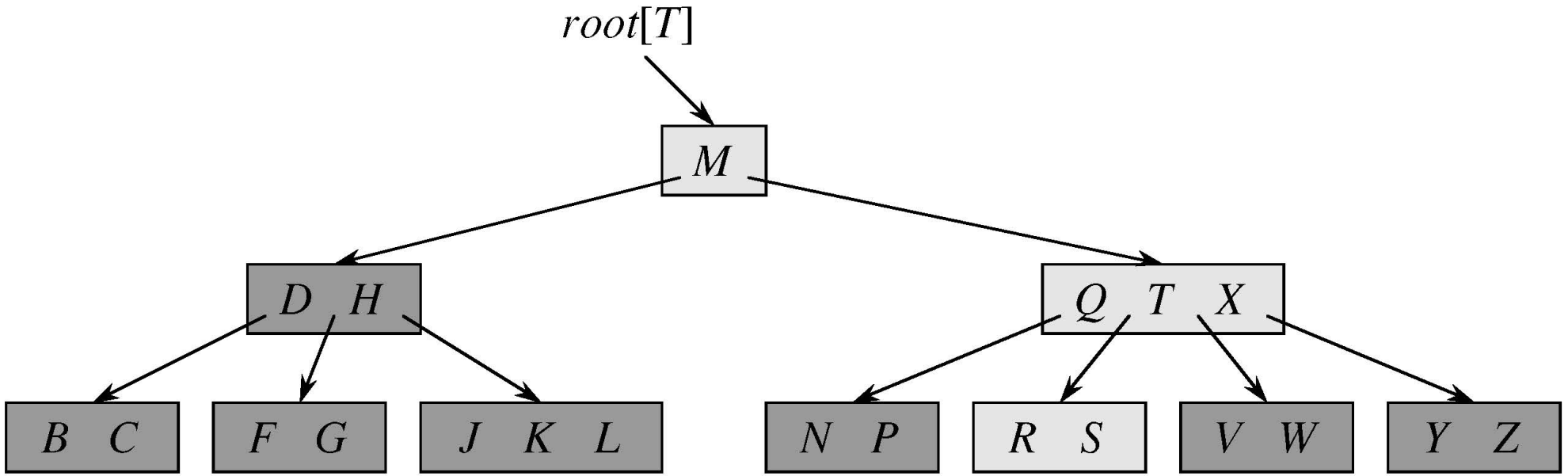
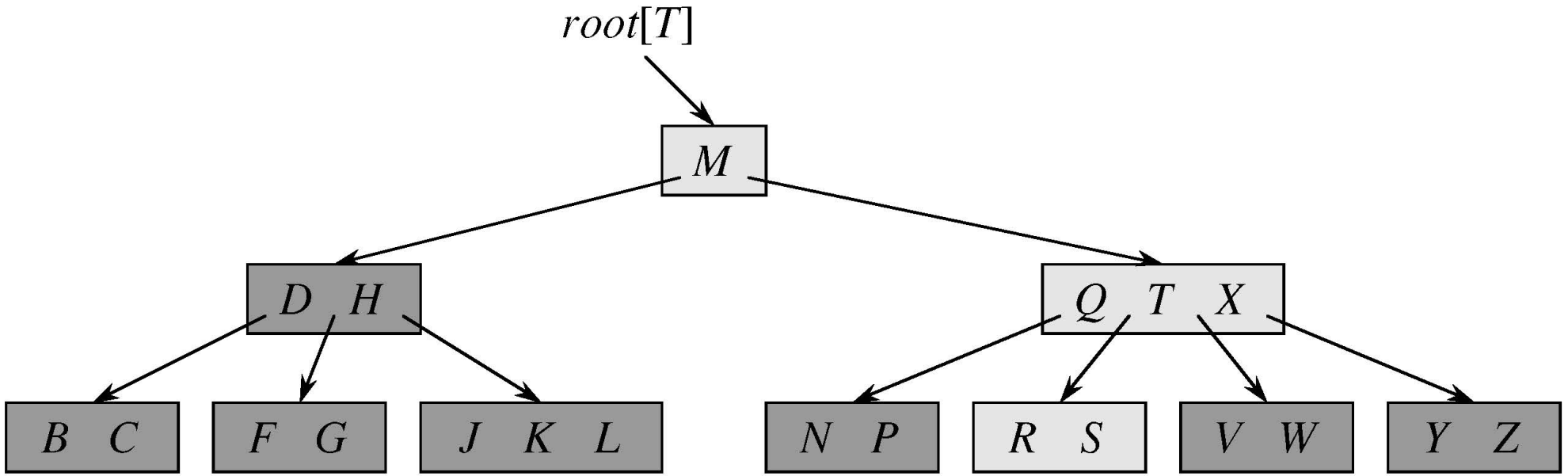
B-Bäume

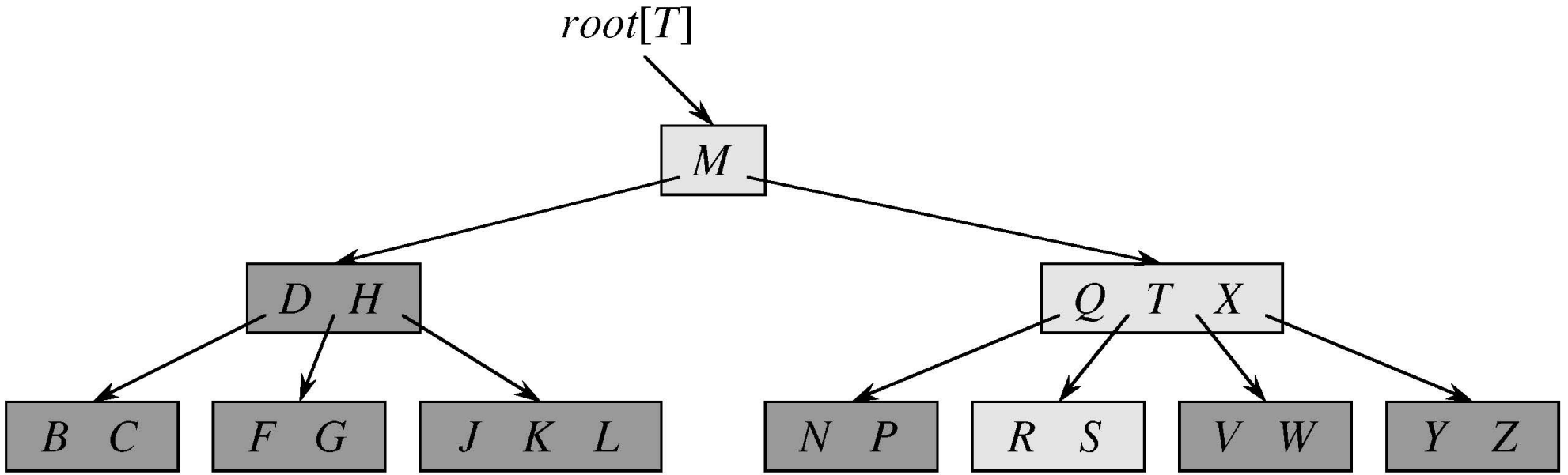
B-Baum



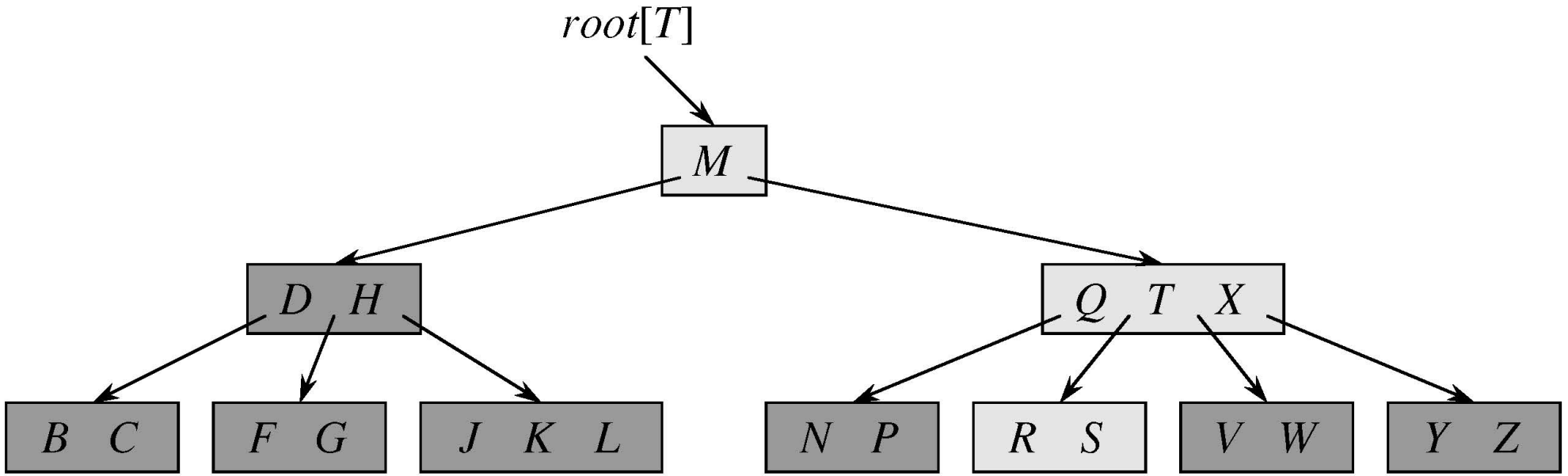
B-Baum
n[root] =



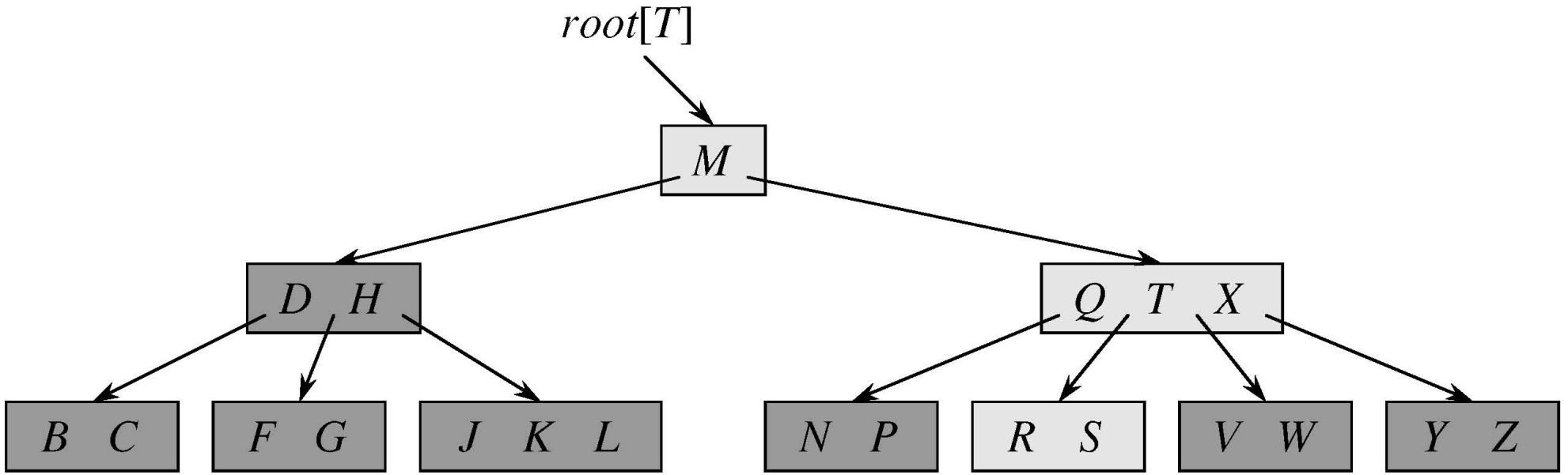
B-Baum
n[root] = 1



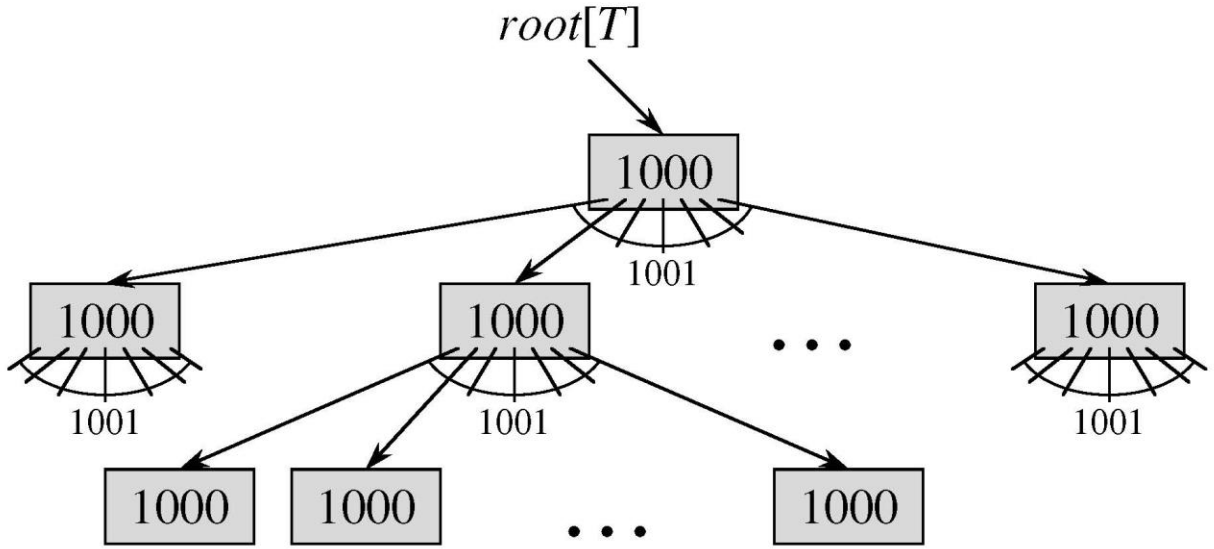
B-Baum
Minimalgrad t =



B-Baum
Minimalgrad t = 3



Großer B-Baum
Minimalgrad t =

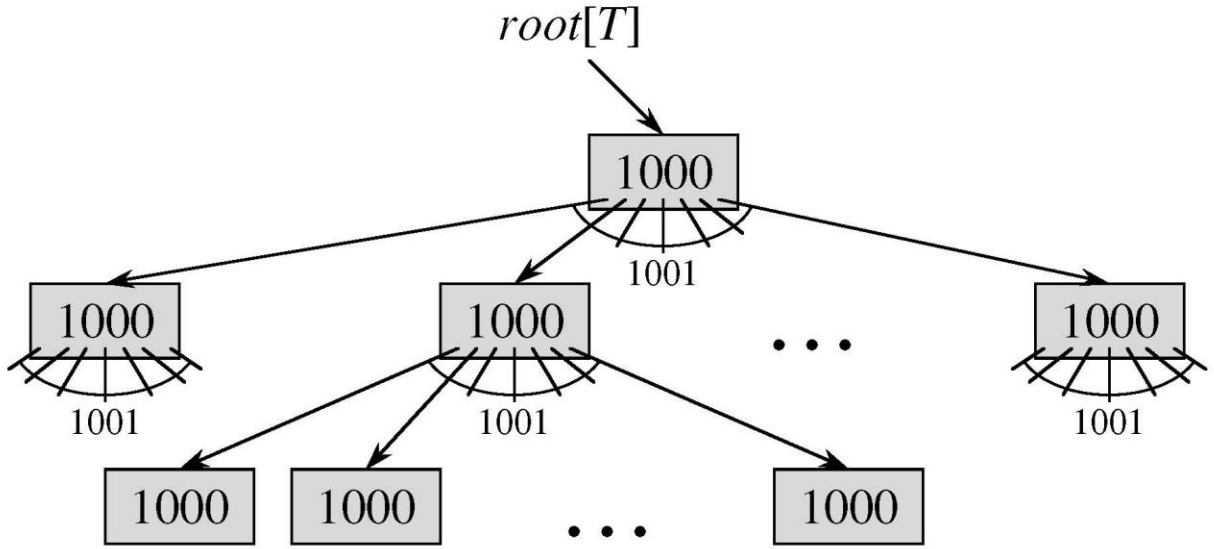


1 node,
1000 keys

1001 nodes,
1,001,000 keys

1,002,001 nodes,
1,002,001,000 keys

Großer B-Baum
Minimalgrad t = 1001

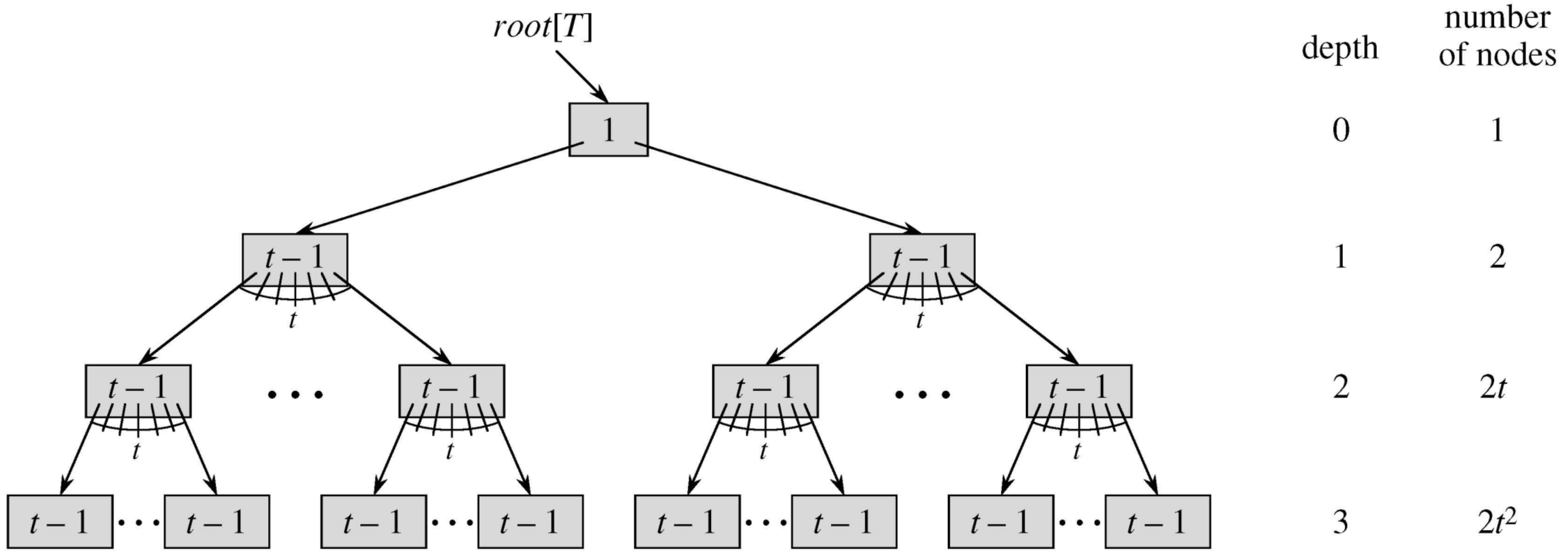


1 node,
1000 keys

1001 nodes,
1,001,000 keys

1,002,001 nodes,
1,002,001,000 keys

Beweis von Theorem 18.1

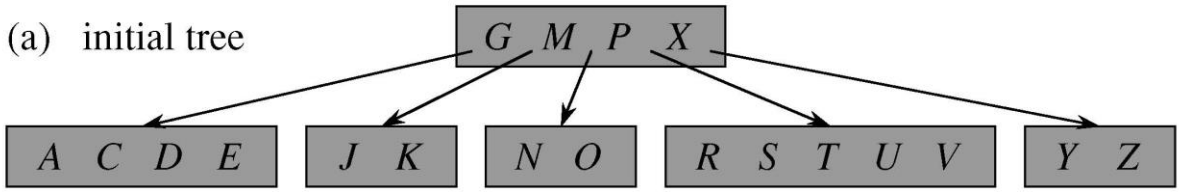


Suche in B-Bäumen

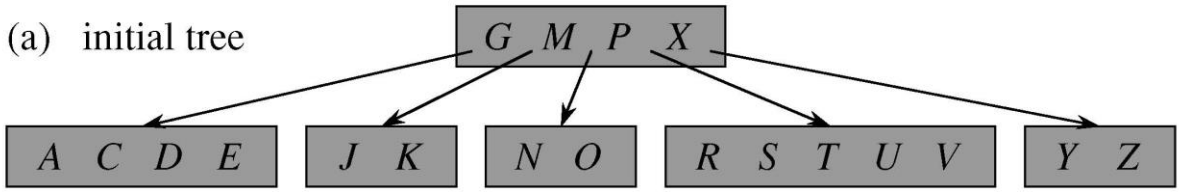
```
B-TREE-SEARCH(x, k)
1  i ← 1
2  while i ≤ n[x] and k > key_i[x]
3        do i ← i + 1
4  if i ≤ n[x] and k = key_i[x]
5     then return (x, i)
6  if leaf[x]
7     then return NIL
8     else DISK-READ(c_i[x])
9          return B-TREE-SEARCH(c_i[x], k)
```

B-Baum mit t = 3

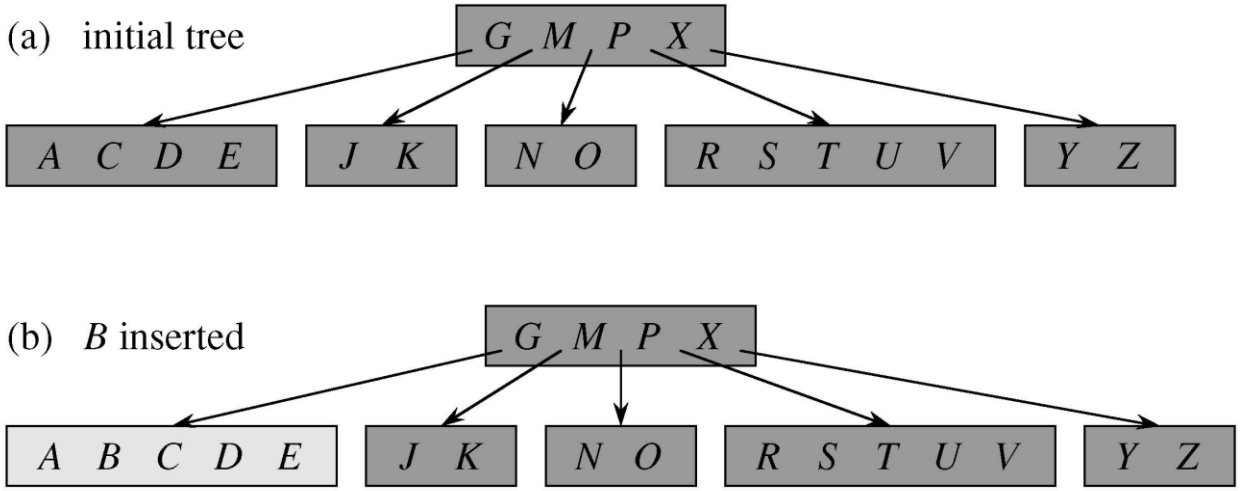
Innere Knoten enthalten zwischen 2 und 5 Schlüsseln



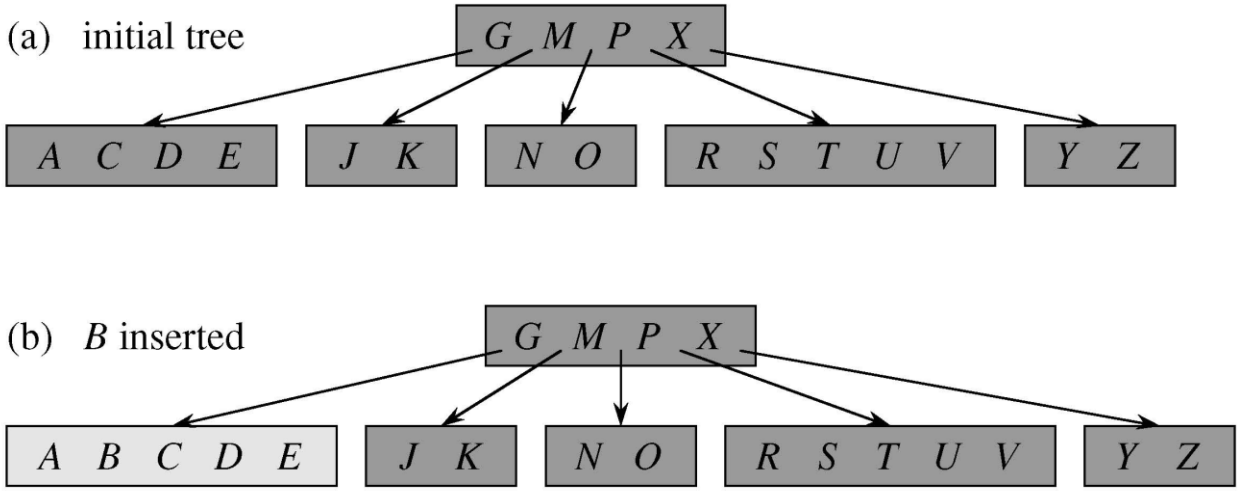
Einfügen von B?



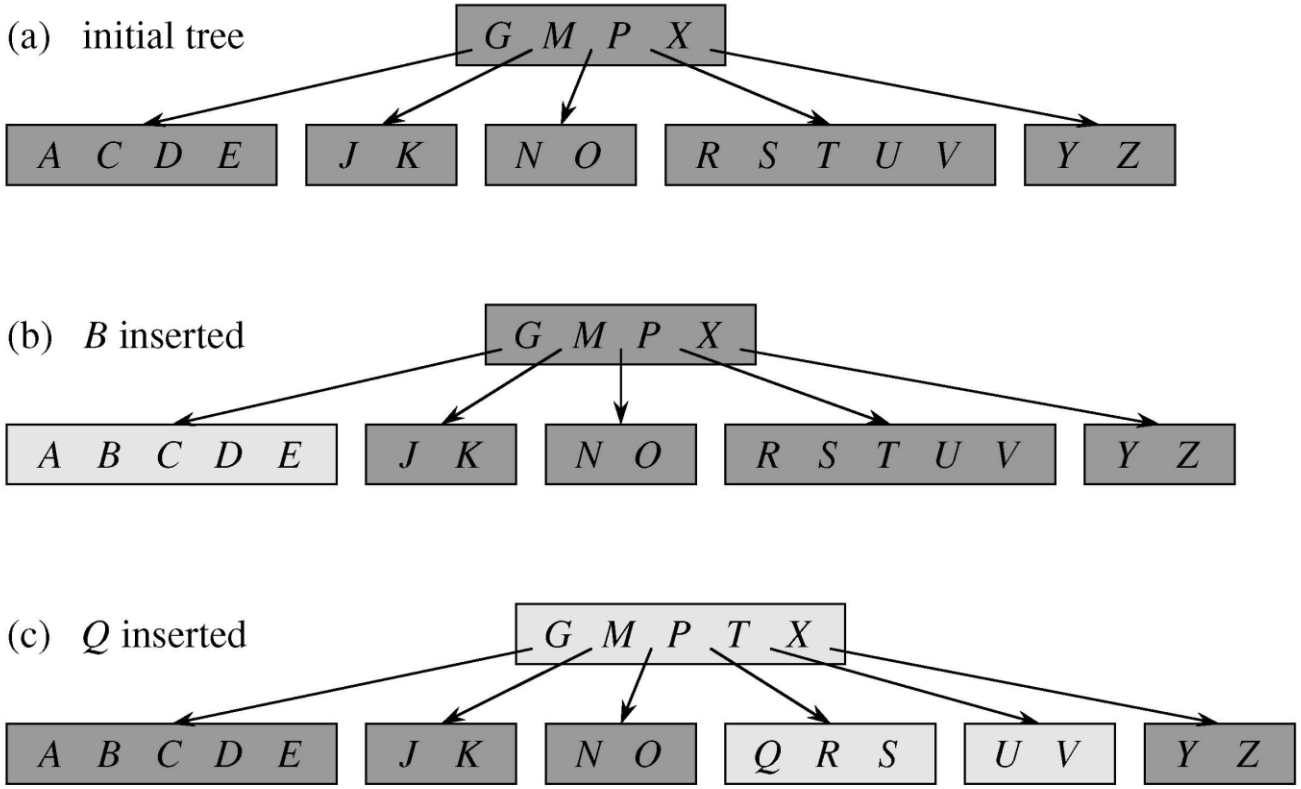
Einfügen von B



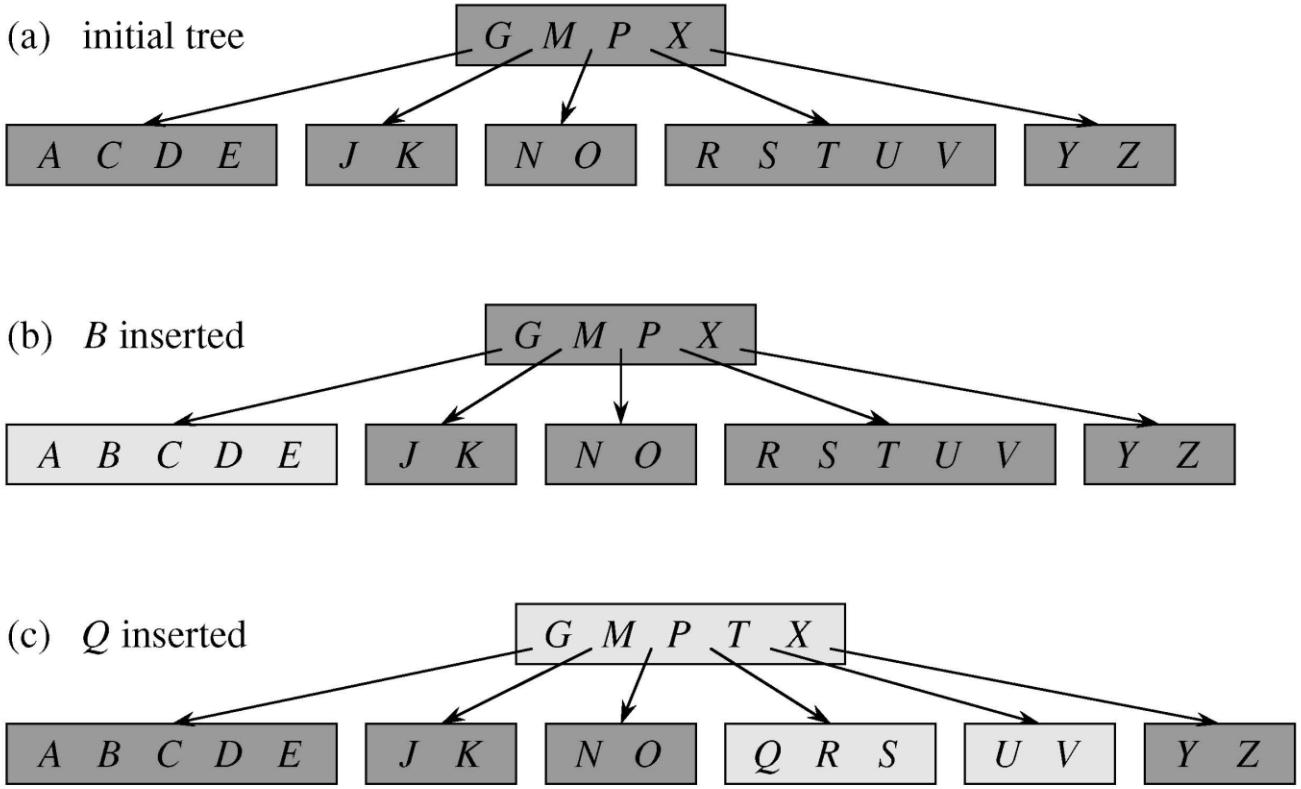
Einfügen von Q?



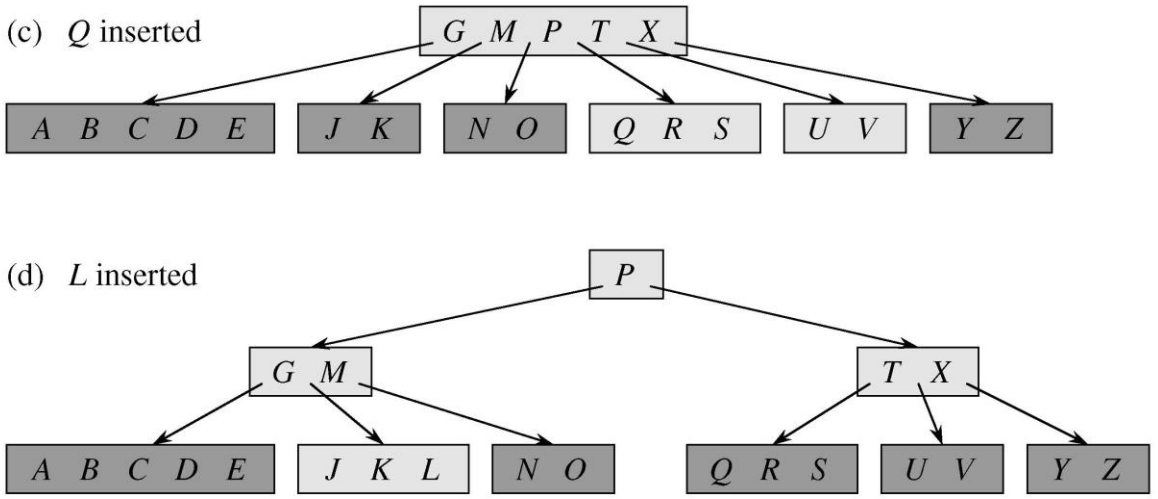
Einfügen von Q: spalten nötig



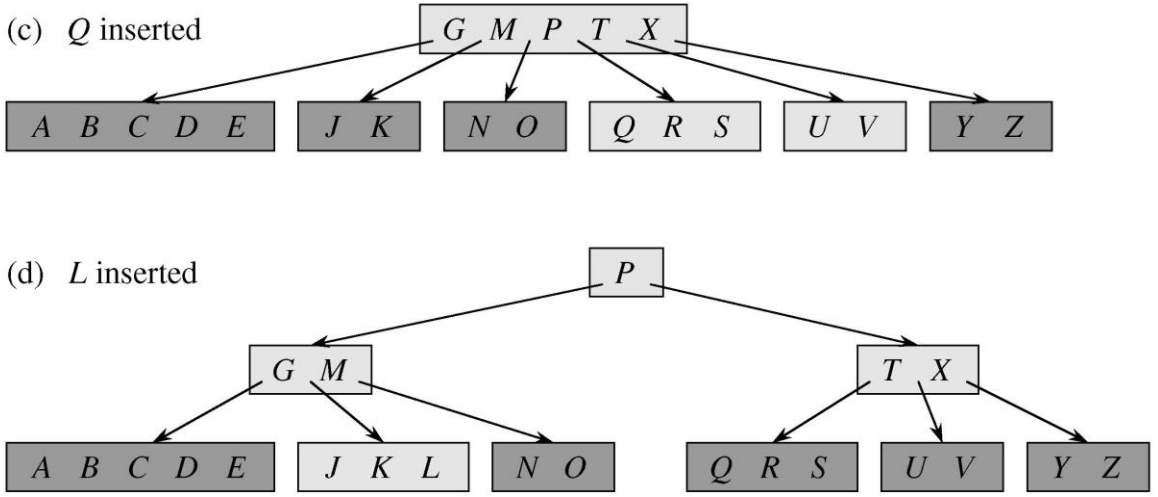
Einfügen von L?



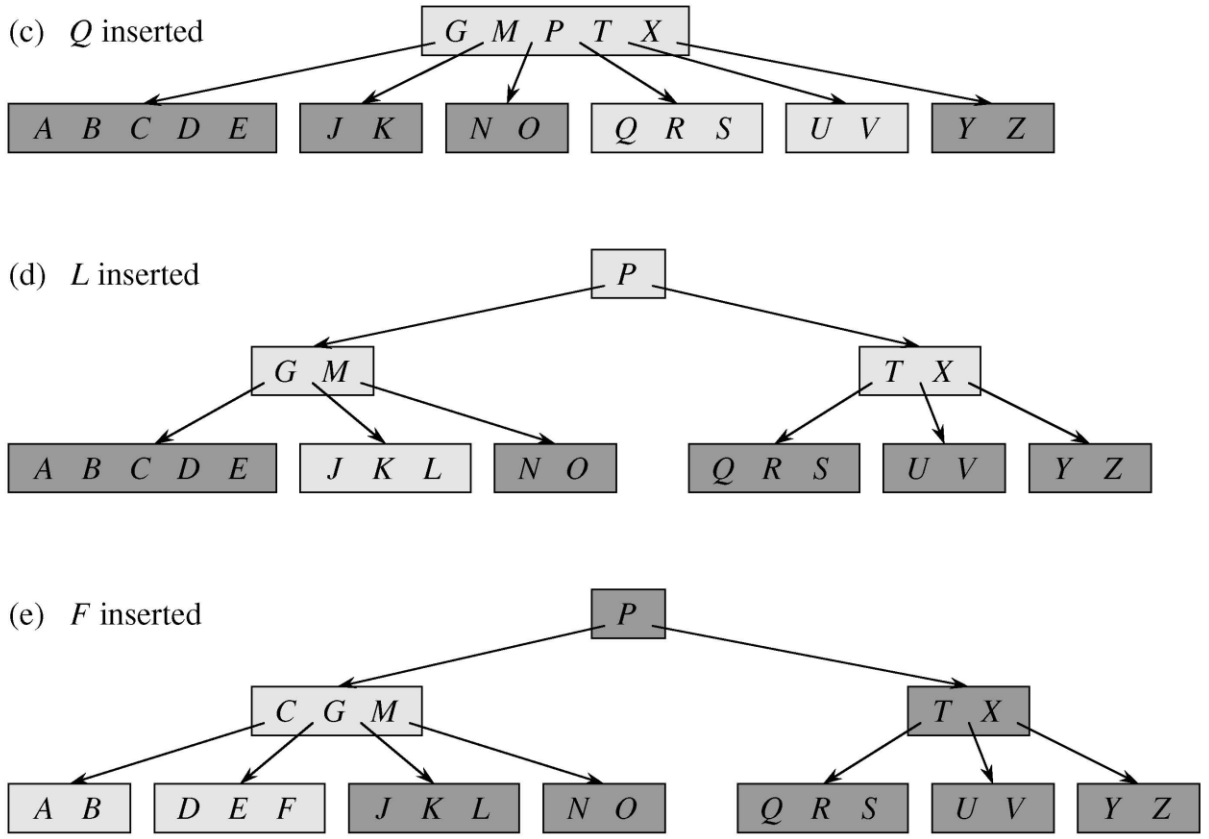
Einfügen von L



Einfügen von F?



Einfügen von F



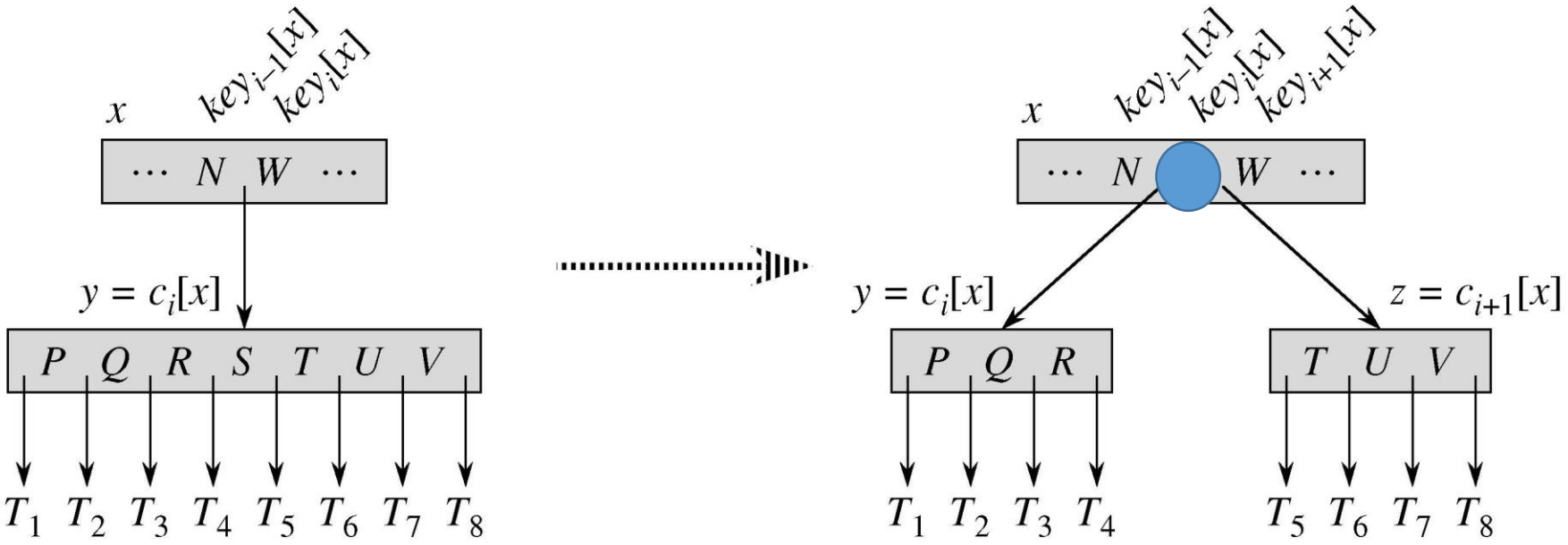
Erzeugung eines B-Baumes

B-TREE-CREATE(T)

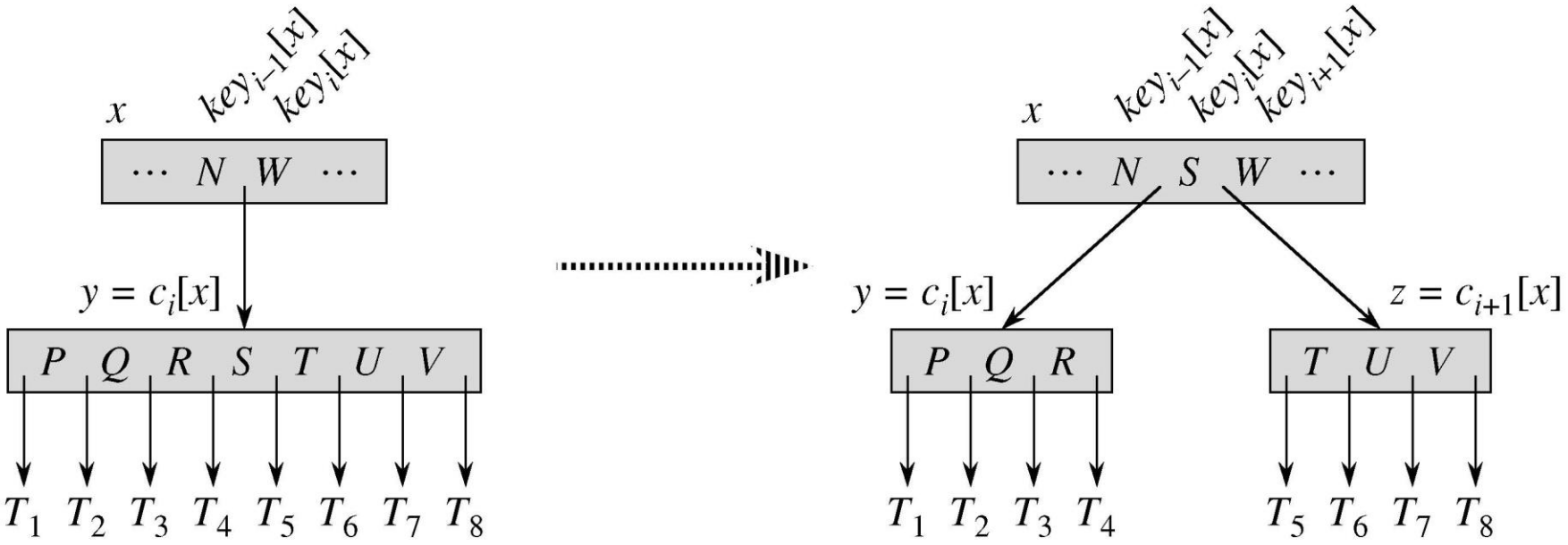
1 $x \leftarrow$ ALLOCATE-NODE()
2 $leaf[x] \leftarrow$ TRUE
3 $n[x] \leftarrow 0$
4 DISK-WRITE(x)
5 $root[T] \leftarrow x$

B-TREE-CREATE requires $O(1)$ disk operations and $O(1)$ CPU time.

Spalten


x
$key_{i-1}[x]$
$key_i[x]$
… N W …
$y = c_i[x]$
P Q R S T U V
T_1 T_2 T_3 T_4 T_5 T_6 T_7 T_8
x
$key_{i-1}[x]$
$key_i[x]$
$key_{i+1}[x]$
… N W …
$y = c_i[x]$
$z = c_{i+1}[x]$
P Q R
T U V
T_1 T_2 T_3 T_4
T_5 T_6 T_7 T_8

Spalten


x
$key_{i-1}[x]$
$key_i[x]$
… N W …
$y = c_i[x]$
P Q R S T U V
T_1 T_2 T_3 T_4 T_5 T_6 T_7 T_8
x
$key_{i-1}[x]$
$key_i[x]$
$key_{i+1}[x]$
… N S W …
$y = c_i[x]$
$z = c_{i+1}[x]$
P Q R
T U V
T_1 T_2 T_3 T_4
T_5 T_6 T_7 T_8

Spalten

```
B-TREE-SPLIT-CHILD(x, i, y)
 1  z ← ALLOCATE-NODE()
 2  leaf[z] ← 1
 3  n[z] ← 2
 4  for j ← 1 to t − 1
 5       do key_j[z] ← 3
 6  if not leaf[y]
 7     then for j ← 1 to t
 8               do c_j[z] ← 3
 9  n[y] ← 4
10  for j ← n[x] + 1 downto i + 1
11       do c_{j+1}[x] ← 5
12  5 ← z
13  for j ← n[x] downto i
14       do key_{j+1}[x] ← 5
15  key_i[x] ← 5
16  n[x] ← n[x] + 1
17  DISK-WRITE(y)
18  DISK-WRITE(z)
19  DISK-WRITE(x)
```

Spalten

```
B-TREE-SPLIT-CHILD(x, i, y)
 1  z ← ALLOCATE-NODE()
 2  leaf[z] ← leaf[y]
 3  n[z] ← 2
 4  for j ← 1 to t − 1
 5       do key_j[z] ← 3
 6  if not leaf[y]
 7     then for j ← 1 to t
 8               do c_j[z] ← 3
 9  n[y] ← 4
10  for j ← n[x] + 1 downto i + 1
11       do c_{j+1}[x] ← 5
12  5 ← z
13  for j ← n[x] downto i
14       do key_{j+1}[x] ← 5
15  key_i[x] ← 5
16  n[x] ← n[x] + 1
17  DISK-WRITE(y)
18  DISK-WRITE(z)
19  DISK-WRITE(x)
```

Spalten

```
B-TREE-SPLIT-CHILD(x, i, y)
 1  z ← ALLOCATE-NODE()
 2  leaf[z] ← leaf[y]
 3  n[z] ← t − 1
 4  for j ← 1 to t − 1
 5        do key_j[z] ← 3
 6  if not leaf[y]
 7     then for j ← 1 to t
 8                do c_j[z] ← 3
 9  n[y] ← 4
10  for j ← n[x] + 1 downto i + 1
11        do c_{j+1}[x] ← 5
12  5 ← z
13  for j ← n[x] downto i
14        do key_{j+1}[x] ← 5
15  key_i[x] ← 5
16  n[x] ← n[x] + 1
17  DISK-WRITE(y)
18  DISK-WRITE(z)
19  DISK-WRITE(x)
```

Spalten

B-TREE-SPLIT-CHILD(x, i, y)

```
 1  z ← ALLOCATE-NODE()
 2  leaf[z] ← leaf[y]
 3  n[z] ← t − 1
 4  for j ← 1 to t − 1
 5       do key_j[z] ← key_{j+t}[y]
 6  if not leaf[y]
 7     then for j ← 1 to t
 8               do c_j[z] ← c_{j+t}[y]
 9  n[y] ← 4
10  for j ← n[x] + 1 downto i + 1
11       do c_{j+1}[x] ← 5
12  5 ← z
13  for j ← n[x] downto i
14       do key_{j+1}[x] ← 5
15  key_i[x] ← 5
16  n[x] ← n[x] + 1
17  DISK-WRITE(y)
18  DISK-WRITE(z)
19  DISK-WRITE(x)
```

Spalten

```
B-TREE-SPLIT-CHILD(x, i, y)
 1  z ← ALLOCATE-NODE()
 2  leaf[z] ← leaf[y]
 3  n[z] ← t − 1
 4  for j ← 1 to t − 1
 5       do key_j[z] ← key_{j+t}[y]
 6  if not leaf[y]
 7     then for j ← 1 to t
 8               do c_j[z] ← c_{j+t}[y]
 9  n[y] ← t − 1
10  for j ← n[x] + 1 downto i + 1
11       do c_{j+1}[x] ← 5
12  5 ← z
13  for j ← n[x] downto i
14       do key_{j+1}[x] ← 5
15  key_i[x] ← 5
16  n[x] ← n[x] + 1
17  DISK-WRITE(y)
18  DISK-WRITE(z)
19  DISK-WRITE(x)
```

Spalten

```
B-TREE-SPLIT-CHILD(x, i, y)
 1  z ← ALLOCATE-NODE()
 2  leaf[z] ← leaf[y]
 3  n[z] ← t − 1
 4  for j ← 1 to t − 1
 5       do key_j[z] ← key_{j+t}[y]
 6  if not leaf[y]
 7     then for j ← 1 to t
 8               do c_j[z] ← c_{j+t}[y]
 9  n[y] ← t − 1
10  for j ← n[x] + 1 downto i + 1
11       do c_{j+1}[x] ← c_j[x]
12  c_{i+1}[x] ← z
13  for j ← n[x] downto i
14       do key_{j+1}[x] ← key_j[x]
15  key_i[x] ← key_t[y]
16  n[x] ← n[x] + 1
17  DISK-WRITE(y)
18  DISK-WRITE(z)
19  DISK-WRITE(x)
```

Einfügen

```
B-TREE-INSERT(T, k)
 1   r ← root[T]
 2   if n[r] = 1
 3       then s ← ALLOCATE-NODE()
 4            root[T] ← s
 5            leaf[s] ← 2
 6            n[s] ← 2
 7            c1[s] ← 2
 8            B-TREE-SPLIT-CHILD(s, 1, r)
 9            B-TREE-INSERT-NONFULL(s, k)
10       else B-TREE-INSERT-NONFULL(r, k)
```

Einfügen

```
B-TREE-INSERT(T, k)
 1  r ← root[T]
 2  if n[r] = 2t − 1
 3     then s ← ALLOCATE-NODE()
 4          root[T] ← s
 5          leaf[s] ← 2
 6          n[s] ← 2
 7          c1[s] ← 2
 8          B-TREE-SPLIT-CHILD(s, 1, r)
 9          B-TREE-INSERT-NONFULL(s, k)
10     else B-TREE-INSERT-NONFULL(r, k)
```

Einfügen

```
B-TREE-INSERT(T, k)
 1  r ← root[T]
 2  if n[r] = 2t − 1
 3     then s ← ALLOCATE-NODE()
 4          root[T] ← s
 5          leaf[s] ← FALSE
 6          n[s] ← 0
 7          c1[s] ← r
 8          B-TREE-SPLIT-CHILD(s, 1, r)
 9          B-TREE-INSERT-NONFULL(s, k)
10     else B-TREE-INSERT-NONFULL(r, k)
```

Einfügen von k wenn Wurzel x nicht voll ist

```
B-TREE-INSERT-NONFULL(x, k)
 1  i ← n[x]
 2  if leaf[x]
 3     then while i ≥ 1 and k < key_i[x]
 4                do key_{i+1}[x] ← [1]
 5                   i ← i − 1
 6          [2] ← k
 7          n[x] ← n[x] + 1
 8          DISK-WRITE(x)
 9     else while i ≥ 1 and k < [1]
10                do i ← i − 1
11          i ← i + 1
12          DISK-READ(c_i[x])
13          if n[c_i[x]] = [3]
14            then B-TREE-SPLIT-CHILD(x, i, c_i[x])
15                 if k > key_i[x]
16                   then [4]
17          B-TREE-INSERT-NONFULL(c_i[x], k)
```

Einfügen von k wenn Wurzel x nicht voll ist

```
B-TREE-INSERT-NONFULL(x, k)
 1  i ← n[x]
 2  if leaf[x]
 3     then while i ≥ 1 and k < key_i[x]
 4               do key_{i+1}[x] ← key_i[x]
 5                  i ← i − 1
 6          [ 2 ] ← k
 7          n[x] ← n[x] + 1
 8          DISK-WRITE(x)
 9     else while i ≥ 1 and k < key_i[x]
10               do i ← i − 1
11          i ← i + 1
12          DISK-READ(c_i[x])
13          if n[c_i[x]] = [ 3 ]
14            then B-TREE-SPLIT-CHILD(x, i, c_i[x])
15                 if k > key_i[x]
16                   then [ 4 ]
17          B-TREE-INSERT-NONFULL(c_i[x], k)
```

Einfügen von k wenn Wurzel x nicht voll ist

```
B-TREE-INSERT-NONFULL(x, k)
 1  i ← n[x]
 2  if leaf[x]
 3     then while i ≥ 1 and k < key_i[x]
 4               do key_{i+1}[x] ← key_i[x]
 5                  i ← i − 1
 6          key_{i+1}[x] ← k
 7          n[x] ← n[x] + 1
 8          DISK-WRITE(x)
 9     else while i ≥ 1 and k < key_i[x]
10               do i ← i − 1
11          i ← i + 1
12          DISK-READ(c_i[x])
13          if n[c_i[x]] = 3
14            then B-TREE-SPLIT-CHILD(x, i, c_i[x])
15                 if k > key_i[x]
16                   then 4
17          B-TREE-INSERT-NONFULL(c_i[x], k)
```

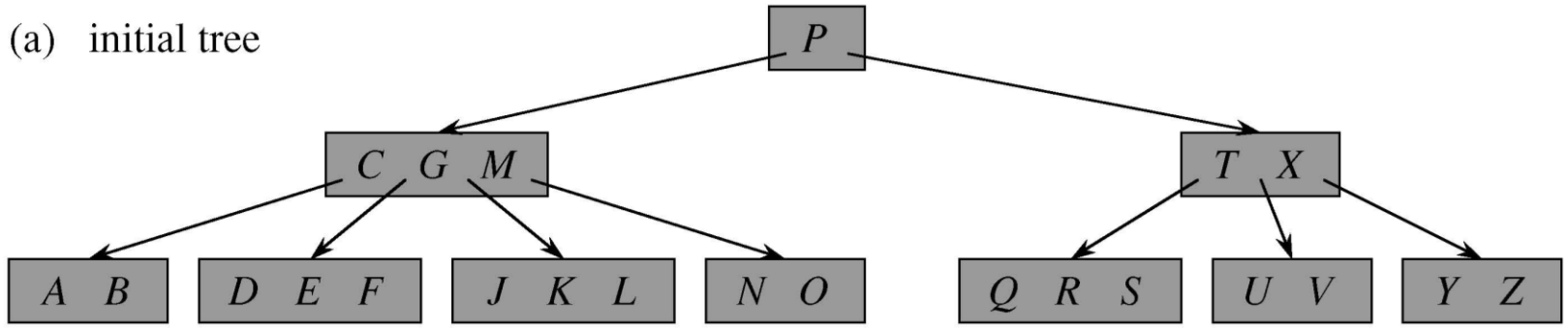
Einfügen von k wenn Wurzel x nicht voll ist

```
B-TREE-INSERT-NONFULL(x, k)
 1  i ← n[x]
 2  if leaf[x]
 3     then while i ≥ 1 and k < key_i[x]
 4                do key_{i+1}[x] ← key_i[x]
 5                   i ← i − 1
 6          key_{i+1}[x] ← k
 7          n[x] ← n[x] + 1
 8          DISK-WRITE(x)
 9     else while i ≥ 1 and k < key_i[x]
10                do i ← i − 1
11          i ← i + 1
12          DISK-READ(c_i[x])
13          if n[c_i[x]] = 2t − 1
14            then B-TREE-SPLIT-CHILD(x, i, c_i[x])
15                 if k > key_i[x]
16                   then 4
17          B-TREE-INSERT-NONFULL(c_i[x], k)
```

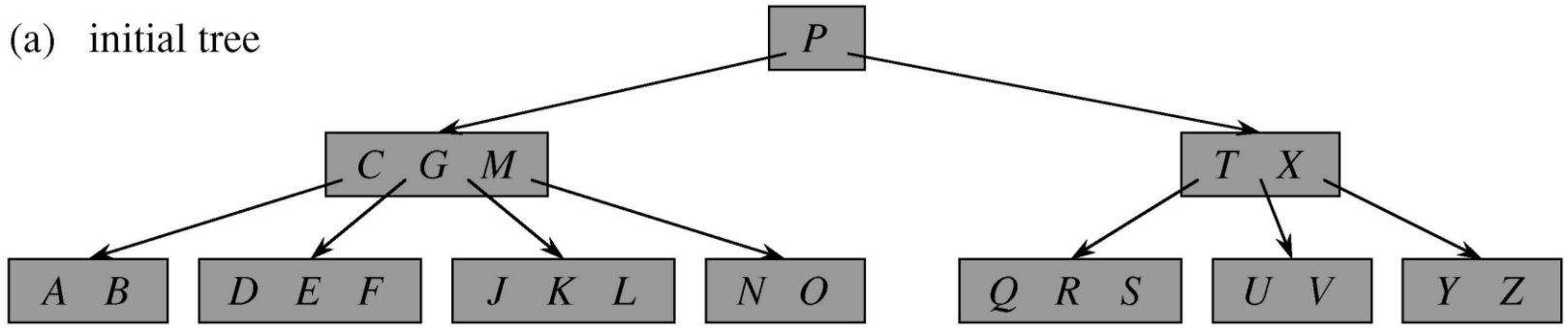
Einfügen von k wenn Wurzel x nicht voll ist

```
B-TREE-INSERT-NONFULL(x, k)
 1  i ← n[x]
 2  if leaf[x]
 3     then while i ≥ 1 and k < key_i[x]
 4                do key_{i+1}[x] ← key_i[x]
 5                   i ← i − 1
 6          key_{i+1}[x] ← k
 7          n[x] ← n[x] + 1
 8          DISK-WRITE(x)
 9     else while i ≥ 1 and k < key_i[x]
10                do i ← i − 1
11          i ← i + 1
12          DISK-READ(c_i[x])
13          if n[c_i[x]] = 2t − 1
14            then B-TREE-SPLIT-CHILD(x, i, c_i[x])
15                 if k > key_i[x]
16                   then i ← i + 1
17          B-TREE-INSERT-NONFULL(c_i[x], k)
```

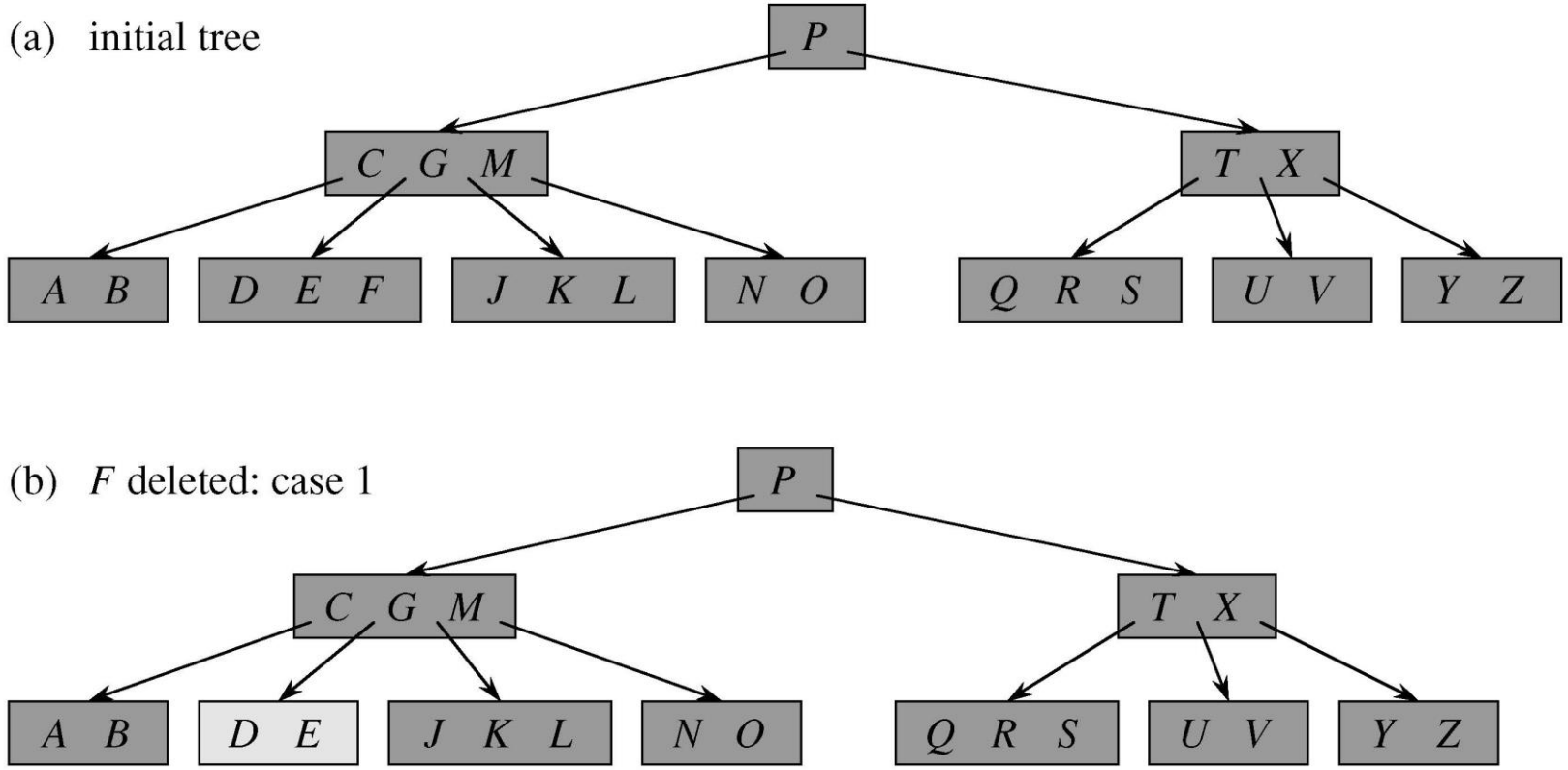
Löschen, t = 3



Fall 1: k ist in x und x ist ein Blatt z.B. löschen von F

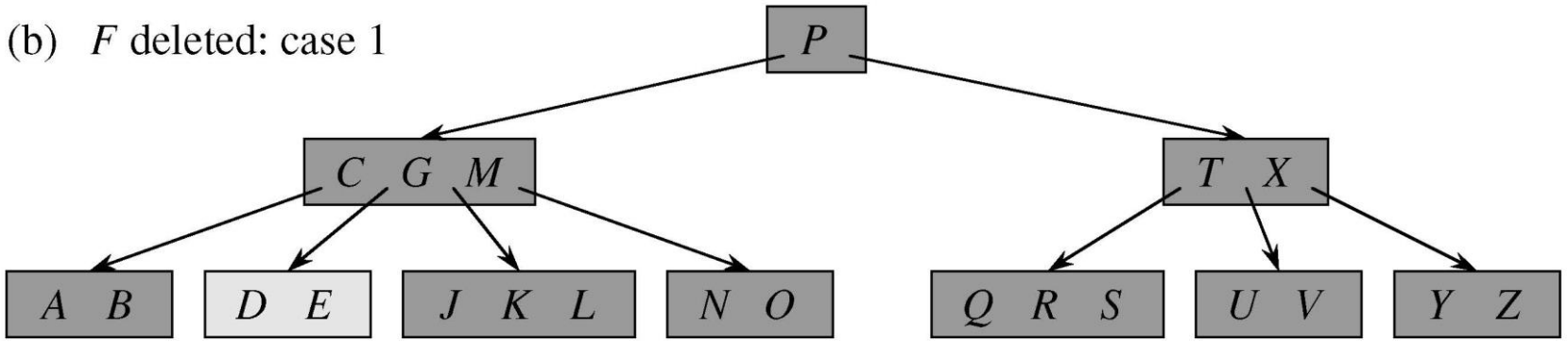


Fall 1: k ist in x und x ist ein Blatt z.B. löschen von F

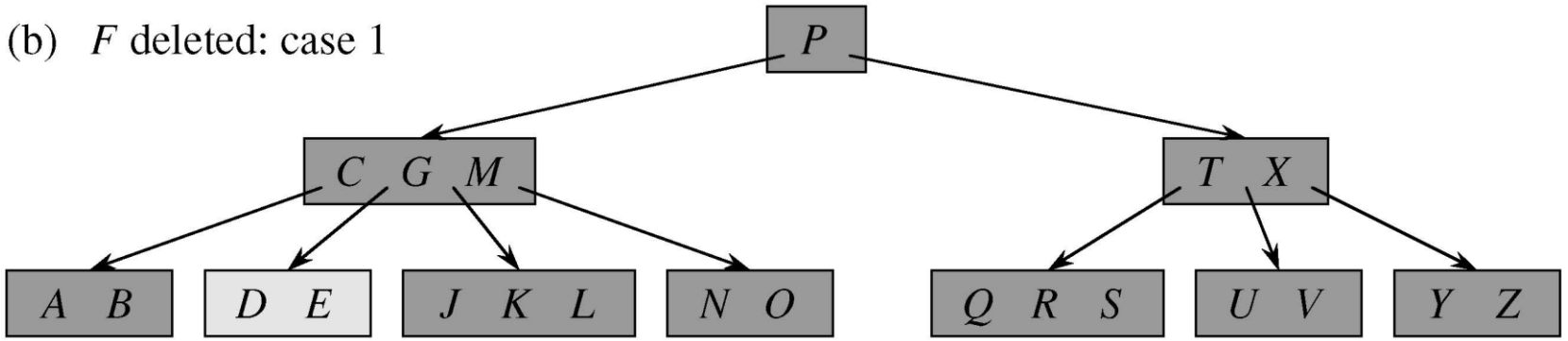


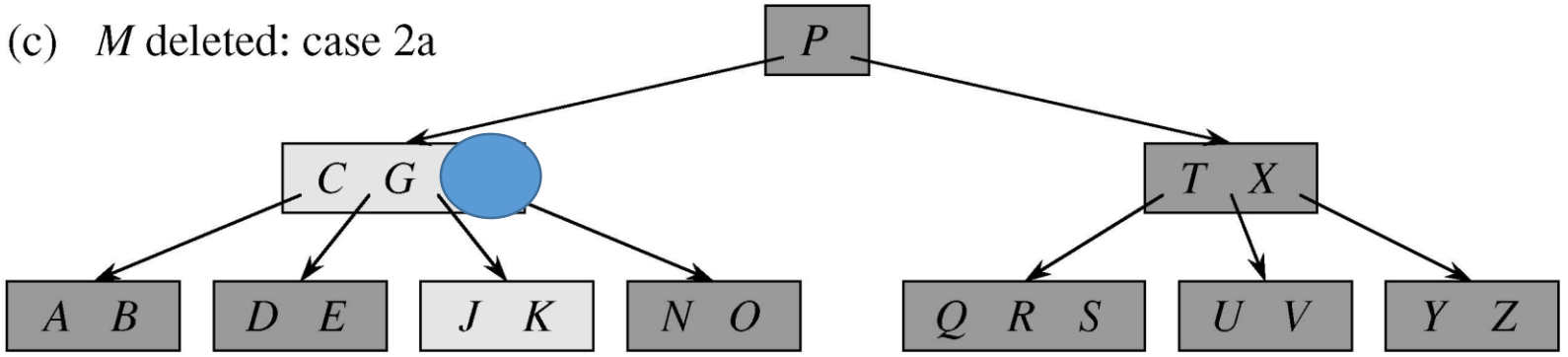
Fall 2: k ist in x und x ist kein Blatt

Fall 2a: Das Kind y unmittelbar vor k hat mindestens t Knoten z.B. lösche M

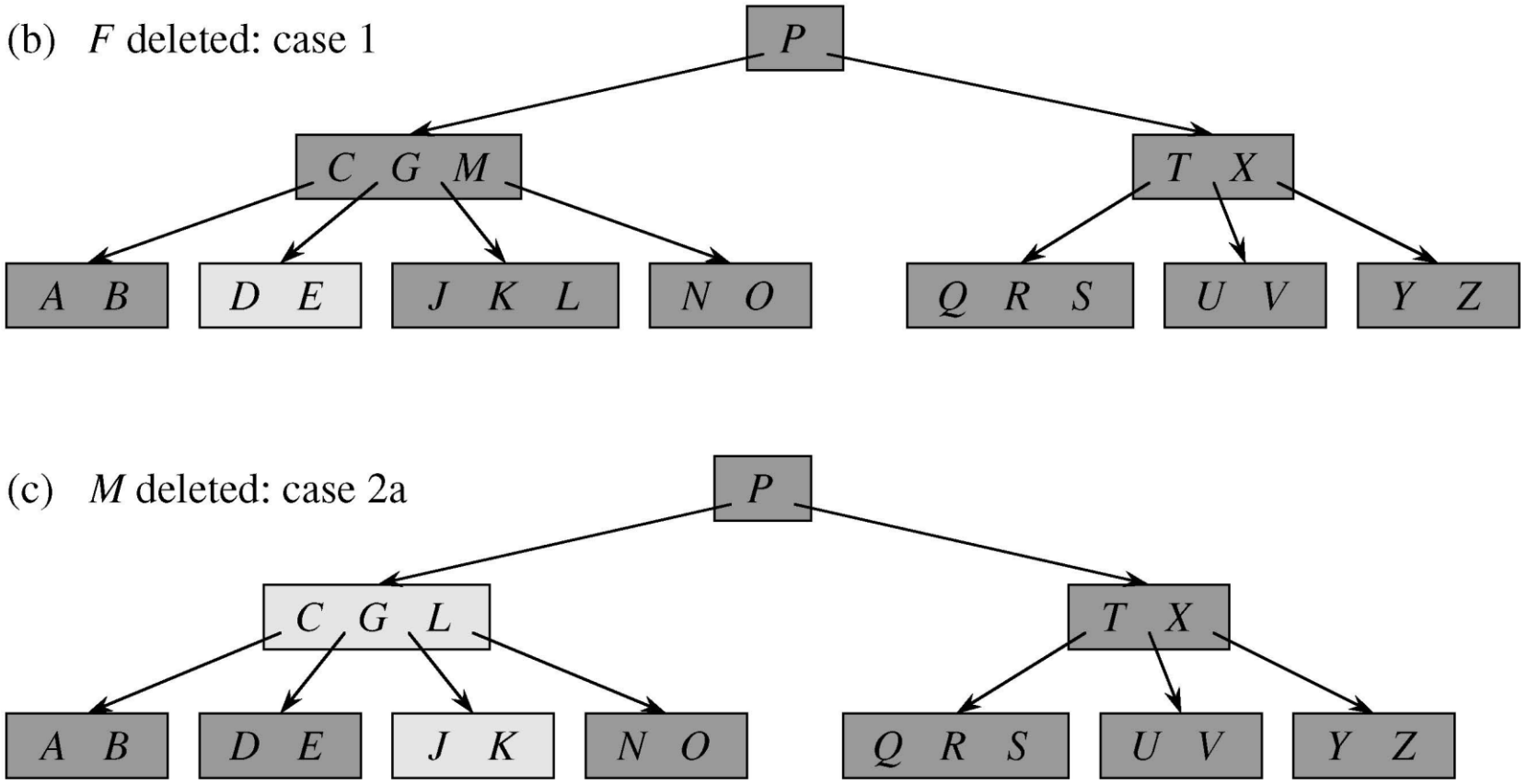


Fall 2a: Das Kind unmittelbar vor k hat mindestens t Knoten
z.B. lösche M



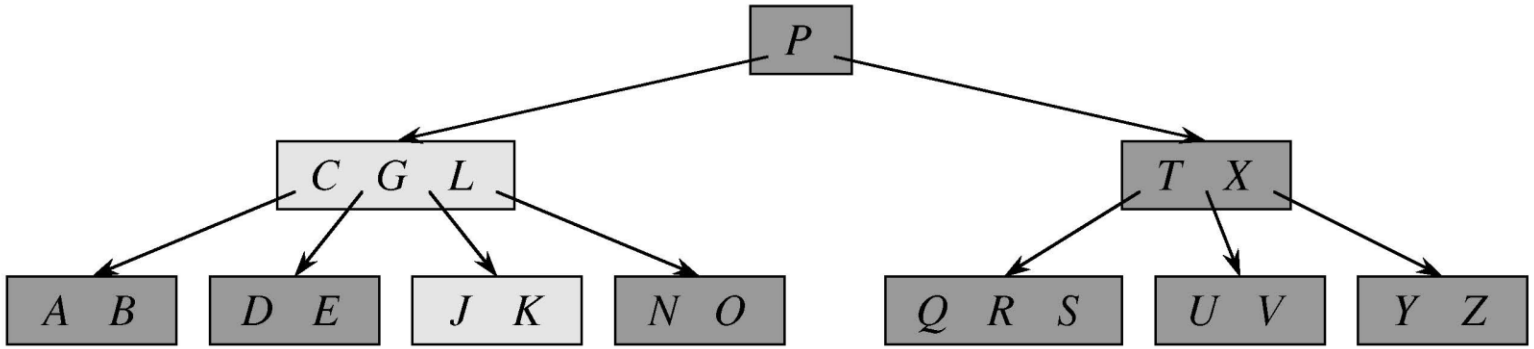


Fall 2a: Das Kind unmittelbar vor k hat mindestens t Knoten
z.B. lösche M

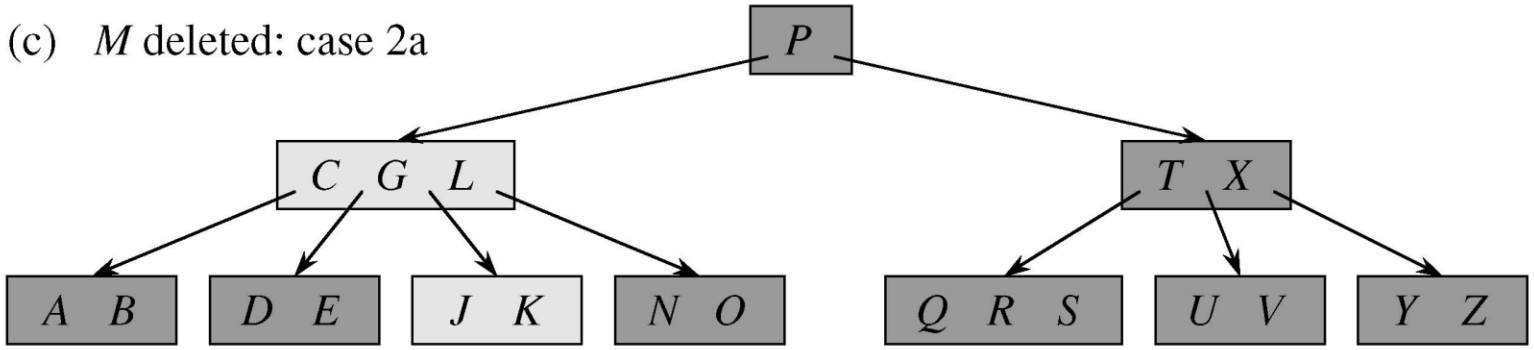


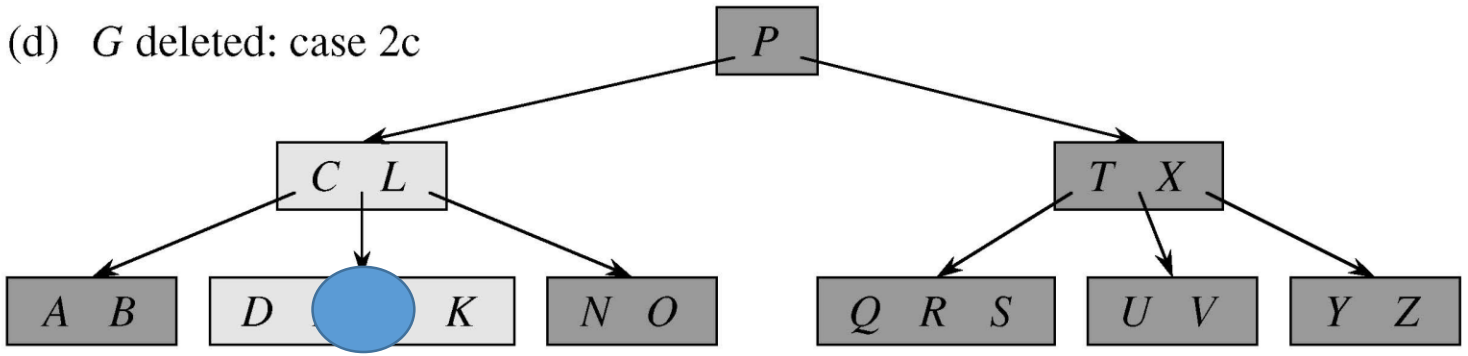
Fall 2b: Das Kind unmittelbar nach k hat mindestens t Knoten analog

Fall 2c: weder a noch b
z.B. lösche G

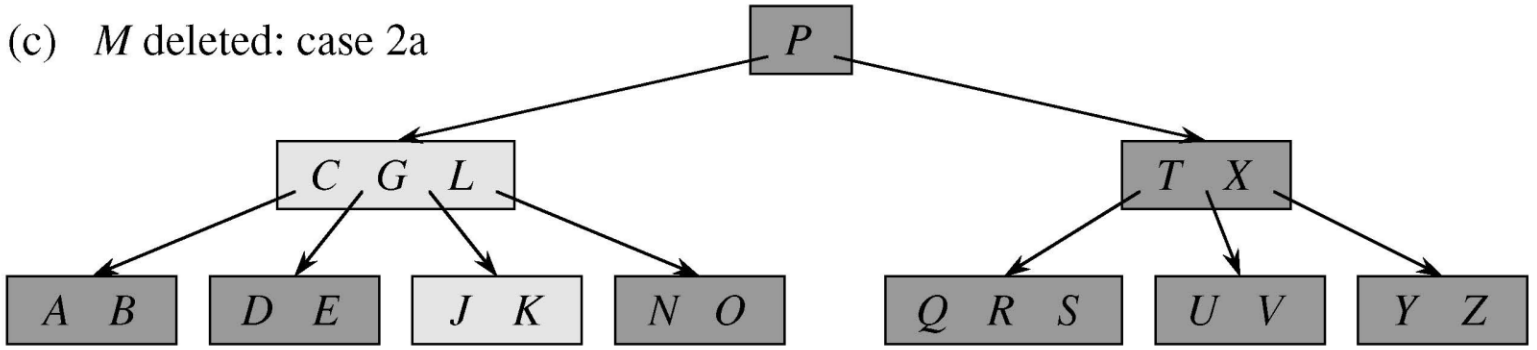


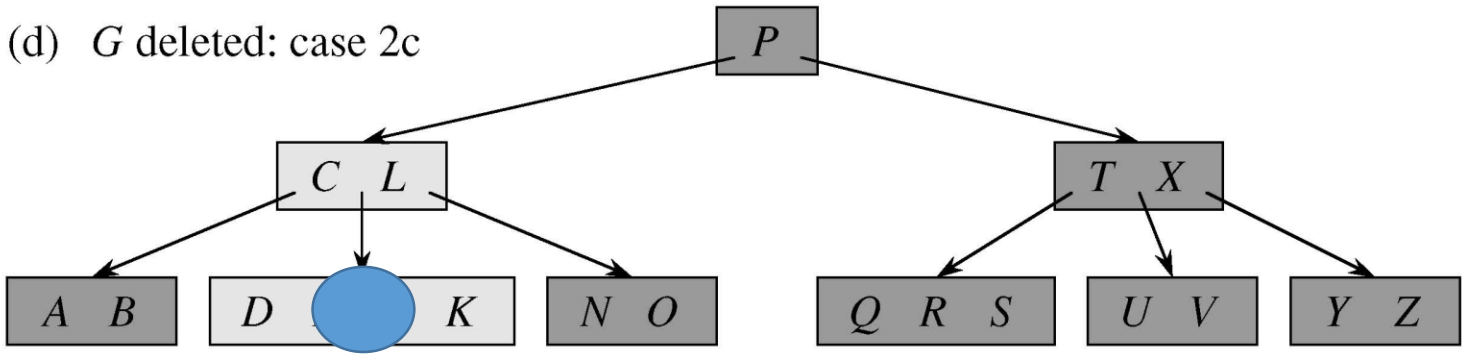
Fall 2c: weder a noch b
z.B. lösche G





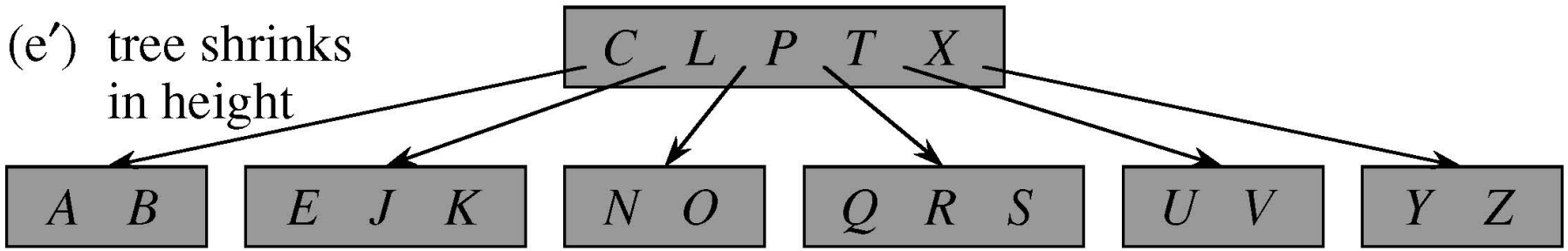
Fall 2c: weder a noch b
z.B. lösche G



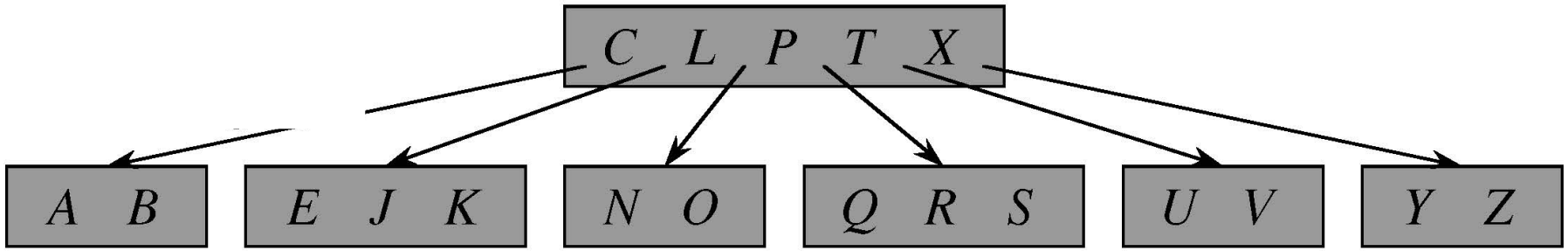


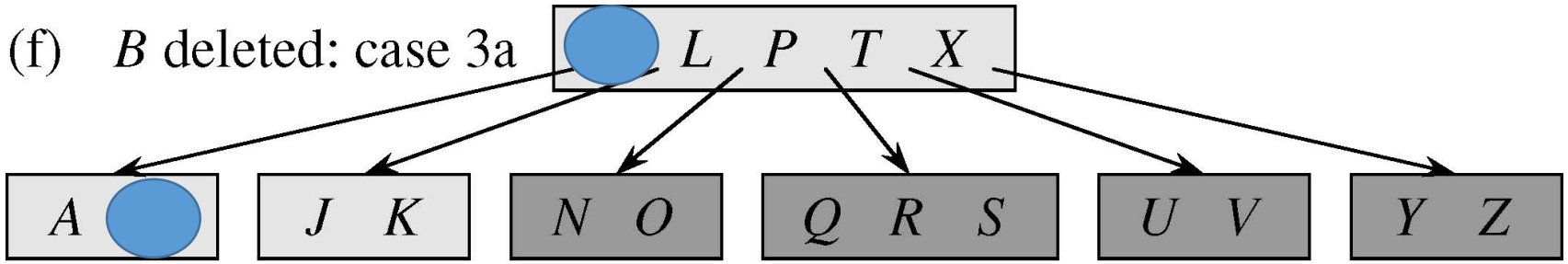
Fall 3: x ist ein innerer Knoten und k ist nicht in x

Fall 3a) c_i[x] hat t-1 Knoten aber ein Geschwister hat t Knoten
z.B. lösche B

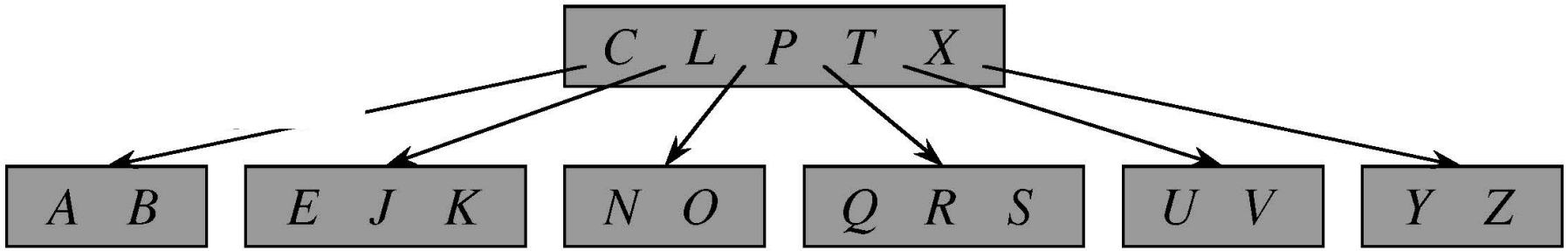


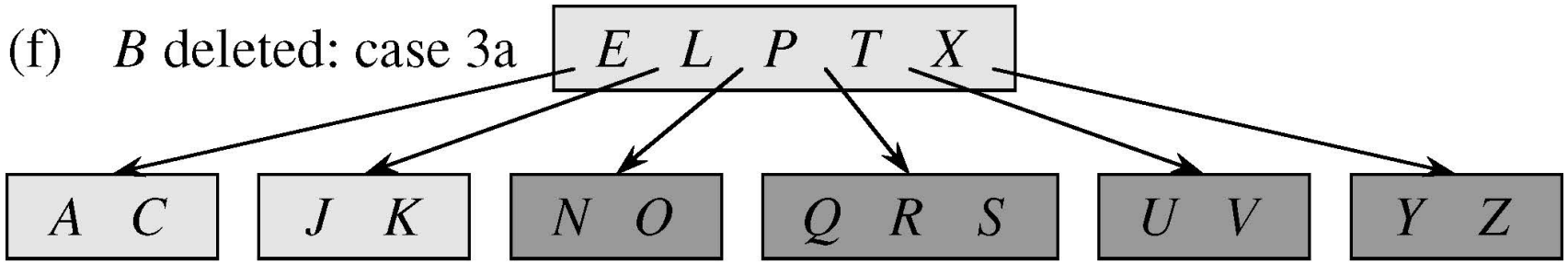
Fall 3a) c_i[x] hat t-1 Knoten aber ein Geschwister hat t Knoten
z.B. lösche B



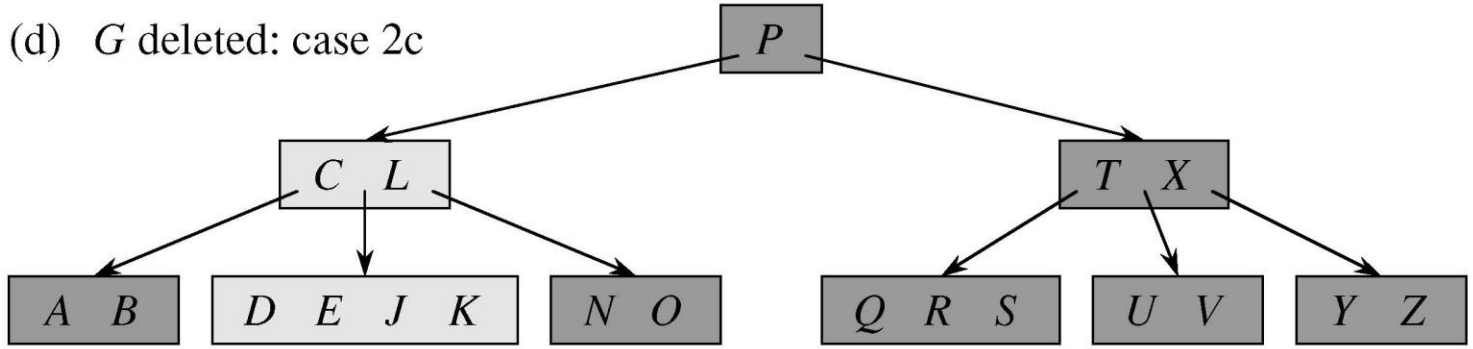


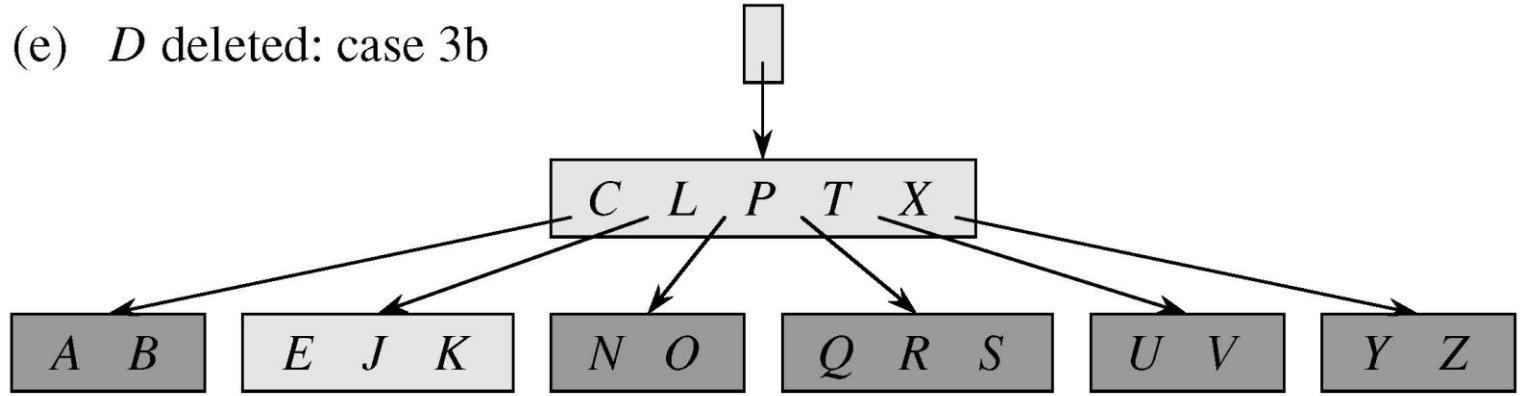
Fall 3a) c_i[x] hat t-1 Knoten aber ein Geschwister hat t Knoten
z.B. lösche B





Fall 3b) c_i[x] und seine Geschwister haben nur t-1 Knoten
z.B. Fange bei P an und lösche D





Fall 3b) c_i[x] und seine Geschwister haben nur t-1 Knoten
z.B. Fange bei P an und lösche D