Tutorial 2: Paradigm

Artificial Intelligence

G.Guérard

Divide and conquer

Exercise 1

Write a pseudo code for a divide-and-conquer algorithm for finding the position of the largest element in an array of n numbers. Write a pseudo code for a brute-force algorithm, compare with the previous one. Show a tree of the divide-and-conquer algorithm's process. What is the max level of the tree for n numbers?

Exercise 2

Write a pseudo code for a divide-and-conquer algorithm for the exponentiation problem of computing a^n where $a > 0$ and n is a positive integer. Write a pseudo code for a brute-force algorithm, compare with the previous one. Show a tree of the divide-and-conquer algorithm's process. What is the max level of the tree for n not given? Verify termination, correctness and completeness.

Dynamic programming: data mining

Exercise 3

You are given an array with integers (positive, negative, zero) and you are supposed to find the maximum contiguous sum in this array. Hence, you have to find a sub-array which results in the largest sum. Example, if the given array is: 5, -6, 7, 12, -3, 0, -11, -6

Exercise 4

Use the Longest Common Substring for : BDCABA and ABCBDAB

Let $M[i,j]$ be the number of letters to the left of (and including) a_i complying with the same number of letters to the left (and including) b_j. The length of the longest common string is the largest number in the matrix M.

M can be defined recursively as follows:

$$M[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0, \\ M[i-1, j-1] + 1 & \text{if } a_i = b_j, \\ 0 & \text{otherwise.} \end{cases}$$

The following algorithm computes M and returns the largest number in M.

```
max ← 0
for j ← 0  to n
    M[0, j] ← 0
for i ← 1  to m
    M[i, 0] ← 0
    for j ← 1  to n
        if a_i = b_j then
            M[i, j] ← M[i − 1, j − 1] + 1
            if M[i, j] > max then max ← M[i, j]
        else M[i, j] ← 0
return max
```

Exercise 5

Adapt the Longest Common Substring to the Longest Common Subsequence: BDCABA and ABCBDAB. The longest common subsequence (LCS) problem is the problem of finding the longest subsequence common to all sequences in a set of sequences (often just two sequences). It differs from problems of finding common substrings: unlike substrings, subsequences are not required to occupy consecutive positions within the original sequences.

Exercise 6

The Levenshtein distance is a string metric for measuring the difference between two sequences. Informally, the Levenshtein distance between two words is the minimum number of single-character edits (i.e. insertions, deletions or substitutions) required to change one word into the other. Propose a dynamic program to solve this problem. Test with $meilenstein$ and $levenshtein$.