Teorie informace a kódování

L05: B-stromy, R-stromy

Jan Konecny

4. listopadu 2021

Plán na dnešek (přednáška i cvičení)

- B-stromy,
- jejich varianty: B^+-stromy, B^*-stromy
- R-stromy
- (zadání úkolů snad za ???+ bodů)

B-strom

- samovyvažující se stromová datová struktura, která udržuje setříděná data a umožňuje vyhledávání, sekvenční přístup, vkládání a mazání v logaritmickém čase.
- uvedeny v:

R. Bayer, E. M. McCreight.
Organization and maintenance of large ordered indexes.
Acta Informatica, 1(3):173–189, 1972.

Poznámka

Nevysvětlili, proč se to tak jmenuje.
Boeing, balanced, broad, bushy, Bayer, . . . ?

- na rozdíl od jiných samovyvažovacích binárních vyhledávacích stromů se B-strom dobře hodí pro úložné systémy, které čtou a zapisují relativně velké bloky dat.
- běžně se používá v databázích a souborových systémech.

Poznámka

Podobná motivace může být i u dokonalého hašování:

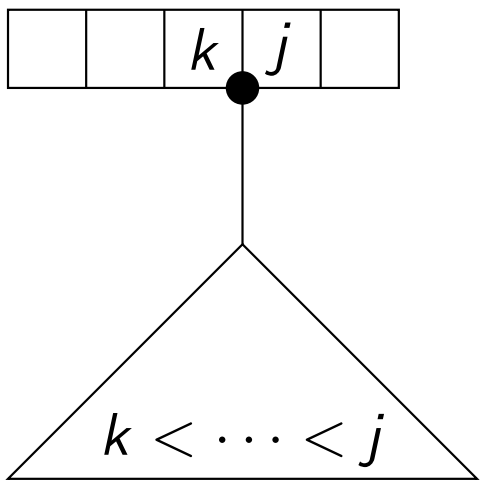
- když máme malou, rychlou paměť ve které můžeme uložit dokonalou hašovací funkci, zatímco klíče a k nim příslušná data jsou uložena v pomalejší ale větší paměti.
- Velikost bloku může být vybrána tak, aby se do něj vešlo k položek, které budou načteny během jednoho přístupu.
- tomu se říká k-dokonalé hašování; algoritmy pro dokonalé hašování lze snadno upravit na algoritmy pro k-dokonalé hašování.

B-strom stupně t je strom, který splňuje následující vlastnosti:

- Všechny listy jsou ve stejné hloubce
- B-strom je definován minimálním stupněm t.*.
- Každý uzel kromě kořene musí obsahovat alespoň $t - 1$ klíčů. Kořen může obsahovat minimálně 1 klíč (pokud není list)
- každý uzel (včetně kořene) může obsahovat nejvýše $2t - 1$ klíčů.
- počet potomků uzlu je roven počtu klíčů v něm $+1$
- všechny klíče uzlu jsou seřazeny ve vzestupném pořadí.
- potomek mezi dvěma klíči k a j obsahuje pouze klíče v rozsahu od k a j.

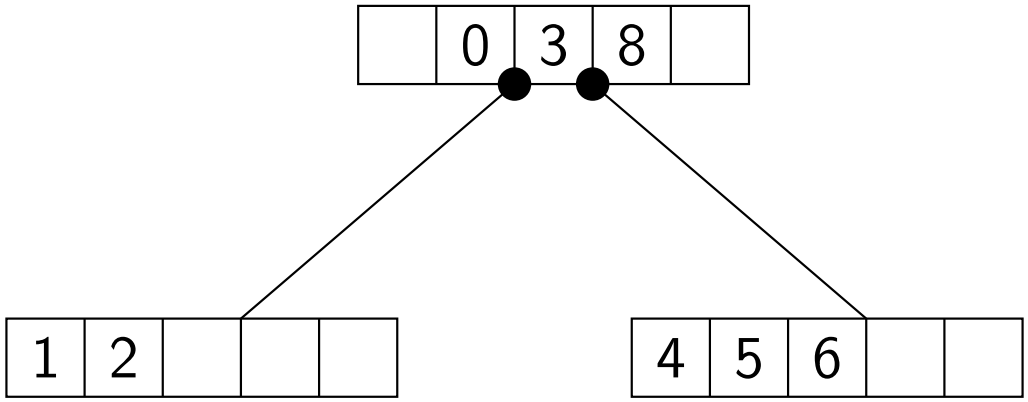
* – v příkladech používám $t = 3$, v praxi je větší (závisí na velikosti bloku)

Potomek mezi dvěma klíči k a j obsahuje pouze klíče v rozsahu od k a j.



Vkládání

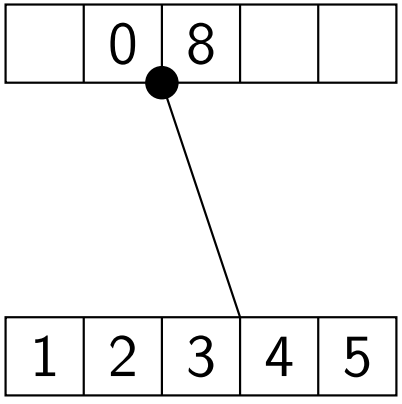
- Vkládá se vždy do listu; najdeme tedy správný list
- Pokud je v něm něm místo, vlož tam; jinak proveď *split*:



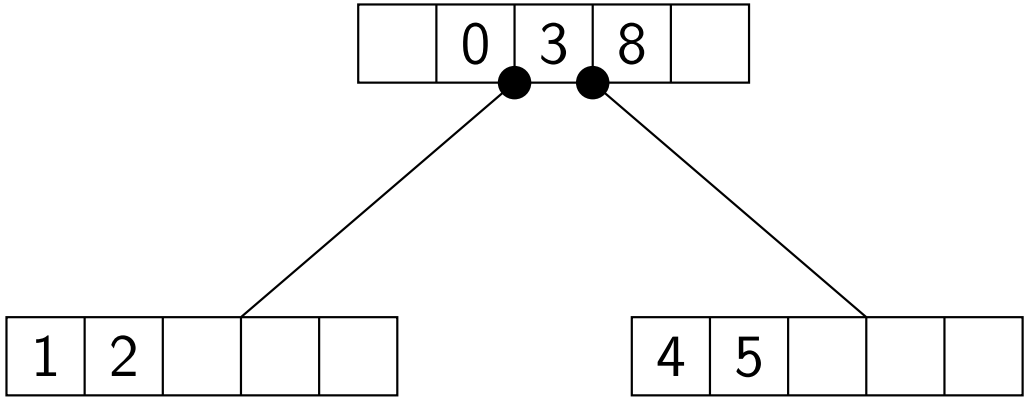
- split se může šířit nahoru

Lepší možnost: preventivní splity:

- kdykoli bychom měli při sestupu sejít do plného uzlu, splitneme ho.



přidáváme 6 a sestupujeme do plného uzlu

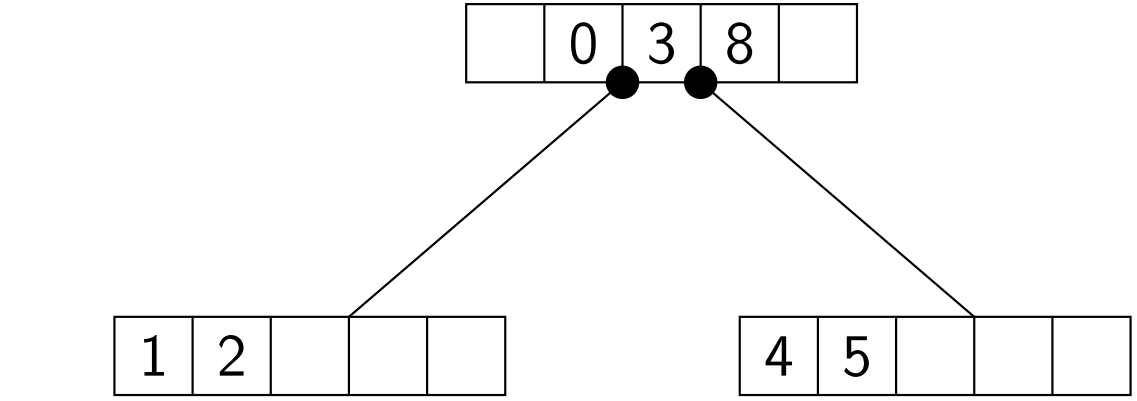


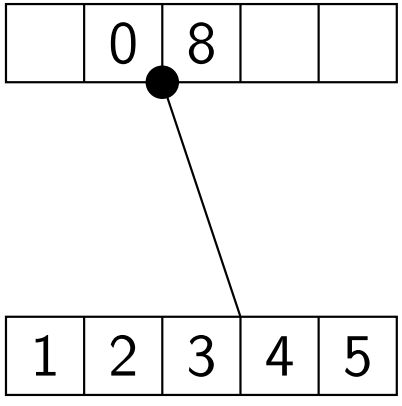
sestoupíme
do odp. uzlu

Mazání klíče k:

- Pokud je k je v uzlu x a x je list, smaž k z x.
- Pokud je k je v uzlu x a x je vnitřní uzel:
  - Pokud má potomek y, který předchází k v uzlu x, alespoň t klíčů, pak najdi předchůdce k_0 klíče k v podstromu s kořenem y.
    Rekurzivně odstraň k_0 a nahraď k za k_0 v x.
  - Pokud má y méně než t klíčů, symetricky zkus totéž pro následníka klíče k:
    Pokud má potomek z, který následuje k v uzlu x, alespoň t klíčů, pak najdi následníka k_0 klíče k v podstromu s kořenem z.
    Rekurzivně odstraň k_0 a nahraď k za k_0 v x.
  - Jinak (pokud y i z) mají jen $t-1$ klíčů, vlož k a všechny klíče ze z do y a smaž z (i ukazatel na něj). Uzel y má teď $2t-1$ uzlů.
    Rekurzivně smaž k z y.

Mažeme 3:



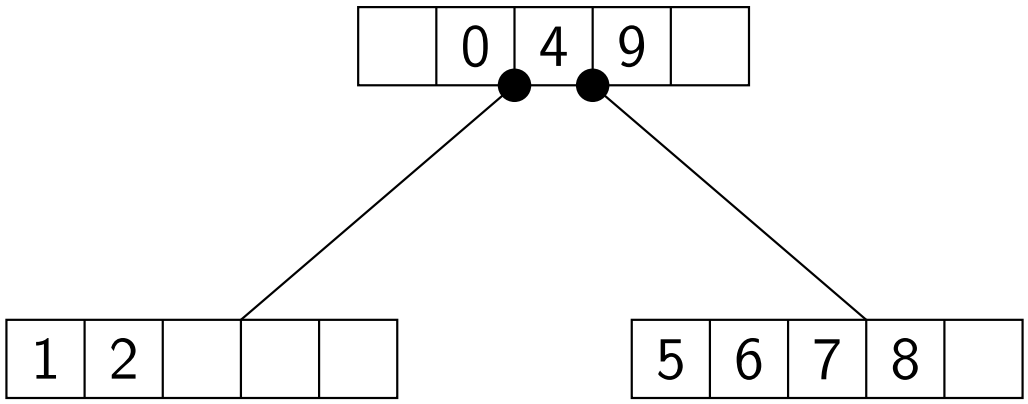


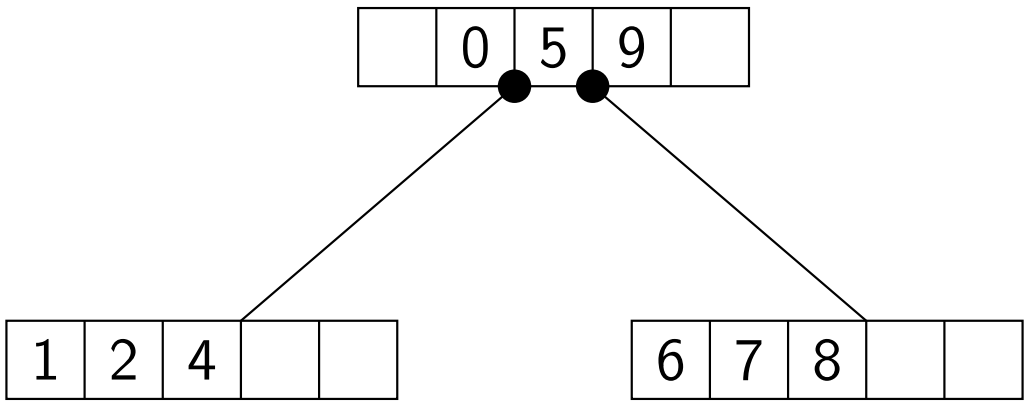
mažeme 3 z tohoto uzlu

- Pokud k není ve vnitřním uzlu x, najdeme kořen $x.c(i)$ příslušného podstromu, který musí obsahovat k (pokud k ve stromu je).
  Pokud má $x.c(i)$ pouze $t-1$ klíčů, proveď podle potřeby krok 3a nebo 3b, aby bylo zaručeno, že sestoupíme do uzlu obsahujícího alespoň t klíčů.
  Sestup tedy do příslušného potomka x.

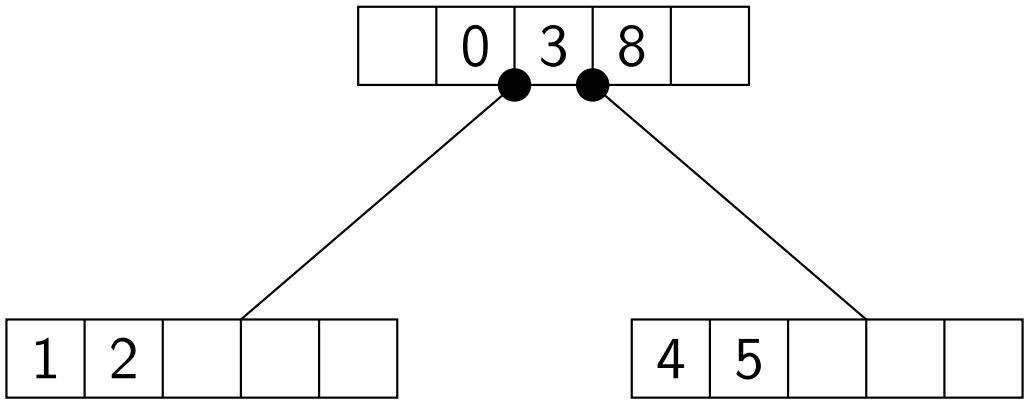
  3a Pokud $x.c(i)$ má jen $t-1$ klíčů, ale má sourozence s alespoň t klíči, proveď rotaci:
  Dej do $x.c(i)$ klíč z x a místo toho klíče x klíč z levého či pravého sourozence $x.c(i)$, přesuň i odpovídající ukazatele.

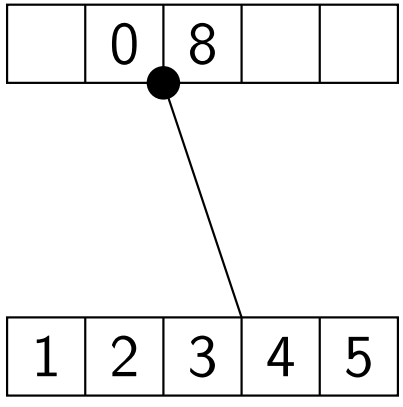
Rotace





3b Pokud $x.c(i)$ a oba jeho bezprostřední sourozenci mají $t - 1$ klíčů, slej $x.c(i)$ s jedním tím sourozencem.





B^+-strom

Struktura vnitřních uzlů B^+-stromu řádu a je následující:

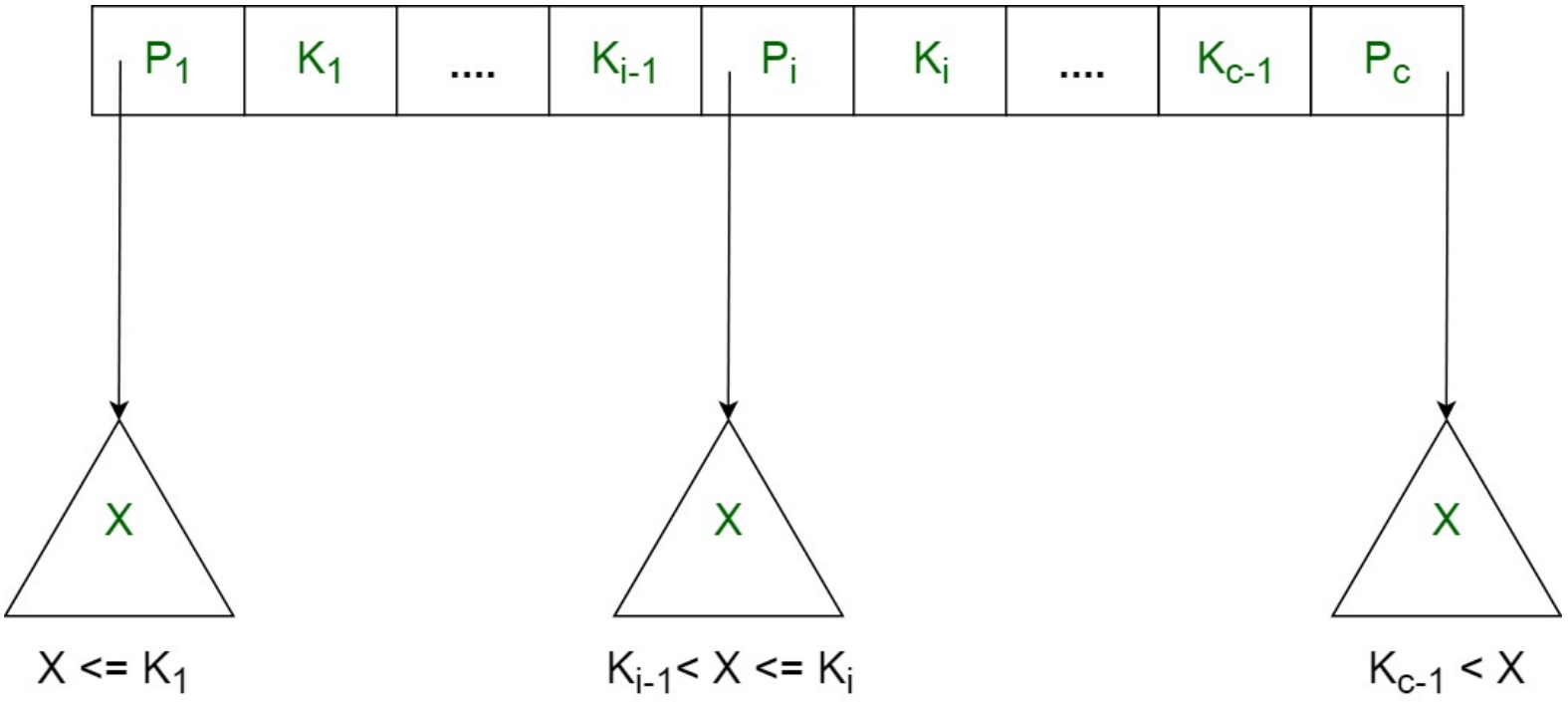
- Každý vnitřní uzel je ve tvaru:

$$\langle P_1, K_1, P_2, K_2, \ldots, P_{c-1}, K_{c-1}, P_c \rangle$$

  kde $c \leq a$,
  - každý P_i je ukazatel na uzel stromu (podstrom)
  - každý K_i je klíč
- Každý vnitřní splňuje má $K_1 < K_2 < \cdots < K_{c-1}$.
- Pro každý klíč X v podstromu P_i platí:

$$K_{i-1} < X \leq K_i, \quad \text{pro } 1 < i < c$$
$$K_{i-1} < X \quad \text{pro } i = c$$

- Kořen má alespoň dva potomky, jiné interní uzly mají alespoň $\lceil \frac{a}{2} \rceil$ potomků.


P1
K1
....
Ki-1
Pi
Ki
....
Kc-1
Pc
X
X
X
X <= K1
Ki-1< X <= Ki
Kc-1 < X

Struktura listových uzlů B^+ stromu řádu b:

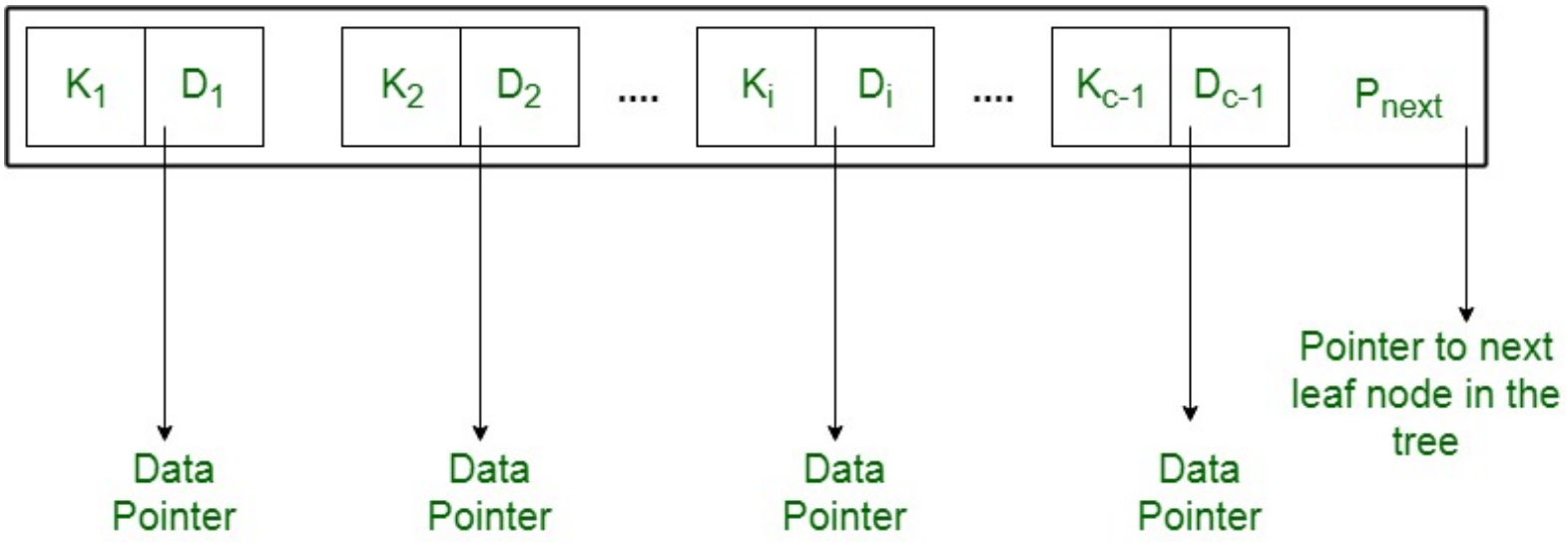
- Každý list je ve tvaru

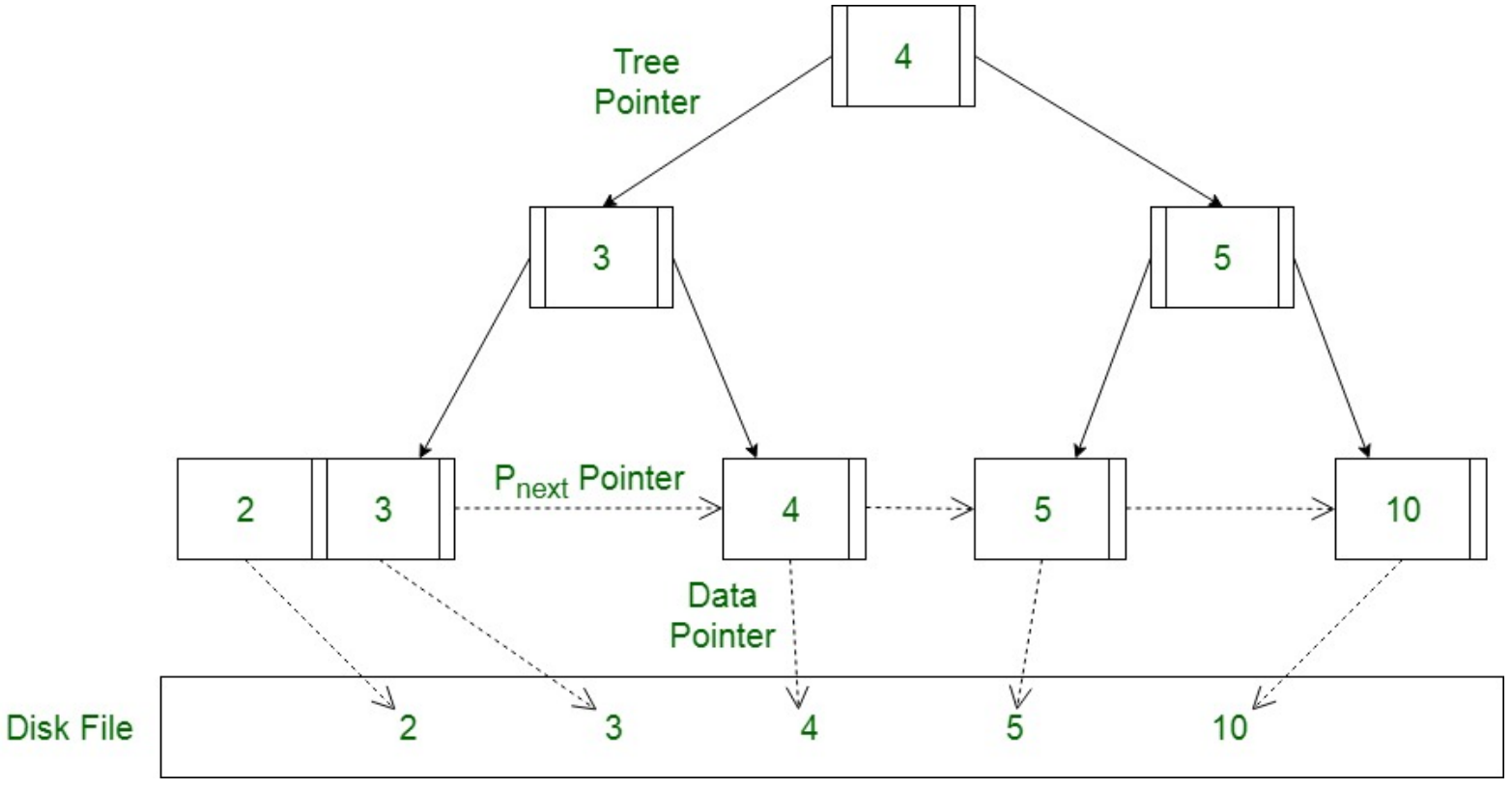
$$\langle \langle K_1, D_1 \rangle, \langle K_2, D_2 \rangle, \ldots, \langle K_{c-1}, D_{c-1} \rangle, P_{\mathrm{next}} \rangle$$

  kde $c \leq b$ a
  - každý D_i je ukazatel na data (tj. ukazuje na skutečný záznam na disku, jehož hodnota klíče je K_i nebo na blok souboru disku obsahující tento záznam)
  - každý K_i je klíč
  - P_{next} ukazuje na následující uzel v B^+-stromu.
- Každý klíč splňuje

$$K_1 < K_2 < \cdots < K_{c-1}$$

- Každý list má alespoň $\lceil \frac{b}{2} \rceil$ klíčů.
- Všechny listy jsou ve stejné hloubce.


K1
D1
K2
D2
....
Ki
Di
....
Kc-1
Dc-1
Pnext
Data Pointer
Data Pointer
Data Pointer
Data Pointer
Pointer to next leaf node in the tree


Tree
Pointer
4
3
5
2
3
P_{next} Pointer
4
5
10
Data
Pointer
Disk File
2
3
4
5
10

Vkládání do B*-stromu

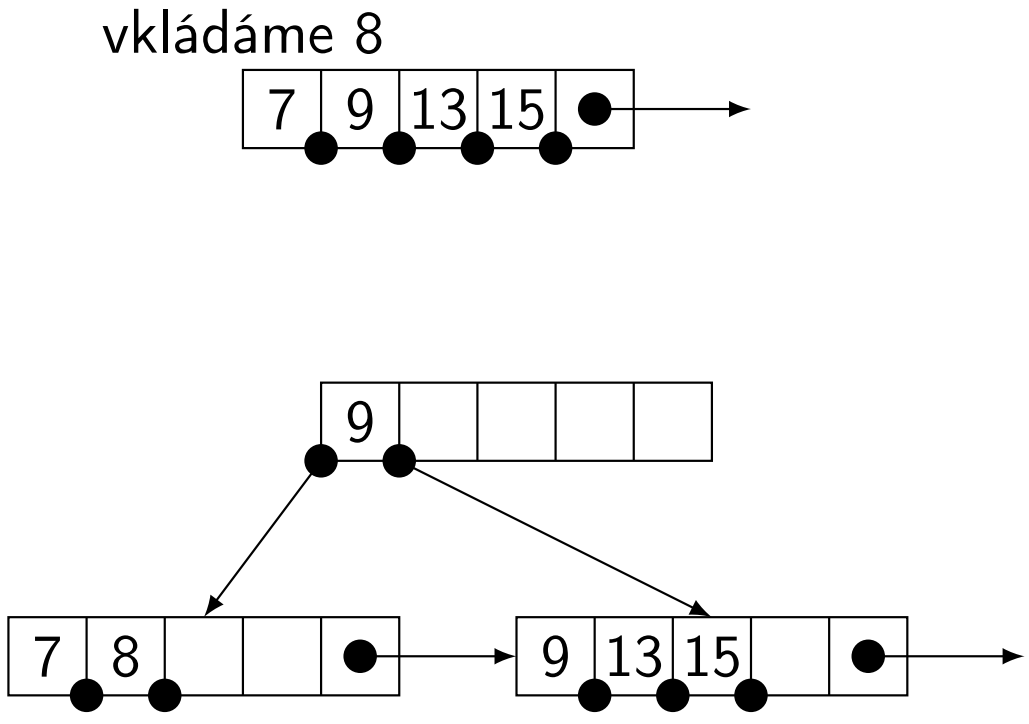
- Každý klíč se vkládá do listového uzlu – takže sejdi do odpovídajícího listu.
- Vlož klíč do listového uzlu v rostoucím pořadí, pokud nenastane přeplnění.
  Pokud by nastalo přeplnění, pokračuj následujícími kroky:

Přeplnění listového uzlu:
Rozděl listový uzel na dva uzly:

- první obsahuje $\lceil \frac{m-1}{2} \rceil$ klíčů.
- druhý obsahuje zbývající klíče.

Nakopíruj nejmenší klíč z pravého uzlu do rodičovského uzlu.

$m = 5$



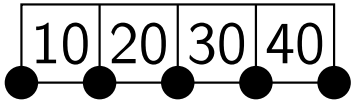
Přeplnění interního uzlu:
Rozděl interního uzel na dva uzly:

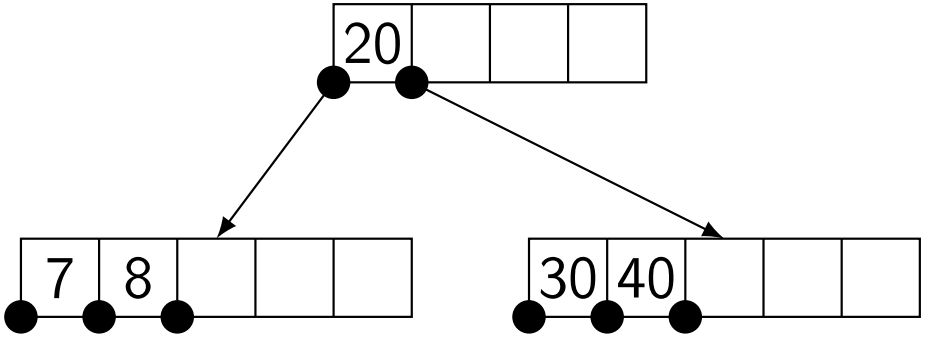
- první obsahuje $\lceil \frac{m-1}{2} \rceil$ klíčů.
- druhý obsahuje zbývající klíče.

Nakopíruj nejmenší klíč z pravého uzlu do rodičovského uzlu.

$m = 5$

zezdola kopírujeme 15



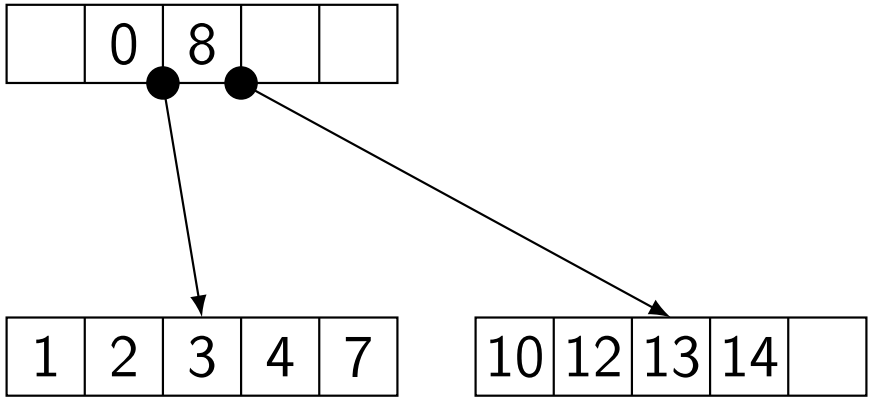


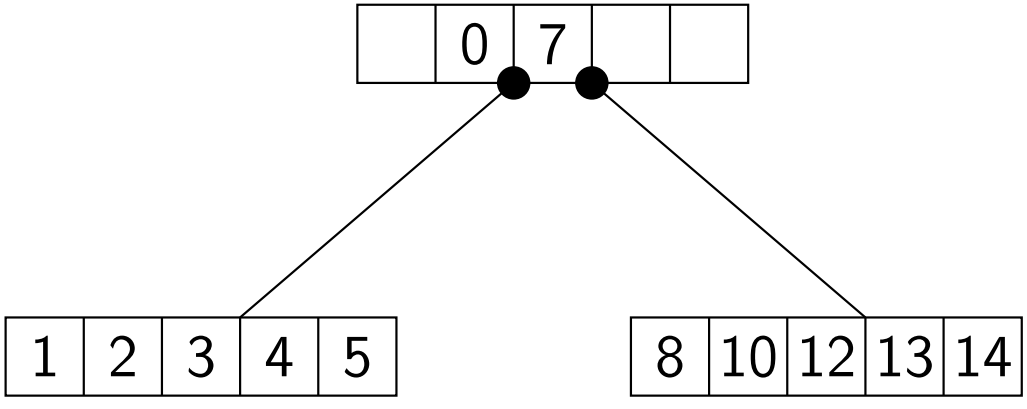
B*-stromy

Knuth D.
The art of computer programming, Vol. 3: sorting and searching,
Addison-Wesley Publ. Co, Reading, Mass., 1973.

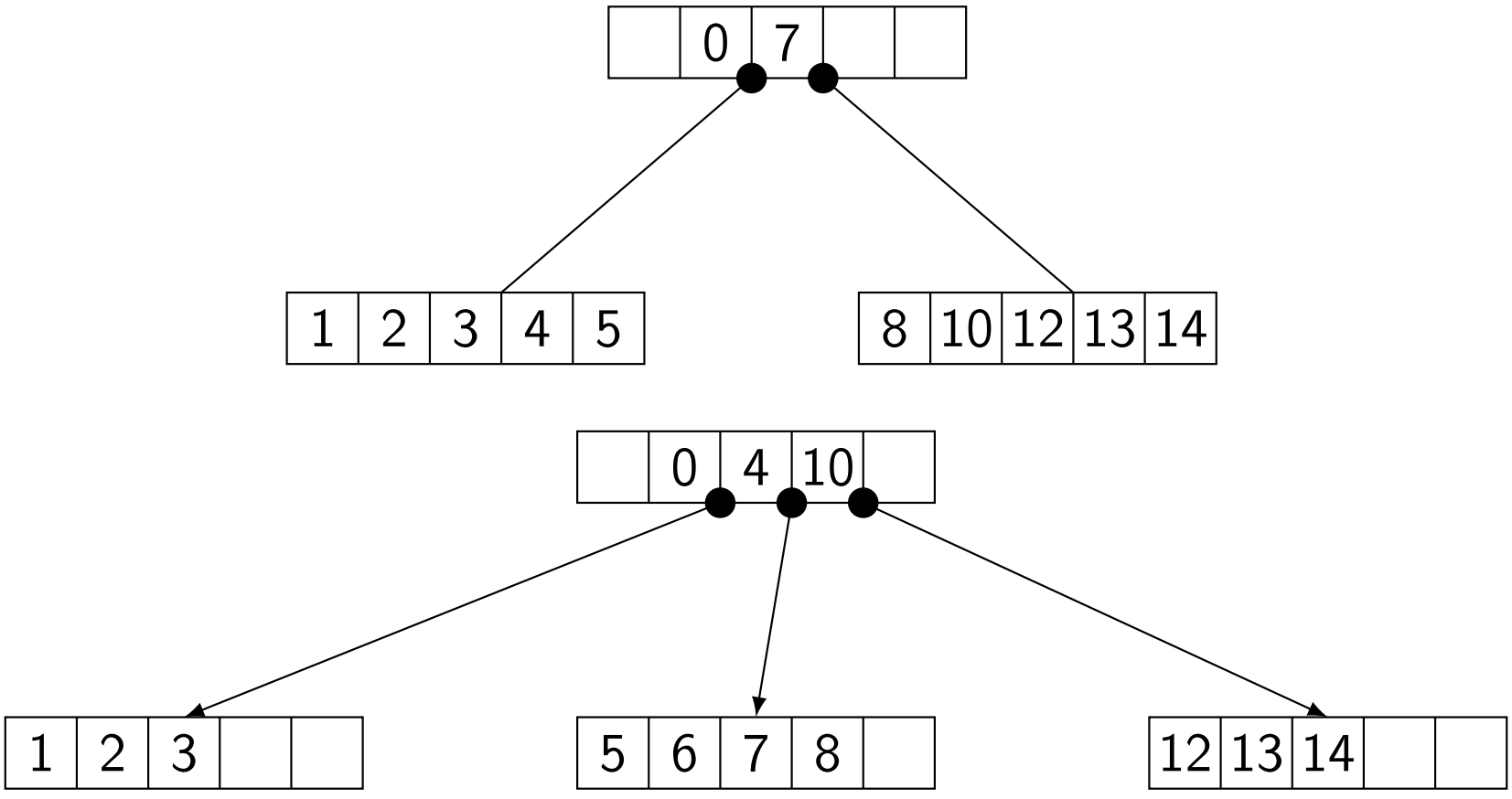
- zajišťuje zaplnění alespoň $\frac{2}{3}$
- vpodstatě provádí rotace při přeplnění

Přidáváme 5





Přidáváme 6:



R-stromy

- R-strom je datová struktura pro správu prostorových dat, který je strukturou podobný B-stromu.
- poprvé byly představeny v:

  A. Guttman,
  R-Trees: A dynamic index structure for spatial searching,
  in Proc. of ACM SIGMOD, June 1984, pp. 47–57

- viz též:

  Y. Manolopoulos, A. Nanopoulos, A. N. Papadopoulos, and Y. Theodoridis.
  R-trees: Theory and Applications.
  Series in Advanced Information and Knowledge Processing.
  Springer, 2005.

Nechť

- M je maximální počet záznamů, které mohou být v uzlu.
- $m \leq \frac{M}{2}$ je minimální počet záznamů, které mohou být v uzlu.

R-strom splňuje následující:

- Každý uzel obsahuje m až M indexových záznamů, pokud to není kořen.
- Pro každý indexový záznam (I, id) v listovém uzlu, I je nejmenší obdélník obsahující n-dimenzionální datový objekt reprezentovaný daným id
- Každý nelistový uzel, vyjma kořene, má m až M potomků
- Pro každý záznam (I, c) ve vnitřním uzlu, I je nejmenší obdélník, který obsahuje všechny obdélníky v potomku c.
- Kořen má alespoň dva potomky, pokud to není list.
- Všechny listy mají stejnou hloubku.

- Výška R-stromu obsahující N indexových záznamů je nejvýše

$$\lceil \log_m N \rceil - 1,$$

  protože počet větví u každého každého uzlu je nejméně m.
- Maximální počet uzlů je

$$\left\lceil \frac{N}{m} \right\rceil + \left\lceil \frac{N}{m^2} \right\rceil + \left\lceil \frac{N}{m^2} \right\rceil + \cdots + 1$$

- Nejhorší prostorové využití R-stromu je $\frac{m}{M}$.

Uzly budou mít spíše větší počet záznamů, což zlepší prostorové využití.

Pokud uzly mají více než 3–4 záznamy, strom je velmi široký a téměř všechen prostor je použitý na listové uzly obsahující indexové záznamy.

Parametr m může být měněn jako součást zlepšování výkonu.

Vyhledávání oblasti

- Vyhledávání v R-stromu sestupuje od kořene dolů – podobně jako vyhledávání v B-stromu.
- Na rozdíl od B-stromu ale může být potřeba vyhledávat ve více než jednom podstromu navštíveného uzlu. Takže není možné zaručit dobrý výkon v nejhorším případě.
- Přesto u většiny druhů dat procedury `R-Tree-Insert` a `Delete` udržují R-strom v podobě, která umožňuje vyhledávacímu algoritmu eliminovat irelevantní části indexovaného prostoru a zkoušet jen data blízko oblasti vyhledávání.

Vyhledávání oblasti

- V následujícím popisu je označen obdélník indexového záznamu E písmeny I_E a id či c jsou označeny p_E.
- Symbol $\Join$ představuje překryv.

`R-Tree-Search`:

- bere jako argumenty R-strom s kořenem T, vyhledávaný obdélník S;
- najde všechny indexové záznamy, které překrývají S.

S1 [Prohledej podstromy] Pokud T není list, prozkoumej všechny záznamy E a zjisti jestli $S \Join E$.
Pro všechny takové E vyvolej `R-Tree-Search` se stromem, který je zakořeněný v p_E.

S2 [Prohledej list] Pokud T je list, projdi všechny záznamy E a zjisti, jestli $S \Join E$.
Pokud ano, E je výsledný záznam.

Vkládání

Vkládání indexových záznamů pro nová data do R-stromu je podobné vkládání do B-stromu:

- nové indexové záznamy jsou přidávané do listů
- uzly které jsou přeplněné se rozlomí a rozlamování uzlu se šíří stromem směrem nahoru.

`R-Tree-Insert` – vkládá nový indexový záznam E do R-stromu T.

I1 [Najdi pozici pro nový záznam]
Vyvolej proceduru R-Tree-Choose-Leaf na výběr uzlu L,
do kterého bude umístěn E.

I2 [Přidej záznam do listového uzlu]
Pokud L má místo pro další záznam, vlož tam E. Jinak vyvolej `R-Tree-Split-Node` a získej L a LL obsahující E a všechny staré záznamy z L.

I3 [Rozšiř změny nahoru]
Vyvolej proceduru R-Tree-Adjust-Tree na L
(i s LL, pokud bylo provedeno rozlomení).

I4 [Zvyš výšku stromu]
Pokud se rozlamování uzlu postupně dostalo až ke kořeni, vytvoř nový kořen, jehož potomci budou výsledné dva uzly (L a LL).

`R-Tree-Choose-Leaf` – vybere list, do kterého se umístí nový indexový záznam E.

CL1 [Inicializace]
Nastav $N \leftarrow$ kořen

CL2 [Kontrola na list]
Pokud N je list, vrať N.

CL3 [Vyber podstrom]
Pokud N není list, najdi záznam F v N tak,
že I_F potřebuje nejmenší rozšíření, aby $I_E \subseteq I_F$.
Remízy vyřeš výběrem záznamu s menším obdélníkem.

CL4 [Sestupuj k až listu]
Nastav $N \leftarrow p_F$ a opakuj od kroku CL2.

`R-Tree-Adjust-Tree:` – vystupuj od listu ke kořenu, upravuj velikosti

AT1 [Inicializace]
Nastav $N \leftarrow L$;
pokud L byl rozlomen, nastav $NN \leftarrow LL$ na výsledný druhý uzel.

AT2 [Kontrola na dokončení]
Pokud N je kořen, skonči.

AT3 [Uprav obdélník v předkovi]
Nechť P je předek uzlu N a E_N je záznam v P obsahující N.
Uprav I_{EN} tak, aby těsně obsahovalo obdélníky všech záznamů v N.

AT4 [Šiř rozlamování uzlů směrem ke kořenu]
Pokud máme NN z předchozího prolamování, vytvoř pro něj nový záznam $E_N N$ s $p_{E_{NN}} \leftarrow NN$ a $I_{E_{NN}} \leftarrow$ obdélník uzavírající všechny obdélníky v NN.
Přidej E_{NN} do P pokud je tam místo.
V opačném případě vyvolej `R-Tree-Split-Node`, aby se vytvořilo P a PP obsahující E_{NN} a všechny staré záznamy z P.

AT5 Nastav $N \leftarrow P$; $NN \leftarrow PP$, pokud nastalo rozlomení,
a opakuj od kroku AT2.

Mazání z R-stromu

`R-tree-delete` – odstraní indexový záznam E z R-stromu.

D1 [Najdi uzel, který obsahuje E]
Vyvolej `R-Tree-Find-Leaf` a najdi tak listový uzel L obsahující E.
Pokud takový uzel neexistuje, skonči.

D2 [Smaž záznam]
Odstraň E z L.

D3 [Šiř změnu]
Vyvolej `R-Tree-Condense` a předej jí L.

D4 [Zmenši výšku stromu]
Pokud kořen má jen jednoho potomka po předchozím kroku, udělej z toho potomka nový kořen.

`R-Tree-Find-Leaf` – vyhledá uzel s konkrétním záznamem E

FL1 [Prohledej podstromy]
Pokud T není list, najdi záznamy F v T, tak že $I_E \subseteq I_F$.
Pro každý takový záznam vyvolej `R-Tree-Find-Leaf` pro strom, který je zakořeněný v p_F dokud záznam E není nalezen, nebo nebyly prohledány všechny záznamy.

FL2 [Prohledej list]
Pokud T je list, najdi v něm záznam shodný s E.
Pokud je takové E nalezeno, vrať T.

`R-Tree-Condense` – pro listový uzel L, ze kterého byl odstraněn záznam:

- pokud má málo záznamů, odstraň uzel, a přesuň jeho záznamy.
- šiř změnu, pokud je to potřeba.
- zmenšuj všechny obdélníky na cestě ke kořeni, pokud to jde.

CT1 [Inicializace]
Nastav $N \leftarrow L$, $Q \leftarrow \emptyset$;
Q je množina eliminovaných uzlů.

CT2 [Najdi předka]
Pokud N je kořen, pokračuj bodem CT6.
Jinak nechť P je předka N, E_N záznam v P, ve kterém je N.

CT3 [Odstraň nedostatečně naplněný uzel]
Pokud N má méně než m záznamů, smaž E_N z P a přidej N do Q.

CT4 [Uprav obdélník]
Pokud N nebyl eliminován, uprav I_{E_N} tak, aby těsně obsahoval všechny záznamy v N.

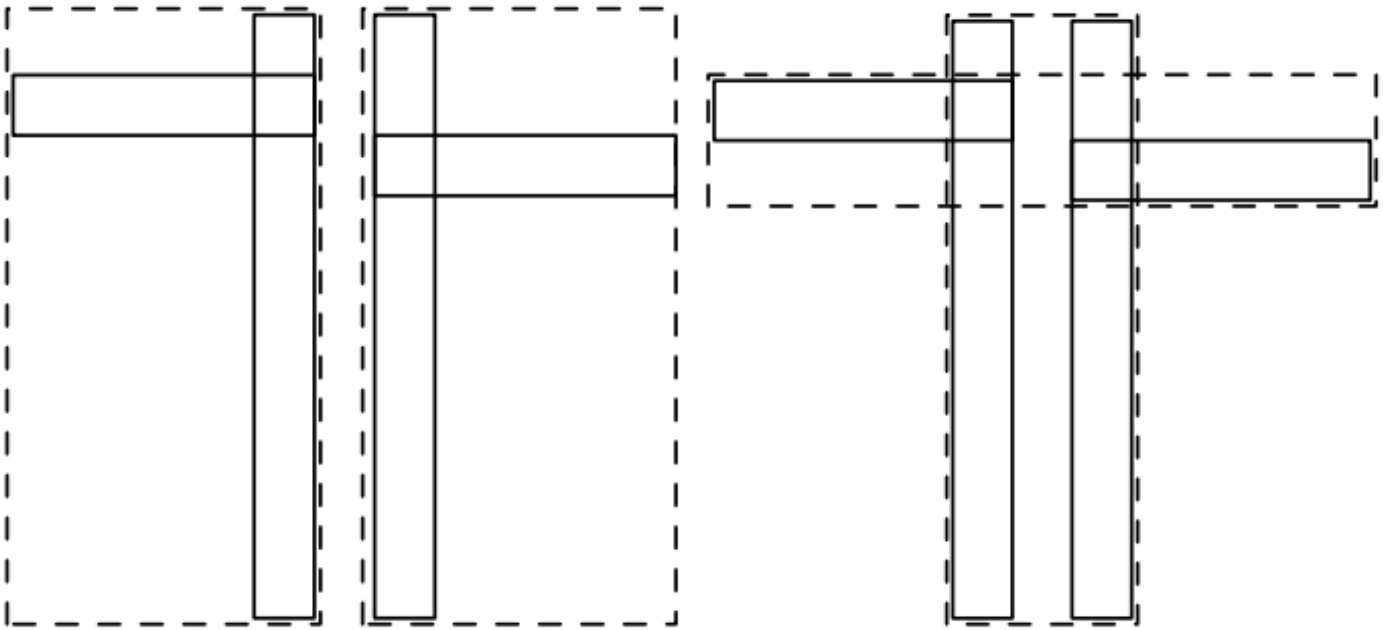
CT5 [Postup nahoru]
Nastav $N \leftarrow P$ a pokračuj bodem CT6.

CT6 [Znovu vlož osiřelé záznamy]
Znovu-vlož všechny záznamy uzlů v množině Q.
Záznamy z listových uzlů jsou vloženy tak jak je popsáno v Algoritmu `R-Tree-Insert`, ale záznamy z vnitřních uzlů musí být uloženy výše ve stromu, tak že listy jejich závislých podstromů budou na stejné úrovni, jako listy hlavního stromu.

Rozlomení uzlu

- Kvůli uložení nového záznamu do plného uzlu, který obsahuje M záznamů, je nutné rozdělit kolekci $M + 1$ záznamů mezi dva uzly.
- Toto rozdělení by mělo být provedeno tak, aby bylo nepravděpodobné, že v následných použití `R-Tree-Search` se neprohledávaly oba nové uzly.
- Protože rozhodnutí, jestli uzel bude navštíven závisí na na tom, jestli jeho obdélník zasahuje hledanou oblast, měla by být oblast dvou nových obdélníků co nejmenší.
- Stejné kritérium se používalo v proceduře `R-Tree-Choose-Leaf` k rozhodnutí kam se vloží nový idexový záznam.
- Na každé úrovni ve stromu byl vybírán ten podstrom, jehož obdélník by se zvětšil nejméně. Nyní uvedu několik algoritmů pro rozklad množiny $M + 1$ záznamů do dvou tříd – každá pro jeden nový uzel.

Špatný split (vlevo) a dobrý split (vpravo):



Kvadratický split:

- pokouší se najít split na malé oblasti, ale není zaručeno, že najde ty nejmenší.
- kvadratická složitost vzhledem k M a lineární vzhledem k počtu dimenzí.
- Algoritmus vybere dva záznamy z celkových $M + 1$ záznamů, jako první prvky nových skupin tak, že vybere tu dvojici prvků, které by měly největší mrtvou zónu, kdyby byly vloženy do jedné skupiny.
- Mrtvá zóna je obsah oblasti pokrývající oba záznamy mínus obsah oblastí samotných záznamů.
- Zbývající záznamy jsou přidělovány skupinám po jednom.
- V každém kroku je pro každý záznam vypočítán obsah o který by musela byt současná oblast skupiny rozšířena.
- Vybrán je ten záznam u kterého je největší rozdíl ve velikosti té oblasti.

`Split` – rozdělí $M + 1$ záznamů do dvou skupin

QS1 [Vyber první záznam pro každou skupinu]
Vyvolej proceduru `PickSeeds` na výběr dvou záznamů, které se stanou prvními prvky skupin. Přiřaď každý jedné skupině.

QS2 [Kontrola na konec]
Pokud všechny záznamy byly přiřazeny, skonči.
Pokud jedna z skupin má tak málo záznamů, že všechny nepřiřazené záznamy se musí přidat do této skupiny, aby měla alespoň m záznamů, proveď toto přiřazení a skonči.

QS3 [Vyber záznam k přiřazení]
Vyvolej `PickNext` k výběru dalšího záznamu, který bude přiřazen.
Přiřaď ho do té skupiny, jejíž pokrývající obdélník bude muset být rozšířen méně, aby zahrnul jako obdélník.
Remízu vyřeš výběrem skupiny s menším pokrývajícím obdélníkem, pak ten s menším počtem záznamů.
Pokračuj krokem QS2

`PickSeeds` – vybere dva záznamy, které se stanou prvními prvky skupin.

PS1 [Vypočítej neefektivnost přiřazení záznamů jedné skupině]
Pro každý pár záznamů E_1 a E_2, vytvoř MBR, který pokrývá I_{E_1} a I_{E_2}, vypočítej

$$d = \mathrm{obsah}(\mathrm{MBR}(I_{E_1} \cup I_{E_2})) - \mathrm{obsah}(I_{E_1}) - \mathrm{obsah}(I_{E_2}).$$

PS2 [Vyber nejméně efektivní pár] Vyber pár s největším d.

`PickNext` – vyber záznam, který má být jako další zařazen do skupiny.

PN1 [Vypočítej cenu vložení každého záznamu do každé třídy]
Pro každý záznam E, který ještě není přiřazený žádné skupině, vypočítej

- d_1 obsah, o který se zvětší pokrývající obdélník první skupiny,
- podobně vypočítej d_2 pro druhou třídu.

PN2 [Vyber záznam s nejvyšší preferencí]
Vyber záznam s maximálním rozdílem d_1 a d_2.

Lineární split:

- je lineární vzhledem k M a vzhledem k počtu dimenzí.
- je identický s kvadratickým splitem, jen používá jinou verzi `PickSeeds`.
- `PickNext` jednoduše vybírá kterýkoli ze zbývajících záznamů.

`PickSeeds:` – vybere dva záznamy, které se stanou prvními prvky skupin.

LS1 [Vyber extrémní obdélníky mezi všemi rozměry]
Najdi obdélník s nejpravějším levým okrajem a obdélník s nejlevějším pravým okrajem.
To udělej pro všechny dimenze (tedy pro vertikální: Najdi obdélník s nejhornějším dolním okrajem a obdélník s nejspodnějším horním okrajem, atd.)
Zapamatuj si tyto páry a jejich separaci (tj. rozdíl mezi nejpravějším levým okrajem a obdélník s nejlevějším pravým okrajem, etc.).

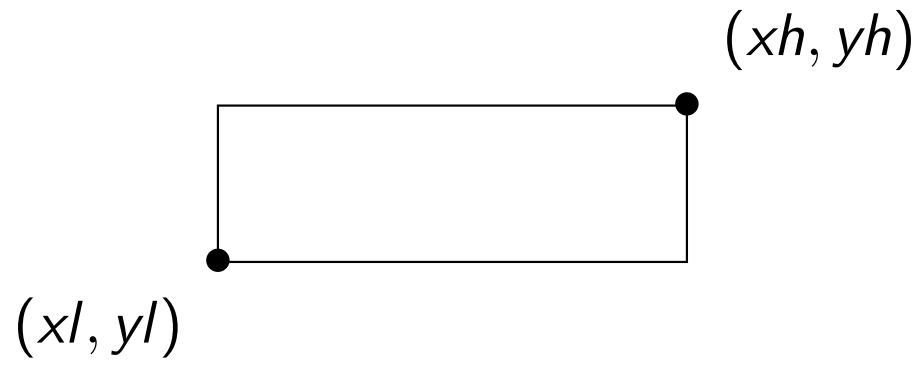
LS2 [Uprav]
Normalizuj separaci vydělením odpovídajícím rozměrem obdélník pokrývající všechny záznamy.

LS3 [Vyber nejextrémnější pár]
Vyber pár s největší normalizovanou separací podle kterékoli dimenze.

Jiný lineární split:
Ukážeme si ještě jeden lineární algoritmus pro distribuci objektů přeplněného uzlu do dvou množin.

- Primárním kritériem tohoto algoritmu je rozdělit objekty mezi dva uzly co nejrovnoměrněji.
- Druhým kritériem je minimalizace jejich překrytí.
- Třetím kritériem je minimalizace celkového pokrytí.

Aby se minimalizovalo překrývání, snažíme se dát všechny obdélníky v přeplněném uzlu co nejdál od sebe, k okrajům MBR přeplněného uzlu. Nechť je každý obdélník označený jako (xl, yl, xh, yh):



MBR přeplněného uzlu N je označen $R_N = (L, B, R, T)$.

Tento algoritmus používá čtyři listy $\mathrm{LIST}_L, \mathrm{LIST}_B, \mathrm{LIST}_R, \mathrm{LIST}_T$, které obsahují obdélníky z N, které jsou blíže odpovídající hranici, než opačné (opačné hranice jsou uvažovány přirozeně: L–R, B–T).

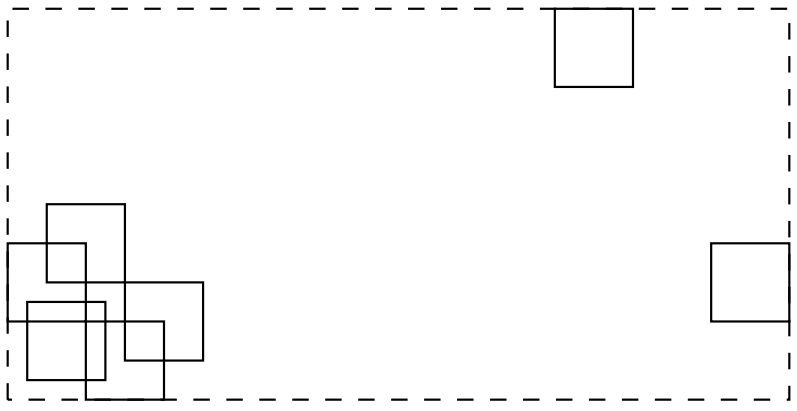
$\mathrm{LIST}_L \leftarrow \mathrm{LIST}_B \leftarrow \mathrm{LIST}_R \leftarrow \mathrm{LIST}_T \leftarrow \emptyset$;
for *rectangle* $S = (xl, yl, xh, yh)$ *in* N *with* $R_N = (L, B, R, T)$ **do**
    if $xl - L < R - xh$ **then**
        $\mathrm{LIST}_L \leftarrow \mathrm{LIST}_L \cup S$
    else
        $\mathrm{LIST}_R \leftarrow \mathrm{LIST}_R \cup S$
    end
    if $yl - B < T - yh$ **then**
        $\mathrm{LIST}_B \leftarrow \mathrm{LIST}_B \cup S$
    else
        $\mathrm{LIST}_T \leftarrow \mathrm{LIST}_T \cup S$
    end
end

---

if $\max(|\mathrm{LIST}_L|, |\mathrm{LIST}_R|) < \max(|\mathrm{LIST}_B|, |\mathrm{LIST}_T|)$ **then**
  Split the node along the x direction.
else if $\max(|\mathrm{LIST}_L|, |\mathrm{LIST}_R|) > \max(|\mathrm{LIST}_B|, |\mathrm{LIST}_T|)$ **then**
  Split the node along the y direction.
else
  if $\mathrm{overlap}(\mathrm{LIST}_L, \mathrm{LIST}_R) < \mathrm{overlap}(\mathrm{LIST}_B, \mathrm{LIST}_T)$ **then**
    Split the node along the x direction.
  else if $\mathrm{overlap}(\mathrm{LIST}_L, \mathrm{LIST}_R) > \mathrm{overlap}(\mathrm{LIST}_B, \mathrm{LIST}_T)$ **then**
    Split the node along the y direction.
  else
    Split the node along the direction with smallest total coverage.
  end
end

- Můžeme si snadno všimnout, že každý obdélník v N bude přítomen buďto v seznamu LIST_L nebo LIST_R.
- Totéž platí pro LIST_B a LIST_T. Rozhodnutí, za udělat split podle horizontální nebo vertikální osy závisí na rozložení obdélníků (tomu odpovídají řádky v pseudokódu, které porovnávají maxima velikostí seznamů).
- V případě remízy, se algoritmus rozhodne podle kritéria překryvu.

Tento algoritmus může mít problém v případě, že většina obdélníků v N tvoří shluk a několik dalších jich je odlehlých.



Tento problém je obvykle možno řešit odebráním a a znovu-vložením uzlů, které odpovídají těm odlehlým obdélníkům, podobně jako v R*-stromu.