B^{+}-*tree* Algorithms

Algorithm 1. Searching for a record with search-key value K, using a B^{+}-*tree*.

Begin
    $n \leftarrow$ block containing root node of B^{+}-*tree*;
    read block n;
    while (n is not a *leaf node* of the B^{+}-*tree*) do
        begin
            $q \leftarrow$ number of tree pointers in node n;
            if $K < n.K_1$ /* $n.K_i$ refers to the ith search-key value in node n */
              then $n \leftarrow n.P_1$ /* $n.P_i$ refers to the ith pointer in node n */
            else if $K \geq n.K_{q-1}$
                then $n \leftarrow n.P_q$
            else
              begin
                  search node n for an entry i such that $n.K_{i-1} \leq K < n.K_i$;
                  $n \leftarrow n.P_i$;
              end; /* ELSE */
            read block n;
        end; /* WHILE */
    search block n for entry K_i with $K = K_i$; /* search leaf node */
    if *found*, then read data file block with address P_i and retrieve record
    else record with search-key value K is *not* in the data file;
end. /* Algorithm 1 */

Algorithm 2. Inserting a record with search-key value K in a B^{+}-*tree* of order p.
/* A B^{+}-*tree* of order p contains at most $p - 1$ values and p pointers. */

Begin
  $n \leftarrow$ block containing root node of B^{+}-*tree*;
  read block n;
  set stack S to empty;
  while (n is not a *leaf node* of the B^{+}-*tree*) do
      begin
        push address of n on stack S; /* S holds parent nodes that are needed in case of split */
        $q \leftarrow$ number of tree pointers in node n;
        if $K < n.K_1$ /* $n.K_i$ refers to the jth search-key value in node n */
          then $n \leftarrow n.P_1$ /* $n.P_i$ refers to the jth tree pointer in node n */
        else if $K \geq n.K_{q-1}$
            then $n \leftarrow n.P_q$
        else
          begin
            search node n for an entry i such that $n.K_{i-1} \leq K < n.K_i$;
            $n \leftarrow n.P_i$
          end; /* ELSE */
        read block n
      end; /* WHILE */
  search block n for entry K_i with $K = K_i$; /* search leaf node n */
  if *found*
    then *return* /* record already in index file - no insertion is needed */
  else
    begin /* insert entry in B^{+}-*tree* to point to record */

```
create entry (P, K), where P points to file block containing new record;
if leaf node n is not full
  then insert entry (P, K) in correct position in leaf node n
else
  begin  /* leaf node n is full - split */
    copy n to temp;  /* temp is an oversize leaf node to hold extra entry */
    insert entry (P, K) in temp in correct position;  /* temp now holds p + 1 entries of the
                form (P_i, K_i) */
    new ← a new empty leaf node for the tree;
    j ← ⌈p/2⌉;
    n ← first j entries in temp (up to entry (P_j, K_j));
    n.P_next ← new;  /* P_next points to the next leaf node */
    new ← remaining entries in temp;
    K ← K_{j+1};
    /* Now we must move (K, new) and insert in parent internal node. */
    /* However, if parent is full, split may propagate */
    finished ← false;
    repeat
      if stack S is empty, then  /* no parent node */
        begin  /* new root node is created for the B+-tree */
          root ← a new empty internal node for the tree;
          root ← < n, K, new >;  /* set P_1 to n & P_2 to new */
          finished ← true;
        end
      else
        begin
          n ← pop stack S;
          if internal node n is not full, then
            begin  /* parent node not full - no split */
              insert (K, new) in correct position in internal node n;
              finished ← true
            end
          else
            begin  /* internal node n is full with p tree pointers - split */
              copy n to temp;  /* temp is an oversize internal node */
              insert (K, new) in temp in correct position; /* temp has p + 1 tree pointers */
              new ← a new empty internal node for the tree;
              j ← ⌈(p + 1)/2⌉;
              n ← entries up to tree pointer P_j in temp;
              /* n contains < P_1, K_1, P_2, K_2, ..., P_{j-1}, K_{j-1}, P_j > */
              new ← entries from tree pointer P_{j+1} in temp;
              /* new contains < P_{j+1}, K_{j+1}, ..., K_{p-1}, P_p, K_p, P_{p+1} > */
              K ← K_j;
              /* now we must move (K, new) and insert in parent internal node */
            end
        end
    until finished
  end; /* ELSE */
end; /* ELSE */
end.  /* Algorithm 2 */
```