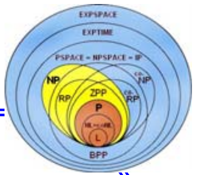
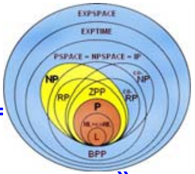


Session 16

- The Classes $\mathcal{P}$ and $\mathcal{NP}$
- An NP-Complete Problem



The Classes $\mathcal{P}$ and $\mathcal{NP}$

The Classes $\mathcal{P}$ and $\mathcal{NP}$

A Turing machine M is said to be of time complexity $T(n)$ if whenever M is given an input w of length n, M halts after making at most $T(n)$ moves, regardless of whether or not M accepts.

This definition applies to any function $T(n)$, such as $T(n) = 50n^2$ or $T(n) = 3^n + 5n^4$; we shall be interested predominantly in the case where $T(n)$ is a polynomial in n.

- A language L is in class $\mathcal{P}$ if there is some polynomial $T(n)$ such that $L = L(M)$ for some *deterministic* TM M of time complexity $T(n)$.

- A language L is in class $\mathcal{NP}$ if there is some polynomial $T(n)$ such that $L = L(M)$ for some *nondeterministic* TM M of time complexity $T(n)$.

☞ Since every deterministic TM is a nondeterministic TM that happens never to have a choice of moves, so we have $\mathcal{P} \subseteq \mathcal{NP}$.

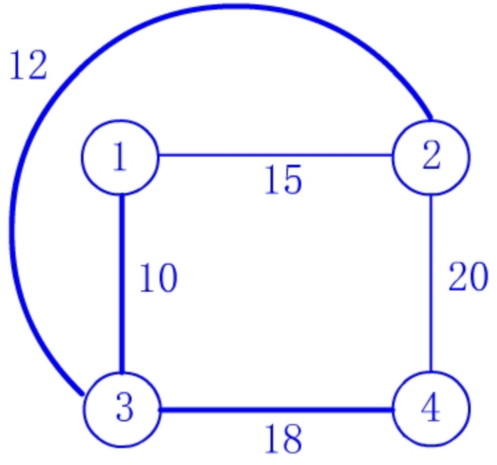
It appears that $\mathcal{NP}$ contains many problems (or languages) not in $\mathcal{P}$.

The intuitive reason is that a NTM running in polynomial time has the ability to guess an exponential number of possible solutions to a problem and check each one in polynomial time, in "parallel". However:

- It is one of the deepest open questions of Mathematics whether $\mathcal{P} = \mathcal{NP}$.

i.e., if in fact everything that can be done in polynomial time by a NTM can in fact be done by a DTM in polynomial time, perhaps with a higher-degree polynomial.

Example **The minimum-weight spanning tree** Finding a minimum-weight spanning tree (MWST) for a graph. For example



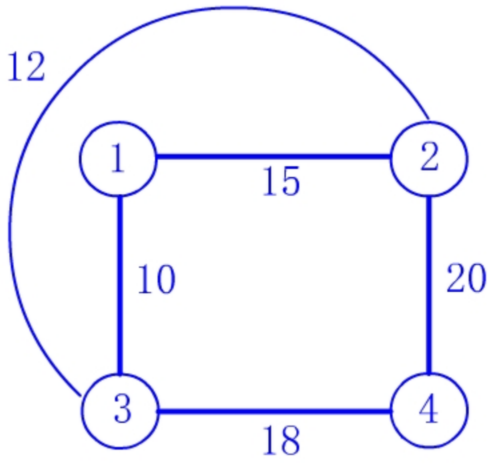
There is a well-known "greedy" algorithm, called Kruskal's Algorithm, for finding a MWST. It is possible to implement this algorithm (using a computer, not a Turing machine) on a graph with m nodes and e edges in time $O(m + e \log e)$.

We can translate Kruskal's algorithm to Turing machine:

- Change the MWST problem as a yes-no version, "does graph G have a MWST of total weight W or less".
- Encode suitably the problem elements such as nodes and edges.

Conclusion: The yes-no version of the MWST problem is in $\mathcal{P}$.

Example **The Traveling Salesman Problem (TSP)** Whether the graph has a "Hamilton circuit" of total weight at most W.



In m-node graphs, the number of distinct cycles grows as $O(m!)$, which is more than 2^{cm} for any constant c.

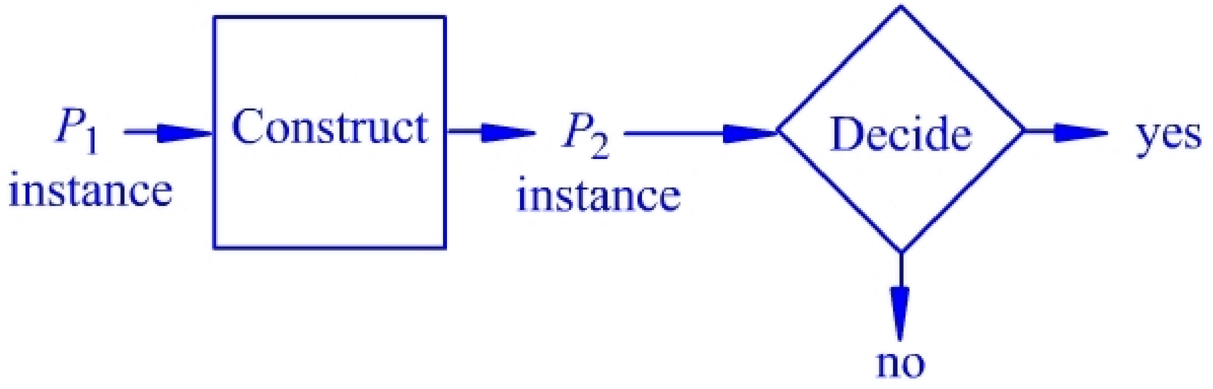
It appears that all ways to solve the TSP involve trying essentially all cycles and computing their total weight. If we had a nondeterministic computer, we could guess a permutation of the nodes, and compute the total weight for the cycle of nodes in that order.

On a multitape NTM, we can guess a permutation in $O(n^2)$ steps and check its total weight in a similar amount of time.

Conclusion: The TSP is in $\mathcal{NP}$.

Reduction

In the theory of undecidability and intractability, we often use a technique which is called the reduction suggested as follow:



For examples, suppose that we know P_1 is undecidable, and P_2 is a new problem that we would like to prove is undecidable as well. We can use such a strategy.

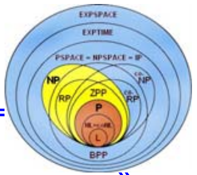
If we could construct such a reduction and solve problem P_2, then we could use its solution to solve problem P_1.

☞ Note that the direction of a reduction is important! The only way to prove a new problem P_2 to be undecidable is to reduce a known undecidable problem P_1 to P_2. Since what we prove is "if P_2 is decidable, then P_1 is decidable."

Similarly, the methodology for proving that a problem P_2 is not in $\mathcal{P}$ is the reduction of a problem P_1, which is known not to be in $\mathcal{P}$, to P_2. Since what we prove is "if P_2 is in $\mathcal{P}$, then so is P_1".

However, the mere existence of the algorithm labeled "Construct' is not sufficient to prove the desired statement! We need some restriction.

The correct restriction to place on the translation from P_1 to P_2 is that requires time that is polynomial in the length of its input. Thus, in the theory of intractability we shall use polynomial-time reductions only.



NP-Complete Problems

Let L be a language (problem). We say L is NP-complete if the following statements are true:

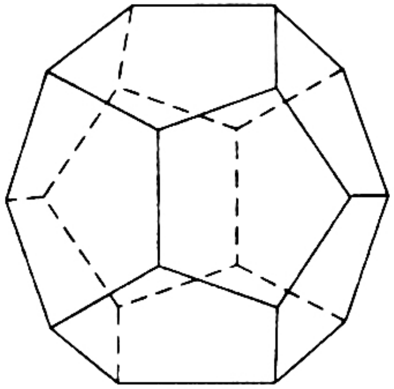
1. L is in $\mathcal{NP}$.
2. For every language L' in $\mathcal{NP}$ there is a polynomial-time reduction of L' to L.

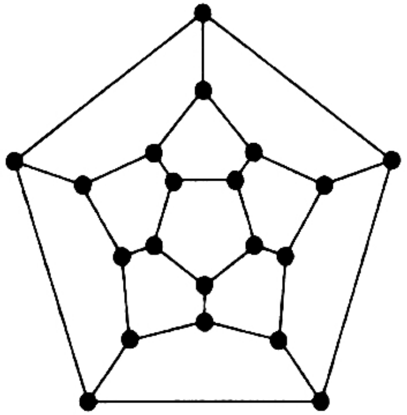
There is a vast collection of NP-complete problems. For example, the Hamilton-Circuit Problem is an NP-complete problem.

The terminology -Hamilton circuit- comes from Hamilton's puzzle consisted of a wooden dodecahedron [a polyhedron with 12 regular pentagons as faces], with a peg at each vertex of the dodecahedron, and string.

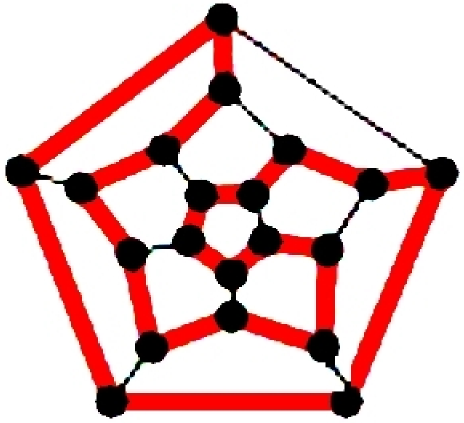
The 20 vertices of the dodecahedron were labeled with different cities in the world. The object of the puzzle was to start at a city and travel along the edges of the dodecahedron, visiting each of the other 19 cities exactly once, and end back at the first city.

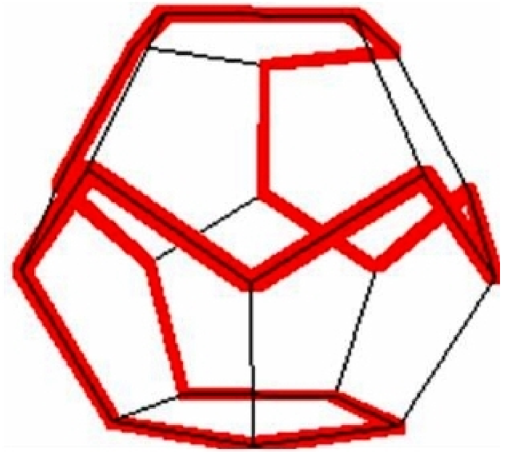
We consider the equivalent question: Is there a circuit in the graph that passes through each vertex exactly once, that is, a Hamilton circuit?

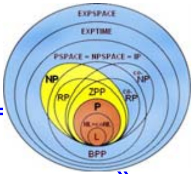




A solution of Hamilton's puzzle is shown as follows:







We have one important theorem about NP-complete problem.

Theorem 8.8 *If some NP-complete problem L is in $\mathcal{P}$, then $\mathcal{P} = \mathcal{NP}$.*

Since we believe strongly that there are many problems in $\mathcal{NP}$ that are *not* in $\mathcal{P}$, we thus consider a proof that a problem is NP-complete to be tantamount to a proof in has no polynomial-time algorithm, and thus has no good computer solution.

We believe strongly that none of the NP-complete problems are in $\mathcal{P}$, and the fact that no one has ever found a polynomial-time algorithm for any of the thousands of known NP-complete problems is mutually re-enforcing evidence that none are in $\mathcal{P}$.

Break for 15 minutes

That’s all for our class.