Assignment 8 – Selected Model Answers

EXERCISE 1. As leaves in a B-tree do not have children they require less space. Alternatively, one might want to use this space to store more keys in leaves.

How must the *insert* and *delete* operations for B-trees be updated to handle such a variation?

SOLUTION. For a non-leaf B-tree node we can use an array of size $3m$ to store the number of keys, a parent pointer, $s+1$ children pointer, and s pairs comprising a key and a data pointer. We then requir $\lceil m/2 \rceil \leq s+1 \leq m$. If there are no children pointer, the filling citerion for leaves can be changed to $\lceil \frac{3m}{4} \rceil \leq s+1 \leq \lfloor \frac{3m}{2} \rfloor$.

Then for an insertion the only change is that the criterion for overflow of a leaf is changed. If we have $s+1 \geq \lfloor \frac{3m}{2} \rfloor \geq s$, and the new keys are $k_1, \ldots k_{s+1}$ (with associated data pointers $p_1, \ldots, p_{s+1}$), we can split the leaf using $i = \lceil \frac{3m}{4} \rceil$. One leaf contains the counter $i-1$, the parent pointer, the keys $k_1, \ldots, k_{i-1}$, and the data pointers $p_1, \ldots, p_{i-1}$, the other one contains the counter $s-i+2$, the parent pointer, the keys $k_{i+1}, \ldots, k_{s+1}$, and the data pointers $p_{i+1}, \ldots, p_{s+1}$, while (k_i, p_i) are inserted into the parent node. For both new leaves the new filling criterion is satisfied.

Analogously, for a deletion the only change is the criterion for the underflow. The merging of two leaves is done is the same way as without he optimisation, and the new filling criterion will be satisfied.

EXERCISE 2. Assume that the nodes of a B-tree or a B^+-tree are stored in external storage.

(i) How many blocks have to be moved from external storage into the buffer during an *insert*, *find* or *delete* operation on a B-tree or a B^+-tree?

(ii) How many blocks have to be moved from external storage into the buffer during a *range* operation on a B-tree or a B^+-tree?

(iii) Discuss which order of B-trees or B^+-trees would be ideal for minimising the number of paging operations (i.e. moving a block from external storage into the buffer), assuming that the number of blocks that can be kept simultaneously in the buffer is bounded by k.

SOLUTION.

(i) According to the estimation shown in the lectures the height h of a B-tree with n keys is (up to some small additive constant) between $\log_m \frac{n}{2}$ and $\log_{m/2} \frac{n}{2}$, where m is the order of the tree. Thus, at most $h+1$ blocks need to be moved between external storage and main memory (one additional block for the data to be inserted or deleted).

For a B+-tree always $h+2$ blocks need to be moved, as always the main data including an overflow block needs to be explored. However, as there are no data pointers in non-leaf nodes, m can be larger and so h will become smaller.

(ii) There is a node N such that all keys in the given range are in the subtree rooted at this node. If the height of this subtree is h_2, the there are at most $m_2^h - 1$ tree nodes that need to be explored, plus h_1 tree nodes between the root and N. Therefore, in a B-tree the number of blocks that need to be moved to main memory is $\leq (h_1 + m^{h_2} - 1)(m + 1)$, as not only the node block, but also the corresponding data blocks need to be moved to main memory. For a B+-tree this reduces to $(h_1 + m^{h_2} - 1 + 2m^{h_2}$, as main data blocks are only associated with leaves of the tree.

(iii) According to (i) and (ii) the number of paging operations is minimised, if blocks transferred between secondary storage and main memory contain as many keys as possible. The maximum number of keys in a B-tree or B+-tree node is bounded by $m-1$, so it will be ideal to maximise m such that a fully filled tree node requires the space of a block.

EXERCISE 3. Specify an algorithm using depth-first search that determines the connected components of an undirected graph.

SOLUTION. We use a start vertex $s \in V$, single fifo queue Q initialised to $Q = [s]$, a unary function symbol *component*, initially undefined, to capture a distinct integer value for the different connected components, a counter *count* initialised to 1, and a subset V' with initial value $V' = V$. Then we iterate the following rule:

IF $\neg$ *isempty*(Q)
THEN $v :=$ *popfront*(Q)
component$(v) :=$ *count*
$V' := V - \{v\}$
FORALL w **WITH** $(v, w) \in E \wedge w \in V'$ **DO**
pushback(Q, w)
IF *isempty*$(Q) \wedge V' \neq \emptyset$
THEN **CHOOSE** v **WITH** $v \in V'$ **IN**
pushback(Q, v)
count := *count* +1

In each iteration we have two possibilities:

1) The queue is not empty. Then we pop the front element v, set the label of its connected component to the current label given by the counter *count*, and push all direct neighbours w that have not yet been visited, i.e. $w \in V'$, onto the queue in some arbitrary order.
2) The queue is empty and there are still vertices for which the connected component is still undefined, i.e. $v \in V'$. We choose one of them and push them onto the queue, increment the counter *count*, as a new connected component will be built.

EXERCISE 4. Describe an implementation of breadth-first search on undirected graphs using a single FIFO queue containing those nodes, for which the outgoing edges still have to be scanned.

SOLUTION. We modify the given dfs algorithm using only a single queue Q. Instead of pushing a vertex of the next layer first onto a queue Q', which becomes Q when all vertices of the current layer have been deakt with, with push the vertices directly to the end of Q.

To keep track of layers we use a unary function symbol *layer*. Initially, we only have $layer(s) = 0$. When processing an edge $(v, w) \in E$, where w is pushed onto Q, we set $layer(w) := layer(v) + 1$. Formally, we iterate the following rule:

IF $\neg\ isempty(Q)$
THEN $v := popfront(Q)$
FORALL w **WITH** $(v, w) \in E \wedge layer(w) = \text{undef}$ **DO**
$pushback(Q, w)$
$layer(w) := layer(v) + 1$