Overview of the Lectures

- Basics of Complexity Theory.
- The Complexity Class P.
- Nondeterminism and Class NP.
- NP-complete Problems and Polynomial Reductions.
- Optimization Problems and NP-hardness.
- How to deal with Difficult Problems:
  - Approximations.
  - Heuristics.

You have sequenced your genome - what do you do with it?

This is known as *genome analysis* or *sequence analysis*.

At present, most of bioinformatics is concerned with sequence analysis. Here are some of the questions that one might want to answer:

- gene finding
- search a given fragment
- finding repetitions and regularities
- protein 3D structure prediction
- gene function prediction (homology search)
- prediction of important sites in proteins (highly conserved)
- reconstruction of phylogeny trees (computing evolutionary distance)

Example: methods for reconstruction of phylogenies

- Maximum parsimony: Given a tree, count the number of character changes that must have occurred during evolution if the tree is correct. The most parsimonious tree is the one that postulates the fewest such changes among all possible trees.

- Maximum likelihood: Given a tree and a model of molecular evolution, compute the probability that the sequences actually observed in the extant species have evolved. The maximum likelihood tree is the one for which this probability is largest among all possible trees.

Problem example 1

THE MAXIMUM PARSIMONY PROBLEM

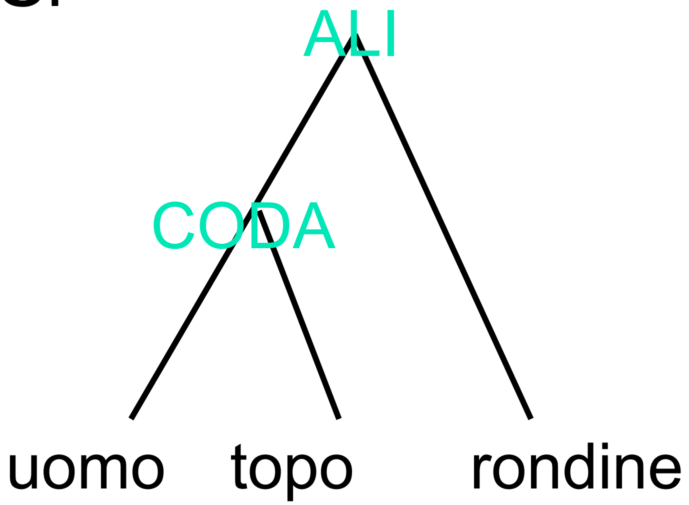
- INPUT: A character matrix C (m rows, one per each species, and n columns, one per each character: C[i,j] describes the values of character j in specie i).
- OUTPUT: The most parsimonious tree among all possible trees (with m leaves and with one character change in each branch).

Example THE MAXIMUM PARSIMONY PROBLEM

- INPUT:

	ALI	CODA
Uomo	No	No
Rondine	Si	Si
Topo	No	Si

- OUTPUT:



Problem example 2

PROBLEM OF SEARCHING A PATTERN IN A (DB OF) SEQUENCE(S)

- INPUT: a (set of) sequence(s) S, a fragment f.
- OUTPUT: All positions in S in which f occurs.

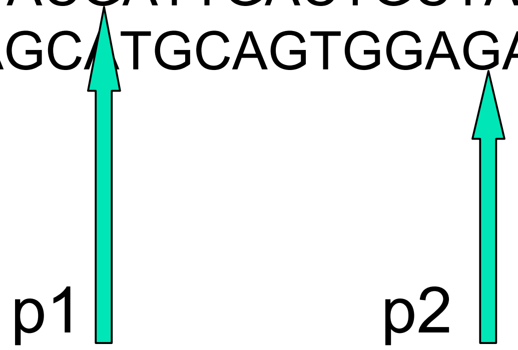
Example

SEARCHING A PATTERN IN A SEQUENCE

- INPUT: f= GATT

  S=TCGATCGTCAGTAACACATACATGAAGACATAGCATAGCATAG
  CATAGCAGTACAGTACGATTGACTGCTAGCTGTACGATAGTCAGT
  CTCCTAGACAGATAGCATGCAGTGGAGATTTTCCAAGTCAAATCA

- OUTPUT:



Problem example 3

PROBLEM OF SEARCHING AN APPROXIMATED PATTERN IN A DB OF SEQUENCES

- INPUT: a (set of) sequence(s) S, a fragment f, a maximum distance d.
- OUTPUT: All positions in S in which f d-approximatively occurs.

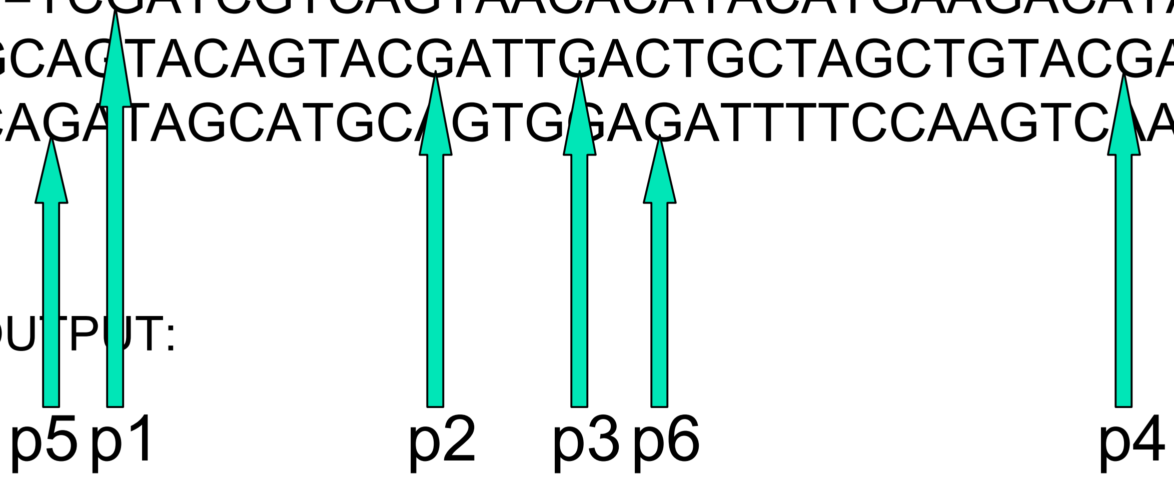
Example

- INPUT: f= GATT,

  maximum Hamming distance (number of substitutions) d=1,

  S=TCGATCGTCAGTAACACATACATGAAGACATAGCATAGCATAGCATA
  GCAGTACAGTACGATTGACTGCTAGCTGTACGATAGTCAGTCTCCTAGA
  CAGATAGCATGCAGTGGAGATTTTCCAAGTCAAATCA

- OUTPUT:



Problem example 4

PROBLEM OF MOTIFS INFERENCE

- INPUT: A set of n sequences s1, s2, …, sn,
  a length L,
  a quorum q.

- OUTPUT: All fragments of length L that occur in at least q of the n sequences.

Example MOTIFS INFERENCE

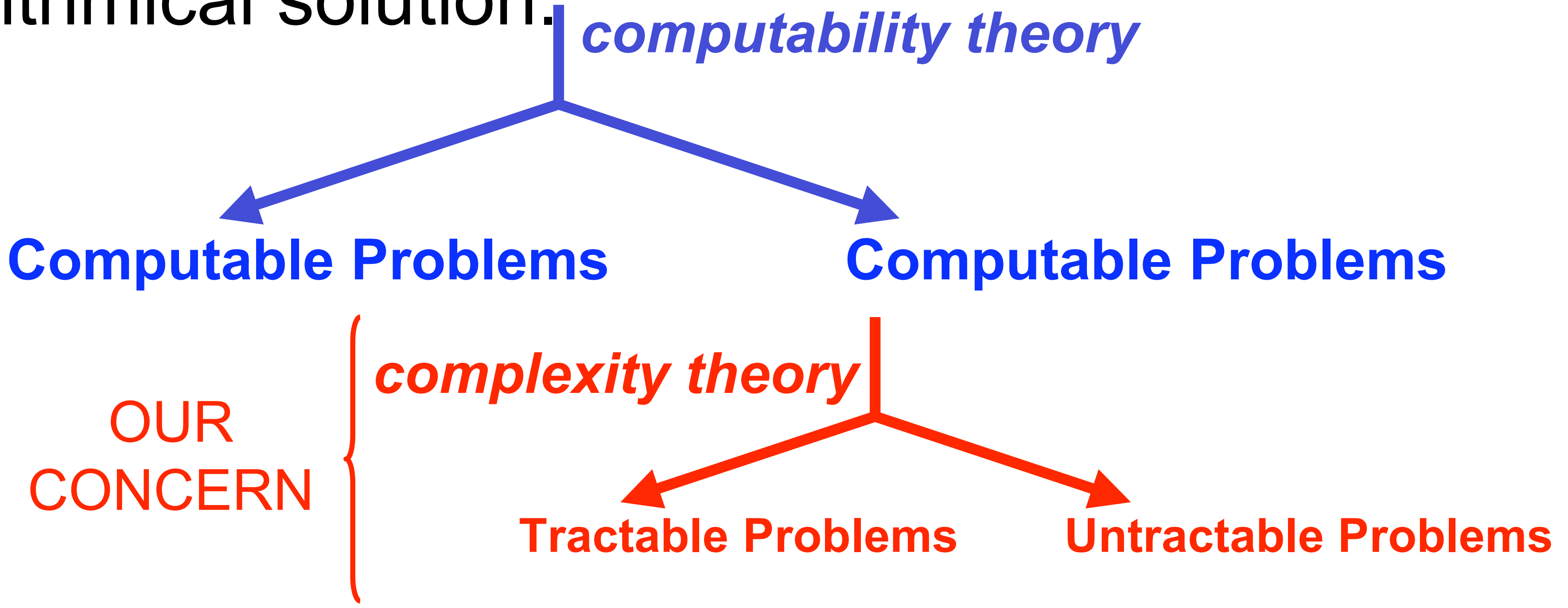
- INPUT: L=6, q=3

  s1=TCGATCGTCAGTAACACATACATGAAGACATAGCATAGCCAGT
  s2=ACAGTACGAACGTCTGCTAGCTGTACGATCGTCAGTCTCCTAG
  s3=ATGCAGTGGAGATCGTCCAAGTCAATGACTGACCGTTACATCA

- OUTPUT: m=GATCGT

Complexity VS Computability

Computational Problems: Problems formulated in a mathematical way, and for which we seek an algorithmical solution.



Complexity Theory

- The theory of Computational Complexity:
  - Classification of problems according to their difficulty.
  - Worst case complexity of the best algorithm.
  - Analisys of tractability of problems according to the availability of resources.

Complexity of a Problem

- The *computational (space or time) cost* of an algorithm that solves a given problem:
  - is expressed in terms of the *size* of the problem itself.
  - represents an *upper bound* to the complexity of the problem: definitely it can be solved in that time/space, but maybe there is a better way...
- The necessary resources to solve a problem represent a *lower bound* to the complexity of the problem: it cannot be done better/faster than that...
- The computational complexity of a problem is determined when the upper bound equals the lower bound.

An example

- Problem: Give me the sum of n given numbers.
  - Problem’s **size**: n.
  - Problem’s **lower bound**: n (at least, I have to read all the numbers).
  - Problem’s **upper bound**:
    - The algorithm that visits each number once and updates a variable with the partial sum requires (“order of”) n operations.
    - The upper bound (given from the algorithm above) is n.
  - Problem’s **time complexity**: n.

[...*in this case we say “linear in the input size”...]*

Cost of an algorithm

- The evaluation of the complexity of an algorithm (hence of a problem), depends from two factors:
  - The adopted **computation model**;
  - The **cost measure**.
- The computational model must be simple and it should not be too far from real machines: the Turing Machine model (taking other models there is a constant multiplicative factor).
- The cost measure for time is “one operation” (and not the actual time spent that depends from technology) making the uniform cost assumption.
  - Actually not all objects have the same size.
  - Actually not all memory access have the same cost.
- The cost measure for space is a byte.

Exercise

- Give me the (lower bound and the upper bound of) the complexity of the following problems:
  - 1. Given an array A of n distinct integers, tell whether there exists an i such that i = A[i].
  - 2. Same as above but A is sorted.
  - 3. Same as above but A has only positive numbers.

Exercise: solution

- 1. Given an array A of n distinct integers, tell whether there exists an i such that i = A[i].
  - Upper and lower bound is n.
- 2. Same as above but A is sorted.
  - Log(n) cause I can do binary search:
    - If i < A[i] I only look left.
    - If i > A[i] I only look right.
- 3. Same as above but A has only positive numbers.
  - O(1) cause I just look A[1]: if A[1]=1 then yes else no.

Asyntotic complexity

- $\Theta(g(n))$ is the set of all functions f(n) for which there exist three positive constant numbers c1, c2, and n0 such that c1 $g(n) \leq f(n) \leq$ c2 g(n) for all $n \geq$ n0.

  … they are functions that behave like g(n) …

- O(g(n)) is the set of all functions f(n) for which there exist two positive constant numbers c and n0 such that $f(n) \leq c\, g(n)$ for all $n \geq$ n0.

  … they are functions that behave at worst like g(n) …

- $\Omega(g(n))$ is the set of all functions f(n) for which there exist two positive constant numvers c and n0 such that $c\, g(n) \leq f(n)$ for all $n \geq$ n0.

  … they are functions that behave at best like g(n) …

- We have that $f(n) = \Theta(g(n))$ if and only if $f(n) = \Omega(g(n))$ and $f(n) = O(n)$.

Asyntotic complexity

- The speed of an algorithm is evaluated with respect to its *asyntotic behaviour*, that is, when the problem's size tends to infinity:
  - A good asyntotic behaviour guarantees that with the growth of the problem's size, we have a *reasonable* growth of the time cost.
  - On the other hand, a bad asyntotic behaviour makes the algorithms applicable to small size instances only.

Some genome's sizes.

HIV2 virus	9671 bp
Mycoplasma genitalis	$5.8 \cdot 10^5$ bp
Haemophilus influenzae	$1.83 \cdot 10^6$ bp
Saccharomyces cerevisiae	$1.21 \cdot 10^7$ bp
Caenorhabditis elegans	10^8 bp
Drosophila melanogaster	$1.65 \cdot 10^8$ bp
Homo sapiens	$3.14 \cdot 10^9$ bp
Some amphibians	$8 \cdot 10^{10}$ bp
Amoeba dubia	$6.7 \cdot 10^{11}$ bp

Polynomial Algorithms

- A general agreement is to classify an algorithm as efficient or inefficient depending on whether its worst case execution time is polynomial or exponential wrt the problem's size.
- For example, given an input size n, we have that $T(n) = n^2 + n \in O(n^2)$ is polynomial and $T(n)=c^n$ is exponential.

n	1	2	3	...	20	...	50
n^2	1	4	9	...	400	...	2500
2^n	2	4	8	...	1048576	...	1073741824

The computational challenge for reconstruction of phylogenies

Maximum parsimony and maximum likelihood require us to consider all possible trees.

Given n species, there are $(2n-3)!/[2^{n-2}(n-2)!]$ distinct possible (rooted) trees for depicting the phylogenetic relationship between these species.

For n = 10, this number is 34,459,425; for n = 15 it is already equal to 213,458,046,676,875.

For this problem, the most obvious solution requires then exponential running time.

Tractability and untractability

- A problem is *untractable* if it is possible to prove that it does not exist a polynomial time algorithm that solves it.
- A problem is *presumibly untractable* if there are no known polynomial algorithms that solve it, but its untractability cannot be proved.
- A problem is *tractable* if it exists a polynomial algorithm that solves it.

Can technology and parallelism help?

- A processor k times faster, or
- A k processor machine.

- If k is a constant things do not change.

Questions

- Is the problem “Give me the maximum among n given numbers” tractable?
- Is the problem “Give me all distinct permutation of n given numbers” tractable?
- Is the problem “Give me the number of all distinct permutations of n given number” tractable?

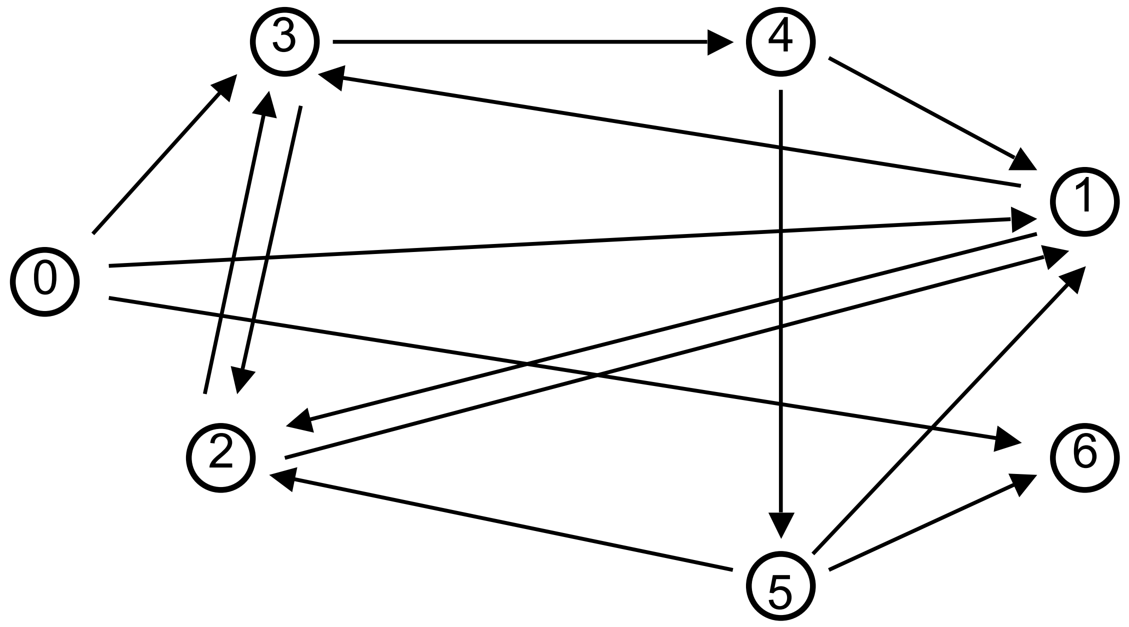
Examples

- The problem of giving all permutations of n numbers is exponential and thus untractable.
  - Many exponential problems are such because its solutions are algorithms that require an exhaustive search.
  - The exhaustive search consists of the generation of all configurations that are (candidate to be) solutions.
- The problem of summing n given number is linear and thus tractable.

The Hamiltonian Path Problem

INPUT: An oriented graph G=(V,A) with n nodes.

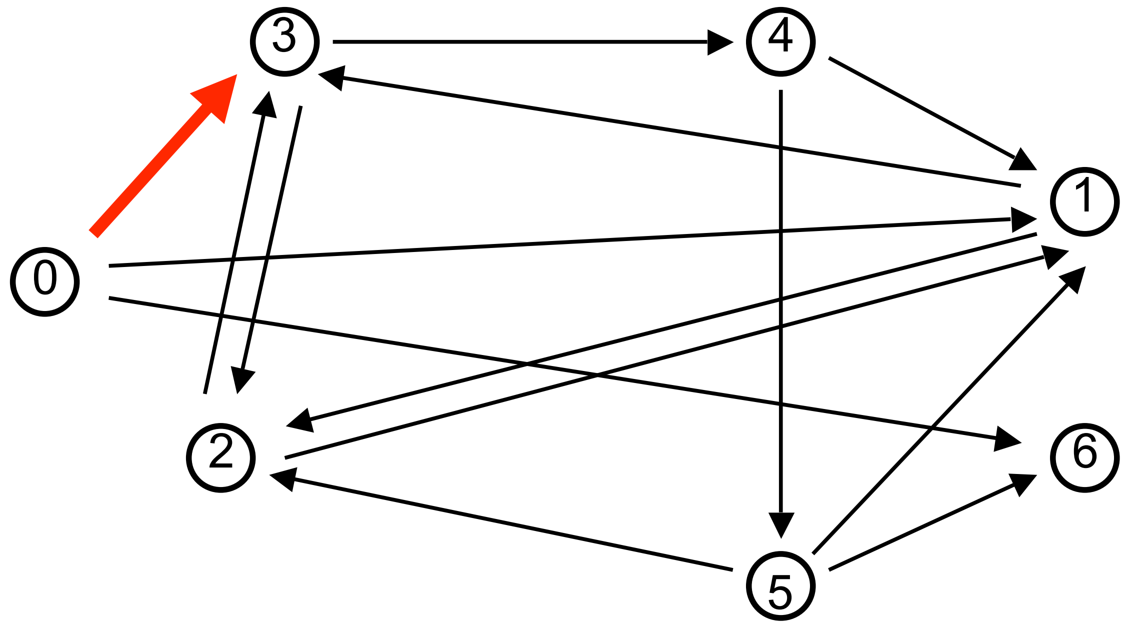
OUTPUT: YES if it exists a path that visits each node exactly once; NO otherwise.



The Hamiltonian Path Problem

INPUT: An oriented graph G=(V,A) with n nodes.

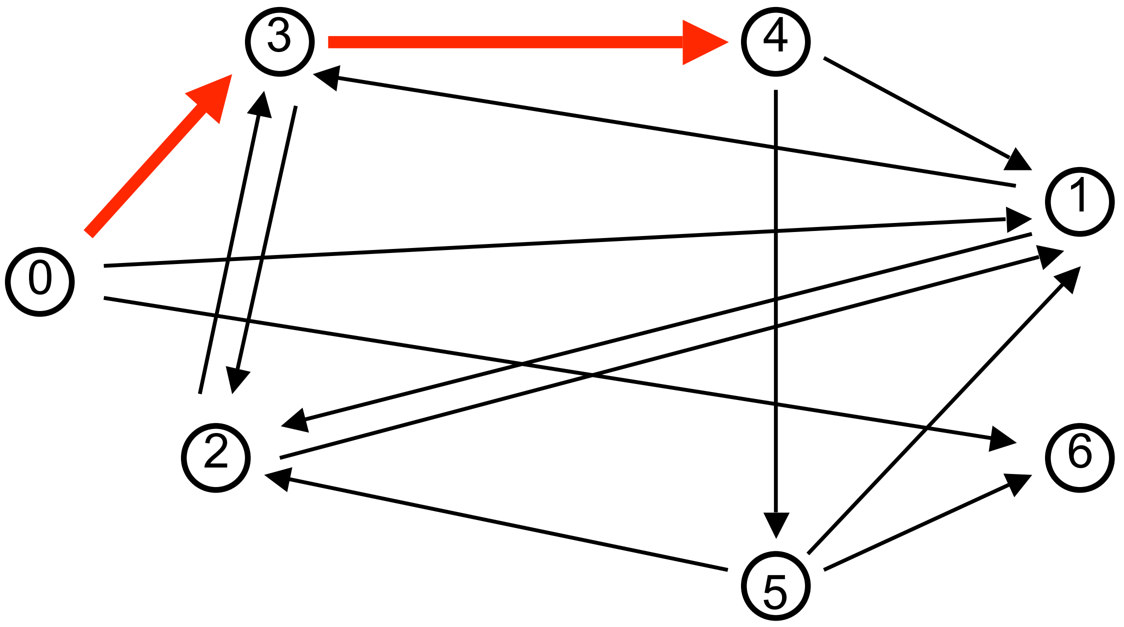
OUTPUT: YES if it exists a path that visits each node exactly once; NO otherwise.



The Hamiltonian Path Problem

INPUT: An oriented graph G=(V,A) with n nodes.

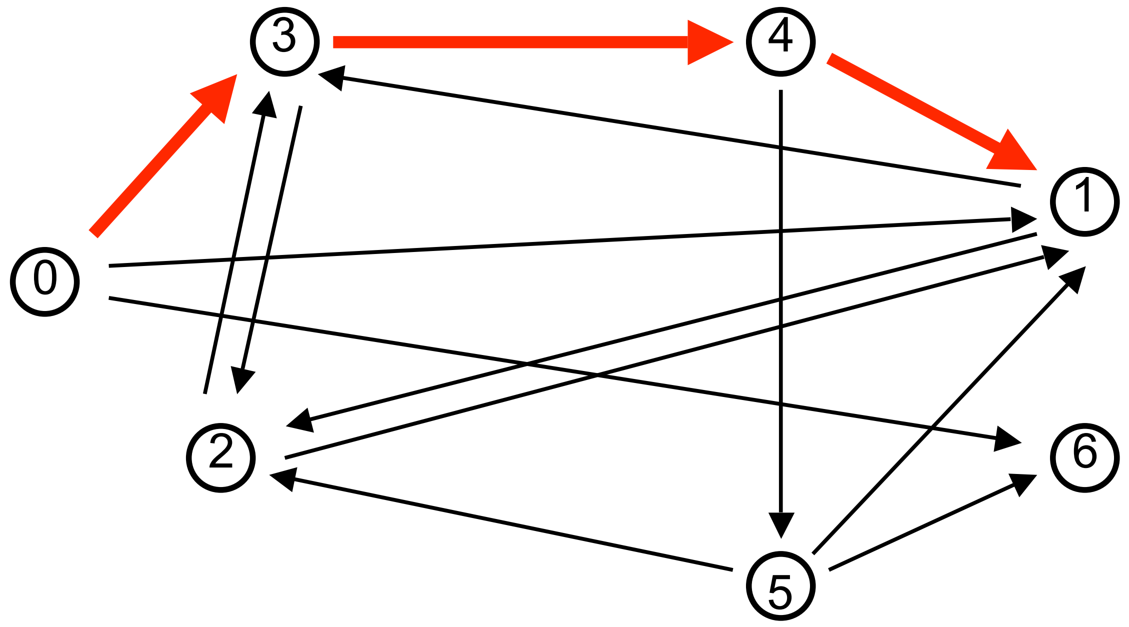
OUTPUT: YES if it exists a path that visits each node exactly once; NO otherwise.



The Hamiltonian Path Problem

INPUT: An oriented graph G=(V,A) with n nodes.

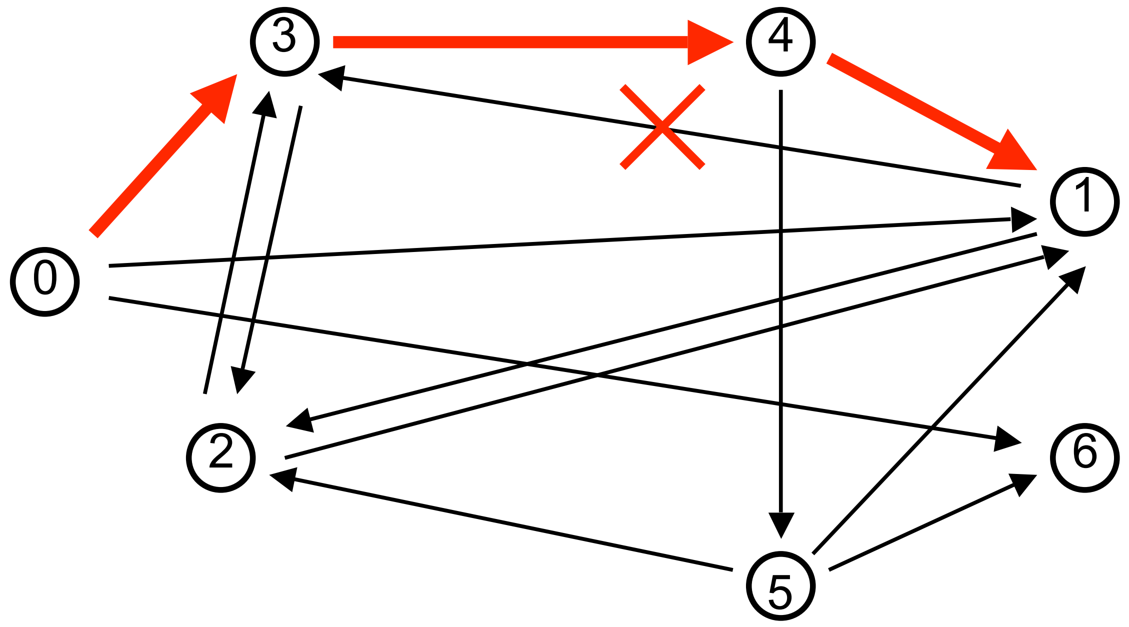
OUTPUT: YES if it exists a path that visits each node exactly once; NO otherwise.



The Hamiltonian Path Problem

INPUT: An oriented graph G=(V,A) with n nodes.

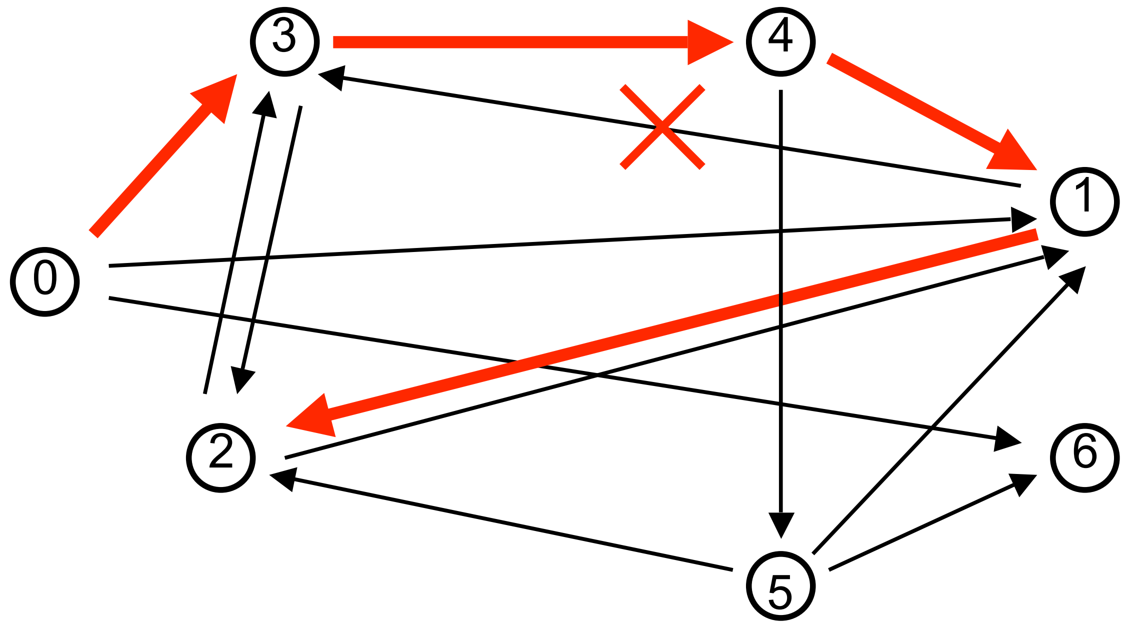
OUTPUT: YES if it exists a path that visits each node exactly once; NO otherwise.



The Hamiltonian Path Problem

INPUT: An oriented graph G=(V,A) with n nodes.

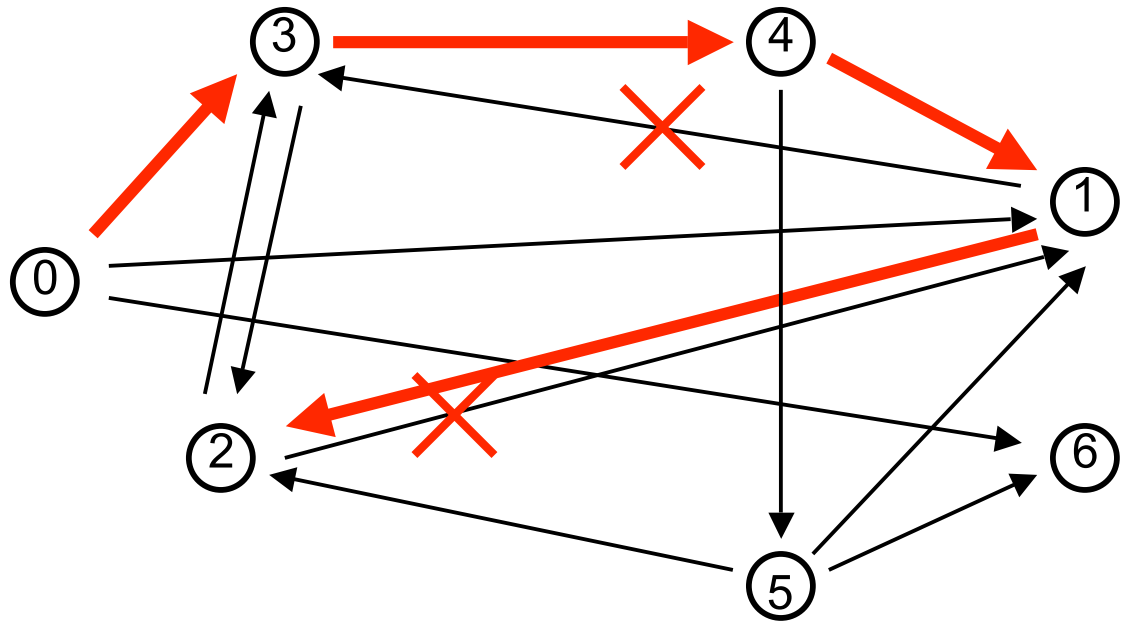
OUTPUT: YES if it exists a path that visits each node exactly once; NO otherwise.



The Hamiltonian Path Problem

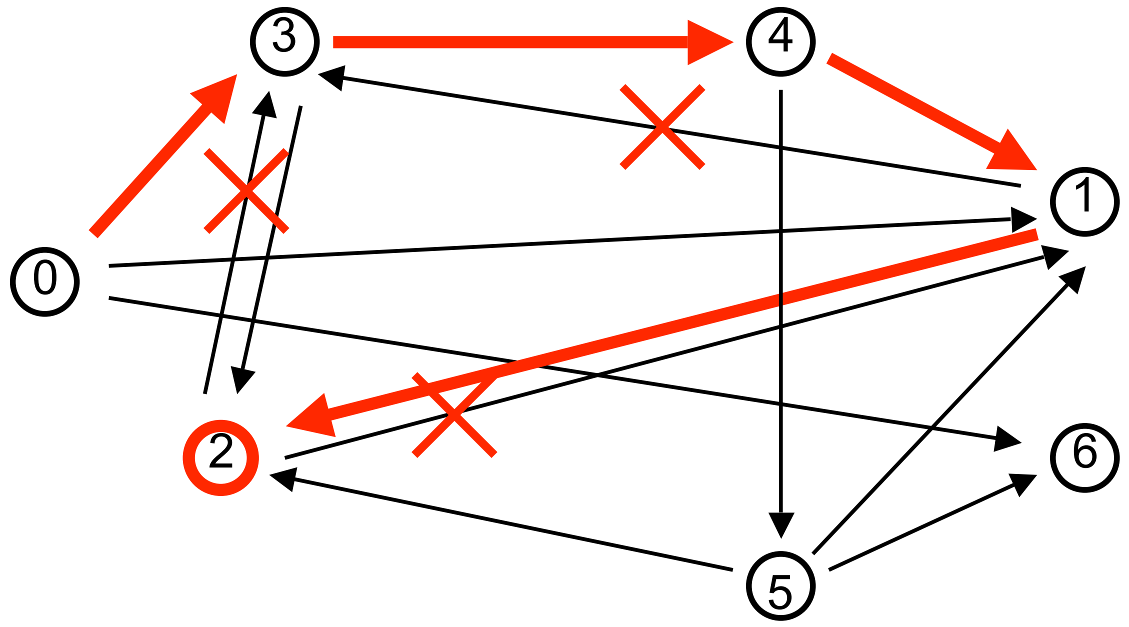
INPUT: An oriented graph G=(V,A) with n nodes.

OUTPUT: YES if it exists a path that visits each node exactly once; NO otherwise.



The Hamiltonian Path Problem

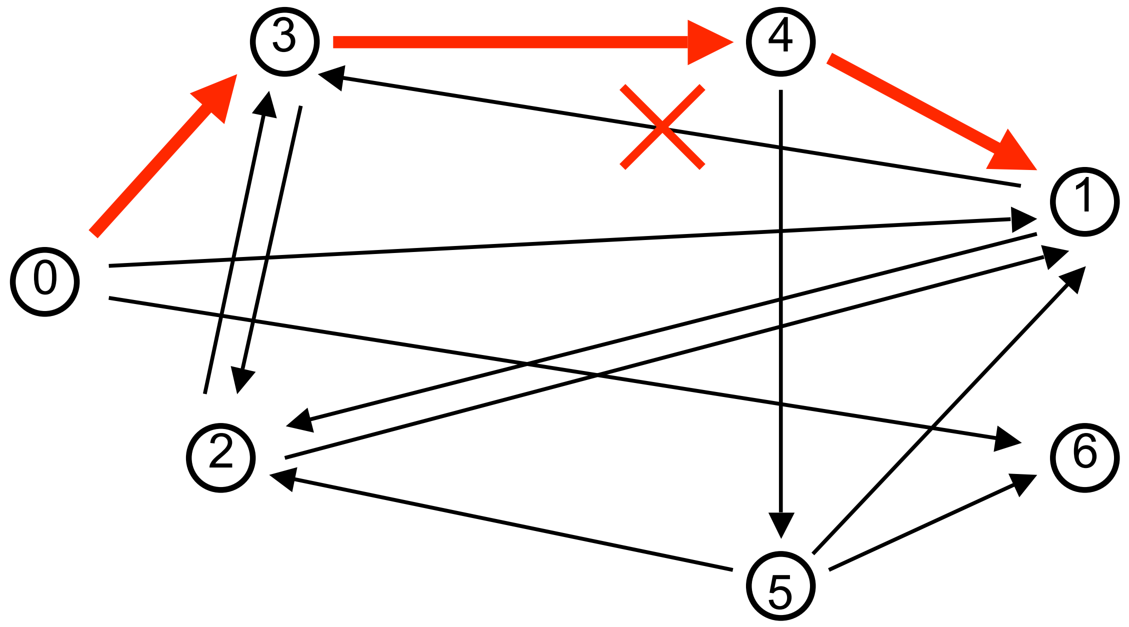
INPUT:	An oriented graph G=(V,A) with n nodes.
OUTPUT:	YES if it exists a path that visits each node exactly once; NO otherwise.



The Hamiltonian Path Problem

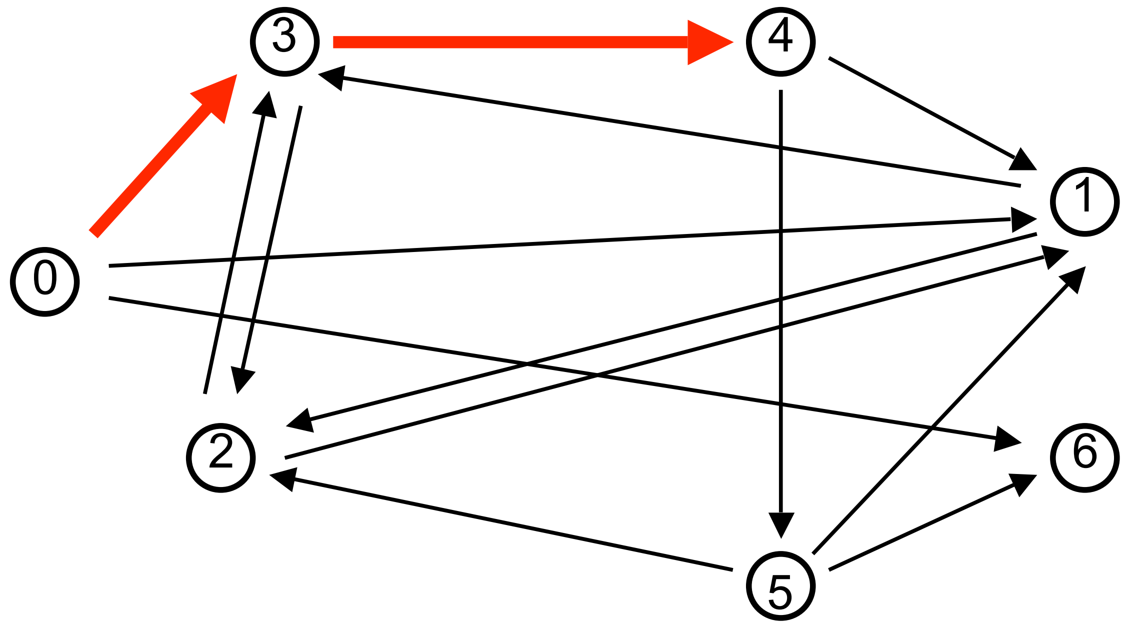
INPUT: An oriented graph G=(V,A) with n nodes.

OUTPUT: YES if it exists a path that visits each node exactly once; NO otherwise.



The Hamiltonian Path Problem

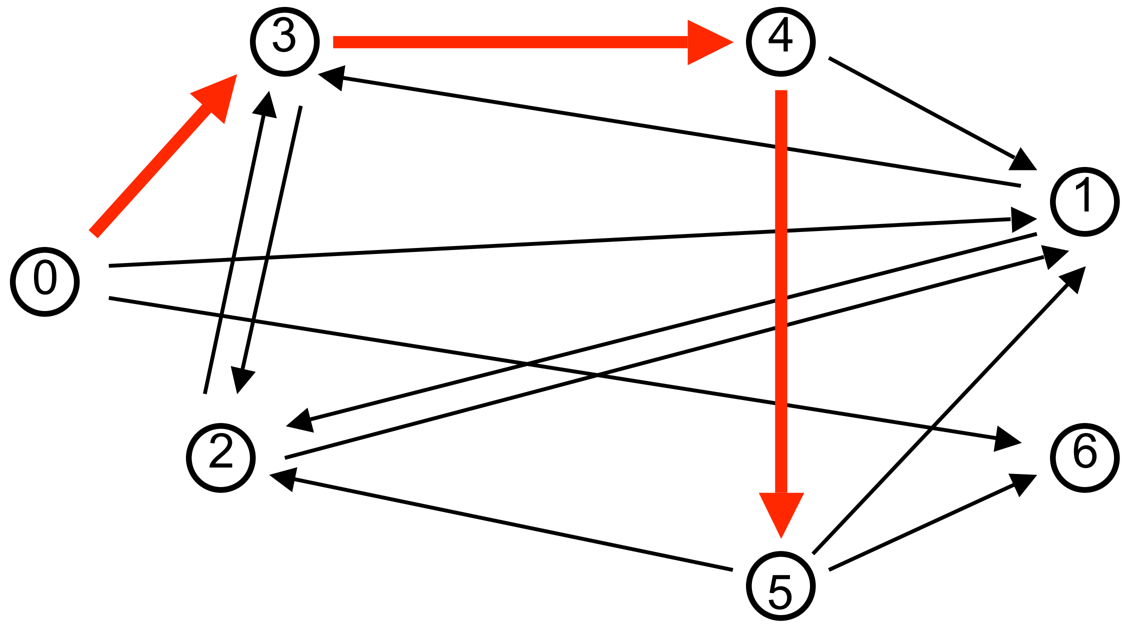
INPUT: An oriented graph G=(V,A) with n nodes.
OUTPUT: YES if it exists a path that visits each node exactly once; NO otherwise.



The Hamiltonian Path Problem

INPUT: An oriented graph G=(V,A) with n nodes.

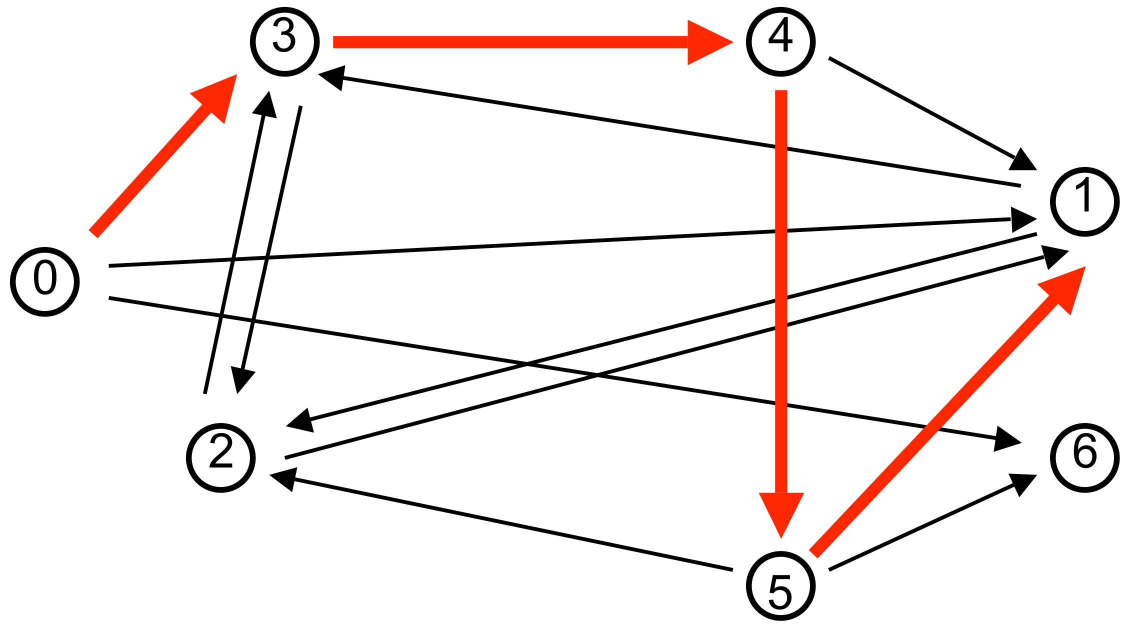
OUTPUT: YES if it exists a path that visits each node exactly once; NO otherwise.



The Hamiltonian Path Problem

INPUT: An oriented graph G=(V,A) with n nodes.

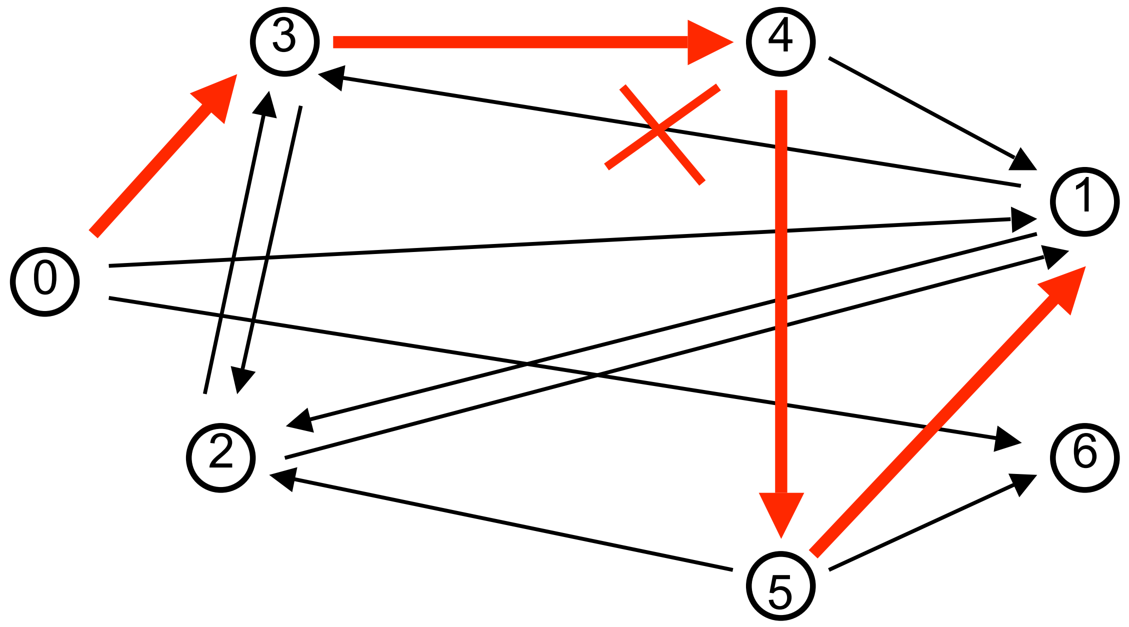
OUTPUT: YES if it exists a path that visits each node exactly once; NO otherwise.



The Hamiltonian Path Problem

INPUT: An oriented graph G=(V,A) with n nodes.

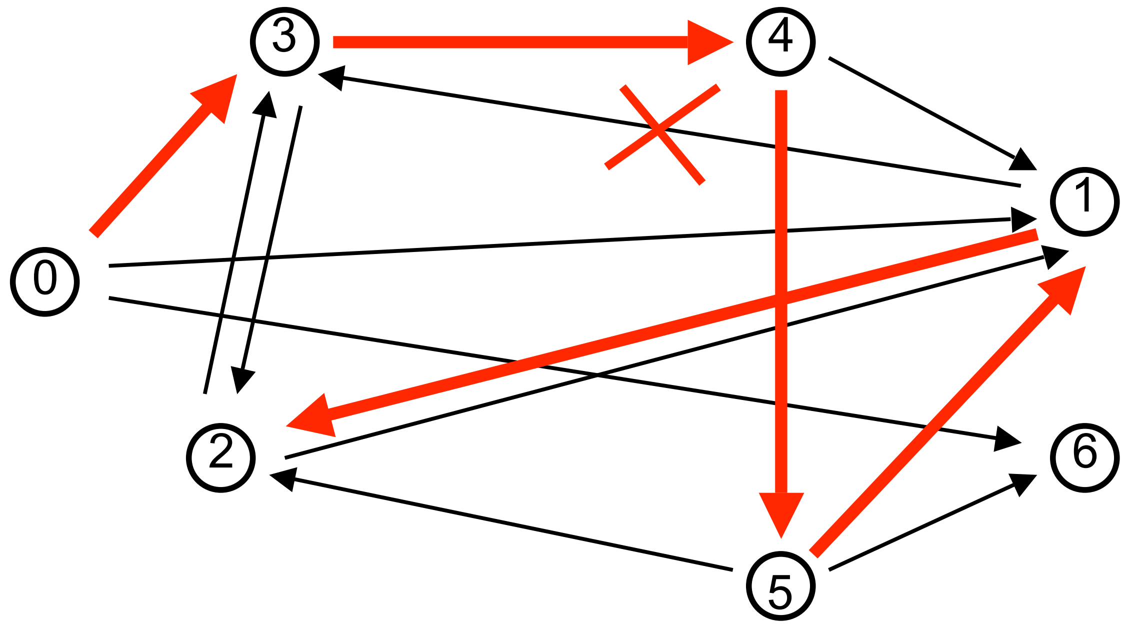
OUTPUT: YES if it exists a path that visits each node exactly once; NO otherwise.



The Hamiltonian Path Problem

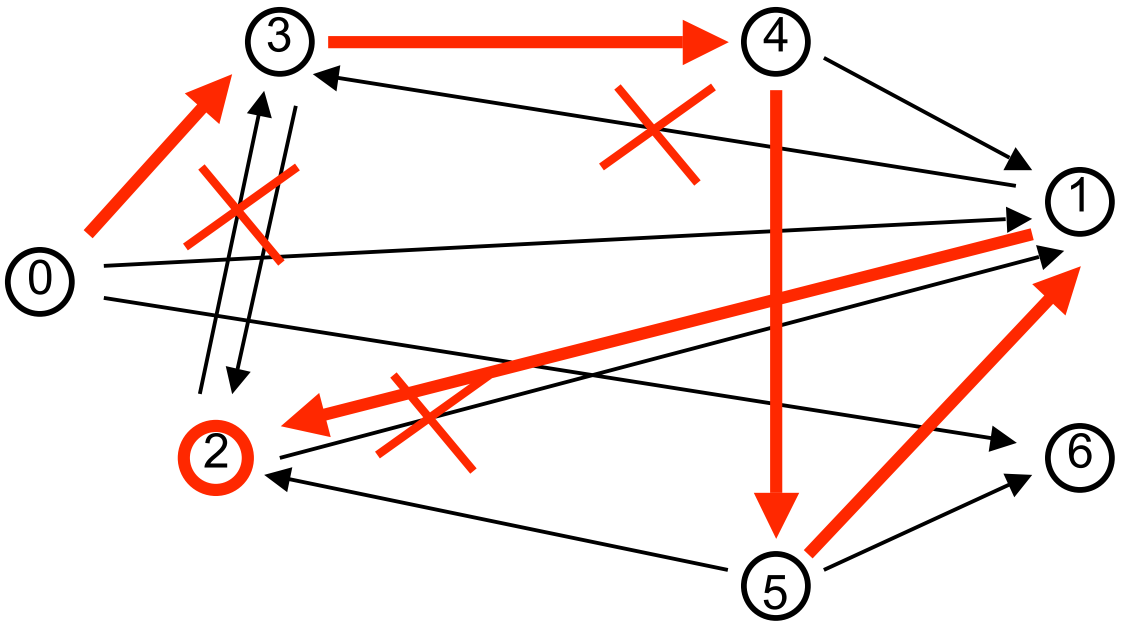
INPUT: An oriented graph G=(V,A) with n nodes.

OUTPUT: YES if it exists a path that visits each node exactly once; NO otherwise.



The Hamiltonian Path Problem

INPUT: An oriented graph G=(V,A) with n nodes.
OUTPUT: YES if it exists a path that visits each node exactly once; NO otherwise.



The Hamiltonian Path Problem

INPUT:	An oriented graph G=(V,A) with n nodes.
OUTPUT:	YES if it exists a path that visits each node exactly once; NO otherwise.

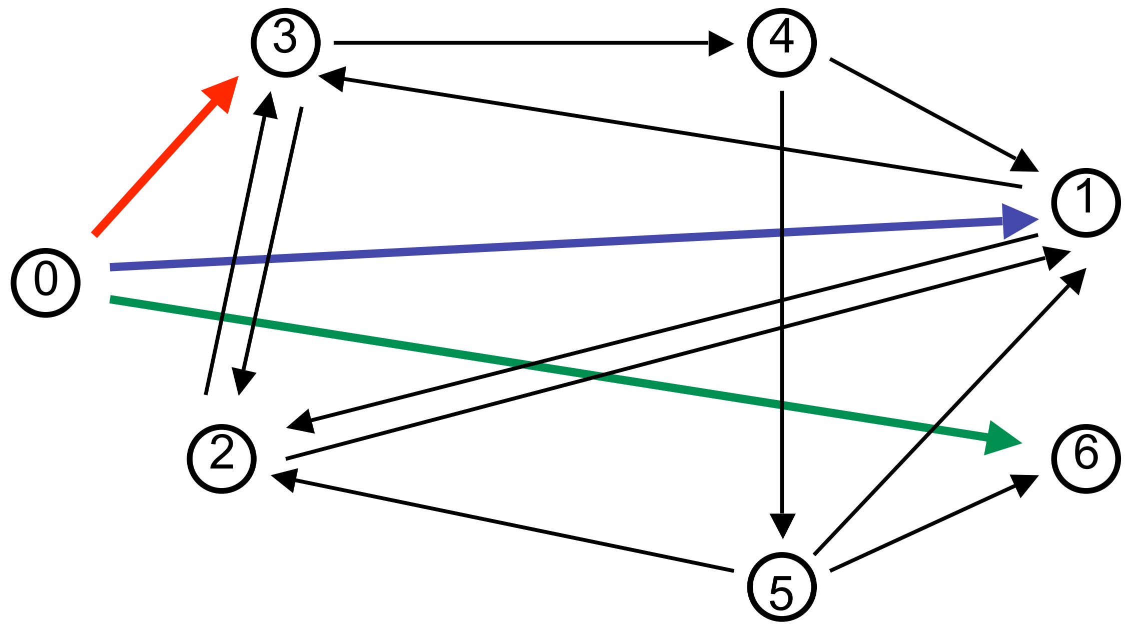
There is no known polynomial algorithm for HPP !!!

$$\Downarrow$$

HPP is presumibly untractable

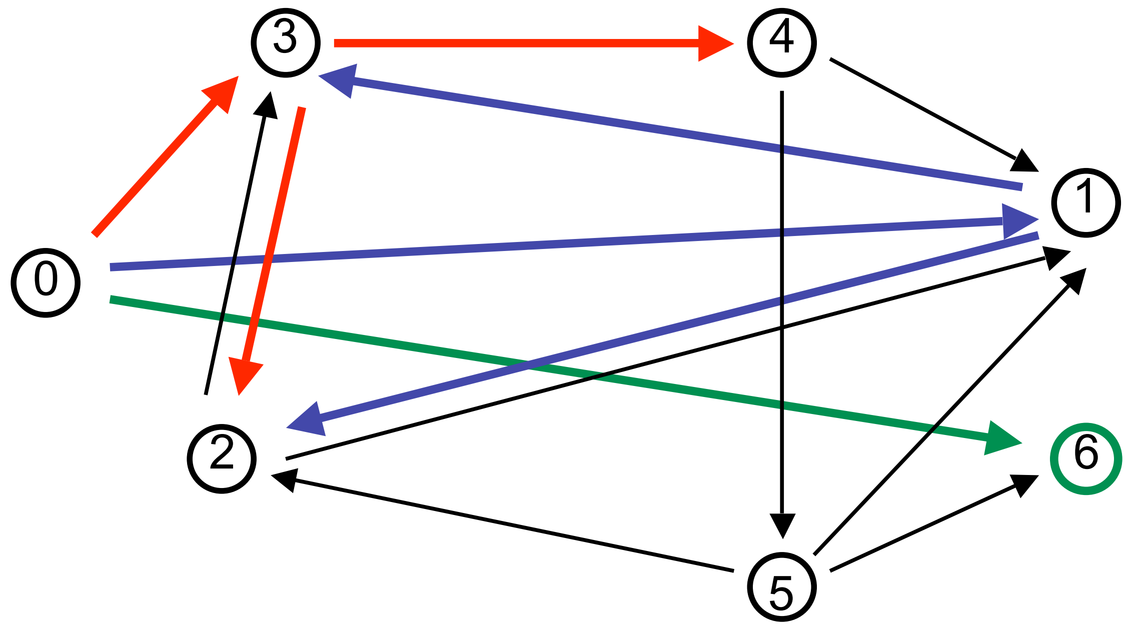
Non determinism

A *non deterministic computation* is a computation that at each step is able to consider simultaneously all possible alternative choices.



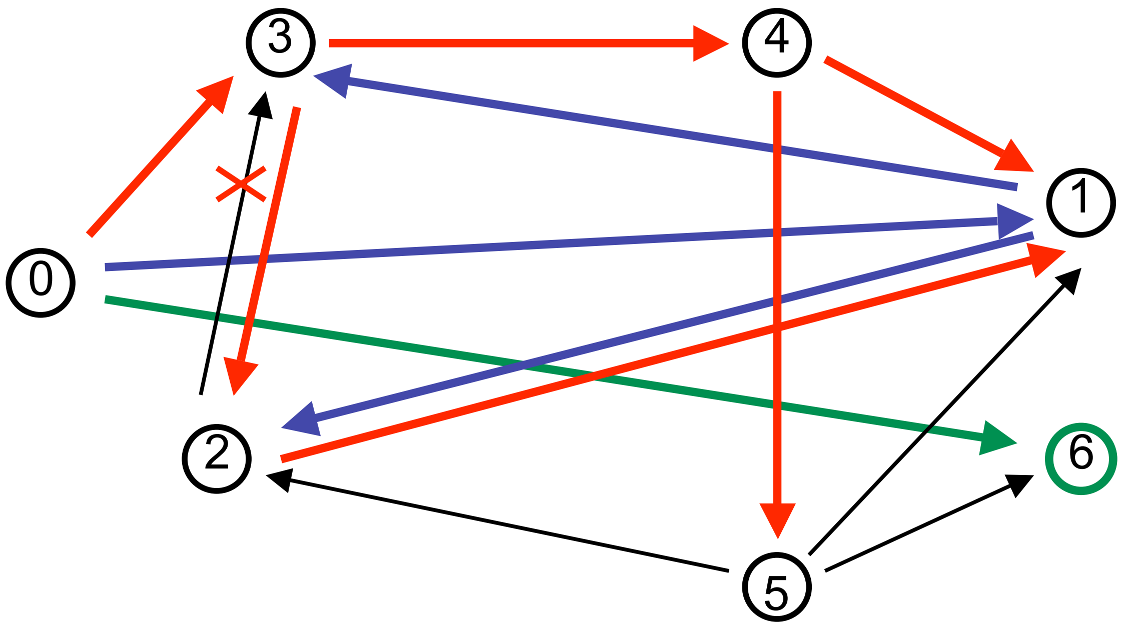
Non determinism

A *non deterministic computation* is a computation that at each step is able to consider simultaneously all possible alternative choices.



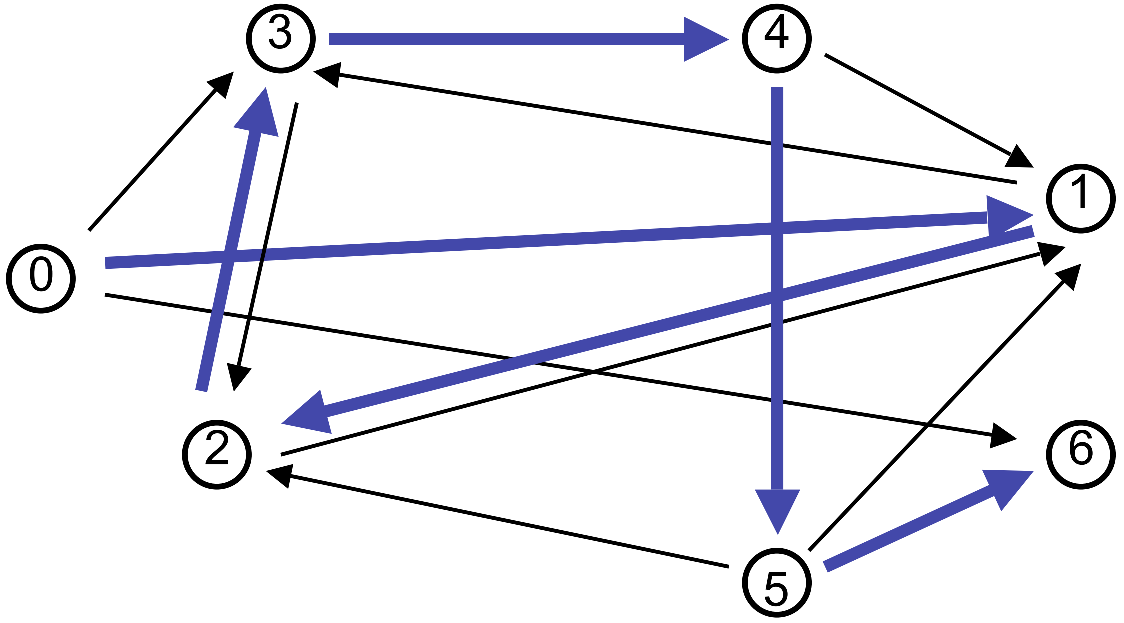
Non determinism

A *non deterministic computation* is a computation that at each step is able to consider simultaneously all possible alternative choices.



Non determinism

A *non deterministic computation* is a computation that at each step is able to consider simultaneously all possible alternative choices.



unfortunately nondeterministic machines do not exist...

Decision Problems

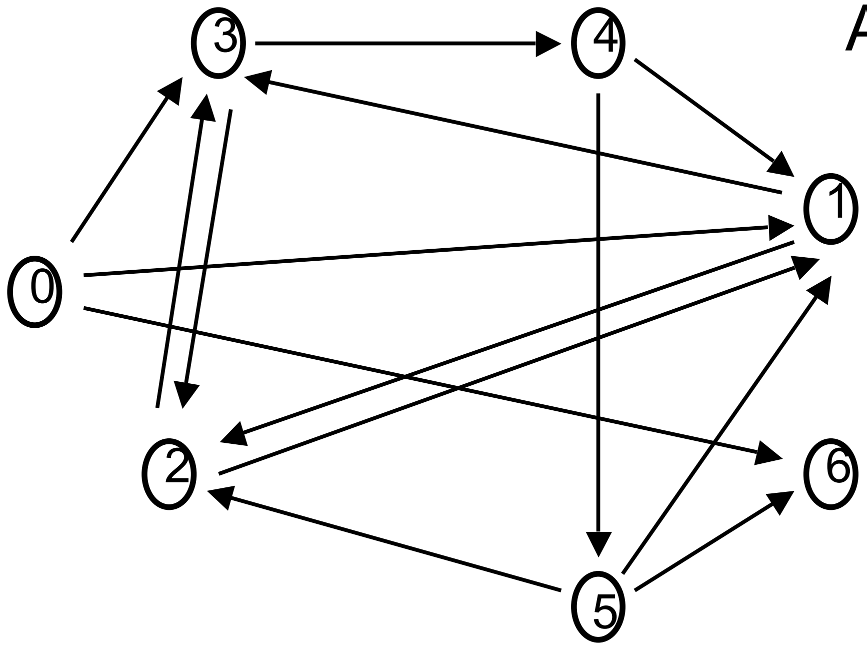
- A decision problem is a problem whose solution is either YES or NO.
  - The HPP is a decision problem.
  - “Is the sum of these n numbers lower than k?” is a decision problem.
- An optimization problem is a problem whose solution is a value (optimal according to a function to maximize or minimize).
  - Finding a (Hamiltonian) path of minimun cost on a weighted graph is an optimization problem.
  - Telling whether there is an Hamiltonian path of cost lower than k is <u>not</u> an optimization problem.

The classes P and NP

- P is the class of all decision problems that admit a deterministic polynomial solution.
  - *"Is the sum of these n numbers < k ?" ∈ P.*
- NP is the class of all decision problem that admit a nondeterministic polynomial solution.
  - HPP ∈ NP.
  - Does HPP belong to P ?
- P ⊆ NP.

Polynomial check

Nobody knows a polynomial deterministic algorithm to find an HP



Assume I give you a path: 0,1,2,3,4,5,6.

Can you check in polynomial time whether it is a HP ?

The class NP: alternative definition

- NP is the class of the decision problems that admit a polynomial check for the solutions.
- Again, it's clear that P $\subseteq$ NP.

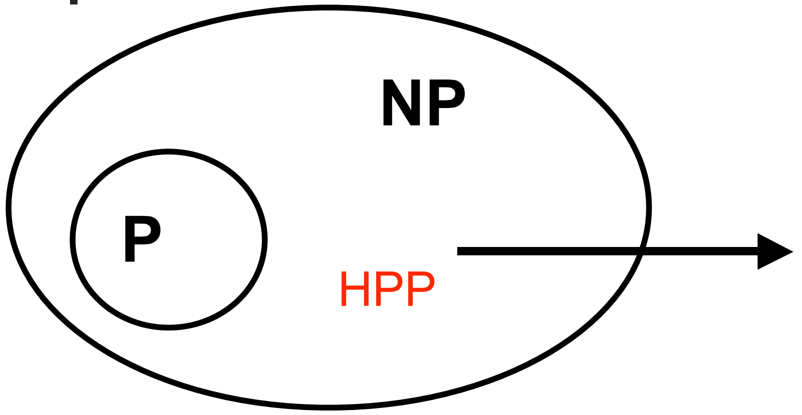
  - P $\neq$ NP ?
  - Is checking easier than finding?

Presumibly yes: problems in NP \ P are assumed untractable

Exercise

- Are the following problems in NP?
  - Given a graph tell me whether there exists a clique.
  - Given a graph tell me the size of the biggest clique.
  - SAT.
  - KNAPSACK.
  - Are these two sequences “similar” (i.e. Their edit distance is at most d)?
  - What is the Hamming distance of these two sequences?

NP \ P



If the conjecture P ≠ NP is correct, then here there must be something.

- HPP is a reasonable candidate. Let’s put it there.
- What else?.