T-79.240 Special Course on Computational Complexity (3 cr)

Lectures: Tuesdays 12-15, room TB353

Teacher: Prof. Ilkka Niemelä
tel. 451 3290, e-mail: Ilkka.Niemela@hut.fi.

Tutorials: Tuesdays 16-18, room TB353

Assistant: Matti Järvisalo
e-mail: mjj@tcs.hut.fi

Home page: `http://www.tcs.hut.fi/Studies/T-79.240/`

© 2004 Teknillinen korkeakoulu, Tietojenkäsittelyteorian laboratorio

Goals

- Identification of computationally hard problems
- Classification of problems according to their complexity
- Choosing appropriate algorithmic approach w.r.t. complexity of the problem

© 2004 Teknillinen korkeakoulu, Tietojenkäsittelyteorian laboratorio

Topics

- Central complexity classes (P, NP, PSPACE, NC, polynomial hierarchy, ...) and related methods for complexity analysis
- Randomized computation
- Parallel computation
- Cryptography

Material: C. Papadimitriou, Computational Complexity, Addison-Wesley, 1994. Ch. 1–4, 7–20

© 2004 Teknillinen korkeakoulu, Tietojenkäsittelyteorian laboratorio

Passing the course

- Seminar talk
- Home assignments (2 assignments/week)

There is **no final exam**.
The grade of the course is determined by the grade for the seminar talk (20%) and the grade for the home assignments (80%) which is obtained using the following table.

Grade	Lower bound (from max. points)
1	50 %
2	57 %
3	64 %
4	71 %
5	78 %

© 2004 Teknillinen korkeakoulu, Tietojenkäsittelyteorian laboratorio

Practical arrangements

- There will be **2 home assignments/week**.

  The **deadline** for returning the assignments is **two weeks** from the lecture where they were given.

  Each assignment is graded with the scale 0–2.

  There is a fall-back deadline for all home assignments:
  Jan 14, 2005.
  For assignments returned by the fall-back deadline the scale is 0–1.5.

- Seminar talks are 45 min each.
  No written report is required but the talks will be graded (by the lecturer and other students).

© 2004 Teknillinen korkeakoulu, Tietojenkäsittelyteorian laboratorio

Problems vs. algorithms

The course focuses on analyzing the computational complexity of *problems* (not algorithms).

- A problem: an infinite set of possible instances with a question
- A *decision problem*: a question with a yes/no answer

Example. REACHABILITY:
INSTANCE: A graph (V,E) and nodes $v,u \in V$.
QUESTION: Is there a path in the graph from v to u?

© 2004 Teknillinen korkeakoulu, Tietojenkäsittelyteorian laboratorio

An algorithm for REACHABILITY:

$S := \{v\}$; mark v;
while $S \neq \{\}$ **do**
    choose a node i and remove it from S;
    for all $(i,j) \in E$ **do**
        if j is not marked **then** mark j and add it to S
        endif
    endfor
endwhile ;
if u marked **then** return 'there is a path from v to u'
else return 'there is no path from v to u'
endif

© 2004 Teknillinen korkeakoulu, Tietojenkäsittelyteorian laboratorio

How efficient is the algorithm?

- Programming language?
- Computer architecture?
- Representation of the graph?
- Representation of the set S?

Given certain assumptions the algorithm terminates in $O(|E|)$ steps.

© 2004 Teknillinen korkeakoulu, Tietojenkäsittelyteorian laboratorio

Rate of growth of functions

Let f, g: $\mathbf{N} \mapsto \mathbf{N}$.

- $f(n) = O(g(n))$ (*f grows as g or slower*), if there are positive integers c and n_0 such that for all $n \geq n_0$, $f(n) \leq c \cdot g(n)$
- $f(n) = \Omega(g(n))$, if $g(n) = O(f(n))$
- $f(n) = \Theta(g(n))$, if $g(n) = O(f(n))$ and $f(n) = O(g(n))$.

Example. If $p(n)$ is a polynomial of degree d, then $p(n) = \Theta(n^d)$.

If $c > 1$ is an integer and $p(n)$ a polynomial, then $p(n) = O(c^n)$ but $p(n) \neq \Omega(c^n)$, i.e., *any polynomial grows strictly slower than any exponential*.

If $k > 1$ is an integer, then $\log^k n = O(n)$

© 2004 Teknillinen korkeakoulu, Tietojenkäsittelyteorian laboratorio

Simplifying assumptions for analyzing the computational complexity of problems:

- A problem is *efficiently solvable* when there is an algorithm solving the problem such that the rate of growth of the solution time is *polynomial* w.r.t. the size n of the input ($O(n^d)$)
- A problem is *intractable* when no polynomial time algorithm available.
- Consider the *worst-case performance* (not, e.g., average case).
- Model of algorithms: *Turing machines*

© 2004 Teknillinen korkeakoulu, Tietojenkäsittelyteorian laboratorio

Possible criticism:

- Not all polynomial time algorithms are efficient in practice.
  There are efficient computations that are not polynomial.
  For instance, consider n^{80} vs $2^{\frac{n}{100}}$.
- Average case analysis is more informative than worst-case.

Adopting polynomial time worst-case performance as our criterion of efficiency results in an elegant and useful theory that says something meaningful about practical computation, and would be impossible without this simplification.

© 2004 Teknillinen korkeakoulu, Tietojenkäsittelyteorian laboratorio

Example problems

MAX FLOW
INSTANCE: Network $N = (V, E, s, t, c)$, where (V, E) is a (directed) graph, $s, t \in V$, s has no incoming edges, t has no outgoing edges and c is a function giving a capacity for each edge ($c(i, j)$ is a positive integer).
QUESTION: What is the largest possible value for the flow in N?

A *flow* is a function f that assigns for each edge (i, j) a nonnegative integer $f(i, j) \leq c(i, j)$ such that for each node (except s, t) the sum of fs of the incoming edges is equal to the sum of fs of the outgoing edges.
The *value* of the flow is the sum of the flows in the edges leaving s.

© 2004 Teknillinen korkeakoulu, Tietojenkäsittelyteorian laboratorio

- MAX FLOW is an *optimization problem*.
- MAX FLOW(D) (decision problem)
  INSTANCE: Network $N = (V, E, s, t, c)$ and integer K (goal/target value)
  QUESTION: Is there a flow of value K or more?
- MAX FLOW and MAX FLOW(D) are roughly equivalent.
- MAX FLOW is a nice example of a problem where the challenge was to find a polynomial time solution method.

  When "the *barrier of exponentiality*" was broken, more and more efficient polynomial time algorithms were developed $(O(n^5), \ldots, O(n^3), \ldots)$

© 2004 Teknillinen korkeakoulu, Tietojenkäsittelyteorian laboratorio

Bipartite matching

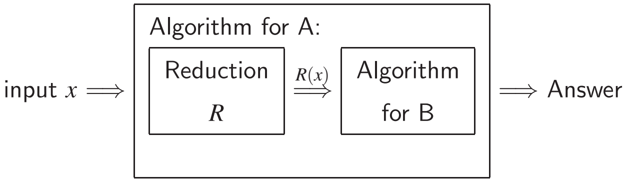
MATCHING

INSTANCE: Bipartite graph $B = (U, V, E)$, where $U = \{u_1, \ldots, u_n\}$, $V = \{v_1, \ldots, v_n\}$, $E \subseteq U \times V$.

QUESTION: Is there a set $M \subseteq E$ of n edges such that for any two edges $(u, v), (u', v') \in M$, $u \neq u'$ and $v \neq v'$
(is there a *perfect matching*)?

© 2004 Teknillinen korkeakoulu, Tietojenkäsittelyteorian laboratorio

- MATCHING can be solved by a *reduction* to MAX FLOW.
- A reduction is an algorithm that solves problem A by transforming any instance of A to an equivalent instance of a problem B (for which an algorithm already exists).



- An efficient algorithm for MAX FLOW provides an efficient algorithm for MATCHING if the reduction is efficient.

© 2004 Teknillinen korkeakoulu, Tietojenkäsittelyteorian laboratorio

- A reduction from MATCHING to MAX FLOW:
  Given any bipartite graph $B = (U, V, E)$, construct a network
  $N = (V \cup U \cup \{s, t\}, E', s, t, c)$,
  where $E' = E \cup \{(s, u) \mid u \in U\} \cup \{(v, t) \mid v \in V\}$ and all capacities equal to 1.
- B has a perfect matching iff N has a flow of value n.

© 2004 Teknillinen korkeakoulu, Tietojenkäsittelyteorian laboratorio

The traveling salesperson problem

TSP

INSTANCE: n cities $1,\ldots,n$ and a nonnegative integer distance d_{ij} between any two cities i and j (such that $d_{ij} = d_{ji}$).

QUESTION: What is the shortest tour of the cities, i.e., permutation π such that

$$\sum_{i=1}^{n} d_{\pi(i)\pi(i+1)}$$

is as small as possible (where $\pi(n+1) = \pi(1)$).

Decision problem TSP(D): is there a tour of length at most B (budget)?

© 2004 Teknillinen korkeakoulu, Tietojenkäsittelyteorian laboratorio

- A naive algorithm: enumerate all possible solutions, compute the cost of each, and pick the best.
  Not very practical; $O(n!)$ tours (10! = 3 628 800)
- For TSP no polynomial algorithm is known (despite very intensive efforts).
- Conjecture: there can be no polynomial-time algorithm for TSP.
  Closely related to one of the most important open problems in computer science:

$$\mathrm{P} = \mathrm{NP}?$$

© 2004 Teknillinen korkeakoulu, Tietojenkäsittelyteorian laboratorio