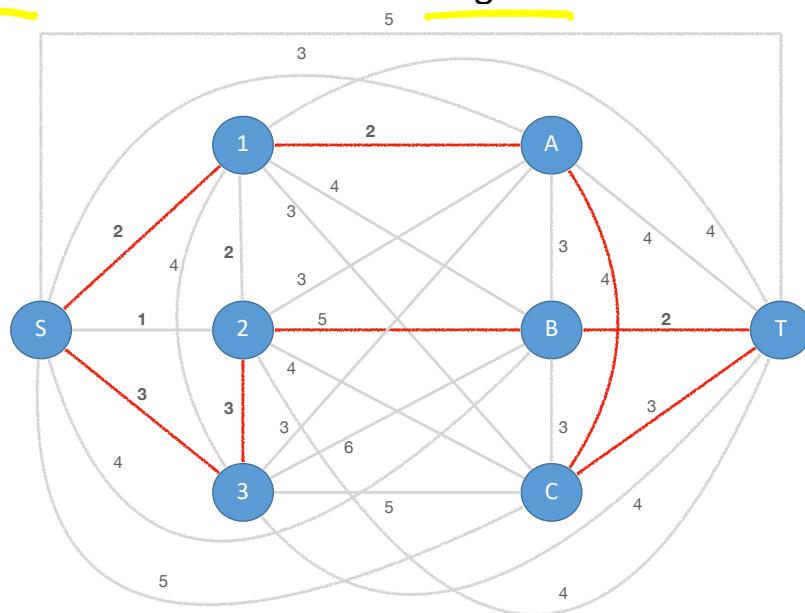
Example - 2

- Double edges



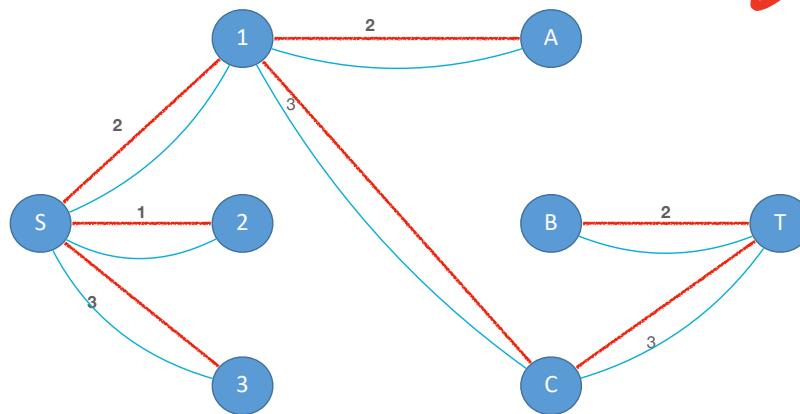
Example - 3

- Euler tour length: 32
- Cut corners: S-1-A-C-T-B-2-3-S. Length 24



Example - 3

- Find Euler Tour: S-1-A-1-C-T-B-T-C-1-S-2-S-3-S. Length 32



- Cut corners: S-1-A-C-T-B-2-3-S. Length