

Introduction to algorithms and data structures.

IT University of Copenhagen.

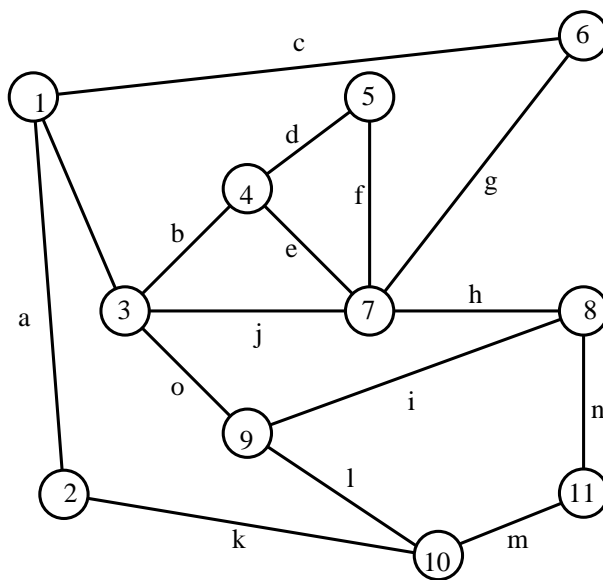
January 7th, 2000

This exam consists of 3 exercises with further subexercises, which sums to 100 points. The subexercises in each exercise has the same weight. You have 4 hours to complete the exam. Remember to number the pages and write your study-number and name on all pages. The exam consists of 9 numbered pages.

CLR refers to “Introduction to Algorithms” by Cormen, Leiserson and Rivest, 18. press, 1997.

Exercise 1 (30 point) Graphs and graph algorithms.

The following exercise is about the construction of the minimum spanning tree for a connected, unoriented graph. Look at the weighted graph below. Each node are named with a number from 1 to 11, and each edge are named with a letter from a to o . In the table beneath the graph, the weight of each edge are specified. For example the edge j has weight 13. As we know Kruskal's algorithm (CLR 24.2) can be used to find the minimum spanning tree for a weighted graph.



Edge	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
Weight	3	15	12	8	2	5	6	14	4	13	7	9	10	1	11

a) Give the edges, which are in a minimum spanning tree for the graph above, and list the order for the tree edges found by the algorithm of Kruskal.

Answer a) $h, e, a, i, f, g, k, l, o, c$.

In the following we consider a connected and unoriented graph $G = (V, E)$. We let $n = |V|$ denote the number of nodes in G . We call a graph for a *thin graph*, if it only has a linear number of edges. That is $|E|$ is $O(n)$.

In the following exercises b), c) and d) it is assumed that G are a connected unoriented thin graph, which are represented by the **adjacency-list** representation (CLR 23.1).

b) Give the running time to find a minimum spanning tree for the thin graph G using Kruskal's algorithm.

Answer b) $O(n \lg n)$.

c) Consider k edges $e_1, e_2, \dots, e_k \in E$ in the thin and connected graph G . Describe an algorithm, which decides, whether there are a spanning tree, which contains the edges e_1, \dots, e_k , and in case this is true, finds the minimum spanning tree with these edges e_1, \dots, e_k . Provide the running time of the algorithm.

Answer c) $E' = e_1, e_2, \dots, e_k \in E$ are contained in a spanning tree if and only if the graph $G' = (V, E')$ doesn't contain cycles. This can be investigated by using a depth first search of G' in $O(n)$ time.

d) Assume that all edge-weights of G has a value between 1 and $3n$. Describe an algorithm, which finds a minimum spanning tree for the thin graph G in time $O(n \lg \lg n)$. It can be used that any sequence of $O(n)$ operations (MAKE-SET, UNION and FIND-SET) for the **disjoint-set** data structure can be executed in time $O(n \lg \lg n)$.

Answer d) We use COUNTING-SORT for sorting in the Kruskal algorithm. The total time is thus $O(n \lg \lg n)$.

Exercise 2 (40 point) Lists and searching

This exercise is mostly about lists, searching and divide techniques. The exercise takes its offspring in linked lists as described in CLR 11.2.

Let L be a singly linked list. Let $head[L]$ denote the pointer to the first element in the list. If $head[L] = \text{NIL}$, the list is empty. Each element has a pointer to the next element in the list. If x is the last element in the list, $next[x] = \text{NIL}$. Each element also has a key field, which are a number. It is assumed that the key fields are sorted in increasing order. That is for two neighbor-elements x and $y = next[x]$ it holds that $[x] \leq key[y]$.

Consider the following procedure TEST:

TEST(L, K)

1. $x \leftarrow head[L]$
2. **while** $x \neq \text{NIL}$ and $key[x] < K$
3. **do** $x \leftarrow next[x]$
4. **if** $x = \text{NIL}$
5. **then return** FALSE
6. **if** $key[x] = K$
7. **then return** TRUE
8. **else return** FALSE

a) Describe what the procedure TEST does, in case L is sorted, and denote the time complexity for the procedure TEST.

Answer a) TEST investigates whether there exists an element with key K in L .

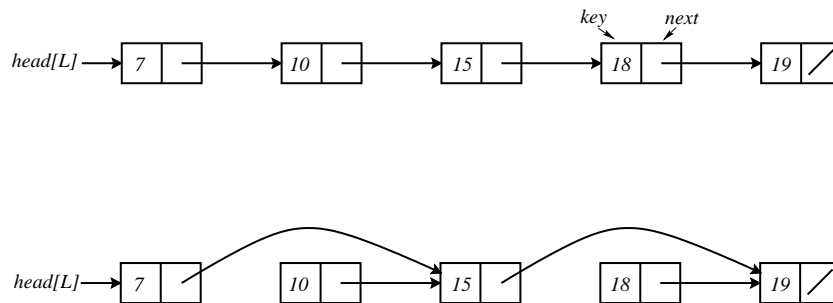
Consider the following two procedures LENGTH and SWAP:

LENGTH(L)

1. $x \leftarrow head[L]$
2. $k \leftarrow 0$
3. **while** $x \neq \text{NIL}$
4. **do** $k \leftarrow k + 1$
5. $x \leftarrow next[x]$
6. **return** k

SWAP(L)

1. \triangleright It is assumed that the procedure LENGTH(L) returns a positive, unequal integer.
2. $x \leftarrow head[L]$
3. **while** $next[x] \neq \text{NIL}$
4. **do** $next[x] \leftarrow next[next[x]]$
5. $x \leftarrow next[x]$



In the figure an example of a list with 5 elements before and after a call to SWAP are seen.

In the remaining part of the text of the exercise, we only consider lists with n elements, where n can be written as $2^k + 1$ for an integer $k \geq 0$. That is the numbers 2, 3, 5, 9, 17, 33, ...

b) Let L be a list with $n = 17 = 2^4 + 1$ elements. First the procedure SWAP(L) are called, next the procedure LENGTH(L). What does the procedure LENGTH(L) return?

Answer b) LENGTH returns $n + 1/2 = 18/2 = 9$.

c) Let L be a list with $n = 2^k + 1$ elements for a $k \geq 1$. How many times must the procedure SWAP(L) be called, before a call to the procedure LENGTH(L) returns 2?

Answer c) k .

Let L be a singly linked list with key and $next$ fields as described above. Each element in the list we further assigns a table, $jump$ of pointers. For a list with $n = 2^k + 1$ elements, the jump table contains k pointers, $jump[1 \dots k]$. For each element x the $jump[x][j]$ pointer are the j 'th entry in x 's jump table.

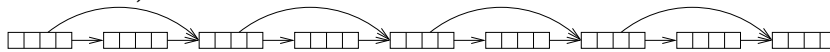
Consider the following procedure JUMP1:

JUMP1(L)

1. \triangleright It is assumed that the procedure LENGTH(L) returns a positive unequal integer.
2. $x \leftarrow head[L]$
3. **while** $next[x] \neq NIL$
4. **do** $jump[x][1] \leftarrow next[next[x]]$
5. $x \leftarrow jump[x][1]$

d) Given is a list L with $n = 9 = 2^3 + 1$ elements. Assume that $jump[x][i]$ for each x and i are initialized to NIL. Draw the datastructure after a call to the procedure JUMP(L), where pointer that are NIL, can be omitted.

Answer d)



Consider the following two procedures JUMP and JUMPTABLE:

JUMP(L, i)

1. $x \leftarrow head[L]$
2. **while** $jump[x][i - 1] \neq NIL$
3. **do** $y \leftarrow jump[x][i - 1]$
4. $jump[x][i] \leftarrow jump[y][i - 1]$
5. $x \leftarrow jump[y][i - 1]$

JUMPTABLE(L)

1. $n \leftarrow LENGTH(L)$
2. Below it is assumed that k is a number such that $n = 2^k + 1$
3. **if** $k > 0$
4. **then** JUMP1(L)
5. **if** $k \geq 2$
6. **then for** $i \leftarrow 2$ **to** k
7. **do** JUMP(L, i)

e) Let L be a list with $n = 2^k + 1$ elements for $k \geq 0$. It is assumed that $jump[x][i]$ for each x and i are initialized to NIL.- Show that the time complexity for a call to the procedure $JUMPTABLE(L)$ is $O(n)$.

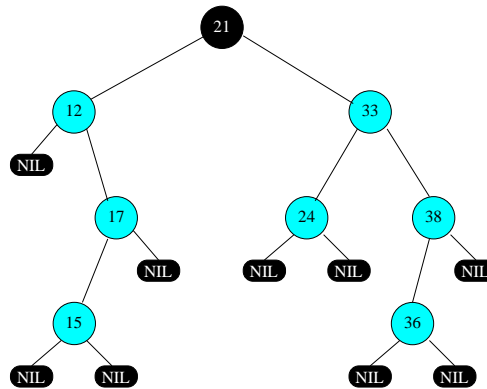
Answer e) Remark that the procedure doubles the length of the interval it jumps each time. The time complexity are thus $O(n/2+n/4+n/8\cdots) = O(n)$.

f) Let L be a list with $n = 2^k + 1$ elements for $k \geq 0$, and the *key* fields ordered in increasing order as described earlier. Assume that $jump[x][i]$ for each x and i are initialized to NIL, and that there is then done a call to the procedure $JUMPTABLE(L)$. Describe in words how one can construct a procedure $NEXT(L, z)$, which returns an element, with the smallest *key* value in the list, which are strictly larger than z . If such a number doesn't exist, the program must return -1 . The described procedure $NEXT$ must have time complexity $O(k)$.

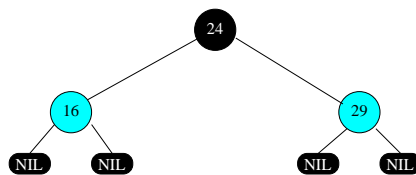
Answer f) Using jump pointers a binary search can be done. The time is $O(\log n) = O(k)$.

Exercise 3 (30 point) Red-black-trees

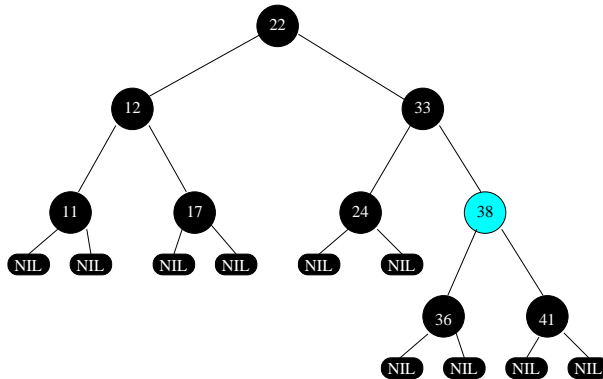
This exercise is about red black trees. The figures in this exercise follows the conventions in CLR. In the your solution the colors of the nodes must be clearly stated. For instance by writing *r* for red and *b* for black beside the nodes.



a) The tree T above doesn't conform to the red-black properties. Show how T can be changed and colored without changing the root in such a way that the tree conforms to the properties. The modified tree must contain the same values as T .



b) Insert the value 8 in the red-black-tree in the figure with three internal nodes above. Insert thereafter the value 13 in the tree, in which 8 are inserted. The algorithm for insertion in red-black-trees described in CLR must be used. Describe the result by illustrating how the algorithm works.



c) Delete the value 22 in the red-black-tree in the figure above. The algorithm for deletion in red-black-trees in CLR must be used. Describe the result by illustrating how the algorithm works.

d) Describe an algorithm $\text{COUNT-BLACK}(r, v)$, where r is the root of a red-black-tree T , and v is a value. COUNT-BLACK returns the number of black nodes in T , where the values in the nodes are larger than or equal to v . The algorithm must have the time complexity $O(k + \lg n)$, where n is the number of nodes in T and k is the number of nodes in T with a value greater than or equal to v .

Answer d) $\text{COUNT-BLACK}(r, v)$ can be implemented by first searching for r in T . Then we find all nodes w , where $\text{key}[w] \geq r$ with a call to TREE-SUCCESSOR . During this, we count the number of black nodes. The search takes $O(\lg n)$ and the total time for TREE-SUCCESSOR is $O(k)$.

e) Given a red-black-tree T , where the root has black-height h . What is the fewest possible number of internal black nodes, which T at least must contain? What is the largest possible number of internal black nodes that T can contain? Remember that all internal nodes has two children.

Answer e) In the minimum case T must contain $2^{h-1} + 1$ internal black nodes. This is in the case of a complete binary tree of height h . To maximize the number of black nodes we can interchangeably color every other level of nodes black starting with the leaves. The maximal number of internal black nodes is thus $\sum_{i=0}^{h/2} 2^{2i}$ if h are equal and $1 + \sum_{i=0}^{h-1/2} 2^{2i+1}$ if h are unequal.

Answer a,b,c) Leafs are omitted.

