Algorithms and Datastructures TDA143

2014-02-19

Birgit Grohe

What is this Lecture about?

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

Why Algorithms?

With the help of algorithms, many problems can be solved:

- Construct a fast search engine (google, yahoo etc)
- Sort huge lists of e.g. students
- Construct schedules for pilots and cabin crew
- Code and decode messages
- Data compression
- Find shortest paths in networks (e.g. västtrafik)
- …

What is an Algorithm?

Informal description:

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

Formal description

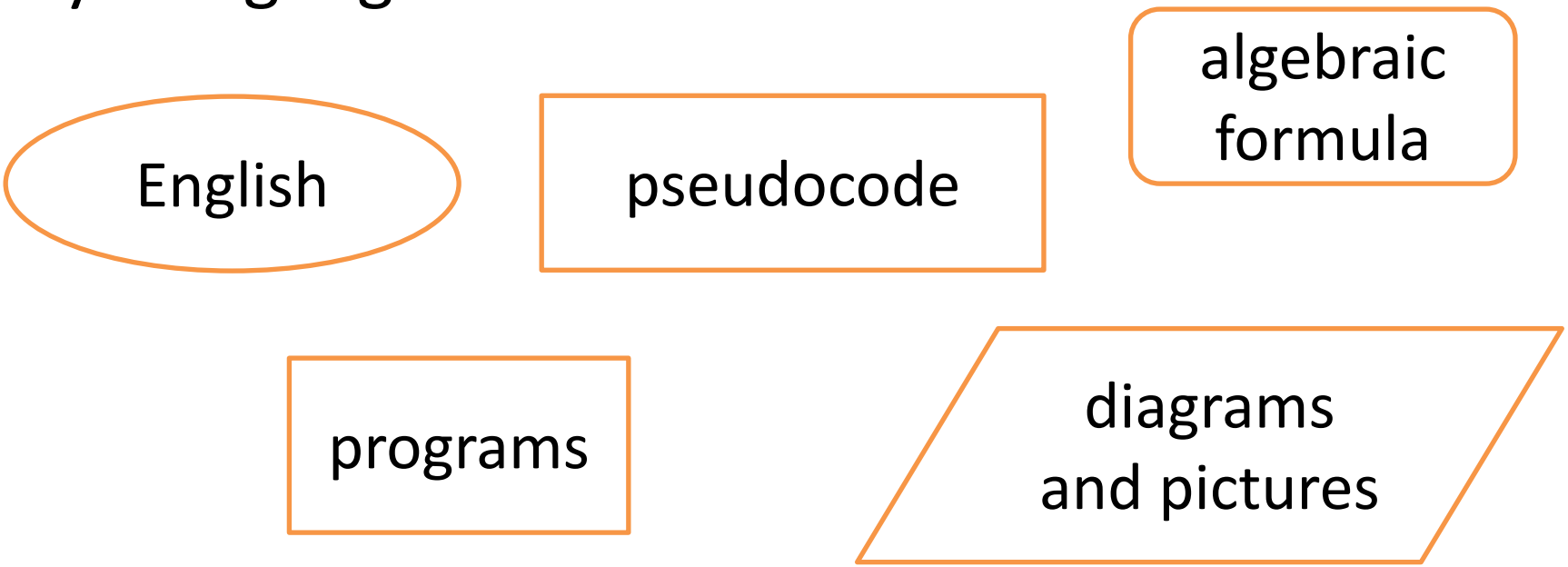
An algorithm is an ordered set of unambiguous, executable steps that defines a terminating process.

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.



An Algorithm and its Representation

Examples:

- Convert a temperature reading given in Celcius to Fahrenheit.
  - Multiply the temperature reading in Celsius by 9/5 and add 32 to the product.
  - F = 9/5*C + 32
- Shelling peas
- Folding a bird from a piece of paper

Pseudocode: Sequential Search Algorithm

For sorted lists of length n≥1 elements

```
procedure SeqSearch( List, Value ) {
while( entries left to be considered) {
    do TestEntry ← next entry from List
    if( Value = TestEntry ) then return ´sucess´
    }
return ´search failed´
}
```

Pseudocode: Primitives

- Assignment: e.g.

  TestEntry ← next entry from List
- Loop: e.g**. while**(condition) **do**(action)
- If-clauses: e.g. **if**(condition) **then**(action1) **else**(action2)

Analysis of Algortihms

Given a problem and a (terminating) algorithm, analysis includes:

- *Correctness:* does the algorithm solve the problem?
- *Efficiency*: what is the *running time* of the algorithm? In the worst case , in the average case?

Big-O notation O()

- The *running time* of an algorithm is measured in terms of *unit operations*, i.e. comparisons, additions, multiplications, assignments. *Not* in seconds!

- Usually, not the exact number of unit operations is measured, but the *asymptotic complexity* O() or Θ().

Formal definition is given later.

Running time of SequSearch

Listlength $n \geq 2$.

Number of unit operations

- in the average case: ca $3n/2$
- In the worst case: $3n+1$

Maybe there is a better/faster algorithm?

The Binary Search Algorithm

```
procedure BinSearch( List, Value ) {
if( List empty ) {
  then return ´failed´}
  else {
    TestEntry ← middle entry of List
    if( Value = TestEntry) return ´sucess´
    if( Value>TestEntry ) BinSearch( RightHalfofList, Value )
    if( value<TestEntry ) BinSearch( LeftHalfofList, Value )
}
```

Running time of BinSearch

Listlength n≥2.

Number of unit operations in the worst case:

$(1 + 4)(\log_2 n) + 2$

$= 5 \log_2 n + 2$

Compare to SequSearch: 3n+1

Asymptotic complexity O()

A function g(n) is O(f(n))

..is order of..

if there is a constant c>0 such that

$$g(n) \leq c\, f(n)$$

for all $n \geq n_0 \geq 0$.

What does the definition mean?

´Find a upper bound f(n) by ignoring the constants and the function´s behaviour for small n.´

Algorithm Choice: Does it make any Difference?

Example: Given a database with 30.000 student's records sorted by their personal numbers.

How long does it take to check the record for 10.000 students given her or his personal number?

We assume that a unit operation takes 1 ms.

Insertion Sort

Given a list of N≥2 entires (e.g. strings or numbers)

```
procedure InsertionSort( List ){
N ← second entry in the list
while ( N ≤ LengthOfList ) do{
  Select the Nth entry in the List as the pivot entry
  Move 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 ← N + 1 }
}
```

Running time of Insertion Sort

Worst case analysis:

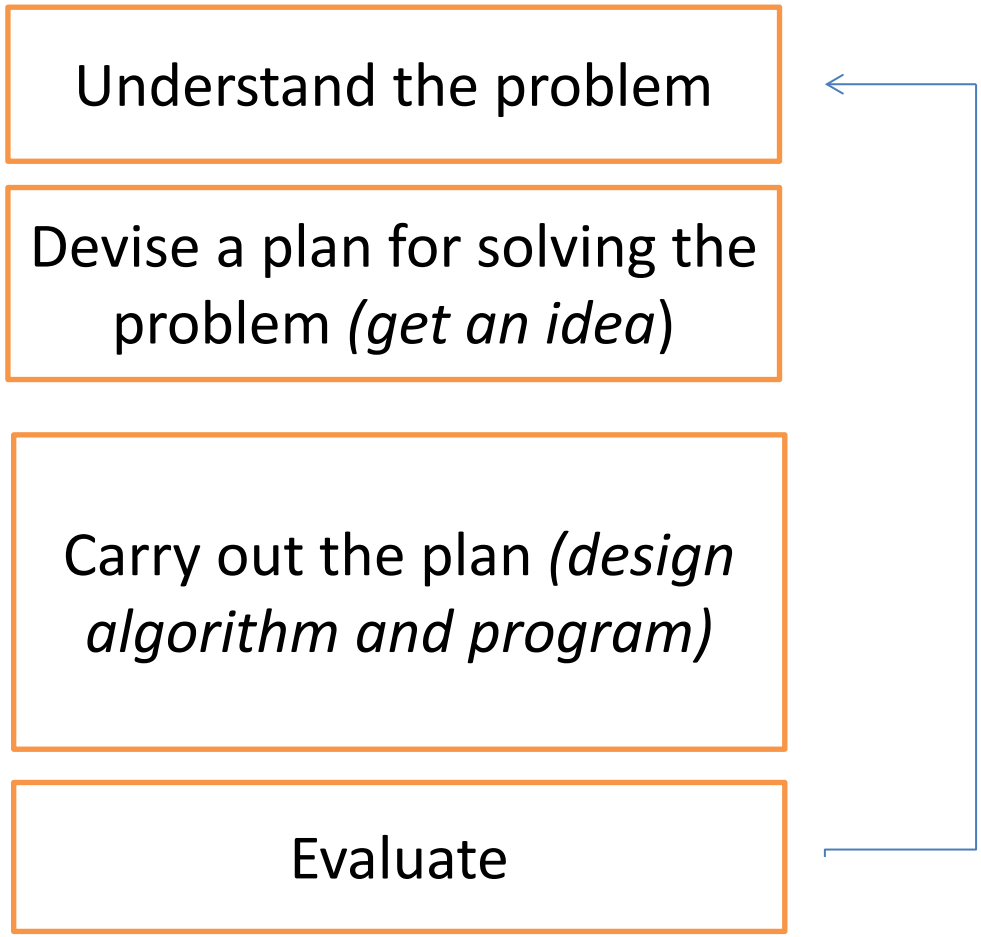
1+

3(n-1) + 3(1+2+3+4+. . .+n−1) + 2(n-1)=

1 + 5(n-1) + 3n(n-1)/2 → O(n^2)

Maybe there is a better/faster algorithm?

Algorithms and Problem Solving



George Polya's 4 problem solving phases

Algorithms and Problem Solving

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

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.

Getting a foot in the door

Problem Solving: Pianoexample

(1,1,36)
(1,2,18)
(1,3,12)
(1,4,9)
(1,6,6)
(2,2,9)
(2,3,6)
(3,3,4)

Product of three numbers = 36

Not unique! We need more clues.

Problem Solving: Pianoexample

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

Still not unique! We need more clues.

Was it relevant that one of the childen played piano? Or violin, or chess?

Getting a Foot in the Door

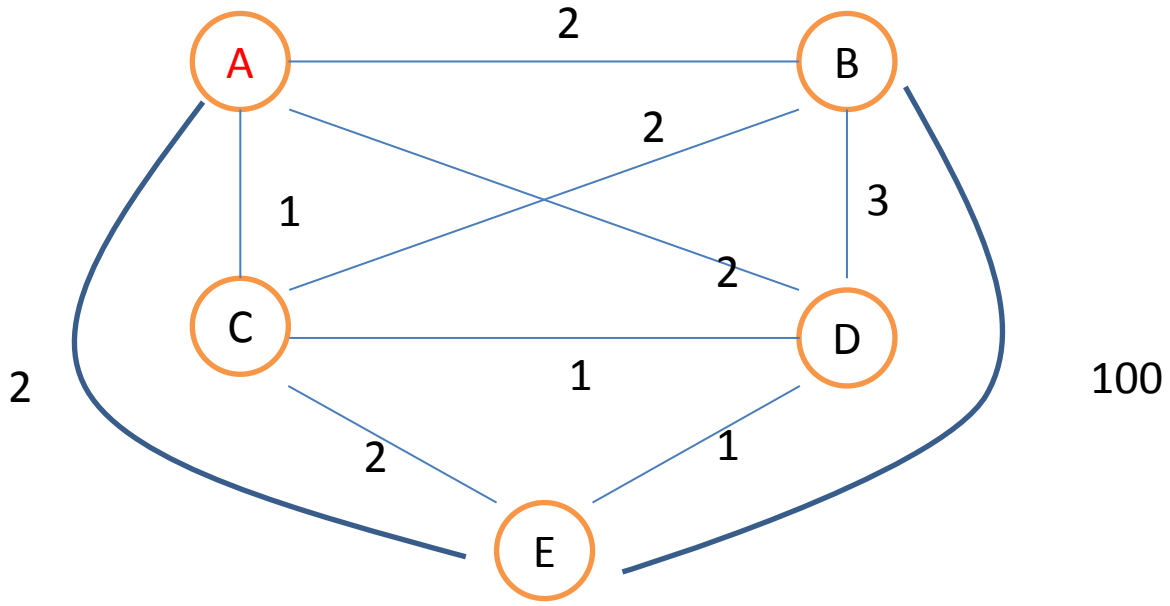
How to start solving a problem/finding an algorithm?

- Top down approach, i.e. stepwise refinement
- Bottom up approach, i.e. solve small parts and combine
- Approach the problem backwards
- Find related problems with known solutions

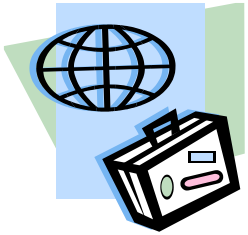
Which method(s) did we use when solving the pianoproblem?

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.

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.

Algorithm Design Principles

- Greedy
- Divide & 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

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

Divide & Conquer

procedure MergeSort(List){

if LengthOfList ≥ 2

then divide list into two halves

MergeSort (RightHalf)

MergeSort (LeftHalf)

Merge (RightHalf, LeftHalf)

O(n log n)

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 such as arrays, lists, etc.

Datastructures and Algorithms

Implementing (and analysing) an algorithm requires datastructures.

Algorithm + suitable ds → fast program

Algorithm + unsuitable ds → slow program

Tight relation of datastructures and algorithms:

Some algorithms are inspired by the existence of special datastructures OR 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: Printing in reverse order, bookeeping in backtracking procedures (e.g. for TSP), processing tasks in LIFO (last in first out) manner.

- *Possible operations:* add or remove a task
- *Implementation:*
  - 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)

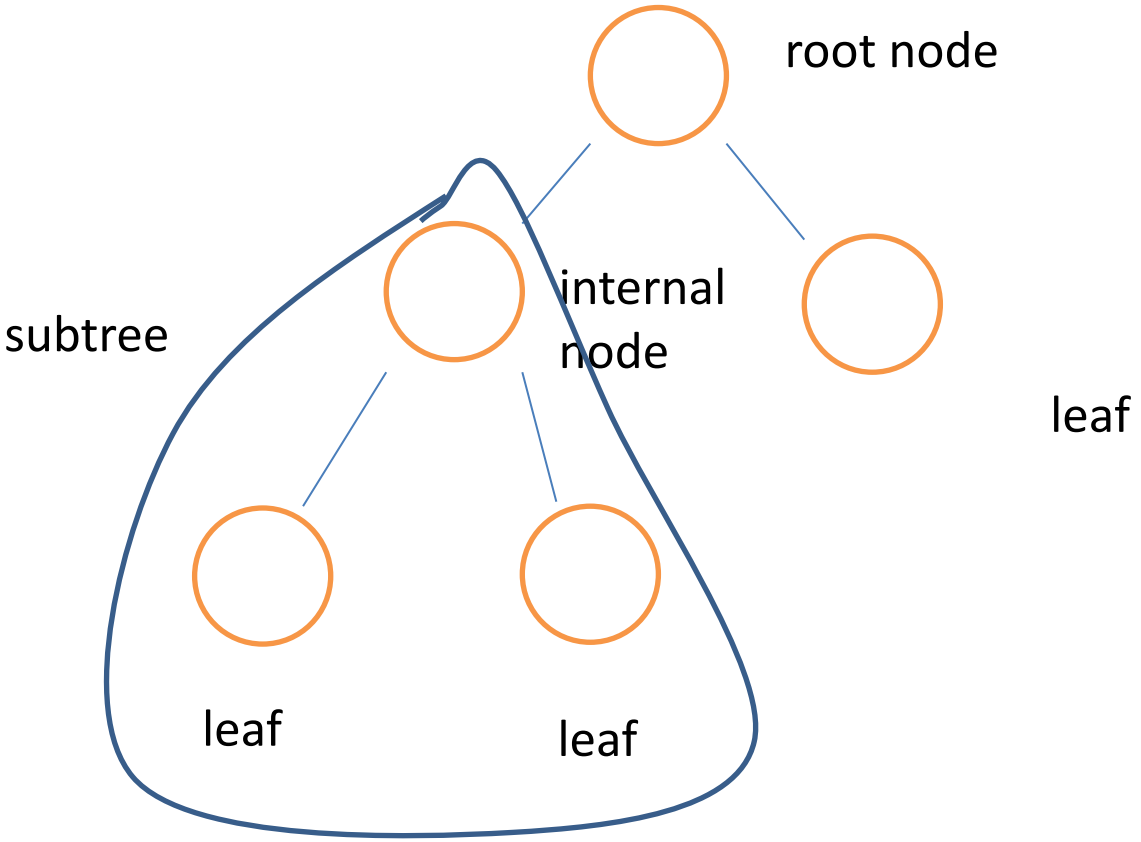
Implemented as static or dynamic arrays

Queues

Example: Queue at apoteket, tasks performed in FIFO (first in first out) manner.

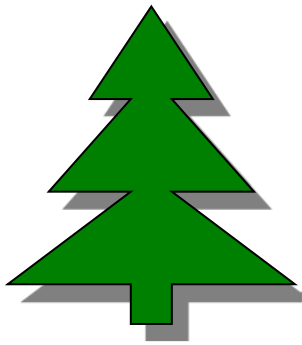
- *Possible operations:* remove a task from the front or add a task at the *tail/rear*.
- *Implementation:*
  - as a double linked list
  - as a static/dynamic array using head- and tail pointer
  - Active queues, even small ones, consume a lot of memory → *cyclic queues*.

Trees



Parents,
children,
siblings

Trees



Most common: ***Binary search trees***

- Fast add and remove into a structure that sorts itself (*balanced search trees*).
- Max two children per node
- For each node it holds that all nodes in the left subtree have value less or equal than, and all nodes in the right subtree have value greater 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 (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’

Jeppesen: A Company built on Algorithms

- Jeppesen, Boeing, Gothenburg
- Software for Airline Scheduling problems (crew pairing, rostering and fleet assignment)
- One of their oldest products, a crew pairing solver, contains a ~20-year old algorithm designed by staff from the Computing Science department at Chalmers.
- 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) ← 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
- Mehlhorn och Näher: LEDA