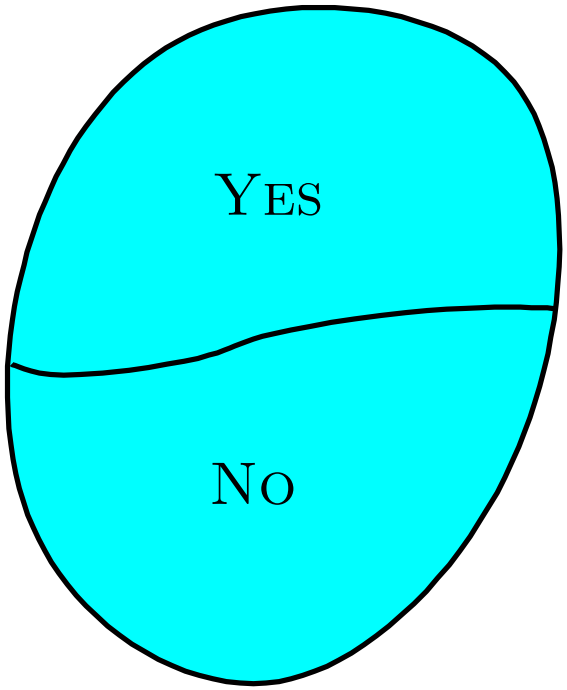
NP-Complete Problems

Polynomial Reductions between Problems

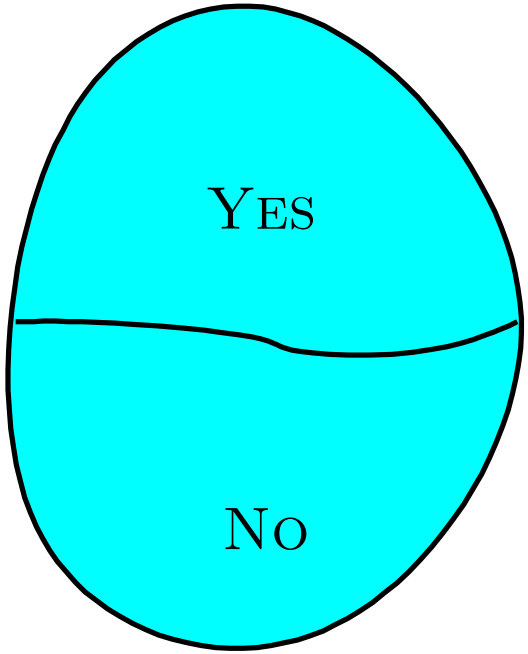
There is a **polynomial reduction** of problem P_1 to problem P_2 if there exists an algorithm Alg with a polynomial time complexity that reduces problem P_1 to problem P_2.

Polynomial Reductions between Problems

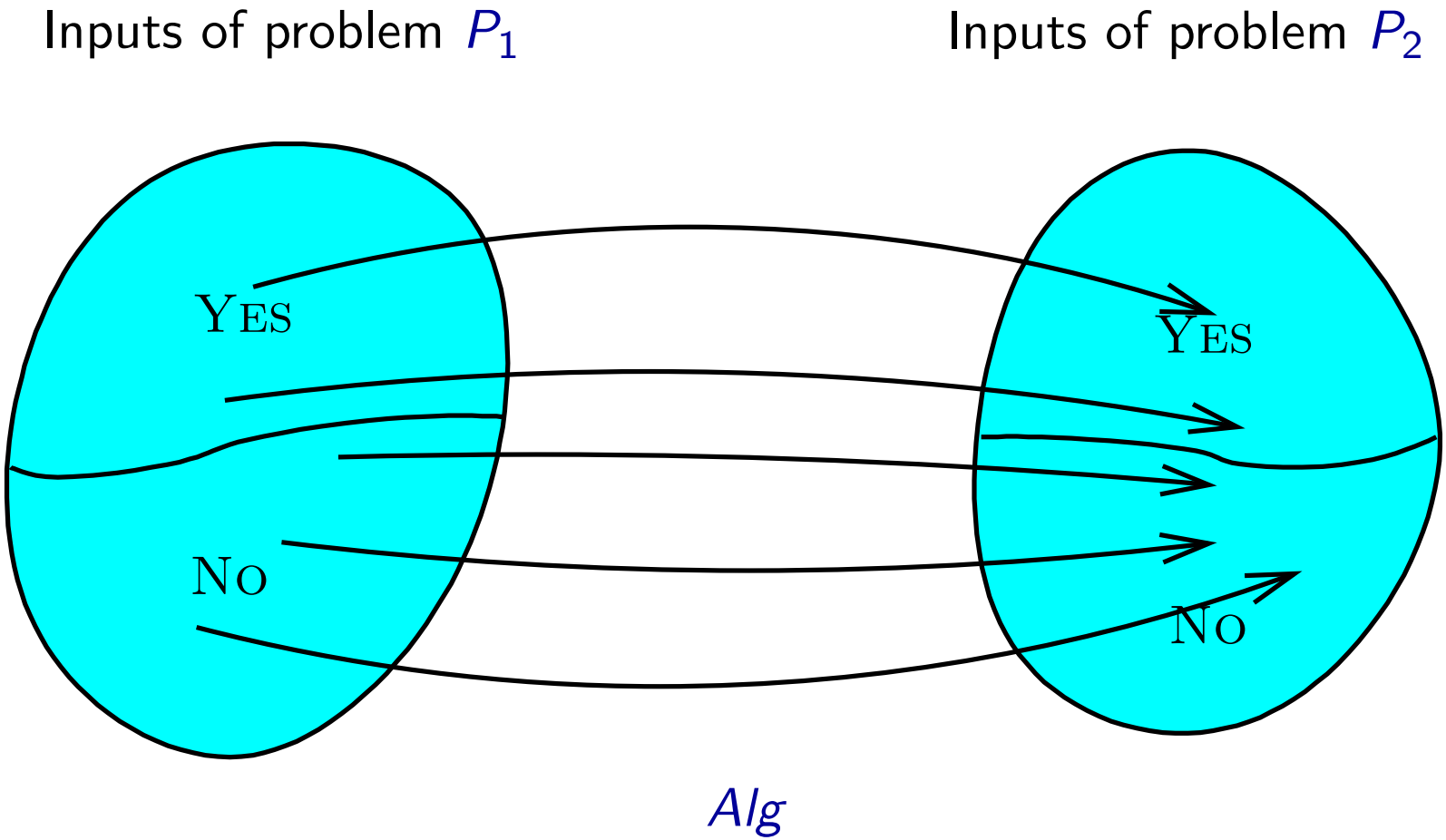
Inputs of problem P_1



Inputs of problem P_2



Polynomial Reductions between Problems



Polynomial Reductions between Problems

Let us say that problem A can be reduced in polynomial time to problem B, i.e., there is a polynomial algorithm P realizing this reduction.

If problem B is in the class PTIME then problem A is also in the class PTIME.

A solution of problem A for an input x:

- Call P with input x and obtain a returned value $P(x)$.
- Call a polynomial time algorithm solving problem B with the input $P(x)$.
  Write the returned value as the answer for A.

That means:

If A is not in PTIME then also B can not be in PTIME.

Polynomial Reductions between Problems

There is a big class of algorithmic problems called **NP-complete** problems such that:

- patří do třídy NPTIME, tj. jsou řešitelné v polynomiálním čase **nedeterministickým** algoritmem
- these problems can be solved by exponential time algorithms
- no polynomial time algorithm is known for any of these problems
- on the other hand, for any of these problems it is not proved that there cannot exist a polynomial time algorithm for the given problem
- every NP-complete problem can be polynomially reduced to any other NP-complete problem

Remark: This is not a definition of NP-complete problems. The precise definition will be described later.

Problem SAT

A typical example of an NP-complete problem is the SAT problem:

SAT (boolean satisfiability problem)

Input: Boolean formula φ.

Question: Is φ satisfiable?

Example:

Formula $\varphi_1 = x_1 \wedge (\neg x_2 \vee x_3)$ is satisfiable:
e.g., for valuation v where $v(x_1) = 1$, $v(x_2) = 0$, $v(x_3) = 1$, the formula φ_1 is true.

Formula $\varphi_2 = (x_1 \wedge \neg x_1) \vee (\neg x_2 \wedge x_3 \wedge x_2)$ is not satisfiable:
it is false for every valuation v.

Problem 3-SAT

3-SAT is a variant of the SAT problem where the possible inputs are restricted to formulas of a certain special form:

3-SAT

Input: Formula φ is a conjunctive normal form where every clause contains exactly 3 literals.

Question: Is φ satisfiable?

Problem 3-SAT

Recalling some notions:

- A **literal** is a formula of the form x or $\neg x$ where x is an atomic proposition.
- A **clause** is a disjuction of literals.

  Examples: $x_1 \vee \neg x_2$ $\neg x_5 \vee x_8 \vee \neg x_{15} \vee \neg x_{23}$ x_6
- A formula is in a **conjuctive normal form (CNF)** if it is a conjuction of clauses.

  Example: $(x_1 \vee \neg x_2) \wedge (\neg x_5 \vee x_8 \vee \neg x_{15} \vee \neg x_{23}) \wedge x_6$

So in the 3-SAT problem we require that a formula φ is in a CNF and moreover that every clause of φ contains exactly three literals.

Example:

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

Problem 3-SAT

The following formula is satisfiable:

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

It is true for example for valuation v where

$$v(x_1) = 0$$
$$v(x_2) = 1$$
$$v(x_3) = 0$$
$$v(x_4) = 1$$

On the other hand, the following formula is not satisfiable:

$$(x_1 \vee x_1 \vee x_1) \wedge (\neg x_1 \vee \neg x_1 \vee \neg x_1)$$

Polynomial Reductions between Problems

As an example, a polynomial time reduction from the 3-SAT problem to the independent set problem (IS) will be described.

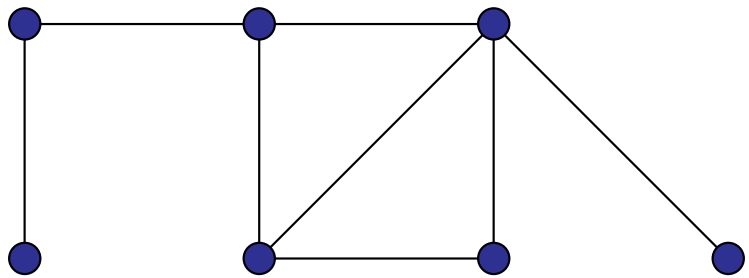
Remark: Both 3-SAT and IS are examples of NP-complete problems.

Independent Set (IS) Problem

Independent set (IS) problem

Input: An undirected graph G, a number k.

Question: Is there an independent set of size k in the graph G?



$k = 4$

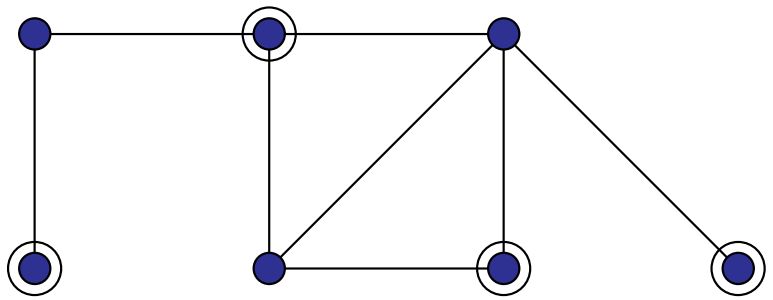
Remark: An **independent set** in a graph is a subset of nodes of the graph such that no pair of nodes from this set is connected by an edge.

Independent Set (IS) Problem

Independent set (IS) problem

Input: An undirected graph G, a number k.

Question: Is there an independent set of size k in the graph G?

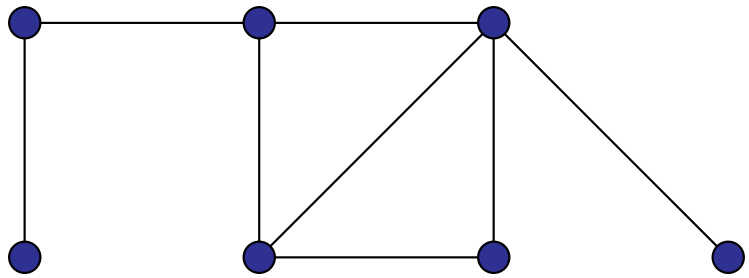


$k = 4$

Remark: An **independent set** in a graph is a subset of nodes of the graph such that no pair of nodes from this set is connected by an edge.

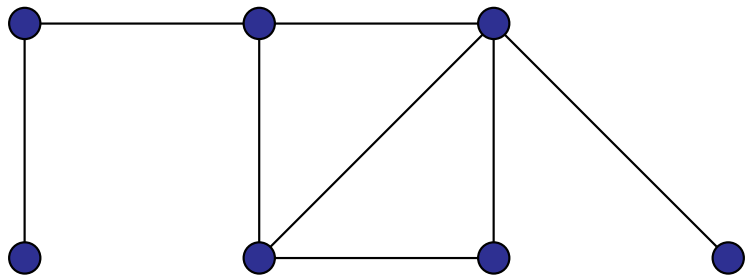
Independent Set (IS) Problem

An example of an instance where the answer is YES:



$k = 4$

An example of an instance where the answer is NO:



$k = 5$

A Reduction from 3-SAT to IS

We describe a (polynomial-time) algorithm with the following properties:

- **Input:** An arbitrary instance of 3-SAT, i.e., a formula φ in a conjunctive normal form where every clause contains exactly three literals.

- **Output:** An instance of IS, i.e., an undirected graph G and a number k.

- Moreover, the following will be ensured for an arbitrary input (i.e., for an arbitrary formula φ in the above mentioned form):

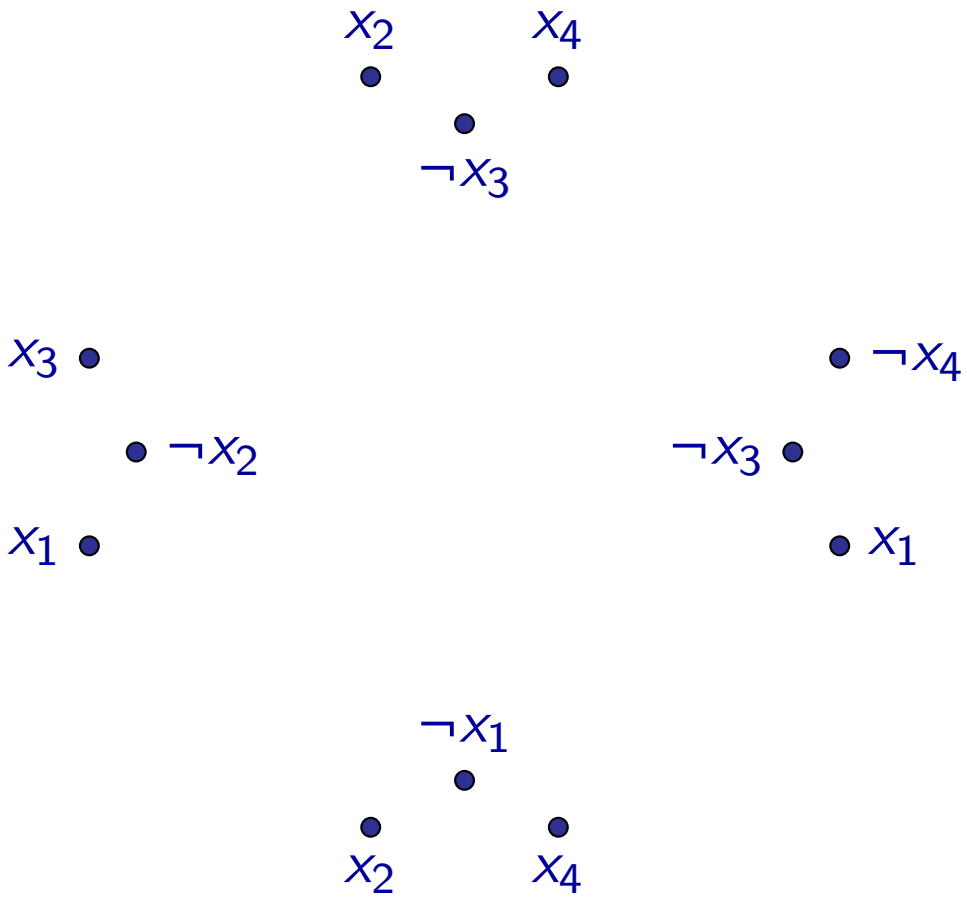
  There will be an independent set of size k in graph G iff formula φ will be satisfiable.

A Reduction from 3-SAT to IS

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

A Reduction from 3-SAT to IS

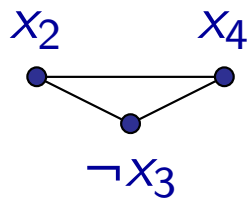
$$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$$

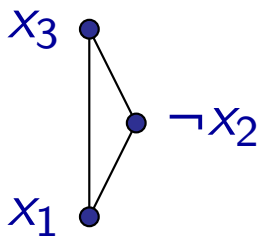


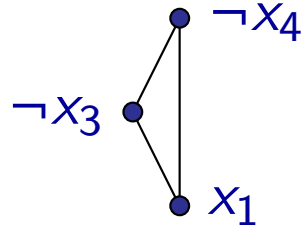
For each occurrence of a literal we add a node to the graph.

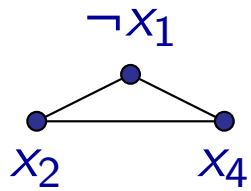
A Reduction from 3-SAT to IS

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





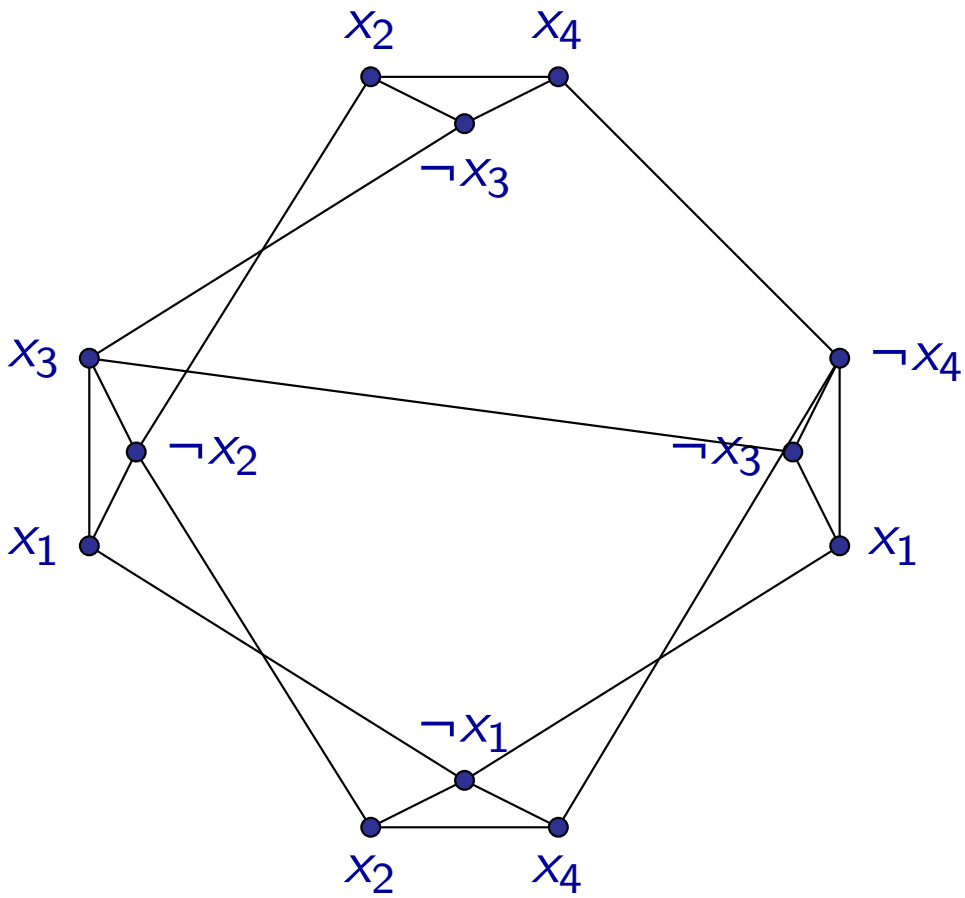




We connect with edges the nodes corresponding to occurrences of literals belonging to the same clause.

A Reduction from 3-SAT to IS

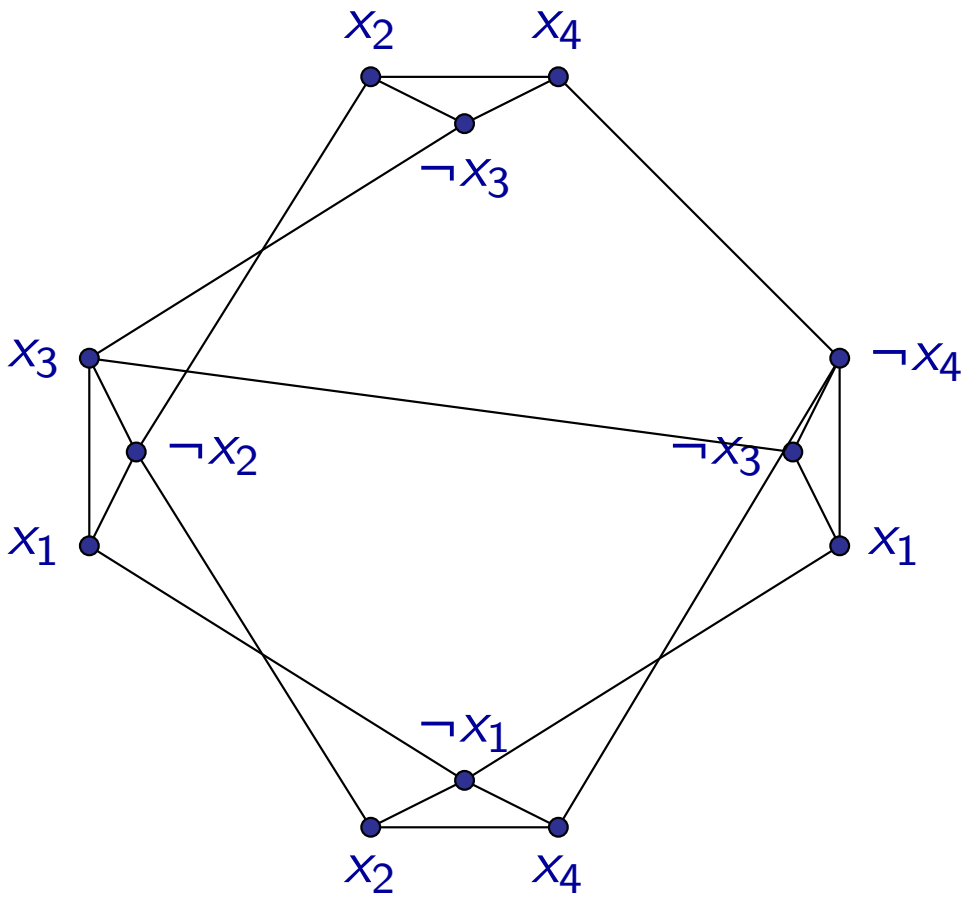
$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$



For each pair of nodes corresponding to literals x_i and $\neg x_i$ we add an edge between them.

A Reduction from 3-SAT to IS

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

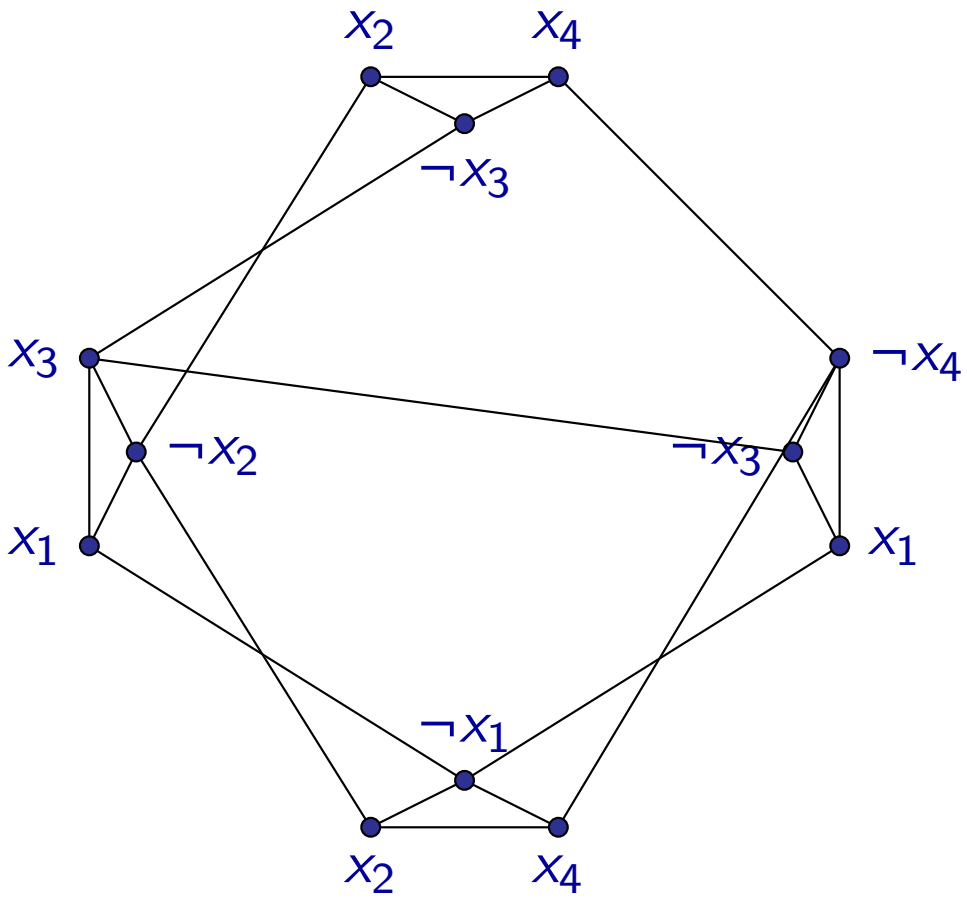


$k = 4$

We put k to be equal to the number of clauses.

A Reduction from 3-SAT to IS

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



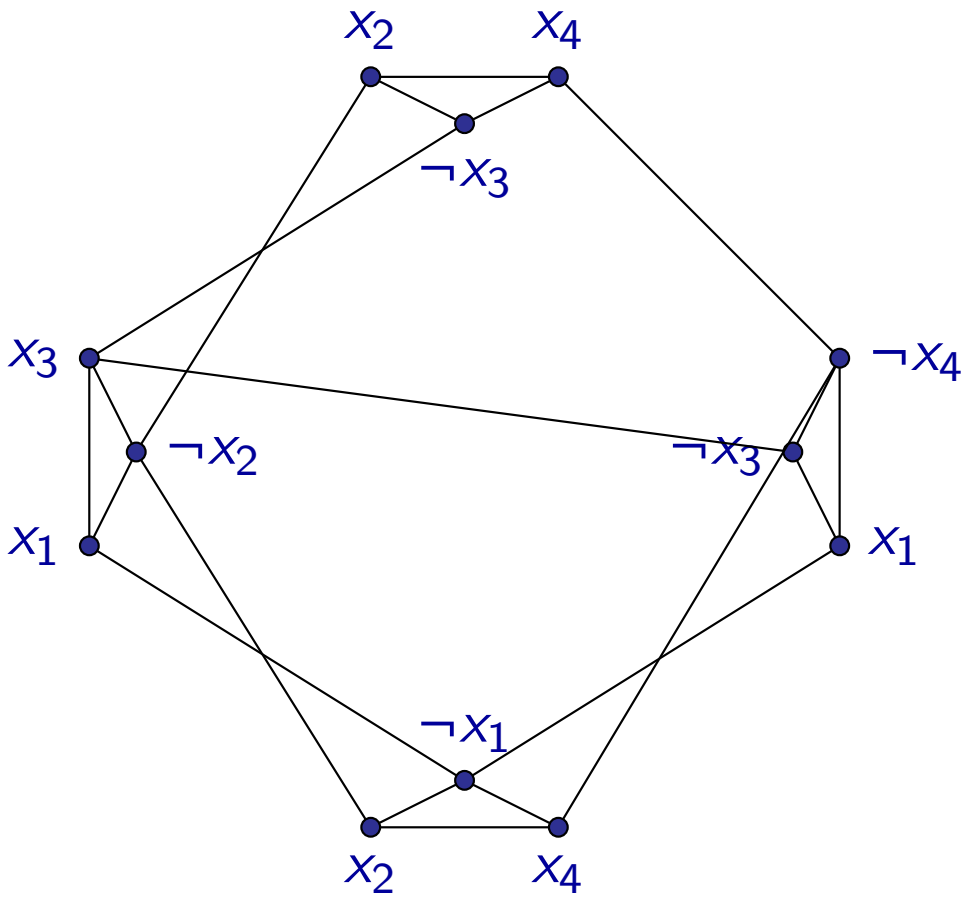
$k = 4$

The constructed graph and number k are the output of the algorithm.

A Reduction from 3-SAT to IS

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

$v(x_1) = 1$
$v(x_2) = 1$
$v(x_3) = 0$
$v(x_4) = 1$



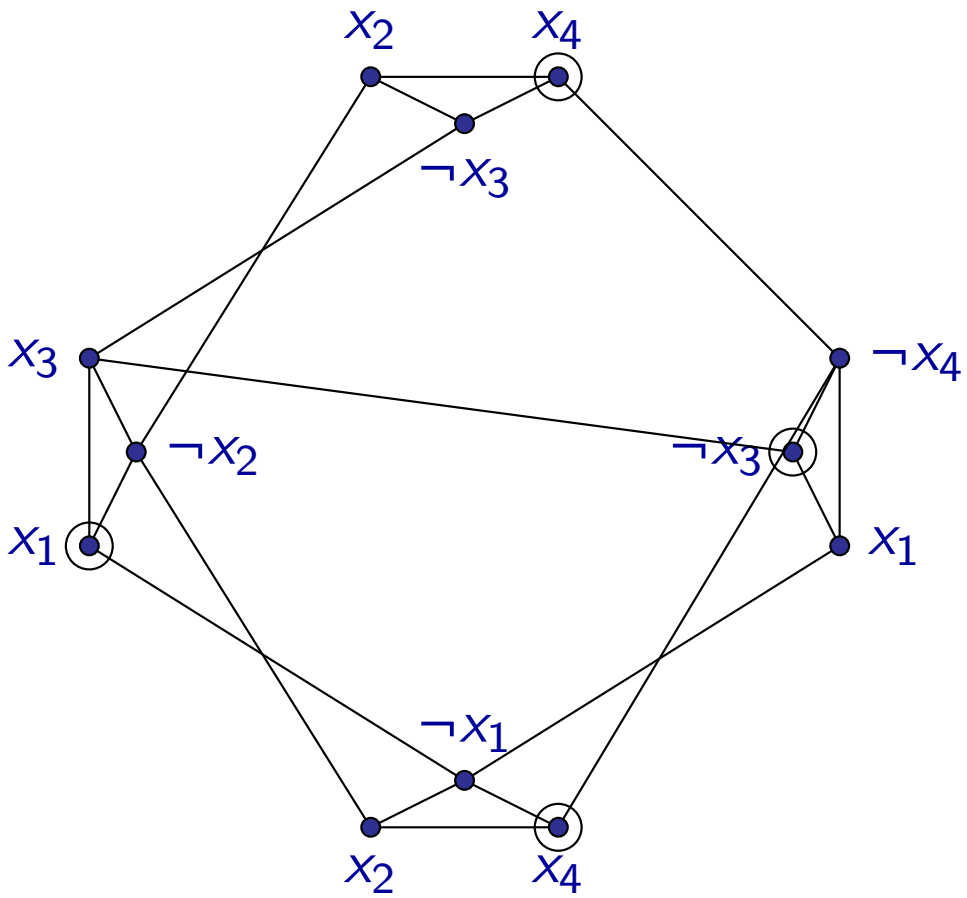
$k = 4$

If the formula φ is satisfiable then there is a valuation v where every clause contains at least one literal with value 1.

A Reduction from 3-SAT to IS

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

$v(x_1) = 1$
$v(x_2) = 1$
$v(x_3) = 0$
$v(x_4) = 1$



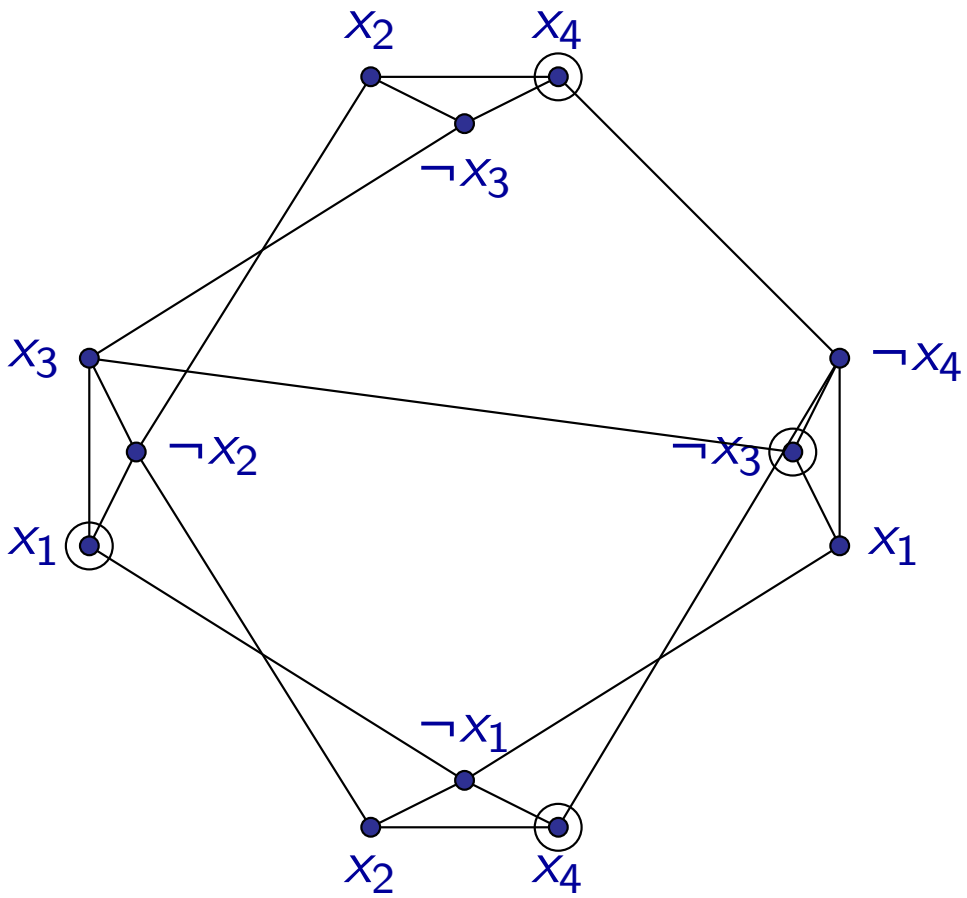
$k = 4$

We select one literal that has a value 1 in the valuation v, and we put the corresponding node into the independent set.

A Reduction from 3-SAT to IS

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

$v(x_1) = 1$
$v(x_2) = 1$
$v(x_3) = 0$
$v(x_4) = 1$



$k = 4$

We can easily verify that the selected nodes form an independent set.

A Reduction from 3-SAT to IS

The selected nodes form an independent set because:

- One node has been selected from each triple of nodes corresponding to one clause.
- Nodes denoted x_i and $\neg x_i$ could not be selected together. (Exactly of them has the value 1 in the given valuation v.)

A Reduction from 3-SAT to IS

On the other hand, if there is an independent set of size k in graph G, then it surely has the following properties:

- At most one node is selected from each triple of nodes corresponding to one clause.

  But because there are k clauses and k nodes are selected, exactly one node must be selected from each triple.

- Nodes denoted x_i and $\neg x_i$ cannot be selected together.

We can choose a valuation according to the selected nodes, since it follows from the previous discussion that it must exist.
(Arbitrary values can be assigned to the remaining variables.)

For the given valuation, the formula φ has surely the value 1, since in each clause there is at least one literal with value 1.

A Reduction from 3-SAT to IS

It is obvious that the running time of the described algorithm polynomial:

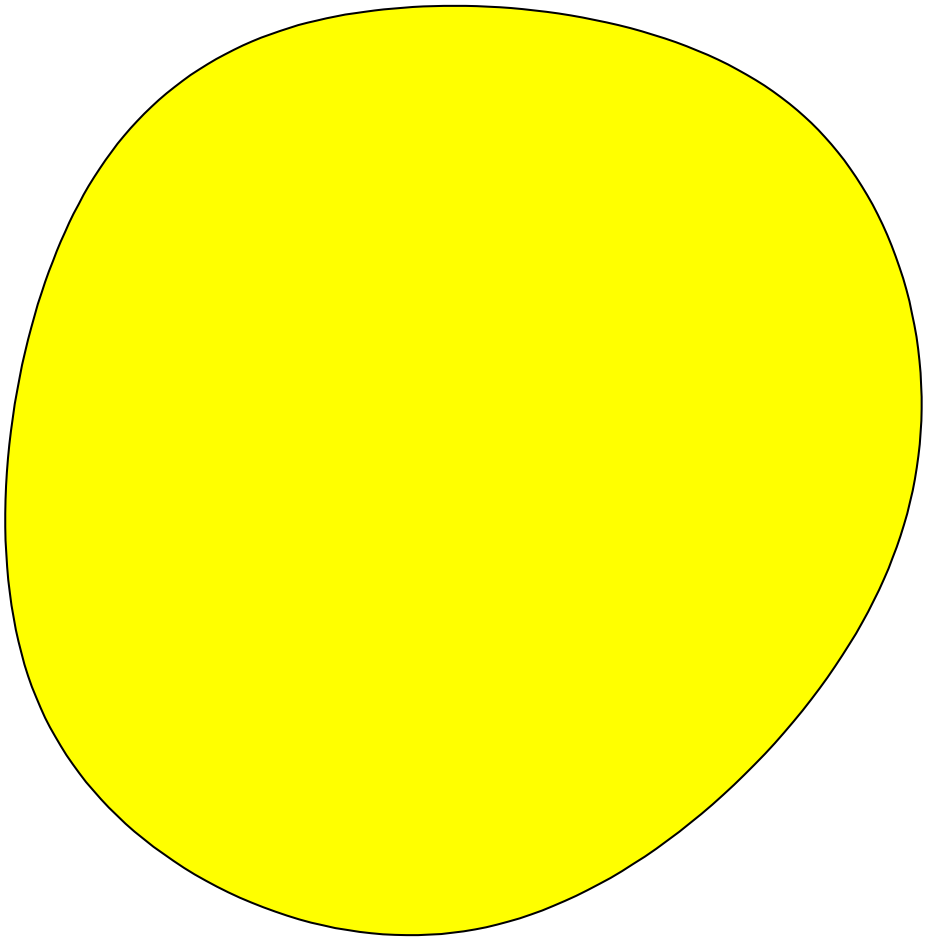
Graph G and number k can be constructed for a formula φ in time $O(n^2)$, where n is the size of formula φ.

We have also seen that there is an independent set of size k in the constructed graph G iff the formula φ is satisfiable.

The described algorithm shows that 3-SAT can be reduced in polynomial time to IS.

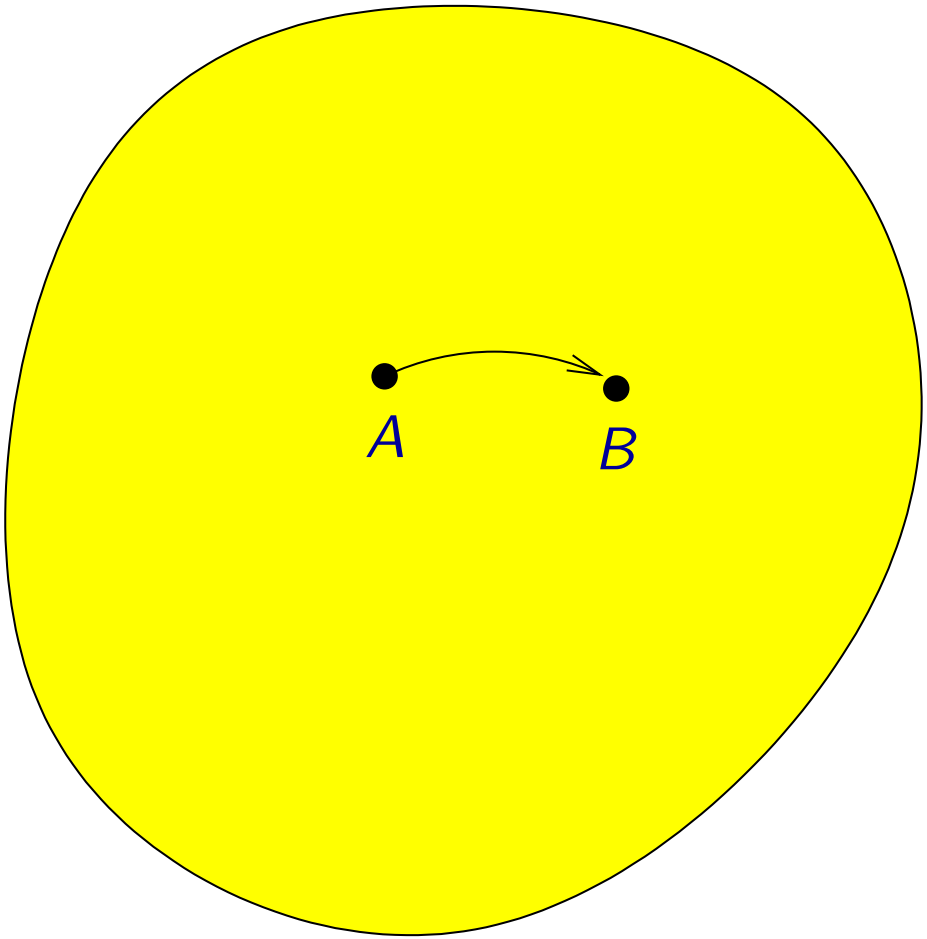
NP-Complete Problems

Let us consider a set of all decision problems.



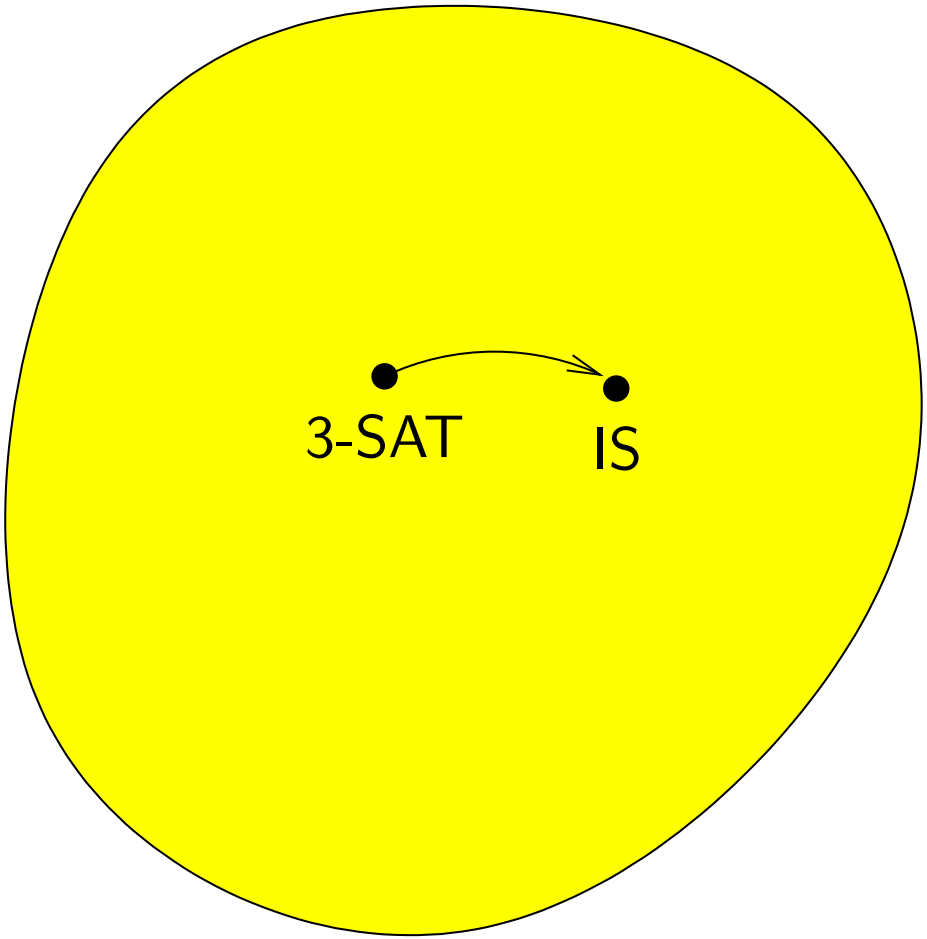
NP-Complete Problems

By an arrow we denote that a problem A can be reduced in polynomial time to a problem B.



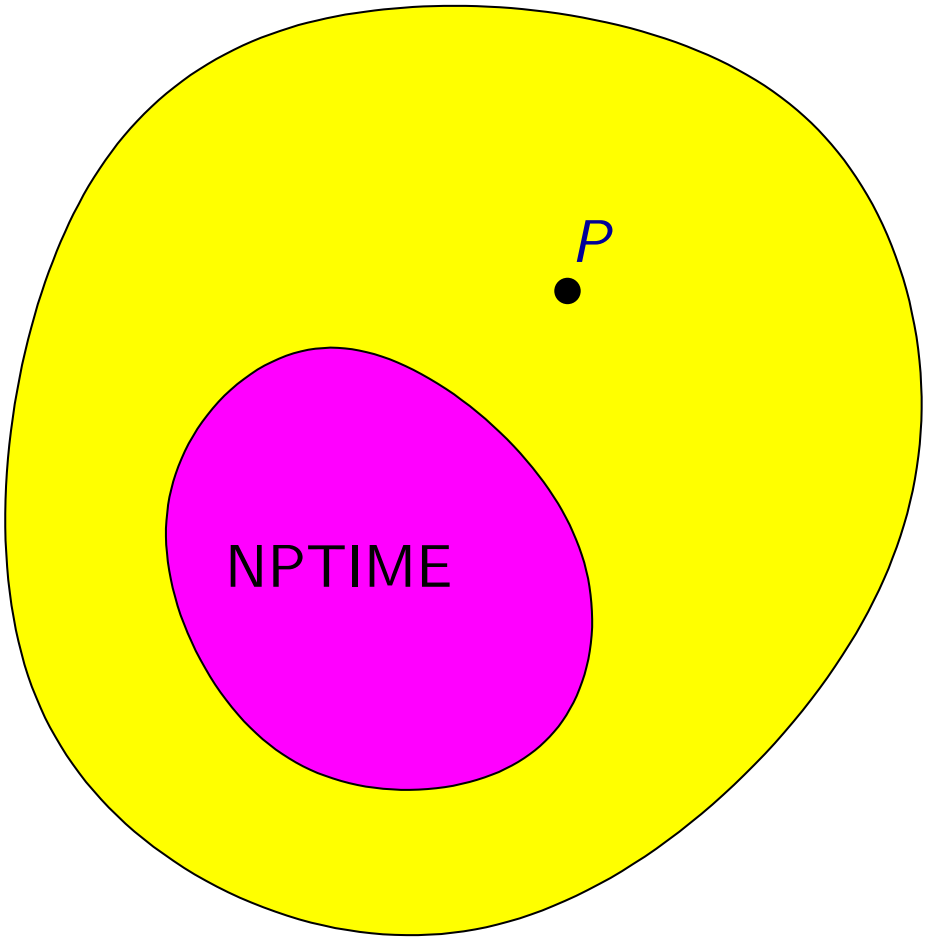
NP-Complete Problems

For example 3-SAT can be reduced in polynomial time to IS.



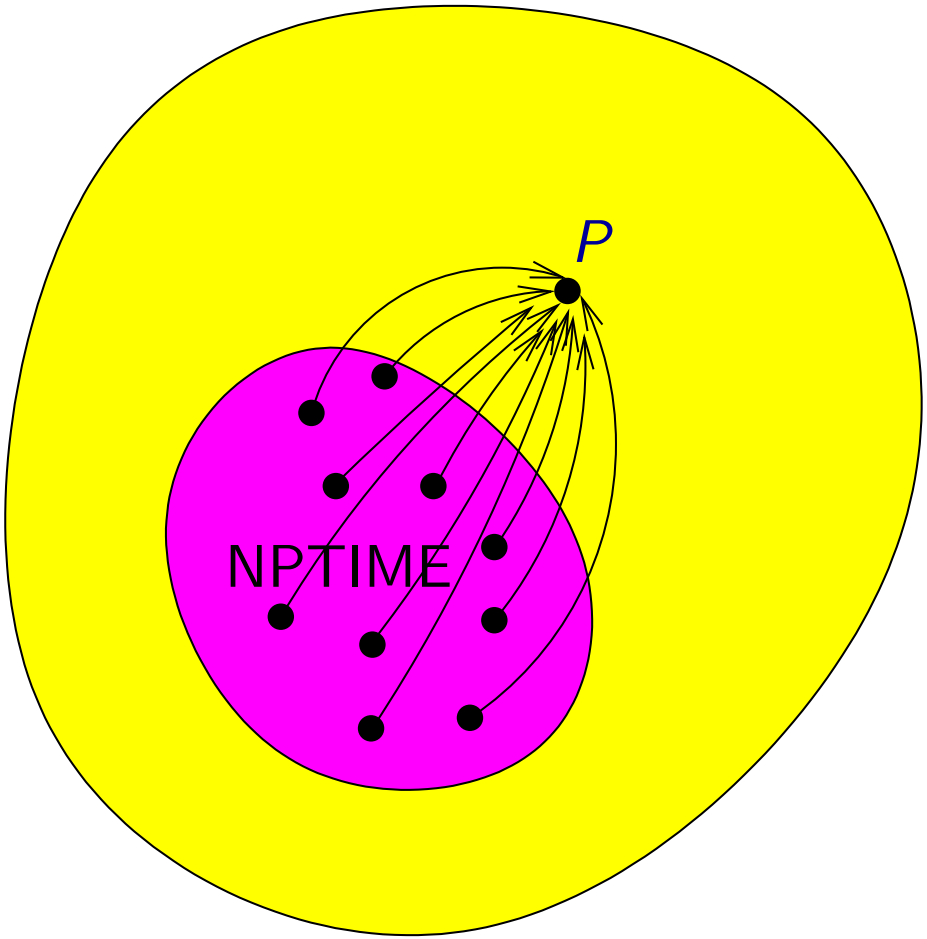
NP-Complete Problems

Let us consider now the class NPTIME and a problem P.



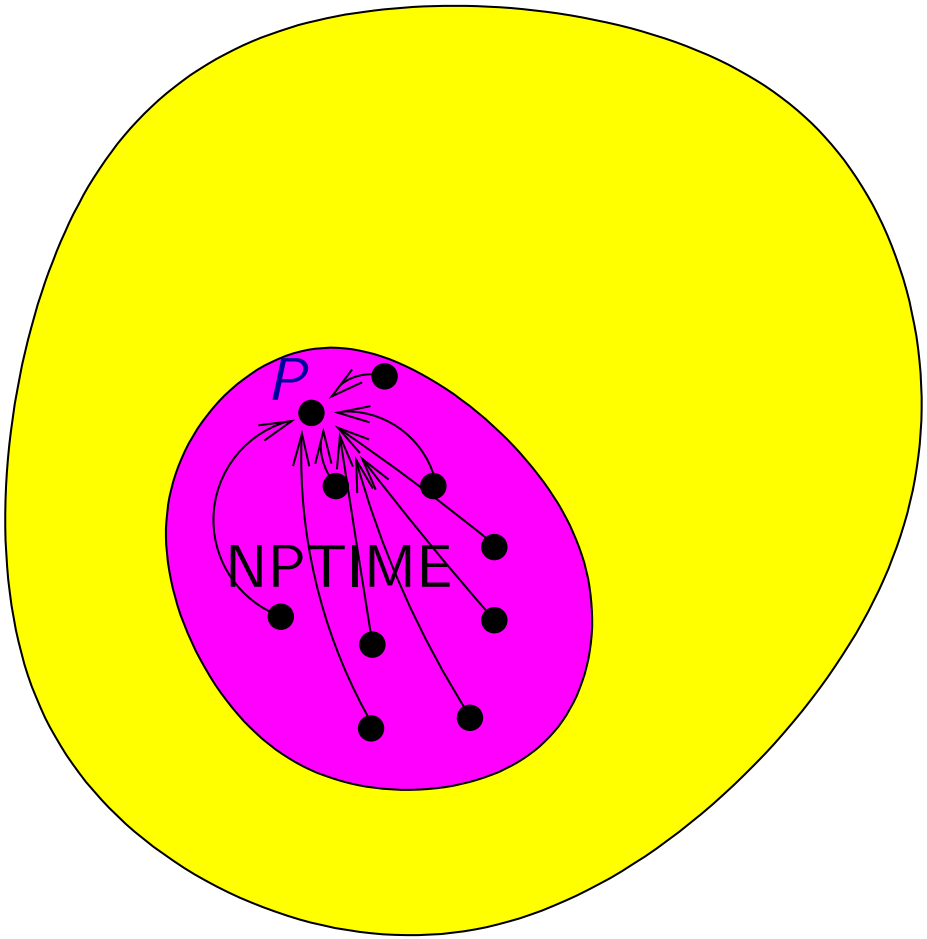
NP-Complete Problems

A problem P is NP-**hard** if every problem from NPTIME can be reduced in polynomial time to P.



NP-Complete Problems

A problem P is **NP-complete** if it is NP-hard and it belongs to the class NPTIME.



NP-Complete Problems

If we have found a polynomial time algorithm for some NP-hard problem P, then we would have polynomial time algorithms for all problems P' from NPTIME:

- At first we would apply an algorithm for the reduction from P' to P on an input of a problem P'.
- Then we would use a polynomial algorithm for P on the constructed instance of P and returned its result as the answer for the original instance of P'.

Is such case, PTIME = NPTIME would hold, since for every problem from NPTIME there would be a polynomial-time (deterministic) algorithm.

NP-Complete Problems

On the other hand, if there is at least one problem from NPTIME for which a polynomial-time algorithm does not exist, then it means that for none of NP-hard problems there is a polynomial-time algorithm.

It is an open question whether the first or the second possibility holds.

NP-Complete Problems

It is not difficult to see that:

If a problem A can be reduced in a polynomial time to a problem B and problem B can be reduced in a polynomial time to a problem C, then problem A can be reduced in a polynomial time to problem C.

So if we know about some problem P that it is NP-hard and that P can be reduced in a polynomial time to a problem P', then we know that the problem P' is also NP-hard.

NP-Complete Problems

Věta (Cook, 1971)

Problem SAT is NP-complete.

It can be shown that SAT can be reduced in a polynomial time to 3-SAT and we have seen that 3-SAT can be reduced in a polynomial time to IS.

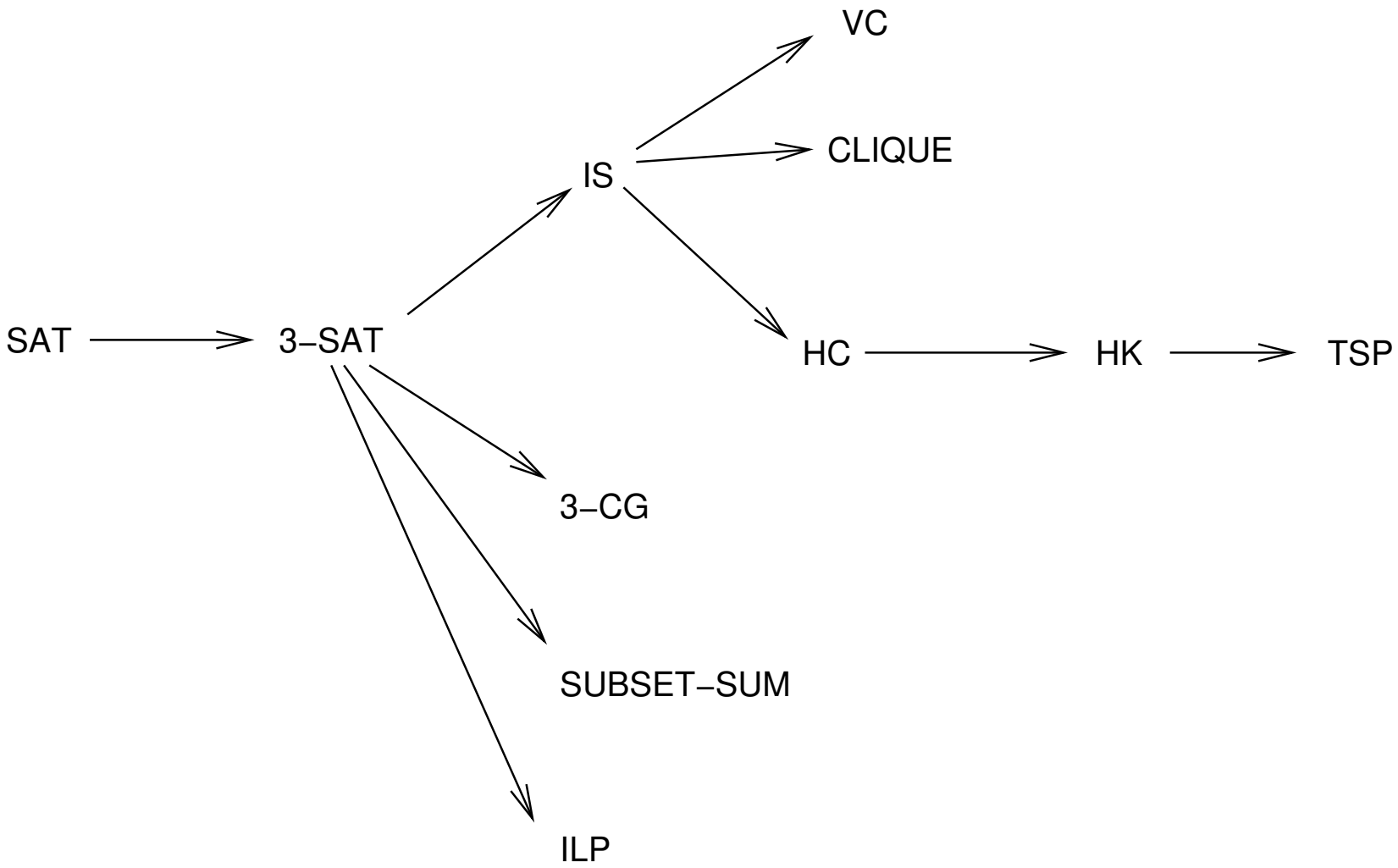
This means that problems 3-SAT and IS are NP-hard.

It is not difficult to show that 3-SAT and IS belong to the class NPTIME.

Problems 3-SAT and IS are NP-complete.

NP-Complete Problems

By a polynomial reductions from problems that are already known to be NP-complete, NP-completeness of many other problems can be shown:



Examples of Some NP-Complete Problems

The following previously mentioned problems are NP-complete:

- SAT (boolean satisfiability problem)
- 3-SAT
- IS — independent set problem

On the following slides, examples of some other NP-complete problems are described:

- CG — graph coloring (remark: it is NP-complete even in the special case where we have 3 colors)
- VC — vertex cover
- CLIQUE — clique problem
- HC — Hamiltonian cycle
- HK — Hamiltonian circuit
- TSP — traveling salesman problem
- SUBSET-SUM
- ILP — integer linear programming

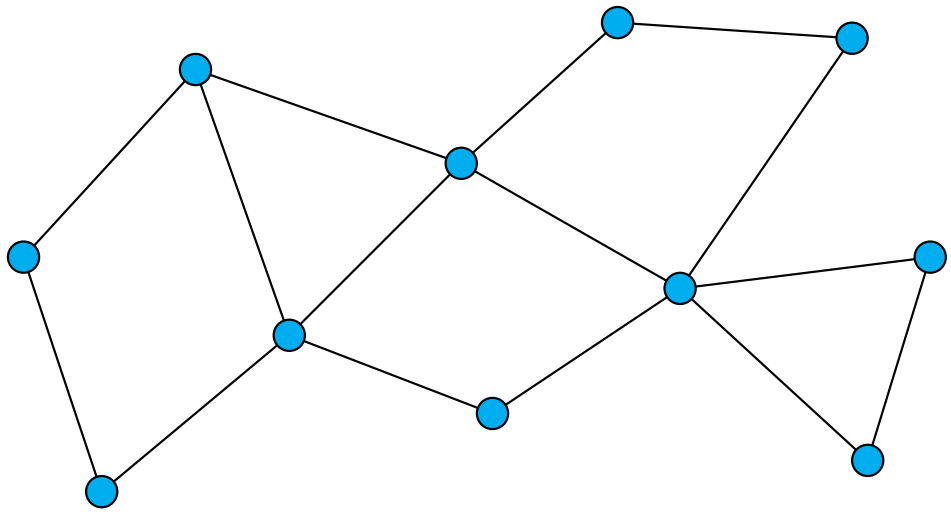
Graph Coloring

Graph coloring

Input: An undirected graph G, a natural number k.

Question: Is it possible to color nodes of the graph G using k colors in such a way that there is no pair of nodes where both nodes are colored with the same color and connected with an edge?

Example: $k = 3$



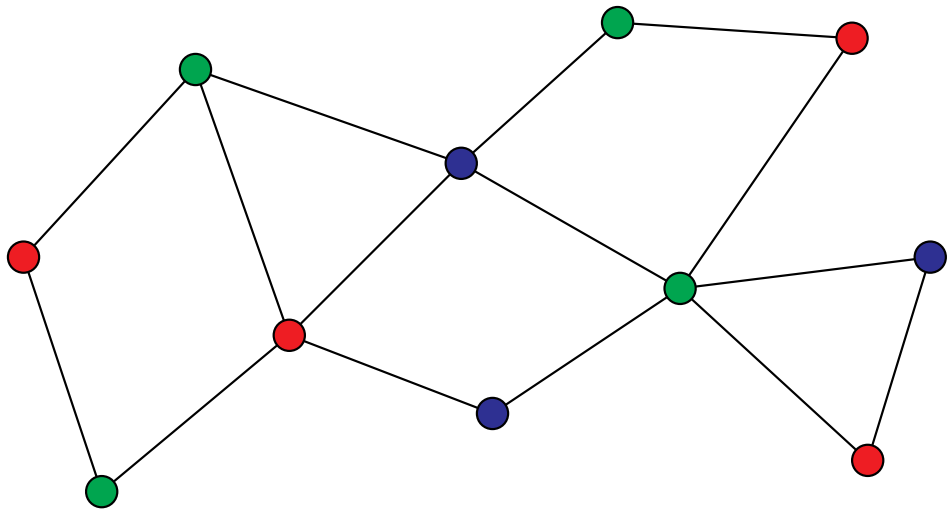
Graph Coloring

Graph coloring

Input: An undirected graph G, a natural number k.

Question: Is it possible to color nodes of the graph G using k colors in such a way that there is no pair of nodes where both nodes are colored with the same color and connected with an edge?

Example: $k = 3$



Answer: YES

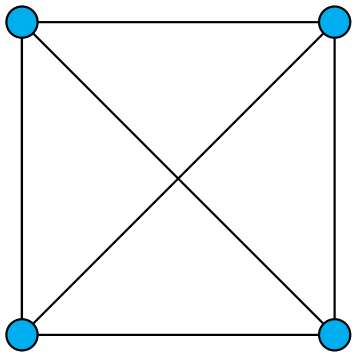
Graph Coloring

Graph coloring

Input: An undirected graph G, a natural number k.

Question: Is it possible to color nodes of the graph G using k colors in such a way that there is no pair of nodes where both nodes are colored with the same color and connected with an edge?

Example: $k = 3$



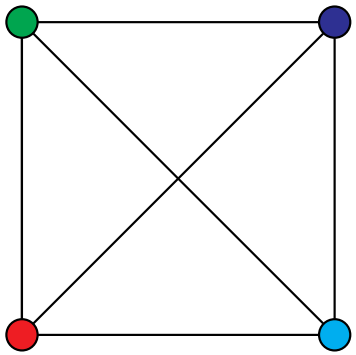
Graph Coloring

Graph coloring

Input: An undirected graph G, a natural number k.

Question: Is it possible to color nodes of the graph G using k colors in such a way that there is no pair of nodes where both nodes are colored with the same color and connected with an edge?

Example: $k = 3$



Answer: No

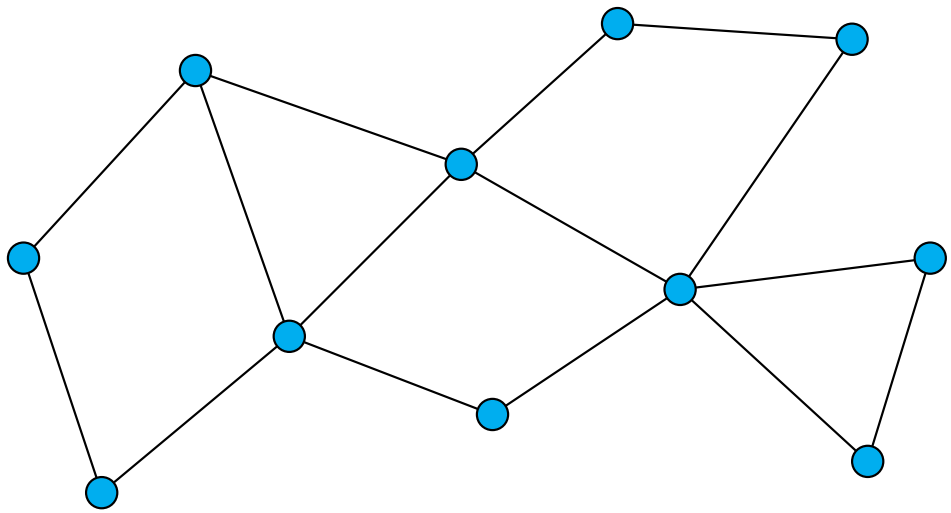
VC – Vertex Cover

VC – vertex cover

Input: An undirected graph G and a natural number k.

Question: Is there some subset of nodes of G of size k such that every edge has at least one of its nodes in this subset?

Example: $k = 6$



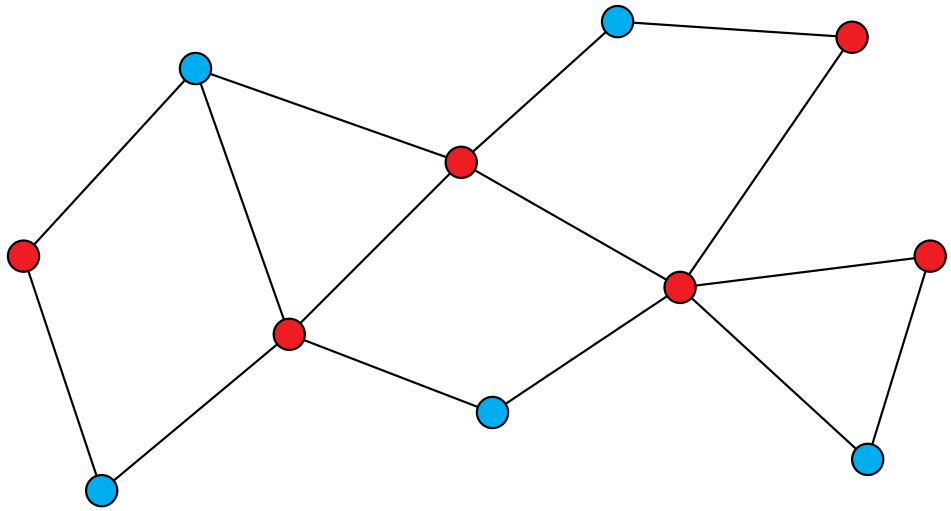
VC – Vertex Cover

VC – vertex cover

Input: An undirected graph G and a natural number k.

Question: Is there some subset of nodes of G of size k such that every edge has at least one of its nodes in this subset?

Example: $k = 6$



Answer: YES

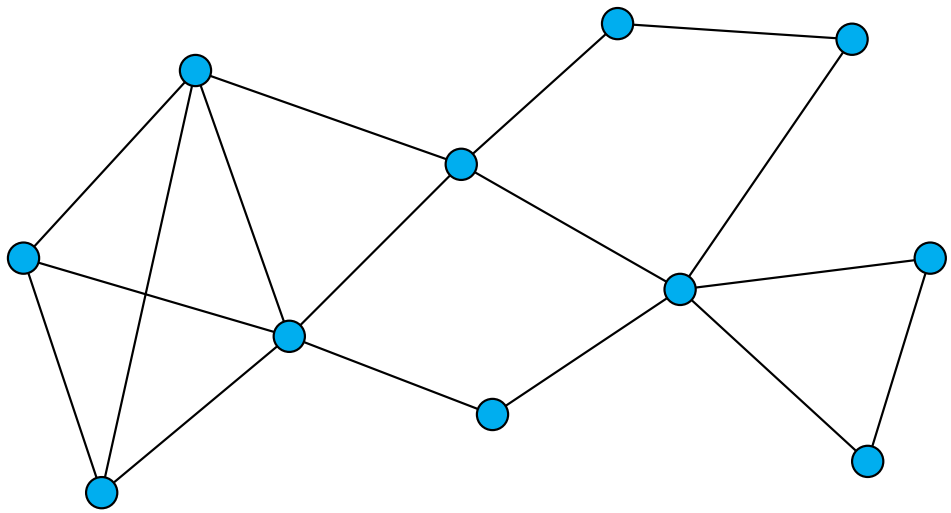
CLIQUE

CLIQUE

Input: An undirected graph G and a natural number k.

Question: Is there some subset of nodes of G of size k such that every two nodes from this subset are connected by an edge?

Example: $k = 4$

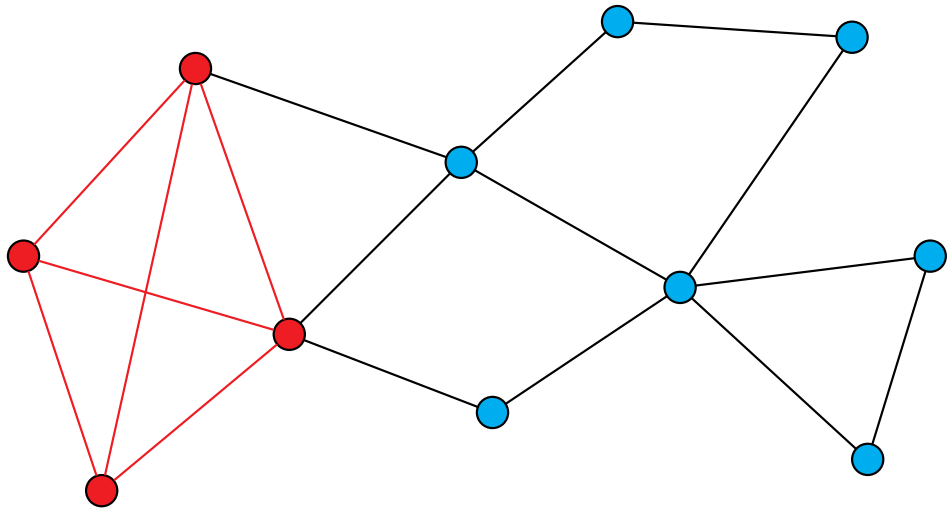


CLIQUE

Input: An undirected graph G and a natural number k.

Question: Is there some subset of nodes of G of size k such that every two nodes from this subset are connected by an edge?

Example: $k = 4$



Answer: YES

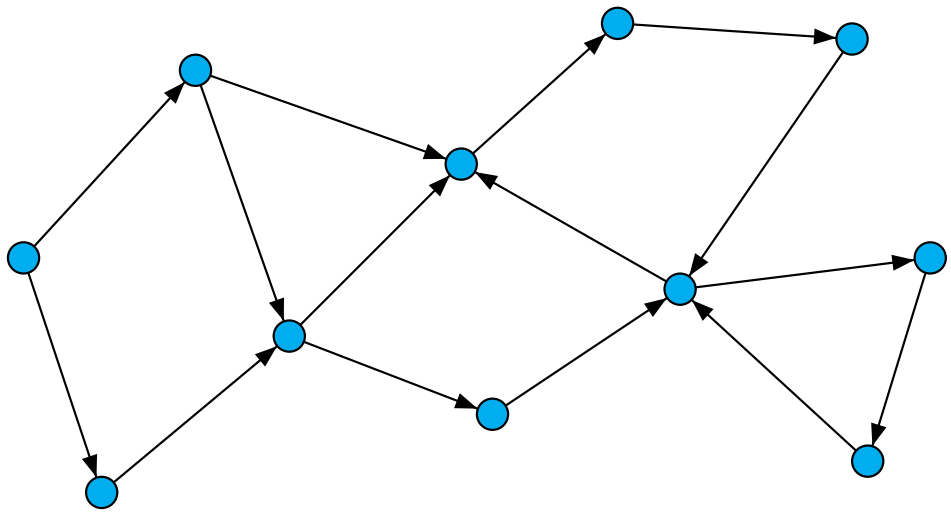
Hamiltonian Cycle

HC – Hamiltonian cycle

Input: A directed graph G.

Question: Is there a Hamiltonian cycle in G (i.e., a directed cycle going through each node exactly once)?

Example:



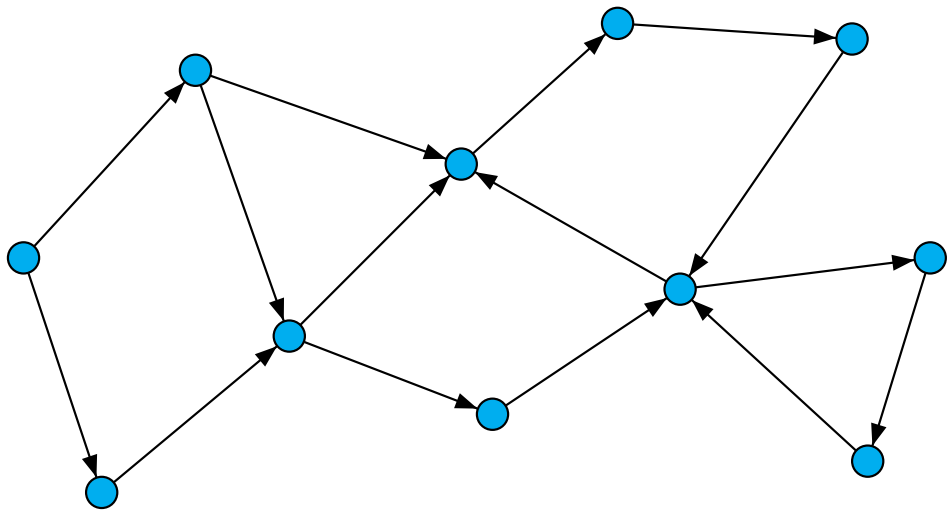
Hamiltonian Cycle

HC – Hamiltonian cycle

Input: A directed graph G.

Question: Is there a Hamiltonian cycle in G (i.e., a directed cycle going through each node exactly once)?

Example:



Answer: No

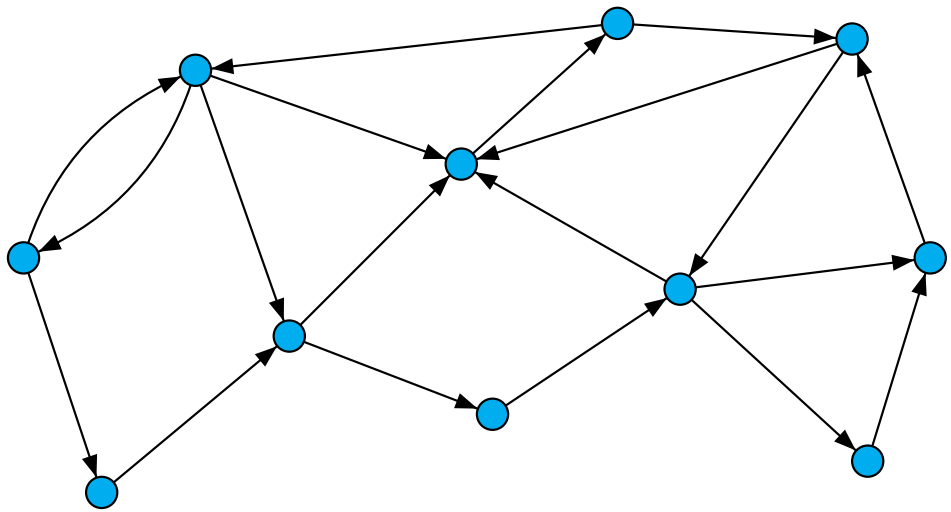
Hamiltonian Cycle

HC – Hamiltonian cycle

Input: A directed graph G.

Question: Is there a Hamiltonian cycle in G (i.e., a directed cycle going through each node exactly once)?

Example:



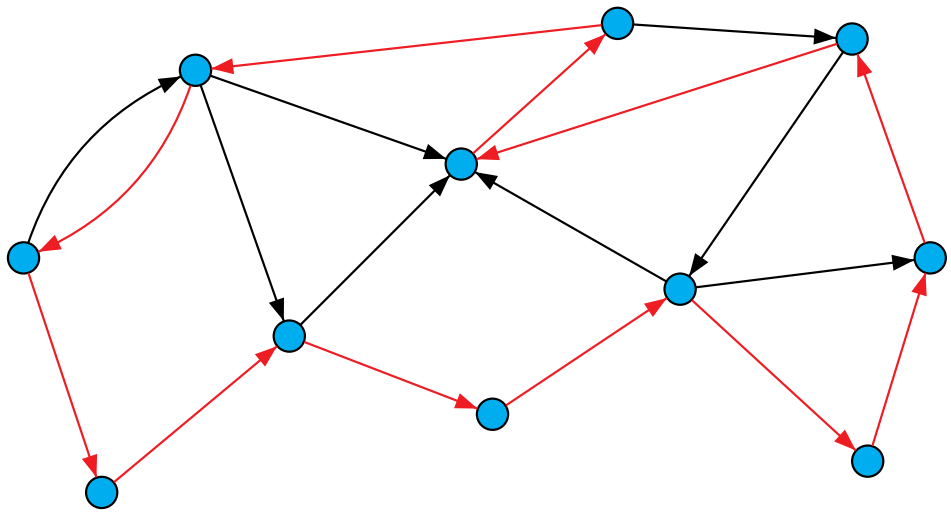
Hamiltonian Cycle

HC – Hamiltonian cycle

Input: A directed graph G.

Question: Is there a Hamiltonian cycle in G (i.e., a directed cycle going through each node exactly once)?

Example:



Answer: YES

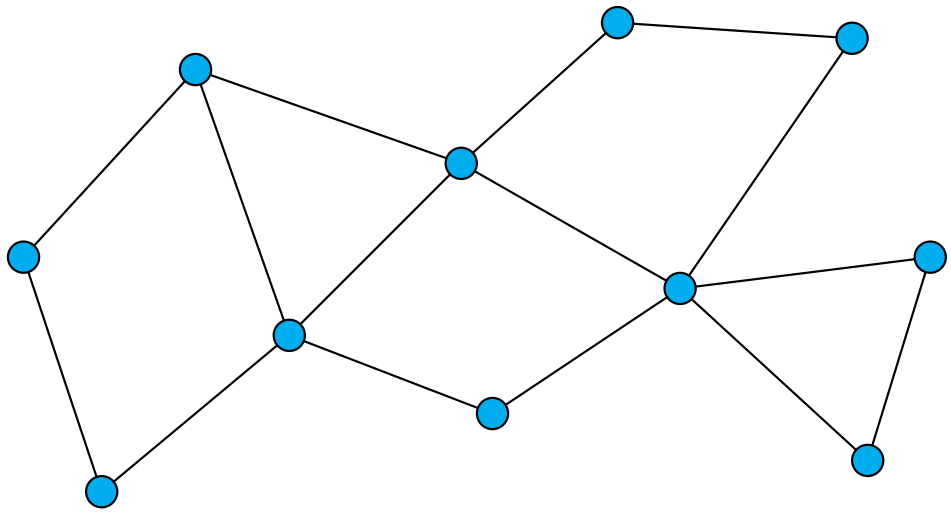
Hamiltonian Circuit

HK – Hamiltonian circuit

Input: An undirected graph G.

Question: Is there a Hamiltonian circuit in G (i.e., an undirected cycle going through each node exactly once)?

Example:



Answer: NO

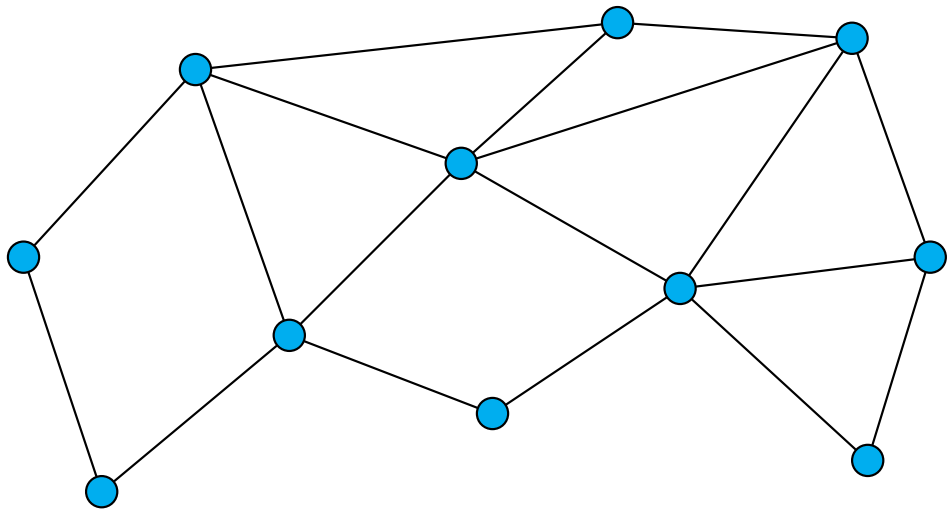
Hamiltonian Circuit

HK – Hamiltonian circuit

Input: An undirected graph G.

Question: Is there a Hamiltonian circuit in G (i.e., an undirected cycle going through each node exactly once)?

Example:



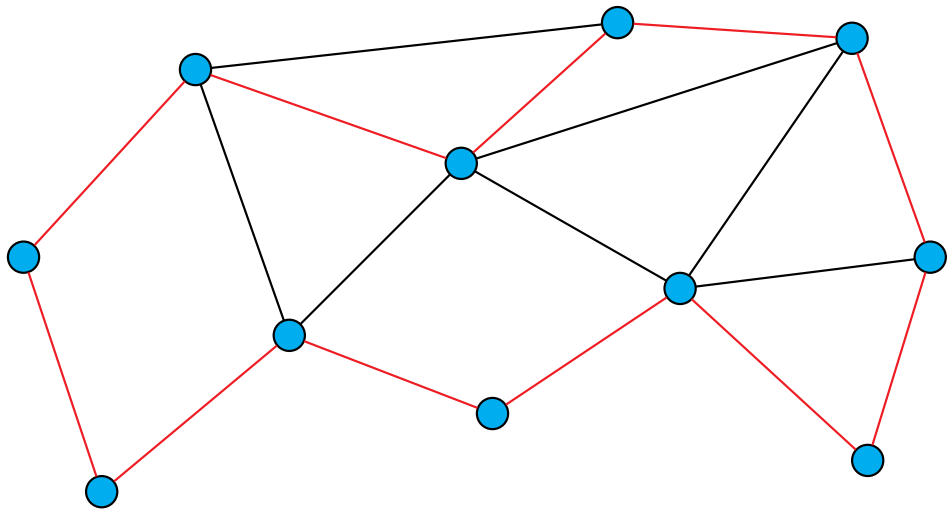
Hamiltonian Circuit

HK – Hamiltonian circuit

Input: An undirected graph G.

Question: Is there a Hamiltonian circuit in G (i.e., an undirected cycle going through each node exactly once)?

Example:



Answer: YES

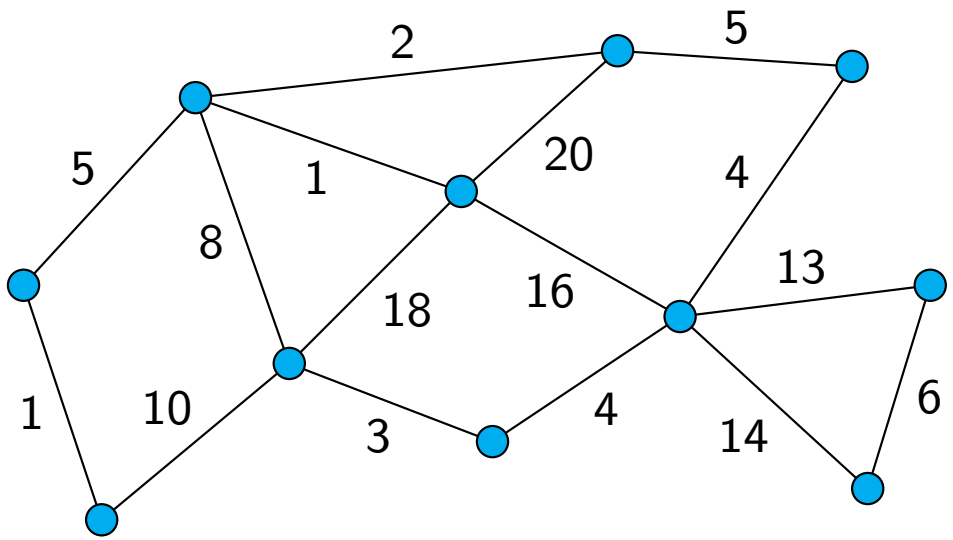
Traveling Salesman Problem

TSP - traveling salesman problem

Input: An undirected graph G with edges labelled with natural numbers and a number k.

Question: Is there a closed tour going through all nodes of the graph G such that the sum of labels of edges on this tour is at most k?

Example: $k = 70$



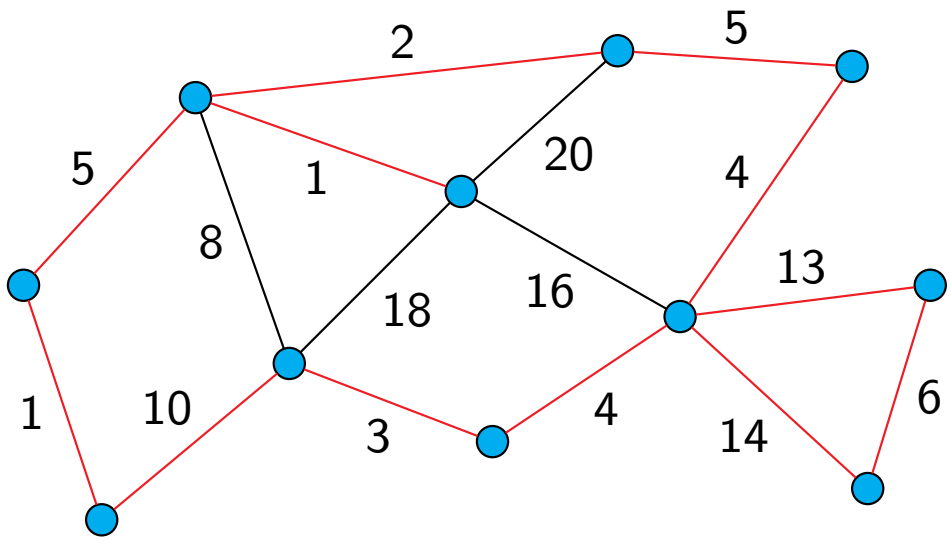
Traveling Salesman Problem

TSP - traveling salesman problem

Input: An undirected graph G with edges labelled with natural numbers and a number k.

Question: Is there a closed tour going through all nodes of the graph G such that the sum of labels of edges on this tour is at most k?

Example: $k = 70$



Answer: YES, since there is a tour with the sum 69.

SUBSET-SUM

Problem SUBSET-SUM

Input: A sequence $a_1, a_2, \ldots, a_n$ of natural numbers and a natural number s.

Question: Is there a set $I \subseteq \{1, 2, \ldots, n\}$ such that $\sum_{i \in I} a_i = s$?

In other words, the question is whether it is possible to select a subset with sum s of a given (multi)set of numbers.

Example: For the input consisting of numbers $3, 5, 2, 3, 7$ and number $s = 15$ the answer is YES, since $3 + 5 + 7 = 15$.

For the input consisting of numbers $3, 5, 2, 3, 7$ and number $s = 16$ the answer is NO, since no subset of these numbers has sum 16.

SUBSET-SUM

Remark:
The order of numbers $a_1, a_2, \ldots, a_n$ in an input is not important.

Note that this is not exactly the same as if we have formulated the problem so that the input is a set $\{a_1, a_2, \ldots, a_n\}$ and a number s — numbers cannot occur multiple times in a set but they can in a sequence.

SUBSET-SUM

Problem SUBSET-SUM is a special case of a **knapsack problem**:

Knapsack problem

Input: Sequence of pairs of natural numbers $(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)$ and two natural numbers s and t.

Question: Is there a set $I \subseteq \{1, 2, \ldots, n\}$ such that $\sum_{i \in I} a_i \leq s$ and $\sum_{i \in I} b_i \geq t$?

SUBSET-SUM

Informally, the knapsack problem can be formulated as follows:

We have n objects, where the i-th object weights a_i grams and its price is b_i dollars.

The question is whether there is a subset of these objects with total weight at most s grams (s is the capacity of the knapsack) and with total price at least t dollars.

Remark:
Here we have formulated this problem as a decision problem.

This problem is usually formulated as an optimization problem where the aim is to find such a set $I \subseteq \{1, 2, \ldots, n\}$, where the value $\sum_{i \in I} b_i$ is maximal and where the condition $\sum_{i \in I} a_i \leq s$ is satisfied, i.e., where the capacity of the knapsack is not exceeded.

SUBSET-SUM

That SUBSET-SUM is a special case of the Knapsack problem can be seen from the following simple construction:

Let us say that $a_1, a_2, \ldots, a_n$, s_1 is an instance of SUBSET-SUM.
It is obvious that for the instance of the knapsack problem where we have the sequence $(a_1, a_1), (a_2, a_2), \ldots, (a_n, a_n)$, $s = s_1$ and $t = s_1$, the answer is the same as for the original instance of SUBSET-SUM.

SUBSET-SUM

If we want to study the complexity of problems such as SUBSET-SUM or the knapsack problem, we must clarify what we consider as the size of an instance.

Probably the most natural it is to define the size of an instance as the total number of bits needed for its representation.

We must specify how natural numbers in the input are represented – if in binary (resp. in some other numeral system with a base at least 2 (e.g., decimal or hexadecimal) or in unary.

- If we consider the total number of bits when numbers are written in **binary** as the size of an input, no polynomial time algorithm is known for SUBSET-SUM.
- If we consider the total number of bits when numbers are written in **unary** as the size of an input, SUBSET-SUM can be solved by an algorithm whose time complexity is polynomial.

ILP – Integer Linear Programming

Problem ILP (integer linear programming)

Input: An integer matrix A and an integer vector b.

Question: Is there an integer vector x such that $Ax \leq b$?

An example of an instance of the problem:

$$A = \begin{pmatrix} 3 & -2 & 5 \\ 1 & 0 & 1 \\ 2 & 1 & 0 \end{pmatrix} \qquad b = \begin{pmatrix} 8 \\ -3 \\ 5 \end{pmatrix}$$

So the question is if the following system of inequations has some integer solution:

$$\begin{aligned} 3x_1 - 2x_2 + 5x_3 &\leq 8 \\ x_1 + x_3 &\leq -3 \\ 2x_1 + x_2 &\leq 5 \end{aligned}$$

ILP – Integer Linear Programming

One of solutions of the system

$$\begin{aligned} 3x_1 - 2x_2 + 5x_3 &\leq 8 \\ x_1 + x_3 &\leq -3 \\ 2x_1 + x_2 &\leq 5 \end{aligned}$$

is for example $x_1 = -4$, $x_2 = 1$, $x_3 = 1$, i.e.,

$$x = \begin{pmatrix} -4 \\ 1 \\ 1 \end{pmatrix}$$

because

$$\begin{aligned} 3 \cdot (-4) - 2 \cdot 1 + 5 \cdot 1 &= -9 \leq 8 \\ -4 + 1 &= -3 \leq -3 \\ 2 \cdot (-4) + 1 &= -7 \leq 5 \end{aligned}$$

So the answer for this instance is YES.

ILP – Integer Linear Programming

Remark: A similar problem where the question for a given system of linear inequations is whether it has a solution in the set of **real** numbers, can be solved in a polynomial time.

PSPACE-Complete and EXPTIME-Complete Problems

- A problem P is PSPACE-**hard** if for every problem P' from PSPACE there is a polynomial time reduction of P' to P.
- A problem P is PSPACE-**complete** if it is PSPACE-hard and belongs to PSPACE.

- A problem P is EXPTIME-**hard** if for every problem P' from EXPTIME there is a polynomial time reduction of P' to P.
- A problem P is EXPTIME-**complete** if it is EXPTIME-hard and belongs to EXPTIME.

$\vdots$

PSPACE-Complete and EXPTIME-Complete Problems

Generally, for arbitrary complexity class $\mathcal{C}$ we can introduce classes of $\mathcal{C}$-**hard** and $\mathcal{C}$-**complete** problems:

Definition

- A problem P is $\mathcal{C}$-**hard** if for every problem P' from the class $\mathcal{C}$ there is a polynomial time reduction of P' to P.
- A problem P is $\mathcal{C}$-**complete** if it is $\mathcal{C}$-hard and belongs to the class $\mathcal{C}$.

So in addition to NP-complete problems, we have PSPACE-complete problems, EXPTIME-complete problems, EXPSPACE-complete problems, 2-EXPTIME-complete problems, ...

Generally speaking, $\mathcal{C}$-complete problems are always the hardest problems in the given class $\mathcal{C}$.

PTIME-Complete Problems, NL-Complete Problems, ...

Remark: The notions of $\mathcal{C}$-hard and $\mathcal{C}$-complete problems defined as above, where a polynomial time reductions are used, do not make much sense for the class PTIME and other classes, which are subsets of this class (such as NLOGSPACE).

For such classes, the notions of $\mathcal{C}$-hard and $\mathcal{C}$-complete problems are defined in a similar way as before but instead of polynomial time reductions they use so called **logspace reductions**:

- an algorithm performing the given reduction must be deterministic and to have a logarithmic space complexity

For example, the following classes are defined this way:

- PTIME-complete and PTIME-hard problems
- NLOGSPACE-complete and NLOGSPACE-hard problems (they are usually denoted with a shorter name as NL-complete and NL-hard)

An Example of NL-Complete Problem

A typical example of NL-complete problem:

Reachability in a Graph

Input: A directed graph G and two of its nodes s and t.

Question: Is there a path from the node s to the node t in the graph G ?

An Example of a PTIME-Complete Problem

A typical example of PTIME-complete problem:

Circuit Value Problem

Input: An acyclic boolean circuit C consisting of gates and wires, and boolean values $x_1, x_2, \ldots, x_n$ on inputs of this circuit.

Question: Is the value on the output of the circuit C equal to 1 for the given values of inputs?

Examples of PSPACE-Complete Problems

A typical example of a PSPACE-complete problem is the problem of **quantified boolean formulas** (**QBF**):

QBF

Input: A quantified boolean formula of the form

$$\exists x_1 \forall x_2 \exists x_3 \forall x_4 \cdots \exists x_{n-1} \forall x_n : \varphi,$$

where φ is a (standard) boolean formula containing variables $x_1, x_2, \ldots, x_n$.

Question: Is the given formula true?

Examples of PSPACE-Complete Problems

EqNFA

Input: Nondeterministic finite automata $\mathcal{A}_1$ and $\mathcal{A}_2$.

Question: Is $\mathcal{L}(\mathcal{A}_1) = \mathcal{L}(\mathcal{A}_2)$?

Universality of NFA

Input: A nondeterministic finite automaton $\mathcal{A}$.

Question: Is $\mathcal{L}(\mathcal{A}) = \Sigma^*$?

Examples of PSPACE-Complete Problems

EqRE

Input: Regular expressions α_1 and α_2.

Question: Is $\mathcal{L}(\alpha_1) = \mathcal{L}(\alpha_2)$?

Universality of RE

Input: A regular expression α.

Question: Is $\mathcal{L}(\alpha) = \Sigma^*$?

Examples of PSPACE-Complete Problems

Consider the following game played by two players on a directed graph G:

- Players alternately move a pebble on the nodes of graph G.
- In moves they mark those nodes that have been already visited.
- A play starts on the specified node v_0.
- Let us say that the pebble is currently on a node v. The player who is on turn choses a node v' such that there is an edge from v to v' and v' has not been visited yet.
- A player that does not have any possible move, loses, and his/her opponent wins.

Generalized Geografy

Input: A directed graph G with a denoted initial node v_0.

Question: Does the player that plays first have a winning strategy in the game played on the graph G with the initial node v_0 ?

Examples of EXPTIME-Complete Problems

A typical example of EXPTIME-complete problem:

Input: A Turing machine $\mathcal{M}$, a word w, and number k written in binary.

Question: Does the computation of the machine $\mathcal{M}$ on the word w halt in k steps?
(I.e., does the machine $\mathcal{M}$ perform at most k steps in the computation on the word w?)

Examples of EXPTIME-Complete Problems

Other examples of EXPTIME-complete problems are for example generalized variants of games such chess, checkers, or Go, played on a board of an arbitrary size (e.g., on a chessboard of size $n \times n$):

- the input is a position in the given game (e.g., in chess, a particular placement of pieces on a chessboard and information whose player is on turn)
- the question is if the player who is currently on turn, has a winning strategy in the given position

Examples of EXPSPACE-Complete Problems

Regular expressions with squaring are defined similarly as standard regular expressions but in addition to the standard operators $+$, $\cdot$, and *, they can contain unary operator ^{2} with the following meaning:

- α^2 is a shorthand for $\alpha \cdot \alpha$.

The following two problems are EXPSPACE-complete:

Input: Regular expressions with squaring α_1 and α_2.
Question: Is $\mathcal{L}(\alpha_1) = \mathcal{L}(\alpha_2)$?

Input: A regular expression with squaring α.
Question: Is $\mathcal{L}(\alpha) = \Sigma^*$?

Presburger Arithmetic

An example of a problem that is decidable but its computational complexity is very high:

Problem

Input: A closed formula of the first order predicate logic where the only predicate symbols are $=$ and $<$, the only function symbol is $+$, and the only constant symbols are 0 and 1.

Question: Is the given formula true in the domain of natural numbers (using the natural interpretation of all function and predicate symbols)?

A deterministic algorithm with time complexity $2^{2^{2^{O(n)}}}$ is known for this problem, and it is also known that every nondeterministic algorithm solving this problem must have a time complexity at least $2^{2^{2^{\Omega(n)}}}$.