Practice problems: Dynamic Programming and Greedy algorithms

1. Consider the numbers $(A_n)_{n>0} = (1, 1, 3, 4, 8, 11, 21, 29, 55, \ldots)$ defined as follows:

$$\begin{array}{ll} A_1 = A_2 = 1 & \\ A_n = B_{n-1} + A_{n-2} & n > 2 \\ B_1 = B_2 = 2 & \\ B_n = A_{n-1} + B_{n-2} & n > 2 \end{array}$$

   A_n can be computed using the following recursive procedures:

```
ComputeA(n)
   if n<3 then
      return 1
   else
      return ComputeB(n-1)+ComputeA(n-2)
   fi
end

ComputeB(n)
   if n<3 then
      return 2
   else
      return ComputeA(n-1)+ComputeB(n-2)
   fi
end
```

   (a) Show that the running time $T_A(n)$ of `ComputeA(`n`)` is exponential in n. (*Hint:* Show for example that $T_A(n) = \Omega(2^{n/2})$)

   (b) Describe and analyze a more efficient algorithm for computing A_n.

2. **(Duke final 2001)** A *palindrome* is a string that reads the same from front and back. Any string can be viewed as a sequence of palindromes if we allow a palindrome to consist of one letter.

> **Example:** "bobseesanna" can e.g be viewed as being made up of palindromes in the following ways:
> "bobseesanna" = "bob" + "sees" + "anna"
> "bobseesanna" = "bob" + "s" + "ee" + "s" + "anna"
> "bobseesanna" = "b" + "o" + "b" + "sees" + "a" + "n" + "n" + "a"

   We are interested in computing $MinPal(s)$ defined as the minimum number of palindromes from which one can construct s (that is, the minimum k such that s can be written as $w_1 w_2 \ldots w_k$ where $w_1, w_2, \ldots, w_k$ are all palindromes).

Example: $MinPal$("bobseesanna")=3 since "bobseesanna" = "bob" + "sees" + "anna" and we cannot write "bobseesanna" with less than 3 palindromes.

We can compute $MinPal(s)$ using the following formula

$$MinPal(s[i,j]) = \begin{cases} 1 & \text{if } s[i,j] \text{ is palindrome,} \\ \min\limits_{i \le k < j} \{MinPal(s[i,k]) + MinPal(s[k+1,j])\} & \text{otherwise} \end{cases}$$

which can be implemented as follows:

```
MinPal(i,j)
  b=i, e=j
  WHILE b<e and s[b]=s[e] DO
    b=b+1
    e=e-1
  OD
  IF b>=e THEN RETURN 1  /* s[i,j] is not palindrome */
  min=j-i+1
  FOR k=i to j-1 DO
    r=MinPal(i,k)+MinPal(k+1,j)
    IF r<min THEN min=r
  END
  RETURN min
END
```

(a) Show that the running time of $MinPal(s)$ is exponential in the length n of s.

(b) Describe an $O(n^3)$ algorithm for solving the problem.

3. In this problem we consider a piece of squared paper where each square is either empty or contains a cross. We represent such a piece a paper using an $n \times n$ array A, with $A[i,j]$ =true if the i,j'th position contains a cross ($A[1,1]$ corresponds to the lowest left corner of the paper).

We are interested in computing for each position the maximal number of crosses in a—vertical, horizontal, or diagonal—sequence (i.e. adjacent crosses) passing through that particular position. The result should be stored in an array B.

Here is an example of such a problem and its solution.

A

×	×				×			×	×
		×			×			×	
			×				×		
		×				×		×	
×	×	×			×		×	×	
		×		×		×			×
		×			×				
	×	×		×					×
		×				×			
		×	×		×			×	

j

i

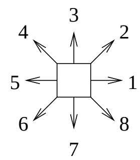
B

2	3	0	0	0	2	0	0	2	6
0	0	3	0	0	2	0	0	6	0
0	0	0	3	0	0	0	6	0	0
0	0	7	0	0	0	6	0	5	0
3	3	7	0	0	6	0	5	2	0
0	0	7	0	6	0	5	0	0	2
0	0	7	0	0	5	0	0	0	0
0	3	7	0	5	0	0	0	0	1
0	0	7	0	0	0	2	0	0	0
0	0	7	3	0	2	0	0	1	0

j

i

The following program solves the problem. Note that for convenience the different directions are numbered as follows:




```
for i=1 to n do
  for j=1 to n do
     B[i,j]=Count1(i,j)
  end
end

Count1(i,j)
   if (not A[i,j]) then
      return 0
   else
      return max(Count2(1,i,j)+Count2(5,i,j)-1,
                 Count2(2,i,j)+Count2(6,i,j)-1,
                 Count2(3,i,j)+Count2(7,i,j)-1,
                 Count2(4,i,j)+Count2(8,i,j)-1)
end

Count2(d,i,j)
   if (i<1) or (j<1) or (i>n) or (j>n) or (not A[i,j]) then
      return 0
   else if d=1 return 1+Count2(1,i+1,j)
   else if d=2 return 1+Count2(2,i+1,j+1)
   else if d=3 return 1+Count2(3,i,j+1)
   else if d=4 return 1+Count2(4,i-1,j+1)
   else if d=5 return 1+Count2(5,i-1,j)
   else if d=6 return 1+Count2(6,i-1,j-1)
   else if d=7 return 1+Count2(7,i,j-1)
   else if d=8 return 1+Count2(8,i+1,j-1)
   end
end
```

(a) Analyze the running time of the program.

(b) Describe an optimal $O(n^2)$ algorithm.

4. We want to write a sentence on a floor using prefabricated tiles. Unfortunately, we cannot buy tiles with single letters and we cannot write all sentences with the available tiles—see Figure 1 for an example. Given a sentence S of length n and a set of m tile types $T = \{t_0, t_1, \ldots, t_{m-1}\}$ we want to decide if it is possible to write S (assuming an unlimited number of tiles). We can solve the problem with the following procedure (using the call **Write**$(0, n-1)$):

Write(i, j)

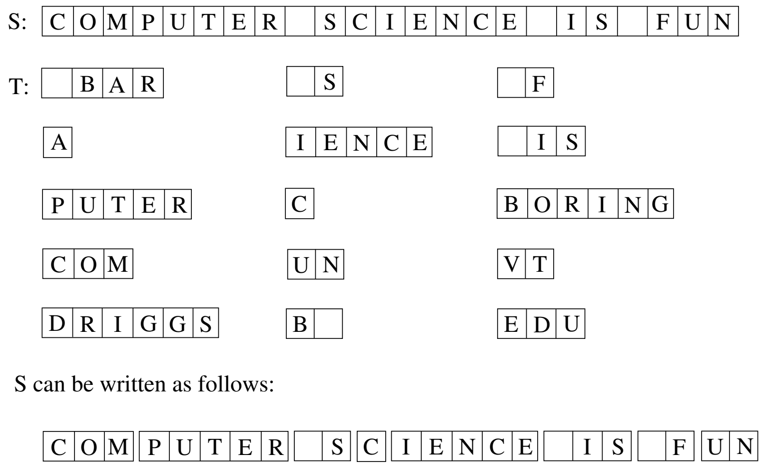


Figure 1: Writing a sentence S using tiles T.

> If $i > j$ THEN return TRUE
> FOR $k = 0$ to $m - 1$ DO
>   IF $S(i \ldots j) = t_k$ THEN return TRUE
> For $l = i$ to $j - 1$ DO
>   IF **Write**(i, l) AND **Write**$(l + 1, j)$ THEN return TRUE
> return FALSE

END **Write**

Here $S(i \ldots j)$ denote the subsentence of S from character i to character j (including both characters). We assume that the test $S(i \ldots j) = t_k$ takes time $O(j - i + 1)$.

(a) Show that the running time of the algorithm is $\Omega(2^n)$.

(b) Design and analyze a more efficient algorithm.

5. **(Duke final 2002)** Let $x = x_1x_2 \ldots x_n$ and $y = y_1y_2 \ldots y_m$ and $z = z_1z_2 \ldots z_{n+m}$ be three strings of length n, m, and $n + m$, respectively. We say that z is a *merge* of x and y if x and y can be found as two disjoint subsequences in z.

   Example: *algo*datastruc*ritu*th*res*ms is a merge of *algorithms* and datastructures.

   For $0 \leq i \leq n$ and $0 \leq j \leq m$, $Merge(i, j)$ is TRUE if $z = z_1z_2 \ldots z_{i+j}$ is a merge of $x = x_1x_2 \ldots x_i$ and $y = y_1y_2 \ldots y_j$ ($x = x_1x_2 \ldots x_i$ is the empty string if $i = 0$. Similarly for y and z.)

   We can compute $Merge(i, j)$ using the following formula

$$Merge(i,j) = \begin{cases} X_{ij} \vee Y_{ij} & \text{if } i, j \geq 1 \\ X_{ij} & \text{if } i \geq 1, j = 0 \\ Y_{ij} & \text{if } i = 0, j \geq 1 \\ \text{TRUE} & \text{if } i = 0, j = 0 \end{cases}$$

where X_{ij} is defined as

$$(z_{i+j} = x_i) \wedge Merge(i-1, j)$$

and Y_{ij} is defined as

$$(z_{i+j} = y_j) \wedge Merge(i, j-1)$$

This can be implemented as follows

```
Merge(i,j)
  IF i=0 AND j=0 THEN RETURN True
  IF i>0 THEN X = (z[i+j]==x[i] AND Merge(i-1,j))
  IF j>0 THEN Y = (z[i+j]==y[j] AND Merge(i,j-1))
  IF i>0 and j>0 THEN RETURN X OR Y
  IF j=0 THEN RETURN X
  IF i=0 THEN RETURN Y
END
```

(a) Show that the running time of $Merge(n, m)$ is exponential in n and m.

(b) Describe an $O(nm)$ algorithm for solving the problem. Remember to argue for both running time and correctness.

If $Merge(n, m) =$ TRUE we are interested in knowing which subsequence of z corresponds to x and which corresponds to y in a possible merge. We can characterize such a merge by the indexes of z where a new subsequence starts.

Example: The merge of *algorithms* and datastructures into *algo*datastruc*rit*u*th*res*ms* is described by the indices $1, 5, 14, 16, 18, 20, 23$.

(c) Describe how your $O(nm)$ algorithm can be extended such that if $Merge(n, m) =$TRUE, the algorithm also returns the list of indexes defining a possible merge.