Technische Universität München
Department of Computer Science
Chair for Efficient Algorithms
Prof. Dr. Sami Khuri

WS 1999/2000
Problem Set 7
December 14, 1999

Efficient Algorithms and Data Structures

Problem 1 (4 credits)

a) Show the data structure that results and the answers returned by the FIND-SET operations in the following program. Use the disjoint-set forests representation with union by rank and path compression.

```
for i ← 1 to 16
    do MAKE-SET(x_i)
for i ← 1 to 15 by 2
    do UNION(x_i, x_{i+1})
for i ← 1 to 13 by 4
    do UNION(x_i, x_{i+2})
UNION(x_1, x_5)
UNION(x_11, x_13)
UNION(x_1, x_10)
FIND-SET(x_2)
FIND-SET(x_9)
```

b) Write a nonrecursive version of FIND-SET with path compression.

Problem 2 (7 credits)

Prove that the spanning tree produced by Prim's algorithm and the spanning tree produced by Kruskal's algorithm are minimum spanning trees. Do not assume that every edge has a different weight.

Hint: You may use the fact that given any division of the vertices of a graph into two sets G_1 and G_2, for every shortest edge with one endpoint in G_1 and the other in G_2 there is a minimum spanning tree containing that edge.

Problem 3 (7 credits)

Prove that the following statement is true: In a graph where every edge has a different weight, there is a unique minimum spanning tree, which is the spanning tree found by Kruskal's algorithm.

Problem 4 (6 credits)

The graph K_8 consists of 8 vertices (numbered from 1 to 8), where each pair of vertices is joined by an edge. (So there are 28 edges in K_8.) Give the edge joining vertex i and j $(i \neq j)$ weight 1 if $i + j$ is even, and weight 2 if $i + j$ is odd.

a) Find the cost of a minimum spanning tree.

b) How many distinct minimum spanning trees are there?

c) Find two minimum spanning trees T and T' that contain no common edges. Then show how T can be transformed into T' using operations of the following type: Add an edge $e \notin T$ to T, thus creating a unique cycle, and then remove an arbitrary other edge from the cycle to obtain a new tree.

Problem 5 (8 credits)

An n-*tally* is a circuit that takes n bits as inputs and produces $1 + \lfloor \log n \rfloor$ bits as output. It counts (in binary) the number of bits equal to 1 among the inputs. For example, if $n = 9$ and the inputs are 011001011, the output is 0101. An (i, j)-*adder* is a circuit that has one i-bit input, one j-bit input, and one $[1 + \max(i, j)]$-bit output. It adds its two inputs in binary. For example, if $i = 3$, $j = 5$, and the inputs are 101 and 10111 respectively, the output is 011100. It is always possible to construct an (i, j)-adder using exactly $\max(i, j)$ 3-tallies. For this reason the 3-tally is often called a *full adder*.

a) Using full adders and (i, j)-adders as primitive elements, show how to build an efficient n-tally. Do not suppose that n has any special form.

b) Give the recurrence, including the initial conditions, for the number of 3-tallies needed to build your n-tally. Do not forget to count the 3-tallies that are part of any (i, j)-adders you might have used.

c) Using the Θ notation, give the simplest expression you can for the number of 3-tallies needed in the construction of your n-tally. Justify your answer.

Problem 6 (8 credits)

You are to organize a tournament involving n competitors. Each competitor must play exactly once against each of his opponents. Moreover, each competitor must play exactly one match every day, with the possible exception of a single day when he does not play at all.

a) If n is a power of 2, give an algorithm to construct a timetable allowing the tournament to be finished in $n - 1$ days.

b) For any integer $n > 1$ give an algorithm to construct a timetable allowing the tournament to be finished in $n - 1$ days if n is even, or in n days if n is odd. For example, the following are possible timetables for tournaments involving five and six players. (If the entry for day i and player j in the table is k, this means that player j plays against player k on day i.)

$n = 5$:

Player / Day	1	2	3	4	5
1	2	1	-	5	4
2	3	5	1	-	2
3	4	3	2	1	-
4	5	-	4	3	1
5	-	4	5	2	3

$n = 6$:

Player / Day	1	2	3	4	5	6
1	2	1	6	5	4	3
2	3	5	1	6	2	4
3	4	3	2	1	6	5
4	5	6	4	3	1	2
5	6	4	5	2	3	1