B-деревья

Дискретный анализ 2012/13

Андрей Калинин, Татьяна Романова

17 сентября 2012 г.

Определение B-дерева

- Общая идея
- Определение
- Минимальная высота B-дерева

Операции с B-деревом

- Создание
- Поиск
- Вставка
- Удаление

Литература

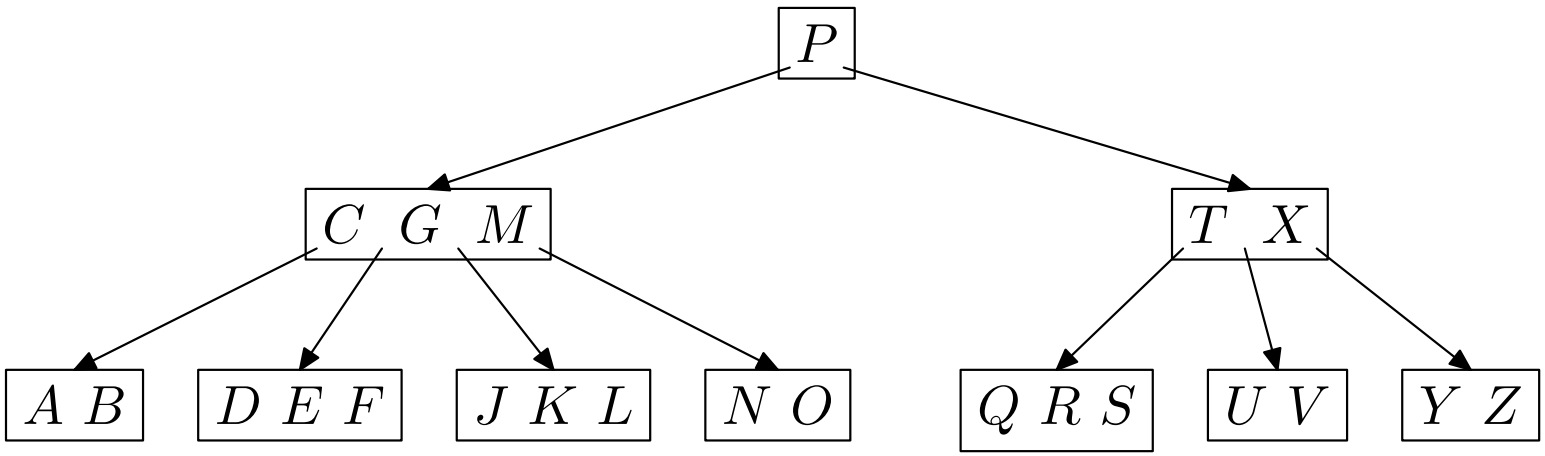
- Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К.. Алгоритмы: построение и анализ, 2-е издание, М.:Вильямс, 2005, стр. 515-536, глава 18, «B-деревья».
- Кнут Д. Искусство программирования, Т.3: Сортировка и поиск, стр. 516-526, глава 6.2.4, «Сильноветвящиеся деревья».

Раздел

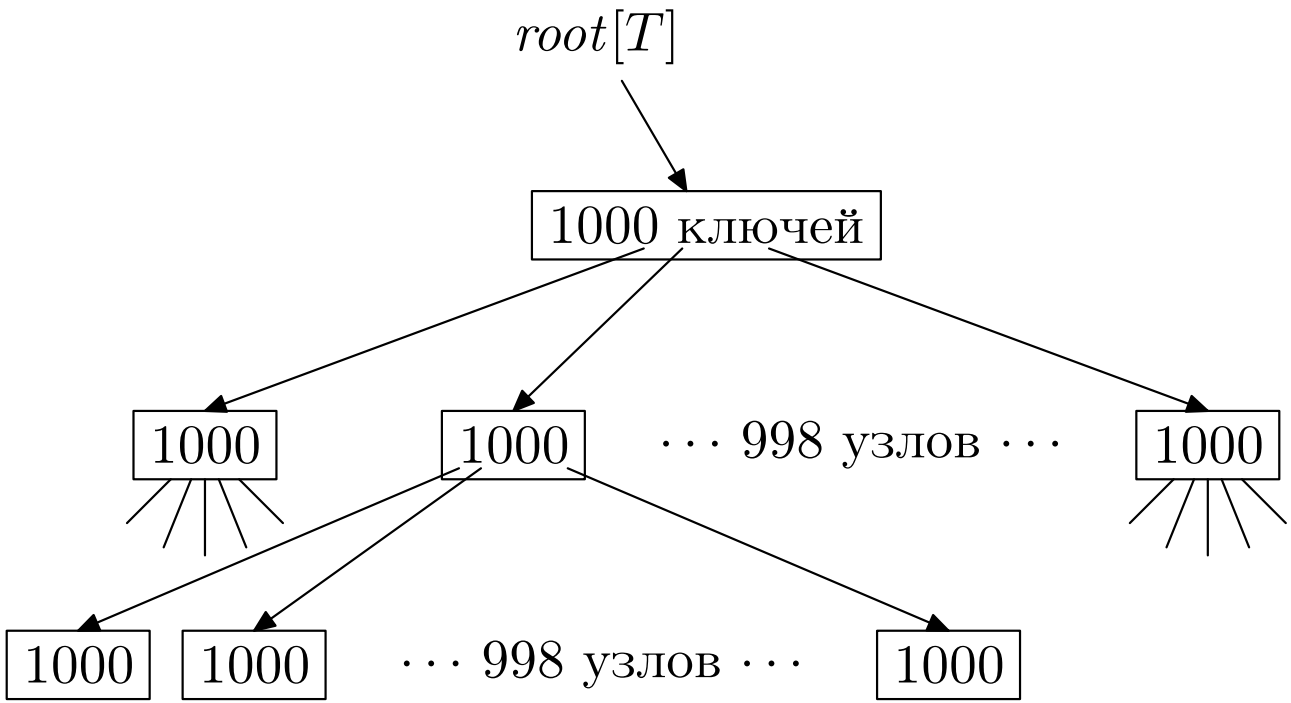
Мотивация

- Данные (и ключи) могут не помещаться в оперативную память.
- Нужно иметь простой механизм загрузки нужной информации.
- Для деревьев просто загружать и выгружать узлы, но у двоичных деревьев в узлах хранится слишком мало информации, чтобы работа с внешнией памятью была бы эффективна (слишком частые обращения неприемлимы).
- Нужны сбалансированные деревья с большой степенью ветвления, в узлах которых хранилось бы много данных, тысячи ключей.

Пример B-дерева



Размер B-дерева



1 узел,
1000 ключей

1001 узел,
1 001 000 ключей

1 002 001 узел,
1 002 001 000 ключей

Нестрогое понятие B-дерева

B-дерево степени t:

- Идеально сбалансированно.
- Во всех узлах хранится до $2t - 1$ ключей и $2t$ соответствующих им указателей на поддеревья.
- Во всех узлах, кроме корневого, хранится как минимум $t - 1$ ключей. В корневом узле хранится как минимум 1 ключ.
- Ключи в узлах отсортированы.
- В поддереве, расположенным между двумя последовательными ключами, находятся значения, большие первого и меньшие второго значения этих ключей.

Раздел

Определение B-дерева

B-дерево T представляет из себя корневое дерево с корнем $root[T]$.
Каждый узел x дерева T содержит следующие поля:

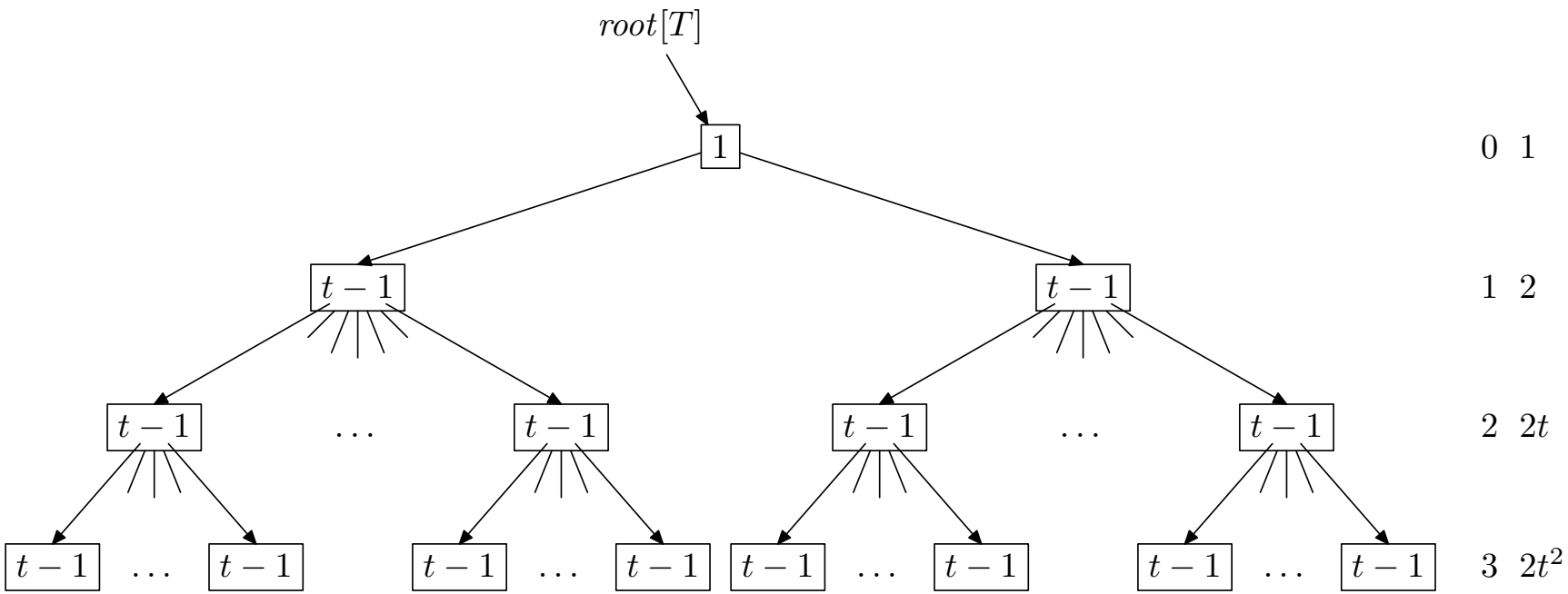
1. $n[x]$, количество ключей в x.
2. $n[x]$ ключей: $key_1[x] \leq key_2[x] \leq \cdots \leq key_{n[x]}[x]$.
3. Признак листа, $leaf[x]$.
4. Внутренние узлы содержат $n[x] + 1$ указателей на поддеревья: $c_1[x]$, $c_2[x]$, $\ldots$, $c_{n[x]+1}[x]$. Листья таких указателей не содержат.

Определение B-дерева

- Ключи $key_i[x]$ определяют диапазоны ключей в поддеревьях. Если k_i — произвольный ключ из поддерева c_i, то:
  $k_1 \leq key_1[x] \leq k_2 \leq key_2[x] \leq \cdots \leq key_{n[x]}[x] \leq k_{n+1}$.
- Все листья расположены на одной и той же высоте, h.
- t: минимальная степень B-дерева.
- Каждый узел, кроме корневого, может содержать как минимум $t-1$ ключ. Корневой узел непустого дерева должен содержать как минимум 1 ключ.
- Каждый узел не может содержать больше $2t-1$ ключей.

Раздел

Минимально заполненное B-дерево



Минимальная высота B-дерева

Теорема

Высота B-дерева T с $n \geq 1$ ключами и минимальной степенью t не превышает $\log_t(n+1)/2$.

Доказательство.

$$n \geq 1 + (t-1)\sum_{i=1}^{h} 2t^{i-1} = 1 + 2(t-1)\frac{t^h - 1}{t-1} = 2t^h - 1,$$

откуда $t^h \leq (n+1)/2$. □

Раздел

Операции доступа к внешней памяти

- DISK-READ($node$) — чтение данных из внешней памяти в узел.
- DISK-WRITE($node$) — запись данных из узла во внешнюю память.
- Конкретная реализация подразумевает наличие LRU (last recent used) буфера некоторого размера, при которой все часто используемые узлы остаются в памяти.
- В алгоритмах предполагается, что корневой узел всегда находится в памяти, хотя на практике это необязательно: при LRU-буферизации корневой узел будет всегда оставаться в памяти и так.

Создание B-дерева

BTree-Create(T)

```
1  x ← Allocate-Node()
2  leaf[x] ← True
3  n[x] ← 0
4  Disk-Write(x)
5  root[T] ← x
```

Раздел

Поиск в B-дереве

BTree-Search(x, k)

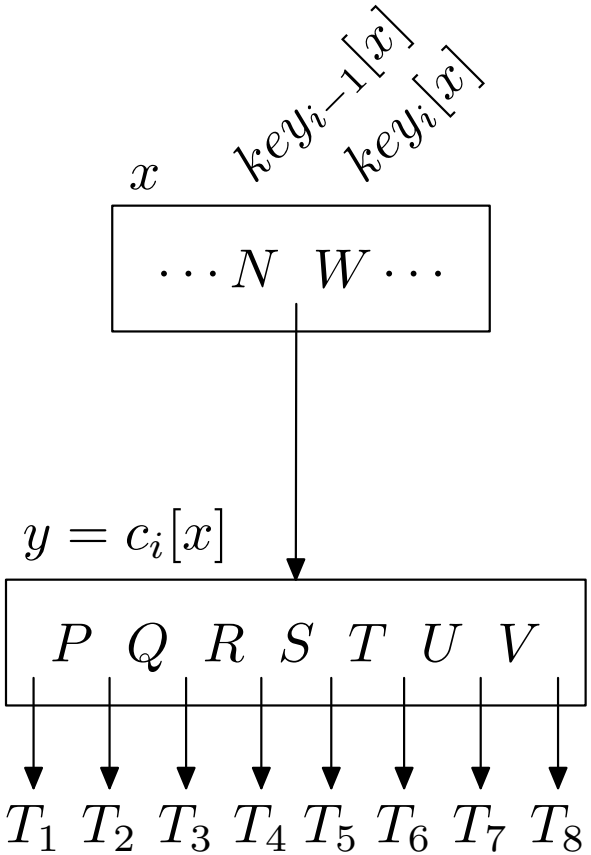
```
1  i ← 1
2  while i ≤ n[x] and k > key_i[x]
3        i ← i + 1
4  if i ≤ n[x] and k = key_i[x]
5        return (x, i)
6  if leaf[x]
7        return NIL
   else
8        Disk-Read(c_i[x])
9        return BTree-Search(c_i[x], k)
```

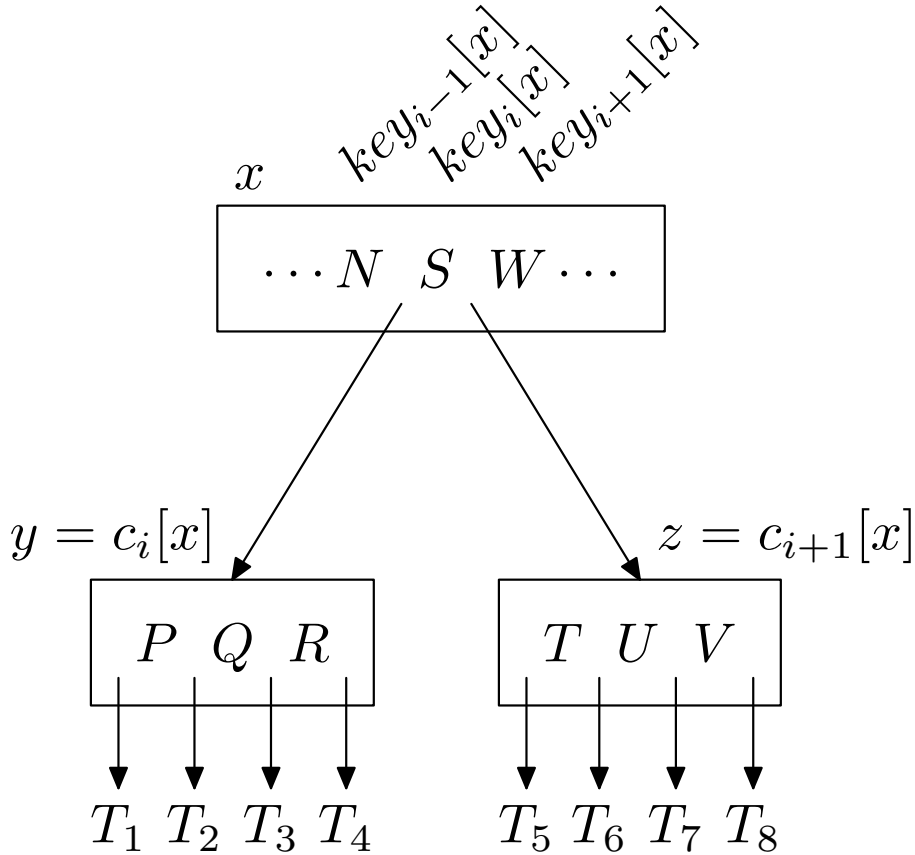
Раздел

Общая идея

- Ищется лист, в который можно вставить новый ключ.
- Нельзя просто так создать новый лист: в листьях должно быть как минимум $t - 1$ ключей.
- Заполненные узлы разбиваются на два по медиане: медиана уходит в родительский узел, вторая половинка существующего узла становится новым узлом.
- Для вставки за один проход при поиске листа разбиваются все заполненные узлы вне зависимости от того, было ли это необходимо или нет.

Разбиение узла, $t = 4$





Разбиение узла B-дерева

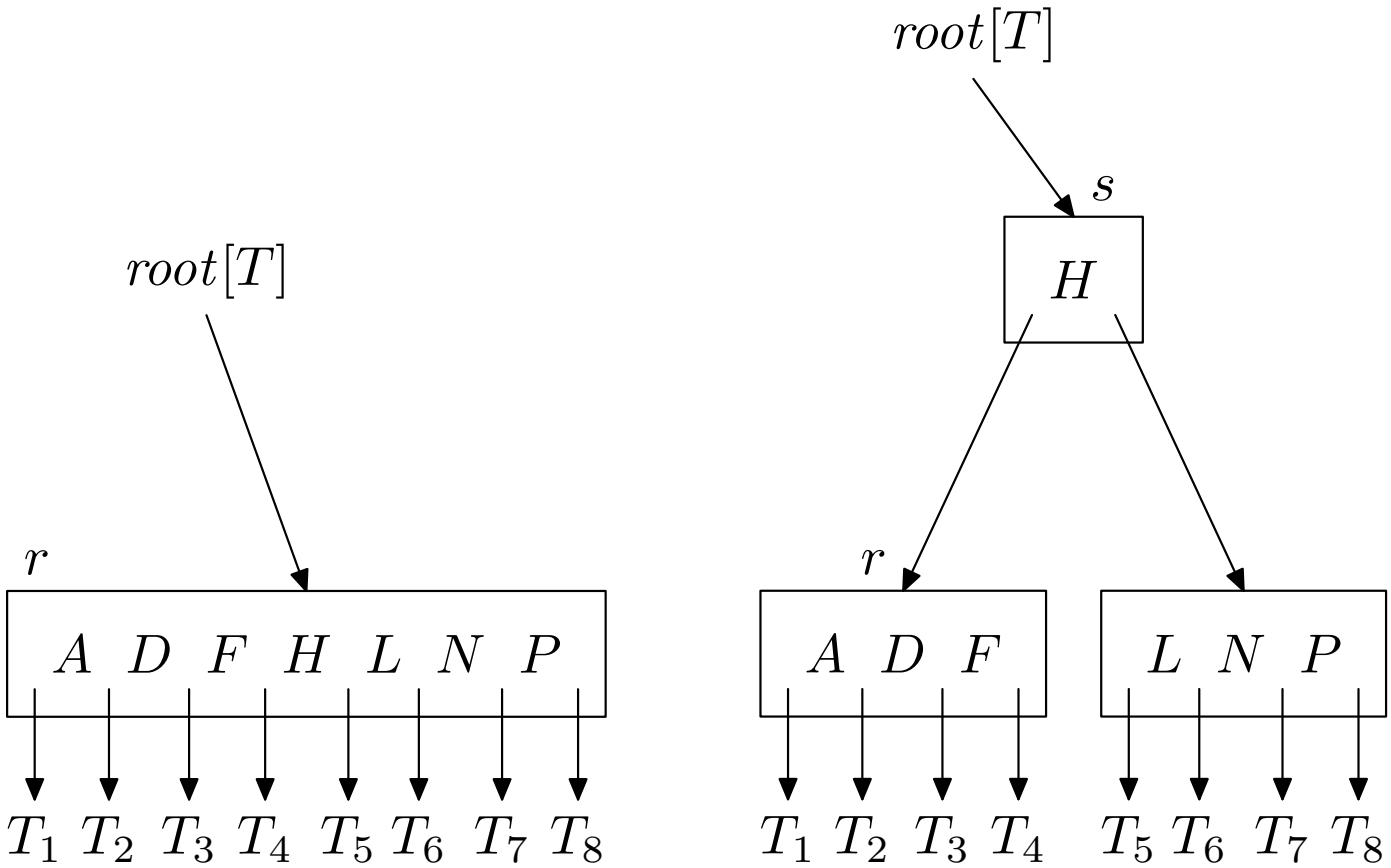
```
BTree-Split-Child(x, i, y)
 1  z ← Allocate-Node()
 2  leaf[z] ← leaf[y]
 3  n[z] ← t − 1
 4  for j ← 1 to t − 1
 5        key_j[z] ← key_{j+t}[y]
 6  if not leaf[y]
 7        for j ← 1 to t
 8              c_j[z] ← c_{j+1}[y]
 9  n[y] ← t − 1
10  for j ← n[x] downto i + 1
11        c_{j+1}[x] ← c_j[x]
12  c_{j+1} ← z
13  for j ← n[x] downto i
14        key_{j+1}[z] ← 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)
```

Вставка ключа k в B-дерево

BTree-Insert(T, k)

```
 1  r ← root[T]
 2  if n[r] = 2t − 1
 3        s ← Allocate-Node()
 4        root[T] ← s
 5        leaf[s] ← FALSE
 6        n[s] ← 0
 7        c_1[s] ← r
 8        BTree-Split-Child(s, 1, r)
 9        BTree-Insert-Nonfull(s, k)
    else
10        BTree-Insert-Nonfull(r, k)
```

Разбиение корня, $t = 4$

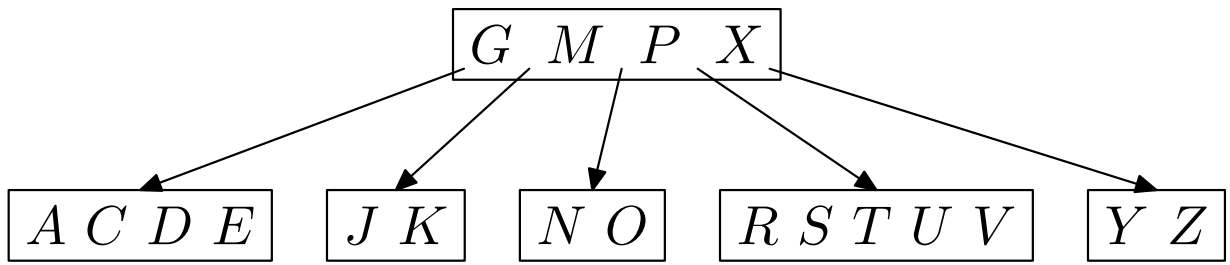


Вставка ключа k в незаполненный узел x

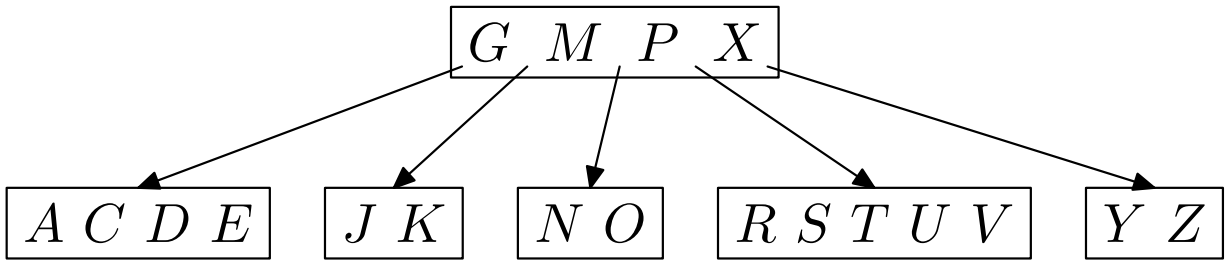
BTree-Insert-Nonfull(x, k)

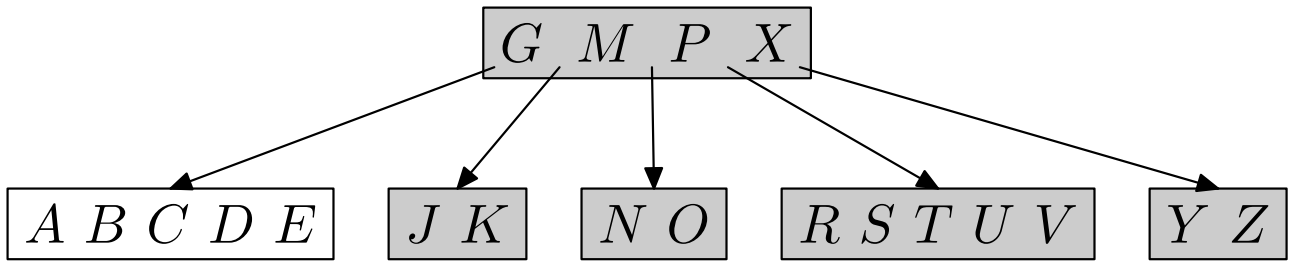
```
 1  i ← n[x]
 2  if leaf[x]
 3        while i ≥ 1 and k < key_i[x]
 4              key_{i+1}[x] ← key_i[x] ; i ← i − 1
 5        key_{i+1}[x] ← k ; n[x] ← n[x] + 1
 6        Disk-Write(x)
 7  else while i ≥ 1 and k < key_i[x]
 8              i ← i − 1
 9        i ← i + 1
10        Disk-Read(c_i[x])
11        if n[c_i[x]] = 2t − 1
12              BTree-Split-Child (x, i, c_i[x])
13              if k > key_i[x]
14                    i ← i + 1
15        BTree-Insert-Nonfull (c_i[x], k)
```

Исходное дерево

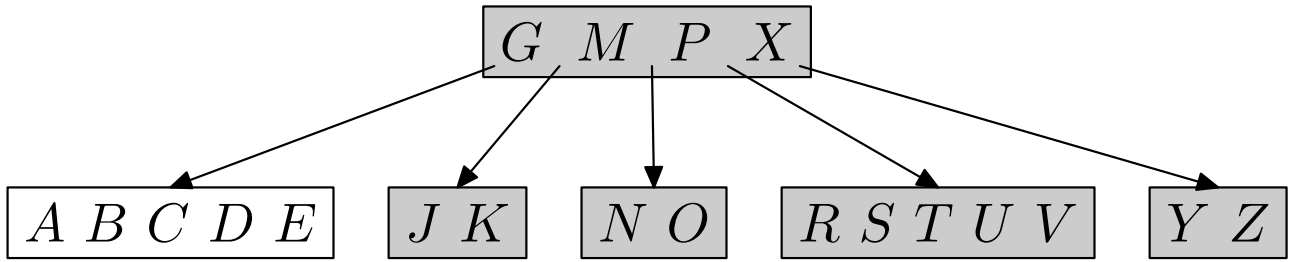


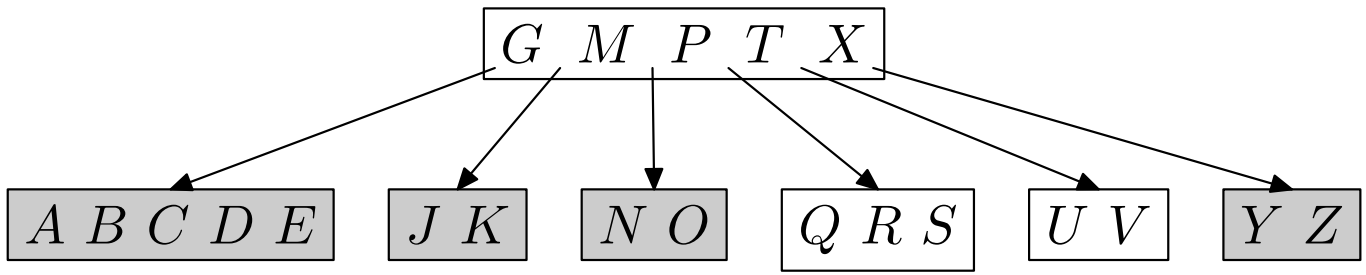
Вставка B



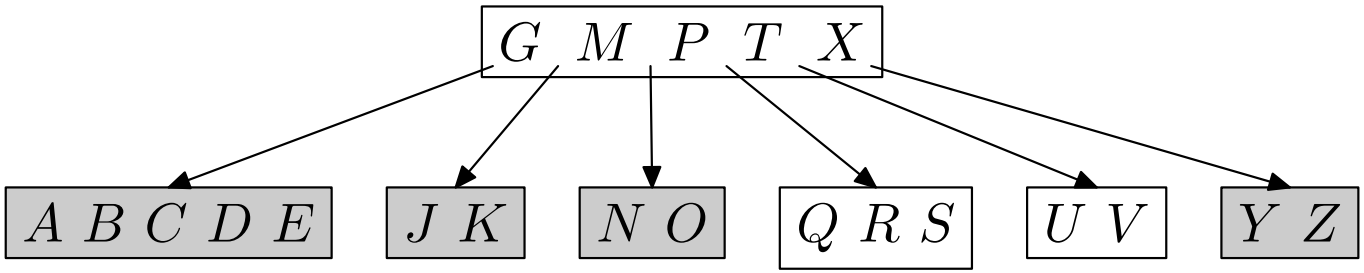


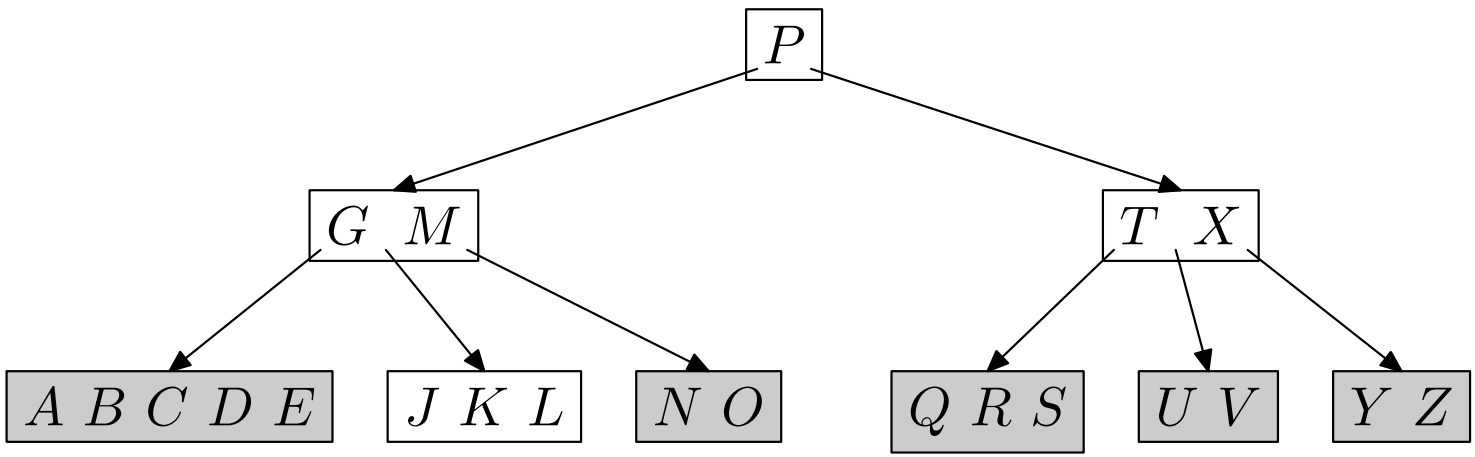
Вставка Q



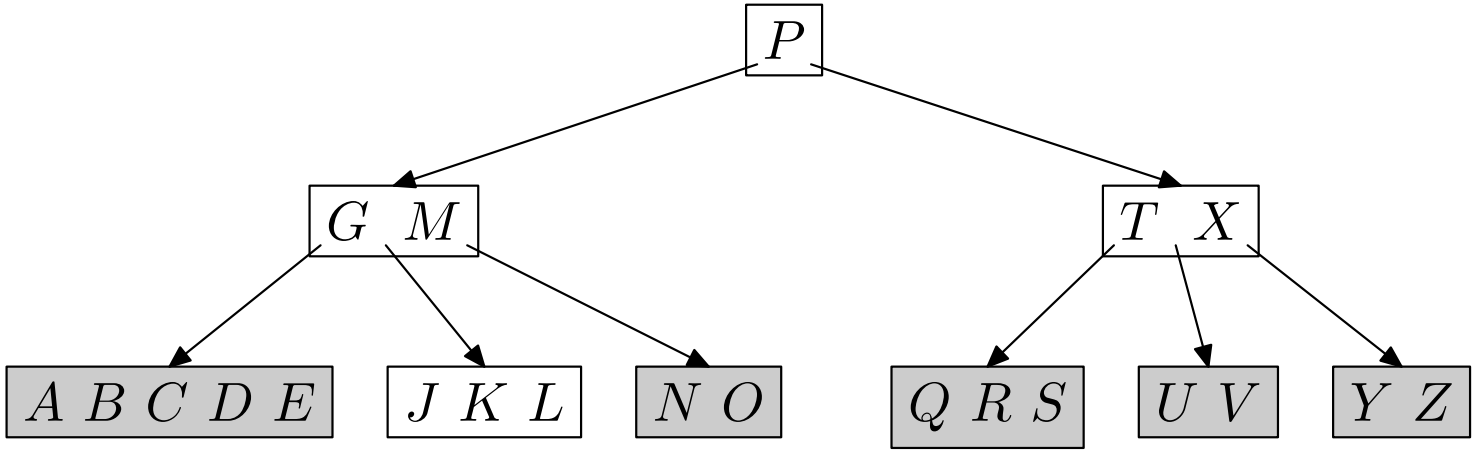


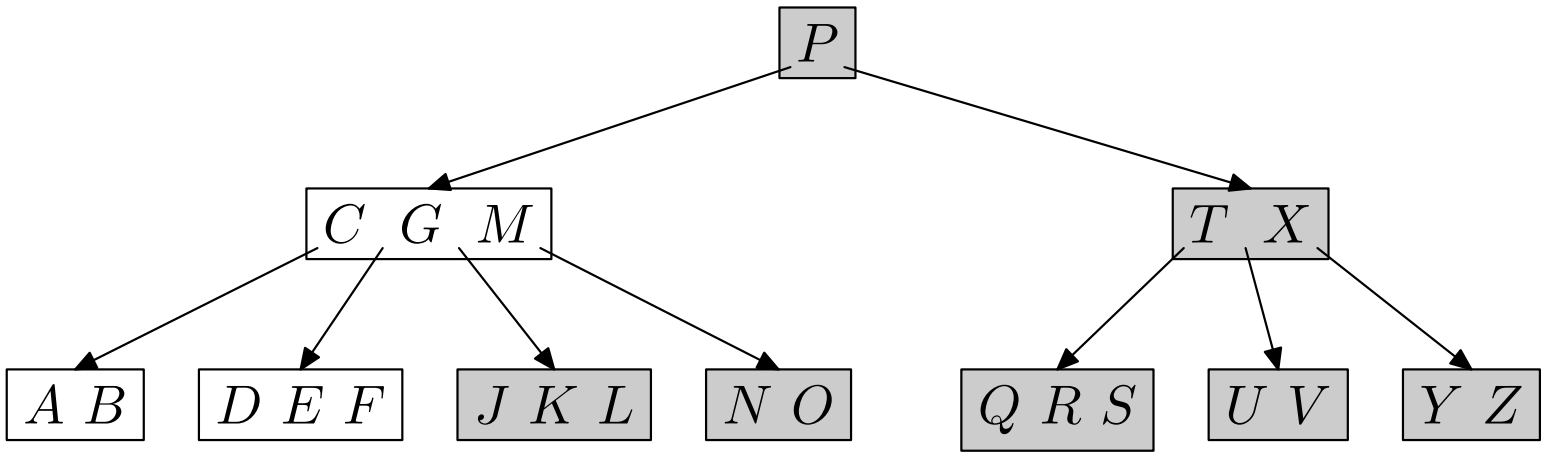
Вставка L





Вставка F





Раздел

Общая идея

- В общем аналогично вставке, но вместо разбиения узлов они либо сливаются, либо используется «перетекание» ключей из соседних узлов через родительский.
- Однако, технически более сложное: ключ может быть удалён откуда угодно, не только из листьев.
- Нужно обеспечить, чтобы везде на пути были бы узлы, из которых при необходимости можно удалить ключ, т.е. минимальное количество ключей t.

Удаление ключа k

При проходе дерева могут возникнуть следующие три случая:

1. Если ключ k находится в узле x и x — лист, то удаляем k из x.
2. Ключ k находится в узле x и x — внутренний узел: необходимо найти предшествующий или следующий ключ k', удалить его и заменить k в узле x на k'.
3. Ключ k отсутствует во внутреннем узле x, то находим корень поддерева $c_i[k]$, в котором находится ключ k и удостоверяемся, что он содержит как минимум t ключей. В противном случае модифицируем дерево таким образом, чтобы в узле $c_i[k]$ было бы t ключей.

Второй случай

Узлы y и z — дочерние по отношению к x, один предшествует ключу k, другой — следует за ним.

- y содержит как минимум t ключей, то находим k' — предшественника k в поддереве y, удаляем его и заменяем его значением позицию ключа k в x.
- Симметричная ситуация в z.
- x и y содержат по $t-1$ ключу: вносим k и все ключи z в y, освобождаем z, удаляем k из y.

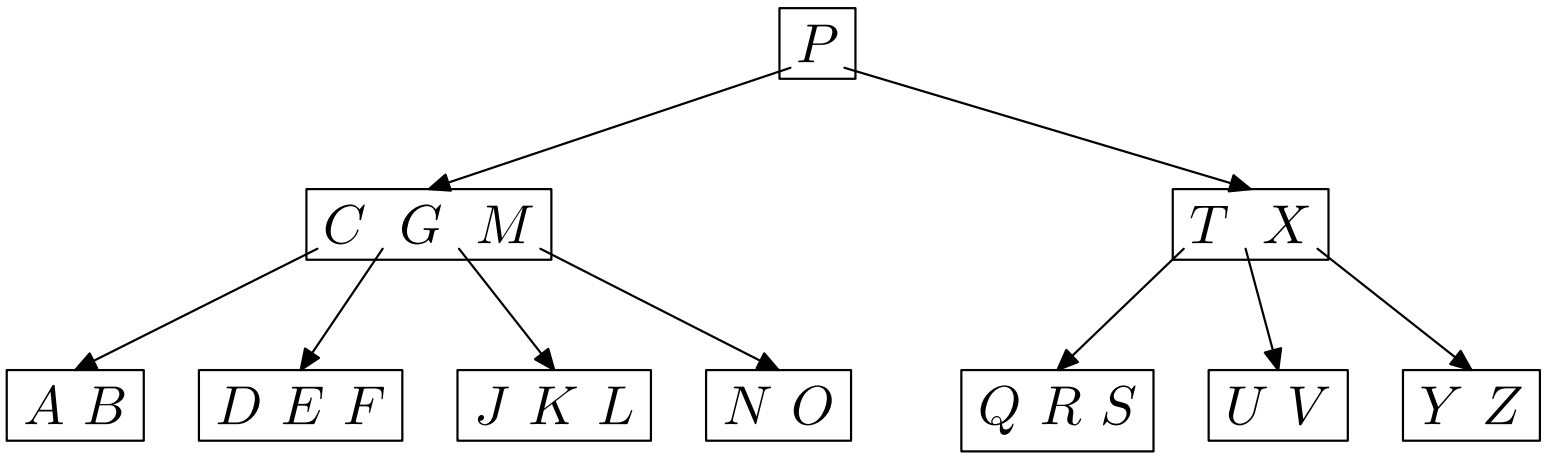
Третий случай

Узел $c_i[x]$ — корень поддерева, сдержащего k. Если в нём содержатся $t-1$ ключей, выполняем одно из следующих действий:

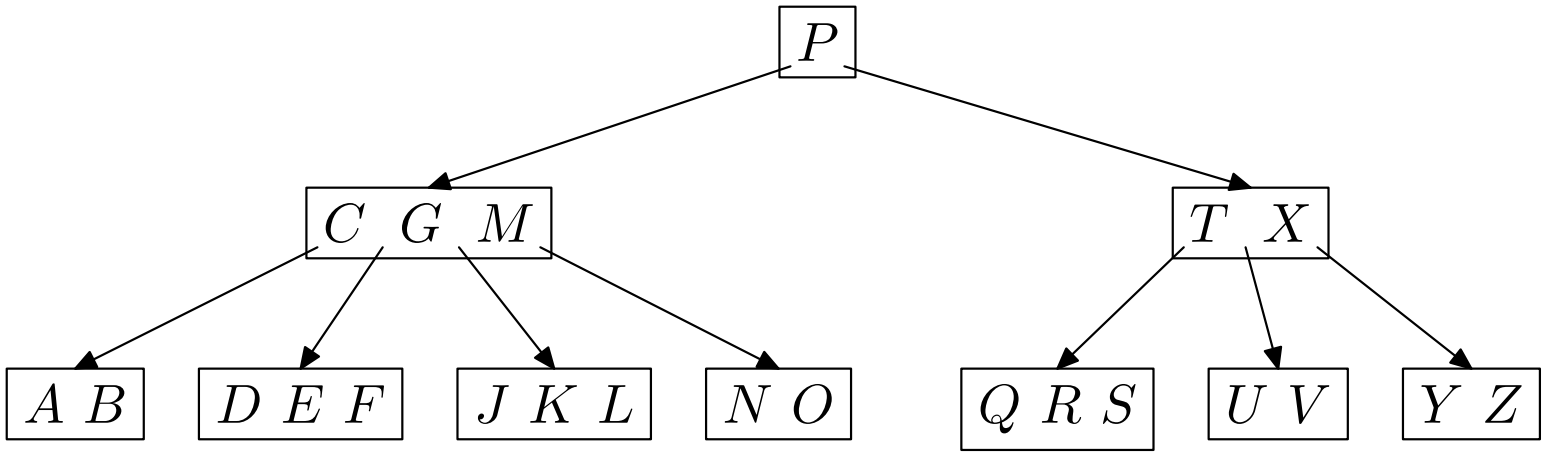
- Один из соседей $c_i[x]$ содержит в себе t ключей. Тогда передадим в $c_i[x]$ ключ-разделитель из x между соседями, а на место разделителя поместим максимальный или минимальный ключ из соседа, с соответствующей передачей поддеревьев.
- Все соседи $c_i[x]$ содераж в себе по $t-1$ ключей: объединим один из них с $c_i[x]$ используя ключ-разделитель из x в качестве медианы нового узла.

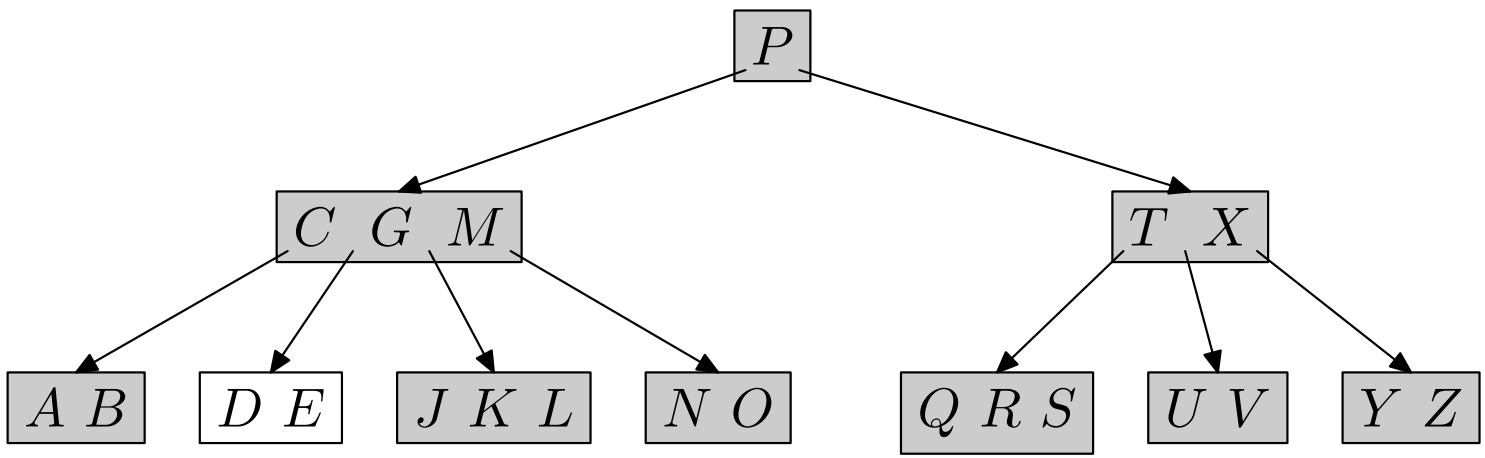
Затем рекурсивно удаляем ключ k из узла $c_i[x]$.

Исходное дерево

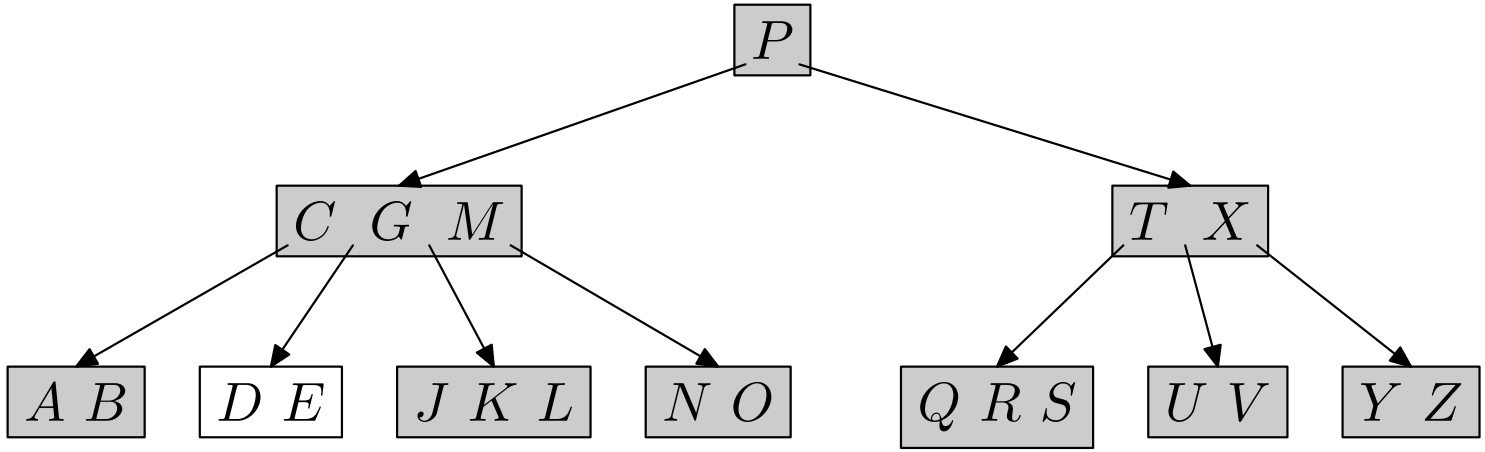


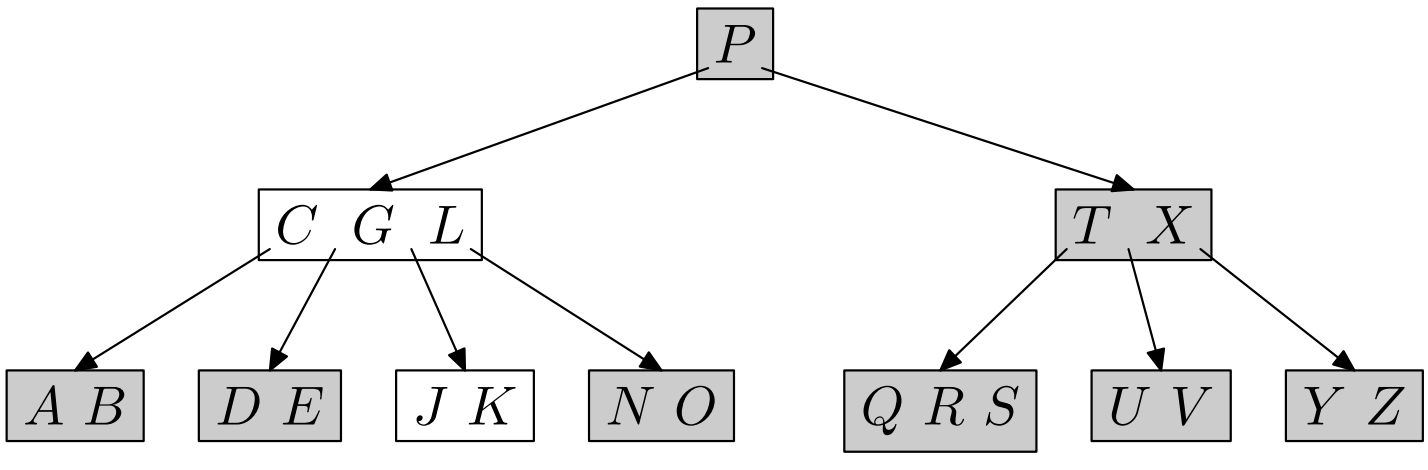
Удаление F



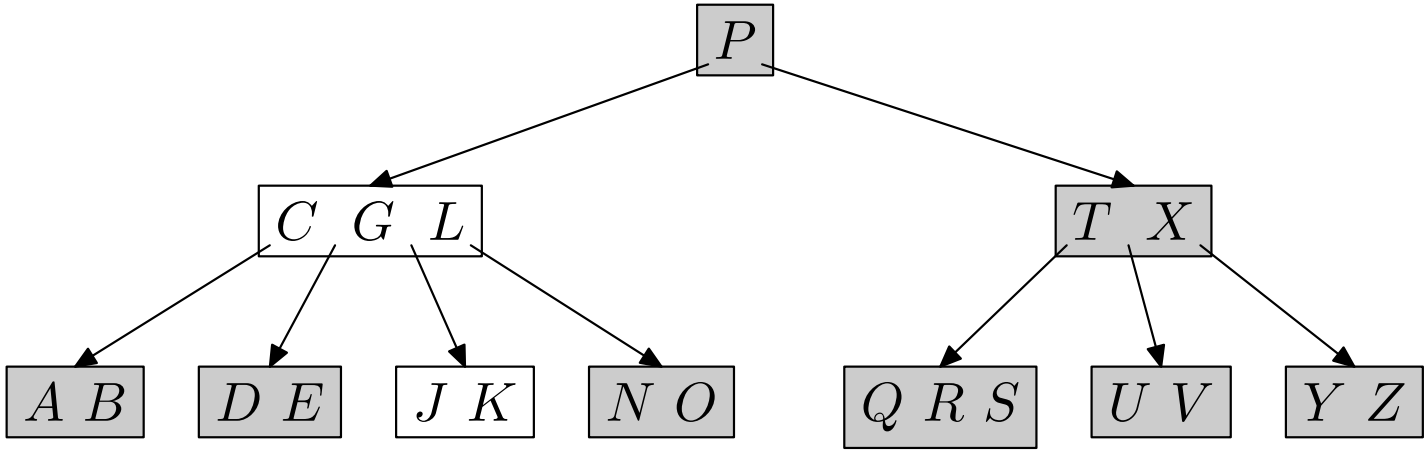


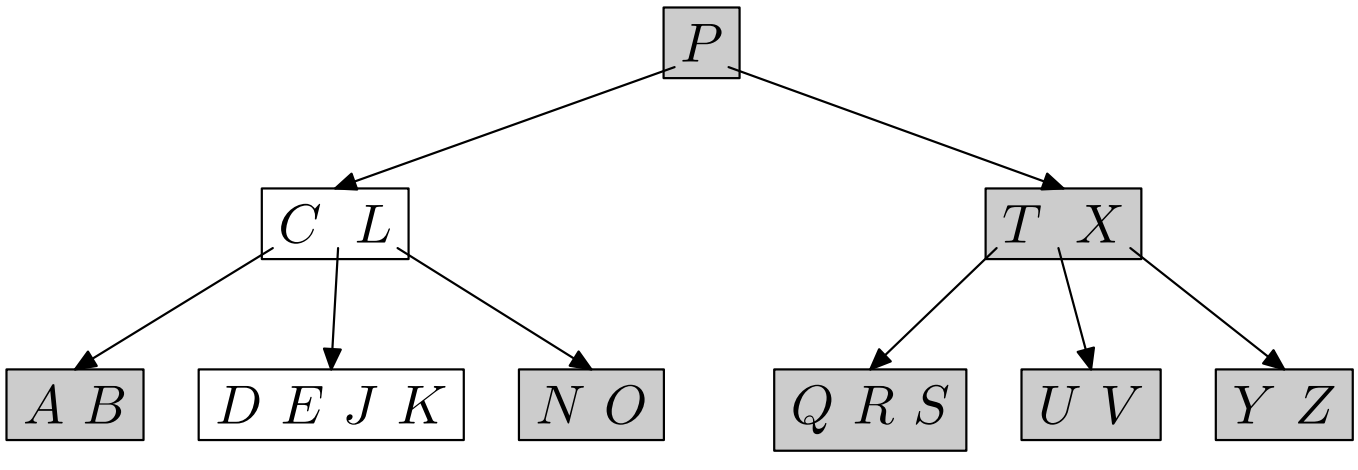
Удаление M



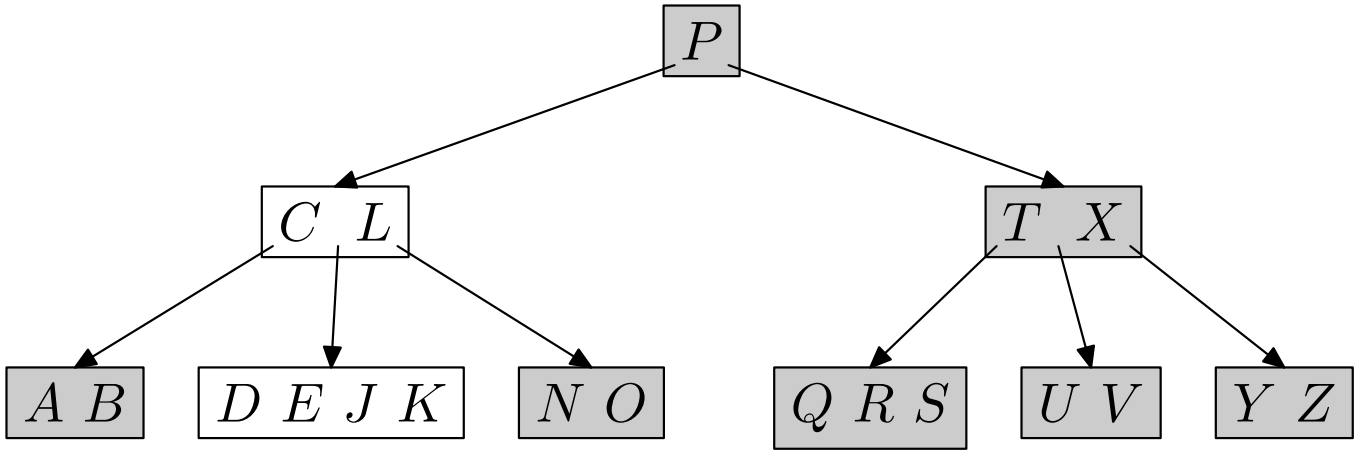


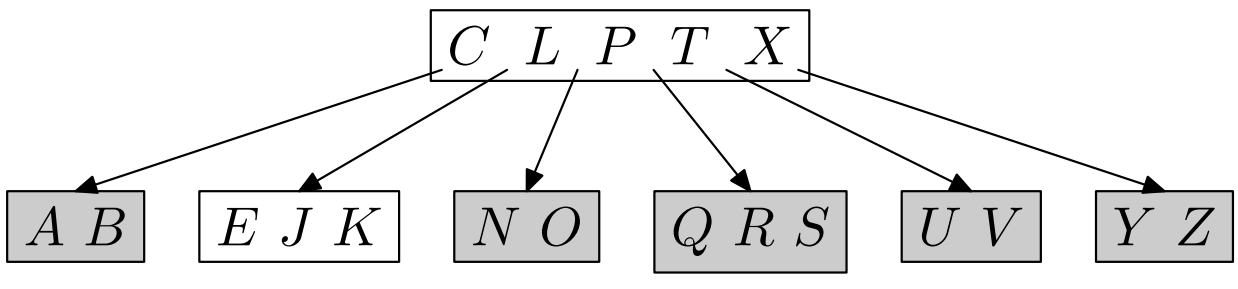
Удаление G





Удаление D





Удаление B