```
Procedure: InvertedIndex()
Begin
    InvIndex ← CreateBP()
    askforFilePos()
    for file in the diretory do
        extract each word from file
        if word isn't a stop word then
            node ← FindBP(word, InvIndex)
        endif
        if word is already in InvIndex then
            increment the frequency and record the position for the word
        else
            InsertBP(word, node, InvIndex)
        endif
    end
End
```

```
Procedure: FindBP($word$: string, $T$: BplusTree)
Begin
    if $T$ has no children then
        isSameword($word$, $T$)
        return $T$
    endif

    $pos \leftarrow -1$
    for $i$ in range(0, $T \rightarrow size$) do // not contains $T \rightarrow size$
        if $word$ has less lexicographical order than $T \rightarrow data[i] \rightarrow word$
        then
            $pos \leftarrow i$
            break
        endif
    end
    if $pos = -1$ then
        $pos \leftarrow i$
    endif
    return FindBP($word$, $T \rightarrow children[pos]$)
End
```

```
Procedure: InsertBP(word: string, node: NodeBP, T: BplusTree)
Begin
    append word as the new data to the node
    sort the data in node
    $T \leftarrow$ SplitBP(node, T)
    return T
End
```

```
Procedure: SplitBP(word: string, node: NodeBP, T: BplusTree)
Begin
    if the node isn't full then
        return T
    endif

    if the node has no parent then
        create a new parent for the node
        let the parent be the root of T
    endif

    lnode $\leftarrow$ CreateBP()
    rnode $\leftarrow$ CreateBP()
    distribute node's data evenly to these two new nodes
    if node isn't a child node then
        also distribute node's children evenly to these two new nodes
    endif
```

```
17      adjust the relationship between *parent* **and** two new
        nodes(*lnode* **and** *rnode*)
18      sort $parent \rightarrow data$ **and** $parent \rightarrow children$
19      $T \leftarrow$ SplitBP($parent$, $T$)
20      **return** $T$
21  **End**
```