CSCE-629 Homework 7

Wang, Han-Yi
wanghy917@email.tamu.edu

October 2, 2020

Exercise 18.1-3

- There are 5 keys, and the minimum degree is 2
- The number of keys per node n, $1 \leq n \leq 3$
- The number of children per node $n + 1$, $2 \leq n + 1 \leq 4$

From the given conditions, we can get the B-trees shown in the below figure.

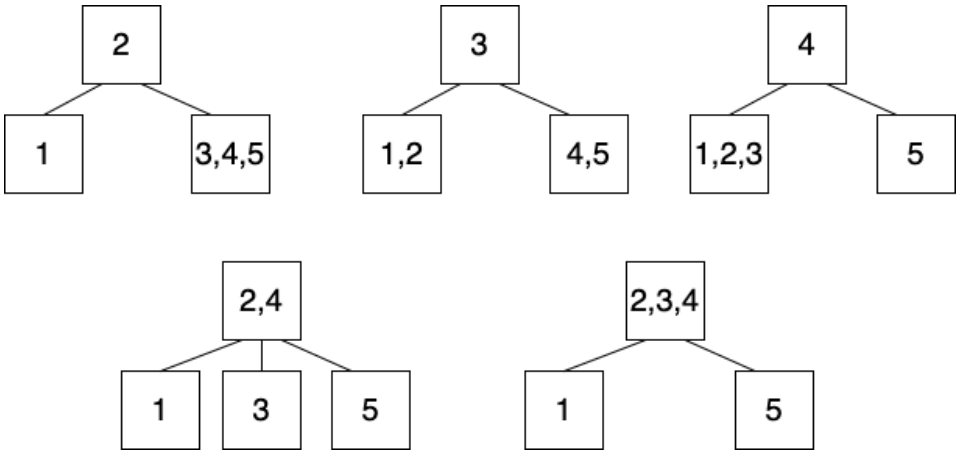

Figure 1: The B-trees of minimum degree 2 that represent {1, 2, 3, 4, 5}.

Exercise 18.2-1

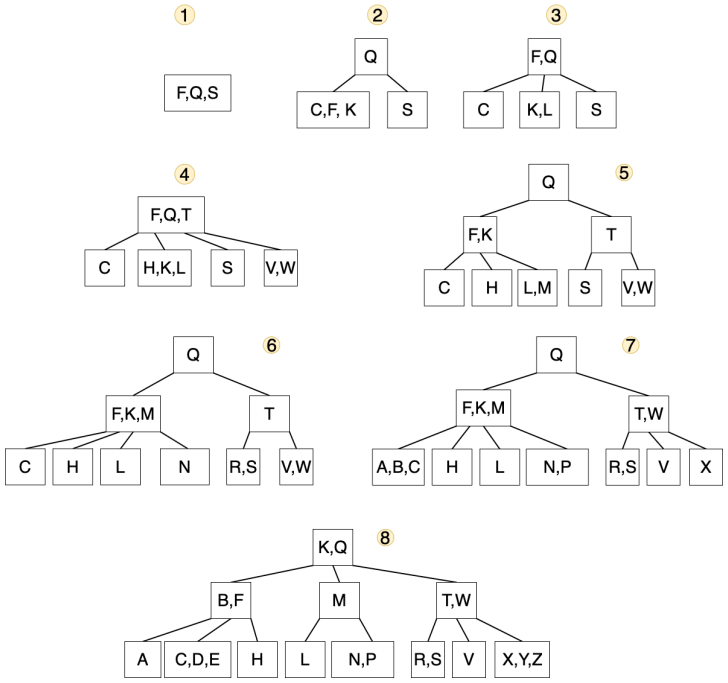

Figure 2: Results of inserting the keys.

Exercise 18.2-3

1. Find the minimum key

Since the $x.n$ keys stored in non-decreasing order, and $x.key_i$ separate the ranges of keys stored in each subtree. We could call B-TREE-MINIMUM($T.root$), and recursively call the algorithm until find the minimum value on the leaf.

Algorithm 1 B-TREE-MINIMUM(x)

```
1 if x == NIL
2     return NIL
3 if x.leaf
4     return x.key_1
5 return B-TREE-MINIMUM(x.c_1)
```

2. Find the Predecessor

To find the predecessor of key k, we use the below B-TREE-PREDECESSOR(x, k, w, i), where $x = w.c_j$. Initially, we call B-TREE-PREDECESSOR($T.root$, k, NIL, 0), we set w to NIL because the $T.root$ has no parent.

The idea is use the search algorithm to find the key, and we also keep tracking the parent node. When we found the key $k == x.key_i$, we find its predecessor by the following rules.

- If x is leaf, and $i > 1$, the predecessor is $x.key_{i-1}$
- If x is not leaf, the predecessor is the maximum key in the subtree rooted at $x.c_i$
- If x is leaf and $i == 1$, we need to trace the call stack, and find the first parent node w, where we find k through $w.c_j$, and $j > 1$, then $w.key_{j-1}$ is the predecessor
- If we cannot find such parent node in the previous rule, it means k is the smallest key, so we return NIL

Algorithm 2 B-TREE-PREDECESSOR(x, k, w, j)

```
1  i = 1
2  while  i ≤ x.n and k > x.key_i
3      i = i + 1
4  if  k == x.key_i
5      if  x.leaf
6          if  i > 1 return (x.key_{i-1})
7          else return NIL
8      else
9          return B-TREE-MAXIMUM(x.c_i)
10 else DISK-READ(x.c_i)
11     y = B-TREE-PREDECESSOR(x.c_i, k, x, i)
12     if  y == NIL and w! = NIL and j > 1
13         return w.key_{j-1}
       return y
```

Algorithm 3 B-TREE-MAXIMUM(x)

```
1  if  x == NIL
2      return NIL
3  if  x.leaf
4      return x.key_n
5  return B-TREE-MAXIMUM(x.c_{x.n+1})
```