

IADS F2005 : Major Week Assignment 2

Episode 10-11, April, 2005

(built on the Alstrup–Rauhe Heritage)

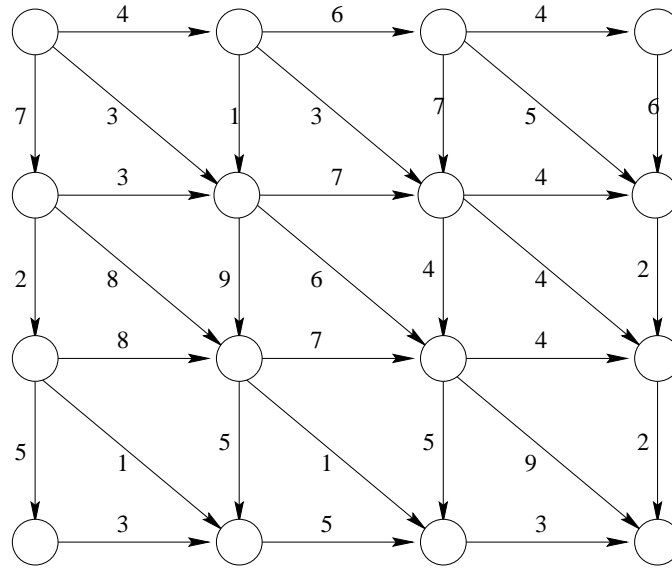
This set consists of 10 exercises in total. The exercise can either be handed in individually or in groups of at most two persons on or before **April 27, 2005, 13.00**. Groups should hand in exercises that sum to at least 120 points. For hand-ins by single individuals, the requirement is 100 points. There should be a maximum of 6 exercises in the hand-in.

- The front page of the hand in must clearly state who the author(s) are.
- Solutions should be described as briefly as possible.
- The algorithms that you design need not be described in pseudocode.
- If an exercise asks for the “construction of a heap that supports the operations”, it should be described how to construct a data structure that supports the operations. It does not matter whether it is similar to the heap in CLRS.
- It is a requirement that all exercises contain arguments for correctness and worst-case running time of the algorithms that you design.
- In some exercises we provide examples of the operations that should be supported. These only serve the purpose of being examples. For instance, if you are asked to support the operation $+$, and the example is $2 + 3 = 5$, then a solution which is only able to add 2 and 3 is not a good solution.
- The bounds you give on running time of algorithms should be as tight as possible. If you give the bound $O(2^n)$ for an $O(\log n)$ algorithm, only a low percentage of points will be awarded.

Exercises

1. **(24 points)** Let L be a singly-linked list, and P a pointer initially pointing to the first element in L . L ends with NIL. We consider the following two operations:
 - $\text{go}(1)$: Moves P to next element in the list. Takes $O(1)$ time.
 - $\text{go}(k)$: Moves P k steps ahead in the list. Takes $O(\min(k, s) + 1)$ time, where s is the number of elements between P and the end of the list.

If P is at the end of the list L , i.e. $P = \text{NIL}$, it remains there after any of the above operations. Suppose L contains n elements and we execute d many $\text{go}(\cdot)$ operations. How much time do these operations take in total expressed in terms of n and d ? The answer is among $O(n)$, $O(d)$, $O(n + d)$, and $O(n \cdot d)$.
2. **(34 points)** Given a list of n elements, show how to find the maximal and minimal element using at most $(3/2)n + O(1)$ comparisons.
3. **(35 points)** Consider the 4×4 network as in the figure below:



In general, an $n \times n$ network has n times n nodes. We call the node in the i th row (from top) and the j th column from the left $v(i, j)$. Thus the upper leftmost node is $v(1, 1)$ and the lower rightmost node is $v(n, n)$. Each node is connected horizontally, vertically and diagonally to its neighbours as in the drawing. In other words, for $i < n$ and $j < n$ there are three edges from the node $v(i, j)$ to $v(i + 1, j)$, $v(i, j + 1)$, and $v(i + 1, j + 1)$. Nodes in the bottom row only have edges to the right, nodes in the rightmost column only have edges down, and node $v(n, n)$ have no outgoing edges.

To each edge there is an associated positive value which corresponds to the distance to be passed when crossing this edge. It is only possible to follow the edges according to their orientation. The exercise is to describe an algorithm that computes the shortest distance from $v(1, 1)$ to $v(n, n)$. For instance, the shortest distance for the network in the drawing from $v(1, 1)$ to $v(4, 4)$ is 13.

a) Write a *recursive* algorithm that computes the shortest distance between $v(1, 1)$ and $v(n, n)$. Then use dynamic programming. Assume the input is given as three 2-dimensional arrays D , H and V whose (i, j) th entries are the values associated to the diagonal, horizontal, and vertical edges respectively leaving the node $v(i, j)$.

4. (55 points) Let $X[1 \dots n]$ be an array of n positive integers. An interval partition of X into $p \leq n$ subarrays consists of a list of $p - 1$ indices $0 < q_1 < q_2 < \dots < q_{p-1} < n$. The first subarray is $X[1..q_1]$, the second $X[q_1 + 1 \dots q_2]$ and so on with the last subarray being $X[q_{p-1} + 1 \dots n]$. A *subsum* is the sum of all elements in a subarray. For instance, the subsum for the j th subarray is $X[q_{j-1} + 1] + X[q_{j-1} + 2] + \dots + X[q_j]$.

A *good* interval partition in p subarrays is an interval partition of X where the *largest* of the p subsums is as small as possible. The p -value of X is the largest subsum of a good interval partition of X into p subarrays. For instance the 2-value of $[2, 4, 2, 7]$ is 8 corresponding to the partition $[2, 4, 2]$ and $[7]$.

Give a dynamic programming algorithm to compute the p -value of X such that the running time is better than $O(n^p)$ for $p \geq 4$.

[Hint: Make a recursive formula for the value to be computed. The following is an attempt by Alstrup & Rauhe, but feel free to come up with your own as long as you argue why it works.

$$value(k, p, d) = \begin{cases} d + X[1], & \text{if } p = 1 \text{ or } k = 1 \\ \infty & \text{if } p > 1 \text{ and } k = 1 \\ value(k - 1, p, d + X[k]), & \text{if } p = 1 \text{ and } k > 1 \\ \min\{ \max\{ value(k - 1, p - 1, 0), d + X[k] \}, & \\ \quad value(k - 1, p, d + X[k]) \} & \text{if } p > 1 \text{ and } k > 1 \end{cases} .$$

5. **(40 points)** CLRS 21–2 pp. 519–520.
6. **(25 points)** (a) Picture the AVL trees resulting from successively inserting the keys 1, 2, 3, ..., 14, 15 (in this order) into an initially empty AVL tree, rebalancing after each insertion. Pay particular attention to the trees resulting after the insertion+rebalancing of the first 3, 7, 15 keys.
- (b) Formulate a conjecture about the form of the AVL tree resulting after insertion into an initially empty AVL tree of the keys 1, 2, 3, ..., $2^n - 2$, $2^n - 1$ (in this order).
- (c) Prove the conjecture from (b) by induction on n .

[You only need to write up the solutions for (b) and (c).]

7. **(30 points)** Develop a Java-class for the UNION-FIND data structure from CLRS. That is, make a class with the following three methods:
- **INIT(n):** Initialize n sets $\{1\}, \{2\}, \{3\}, \dots, \{n\}$.
 - **UNION(i, j):** Unites the two sets containing the elements i and j . If both i and j are in the same set nothing changes.
 - **FIND(i):** Returns a representative for the set containing element i . For instance, $\text{find}(i) == \text{find}(j)$ is true if and only if i and j are in the same set.
8. **(15 points)** The following exercise is about binary search trees. These are introduced in CLRS, Chap. 12. CLRS also describes how the following two operations can be implemented in time $O(h)$, where h is the height of the search tree involved:
- **SEARCH(x, k):** Determines if there is a node with key k in the search tree with root x .
 - **INSERT(x, k):** Inserts a new node with key value k into the search tree with root x so that the search tree property is preserved.

Implement these two operations in a Java class. Both operations should run in time $O(h)$.

9. **(15 points)** Extend the Java-implementation from exercise 8 with the following two additional operations:

- **DELETE(x, k):** Removes all nodes with key value k from the tree with root x so that the tree remains a binary search tree.
- **PRINT(x):** Prints the key values of all the elements in the tree with root x in sorted order.

PRINT must run in linear time.

10. **(40 points)** Let G be a directed graph with m edges and n nodes, where the edge weights are positive integers. We consider the problem of computing a shortest path from node s to all other nodes in G . Suppose that the distance from s to any other node is bounded by W . Construct an algorithm which solves this problem in time $O(m + n + W)$.
11. **(30 points)** Let T be a binary tree with root x , n nodes, and height h . Let **COUNT(x)** which, given the root x , returns the number of nodes in the tree. Assume that **COUNT(x)** uses time linear in the number of nodes returned. The procedure **MULTICOUNT(x)** calls **COUNT(x)** and thereafter recursively calls **MULTICOUNT(y)** for all of x 's children y . (If x is a leaf, there are no recursive calls.) Estimate the complexity of **MULTICOUNT(x)** in terms of n and h .
12. **(20 points)** Let the *sum depth* of a binary tree T denote the sum of the depths of all nodes in T . Design a recursive procedure **SUMDEPTH(x)**, which for a tree T with n nodes and root x returns the sum depth of T in time $O(n)$.
13. **(20 points)** Let A and B be two rooted binary trees with roots a and b respectively. The two trees are said to be *isomorphic* if (1) both a and b are leaves or (2) a 's left subtree is isomorphic with b 's left subtree and a 's right subtree is isomorphic with b 's right subtree. Construct a recursive procedure which decides whether two binary trees are isomorphic.