Efficient Sequential Algorithms

Michele Zito

University of Liverpool

http://www.csc.liv.ac.uk/{~ctag,~michele}

Overview

Let's start with a very quick account of what you may have learnt so far in previous Algorithmic courses (COMP108, COMP202, COMP209).

1. Models of computation (Random Access Machines, Pseudo-code). Non-determinism might have been mentioned.
2. Complexity measure has always been "time" and only a rough classification of tractable/possibly-intractable was considered: the relationship between P vs. NP(-complete).
3. Algorithm paradigms: divide and conquer, greedy, dynamic programming.
4. Algorithms for various problems: searching and sorting, various graph algorithms.

In this module (1)

More examples of two important design paradigms:

1. Greedy algorithms (CLR chap 17): Activity Selection. Why/when does it work?
2. Dynamic programming (CLR chap 16): Matrix-chain multiplication. Why/when does it work?

This will give a more comprehensive view on the process of algorithm design.

In this module (2)

Investigation of 2 important problem areas:

1. String algorithms (CLR chap 34 + 17.3): from the brute-force algorithm to Knuth-Morris-Pratt linear time algorithm. Text compression through Huffman codes (example of winning greedy heuristic) and Lempel-Ziv encoding scheme.
2. Matchings in graphs: Maximum matching in trees (optimal greedy solution); Bipartite maximum matching (use flow or don't use flow, the Hungarian method); Maximum matching in general graphs.

In this module (3)

Complexity theory.

In COMP202 you mainly looked at TIME ... and you got into troubles with the P$\stackrel{?}{=}$NP question.

This time we focus on SPACE (in other words *computer memory*). We will prove that

PSPACE = NPSPACE

(i.e. if we used Space instead of Time we wouldn't have to worry about non-determinism) Savitch algorithm.

We will also go "beyond" the P vs NP question, talking about approximation algorithms for optimisation problems (definitions + problems: scheduling/bin packing, knapsack, vertex cover).

Main Issues: Efficiency

What is a good (or *effi cient*) program?

Non-trivial question. We will give techniques to design "good" algorithms.

Sometimes this process is complicated and we need to use our brain to improve on a simpler, but ineffi cient solution.

Main Issues: Mathematics

Can we really do it without Mathematics?

Mathematics is not too good at analysing human processes or anything that cannot easily be coded as a sequence of symbols.

However our programs deal with fi nite structures that are processed in prescribed ways. Mathematics is slightly more useful with this situation.

We will see cases where we will want to assert "this program is good"! Often Mathematics will give us ways to do so.

We will also see cases where the Mathematical answer is weak, and a good programmer may devise a solution that is better in practice.

So, be prepared to face and digest some Mathematics.

Pseudocode conventions

We will describe algorithms using pseudo-code.

- Assignments will have the form $a \leftarrow b$ ($a = b$ refers to the comparison between a and b).
- Sometimes elementary instructions will be expressed in natural language (for instance "sort the array A in increasing order" will be an elementary instructions).
- **if-else**, **for**, and **while** control structures will be used for selection statements and loops.
- The instruction **return** e returns (i.e. "prints") the value of e and exits from the program execution.
- Indentation will indicate block structure.

- (Unless otherwise stated) variables will be local to the given procedure.
- Arrays are denoted by $A[i]$, but also $A[i..j]$. length(A) denotes the number of elements of the array A.
- Parameters are always passed *by value*.

More conventions as we meet them.

References

T. H. Cormen, C. E. Leiserson, R. L. Rivest **Introduction to Algorithms** - M.I.T. Press (2001).

J. L. Balcazar, J. Diaz, J. Gabarro **Structural Complexity I** - Springer-Verlag (1995).

D. P. Bovet, P. Crescenzi **Introduction to the Theory of Complexity** - Prentice Hall (1994).

C. Papadimitriou **Computational Complexity** Addison-Wesley (1994).

DO TAKE NOTES!! It is a real-time revision exercise. Slides are dry in some cases, too extended in others (they should be well commented).

Problems

Most computational problems can be viewed as the seeking of information about a relationship between different sets of data (*instances* and *solutions*).

Examples. Boolean functions and satisfying assignments. A web search. Your module settings and your exam results.

In this view a *decision problem* (also known as *existence problem*) answers the following question:

Is there any solution for the given instance?

Another type of decision problem (which will be referred to as the *membership problem*) answers the question:

Is some set of data Y a solution for the given instance?

Several other types of problems are definable in this setting.

1. a *construction problem* (or *search problem*) aims at exhibiting (i.e. producing as output) a solution for the given instance; if we are interested in the list of ALL solutions we are dealing with a *listing problem*.
2. we may also associate a *cost*, a numerical function, to every pair (X, Y) (problem instance, possible solution); an *optimisation problem* aims at finding one of the solutions for which the cost is optimised (e.g. maximised/minimised), for the given instance;
3. a *counting problem*, aims at finding the number of solutions associated with the given instance.

Example 1

A boolean formula is an expression like

$$(x_{25} \wedge x_{12}) \vee \neg(\neg x_{70} \vee (\neg x_3 \wedge x_{34}))$$

built up from the elements x_i called *propositional variables*, a finite set of *connectives* (usually including $\wedge$, $\vee$ and $\neg$) and the brackets.

If variables are assigned one of the (*truth*-)values true or false and connectives are interpreted in the usual way then each formula can be thought of as a simple machine producing true or false depending on the values given to the variables.

In this setting, SAT is the well known problem of deciding whether a propositional boolean formula (like the one above) admits an assignment of truth-values to all variables such that the value produced by the formula is TRUE.

Review questions

1. Define a natural counting problem associated with boolean formulae.
2. Define a maximization problem associated with boolean formulae.

Example 2

Let S be a string of text representing a query (e.g. $S \equiv$ "Efficient algorithms" or "Liverpool players") and $\mathcal{W} = \{W_1, W_2, \ldots\}$ be the collection of all web pages indexed by a particular search engine.

The *web search* problem $(S, \mathcal{W})$ is that of retrieving all web pages W_i containing the string S.

It is a listing problem.