

Data Structures and Algorithms

Exam

Prof. Dr. M. Böhlen

September 22, 2006

08:30 - 10:30

The exam consists of 4 exercises. For each exercise you can get 20 points. It is important that you argue for your answers and that you present your solutions in a readable form. Fully specify all algorithms that you use as part of your solutions. As auxiliary material you may use 1 A4 sheet with notes.

1 Linked Lists

Assume a linked list of integers. The corresponding definition in C is as follows:

```
struct node {
    int key;
    node *next;
}

struct node *head;
```

The task is to develop a sorting algorithm for linked lists that is based on the idea of the insertion sort algorithm for arrays. The sorting shall be done by changing pointers and without copying any elements.

1. Illustrate the sorting on the example list shown in Figure 1.



Figure 1: Linked List

2. Propose an insertion sort based algorithm for a linked list.
3. Determine and explain the asymptotic complexity of your solution. Compare it with the algorithmic complexity of the insertion sort algorithm for arrays.

2 Treap

A treap is a binary search tree that orders nodes by adding a priority attribute to a node, as well as a key. The nodes are ordered so that the keys form a binary search tree and the priorities obey the min heap order property.

Each node u of a treap satisfies the following properties:

- If v is a left child of u , then $v.\text{key} < u.\text{key}$;
- If v is a right child of u , then $v.\text{key} > u.\text{key}$;
- If v is a child of u , then $v.\text{priority} > u.\text{priority}$.

Note that a treap does not have to be a nearly complete binary tree.

Figure 2 shows an example of a treap.

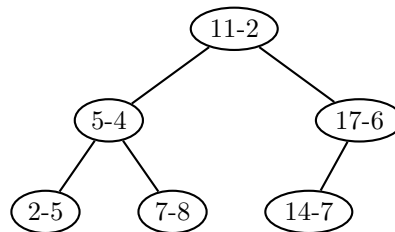


Figure 2: Treap

Assume a treap as follows:

```
struct TreapNode {
    int key;
    int priority;
    TreapNode *lChild;
    TreapNode *rChild;
}

struct TreapNode *root;
```

Inserting a node into a treap works as follows: Initially the insertion proceeds in a manner identical to the binary search tree insertion. After having inserted the node, basic tree rotations as, e.g., used by red-black trees are used to restore the heap property.

1. Insert key-priority pairs (3-3) and (18-1) into the treap shown in Figure 2. Draw the tree after each individual step.
2. Propose an algorithm for `heapInsert`, which inserts a node with a given key and priority into a treap.
3. Determine and explain the asymptotic complexity of your algorithm.

3 Longest Common Subsequence

Consider text strings $S_1 = \text{"MEMOISATION"}$ and $S_2 = \text{"RECURSION"}$.

1. Fill in Table 1 for the longest common subsequence algorithm.
2. Determine the longest common subsequence (LCS) of S_1 and S_2 .
3. Give a memoized version of LCS-Length.
4. Estimate the asymptotical complexity for your algorithm.

	j	0	1	2	3	4	5	6	7	8	9	10	11
i		S_1	M	E	M	O	I	S	A	T	I	O	N
0	S_2												
1	R												
2	E												
3	C												
4	U												
5	R												
6	S												
7	I												
8	O												
9	N												

Table 1: Longest Common Subsequence

4 Depth-First Search, Topological Sorting

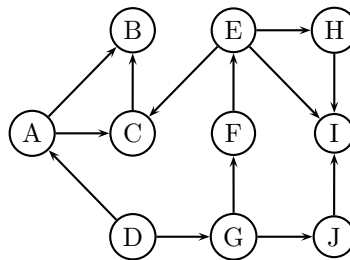


Figure 3: Graph

1. For each vertex of a graph in Figure 3 show the visiting start and end times of a DFS search that starts at vertex A. In cases of multiple options choose nodes in alphabetical order.
2. List the vertices of the graph in Figure 3 in a topological order.
3. Write topological sorting implementation.
4. Evaluate the complexity of a given algorithm.