ALG 05

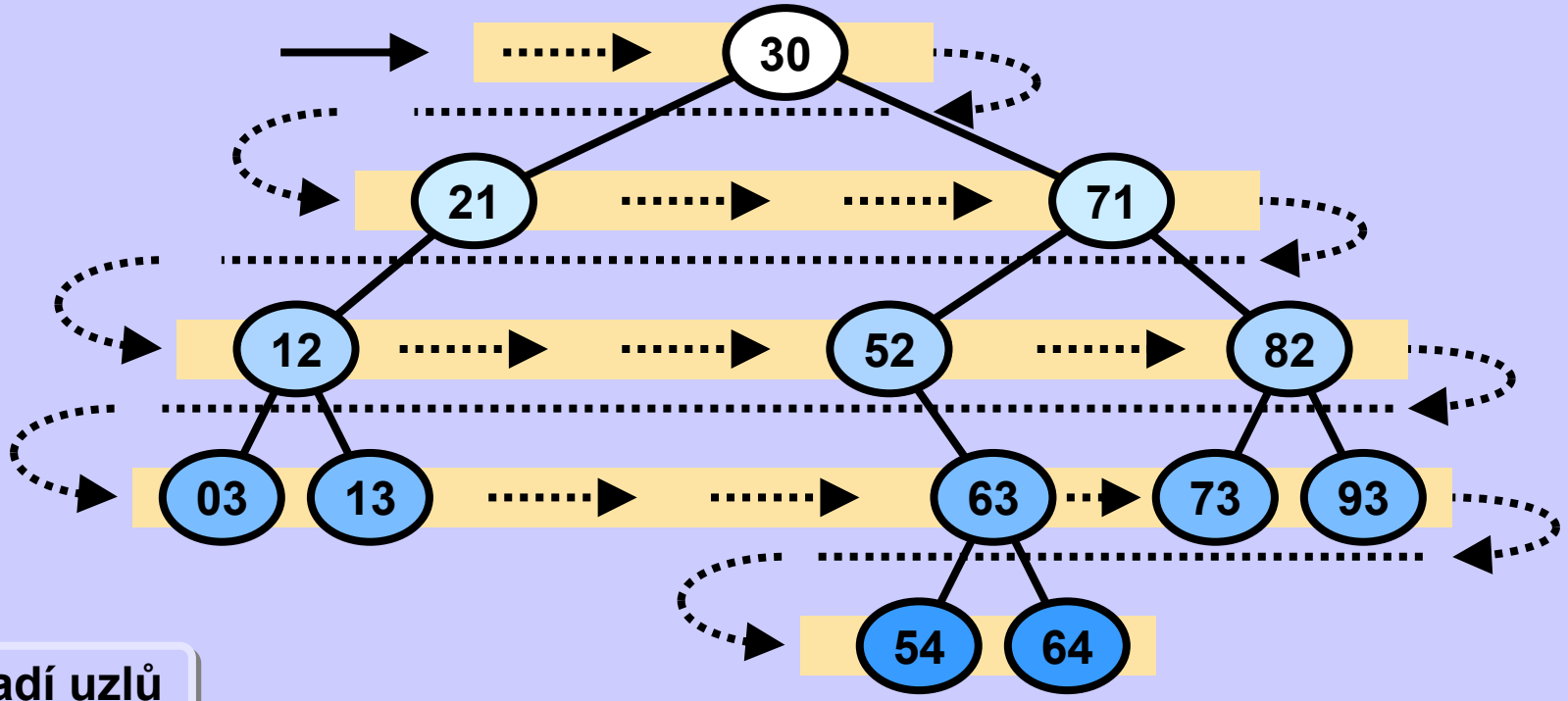
Průchod stromem do šířky
Fronta
  Operace Enqueue, Dequeue, Front, Empty....

Cyklická implementace fronty

B-strom
  Operace Find, Insert, Delete

Průchod stromem do šířky

Strom s naznačeným směrem průchodu

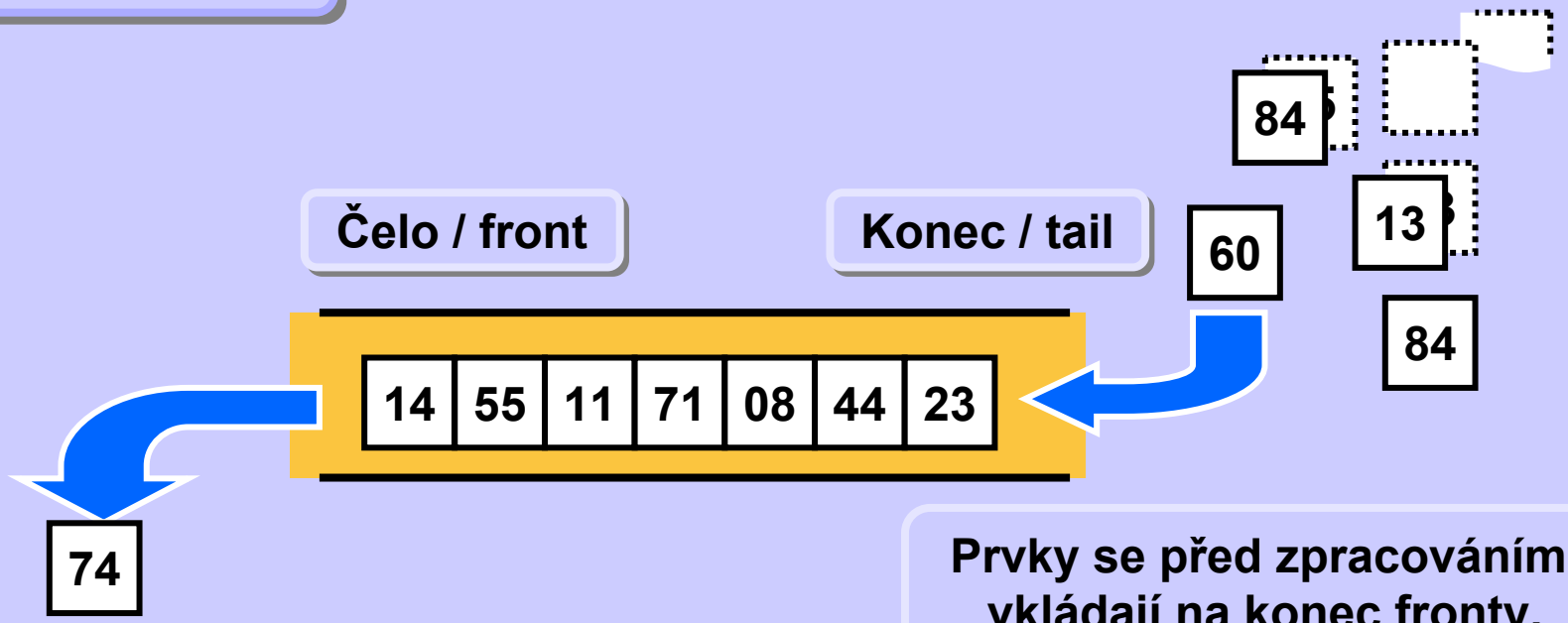


Pořadí uzlů

30	21	71	12	52	82	03	13	63	73	93	54	64

Struktura stromu ani rekurzivní přístup tento průchod nepodporují.

Fronta / queue



Prvky se před zpracováním vkládají na konec fronty.

Prvky se odebírají z čela fronty a pak se zpracovávají.

Operace

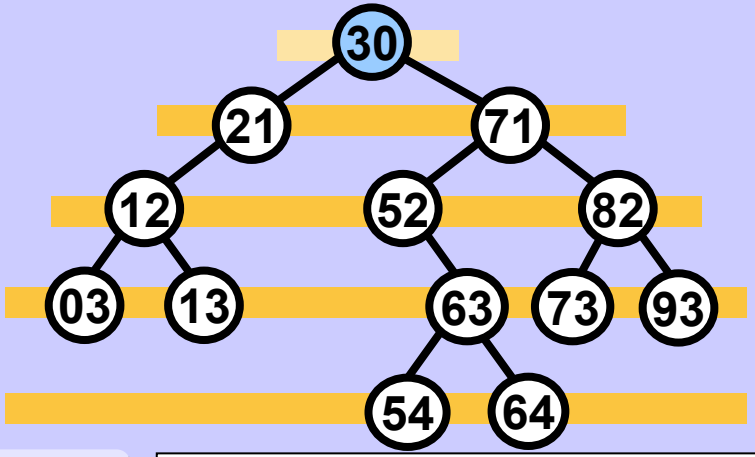
Vlož na konec	Enqueue / InsertLast / Push ...
Odeber ze začátku	Dequeue / delFront / Pop ...
Čti začátek	Front / Peek ...
Je prázdná?	Empty

Fronta

Jednoduchý příklad života fronty

Průchod stromem do šířky

Inicializace



Vytvoř prázdnou frontu

Do fronty vlož kořen stromu

Výstup

2.

Čelo

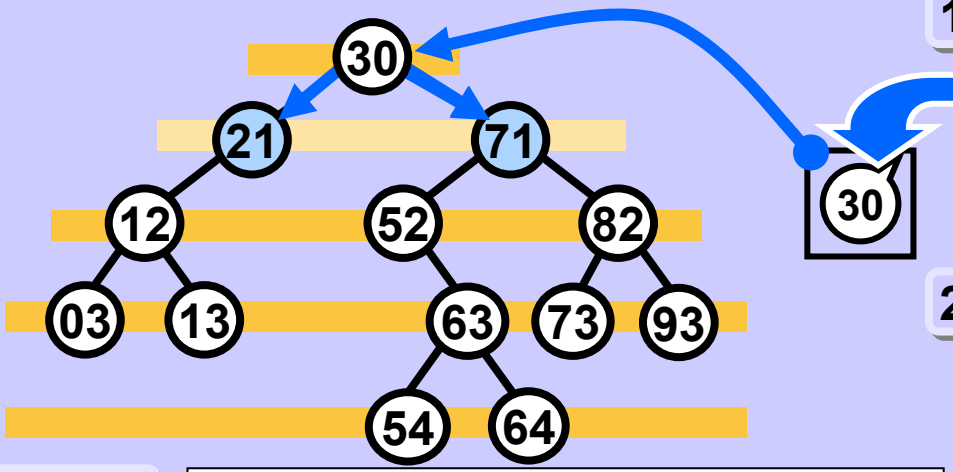
Konec

Hlavní cyklus

Dokud není fronta prázdná, opakuj:

1. Odeber první uzel z fronty a zpracuj ho.
2. Do fronty vlož jeho potomky, pokud existují.

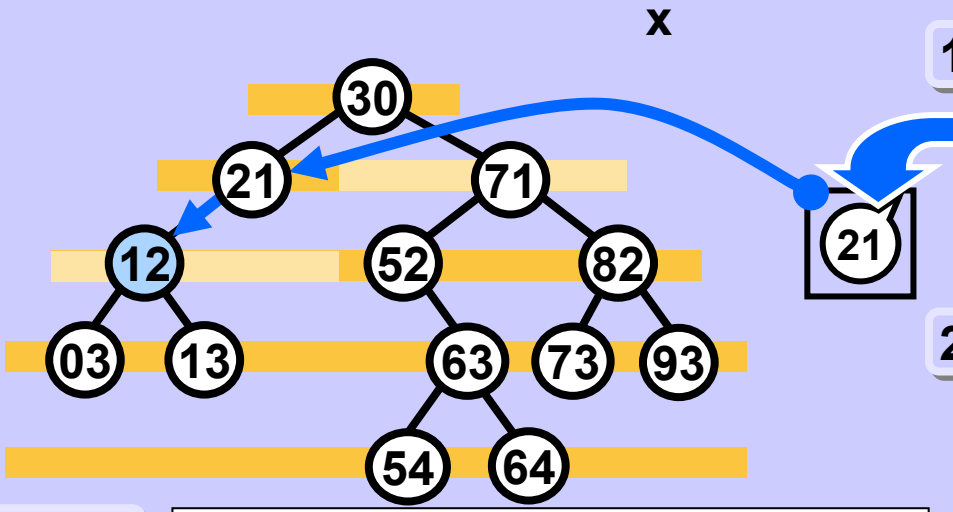
Průchod stromem do šířky



1. x = Odeber(), tisk (x.key).

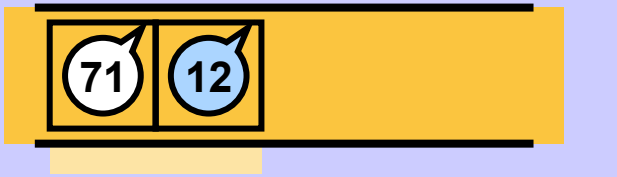
2. Vlož(x.left), vlož(x.right). *)

Výstup | 30



1. x = Odeber(), tisk(x.key).

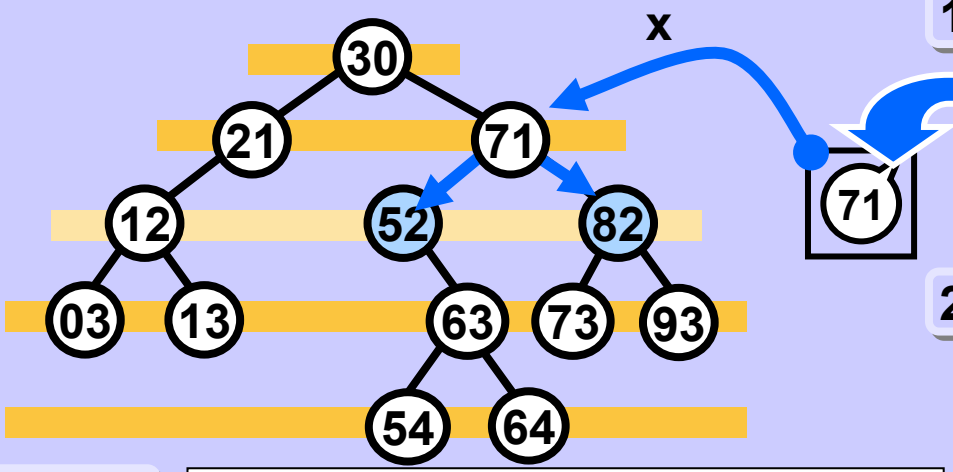
2. Vlož(x.left), vlož(x.right). *)



Výstup | 30 21

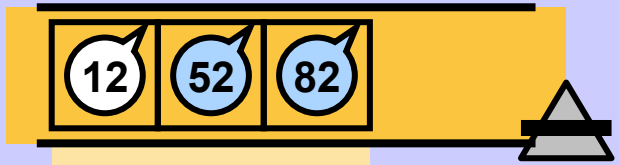
*) pokud existuje

Průchod stromem do šířky

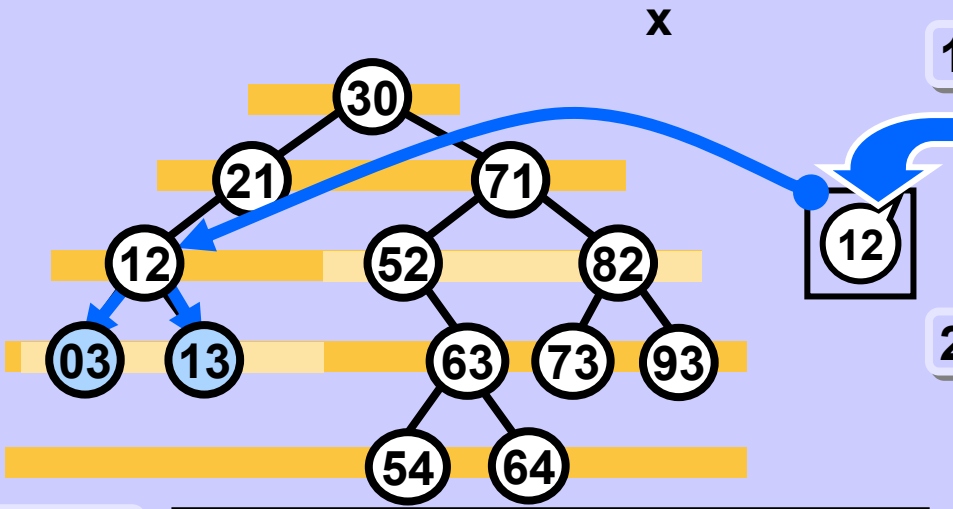


1. x = Odeber(), tisk (x.key).

2. Vlož(x.left), vlož(x.right). *)



Výstup: 30 21 71



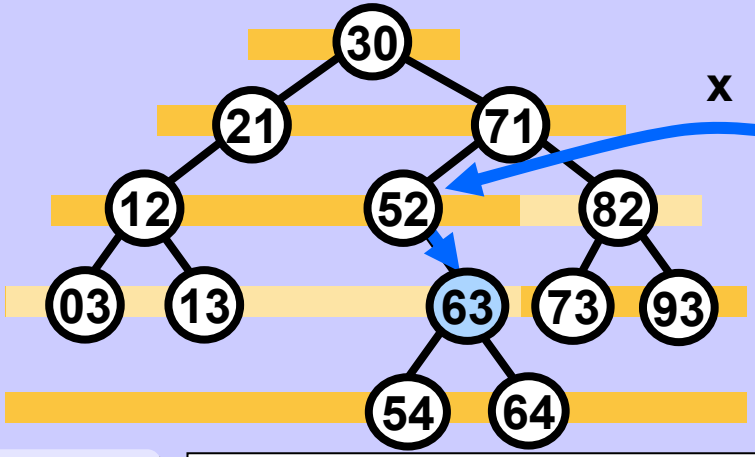
1. x = Odeber(), tisk(x.key).

2. Vlož(x.left), vlož(x.right). *)

Výstup: 30 21 71 12

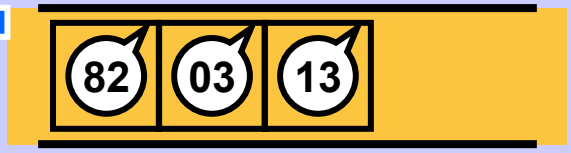
*) pokud existuje

Průchod stromem do šířky

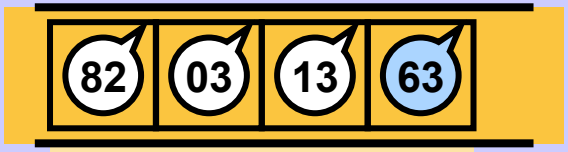


x

1. x = Odeber(), tisk (x.key).

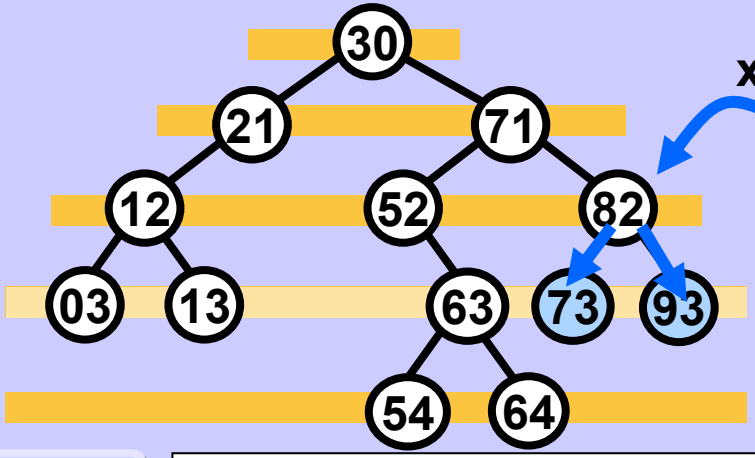


2. Vlož(x.left), vlož(x.right). *)



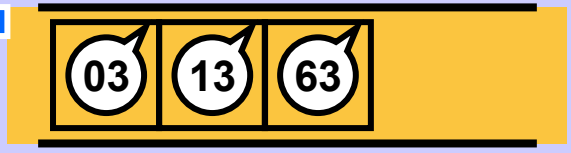
Výstup: 30 21 71 12 52

*) pokud existuje

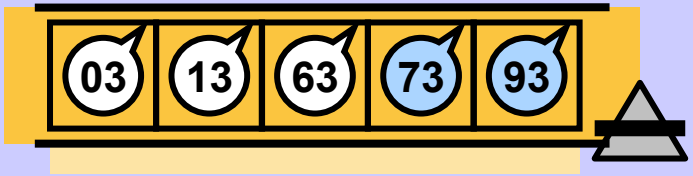


x

1. x = Odeber(), tisk(x.key).

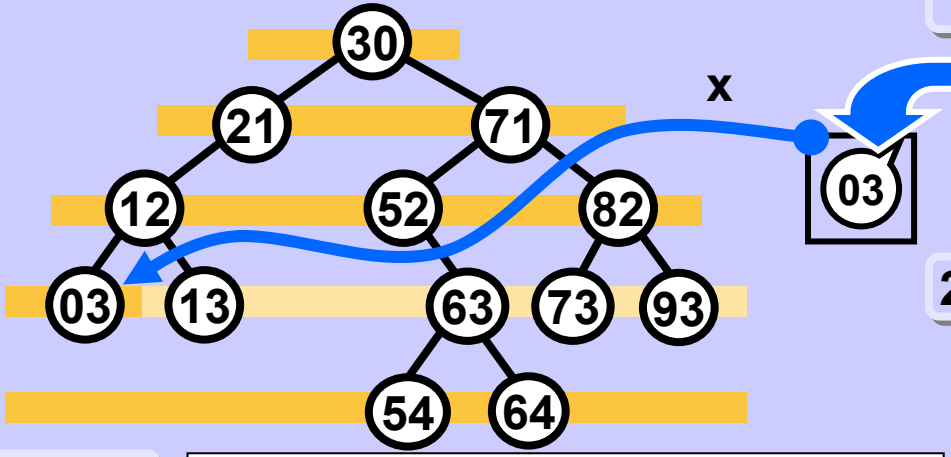


2. Vlož(x.left), vlož(x.right). *)



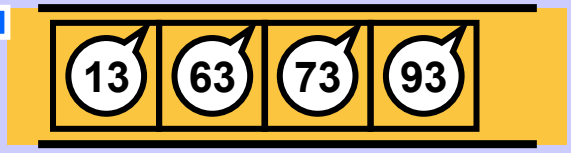
Výstup: 30 21 71 12 52 82

Průchod stromem do šířky

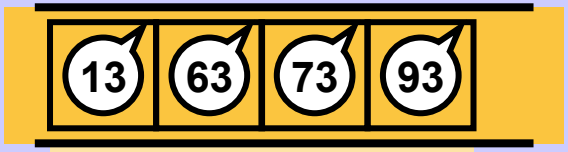


x

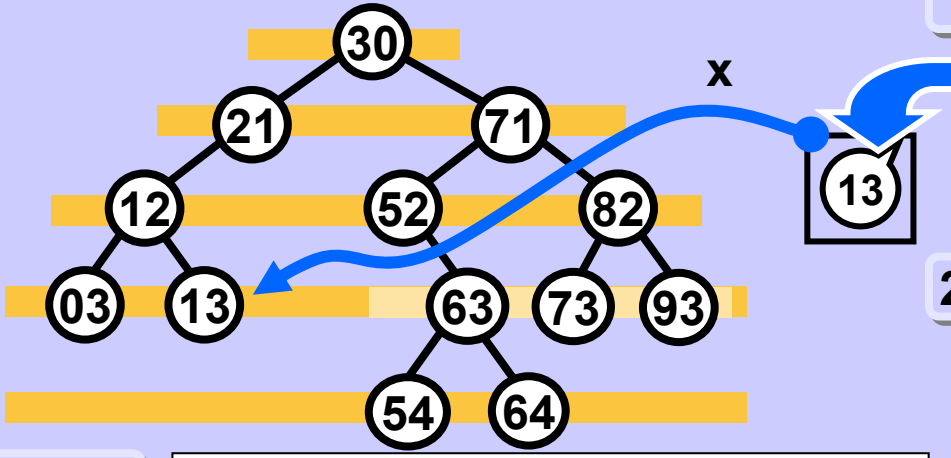
1. x = Odeber(), tisk (x.key).



2. Vlož(x.left), vlož(x.right). *)

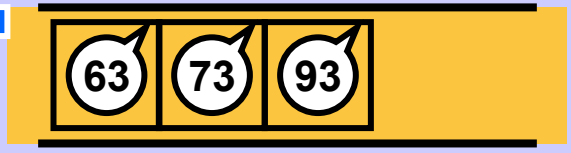


Výstup: 30 21 71 12 52 82 03

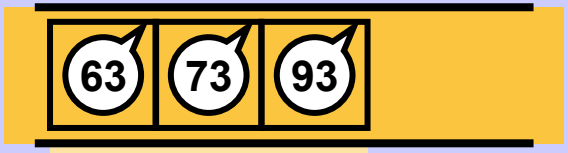


x

1. x = Odeber(), tisk(x.key).



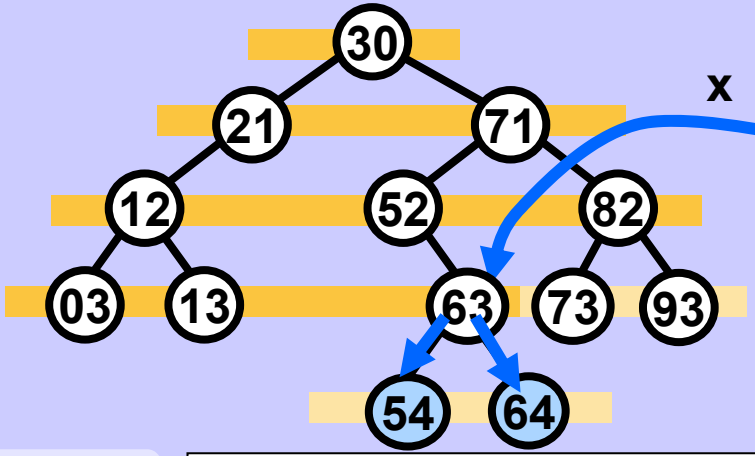
2. Vlož(x.left), vlož(x.right). *)



Výstup: 30 21 71 12 52 82 03 13

*) pokud existuje

Průchod stromem do šířky

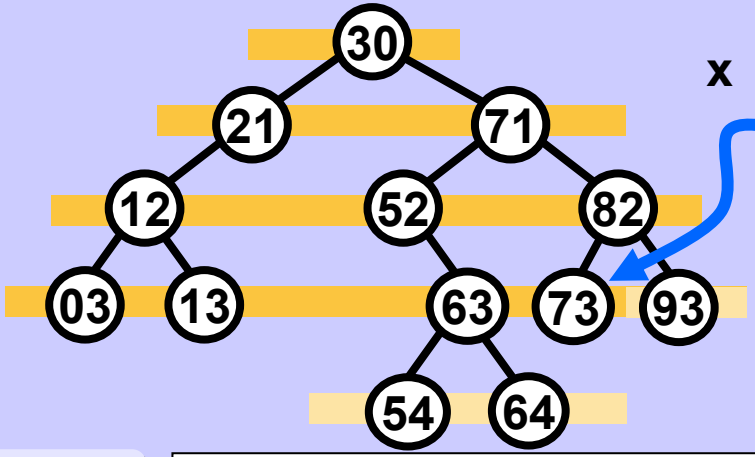


x

1. x = Odeber(), tisk (x.key).

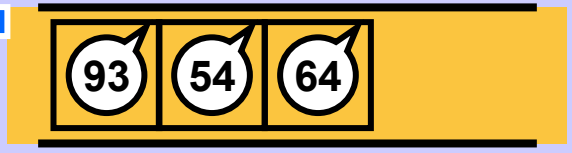
2. Vlož(x.left), vlož(x.right). *)

Výstup: 30 21 71 12 52 82 03 13 63

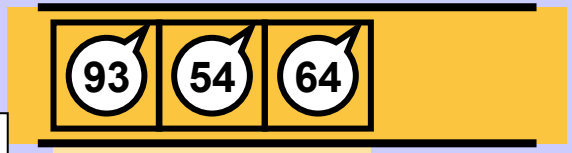


x

1. x = Odeber(), tisk(x.key).



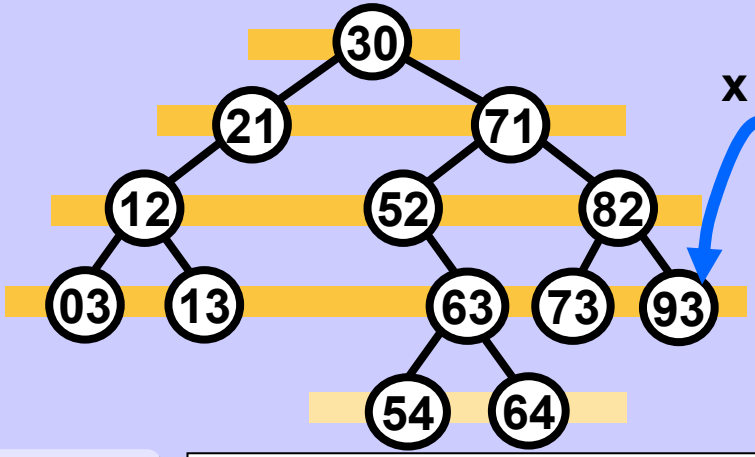
2. Vlož(x.left), vlož(x.right). *)



Výstup: 30 21 71 12 52 82 03 13 63 73

*) pokud existuje

Průchod stromem do šířky

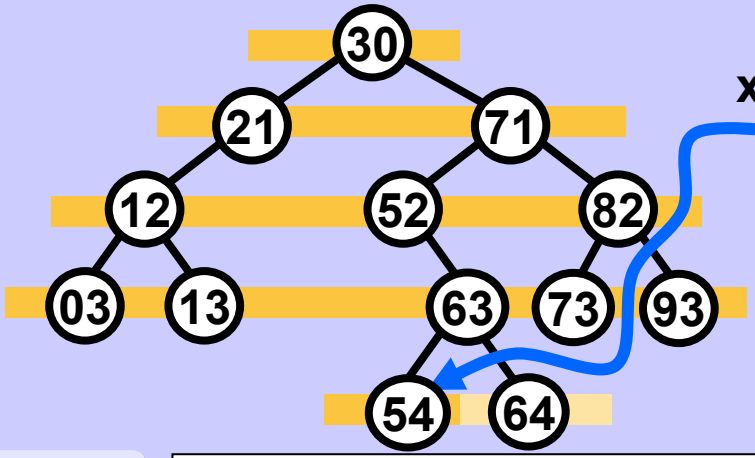


x

1. x = Odeber(), tisk (x.key).

2. Vlož(x.left), vlož(x.right). *)

Výstup: 30 21 71 12 52 82 03 13 63 73 93



x

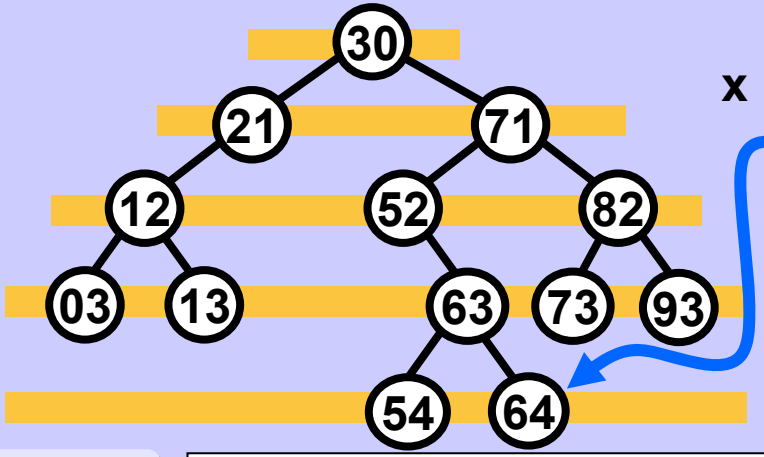
1. x = Odeber(), tisk(x.key).

2. Vlož(x.left), vlož(x.right). *)

Výstup: 30 21 71 12 52 82 03 13 63 73 93 54

*) pokud existuje

Průchod stromem do šířky



x

1. x = Odeber(), tisk (x.key).

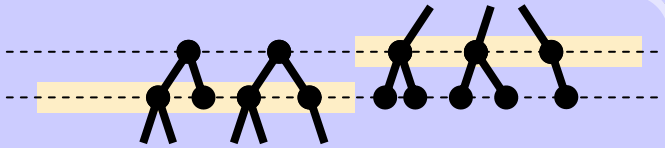
2. Vlož(x.left), vlož(x.right). *)

*) pokud existuje

Výstup: 30 21 71 12 52 82 03 13 63 73 93 54 64

Fronta je prázdná, průchod stromem končí.

V neprázdné **frontě** jsou vždy právě
-- některé (třeba všechny) uzly jednoho patra
-- a všichni potomci těch uzlů tohoto patra, které už nejsou ve frontě.



Někdy jsou ve frontě přesně všechny uzly jednoho patra. Viz výše.

Průchod stromem do šířky

```
void listBreadth (Node node) {
    if (node == null) return;
    Queue q = new Queue();          // init
    q.Enqueue(node);                // root into queue
    while (!q.Empty()) {
        node = q.Dequeue();
        print(node.key);            // process node
        if (node.left  != null) q.Enqueue(node.left);
        if (node.right != null) q.Enqueue(node.right);
    }
}
```

Cyklická implementace fronty polem

Prázdná fronta
v poli pevné délky

Vlož 24, 11, 90, 43, 70.

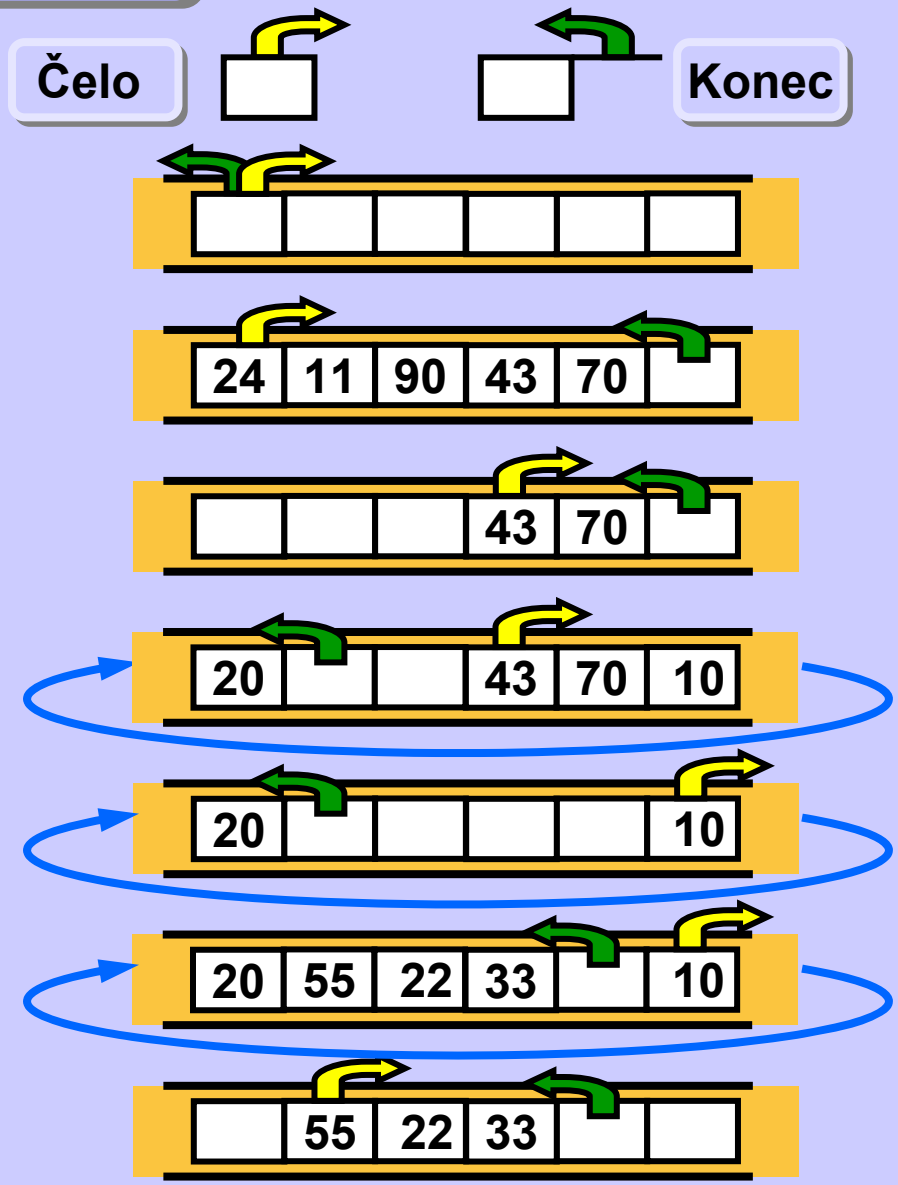
Odeber, odeber, odeber.

Vlož 10, 20.

Odeber, odeber.

Vlož 55, 22, 33.

Odeber, odeber.



Cyklická implementace fronty polem

Index/ukazatel konce fronty ukazuje na první volnou pozici za posledním prvkem fronty. Index/ukazatel čela fronty ukazuje na první obsazenou pozici. Pokud oba ukazují tamtéž, fronta je prázdná.

```
class Queue {
  Node q [];
  int size;
  int front;
  int tail;

 Queue(int qsize) {
   size = qsize;
   q = new Node[size];
   front = 0;
   tail = 0;
   }

 boolean Empty() {
   return (tail==front);
 }
```

```
void Enqueue(Node node) {
  if ((tail+1 == front) ||
      (tail-front == size-1))
      ... // queue full, fix it

  q[t++] = node;
  if (tail==size) tail=0;
}

Node Dequeue() {
  Node n = q[front++];
  if (front==size) front=0;
     return n;
}

} // end of Queue
```

B-strom

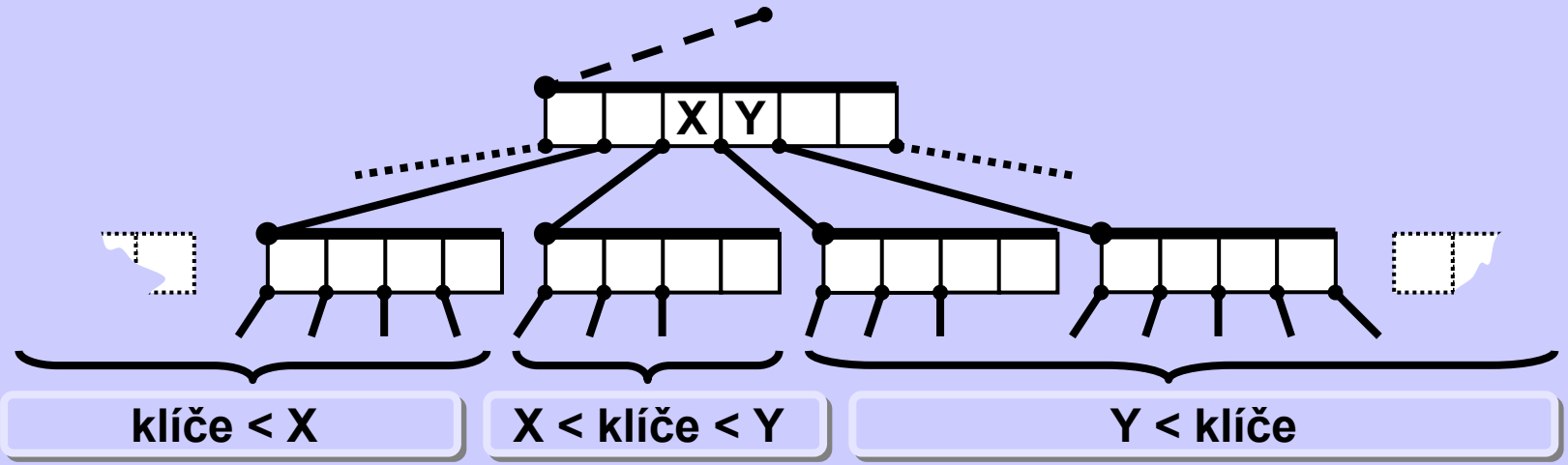
Všechny cesty z kořene do listu jsou stejně dlouhé
tj. B-strom je ideálně vyvážený.

Klíče jsou v uzlu seřazené.

Fixní k > 1 pro celý strom určuje velikost všech uzlů.

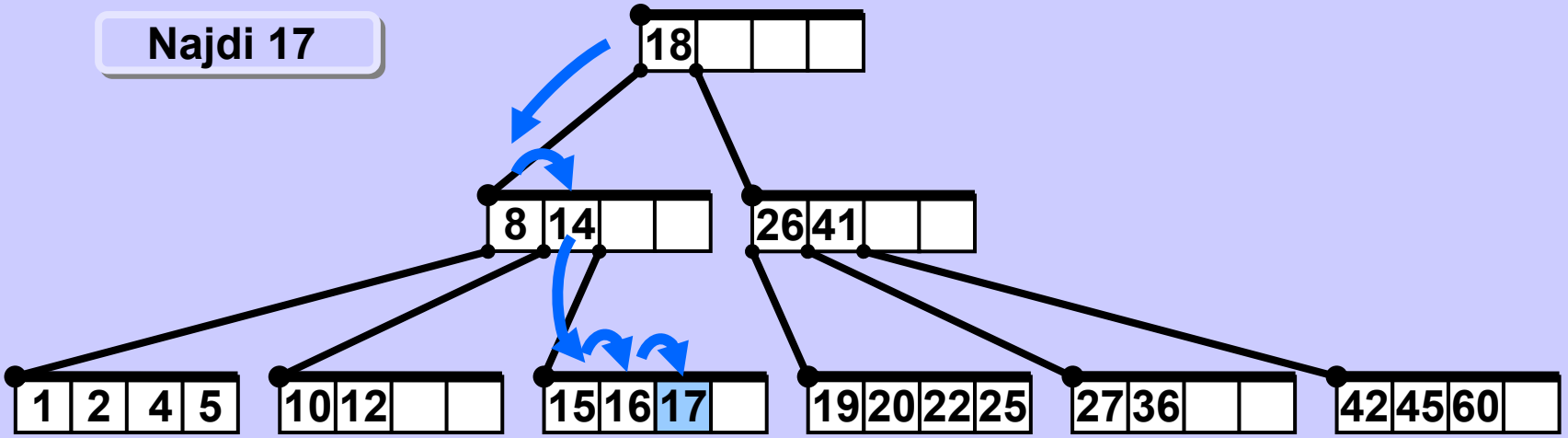
Každý uzel kromě kořene má nejméně k a nejvýše 2k klíčů,
každý vnitřní uzel tedy má nejméně k+1 a nejvýše 2k+1 potomků.

Kořen může mít libovolný počet klíčů. Není-li zároveň listem,
má alespoň 2 potomky.



B-strom -- Find

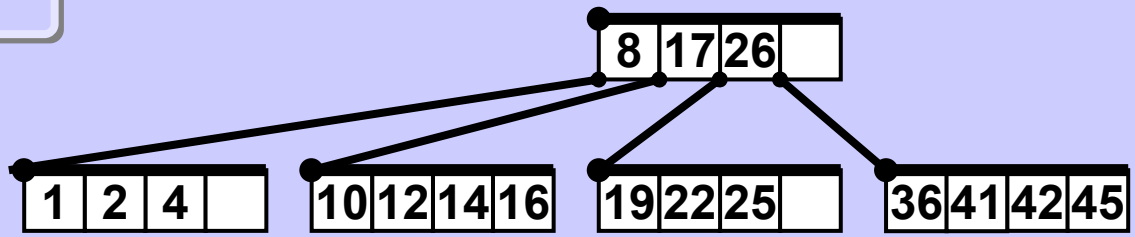
Najdi 17



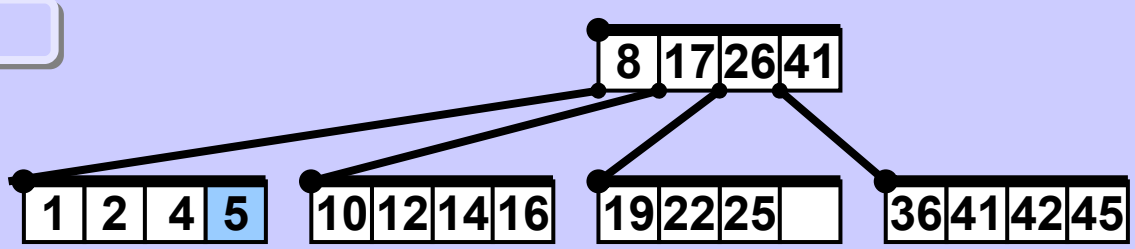
V uzlu se vyhledává sekvenčně.
Pokud uzel není listem a klíč v něm není,
hledání pokračuje v odpovídajícím potomku.
Pokud uzel je listem a klíč v něm není, nenalezeno.

B-strom -- Insert

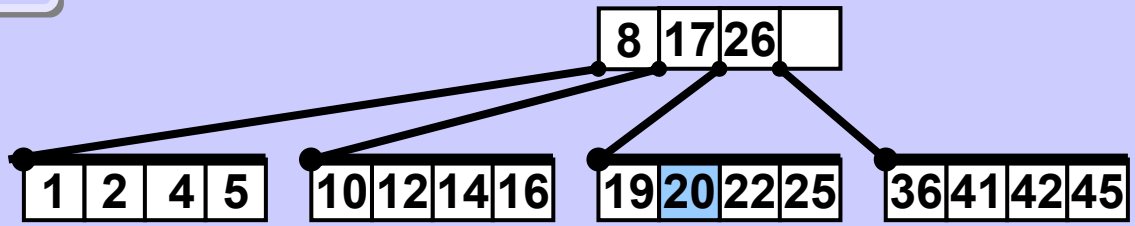
B-strom



Vlož 5

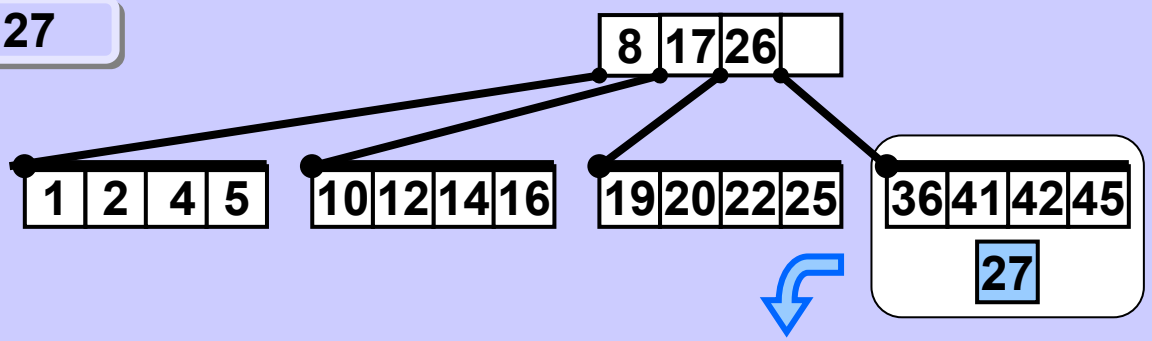


Vlož 20

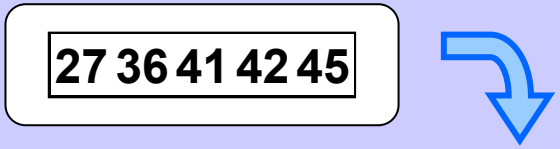


B-strom -- Insert

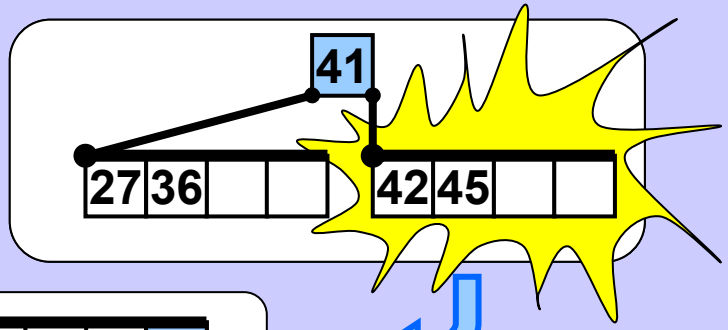
Insert 27



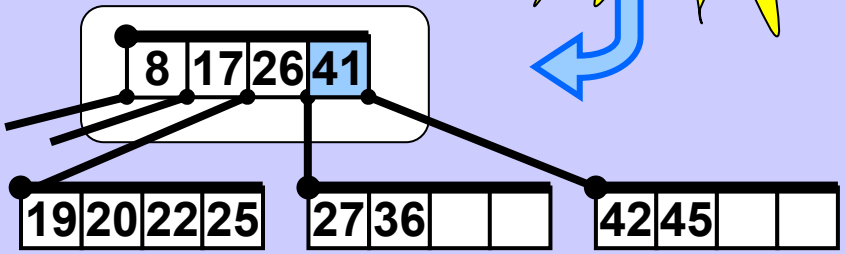
Seřaď mimo strom.



Vyber medián, vyvoř nový uzel, přesuň do něj hodnoty větší než medián.



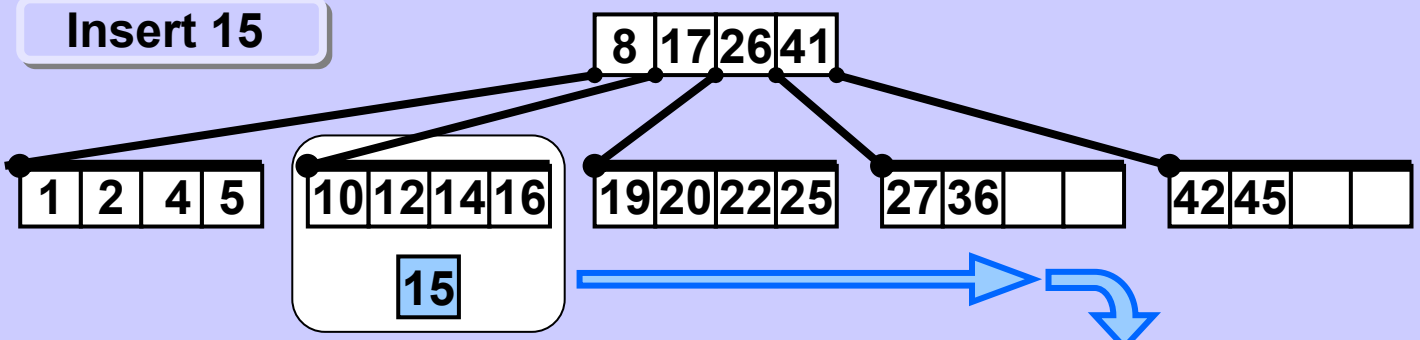
Medián zkus vložit do rodiče.



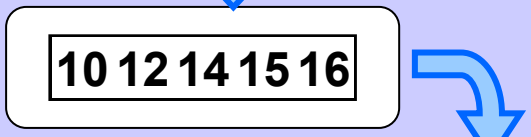
Zdařilo se.

B-strom -- Insert

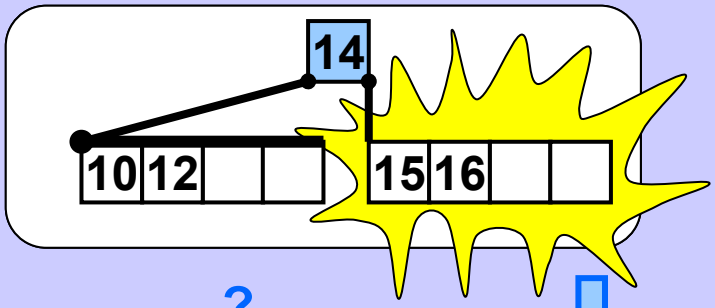
Insert 15



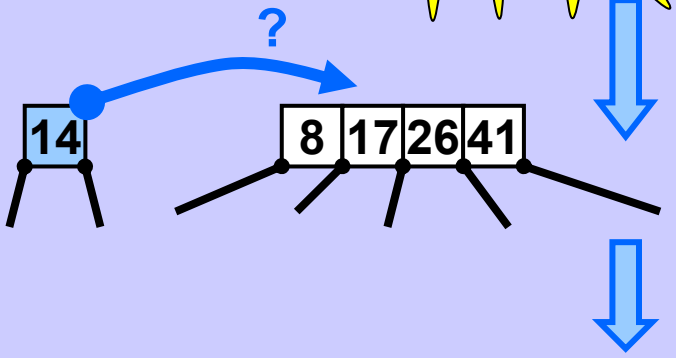
Seřaď mimo strom.



Vyber medián,
vyvoř nový uzel,
přesuň do něj hodnoty
větší než medián.



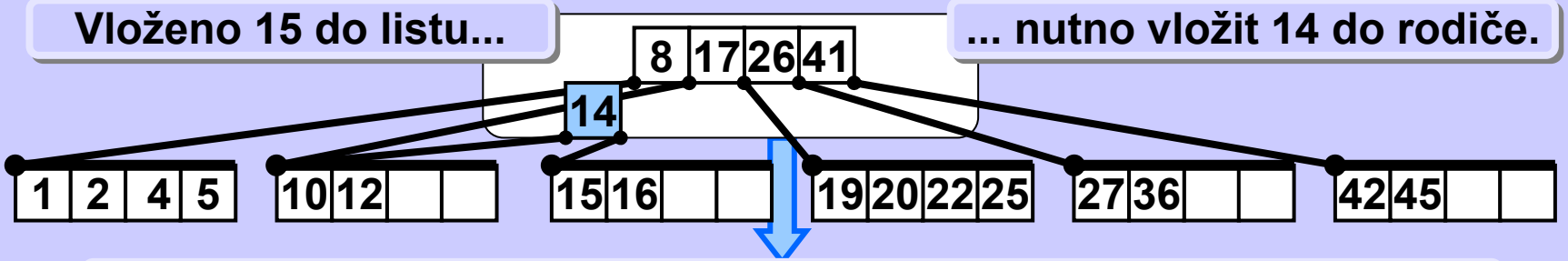
Medián zkus vložit
do rodiče.



B-strom -- Insert

Vloženo 15 do listu...

... nutno vložit 14 do rodiče.

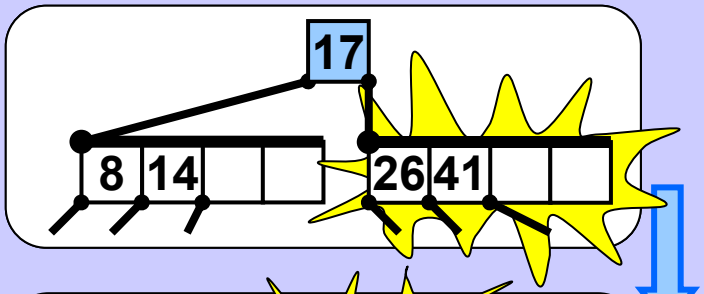


Rodič je zaplněn – Analogický další postup směrem ke kořeni.

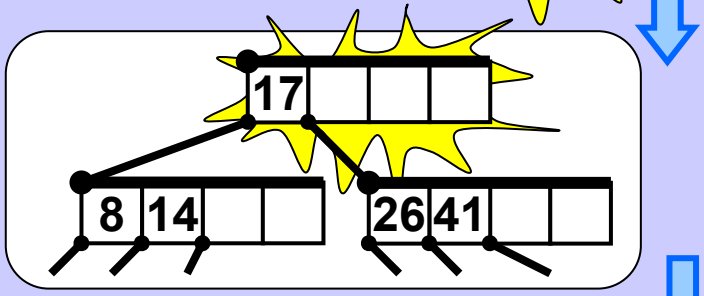
Seřaď mimo strom.

8 14 17 26 41

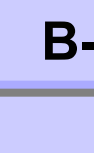
Vyber medián,
vyvoř nový uzel,
přesuň do něj hodnoty
větší než medián.



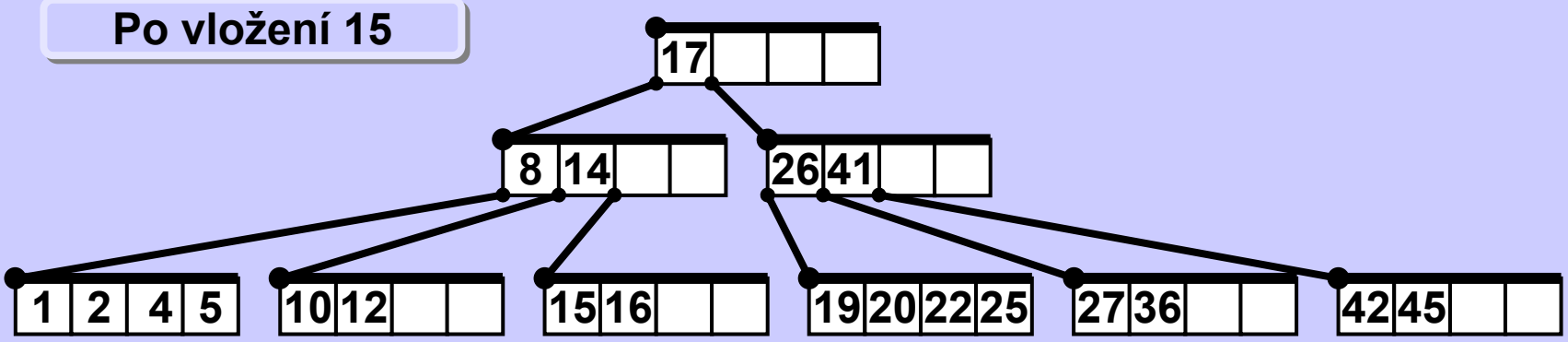
Medián nelze vložit do
rodiče, žádný není, tedy
se zřídí nový kořen.



B-strom -- Insert



Po vložení 15

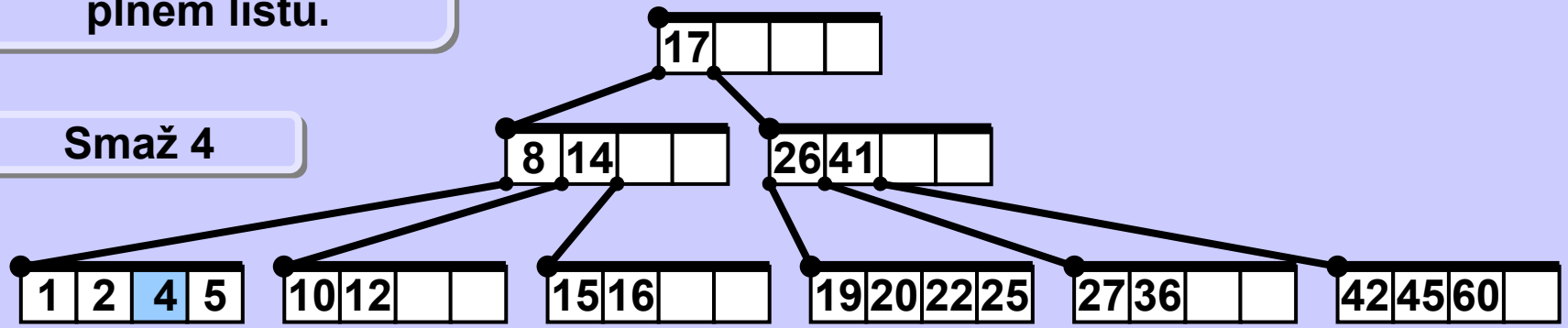


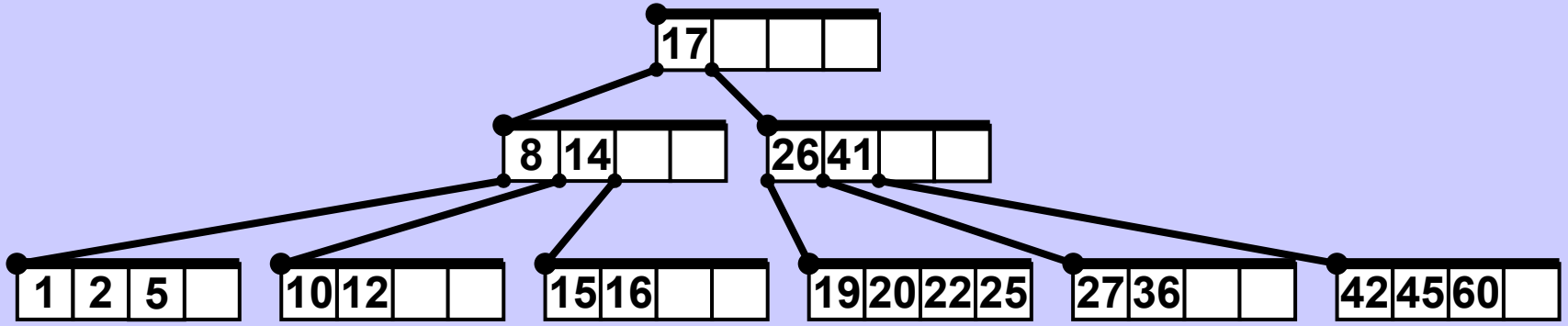
V každém patře přibyl jeden uzel, kromě toho přibyl nový kořen, strom ale roste směrem "vzhůru", zůstává ideálně vyvážený.

B-strom -- Delete

Mazání v dostatčně plném listu.

Smaž 4



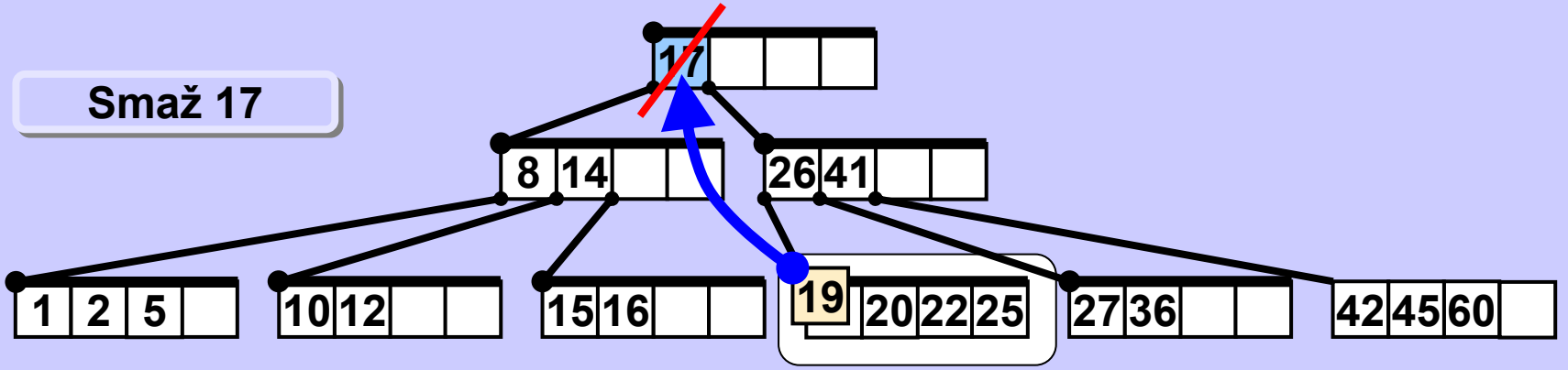


B-strom -- Delete

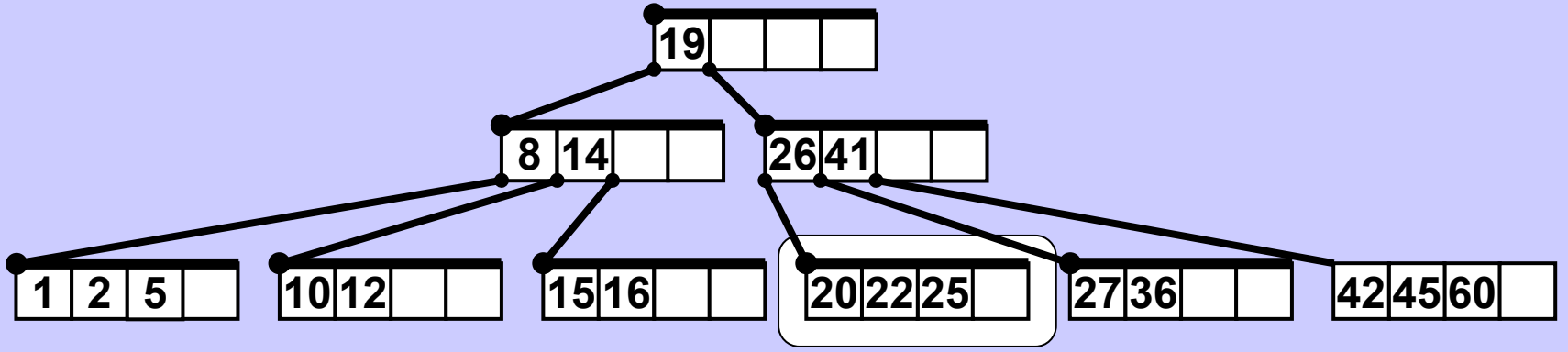
Mazání ve vnitřním uzlu

Smazaný klíč se nahradí nejbližším větším klíčem, podobně jako v BVS.

Smaž 17



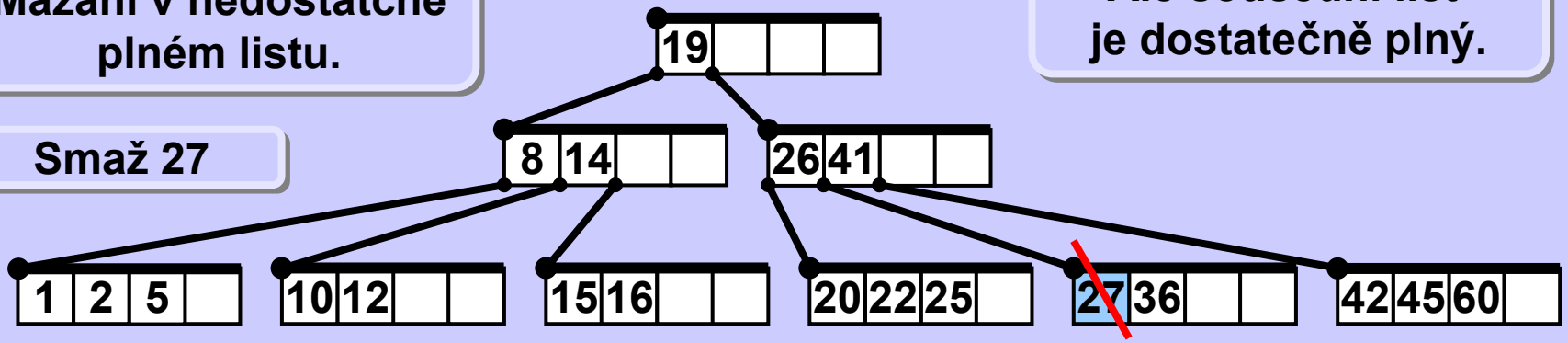
Nejbližší větší(menší) klíč je vždy v B-stromu v listu,
má-li tento list dostatečný počet klíčů, jsme hotovi.



B-strom -- Delete

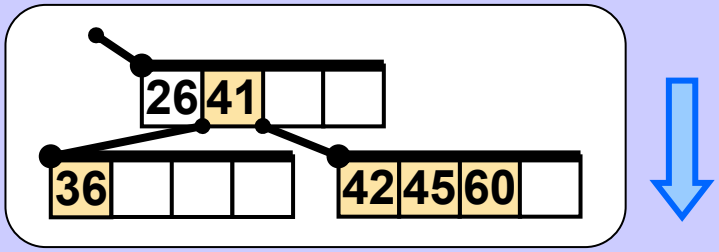
Mazání v nedostatčně plném listu.

Smaž 27



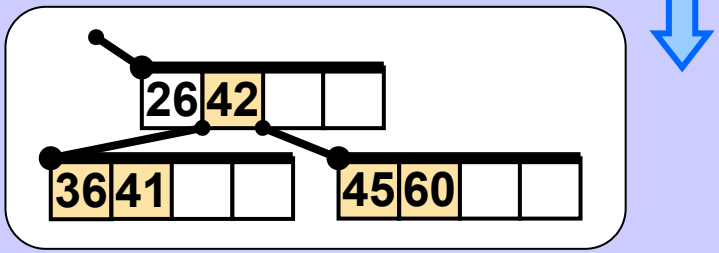
Ale sousední list je dostatečně plný.

Sjednoť klíče s klíči v sousedním listu a s dělícím klíčem v rodiči a seřaď.



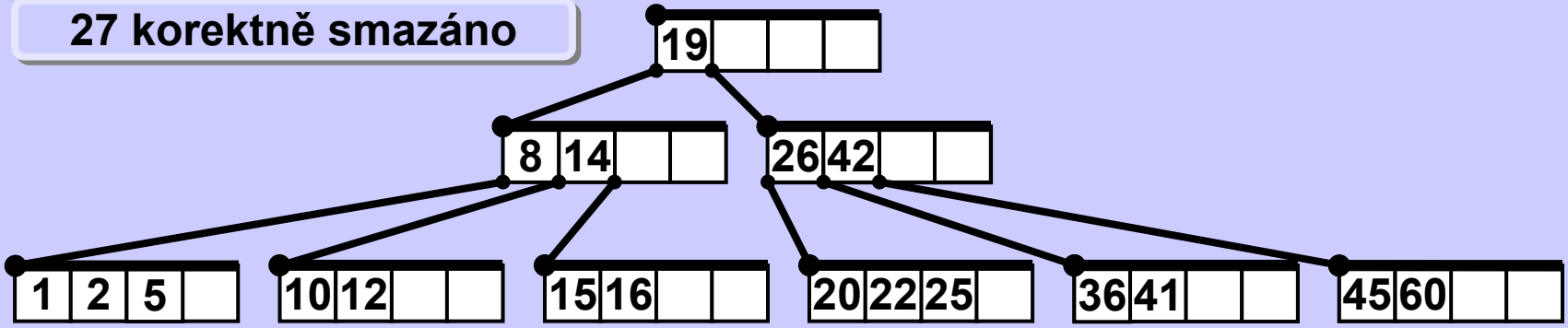
36 41 42 45 60

Medián sjednocení vlož na místo původně dělícího klíče, menší a větší klíče než medián rozděl do levého a pravého listu.



B-strom -- Delete

27 korektně smazáno

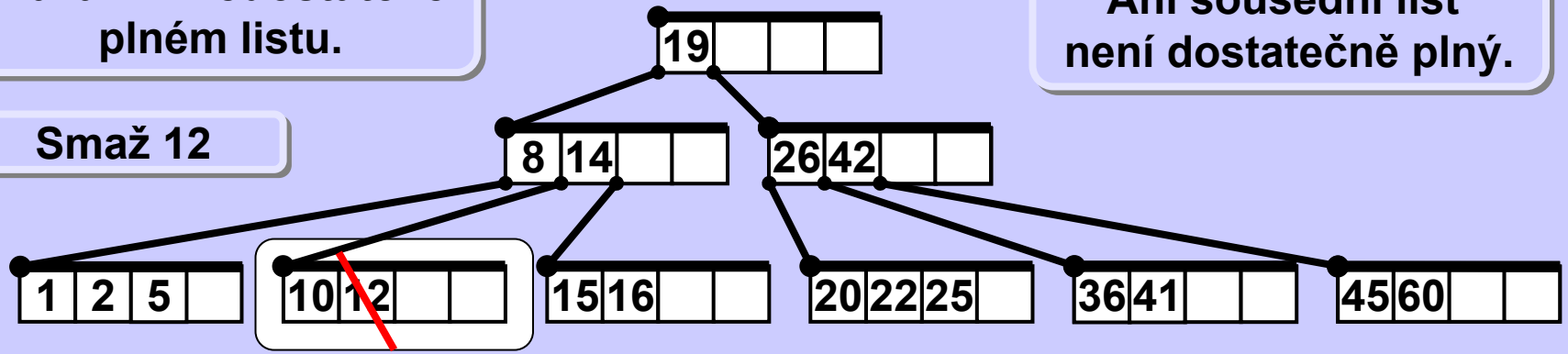


B-strom -- Delete

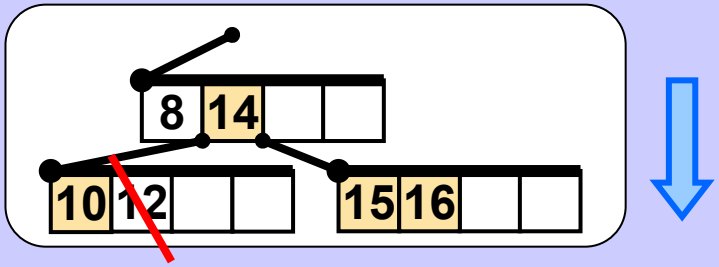
Mazání v nedostatčně plném listu.

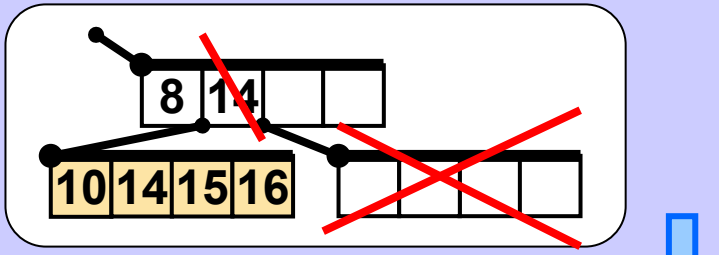
Ani sousední list není dostatečně plný.

Smaž 12



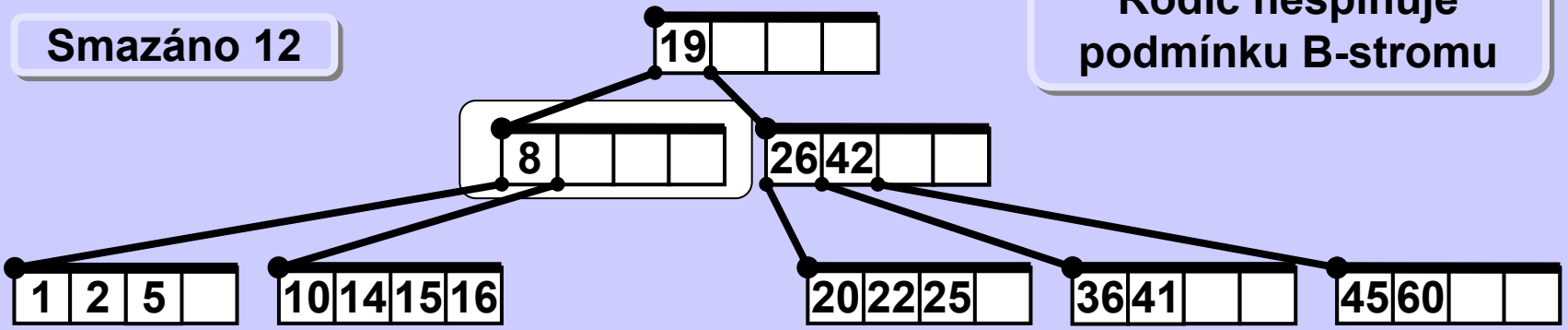
Sjednoť klíče s klíči v sousedním listu a s dělícím klíčem v rodiči a seřaď.
Vše vlož do původního listu,
sousední list smaž,
dělící klíč v rodiči také smaž.





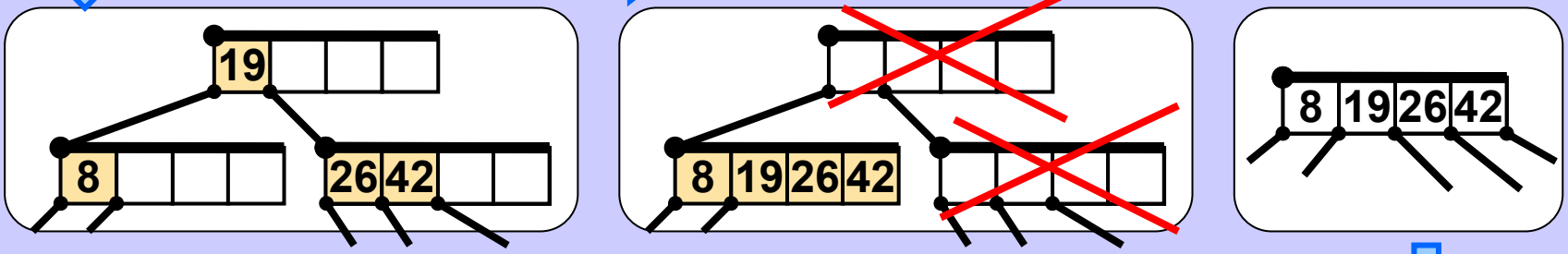
B-strom -- Delete

Smazáno 12



Rodič nesplňuje podmínku B-stromu

Rodič, který poskytl klíč potomku, není dostatečně plný.
Aplikujeme na něj (a případně iterativně na jeho rodiče) tentýž postup spojení klíčů a sousedních uzlů a přesun dělícího prvku z rodiče.



B-strom -- Delete

Smazáno 12
a strom byl adekvátně restrukturován.