ALGORITHMS AND DATA STRUCTURES I

Course 3

Previous Course

Some simple examples

Subalgoritms

Specification

Usage

Course content

WHAT IS THE ANALYSIS OF THE EFFICIENCY OF ALGORITHMS?

HOW CAN THE EFFICIENCY OF ALGORITHMS BE MEASURED?

EXAMPLES

MOST FAVORABLE, WORST CASE AND MEDIUM CASE ANALYSIS

What is the analysis of the efficiency of algorithms?

- Analysis of the efficiency of algorithms means:
  - estimating the amount of computing resources needed to execute algorithms

- Note: the term complexity analysis is sometimes used

- Utility
  - efficiency analysis is useful to compare algorithms with each other and obtain information on the computing resources needed to execute algorithms

What is the analysis of the efficiency of algorithms? Resources

- Computing resources
  - Memory space
    - space required to store data processed by the algorithm
  - Execution time
    - time required for the execution of processing within the algorithm

- Efficient algorithm
  - algorithm that requires a reasonable amount of computing resources

- If one algorithm uses fewer computing resources than another algorithm, then it is considered more efficient

What is the analysis of the efficiency of algorithms?

- Efficiency types
  - Efficiency in relation to memory space
    - refers to the memory space required to store all data processed by the algorithm
  - Efficiency in relation to execution time
    - refers to the time required to execute the processing in the algorithm
- Both types of algorithm efficiency analysis are based on the following assumption:
  - The amount of computing resources required depends on the volume and/or characteristics of the input data
    => the size of the problem
- The purpose of efficiency analysis is to answer the question:
  - How does the amount of resources needed depend on the size of the problem?

How to determine the size of the problem?

- ... usually very simple, starting from the problem statement and the properties of the input data;
- Problem size = amount of memory required to store all problem input data
- The size of the problem is expressed in one of the following:
  - Number of components (real values, integer values, characters, etc.) of the input data
  - The number of bits (internal representation of the data type) needed to store input data
- The choice is made depending on the level at which the processing is done ...

How to determine the size of the problem?

- Examples:
  - Determining the minimum value stored into an array *x[1..n]*
    - Problem size: **n**
    - Note: the number of elements of the array
  - Calculation of the value of a polynomial of order *n*
    - Problem size: **n**
    - Note: the number of coefficients is actually *n+1*
  - Calculation of the sum of two matrices with *m* lines, and *n* columns
    - Problem size: ***(m,n)*** or ***m·n***
    - Note: the number of elements of the matrix
  - Check the primality of a number, *n*
    - Problem size: **n** or **$\log_2 n$**
    - Note: the number of bits required to store the value of *n* is actually $\lfloor log_2 n \rfloor + 1$ ($\lfloor . . \rfloor$ represents the lower integer part of a real number)

Course content

WHAT IS THE ANALYSIS OF THE EFFICIENCY OF ALGORITHMS?

HOW CAN THE EFFICIENCY OF ALGORITHMS BE MEASURED?

EXAMPLES

MOST FAVORABLE, WORST CASE AND MEDIUM CASE ANALYSIS

How can the efficiency of algorithms be measured?

- Next we will refer only to the analysis of efficiency in terms of execution time.

- For the (theoretical) estimation of the execution time it is necessary to establish:
  - A calculation model
  - A unit of measurement of execution time

How can the efficiency of algorithms be measured?

- Calculation model: random access machine (Random Access Machine = RAM)

- Features (simplifying assumptions):
  - All processing is carried out sequentially (there is no parallelism in the execution of the algorithm)

  - The execution time of an elementary processing does not depend on the values of the operands (the execution time required to calculate 1+2 does not differ from the execution time required for 12433+4567)

  - The time required to access the data does not depend on the location of the data in memory (there is no difference between the time required to process the first element of an array and that of processing the last element) – we will return to this restriction when we discuss the analysis of the processing performed on different data structures (in context of amortized analysis)

Way simplifying assumptions?

- Execution time can depend also on
  - Speed of the computer on which the program will run (e.g., a smartphone vs. a Google server)
  - The programming language used to implement the algorithm (e.g., C vs. Python)
    - Different optimizations of the programming languages
    - Interpreter, garbage collector
  - The efficiency of the implementation, for the specific machine

Python code

```
dim = 50000
x = [0] * dim
 y = [0] * dim
for i in range(dim):
   y[i] = i * 1.0
for i in range(dim):
    for j in range(dim):
          x[i] += y[j]
```

Is the same algorithm implemented in both languages?

Is the running time the same?

C code

```
int main() {
  int dim = 50000;
  float x[dim], y[dim];
  int i;
  for (i = 0; i < dim; i++) {
       y[i] = i * 1.0;
  }
  for (i = 1; i < dim; i++) {
     for (int j = 1; j < dim; j++) {
          x[i] += y[j];
     }
  }
}
```

How can the efficiency of algorithms be measured?

- Unit of measurement = time required to carry out basic processing (basic operation)

- Basic operations
  - Assign
  - Arithmetic operations (addition, subtraction, multiplication, division)
  - Comparisons
  - Logical operations

- Algorithm execution time = number of elementary operations executed

- Remark: The execution time estimate is aimed at determining the dependence between the number of elementary operations executed and the problem size

Course content

WHAT IS THE ANALYSIS OF THE EFFICIENCY OF ALGORITHMS?

HOW CAN THE EFFICIENCY OF ALGORITHMS BE MEASURED?

EXAMPLES

MOST FAVORABLE, WORST CASE AND MEDIUM CASE ANALYSIS

Example. Sum of the elements of an array

Input data: x[1..n], n>=1

Output: S=x[1]+x[2]+…+x[n]

Problem size: n

Algorithm

```
Sum(x[1..n])
1: S←0
2: i ← 0
3: WHILE i<n DO
4:         i ← i+1
5:         S ← S+x[i]
6:  ENDWHILE
7: RETURN S
```

Cost table

Operation	Cost	Times
1	c1	1
2	c2	1
3	c3	n+1
4	c4	n
5	c5	n

Execution time:

$T(n)=(c3+c4+c5)n+(c1+c2+c3)= a*n +b$

Remark: Some of the costs are identical (C1=C2) and others can be considered different (C5>C1). It is necessary such a detailed analyses – in general case no, use the simplifying assumptions.

Example. Sum of the elements of an array

- Remarks
  - The simplest option is to consider all basic processing as having the same cost
    - If in the previous example it is considered that all elementary operations have unit cost, then the following estimate for execution time is obtained: $T(n)=3(n+1)$
  - The constants that intervene in the expression of execution time are not very important. The important element is that the execution time depends linearly on the size of the problem.
- The previous algorithm is equivalent to:

```
S ← 0
FOR i ← 1,n DO S ← S + x[i] ENDFOR
```

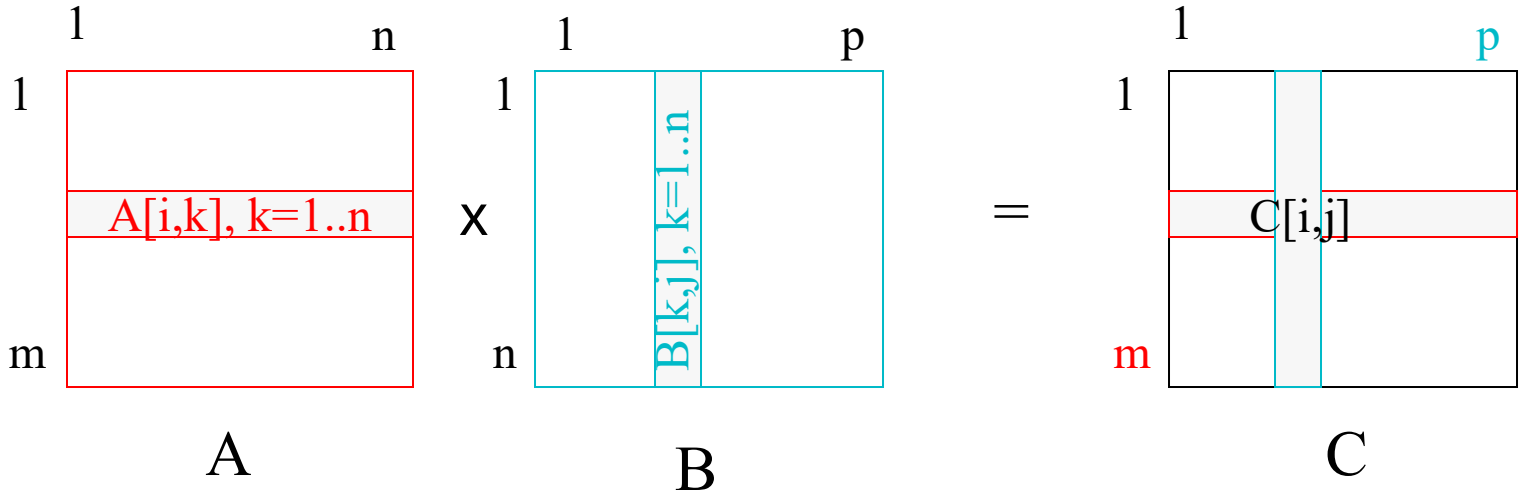
  - Note that the management of the FOR cycle counter involves the execution of $2(n+1)$ operations; the other $(n+1)$ operations correspond to the calculation of the sum (initialization of S and update from each repetition of the cycle body)

Example. Matrix multiplication

Input data: $A_{m\times n}, B_{n\times p}$

Output: C=A*B

Problem size: (m,n,p)



$$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}$$

C[i,j]=A[i,1]*B[1,j]+A[i,2]*B[2,j]+…+A[i,n]*B[n,j],
i=1..m, j=1..p

Example. Matrix multiplication

Basic idea: for each *i=1..m* and *j=1..p* calculate the sum after *k*

Algorithm

```
Product(A[1..m,1..n],B[1..n,1..p])
1:  FOR i ← 1,m DO
2:   FOR j ← 1,p DO
3:       C[i,j] ← 0
4:       FOR k ← 1,n DO
5:           C[i,j] ← C[i,j]+A[i,k]*B[k,j]
6:       ENDFOR
7:    ENDFOR
8:  ENDFOR
9: RETURN C[1..m,1..p]
```

Cost table

Operation	Cost	Times.	Total
1	2(m+1)	1	2(m+1)
2	2(p+1)	m	2m(p+1)
3	1	mp	mp
4	2(n+1)	mp	2mp(n+1)
5	2	mpn	2mnp

T(m,n,p)=4mnp+5mp+4m+2

Example. Matrix multiplication

Algorithm

```
Product(A[1..m,1..n],B[1..n,1..p])
1:  FOR i ← 1,m DO
2:   FOR j ← 1,p DO
3:       C[i,j] ← 0
4:       FOR k ← 1,n DO
5:           C[i,j] ← C[i,j]+A[i,k]*B[k,j]
6:       ENDFOR
7:    ENDFOR
8:  ENDFOR
9: RETURN C[1..m,1..p]
```

Note: As a rule, it is not necessary to fill in the entire cost table, but it is enough to count only the dominant operation

Dominant operation: the most common (expensive) operation

Execution time

estimated: $T(m,n,p)=mnp$

Example. Minimum element of an array

Input data: x[1..n], n>=1

Output: m=min(x[1..n])

Problem size: n

Algorithm:

```
Minimum(x[1..n])
1:  m ← x[1]
2:   FOR i ← 2,n DO
3:       IF x[i]<m THEN
4:           m ← x[i]
5:       ENDIF
6:   ENDFOR
7:RETURN m
```

Cost table:

Operation	Cost	Times	Total
1	1	1	1
2	2n	1	2n
3	1	n-1	n-1
4	1	t(n)	t(n)

T(n)=3n+t(n)

t(n) = number of changes of variable *m* (depends on how the elements of the array are arranged)

Remark: The execution time depends not only on the size of problem but also the properties of input data

Example. Minimum element of an array

- If the execution time depends on the properties of the input data, then at least the extreme cases should be analyzed:
  - Most favorable case:

    x[1]<=x[i], i=1..n => t(n)=0 => T(n)=3n
  - Worst-case scenario:

    x[1]>x[2]>...>x[n] => t(n)=n-1 => T(n)=4n-1

  It results that: 3n<=T(n)<=4n-1
- Both the lower limit and the superior limit linearly depend on the size of the problem
- The variant of analysis that takes into account only the dominant operation, the comparison (*x[i]<m*), leads to T(n) = n-1 (regardless of the position of the minimum value, all elements of x must be analyzed)

Algorithm

```
Minimum(x[1..n])
1:  m ← x[1]
2:  FOR i ← 2,n DO
3:      IF x[i]<m THEN
4:          m ← x[i]
5:      ENDIF
6:  ENDFOR
7:RETURN m
```

Example. Seach an element in an array

Input data: x[1..n], n>=1

Output: “the value is found” or ”the value is not found”

Rezult: the logical variable "found" contains the truth value of the statement "the value v is in array x[1..n]"

Problem size: n

Algorithm (sequential search):

```
search(x[1..n],v)
1:  found ← False
2:  i ← 1
3:  WHILE (found==False) AND (i<=n) DO
4:      IF x[i]==v                          //t1(n)
5:          THEN found ← True               //t2(n)
6:          ELSE  i ← i+1                   //t3(n)
7:      ENDIF
8:  ENDWHILE
9: RETURN found
```

Cost table:

Operation	Times
1	1
2	1
3	t1(n)+1
4	t2(n)
5	t3(n)

Example. Seach an element in an array

The execution time depends on the properties of array x[1..n].

Case 1: the value v belongs to the array (consider k the smallest index with the property that x[k]= v)

Case 2: value v is not in the array

$$t1(n)=\begin{cases} k & \text{if v in } x[1..n] \\ n & \text{if v is not in } x[1..n] \end{cases}$$

$$t2(n)=\begin{cases} 1 & \text{if v is in } x[1..n] \\ 0 & \text{if v is not in } x[1..n] \end{cases}$$

$$t3(n)=\begin{cases} k-1 & \text{if v is in } x[1..n] \\ n & \text{if v is not in } x[1..n] \end{cases}$$

Algorithm (sequential search):

```
search(x[1..n],v)
1:  found ← False
2:  i ← 1
3:  WHILE (found==False) AND (i<=n) DO
4:      IF x[i]==v                          //t1(n)
5:          THEN found ← True               //t2(n)
6:          ELSE  i ← i+1                   //t3(n)
7:      ENDIF
8:  ENDWHILE
9: RETURN found
```

Example. Seach an element in an array

Best-case scenario: x[1]=v

t1(n)=1, t2(n)=1, t3(n)=0

T(n)= 6

Worst-case scenario: v is not in x[1..n]

t1(n)=n, t2(n)=0, t3(n)=n

T(n)=3n+3

The edges (lower and upper) of execution time are:

6<= T(n) <= 3(n+1)

Note: the lower limit is constant and the upper limit depends linearly on the problem size

$$t1(n)=\begin{cases} k & \text{if v in } x[1..n] \\ n & \text{if v is not in } x[1..n] \end{cases}$$

$$t2(n)=\begin{cases} 1 & \text{if v is in } x[1..n\ ] \\ 0 & \text{if v is not in } x[1..n] \end{cases}$$

$$t3(n)=\begin{cases} k-1 & \text{if v is in } x[1..n\ ] \\ n & \text{if v is not in } x[1..n] \end{cases}$$

Example. Seach an element in an array – other variant

Algorithm (sequential search):

```
search2(x[1..n],v)
1:  i ← 1
2:  WHILE i<n  AND x[i]!=v DO
3:      i ← i+1
4:  ENDWHILE
5: IF x[i]!=v THEN found ← False
6:                ELSE found ← True
7: ENDIF
8: RETURN found
```

The most favorable case (v - value is in first position):

$T(n)=5$

Worst case scenario (v - value is in last position or not in the array):

$T(n)=1+n+(n-1)+2=2n+2$

If the comparison x[i]!=v is considered as the dominant operation, then:

- Best-case scenario: $T(n)=2$
- Worst-case scenario: $T(n)=n+1$

Course content

WHAT IS THE ANALYSIS OF THE EFFICIENCY OF ALGORITHMS?

HOW CAN THE EFFICIENCY OF ALGORITHMS BE MEASURED?

EXAMPLES

MOST FAVORABLE, WORST CASE AND MEDIUM CASE ANALYSIS

Best case and worst case analysis

Best-case analysis (most favorable)

- Provides a lower margin for execution time
- It allows to identify inefficient algorithms (if an algorithm has a high cost, even in the most favorable case, then it is not an acceptable solution)

Worst case analysys (most unfavorable)

- Provides the highest execution time relative to all input data dimensions (represents an upper margin of execution time)
- The upper margin of execution time is more important than the lower margin

Analysis of the medium case

For certain problems, the most favorable and unfavorable cases are rare cases (exceptions)

Thus… the best and worst case execution time does not provide sufficient information

In these cases, another analysis is carried out… Medium case analysis

The purpose of this analysis is to provide information on the behavior of the algorithm in the case of arbitrary input data (which does not necessarily correspond to either the most favorable or the worst case)

Analysis of the medium case

This analysis is based on knowledge of the probability distribution of input data

This means knowing (estimating) the probability of occurrence of each of the possible instances of the input data (how frequently each of the possible values of the input data occurs)

The average execution time is the average value (in the statistical sense) of the execution times corresponding to the different instances of the input data

Analysis of the medium case

- Assumptions. We assume that the following assumptions are satisfied:
  - Input data can be grouped into classes so that the execution time corresponding to data of the same class is the same
  - are $m=M(n)$ classes with input data
  - The probability of occurrence of a k-class data is P_k
  - The execution time of the algorithm for input data belonging to class k is $T_k(n)$

- In these assumptions the average execution time is:

$$T_{average}(n)= P_1T_1(n)+P_2T_2(n)+...+P_mT_m(n)$$

- Remark: if all data classes have the same probability then the average execution time is:

$$T_{average}(n)=(T_1(n)+T_2(n)+...+T_m(n))/m$$

Analysis of the medium case

- Example: sequential search (dominant operation: comparison)

- Assumptions about the probability distribution of input data:
  - Probability that the value v is in the array: p
    - The value v appears with the same probability on each of the positions in the table
    - The probability that the value v is at position k is 1/n
  - Probability that v is not in the array: 1-p

  (in the case of alg search2 if the searched value is at position k then the number of comparisons performed is (k+1))

  $T_{average}(n)=p(2+3+...+(n+1))/n+(1-p)(n+1)=p(n+1)/2+(1-p)n =(1-p/2)n+p/2+p/(2n)$

  If p=0.5 we get $T_{average}(n)=3/4\ n+1/4+1/4n$

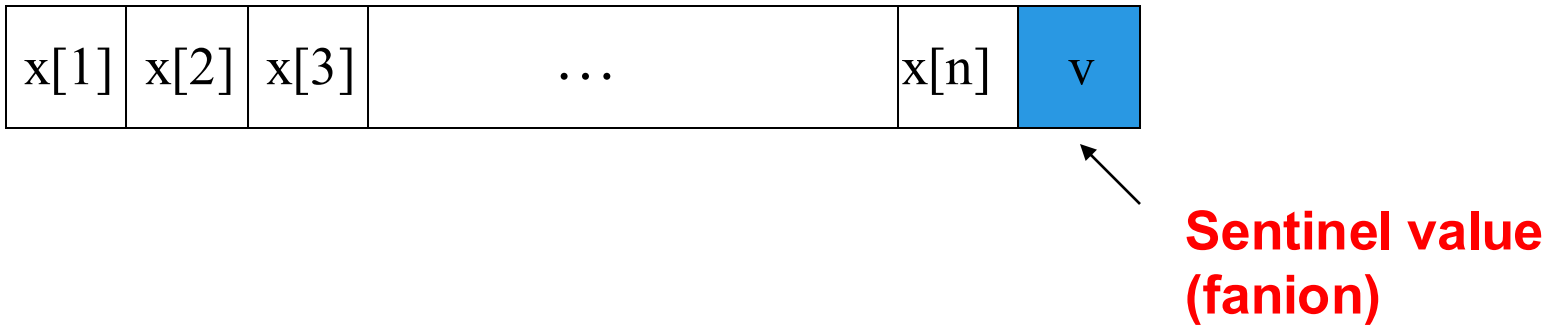
- Conclusion: The average execution time of the sequential search algorithm is linearly dependent on the size of the input data

Analysis of the medium case

Example: Sequential search (sentinel or pennant)

Basic idea:

- The table is extended by an additional element (position n+1) whose value is v
- The table is completed until the value v is reached (the value will be found in the worst case at position n+1 – corresponds to the situation when the table x[1..n] does not contain the value v)



Analysis of the medium case

Algorithm:

```
Search_ sentinel(x[1..n],v)
i ← 1
WHILE x[i]!=v DO
    i ← i+1
ENDWHILE
RETURN i
```

Dominant operation: comparison

Hypothesis: The probability that the value v is at position k in {1,2,...,n+1} is 1/(n+1)

Average execution time:

$T_{average}(n) = (1+2+...+(n+1))/(n+1) = (n+2)/2$

Remark:

Changing the assumption about the distribution of input data, the mean time value changes (in the case of sequential search, the dependence on the size of the problem remains linear);

Remark. The average execution time is NOT necessarily the arithmetic average between execution times corresponding to extreme cases (most favorable and unfavorable).

Abstract: stages of efficiency analysis

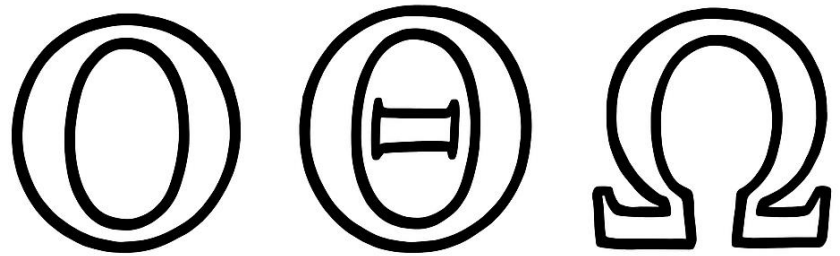
- Step 1: Identify the size of the problem
- Step 2: Establish the dominant operation
- Step 3: Determine the number of executions of the dominant operation
- Step 4. If the number of executions of the dominant operation depends on the properties of the input data, then it is analyzed:
  - The most favorable case
  - Worst-case scenario
  - Medium case
- Step 5. The order (class) of complexity is established (see next course)

Next course

.. All about analyzing the efficiency of algorithms.

More specifically, the following will be presented:

- Growth order
- Asymptotic notations (O, Theta, Omega)
- Complexity classes



Q&A