Satisfiability

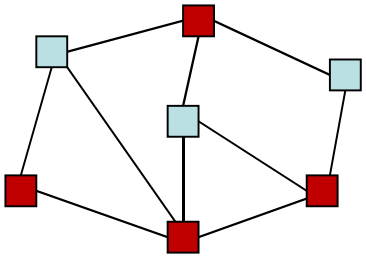
$$(\neg x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee x_4) \wedge (x_2 \vee \neg x_4 \vee \neg x_5)$$

Input: a logic formula of size **m** containing **n** variables

Output: An assignment of Boolean values to the variables in the formula such that the formula is true

Algorithm: Try every variable assignment

Vertex Cover:



Input: A graph (**V**,**E**) and a number **m**

Output: A subset **S** of **V** such that *for every edge* (**u**,**v**) in **E**, at least one of **u** or **v** is in **S** and **|S|=m** (if such an **S** exists)

Algorithm: Try every subset of vertices of size **m**

Traveling Salesman

Input: A *complete* *weighted* graph (**V**,**E**) and a number **m**

Output: A circuit that visits each vertex exactly once and has total cost < **m** if one exists

Algorithm: Try every path, stop if find cheap enough one

The Complexity Class NP

- *Definition*: NP is the set of all problems for which a given *candidate solution* can be *tested* in polynomial time

- Examples of problems in NP:
  - *Hamiltonian circuit:* Given a candidate path, can test in linear time if it is a Hamiltonian circuit
  - *Vertex Cover:* Given a subset of vertices, do they cover all edges?
  - *All problems that are in P (why?)*

Why do we call it "NP"?

- NP stands for *Nondeterministic Polynomial time*
  - Why "nondeterministic"? Corresponds to algorithms that can guess a solution (if it exists), the solution is then verified to be correct in polynomial time
  - Can also think of as allowing a special operation that allows the algorithm to magically guess the right choice at each branch point.
  - Nondeterministic algorithms don't exist – purely theoretical idea invented to understand how hard a problem could be

NP-completeness

- Set of problems in NP that (we are pretty sure) ***cannot*** be solved in polynomial time.
- These are thought of as the **hardest** problems in the class NP.
- **Interesting fact:** If any one NP-complete problem could be solved in polynomial time, then ***all*** NP-complete problems could be solved in polynomial time.
- Also: If any NP-complete problem is in P, then all of NP is in P

Travelling Salesman Problem (TSP)

- Your third task is basically TSP:
  - Given complete weighted graph G, integer k.
  - Is there a cycle that visits all vertices with cost <= k?
- One of the canonical problems.

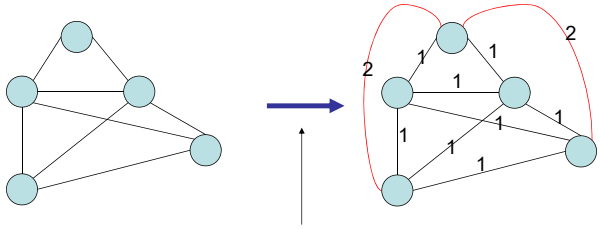
- Note difference from Hamiltonian cycle:
  - graph is complete
  - we care about weight.

Transforming Hamiltonian Cycle to TSP

- We can "reduce" Hamiltonian Cycle to TSP.
- Given graph G=(V, E):
  - Assign weight of 1 to each edge
  - Augment the graph with edges until it is a complete graph G'=(V, E')
  - Assign weights of 2 to the new edges
  - Let k = |V|.

Notes:
  - The transformation must take polynomial time
  - You reduce the known NP-complete problem into your problem (not the other way around)
  - In this case we are assuming Hamiltonian Cycle is our known NP-complete problem (in reality, both are known NP-complete)

Example



G

Input to Hamiltonian Circuit Problem

Polynomial time transformation

G'

Input to Traveling Salesman Problem

What do we do about it?

- Approximation Algorithm:
  - Can we get an efficient algorithm that guarantees something *close* to optimal? (e.g. Answer is guaranteed to be within 1.5x of Optimal, but solved in polynomial time).
- Restrictions:
  - Many hard problems are easy for restricted inputs (e.g. graph is always a tree, degree of vertices is always 3 or less).
- Heuristics:
  - Can we get something that seems to work well (good approximation/fast enough) *most* of the time? (e.g. In practice, n is small-ish)