

GEM 1501 Problem Solving With Computers

Lecture 7:

Inefficiency and Intractability

Frank Stephan

Previous Lecture

- 10.00-10.45h: Midterm Test
- Java Script Arrays
- Input and Output in Java Script
- Optimizing Speed

Moving Array Elements Around

Command	x	y	
var x = 0;	0	-	
var y = new Array(1,2,3);	0	(1,2,3)	
x = y[1];	2	(1,2,3)	<< y[0],y[1],y[2]
y.push(4,5,6,7,8);	2	(1,2,3,4,5,6,7,8)	
y.push(9);	2	(1,2,3,4,5,6,7,8,9)	
x = y.pop();	9	(1,2,3,4,5,6,7,8)	
x = y.shift();	1	(2,3,4,5,6,7,8)	
y.unshift(0,x);	1	(0,1,2,3,4,5,6,7,8)	
x = y.slice(2,4);	(3,4)	(0,1,2,3,4,5,6,7,8)	<< NOTE THE
x = y.splice(2,4);	(3,4,5,6)	(0,1,7,8)	<< DIFFERENCE
var z = x.concat(y);	x,y as above,	z is (3,4,5,6,0,1,7,8)	

Input and Output in Java Script

- Usage of current window (document)
document.write(...); usage of html-commands
- Opening of Additional windows
- Preprogrammed windows

```
do { n = window.prompt("Input a value ",4);  
    k=1; for (m=1;m<=n;m=m+1) { k=k*m; }  
    window.alert("The faculty of "+n+" is "+k); }  
while (window.confirm("Press 'OK' for more computations"))
```

- Forms instead of Writing and Prompting
buttons and other environments in forms
The variable document.forms[i].elements[j].value

Question 1

Numbering System:

Binary	1000001000	is	$1 \cdot 2^8$	plus	$1 \cdot 2^3$	
Ternary	100210	is	$1 \cdot 3^5$	plus	$2 \cdot 3^2$	plus $1 \cdot 3$
Quinary	2023	is	$2 \cdot 5^3$	plus	$2 \cdot 5$	plus $4 \cdot 1$
Decimal	264	is	$2 \cdot 100$	plus	$6 \cdot 10$	plus $4 \cdot 1$

Base Sixty: 10000 Seconds is 2 Hours plus 46 Minutes plus 40 Seconds

Importance for Computing: Many operations are based on binary system.

Space of memory: 1 kbit = 1024 bits, 1 Mbit = 1048576 bits.

Operations & and |: $11001100 \& 10101010 == 10001000$,
 $11001100 | 10101010 == 11101110$.

Programming Languages

- Programming Languages evolve with time; successful languages do no longer require line numbers and strange formatting; all have now for-loops and while-loops
- Certain syntactic characteristics like “:=” in Pascal and “==” in C, Java and Java Script help identifying languages.
- Most characteristic things are input and output

C: `scanf, printf`

Java: `System.out.print, System.out.println`

Java Script: `document.write, window.prompt`

Cobol: `Input-Output Section//File-Control//Select Input-File`

Names

ADA	Named after Ada Lovelace, first programmer
ALGOL 58	Algorithmic Language from 1958, also: ALGOL 60, ALGOL 68
APL	A Programming Language
ASSEMBLER	Name for machine languages directly programming the CPU
BASIC	Beginner's All-purpose Symbolic Instruction Code
C	Successor of B which was never released
C++	Object-Oriented Variant of C
COBOL	Common Business Oriented Language
FORTRAN	Formula Translator, also: FORTRAN 77, FORTRAN 90, FORTRAN 95
HTML	Hypertext Markup Language
JAVA	Named after Java Coffee from Indonesian island Java
MODULA	Uses modules (procedures) for writing subroutines
PASCAL	Named after Blaise Pascal, French mathematician
SQL	Structured Query Language (used for databases)
XHTML	Extensible Hypertext Markup Language
XML	Extensible Markup Language

Question 9

Function 1, Frequent errors: x modified, not converted to string
Character not transformed back into a number

```
function digitsum(x)
  { var y=0; var k;
    var z="0"+x;
    for (k=0;k<z.length;k++)
      { y = y+z.charCodeAt(k)-z.charCodeAt(0); }
    return(y); }
```

Function 2, Frequent errors: x modified, Math.floor forgotten

```
function digitsum(x)
  { var y=0; var z=x;
    while (z>0)
      { y = y+(z%10);
        z = Math.floor(z/10); }
    return(y); }
```

Today's Lecture

- Growth of functions
- NP-complete problems
- Is P equal NP ?
- Even worse problems

Complexity of Towers of Hanoi

- For n rings and 3 pegs, the number of moves is $2^n - 1$.
- On a 1 GHz computer with one move per clock cycle, it would take 36 years to move 60 rings!
- 70 rings would take about 40000 years!
- 80 rings would take about 50 million years!

Monkey Puzzles

- Some tiles in an array, for example a 3 times 3 array on <http://www.comp.nus.edu.sg/~gem1501/monkeypuzzle.html>
Successfully tested for Internet Explorer only
- Interactive: Exchange tiles until neighbouring tiles have the same digits in those corners where they touch. Solution is sequence of moves.
- Puzzle: given a list of tiles, arrange them in the array such that neighbouring tiles have the same digits in those corners where they touch.

Example Monkey Puzzles

m times n tiles, here an example of 3 times 5

12	14	21	22	35	38	41	43	48	54	54	75	88	89	92
61	22	12	11	89	75	76	38	21	47	92	24	12	54	41

Can they be arranged such that the numbers in corners are the same?

43	35	54	48	88
38	89	92	21	12

38	89	92	21	14
75	54	41	12	22

75	54	41	12	22
24	47	76	61	11

Naive Algorithm: Test all possibilities

This takes a long time: 130 76743 68000 possibilities.

Example: Monkey Puzzle

- Assume an $m \times k$ board with tiles.
- Arrange $n = m \times k$ tiles so that the edges match.
- Simple solution: Try all possible ways!
- This approach is “impractical”! Better algorithms needed.
- There are n factorial many possibilities.
Note that $2^n < n! < n^n$ for $n \geq 4$.
- Next pages: $n!$ for $n = 32$ (4×8 board) and $n = 256$ (16×16 board).

Large Numbers, Powers of 32

Square of 32.

1024

Cube of 32.

32768

Fourth power of 32.

10 48576

32-th power of 2.

42949 67296

Factorial of 32.

2 63130 83693 36935 30167 21801 21600 00000

32-th power of 32.

1461 50163 73309 02918 20368 48327 16283 01965 59325 42976

<http://www.comp.nus.edu.sg/~gem1501/factorial.html>

Powers of 256

Square of 256.

65536

Cube of 256.

167 77216

Fourth power of 256.

42949 67296

256-th power of 2.

08687 90785 32699 84665 115 79208 92373 16195 42357 09850
64056 40394 57584 00791 31296 39936

Factorial of 256.

85 78177
75342 84265 41190 82271 68123 26251 57781 52027 94856 19859
65565 03772 69452 55314 75893 77440 29136 04514 08450 37588
53423 36584 30615 71968 34693 69647 53222 89288 49742 60256
79637 33256 33687 86442 67520 76267 94560 18796 88679 71521
14330 77020 77526 64645 14647 09187 32610 08328 76325 70281
89807 73671 78145 41702 50523 01860 84953 19068 13825 74810
70252 81755 94594 76987 03466 57127 38139 28620 52347 56808
21886 07012 03611 08315 20935 01947 43710 91017 26968 26286
16062 63662 43502 28409 44191 40842 46159 36000 00000 00000
00000 00000 00000 00000 00000 00000 00000 00000 00000 00000

Intractable Problems

- Polynomial problems can be solved in time $O(n^k)$ for some constant k , for example $O(n^2)$, $O(n^3)$, $O(n^4)$.
- Exponential problems: $O(2^n)$, $O(3^n)$, $O(n!)$, $O(n^n)$, $O(2^{2^n})$.
- 10^{10} steps can easily be done, 10^{20} cannot be done.

Category	Order	Tractable	Clearly Intractable
Polynomial	n^2	100000	10000000000
	n^5	100	10000
Exponential	$2^{\text{Sqrt}(n)}$	160	500
	$(4/3)^n$	70	160
	2^n	40	80
	3^n	22	44
	$n!$	15	22
	n^n	10	16
Double Exponential	$(n*n)!$	3	5
	$2^{(2^n)}$	5	7

Can we wait for faster computers?

- What if we use a computer 100 times faster?
What if we use 1000 computers hooked together?
- 1000^4 and 2^{50} operations are doable, $50!$ and 2^{100} not.
- If size m is solvable and speed increased 100 times:
This solves size $10 * m$ problems for $O(n^2)$ algorithm;
This solves size $m + 6$ problems for $O(2^n)$ algorithm.
- Polynomial problems and exponential problems really behave differently!

Computing and Verifying Solutions

Mathematical Task

Find numbers $a, b, c, d, e, f, g, h, i, j, k, l$ in $\{-3, -1, +1, +3\}$ such that

$$(a+2*b+3*c+4*d+5*e+6*f+7*g+8*h+9*i+10*j+4+11*k+12*l)^2 + (a+b+c+e)^2 + (i+j+k+l)^2 + (a*b*c*d*e*f+g*h*i*j*k*l)^2$$

is 0 for these numbers, that is, compute a solution to the equation making the above polynomial 0.

Can be done but needs some time.

Verify Solution

Proposed solution:

a is +1, b is +1, c is -1, d is +1, e is -1, f is +1,
g is +1, h is -1, i is +1, j is +1, k is -1, l is -1.

The following expressions are all 0:

$a+2*b+3*c+4*d+5*e+6*f+7*g+8*h+9*i+10*j+4+11*k+12*l$

$a+b+c+e$

$i+j+k+l$

$a*b*c*d*e*f+g*h*i*j*k*l$

Thus the sum of their squares is also 0.

For most humans, verifying solutions is easier than solving problems.

Is this true as well for computers?

NP Problems and NP-Complete Problems

- Equations or Puzzles, computer should say: “there is a solution” or “there is no solution” .
- Mathematical formulation: A is an NP-problem if for some polynomial time computable function f and polynomial p :

$$x \in A \Leftrightarrow \exists y (\text{length}(y) \leq p(\text{length}(x)) \wedge f(x, y) = 1).$$

If $x \in A$, y is called “certificate” for x .

- To test all y for given x is $2^{p(n)}$. Poly(n) time, slightly better algorithms often known.
- A problem is NP-complete iff it is at least as hard as all other NP problems, as specified on Slide 34 below.
- Monkey Puzzles form an NP-complete problem.
- It is unknown whether NP-complete problems are polynomial or exponential.

Zeroes of Multivariate Polynomials

- Mathematics: Has a function a 0 on some set?
Limiting complexity: Considering only quadratic multivariate polynomials and only places where variables are -1 or 1 .

- Let QMP contain all functions of the form

$$P(x_1, x_2, \dots, x_n) = \sum_{i,j} a_{i,j} \cdot x_i \cdot x_j.$$

where all $a_{i,j} \in \{n, -n + 1, \dots, -2, -1, 0, 1, 2, \dots, n - 1, n\}$.

An example of a function in QMP is

$$P(x_1, x_2, x_3) = x_1 \cdot x_2 - x_3^2$$

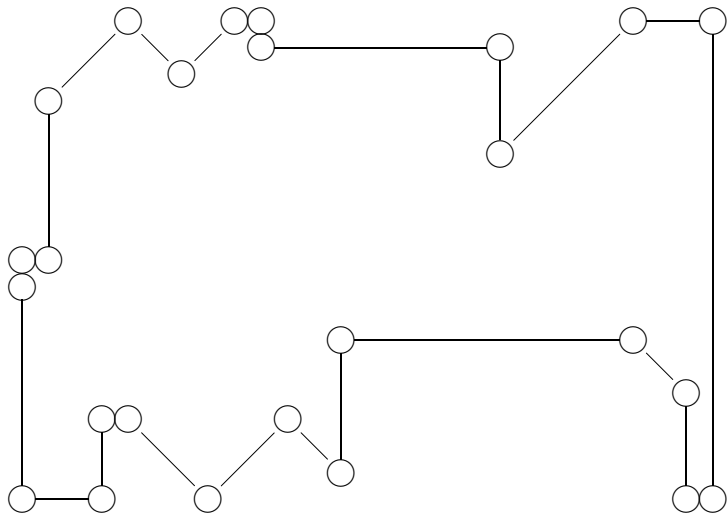
where $a_{1,2} = 1, a_{3,3} = -1$ and all other $a_{i,j} = 0$.

- Question: Is $P(x_1, x_2, \dots, x_n) = 0$ for some $x_1, x_2, \dots, x_n \in \{-1, 1\}$?
- The set $\{P \in \text{QMP} : \exists x_1, x_2, \dots, x_n \in \{-1, 1\} (P(x_1, \dots, x_n) = 0)\}$ is an NP-complete problem.

Travelling Salesman Problem

A salesman wants to visit from his hometown 24 other places for doing business and then return home.

He does not visit any place twice.



NP-Formulation

Given the cities
and a constant c ,
is there a tour
of length up to c ?

Shortest tours can be found for several 10000 cities.

Colourability

- The Four Colour Theorem provides an algorithm to colour a map of connected countries with four colours such that no neighbouring countries have the same colour.
- For some maps, this can be done with three colours.
- Given a map as a graph (countries involved, pairs of countries which are neighbouring to each other), find out whether it is possible to assign to the countries colours "white", "red" and "blue" such that for no pair of countries, both have the same colour.
- The type of graphs which come from maps are called "planar".
- For non-planar graphs, the problem which graphs can be coloured with m colours ($m \geq 3$) is NP-complete for each fixed m .

Satisfiability

- Let $\phi = x_1 \wedge (x_2 \vee x_3) \wedge (\neg x_2 \vee x_4 \vee (x_5 \wedge \neg x_6)) \wedge (\neg x_2 \vee x_5)$ be a Boolean formula over Boolean variables.
- An assignment defines to every variable a truth-value; a common short hand is 0 for false and 1 for true
- The assignment $(1, 0, 1, 0, 0, 0)$ satisfies ϕ ; it evaluates to 1:
 $1 \wedge (0 \vee 1) \wedge (\neg 0 \vee 0 \vee (0 \wedge \neg 0)) \wedge (\neg 0 \vee 0)$ is $1 \wedge 1 \wedge (1 \vee 0 \vee (0 \wedge 1)) \wedge (1 \vee 0)$
which is the conjunction of four times a 1.
- The assignment $(0, 1, 0, 1, 0, 1)$ does not satisfy ϕ ; it evaluates to 0:
 $0 \wedge (1 \vee 0) \wedge (\neg 1 \vee 1 \vee (0 \wedge \neg 1)) \wedge (\neg 1 \vee 0)$.
- A formula is satisfiable iff there is a satisfying assignment of its variables. So the above formula is satisfiable and $x_1 \wedge x_2 \wedge (\neg x_1 \vee \neg x_2)$ is not.

3SAT

- 3SAT (special variant)
- n variables and a set of up to $8n^3$ disjunctions (called clauses) of up to 3 variables and negated variables, for example, $x_1 \vee x_2 \vee x_3$, $x_4 \vee \neg x_5$, $\neg x_2 \vee \neg x_5$, $x_3 \vee x_4$.
- Is it possible to assign truth-values to the variables such that all clauses are true?
- Satisfiability has an $O(2^n)$ algorithm and 3SAT has an $O((4/3)^n)$ algorithm. Better algorithms might exist.

A Positive Example

- The system of equations:

$$Cl_1: x_1 \vee x_2 \vee \neg x_5;$$

$$Cl_2: x_1 \vee x_2 \vee x_3;$$

$$Cl_3: \neg x_1 \vee x_4;$$

$$Cl_4: \neg x_1 \vee \neg x_4;$$

$$Cl_5: \neg x_2;$$

Full formula: $Cl_1 \wedge Cl_2 \wedge Cl_3 \wedge Cl_4 \wedge Cl_5$.

- This formula is satisfiable:

Take x_5 to be false to satisfy clause Cl_1 ;

Take x_3 to be true to satisfy clause Cl_2 ;

Take x_1 to be false to satisfy clauses Cl_3 and Cl_4 ;

Take x_2 to be false to satisfy clause Cl_5 ;

The variable x_4 can be either true or false (does not matter).

- Solutions are not always so easy to find.

A Negative Example

- The system of equations:

$$Cl_1: x_1 \vee x_2 \vee \neg x_3;$$

$$Cl_2: x_1 \vee x_2 \vee x_3;$$

$$Cl_3: \neg x_1 \vee x_4;$$

$$Cl_4: \neg x_1 \vee \neg x_4;$$

$$Cl_5: \neg x_2;$$

Full formula: $Cl_1 \wedge Cl_2 \wedge Cl_3 \wedge Cl_4 \wedge Cl_5$.

- This formula is not satisfiable:

x_2 must be false since otherwise clause Cl_5 is;

x_1 must be false since otherwise one of the clauses Cl_3 and Cl_4 is by case distinction over x_4 ;

x_3 must be true since otherwise clause Cl_2 is false;

All literals are false in Cl_1 and Cl_1 cannot be made true.

- Such derivations of unsatisfiability can have exponential length, as one might have conditions involving several variables instead of only one.

Resolution: Algorithm to check Satisfiability

For each variable y do the case which applies

- If neither y nor $\neg y$ occur in any clause then this step is void
- If y but not $\neg y$ occurs in clauses then remove all clauses in which y appears
- If $\neg y$ but not y occurs in clauses then remove all clauses in which $\neg y$ appears
- If y and $\neg y$ both occur as clauses consisting of 1 literal only then return with output “Instance is not satisfiable.”
- Otherwise, for each clause of the form $y \vee \psi$ and each clause of the form $\neg y \vee \phi$, if $\psi \vee \phi$ does not contain $z \vee \neg z$ for any z then add $\psi \vee \phi$ to the set of all clauses
Having done this, remove all clauses containing y or $\neg y$ from the set of all clauses

If the algorithm has not terminated within the loop then return with the output “Instance is satisfiable.”

Resolution Example 1

- Given $x_1 \vee x_2 \vee \neg x_5$; $x_1 \vee x_2 \vee x_3$; $\neg x_1 \vee x_4$; $\neg x_1 \vee \neg x_4$; $\neg x_2$
- Resolving x_1 gives:
 $x_2 \vee x_4 \vee \neg x_5$; $x_2 \vee \neg x_4 \vee \neg x_5$; $x_2 \vee x_3 \vee x_4$; $x_2 \vee x_3 \vee \neg x_4$; $\neg x_2$
- Resolving x_2 gives:
 $x_4 \vee \neg x_5$; $\neg x_4 \vee \neg x_5$; $x_3 \vee x_4$; $x_3 \vee \neg x_4$
- Assume x_3 to be true gives:
 $x_4 \vee \neg x_5$; $\neg x_4 \vee \neg x_5$
- Resolving x_4 gives:
 $\neg x_5$
- Assume x_5 to be false gives a void instance.
So the original one is satisfiable.

Resolution Example 2

- Given $x_1 \vee x_2 \vee \neg x_3$; $x_1 \vee x_2 \vee x_3$; $\neg x_1 \vee x_4$; $\neg x_1 \vee \neg x_4$; $\neg x_2$
- Resolving x_1 gives:
 $x_2 \vee \neg x_3 \vee x_4$; $x_2 \vee \neg x_3 \vee \neg x_4$; $x_2 \vee x_3 \vee x_4$; $x_2 \vee x_3 \vee \neg x_4$; $\neg x_2$
- Resolving x_2 gives:
 $\neg x_3 \vee x_4$; $\neg x_3 \vee \neg x_4$; $x_3 \vee x_4$; $x_3 \vee \neg x_4$
- Resolving x_3 gives:
 $x_4 \vee x_4$; $\neg x_4 \vee \neg x_4$; $x_4 \vee \neg x_4$ (ignored)
- Not Satisfiable as x_4 and $\neg x_4$ both exist

Evaluation

- In the case of 2SAT, Resolution runs in polynomial time as whenever $\psi \vee \phi$ is introduced while resolving a variable, both parts contain at most one literal and so all clauses remain to have at most 2 variables. 2SAT is polynomial time computable.
- In the case of 3SAT, the number of literals per clause can grow and the number of clauses can become exponential. Resolution needs exponential time on some 3SAT instances.

What does “NP” stand for?

- P stands for the “polynomial time computable problems.”
- NP stands for “non-deterministic polynomial computable problems.”
- A “non-deterministic computer” is a computer that guesses a solution before it says “This equation has a solution” but it has no proof when saying “This equation does not have a solution.”
- The basic phenomenon of NP problems is known from mathematics: Given a function, it might be difficult to find the places where it is 0. But given such a place, one can easily verify that the function is there really 0.
- “Intuition” and “experience” might tell an expert how to solve an NP-type problem. Before handing out the solution to others, an easy verification is done to make sure that the solution is correct.

Why Only Decision Problems

- Let A_0 be a solvable 3SAT instance. For $m = 1, 2, \dots, n$:
 - Test whether $A_{m-1} \cup \{x_m\}$ or $A_{m-1} \cup \{\neg x_m\}$ is solvable.
 - Let $A_m = A_{m-1} \cup \{x_m\}$ if that is solvable and $A_m = A_{m-1} \cup \{\neg x_m\}$ otherwise.
- Now A_n has a unique solution where x_m is set to be true iff the clause x_m is contained in A_n and x_m is set to be false iff $\neg x_m$ is contained in A_n .
- The unique solution of A_n is fast to find and is a solution of A_0 .
- If 3SAT is in P one can compute a solution in polynomial time.
- In theoretical investigations, one just wants to know whether certain 3SAT instances have solutions. Every related information can be computed from the 3SAT decision problem fast.
- This is typical for all NP-complete decision problems.

NP-complete Problems Stand and Fall Together

- All NP-complete problems are equivalent!
- One NP-complete problem is in P \Leftrightarrow all NP-problems are in P.
- One problem solvable in $O(n^k)$ \Rightarrow for every problem there is a constant c such that has an $O(n^c)$ algorithm, c can be very large.
- One problem requires $O(2^n)$ \Rightarrow for every problem there is a c such that it requires at least $O(2^{n^c})$ steps, c can be a very small constant.
- It can be that some problem requires 5^{n^3} and another one can be solved in time $1.1^{\sqrt{n}}$ – which is still a large difference: first is intractable at $n = 4$ and second tractable at $n = 40000$.
- Reason for equivalence: NP-complete problems are transformable into each other.

Reducing 4SAT to 3SAT

- 4SAT Instance A : Does the system of conditions $x_1 \vee x_2 \vee \neg x_3 \vee \neg x_4$, $x_1 \vee \neg x_2$, $\neg x_1 \vee \neg x_2 \vee x_3 \vee x_5$, $x_3 \vee x_5$ have a solution?
- Idea: Split conditions with four variables into equivalent one with three variables by introducing new variables.
- Equivalent 3SAT Instance B : $x_1 \vee x_2 \vee y_1$, $\neg y_1 \vee \neg x_3 \vee \neg x_4$, $x_1 \vee \neg x_2$, $\neg x_1 \vee \neg x_2 \vee y_2$, $\neg y_2 \vee x_3 \vee x_5$, $x_3 \vee x_5$.
- A is solvable iff B is solvable.
- If A has n variables then B has at most $n + 16 \cdot n^4$ variables.
- If 3SAT has an $O(n^3)$ algorithm then 4SAT has an $O(n^{12})$ algorithm: Every instance of size n can be translated into one of size $O(n^4)$ which then can be solved in time $O((n^4)^3)$ by the 3SAT algorithm.
- If 4SAT needs 2^n steps then 3SAT needs at least $O(2^{n^{0.25}/17})$ steps.

Is P Equal to NP?

- Question lies at the heart of “Algorithmics”
- Unsolved since it was posed in 1971
- Many people believe that $P \neq NP$, but no one knows for sure.
- Mathematics: Is finding of a solution harder than verifying it?

Intermediate Problems

- If $P \neq NP$ then there are intermediate problems which are neither in P nor NP -complete. They are artificially constructed, but are there also natural ones? A natural candidate was the set of prime numbers.
- In 2002, Prof. Manindra Agrawal (IIT Kanpur) and two of his students, Nitin Saxena and Neeraj Kayal, gave a polynomial time algorithm. The test whether n is prime needs $O((\log n)^6 f(\log \log n))$ where f is a polynomial.
- Manindra Agrawal has been visiting the NUS in 2005.

Another Candidate

- A graph is a set of vertices together with a set of edges, each edge connecting two vertices.
- Consider the following two undirected graphs:
 $V_1 = \{1, 2, 3, 4\}$, $E_1 = \{(1, 2), (2, 3), (3, 4), (1, 3)\}$;
 $V_2 = \{1, 2, 3, 4\}$, $E_2 = \{(1, 2), (2, 3), (3, 4), (2, 4)\}$.
- These graphs are isomorphic:
The nodes i, j in V_1 are connected iff $5 - i, 5 - j$ are connected in V_2 . The mapping $i \rightarrow 5 - i$ verifies that one can obtain V_2 from V_1 by renaming the nodes. Such a mapping is called an isomorphism and graphs are isomorphic whenever an isomorphism exists.
- The problem whether two graphs are isomorphic is in NP: one can guess the isomorphism as a table of which vertex in V_1 corresponds to which vertex in V_2 and then verify that this table is indeed an isomorphism. Some people believe that it is neither in P nor NP-complete.

More difficult problems

- Let ϕ be a formula with variables x_1, y_2, \dots, x_n and y_1, y_2, \dots, y_n .
- Consider the following problems:
 1. What is $\exists x_1 \exists x_2 \dots \exists x_n \forall y_1 \forall y_2 \dots \forall y_n \phi$?
 2. What is the k -th bit of the number of assignments satisfying ϕ ?
 3. What is $\exists x_1 \forall y_1 \exists x_2 \forall y_2 \dots \exists x_n \forall y_n \phi$?
- All these problems can be computed by testing out all possible values for the variables in suitable order. Such an algorithm needs exponential time as it analyzes 4^n possible assignments, but its variables need only polynomial space. The complexity class PSPACE captures the decision problems which can be solved by such algorithms.
- The third problem is complete for PSPACE. The first and second one are complete for subclasses of PSPACE which are called NP^{NP} and PCOUNT (or just #P).
- $\text{P} \subseteq \text{NP} \subseteq \text{NP}^{\text{NP}} \subseteq \text{PCOUNT} \subseteq \text{PSPACE}$. More is not known.

A Provably Intractable Problem

- Game checkers on an $n \times n$ board and players Anke and Boris.
<http://www.darkfish.com/checkers/Checkers.html>
- A winning strategy F tells Anke how to win whenever possible and how to reach a draw whenever a draw is possible but not a win.
- More precisely, F assigns to every n and every situation B in an game on the $n \times n$ board a move $F(n, B)$ such that whenever Anke can win from situation B against all possible counter moves of Boris then Anke can do it by following the moves suggested by F from now onwards and similar in the case of a draw.
- One can prove that every winning-strategy F needs exponential computation time: there is a rational constant $q > 0$ such that the computation of $F(n)$ needs at least 2^{n^q} steps for all n .
- There will *never* be a program that can efficiently compute a winning strategy.

Games and Decision Problems

- The class EXPTIME captures all decision problems which can be solved in exponential time.
- The decision problem “Given situation X of Checkers and move Y , is this move optimal” (that is, not a mistake) is EXPTIME complete.
- EXPTIME-completeness needs that the game can run in exponential time.
- If a game ends after $m = p(n)$ moves for a polynomial p and board size n , then one can use the formula

$$\exists x_1 \forall y_1 \exists x_2 \forall y_2 \dots \exists x_m \forall y_m (\text{Player wins from } a \text{ with these moves})$$

to check whether the player can win from a given situation a . This problem is in PSPACE.

Worse Than Exponential

- There are problems that need a runtime of 2^{2^n} .
- Example: Presburger arithmetic
Set of all true quantified formulas over integer numbers using addition, subtraction and comparisons like $\exists x \forall y (x + x + x + y + y \neq 0)$ and $\forall x \exists y (y > x + 5)$.
- These problems are called “double exponential”.
- There are also “triple exponential” and “four times exponential” problems and so on.
- Beyond that: “nonelementary problems”.

Overview of Today's Lecture

- Growth of functions
- NP-complete problems
- Is P equal NP ?
- Even worse problems

Next Week - The Undecidable

- Certain things are not only slow but just impossible
- Unbounded Puzzles, Post's Correspondence Problem
- The Halting-Problem
- Recursively Enumerable Sets
- Rice's Theorem