Algorithms and Datastructures

TDA143 Programmerande System

February 10, 2012

Birgit Grohe

What is this lecture about?

- What is an algorithm - definition and examples
- Algorithms analysis - efficiency and correctness
- Searching and Sorting
- Algorithms and problem solving
- A difficult standard problem: TSP
- Construction of algorithms: algorithm design principles
- Datastructures
- A datastructure for searching: binary search tree
- Is it possible to make money with datastructures and algorithms?
- Summary

Why Algorithms?

Because with the help of algorithms, we can solve many different problems:

- Construct a fast search engine (google, yahoo)
- Sort huge lists of students alphabetically
- Construct schedules for pilots and aircrafts for an airline
- Code and decode messages
- Data compression
- Find the shortest path in a network (e.g. västtrafik)
- ...

What is an Algorithm?

Informal description:

A set of steps that defines how a task is performed.

Formal definition:

An algorithm is an ordered set of unambiguous, executable steps that defines a terminating process.

[Brookshear: Computer Science, an overview]

An Algorithm and its Representation

Algorithm: abstract idea for solving a problem

Representation: formulation of the abstract idea by using e.g.

- English,
- algebraic formulae,
- pseudocode,
- programs,
- diagrams or pictures, ...

Example: Convert a temperature given in Celsius to Fahrenheit.

- Multiply the temperature reading in Celsius by $\frac{9}{5}$ and add 32 to the product.
- $F = \frac{9}{5}C + 32$

More Examples for Algorithms and their representation

Example 1: Shelling Peas

Example 2: Folding a bird from a square piece of paper

Brookshear, chapter 4 'Algorithms'

Pseudocode: The Sequential Search Algorithm

Given a sorted list with $n > 0$ elements.

procedure SequentialSearch (List,Value){

while(entries left to be considered) {

do TestEntry $\leftarrow$ next entry from List

if (Value = TestEntry) **then return** 'Search successful'

}

return 'Search failed'

}

Primitives:

assignment: TestEntry $\leftarrow$ next entry from List

loop: **while** (condition) **do** (action)

if (condition) **then** (action1) **else** (action2)

Analysis of Algorithms

Given a problem and a (terminating) algorithm. Then the analysis of algorithms includes:

- *correctness:* does the algorithm solve the problem?
- *efficiency:* what is the *running time* of the algorithm? In the best case, the worst case and the average case?

The running time is measured in number of *unit operations* (addition, multiplication, assignment, comparison, etc.), not in seconds.

Usually, not the exact nr of operations is considered, but the *asymptotic complexity*. Notation $O()$ or $\Theta()$.

Running time of the Sequential Search Algorithm

For lists of length $n \geq 2$:

Number of operations in the best case: $1 + 1 + 1 + 1 = 4$

Number of operations in the worst case: $3(n - 1) + 1$

Number of operations in the average case: ca $3n/2$

Is there a better algorithm?

The Binary Search Algorithm

Given a sorted list with $n > 0$ elements:

procedure BinSearch (List,Value){

if (List empty)

  then return 'Search failed'

  else

    (Select the middle entry from List and call it TestEntry)

    if (Value = TestEntry) **return** 'Search successful'

    if (Value > TestEntry) **BinSearch**(RightHalfOfList, Value)

    if (Value < TestEntry) **BinSearch**(LeftHalfOfList, Value)

}

Running time of the Binary Search Algorithm

For lists of length $n \geq 2$

Number of operations in the best case: $1 + 1 + 2 = 4$

Number of operations in the worst case: $1 + 2(\log_2 n) + 2$

Algorithm Choice: Does it Make Any Difference?

Example: Given a database with 30.000 student's records sorted by their id-numbers. How long does it take to check the record of 10.000 students given her or his personal number? We assume that one comparison takes 1 ms.

Sequential Search: 45 sec/student (on average); total ca 5 dygn

Binary Search: 0.033 sec/student (worst case): totalt 5,5 min

Insertion Sort

Task: Sort a list of $N > 2$ entries (e.g. strings) i increasing order.

procedure InsertionSort (List){

$N \leftarrow 2$

while ($N <$ LengthOfList) **do**

Select the Nth entry in the List as the pivot entry

Move the pivot to a temporary location leaving a hole in the list

while ((exists entry above the hole) and (entry $>$ pivot)) **do**

Move the entry above the hole down into the hole

Move the pivot entry into the hole in the List

$N \leftarrow N + 1$

}

Running time of the Insertion Sort Algorithm

For lists of length $n \geq 2$

Best case: $1 + 3(n-1) \rightarrow O(n)$

Worst case (if the list is in reverse order):
$1 + 3(1 + 2 + 3 + 4 + \ldots + n - 1) + (n-1) = \frac{3n(n-1)}{2} + n \rightarrow O(n \cdot n)$

Is there a faster sorting algorithm?

Algorithms and Problem Solving

George Polya's 4 problem solving phases:

1. Understand the problem.
2. Devise a plan for solving the problem *(get an idea for an algorithm or find a similar, already solved problem).*
3. Carry out the plan *(formulate the algorithm and represent it as a program).*
4. Evaluate the solution for accuracy and for its potential tool for solving other problems.

To solve a (difficult) problem, it is often necessary to go through all phases several times.

Algorithms and Problem Solving

Warning!

Some problems are unsolvable ('undecidable'), and some problems are 'difficult'!

And many are already solved - so called 'Standard Problems'.

Problem Solving: An example

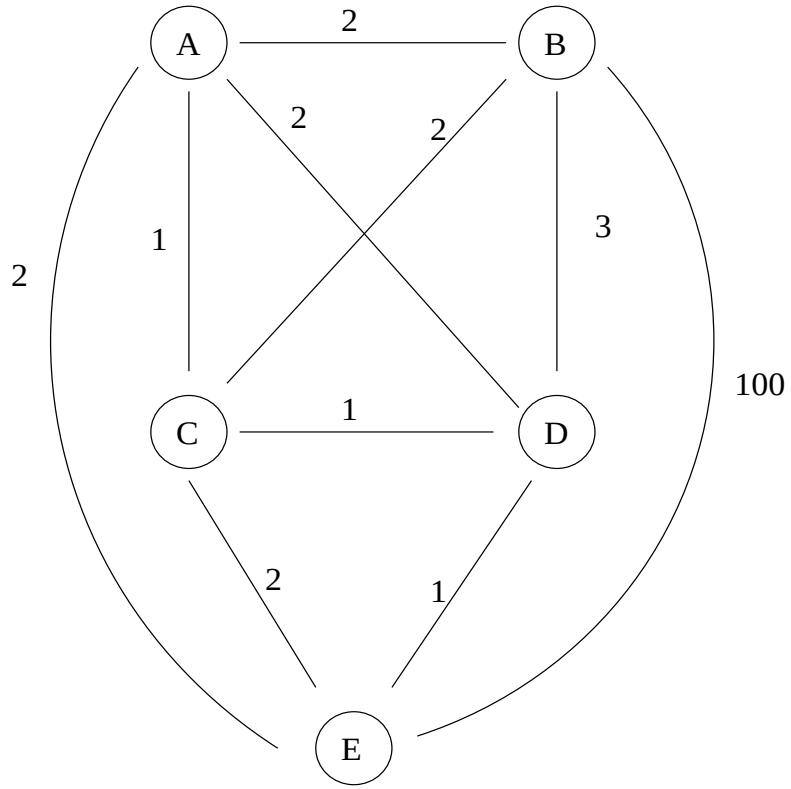
Person A should guess the age of person B's three children. B tells A that the product of the children's ages is 36. B requests another clue. B tells A the sum of the children's ages. Again, A replies that another clue is needed and finally B tells A that the oldest child plays piano.[a]

Products		Sums	
$(1, 1, 36)$	$(1, 6, 6)$	$1 + 1 + 36 = 38$	$1 + 6 + 6 = 13$
$(1, 2, 18)$	$(2, 2, 9)$	$1 + 2 + 18 = 21$	$2 + 2 + 9 = 13$
$(1, 3, 12)$	$(2, 3, 6)$	$1 + 3 + 12 = 16$	$2 + 3 + 6 = 11$
$(1, 4, 9)$	$(3, 3, 4)$	$1 + 4 + 9 = 14$	$3 + 3 + 4 = 10$

[a] Brookshear

The Travelling Salesperson Problem (TSP)

Given n cities and distances between them. Find the shortest round tour, i.e. visit each city exactly once and return to the starting city.



TSP (continued)

The TSP is a typical example for a 'difficult' problem. Difficult means that no one has found an efficient, i.e. *(polynomial)*, algorithm despite of extensive research[a].

The theory of NP-complete ('difficult') problems is an important research field in algorithms.

All algorithms for solving the TSP more or less enumerate all tours and pick the best, there are $O(n!)$ tours.

Large problems can only be solved approximatively.

[a] TSP homepage: http://www.tsp.gatech.edu

Construction of Algorithms: Design Principles

Algorithm design principles:

- Greedy
- Divide-and-Conquer
- Dynamic Programming
- Complete search: enumerate all possible solutions explicitly or implicitly
- Heuristics
- ...

[Philosophical aspect in algorithms: Are new algorithms *discovered* or *created*? What about patents for algorithms?]

Merge Sort: An Example for Divide-and-Conquer

Idea: Divide an unsorted list into halves, sort each half recursively. Then combine the two smaller sorted list into one list again ('merge').

procedure MergeSort (List){

if LengthOfList ≥ 2

  then divide list into two halves

  MergeSort (RightHalf)

  MergeSort (LeftHalf)

  Merge (RightHalf, LeftHalf)

Running time of Merge for a list of n elements: 2n+1.

Running time of Mergesort: $O(n \log n)$ (better than Insertion Sort!)

Datastructures

Goal: Provide convenient ways of accessing data storage.

Other issues in datastructures:

- abstraction
- static versus dynamic structures
- pointers

Most programming languages provide a number of basic datastructures: arrays, lists, etc.

Datastructures and Algorithms

Implementing (and analysing) an algorithm requires datastructures.

Algorithm + suitable datastructures	$\rightarrow$	fast program
Algorithm + unsuitable datastructures	$\rightarrow$	slow program

Tight relation of datastructures and algorithms:
Some algorithms are inspired by the existence of special datastructures – datastructures are invented to support solving a specific algorithmic problem.

Development of datastructures and algorithms driven by *efficiency*.

Datastructures

- arrays
- lists
- stacks
- queues
- trees
- graphs
- sets
- dictionaries
- combined datastructures
- ...

Stacks

Example: Researcher at Chalmers processing a number of tasks in LIFO (last in first out) manner.

Conceptual structure: a pile of tasks.

Possible operations: add or remove a task

Implementation:

- Reserve a large enough block of memory
- Pointer *top* points to the item on the top of the pile
- Function *push* adds an item to the stack (top-pointer is adjusted)
- Function *pop* removes the top item (top-pointer is adjusted)

Stacks (continued)

Other applications: printing a list in reverse order, bookkeeping in Backtracking procedures, etc.

Note: Even if a stack is used as a dynamic structure, the underlying structure is a static array (so the stack can be 'full').

It is possible to choose a dynamic array instead of a static array.

Queues

Example: Researcher at Chalmers processing a number of tasks in FIFO (first in first out) manner.

Conceptual structure: a queue of tasks.

Possible operations: remove a task from the front or add a task at the *tail*.

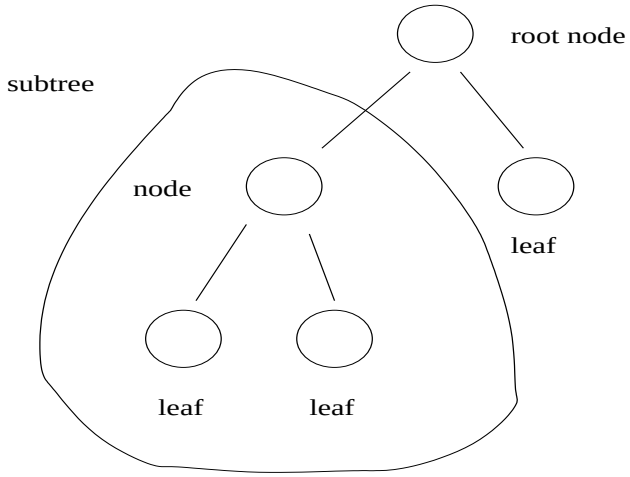
Implementation:

- as a double linked list
- as a static/dynamic array using a head- and a tail-pointer

Active queues, even small ones, consume a lot of memory $\rightarrow$ *cyclic queues*.

Trees

Some terminology



More terminology: parent, child, siblings

Trees (continued)

Example: Store names of persons.

Possible operations: search for a name, print all names in alphabetical order, insert a new name

Conceptual structure: Binary search tree (property: for every node, all nodes in the left subtree have value less or equal and all nodes in the right subtree have value greater or equal than the node's value).

Binary Search (revisited)

```
procedure BinSearch (Tree, Value){
if (root pointer = NIL)
   then return 'Search failed'
   else
        (TestEntry ← value root node)
        if (Value = TestEntry) return 'Search successful'
        if (Value > TestEntry)
           then BinSearch(RightSubtree, Value)
           else BinSearch(LeftSubtree, Value)
}
```

LEDA: Library of Efficient Data Types and Algorithms

LEDA program package developed by the researchers K. Mehlhorn and S. Näher starting in 1988

Goals: Provide efficient implementations of basic and advanced datastructures to

- save users from reinventing datastructures possibly loosing efficiency
- speed up transfer of research into practice

Now commercial product sold by 'Algorithmic Solutions GmbH'

http://www.algorithmic-solutions.com/

Jeppesen (Carmen Systems): A Company built on Algorithms

Jeppesen, Boeing (former Carmen Systems) Gothenburg

Software for Airline Scheduling problems (crew pairing, rostering and fleet assignment)

Their oldest product, the crew pairing solver, contains a 15-year old algorithm designed by Dag Wedelin, Associate Professor at Chalmers, Computing Science.

The software is used by many major European airlines and railway companies, e.g. LH, SAS, British Airways, Spanair, Air France, northwest airlines, iberia, KLM, Finnair, Deutsche Bahn, SJ ...

Summary

- Algorithms is an important part of Problem Solving.
- Datastructures are necessary tools for efficient implementation of algorithms.
- There exist a large number of standard problems with already known solutions.
- Many problems are difficult to solve (e.g. TSP) $\leftarrow$ Theory of NP-completeness. Some problems are not solvable at all.
- Use algorithm design principles for constructing algorithms!
- Algorithms is both of theoretical AND practical interest.
- Ethical issues in algorithms: does the inventor of an algorithm or the programmer have any responsibility for what the algorithm/program is used for?

Literature

J.G.Brookshear, Computer Science: An overview. (chapter 5 and 8)

Cormen, Leiserson, Rivest, Stein: Introduction to Algorithms

Brassard, Bratley: Fundamentals of Algorithmics

G. Polya: How to Solve it

http://www.jeppesen.com

Mehlhorn och Näher: LEDA