PAPER CODE NO.	EXAMINER : Professor Leslie Goldberg
COMP309	DEPARTMENT : Computer Science Tel. No. 0151 795 4255

THE UNIVERSITY
of LIVERPOOL

JANUARY EXAMINATIONS 2007

Bachelor of Arts : Year 3
Bachelor of Science : Year 3
No qualification aimed for: Year 1

Efficient Sequential Algorithms

TIME ALLOWED : Two Hours and a Half

INSTRUCTIONS TO CANDIDATES

Please answer four of the following five questions. If you answer more than four questions, then credit will be given to the best four answers. Each question is worth 25 marks.

THE UNIVERSITY
of LIVERPOOL

1. Greedy algorithms: the Activity Selection problem.

An instance is a set $S = \{A_1, A_2, \ldots, A_n\}$ of activities. Each activity A_i has a start time s_i and a finish time f_i with $0 \leq s_i < f_i$. Two activities A_i and A_j are compatible if $s_i \geq f_j$ or $s_j \geq f_i$. The goal is to choose a subset of compatible activities that is as large as possible.

(a) Briefly describe an efficient greedy algorithm that solves this problem. **(7 marks)**

(b) Simulate the algorithm on this input.

i	1	2	3	4	5	6	7	8
s_i	4	7	7	10	9	8	15	4
f_i	8	20	9	19	10	16	17	18

(7 marks)

(c) Now suppose that each activity A_i has a weight w_i and the goal is to choose a subset of compatible activities with *maximum total weight*. Consider the following greedy algorithm. Let A be an activity in S with maximum weight. Form a sub-problem P from S by removing activity A and removing any activities that are incompatible with activity A. Recursively choose an optimal solution Y for the instance P. Return $Y \cup \{A\}$. Show that this algorithm does not always return a solution of maximum total weight.

(7 marks)

(d) If we wanted to prove that the algorithm from part (c) was correct, we would try to establish two properties

- There is always an optimal solution for S which includes A, and
- If Y is an optimal solution to P then $Y \cup \{A\}$ is as large as possible amongst feasible solutions for S that include A.

Are either of these properties true for the algorithm from part (c)? If so, which one or ones? **(4 marks)**

2. Pattern matching.

For this question, the alphabet $\mathcal{A}$ will be $\{a, b\}$.

(a) Give a pseudo-code description of the simple brute-force algorithm for finding all instances of a pattern P in a text T.

(7 marks)

(b) Simulate the process of running the algorithm from part (a) on the pattern $P = a\,b\,a\,b$ and the text $T = a\,a\,b\,a\,a\,b\,a\,b$. How many comparisons of characters does the algorithm make when it is run with inputs P and T?

(8 marks)

(c) Recall the finite automaton pattern matching algorithm for a pattern P with length m.

```
FINITE-AUTOMATON-MATCHING (T, δ, m)
    n ← length(T)
    q ← 0
    for i ← 1 to n
        q ← δ(q, T[i])
        if q = m
            print "pattern P occurs with shift i − m in T"
```

Construct the finite automaton (the transition function δ) so that this algorithm can be used to find matches of the pattern P from part (b).

(5 marks)

(d) Simulate the process of running the algorithm from part (c) (using your transition function) on the pattern P and text T from part (b).

(5 marks)

THE UNIVERSITY
of LIVERPOOL

3. Dynamic Programming: Longest Common Subsequence.

Recall the algorithm LCS. Given two sequences X and Y, LCS(X, Y) constructs the tables lcs and b.

```
LCS (X, Y)
    m ← length(X)
    n ← length(Y)
    for i ← 1 to m lcs[i, 0] ← 0
    for j ← 0 to n lcs[0, j] ← 0
    for i ← 1 to m
        for j ← 1 to n
            if x_i = y_j
                lcs[i, j] ← lcs[i − 1, j − 1] + 1
                b[i, j] ← "↖"
            else if lcs[i − 1, j] ≥ lcs[i, j − 1]
                lcs[i, j] ← lcs[i − 1, j]
                b[i, j] ← "↑"
            else
                lcs[i, j] ← lcs[i, j − 1]
                b[i, j] ← "←"
    return lcs and b
```

(a) Let S be the sequence $a\,a\,b$ and let T be the sequence $a\,b\,a\,b$. simulate the algorithm LCS with input $(X, Y) = (S, T)$ to construct the tables lcs and b.

(10 marks)

(b) What is the length of the longest common subsequence of S and T?

(5 marks)

(c) Give an efficient algorithm PRINT-LCS that takes inputs X, b, i and j where X is a length-m sequence and Y is a length-n sequence and $0 \leq i \leq m$ and $0 \leq j \leq n$ and b is the table b produced by LCS(X, Y). The output should be a longest common subsequence of X_i and Y_j.

(5 marks)

(d) Give an efficient algorithm NEW-PRINT-LCS that takes inputs X, Y, lcs, i and j where X is a length-m sequence and Y is a length-n sequence and $0 \leq i \leq m$ and $0 \leq j \leq n$ and lcs is the table lcs produced by LCS(X, Y). The output should be a longest common subsequence of X_i and Y_j.

(5 marks)

4. Matchings.

(a) What is a matching in a graph? **(2 marks)**

(b) Give an example of a *maximal* matching that is not a *maximum* matching. **(3 marks)**

(c) Consider the following variant of an algorithm that we studied in class. The input is a graph G whose m edges are labelled $e_1, \ldots, e_m$.

$M \leftarrow \emptyset$
For $i \leftarrow 1$ to m
    if e_i is not adjacent to any $f \in M$
        $M \leftarrow M \cup \{e_i\}$
return M

Give an example of a graph G with edge labels $e_1, \ldots, e_m$ so that, when the algorithm is run with input G, the output M is a maximal matching of G that is not a maximum matching of G. **(5 marks)**

(d) Simulate the algorithm on your example from Part (c). **(5 marks)**

(e) Give an efficient algorithm for finding a maximum matching in a tree. **(5 marks)**

(f) Simulate your algorithm on an example. **(5 marks)**

THE UNIVERSITY
of LIVERPOOL

5. Complexity Theory: NP-Completeness and Approximation.

(a) Define the complexity class P. **(5 marks)**

(b) Define the complexity class NP. **(5 marks)**

(c) What does it mean for a problem to be NP-Complete? **(5 marks)**

(d) Recall the problem **Vertex Cover**, which we showed to be NP-complete.

Vertex Cover

- **Input:** An undirected graph G and an integer k
- **Output:** Is there a set U of k vertices of G such that for every edge (u, v) of G, at least one of u and v is in U?

A Boolean formula is a "monotone-2 clause" if it is the OR of two Boolean variables, for example $(x_1 \vee x_2)$ is a monotone-2 clause and so is $(x_3 \vee x_4)$. There is no negation allowed in monotone 2-clauses. A "monotone 2-SAT formula" is the AND of some monotone-2 clauses. For example, $F = (x_1 \vee x_3) \wedge (x_2 \vee x_3) \wedge (x_4 \vee x_5)$ is a monotone 2-SAT formula. Consider the following computational problem.

Monotone 2-SAT

- **Input:** A Monotone 2-SAT formula F and an integer j
- **Output:** Is there a set of j variables of F, such that if these variables are set to TRUE and all other variables are set to FALSE, every clause is satisfied?

Show that Vertex Cover $\overset{\text{poly}}{\rightarrow}$ Monotone 2-SAT

(5 marks)

(e) How well can the **Vertex Cover** problem be approximated?

(5 marks)