POLITECNICO
DI TORINO

Dipartimento
di Automatica e Informatica

Algorithms and Problem-solving

Paolo Camurati

Algorithm

Muḥammad ibn Mūsā al-Khwārizmī
محمد بن موسى الخوارزمي

Finite sequence of instructions that:

- solve a problem
- satisfy the following criteria:
  - They receive input values
  - They produce output values
  - They are clear, non ambiguous and executable
  - They terminate after a finite number of steps
- work on data structures

Algorithm: al-Khwarizmi, Persian/Uzbek mathematician 11th cent. a.C.

Algorithm vs. procedure

If, for every possible input value, termination is guaranteed in a finite number of steps, then

$\Downarrow$

algorithm

else

$\Downarrow$

procedure

Collatz's Conjecture (1937)

Let function f(n) be defined as:

$$f(n) = \begin{cases} n/2 \text{ for even } n \\ 3n + 1 \text{ for odd } n \end{cases}$$

Given any natural number n, will function f(n) converge to 1 in a finite number of steps?

We are unable to answer!

n = 11

11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

f(11) converges in 15 steps

n = 27

f(27) converges in 111 steps

We can't guarantee that f(n) converges to 1 for all values of n. We have no guarantee that the process terminates, thus it is a precedure, not an algorithm.

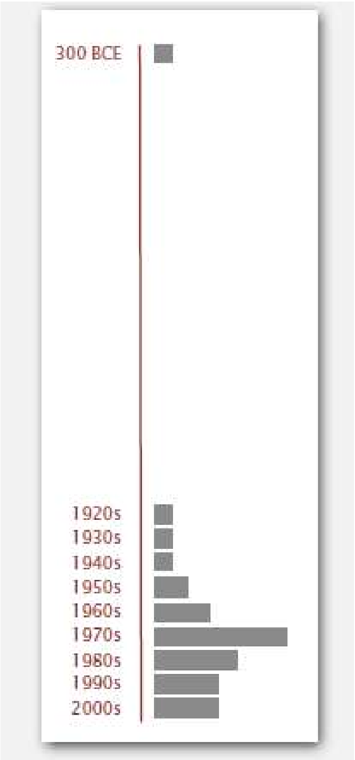
Since we cannot tell in advance how long will it take

```c
#include <stdio.h>
int main() {
  int n;
  printf("Input natural number:  ");
  scanf("%d", &n);

  while (n > 1) {
    printf("%d   ", n);
    if ((n%2) ==1)
      n = 3*n + 1;
    else
      n = n/2;
  }
  printf("%d \n", n);
  return 0;
}
```

Algorithms in History

- Egyptian Multiplication (Rhind and Ahmes papyrus, around 1900 b.C.)
- Greatest Common Divisor (Euclid's algorithm, 4th cent. b.C.)
- Formalization by Church and Turing (20th cent., Thirties)
- Recent developments



Egyptian Multiplications

Assumptions:

- No need for multiplication tables
- Enough to multiply by 2, thus it is easy to compute the powers of 2.

Let *x* and *y* be natural numbers, create 2 columns: the left one with multiples of x (according to the powers of 2), the right one with powers of 2 $\leq$ y.

Find the powers of 2 whose sum is y, the result is the sum of the corresponding rows of the left column.

Example: $x = 18$ $y = 33$

$18 \cdot 33 = 18 + 576 = 594$

Mathematically: $33 = 32 + 1 = 2^5 + 2^0$

$18 \cdot 33 = 18 \cdot (1 + 32) = 18 + 576 = 594$

18	1
36	2
72	4
144	8
288	16
576	32

Greatest Common Divisor

The Greatest Common Divisor *gcd* of 2 non-zero integers *x* and *y* is the greatest among the common divisors of *x* and *y.* Assume that initially *x* > *y.*

Inefficient algorithm based on the decomposition in prime factors:

x = $p_1^{e_1} \cdot p_2^{e_2} \cdot \cdot \cdot p_r^{e_r}$ y = $p_1^{f_1} \cdot p_2^{f_2} \cdot \cdot \cdot p_s^{f_s}$

gcd(x,y) = $\prod p_i^{\min(e_i, f_i)}$

Example: gcd(96,54) = 6

$96 = 2^5 \cdot 3^1$
$54 = 2^1 \cdot 3^3$
gcd(96, 54) = $2^1 \cdot 3^1 = 6$

Euclid’s Algorithm

Recursive! Topic of 2nd year course

Version 1: subtraction

```
if x > y
    gcd(x, y) = gcd(x-y, y)
else
    gcd(x, y) = gcd(x, y-x)
termination:
    if x=y  return x
```

gcd(96, 54) = gcd(42, 54)
gcd(42, 54) = gcd(42, 12)
gcd(42, 12) = gcd(30, 12)
gcd(30, 12) = gcd(18, 12)
gcd(18, 12) = gcd(6, 12)
gcd(6, 12) = gcd(6, 6) = 6

Version 2 Euclid-Lamé (1844)-Dijkstra: remainder of integer division (%)

```
if x > y
    gcd(x, y) = gcd(y, x%y)
termination:
    if y = 0 return x
```

gcd(96, 54) = gcd(54, 42)
gcd(54, 42) = gcd(42, 12)
gcd(42, 12) = gcd(12, 6)
gcd(12, 6) = gcd(6, 0) = 6

Why algorithms?

To solve problems.

Used on computers:

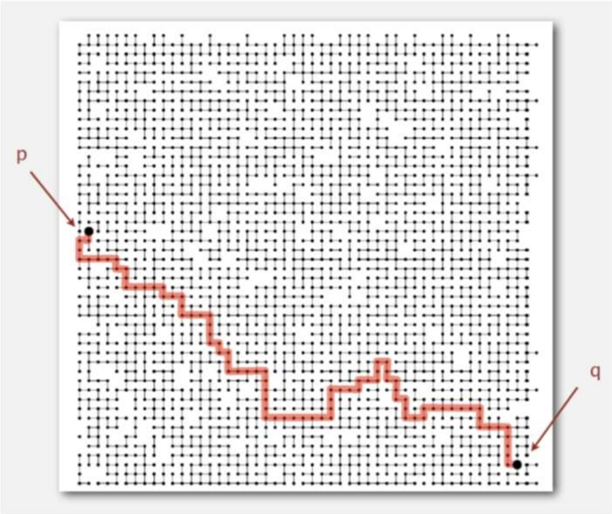
- to increase speed
- to process more data
- to do something impossible otherwise
- to satisfy intellectual curiosity
- to bette program
- to create models
- for fun or for money.

To solve problems in many domains:

- Internet: Web search, packet routing, distributed file sharing
- Biology: human genome
- Computers: CAD tools, file systems, compilers
- Graphics: virtual reality, videographics
- Multimedia: MP3, JPG, DivX, HDTV
- Social Networks: recommendations, news feed, advertisement
- Security: e-commerce, cell phones
- Physics: particle collision simulation …

To do something otherwise impossible:

- are the two dark dots connected in this network (network connectivity)?



To better program:

"I will, in fact, claim that the difference between a bad programmer and a good one is whether he considers his code or his data structures more important. Bad programmers worry about the code. Good programmers worry about data structures and their relationships."

— Linus Torvalds (creator of Linux)

To create models:

- in many sciences computational models are replacing mathematical ones

$$E = mc^2$$

$$F = ma \qquad F = \frac{Gm_1 m_2}{r^2}$$

$$\left[-\frac{\hbar^2}{2m}\nabla^2 + V(r)\right]\Psi(r) = E\,\Psi(r)$$

```
for (double t = 0.0; true; t = t + dt)
   for (int i = 0; i < N; i++)
   {
      bodies[i].resetForce();
      for (int j = 0; j < N; j++)
         if (i != j)
            bodies[i].addForce(bodies[j]);
   }
```

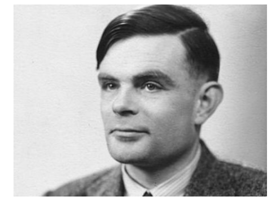
Mathematical formulae　　　　Computational Model

The computational model

- Who or what executes an algorithm?
  - Human being
  - Machine
- Are there limits on the power of the machines we can build?
- Does a universal model of computation exist?

The Turing Machine

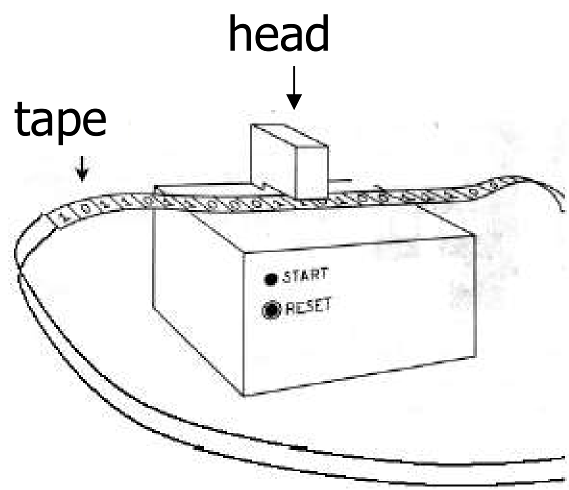
The Turing Machine



The Turing Machine is a model that, given a function f and an *input*, when computation is over, returns the corresponding *output*.

It is defined as:

- A tape
- A head
- An internal state
- A program
- An initial state

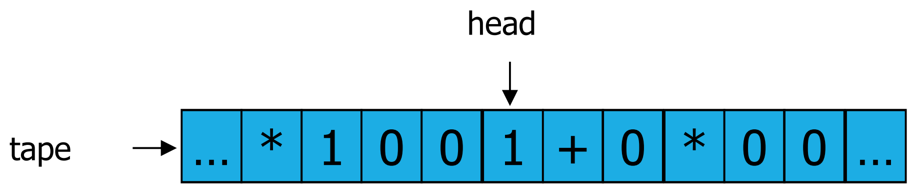


The tape:

- Stores input, output and intermediate results
- Has infinite length and is divided in cells
- Each cell contains a symbol $\in$ alphabet

The head:

- Points to a cell on the tape
- Reads or writes the current cell
- Moves 1 cell to the right or to the left



The internal state is the configuration of the machine as a function of the current input and of the past «history».

The machine, as a function of the current state and input, writes a value on the tape and moves the head to the left or to the right (program).

The Church-Turing Thesis(1936)

«*The Turing Machine can compute any function that may be computed by a physically harnessable machine*».

Thesis, not theorem, because it is a statement on the physical world not subject to proof, but in 80 years no counterexamples have been found.

All computational models found so far are equivalento to the Turing Machine.

Problem-solving

It an activity of thought that:

- An organism or an Artificial Intelligence device put in place to reach a desired condition starting from an initial condition

It is highly creative, design-oriented.

An Approach to problem-solving

1. Problem analysis:
   Reading specifications, understanding the problem, identification of the known class of problems the current one belongs to
2. Methodology identification:
   selection among known algorithmic paradigms (incremental, divide and conquer, dynamic programming, greedy, etc.)
3. Approach selection:
   Selection of the best approach in terms of complexity analysis

4. Decomposition in subproblems:
   identification of subproblems and of their interaction in view of a modular approach
5. Definition of the resolving algorithm:
   Identification of the sequence of elementary steps, of the data it operates on and demonstration of correctness
6. Critical reflection:
   Identification of critical issues and of possible improvements.

Computational problems

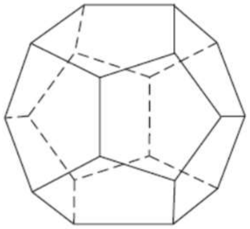
- Formal models associated to a set of questions answered by a computer program processing data
- Set of infinite instances of a problem, each being associated with a solution
- Problem instance: problem operating on specific data.

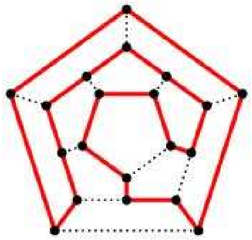
Types of problems

- Decision problems: problems with a yes/no answer
  - given 2 integers x and y, does x exactly divide y?
  - given a positive integer x, is it prime?
  - given a positive integer n, do 2 positive and > 1 integers p and q exist such that n = pq?

Search problems: does a valid solution exist and which one is it? The solution belongs to a possibly infinite space of solutions:

- Hamilton’s game (1859): in dodecahedron (some say an icosahedron), assigning the name of a town to each vertex, find a path that spans all towns, crossing each town once and only once and returning at the town it started from
- Graph Theory: Hamiltonian cycle: given an undirected graph, does a simple cycle spanning all vertices exist? Which one is it?
- which is the k-th prime number
- given an array of integers, sort it in ascending order

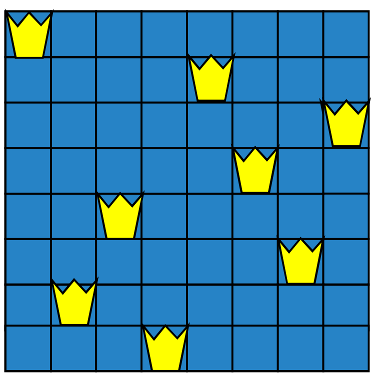




Verification problems: given a solution (certificate), make sure that it is really one:

- Does any queen threaten any other queen?
- The puzzle on the right complies with Sudoku rules (digits from 1 to 9 on rows, columns and 3x3 squares without repetitions)

8 queens



Sudoku

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Optimization problems: if a solution exists, which one is the best one?

- shortest path: given a weighted directed graph, which is the shortest simple path, if it exists, between nodes i and j?
- Travelling Salesman Problem: shortest Hamiltonian cycle.

Optimization problems may have a decision version if we set a limit on valid solutions:

- shortest paths: given a weighted and directed graph, given nodes i and j and a maximum distance d, does a simple path between i and j exist, whose length is ≤ d?

Every optimization problem is at least as hard as its decision version.

Decision problems

They may be:

- decidable (there exists an algorithm that solves them)
  - Example: determine whether a number is prime

Algorithm #1: try the numbers between 2 and at most n, stop:

- either when a factor is found (n%fact==0, n isn't prime)
- or because fact becomes n (n is prime)

```
int Prime(int n) {
  int fact;
  if (n == 1)
    return 0;
  fact = 2;
  while (n % fact != 0)
    fact  = fact +1;
  return (fact == n);
}
```

Algorithm #2: try all numbers between 2 and √n, stop:

- as soon as a factor is found (n%fact==0, n isn't prime)
- or when the loop is over (n is prime)

```
int PrimeOpt(int n) {
  int fact, found=0;
  if (n == 1)
    return 0;
  fact = 2;
  while (fact <= sqrt(n) && found ==0) {
    if (n % fact == 0)
      found = 1;
     else
      fact  = fact +1;
  }
  return found;
}
```

Differences between Algorithm #1 and #2? In the maximum number of steps they might execute:

- Algorithm #1: at most n steps
- Algorithm #2: at most $\lceil\sqrt{n}\rceil$ steps

$\Rightarrow$ They differ in their **COMPLEXITY**!

They may be:

- undecidable (there is no algorithm that solves them)
  - given an algorithm A and data D, both arbitrary, decide whether computation A(D) terminates in a finite number of steps (Turing halting problem, 1937).
  - search for a counterexample to Goldbach's Conjecture (XVII cent.): every even integer greater than 2 is the sum of 2 prime numbers p and q

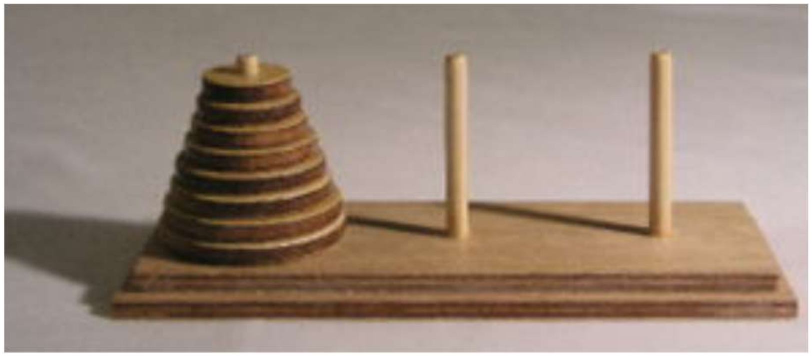
$$\forall\ n \in \aleph, (n > 2) \wedge (n \text{ even})) \Rightarrow (\exists\ p, q \in \wp \in, n = p + q)$$

```
#define  upper 20
```

```
void Goldbach(void) {
  int n = 2, counterexample, p, q;
  do {
    n = n + 2;
    printf("I try for n = %d\n", n);
    counterexample = 1;
    for (p = 2; p <= n-2; p++){
      q = n - p;
      if (Prime(p) == 1 && Prime(q) == 1){
        counterexample = 0;
        printf("%d %d\n", p, q);
      }
    }
  } while (counterexample == 0 && n < upper);
    if (counterexample == 1)
      printf("Counterexample is: %d \n", n);
    else
      printf("Until n= %d none found\n", upper);
  return;
}
```

Decidable decision problems may be:

- tractable, i.e., solvable in "reasonable" time:
  - sort an array of n integers
- intractable, i.e. non solvable in "reasonable" time:
  - The Towers of Hanoi



The Towers of Hanoi (E. Lucas 1883)

- Initial configuration:
  - 3 pegs, 3 disks of decreasing size on first peg
- Final configuration:
  - 3 disks on third peg
- Rules
  - access only to the top disk
  - on each disk only smaller disks
- Generalization: n disks and k pegs.

Class P

Decidable and tractable decision problems

$\Leftrightarrow$

$\exists$ a polynomial algorithm that solves them (Edmonds-Cook-Karp thesis, Seventies)

An algorithm is polynomial iff, working on n data, given a constant $c>0$, it terminates in a finite number of steps upper-bounded by n^c.
In practice c should not exceed 2.

Class NP

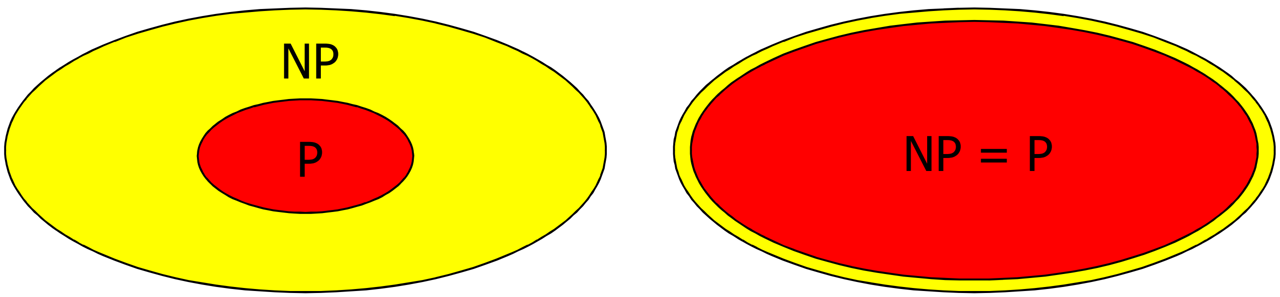
There exist decidable problems for which we have exponential algorithms, but we don’t know any polynomial algorithms.However we can’t rule out the existence of polynomial algorithms

We have polynomial **verification** algorithms, to check whether a solution (certificate) is really such

- Sudoku, satisfyability of a Boolean function

PS: NP stands for Non-deterministic Polynomial and refers to the non-deterministic Turing machine.

P $\subseteq$ NP, but we don't know whether P is a proper subset of NP or it coincides with NP. It is probable that P is a proper subset of NP.



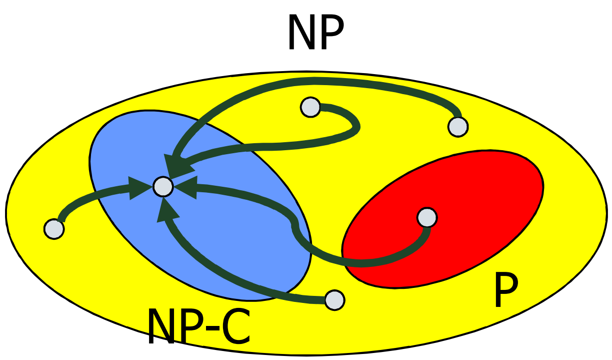
probable

improbable

Class NP-C

A problem is NP-**complete** if:

- it is NP
- any other problem in NP may be reduced to it by means of a polynomial transformation



If we find a polynomial algorithm for any problem in this class, we could find polynomial algorithms for all NP problems, through transformations

HIGHLY UNLIKELY!

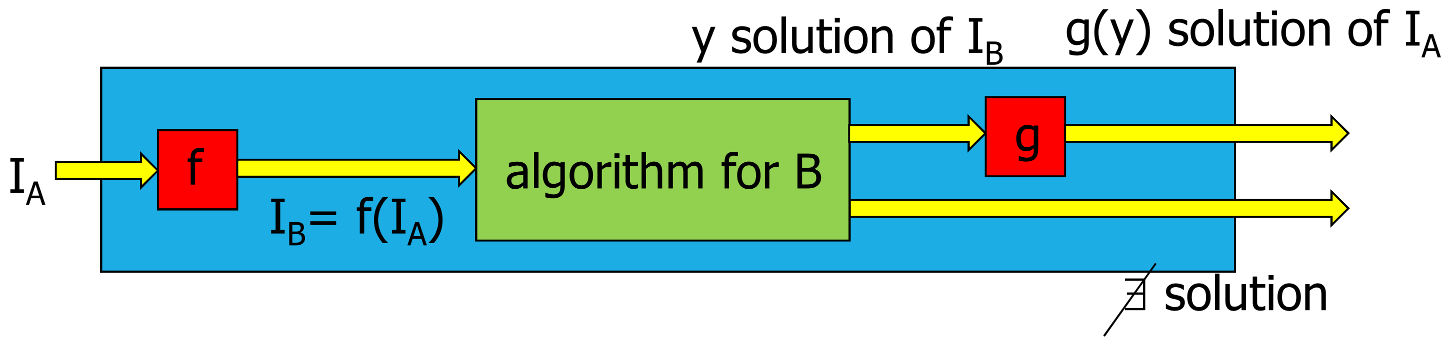
The existence of the NP-C class makes it probable that P $\subset$ NP

Example of NP-C problem: satisfiability

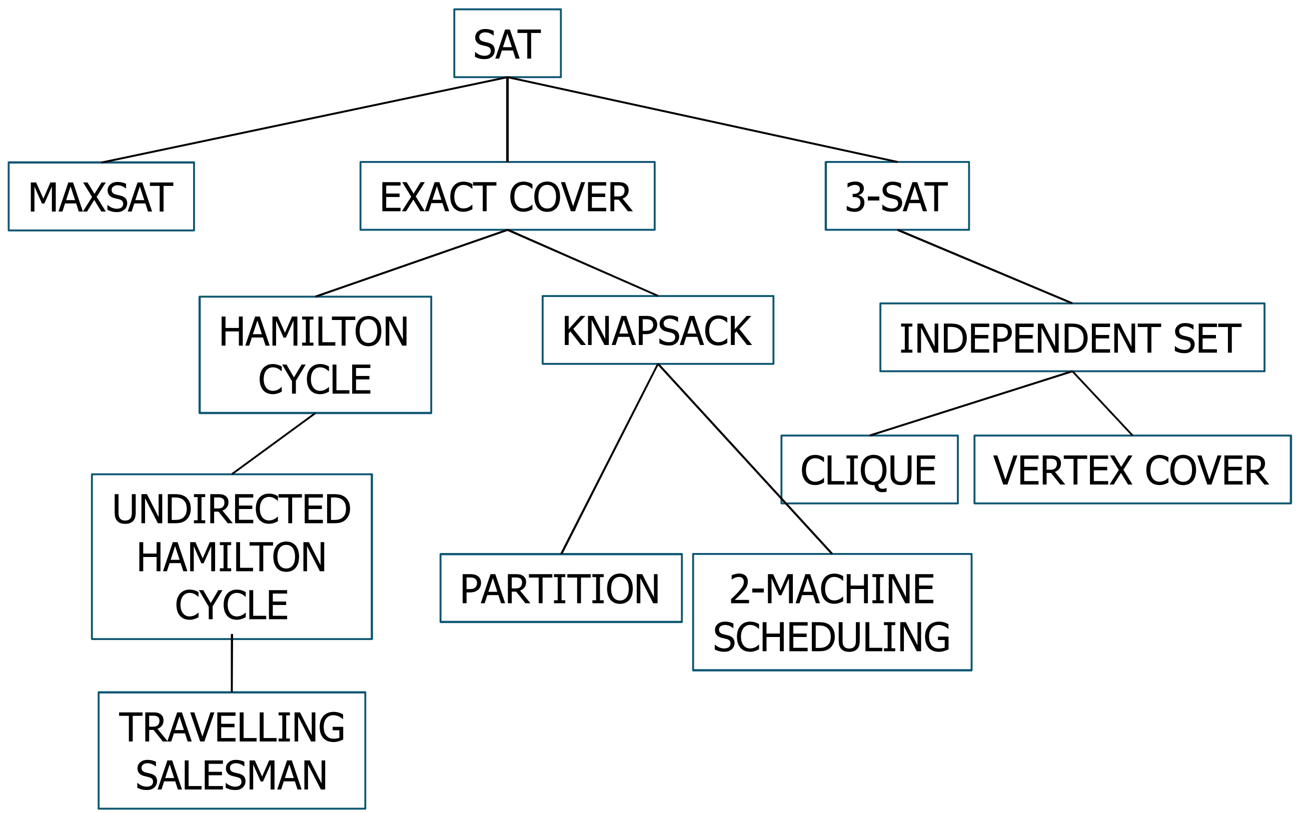
given a Boolean function, find if there exists an assignment to the input variables such that the function is true.

Given 2 decision problems A a B, a polynomial reduction from A to B ($A \leq_P B$) is a procedure that transforms every instance I_A of A into an instance I_B of B:

- with polynomial cost
- such that the answer to I_A is yes iff the answer to I_B is yes



Examples of reductions



The Graph

example: cities,
V="name of the cities"
E="connections between the cities"
In this case it is indirected since we can move in both directions

Definition: G = (V,E) composed of two sets

- V: finite and non eompty set of vertices (containing simple or compound data)
- E: finite set of edges, that define a binary relation on V

Directed/Undirected Graphs:

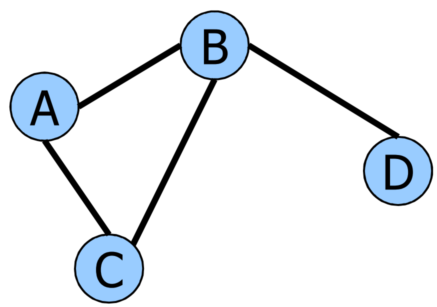
- directed: edge = ordered pair of vertices $(u, v) \in E$ e $u, v \in V$
- undirected: edge = unordered pair of vertices $(u, v) \in E$ e $u, v \in V$

Vertex-cover

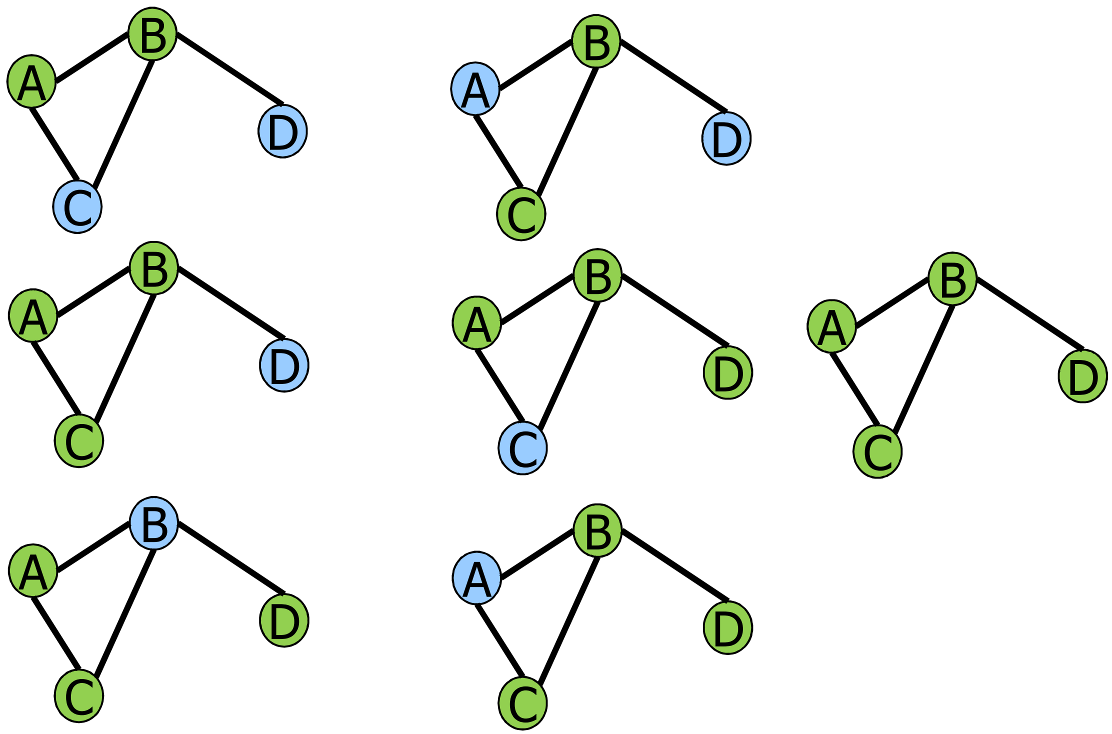
Let G = (V, E) be an undirected graph, a vertex-cover is a subset of vertices $W \subseteq V$ such that for all the edges $(u,v) \in E$ either $u \in W$ or $v \in W$

cover all the vertives

Example

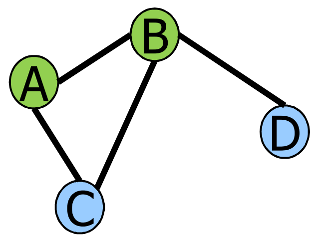


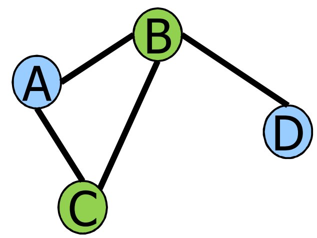
7 solutions



- Search problem: find a vertex-cover
- Optimization problem: find a minimum-cardinality vertex-cover
- Decision problem: does a vertex-cover whose cardinality is $\leq k$ exist?

Decision problem with k = 2
2 solutions





Set-cover

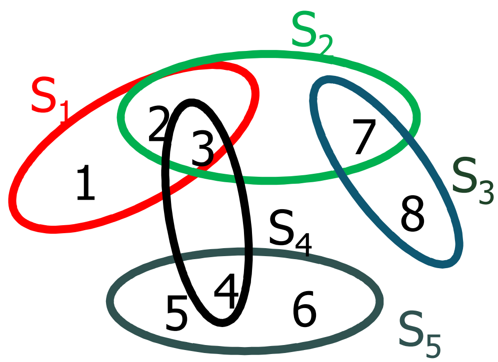
Decision problem:

- let U be a set of elements
- let k be an integer in the range $1 \leq k \leq n$
- $S_1, S_2, \ldots, S_n$ is a collection of subsets of U

does a collection of at most k subsets exist whose union is U?

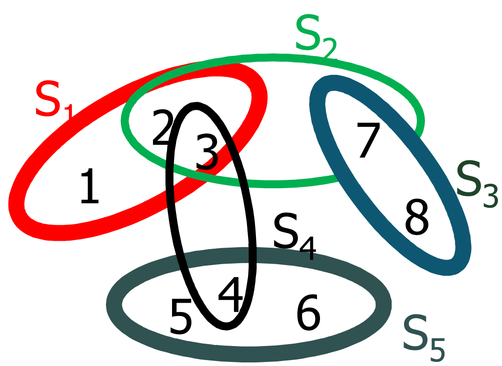
smallest subset such that performing unions evert set is included

Example



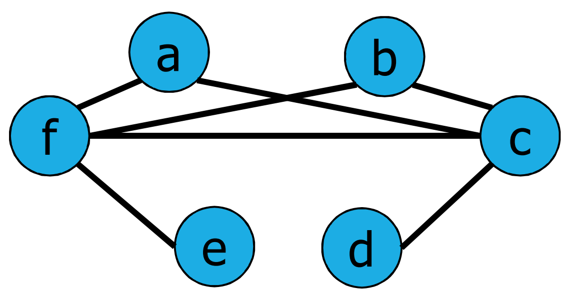
$U = \{1,2,3,4,5,6,7,8\}$
$S_1 = \{1,2,3\}$
$S_2 = \{2,3,7\}$
$S_3 = \{7,8\}$
$S_4 = \{3,4\}$
$S_5 = \{4,5,6\}$

Solution for k = 3



Example

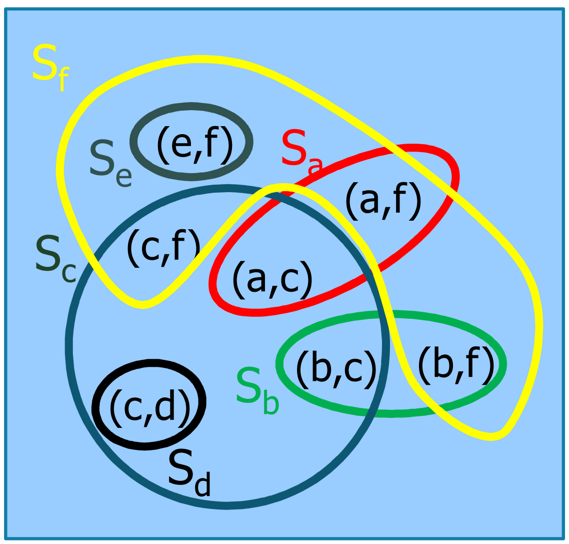
Decision problem: let G = (V, E) be an undirected graph, does a vertex-cover with cardinality $\leq$ 2 exist?



G = (V, E)
V = {a,b,c,d,e,f}
E = {(a,c),(a,f),(b,c)
(b,f),(c,d),(c,f)(e,f)}

vertex-cover ≤P set-cover

Create a set-cover decision problem with U = E and S_i= {edges that insist on vertex i}



$U = \{(a,c),(a,f),(b,c)$
$(b,f),(c,d),(c,f)(e,f)\}$

$S_a = \{(a,c),(a,f)\}$

$S_b = \{(b,c),(b,f)\}$

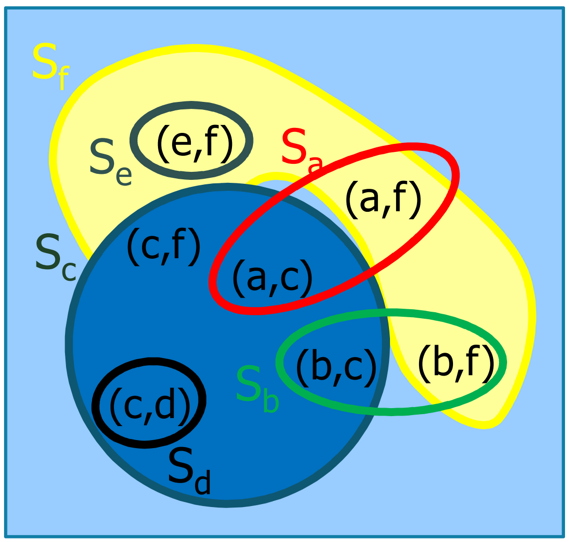
$S_c = \{(a,c),(b,c),(c,d),(c,f)\}$

$S_d = \{(c,d)\}$

$S_e = \{(e,f)\}$

$S_f = \{(a,f),(b,f),(c,f)(e,f)\}$

Solving the set-cover problem



$U = \{(a,c),(a,f),(b,c)$
$\quad (b,f),(c,d),(c,f)(e,f)\}$

$S_a = \{(a,c),(a,f)\}$

$S_b = \{(b,c),(b,f)\ \}$

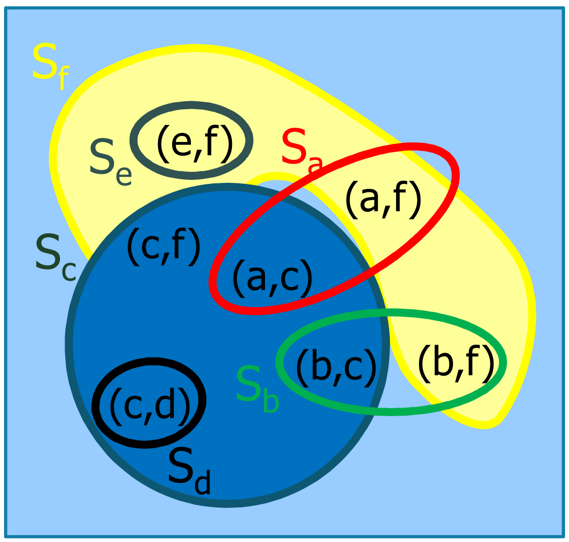
$S_c = \{(a,c),(b,c),(c,d),(c,f)\}$

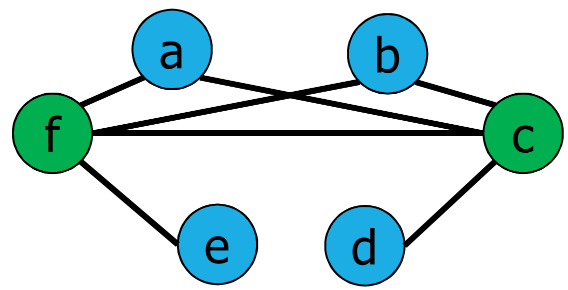
$S_d = \{(c,d)\}$

$S_e = \{(e,f)\}$

$S_f = \{(a,f),(b,f),(c,f)(e,f)\}$

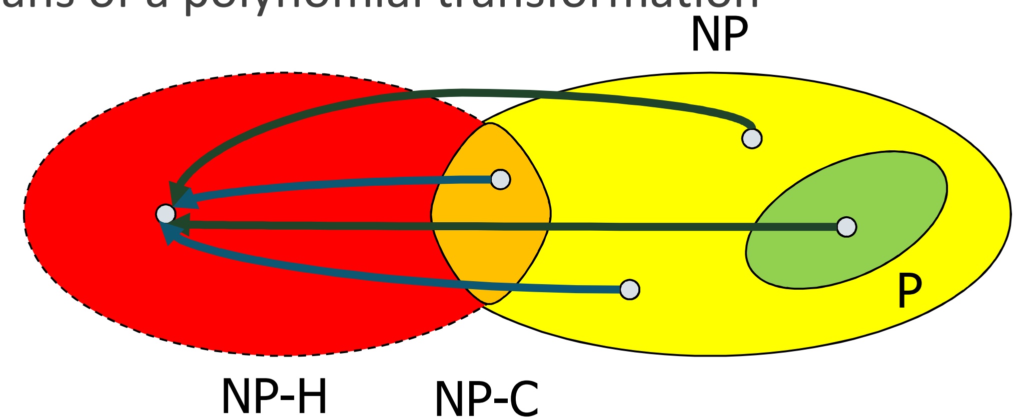
Solving the vertex-cover problem



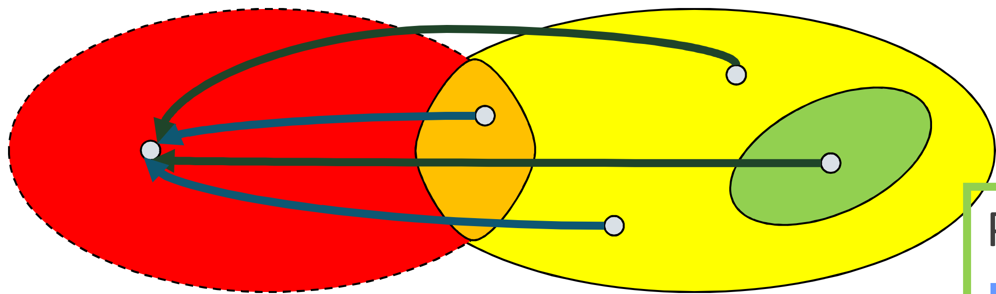


Class NP-H

- A problem is NP-**hard** if every problem in NP may be reduced to it in polynomial time (even if it doesn't belong to NP*)*
  - Any other problem in NP may be reduced to it by means of a polynomial transformation



NP:
- Factorization
- Graph isomorphism



P:
- Graph connettivity
- Primality
- Determinant

NP-H
- Matrix Permanent

NP-C:
- Satisfiability
- Hamilton cycle
- Clique

Problem-solving in this Course

- **Elementary** problem-solving
- Problems in class P
- Classification criteria:
  - Usual approach: based on data structures and/or control statements used
  - Alternative approach closer to the user and less close to the programmer based on:
    - Problem type
    - Application domain: type of information items and problem class
    - Used data structures
    - Algorithmic strategies.

Types of Problems in this Course

- decision:
  - Problems whose answer for each instance is binary.
  - Example: given a natural number $n > 1$, is n prime?
- search:
  - The answer for each instance is a string of bits that represents an information item.
  - Example: given n distinct data, identify among the n! permutations the one that satisfies an ordering relation (< or >). This is the sorting problem.

- verification:
  - Given an instance and a supposed solution (certificate), prove that the certificate is indeed a solution for the instance.
  - Example: let f be a Boolean function and given a variable assignment, prove that f is 1 for that assignment
- selection: filter problems
  - subcase of verification: given solutions and acceptance criterion, partition solutions in 2 subsets: solutions that satisfy the criterion and solutions that do not satisfy it
  - Example: given the transcripts of records of a group of students, identify all and only those students whose average grade is above a given threshold.

- simulation:
  - Interactive representation of reality based on a computational model of a system, developed to analyze its operation
  - Example: given n counters a a sequence of customer arrivals, each customer requiring attention for a known time, estimate waiting time
- optimization:
  - Given an instance a a cost/gain function, select among several solutions the one whose cost is minimum or whose gain is maximum.
  - Example: given a knapsack with known capacity, given objects with weight and value, find the subset of objects that may be contained in the knapsack with maximum value (knapsack problem).

Other classification criteria

- Based on the application domain:
  - numeric
  - mathematical
  - text processing
  - non-numerical data processing
  - etc.

problems

- Based on the nature of primitive data:
  - scalar and/or aggregate data
  - vector data.

- Based on the types of data:
  - numbers (integers or reals)
  - characters (characters or strings)
  - Abstract Data Types
- Based on the statements used:
  - elementary (conditional constructs, simple or nested iterative constructs)
  - advanced constructs and/or functions.

Steps towards the solution

- Non ambiguous and complete problem identification: requires analysis and possibly completing the specs
- Building a formal model for the problem
- Definition of a resolution algorithm and complexity analysis
- Algorithm encoding in a programming language
- Validation of the algorithm and of its implementation on significant instances.

Algorithm = strategy

- Selecting an algorithm is often selecting the best strategy to solve a problem
- Selection is based on knowledge of:
  - elementary constructs and of the library functions supported by the C language
  - language statements (data types, conditional statements, iterative statements)
  - algorithms known from the literature
  - past experience on different kinds of problems.

Strategy

- Most problems solved by programs consist in processing input data to produce output data
- The most important step is processing. It requires:
  - The identification of data (intermediate results)
- Data may be scalar and/or aggregate
  - The formalization of the steps (operations) needed to evaluate intermediate results (starting from other data):
- intermediate steps are often formalized in terms of conditional and/or iterative statements. They can later be modularized by means of functions.

In pratice!

- Experience is a fundamental requisite (as in many other disciplines) to choose a good resolution strategy: an algorithm (a program) is a design!
- Knowledge of problems solved in the literature is a good starting point:
  - **wrong** attitude: solve a problem as if it were seen for the first time, without realizing that work has already been done
  - **right** attitude: one should know and understand underlying theory and be able to apply it.
- Use of available web resources:
  - **wrong** attitude: copy without understanding
  - **right** attitude: take on a problem, then discuss with colleagues.
- Sometimes, when no other tools are available, a good starting point is to analyze the solution to the problem “by hand” or “on paper”.

Data Structures

- The choice of the data structures depends on:
  - The nature of the problem, the input data, the requested results
  - The algorithmic choices
- Choice of a data structure = deciding which (what type) and how many variables are needed to store input, intermediate and output data
- Sometimes the data structure may be selected before defining the algorithm, but often it is necessary to consider data and algorithm at the same time.

Choice of the data structure

- It consists in:
  - identifying the types of information items to represent (input, intermediate and output data): numbers (integers or reals), characters or strings, struct
  - deciding whether it is necessary to collect data in arrays or matrices (countable aggregates) or scalar data (or struct) are enough
- Some problems may be solved with a few statements (conditional constructs) on scalar data.
- Many problems need iterations on data.

Iteration-based Problems and Arrays

- An iteration-based problem doesn't need an array when it is enough to work on the generic i-th data and there is no need to «remember» previous values. Scalar or aggregate data are enough.
  - Example: summing up numbers, searching for the maximum value
- An iteration-based problem needs an array if it necessary to collect and store all data before processing them. Vector data are required.
  - Example: input data and output them in reverse order

Choice of the Algorithm

- Selecting the algorithm (conditional and/or iterative statements) may be very simple (for example when suggested directly by the problem)
  - Example: numeric and mathematical problems
- Else selecting an algorithm may be a true «design», comparing strategies, complexity, evaluating pro’s and con’s
  - Example: activity planning based on optimization criteria.