EXAMPLES OF PROBLEMS

- Representation of problems
- Solving problems with algorithms
- Rates of growth
- Further examples
- Reductions

(C. Papadimitriou, *Computational complexity*, Chapter 1)

Problems vs. Algorithms

This 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?

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

Questions

How efficient is the algorithm?

How is it affected by

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

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

Rates of Growth

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

- $f(n) = \mathrm{O}(g(n))$ (*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 $g(n) = \mathrm{O}(f(n))$
- $f(n) = \Theta(g(n))$, if $g(n) = \mathrm{O}(f(n))$ and $f(n) = \mathrm{O}(g(n))$.

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 $p(n) = \mathrm{O}(c^n)$ but $p(n) \neq \Omega(c^n)$, i.e.,

any polynomial grows strictly slower than any exponential.

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

Simplifying Assumptions

The following simplifying assumptions are introduced when the computational complexity of problems is analyzed:

- 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 ($\mathrm{O}(n^d)$)
- A problem is *intractable* when no polynomial time algorithm available for it.
- Consider the *worst-case performance* (not, e.g., average case).
- Mathematical model of algorithms: *Turing machines*

Discussion

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.*"

Further Examples

- Maximum flow
- Bipartite matching
- The traveling salesperson problem

Maximum Flow

MAX FLOW

INSTANCE: Network $N = (V, E, s, t, c)$, where (V, E) is a (directed) graph, $s, t \in V$, the *source* s has no incoming edges, the *sink* t has no outgoing edges and c is a function giving a *capacity* for each edge (each $c(i, j)$ is a positive integer).

QUESTION: What is the largest possible value for the flow in N?

Definition. 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.

Discussion

- MAX FLOW is an *optimization problem*.
- MAX FLOW(D) (decision problem)
  INSTANCE: Network N 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 $(\mathrm{O}(n^5), \ldots, \mathrm{O}(n^3), \ldots)$

© 2006 TKK, Laboratory for Theoretical Computer Science

Bipartite Matching

MATCHING

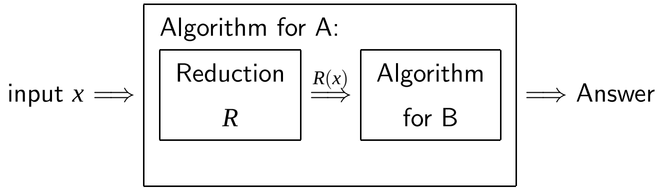
INSTANCE: Bipartite graph $B = (U, V, E)$, where $U = \{u_1, \ldots, u_n\}$, $V = \{v_1, \ldots, v_n\}$, and $E \subseteq U \times V$.

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*)?

© 2006 TKK, Laboratory for Theoretical Computer Science

Reductions

- A reduction is an algorithm that solves problem A by transforming any instance x of A to an equivalent instance of a problem B (for which an algorithm already exists).



- An efficient algorithm for B provides an efficient algorithm for A if the reduction R from A to B is efficient.

© 2006 TKK, Laboratory for Theoretical Computer Science

Example

- MATCHING can be solved by a *reduction* 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.

The Traveling Salesperson Problem

TSP

INSTANCE: n cities $1, \dots, n$ 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., a 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)?

Discussion

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

Learning Objectives

- Ability to read and formulate decision/optimization problems
- Basic understanding of growth rates (polynomial vs. exponential)
- The idea of reducing one problem in another